Mesterséges Intelligencia

Modern megközelítésben

Második, átdolgozott, bővített kiadás

Russell, Stuart

Norvig, Peter

Created by XMLmind XSL-FO Converter.

Mesterséges Intelligencia: Modern megközelítésben: Második, átdolgozott, bővített kiadás

Russell, Stuart
Norvig, Peter
Édelkraut, Róbert
Antal, Péter
Dobrowiecki, Tadeusz
Haidegger, Géza
Haidegger, Tamás
Héja, Gergely
Mészáros, Tamás
Pataki, Béla
Román, Gyula
Strausz, György
Tilly, Károly
Valyon, József
Varga, Péter
Várkonyiné Kóczy, Annamária

Ez a könyv az Oktatási Minisztérium támogatásával, a Felsőoktatási Tankönyv- és Szakkönyv-támogatási Pályázat keretében jelent meg.

Publication date 2005
Szerzői jog © 2005 Hungarian Translation Panem Könyvkiadó, Budapest

A mű eredeti címe: ARTIFICIAL INTELLIGENCE. A MODERN APPROACH. 2nd Edition. ISBN 0137903952, by Russell, Stuart and Norvig, Peter, published by Pearson Education, Inc., publishing as Prentice Hall, Upper Saddle River, New Jersey, 07458 Copyright © 2003, 1995 Pearson Education Inc.

A kiadásért felel a Panem Kft. ügyvezetője, Budapest, 2005

Minden jog fenntartva. Jelen könyvet, illetve annak részeit tilos reprodukálni, adatrögzítő rendszerben tárolni, bármilyen formában vagy eszközzel - elektronikus úton vagy más módon - közölni a kiadók engedélye nélkül

Created by XMLmind XSL-FO Converter.

Tartalom

Előszó xxxiv
1. Bevezetés 1
1. Mi az MI? 1
1.1. Emberi módon cselekedni: Turing-teszt megközelítés 2
1.2. Emberi módon gondolkodni: a kognitív modellezés 3
1.3. Racionálisan gondolkodni: a gondolkodás törvénye 3
1.4. Racionálisan cselekedni: a racionális ágens 4
2. A mesterséges intelligencia alapjai 5
2.1. Filozófia (i. e. 428-tól napjainkig) 5
2.2. Matematika (kb. 800-tól napjainkig) 6
2.3. Gazdaságtan (1776-tól napjainkig) 8
2.4. Neurális tudományok (1861-től napjainkig) 8
2.5. Pszichológia (1879-től napjainkig) 10
2.6. Számítógépes tudományok (1940-től napjainkig) 11
2.7. Irányításelmélet és kibernetika (1948-tól napjainkig) 12
2.8. Nyelvészet (1957-től napjainkig) 13
3. A mesterséges intelligencia története 13
3.1. A mesterséges intelligencia érlelődése (1943–1955) 13
3.2. A mesterséges intelligencia megszületése (1956) 14
3.3. Korai lelkesedés, nagy elvárások (1952–1969) 15
3.4. Egy adag realitás (1966–1973) 17
3.5. Tudásalapú rendszerek: a hatalom kulcsa? (1969–1979) 18
3.6. Az MI iparrá válik (1980-tól napjainkig) 20
3.7. A neurális hálók visszatérése (1986-tól napjainkig) 20
3.8. Az MI tudománnyá válik (1987-től napjainkig) 21
3.9. Az intelligens ágensek kialakulása (1995-től napjainkig) 22
4. A mesterséges intelligencia jelenlegi helyzete 22
5. Összefoglalás 23
5.1. Irodalmi és történeti megjegyzések 24
5.2. Feladatok 24
2. Intelligens ágensek 27
1. Bevezetés 27
2. Jó viselkedés: a racionalitás koncepciója 29
2.1. Teljesítménymértékek 29
2.2. Racionalitás 30
2.3. Mindentudás, tanulás és autonómia 31
3. A Környezetek természete 32
3.1. A környezet meghatározása 32
3.2. A környezetek tulajdonságai 34
4. Az intelligens ágensek struktúrája 36
4.1. Ágensprogramok 36
4.2. Egyszerű reflexszerű ágensek 38
4.3. Modellalapú reflexszerű ágensek 40
4.4. Célorientált ágensek 41
4.5. Hasznosságorientált ágensek 42
4.6. Tanuló ágensek 43
5. Összefoglalás 46
5.1. Irodalmi és történeti megjegyzések 46
5.2. Feladatok 48
3. Problémamegoldás kereséssel 50
1. Problémamegoldó ágensek 50
1.1. Jól definiált problémák és megoldások 52
1.2. A problémák megfogalmazása 53
2. Példaproblémák 54
2.1. Játékproblémák 54
2.2. Valósvilág-beli problémák 57

Created by XMLmind XSL-FO Converter.

3. Megoldások keresése ... 58
3.1. A problémamegoldó hatékonyság mérése ... 62
4. Nem informált keresés ... 63
4.1. Szélességi keresés ... 63
4.2. Egyenletes költségű keresés ... 64
4.3. Mélységi keresés ... 65
4.4. Mélységkorlátozott keresés ... 66
4.5. Iteratívan mélyülő mélységi keresés ... 67
4.6. Kétirányú keresés ... 69
4.7. A keresési stratégiák összehasonlítása ... 70
5. Az Ismételt állapotok elkerülése ... 70
6. Keresés részleges információ mellett ... 72
6.1. Szenzor nélküli problémák ... 72
6.2. Eshetőségi problémák ... 74
7. Összefoglalás ... 75
7.1. Irodalmi és történeti megjegyzések ... 76
7.2. Feladatok ... 77
4. Informált keresési és felfedező módszerek ... 82
1. Informált (heurisztikus) keresési stratégiák ... 82
1.1. A mohó legjobbat-először keresés ... 83
1.2. A* keresés: a teljes becsült útköltség minimalizálása ... 84
1.3. Memóriakorlátozott heurisztikus keresés ... 90
1.4. Tanuljunk, hogy jobban keressünk! ... 93
2. Heurisztikus függvények ... 93
2.1. A heurisztikus függvény pontosságának hatása a megoldás hatékonyságára ... 94
2.2. Elfogadható heurisztikus függvények kitalálása ... 95
2.3. A heurisztikus függvény tanulása tapasztalatból ... 97
3. Lokális kereső algoritmusok és optimalizációs problémák ... 98
3.1. Hegymászó keresés ... 99
3.2. Szimulált lehűtés ... 102
3.3. Lokális nyaláb keresés ... 103
3.4. Genetikus algoritmusok ... 103
4. Lokális keresés folytonos terekben ... 106
5. Online kereső ágensek és ismeretlen környezetek ... 108
5.1. Online keresési problémák ... 108
5.2. Online kereső ágensek ... 111
5.3. Online lokális keresés ... 112
5.4. Tanulás online keresés során ... 114
6. Összefoglalás ... 115
6.1. Irodalmi és történeti megjegyzések ... 115
6.2. Feladatok ... 119
5. Kényszerkielégítési problémák ... 122
1. Kényszerkielégítési problémák ... 122
2. A visszalépéses keresés alkalmazása kényszerkielégítési problémákra ... 125
2.1. Változó- és értékrendezés ... 127
2.2. Az információ terjesztése a kényszereken keresztül ... 128
2.2.1. Előrenéző ellenőrzés ... 128
2.2.2. A kényszerek terjesztése ... 128
2.2.3. Speciális kényszerek kezelése ... 130
2.3. Intelligens visszalépés: visszanézni ... 131
3. Lokális keresés kényszerkielégítési problémáknál ... 132
4. A problémák struktúrája ... 134
5. Összefoglalás ... 137
5.1. Irodalmi és történeti megjegyzések ... 138
5.2. Feladatok ... 140
6. Keresés ellenséges környezetben ... 143
1. Kétszemélyes játékok ... 143
2. Optimális döntések kétszemélyes játékokban ... 144
2.1. Optimális stratégiák ... 145
2.2. A minimax algoritmus ... 146

Created by XMLmind XSL-FO Converter.

2.3. Optimális döntések többszemélyes játékokban ... 147
3. Alfa-béta nyesés ... 148
4. Nem tökéletes, valós idejű döntések ... 152
4.1. Kiértékelő függvények ... 153
4.2. A keresés levágása ... 154
5. Véletlen elemet is tartalmazó játékok ... 157
5.1. Az állás kiértékelése véletlen csomópontokat tartalmazó játékok esetén ... 160
5.2. A várhatóminimax komplexitása ... 160
5.3. Kártyajátékok ... 161
6. A jelenleg legfejlettebb játékprogramok ... 162
6.1. Sakk ... 162
6.1.1. Dámajáték ... 163
6.1.2. Othello ... 164
6.1.3. Ostábla ... 164
6.1.4. Gó ... 164
6.1.5. Bridzs ... 164
7. Értékelés ... 164
8. Összefoglalás ... 166
8.1. Irodalmi és történeti megjegyzések ... 166
8.2. Feladatok ... 169
7. Logikai ágensek ... 175
1. A tudásbázisú ágens ... 176
2. A wumpus világ ... 177
3. A logika ... 180
3.1. Szintaxis ... 183
3.2. Szemantika ... 185
3.3. Egy egyszerű tudásbázis ... 186
3.4. Következtetés ... 187
3.5. Ekvivalencia, érvényesség és kielégíthetőség ... 188
4. Az ítéletkalkulus következtetési mintái ... 189
4.1. Rezolúció ... 191
4.1.1. Konjunktív normál forma ... 193
4.1.2. A rezolúció algoritmus ... 193
4.1.3. A rezolúció teljessége ... 195
4.2. Előre- és hátrafelé láncolás ... 195
5. Hatékony ítéletkalkulus következtetés ... 199
5.1. Egy teljes visszalépéses algoritmus ... 199
5.2. Lokális keresés algoritmus ... 200
5.3. Nehéz kielégíthetőségi problémák ... 201
6. Ítéletlogikát alkalmazó ágensek ... 203
6.1. Csapdák és wumpusok megtalálása logikai következtetés felhasználásával ... 203
6.2. A hely és az irány nyomkövetése ... 204
6.3. Az áramkörön alapuló ágens ... 205
6.4. Összehasonlítás ... 208
7. Összefoglalás ... 209
7.1. Irodalmi és történeti megjegyzések ... 209
7.2. Feladatok ... 212
8. Elsőrendű logika ... 216
1. Még egyszer a reprezentációról ... 216
2. Az elsőrendű logika szintaxisa és szemantikája ... 219
2.1. Az elsőrendű logika modelljei ... 219
2.2. Szimbólumok és interpretációk ... 221
2.3. Termek ... 222
2.4. Atomi mondatok ... 222
2.5. Összetett mondatok ... 223
2.6. Kvantorok ... 223
2.7. Univerzális kvantor (∀) ... 223
2.8. Egzisztenciális kvantor (∃) ... 224
2.9. Egymásba ágyazott kvantorok ... 225
2.10. Az ∀ és az ∃ kapcsolata ... 226

Created by XMLmind XSL-FO Converter.

2.11. Egyenlőség 226
3. Az elsőrendű logika használata 227
3.1. Kijelentések és lekérdezések az elsőrendű logikában 227
3.2. A rokonsági tárgyterület 228
3.3. Számok, halmazok és listák 229
3.4. A wumpus világ 231
4. Tudástervezés az elsőrendű logikában 233
4.1. A tudástervezés folyamata 233
4.2. Az elektronikus áramkörök tárgyterülete 235
4.3. A feladat meghatározása 235
4.4. A releváns tudás összegyűjtése 235
4.5. A szótár meghatározása 236
4.6. A tárgyterülettel kapcsolatos általános tudás kódolása 237
4.7. A problémaspecifikus példányok kódolása 237
4.8. Lekérdezések megfogalmazása a következtetési eljárás felé 238
4.9. Hibák kiszűrése a tudásbázisból 238
5. Összefoglalás 238
5.1. Irodalmi és történeti megjegyzések 239
5.2. Feladatok 240
9. Következtetés elsőrendű logikában 244
1. Ítéletlogikai következtetés kontra elsőrendű logikai következtetés 244
1.1. Kvantorokra vonatkozó következtetési szabályok 244
1.2. Redukálás ítéletlogikára 245
2. Egyesítés és kiemelés 246
2.1. Egy elsőrendű következtetési szabály 247
2.2. Egyesítés 248
2.3. Tárolás és visszakeresés 249
3. Előrefelé láncolás 251
3.1. Elsőrendű határozott klózok 251
3.2. Egy egyszerű előrefelé láncolási algoritmus 252
3.3. Hatékony előrefelé láncolás 254
3.3.1. Szabályok illesztése az ismert tényekhez 254
3.3.2. Inkrementális előrefelé láncolás 255
3.3.3. Irreleváns tények 257
4. Hátrafelé láncolás 257
4.1. Egy hátrafelé láncolási algoritmus 257
4.2. Logikai programozás 259
4.3. A logikai programok hatékony megvalósítása 260
4.4. Redundáns következtetés és végtelen hurkok 261
4.5. Korlátozott logikai programozás 263
5. Rezolúció 264
5.1. Az elsőrendű logika konjunktív normál formája 264
5.2. A rezolúciós következtetési szabály 266
5.3. Példabizonyítások 266
5.4. A rezolúció teljessége 268
5.5. Az egyenlőség kezelése 271
5.6. Rezolúciós stratégiák 272
5.6.1. Egységpreferencia 272
5.6.2. Támogató halmaz 273
5.6.3. Bemeneti rezolúció 273
5.6.4. Bennfoglalás 273
5.7. Tételbizonyítók 273
5.7.1. Egy tételbizonyítás szerkesztése 274
5.7.2. A Prolog kiterjesztése 275
5.7.3. Tételbizonyítások mint segédeszközök 276
5.7.4. A tételbizonyítások gyakorlati felhasználása 276
6. Összefoglalás 277
6.1. Irodalmi és történeti megjegyzések 278
6.2. Feladatok 281
10. Tudásbázis reprezentáció 286

Created by XMLmind XSL-FO Converter.

1. Ontológiaszervezés 286
2. Kategóriák és objektumok 288
2.1. Fizikai összetétel 289
2.2. Mértékek 290
2.3. Szubsztanciák és objektumok 292
3. Cselekvések, szituációk és események 293
3.1. A szituációkalkulus ontológiája 293
3.2. Cselekvések leírása a szituációkalkulusban 295
3.3. A reprezentációs probléma megoldása 296
3.4. A következtetési keretprobléma megoldása 297
3.5. Idő- és eseménykalkulus 298
3.6. Általánosított események 299
3.7. Folyamatok 301
3.8. Intervallumok 302
3.9. Folyó események és objektumok 303
4. Mentális események és mentális objektumok 304
4.1. A hiedelmek formális elmélete 304
4.2. Tudás és hiedelem 306
4.3. Tudás, idő és cselekvés 306
5. Az internetes bevásárlás világa 307
5.1. Ajánlatok összehasonlítása 310
6. Következtető rendszerek kategóriák számára 311
6.1. Szemantikus hálók 311
6.2. Leíró logikák 314
7. Következtetés alapértelmezett információval 316
7.1. Nyitott és zárt világok 316
7.2. Negálás mint kudarc és stabil modell szemantika 318
7.3. Körülírás és alapeseti logika 319
8. Igazság-karbantartó rendszerek 321
9. Összefoglalás 322
9.1. Irodalmi és történeti megjegyzések 323
9.2. Feladatok 328
11. Tervkészítés 334
1. A tervkészítési probléma 334
1.1. A tervkészítési problémák nyelve 335
1.2. Kifejezőképesség és kiterjesztések 336
1.3. Példa: Légi teherszállítás 338
1.4. Példa: A pótkerék probléma 338
1.5. Példa: A kockavilág 339
2. Tervkészítés állapottér-kereséssel 340
2.1. Előrefelé keresés az állapottérben 340
2.2. Hátrafelé keresés az állapottérben 341
2.3. Állapottér-keresési heurisztikák 342
3. Részben rendezett tervkészítés 344
3.1. Példa a részben rendezett tervkészítésre 347
3.2. Részben rendezett tervkészítés kötetlen változókkal 349
3.3. Heurisztikák a részben rendezett tervkészítésre 350
4. Tervkészítési gráfok 350
4.1. Tervkészítési gráfok heurisztikus becslésekre 352
4.2. A Graphplan algoritmus 353
4.3. A Graphplan algoritmus leállása 355
5. Tervkészítés ítéletlogikával 356
5.1. Tervkészítési problémák ítéletlogikai leírása 356
5.2. Az ítéletlogikai leírás bonyolultsága 358
6. A tervkészítési módszerek elemzése 360
7. Összefoglalás 361
7.1. Irodalmi és történeti megjegyzések 361
7.2. Feladatok 363
12. Tervkészítés és cselekvés a való világban 369
1. Idő, ütemezés és erőforrások 369

Created by XMLmind XSL-FO Converter.

1.1. Ütemezés erőforráskorlátokkal 371
2. Hierarchikus feladatháló tervkészítés 373
2.1. A cselekvésdekompozíciók reprezentációja 374
2.2. A tervkészítő módosítása a dekompozíciók kezeléséhez 376
2.3. Elemzés 378
3. Tervkészítés és cselekvés nemdeterminisztikus problémakörökben 380
4. Feltételes tervkészítés 382
4.1. Feltételes tervkészítés teljesen megfigyelhető környezetekben 382
4.2. Feltételes tervkészítés részlegesen megfigyelhető környezetekben 386
5. Végrehajtás monitorozása és újratervezése 389
6. Folytonos tervkészítés 393
7. Multiágens tervkészítés 396
7.1. Kooperáció: közös célok és tervek 397
7.2. Többtestű tervezés 398
7.3. Koordináló mechanizmusok 399
7.4. Versengés 400
8. Összefoglalás 400
8.1. Irodalmi és történeti megjegyzések 401
8.2. Feladatok 404
13. Bizonytalanság 408
1. Cselekvés bizonytalan tudás esetén 408
1.1. Bizonytalan tudás kezelése 408
1.2. Bizonytalanság és racionális döntések 410
1.3. Egy döntéselméleti ágens tervezése 411
2. Valószínűségi alapfogalmak 411
2.1. Állítások 412
2.2. Elemi események 412
2.3. A priori valószínűség 413
2.4. Feltételes valószínűség 414
3. Valószínűségi axiómák 415
3.1. A valószínűségi axiómák használata 416
3.2. Ami a valószínűségi axiómákat indokolja 417
4. Teljes együttes valószínűség-eloszláson alapuló következtetés 418
5. Függetlenség 421
6. A Bayes-tétel és használata 422
6.1. Bayes tételének alkalmazása: egyszerű eset 422
6.2. A Bayes-tétel alkalmazása: több együttes tény figyelembevétele 423
7. A wumpus világ újralátogatása 425
8. Összefoglalás 428
8.1. Irodalmi és történeti megjegyzések 429
8.2. Feladatok 430
14. Valószínűségi következtetés 434
1. A tudás reprezentálása bizonytalanság esetén 434
2. A Bayes-hálók szemantikája 436
2.1. Az együttes valószínűség-eloszlás függvény leírása 436
2.2. Feltételes függetlenségi relációk Bayes-hálókban 440
3. Feltételes eloszlások hatékony reprezentációja 440
3.1. Bayes-hálók folytonos változókkal 441
4. Egzakt következtetés Bayes-hálókban 444
4.1. Következtetés felsorolással 444
4.2. A változó eliminációs algoritmus 447
4.3. Az egzakt következtetés komplexitása 449
4.4. Csoportosító algoritmusok 449
5. Közelítő következtetés Bayes-hálókban 450
5.1. Közvetlen mintavételezési módszerek 450
5.2. Következtetés Markov-lánc szimulációval 455
6. Elsőrendű reprezentációk valószínűségi kiterjesztése 457
7. Egyéb módszerek a bizonytalan környezetben történő következtetéshez 460
7.1. Bizonytalansági következtetés szabályalapú eljárásokkal 461
7.2. Az ismerethiány reprezentálása: a Dempster–Shafer-elmélet 463

Created by XMLmind XSL-FO Converter.

7.3. A meghatározatlanság reprezentálása: fuzzy halmazok és logikák 464
8. Összefoglalás 465
8.1. Irodalmi és történeti megjegyzések 465
8.2. Feladatok 469
15. Időbeli valószínűségi következtetés 473
1. Idő és bizonytalanság 473
1.1. Állapotok és megfigyelések 473
1.2. Stacionárius folyamatok és a Markov-feltétel 474
2. Következtetés időbeli modellekben 476
2.1. Szűrés és előrejelzés 477
2.2. Simítás 479
2.3. A legvalószínűbb sorozat megtalálása 481
3. Rejtett Markov-modellek 482
3.1. Egyszerűsített mátrix algoritmusok 483
4. Kalman-szűrők 485
4.1. Gauss-eloszlások frissítése 486
4.2. Egy egyszerű egydimenziós példa 487
4.3. Az általános eset 489
4.4. A Kalman-szűrés alkalmazhatósága 490
5. Dinamikus Bayes-hálók 491
5.1. DBH-k létrehozása 492
5.2. Egzakt következtetés DBH-kban 495
5.3. Közelítő következtetés DBH-kban 496
6. Beszédfelismerés 499
6.1. Beszédhangok 500
6.2. Szavak 502
6.3. Mondatok 504
6.4. Egy beszédfelismerő építése 506
7. Összefoglalás 507
7.1. Irodalmi és történeti megjegyzések 508
7.2. Feladatok 510
16. Egyszerű döntések meghozatala 513
1. Meggyőződések és kívánságok összekapcsolása bizonytalanság esetén 513
2. A hasznosságelmélet alapjai 514
2.1. Megkötések a racionális preferenciákra 514
2.2. És aztán jött a hasznosság 516
3. Hasznosságfüggvények 517
3.1. A pénz hasznossága 517
3.2. Hasznosságskálák és a hasznosság megbecslése 519
4. Többváltozós hasznosságfüggvények 520
4.1. Dominancia 521
4.2. A preferenciák rendszere és a többattribútumos hasznosság 522
5. Döntési hálók 524
5.1. Döntési problémák reprezentálása döntési hálókkal 524
5.2. Döntési hálók kiértékelése 525
6. Az információ értéke 526
6.1. Egy egyszerű példa 527
6.2. Egy általános képlet 527
6.3. Az információ értékének tulajdonságai 529
6.4. Egy információgyűjtő ágens megvalósítása 529
7. Döntéselméleti szakértő rendszerek 530
8. Összefoglalás 532
8.1. Irodalmi és történeti megjegyzések 533
8.2. Feladatok 534
17. Komplex döntések meghozatala 538
1. Szekvenciális döntési problémák 538
1.1. Egy példa 538
1.2. Optimalitás szekvenciális döntési problémákban 541
2. Értékiteráció 542
2.1. Az állapotok hasznossága 543

Created by XMLmind XSL-FO Converter.

2.2. Az értékiteráció algoritmus 544
2.3. Az értékiteráció konvergenciája 545
3. Eljárásmód-iteráció 548
4. Részlegesen megfigyelhető Markov döntési folyamatok 550
5. Döntéselméleti ágensek 553
6. Többágenses döntések: a játékelmélet 555
7. Működési mód tervezés 563
8. Összefoglalás 565
8.1. Irodalmi és történeti megjegyzések 566
8.2. Feladatok 567
18. Megfigyelések alapján történő tanulás 571
1. Tanulási formák 571
2. Induktív tanulás 572
3. Döntési fák megalkotása tanulással 574
3.1. Cselekvő komponensként használt döntési fák 574
3.2. A döntési fák kifejezőképessége 575
3.3. A döntési fák példák alapján történő felépítése 576
3.4. Attribútumteszt-választás 579
3.5. A tanuló algoritmus teljesítményének becslése 580
3.6. Zaj és túlilleszkedés 582
3.7. A döntési fák alkalmazhatóságának kiterjesztése 583
4. Hipotézishalmaz együttes tanulása 584
5. Miért működik a tanulás: a tanulás számítási elmélete 588
5.1. Hány példára van szükség? 588
5.2. Döntési listák tanulása 590
5.3. Elemzés 592
6. Összefoglalás 593
6.1. Irodalmi és történeti megjegyzések 593
6.2. Feladatok 595
19. A tudás szerepe a tanulásban 598
1. A tanulás logikai megfogalmazása 598
1.1. Példák és hipotézisek 598
1.2. A pillanatnyilag legjobb hipotézis keresése 599
1.3. Legkisebb megkötés elvű keresés 601
2. A tudás szerepe a tanulásban 605
2.1. Néhány egyszerű példa 606
2.2. Néhány általános séma 607
3. Magyarázatalapú tanulás 608
3.1. Általános szabályok kinyerése példákból 609
3.2. A hatékonyság javítása 610
4. Tanulás releváns információ alapján 612
4.1. A hipotézistér meghatározása 612
4.2. Tanulás releváns információ felhasználásával 613
5. Induktív logikai programozás 615
5.1. Egy példa 615
5.2. Felülről lefelé tanulási módszerek 618
5.3. Induktív tanulás inverz rezolúcióval 620
5.4. Felfedezés induktív logikai programozással 622
6. Összefoglalás 623
6.1. Irodalmi és történeti megjegyzések 623
6.2. Feladatok 625
20. Statisztikai tanulási módszerek 627
1. Statisztikai tanulás 627
2. Teljes adattal történő tanulás 630
2.1. Maximum-likelihood paramétertanulás: diszkrét modellek 630
2.2. Naiv Bayes-modellek 632
2.3. Maximum-likelihood paramétertanulás: folytonos eset 633
2.4. Bayes-paramétertanulás 635
2.5. Bayes-hálóstruktúrák tanulása 637
3. Rejtett változókkal történő tanulás: az EM algoritmus 638

Created by XMLmind XSL-FO Converter.

3.1. Nem ellenőrzött osztályozás: Gauss-eloszlások keverékének tanulása 639
3.2. Rejtett változókkal felépített Bayes-hálók tanulása 641
3.3. Rejtett Markov-modellek tanulása 644
3.4. Az EM algoritmus általános alakja 645
3.5. Bayes-hálóstruktúra tanulása rejtett változók esetén 645
4. Példányalapú tanulás 646
4.1. Legközelebbi-szomszéd modellek 646
4.2. Kernelmódszerek 648
5. Neurális hálók 649
5.1. A neurális háló egységei 650
5.2. Hálóstruktúrák 651
5.3. Egyrétegű előrecsatolt neurális hálók (perceptronok) 652
5.4. Többrétegű előrecsatolt neurális hálók 656
5.5. Neurális hálóstruktúrák tanulása 660
6. Kernelgépek 660
7. Esettanulmány: kézzel írott számjegyek felismerése 664
8. Összefoglalás 665
8.1. Irodalmi és történeti megjegyzések 666
8.2. Feladatok 669
21. Megerősítéses tanulás 673
1. Bevezetés 673
2. Passzív megerősítéses tanulás 674
2.1. Közvetlen hasznosságbecslés 675
2.2. Adaptív dinamikus programozás 676
2.3. Az időbeli különbség tanulása 677
3. Aktív megerősítéses tanulás 679
3.1. Felfedezés 680
3.2. Egy cselekvésérték-függvény tanulása 683
4. A megerősítéses tanulás általánosító-képessége 684
4.1. Alkalmazások a játékok területén 687
4.2. Robotirányítási alkalmazások 687
5. Stratégiakeresés 688
6. Összefoglalás 691
6.1. Irodalmi és történeti megjegyzések 692
6.2. Feladatok 694
22. Kommunikáció 697
1. A kommunikáció mint cselekvés 697
1.1. A nyelv alapjai 698
1.2. A kommunikációt alkotó lépések 699
2. formális nyelvtan az angol nyelv egy töredékére 702
2.1. Az $\mathcal{E}_0$ szókincse 702
2.2. Az $\mathcal{E}_0$ nyelvtana 703
3. Szintaktikai analízis (elemzés) 704
3.1. Hatékony elemzés 706
4. Kiterjesztett nyelvtanok 711
4.1. Igék alkategóriákba osztása 713
4.2. Kiterjesztett nyelvtanok generálóképessége 715
5. Szemantikai értelmezés 716
5.1. Az angol nyelv egy részletének szemantikája 717
5.2. Idő és igeidő 718
5.3. Kvantifikálás 719
5.4. Pragmatikus értelmezés 721
5.5. Nyelv generálása DCG-kkel 722
6. Többértelműség és feloldása 723
6.1. A többértelműség feloldása 724
7. Szövegértés 725
7.1. Utalásfeloldás 725
7.2. Egy koherens szöveg struktúrája 727
8. A nyelvtan indukciós tanulása 728
9. Összefoglalás 730

Created by XMLmind XSL-FO Converter.

9.1. Irodalmi és történeti megjegyzések 731
9.2. Feladatok 734
23. Valószínűségi nyelv-feldolgozás 737
1. Valószínűségi nyelvi modellek 737
1.1. Valószínűségi környezetfüggetlen nyelvtanok 739
1.2. PCFG-valószínűségek tanulása 741
1.3. PCFG-szabálystruktúrák tanulása 742
2. Információkeresés 742
2.1. Az IR-rendszerek értékelése 745
2.2. Az IR-rendszerek továbbfejlesztése 745
2.3. Az eredményhalmaz prezentálása 746
2.4. Az IR-rendszerek megvalósítása 747
3. Információkinyerés 748
4. Gépi fordítás 751
4.1. Gépi fordító rendszerek 752
4.2. Statisztikai gépi fordítás 753
4.3. Valószínűségek tanulása gépi fordításhoz 756
5. Összefoglalás 757
5.1. Irodalmi és történeti megjegyzések 757
5.2. Feladatok 759
24. Az észlelés 762
1. Bevezetés 762
2. Képalkotás 763
2.1. Lencsék nélküli képek: a sötétkamra 763
2.2. Lencserendszerek 764
2.3. A fény: a képalkotás fotometriája 765
2.4. Színek: a képalkotás spektrális fotometriája 766
3. Előzetes képfeldolgozási műveletek 766
3.1. Éldetektálás 767
3.2. A kép szegmentálása 770
4. D információ kinyerése 770
4.1. A mozgás 771
4.2. Kétkamerás (binokuláris) térbeli látás 774
4.3. Textúragradiensek 775
4.4. Árnyalás 776
4.5. Kontúrok 777
5. Objektumok felismerése 780
5.1. Fényességalapú felismerés 782
5.2. Jellemzőalapú felismerés 783
5.3. Az elhelyezkedés becslése 785
6. Navigálás és manipulálás a látás segítségével 787
7. Összefoglalás 789
7.1. Irodalmi és történeti megjegyzések 789
7.2. Feladatok 791
25. Robotika 795
1. Bevezetés 795
2. Robothardver 796
2.1. Érzékelők 796
2.2. Beavatkozó szervek 797
3. Érzékelés a robotikában 799
3.1. Helymeghatározás 800
3.2. Térképezés 805
3.3. További érzékelési típusok 808
4. Mozgástervezés 808
4.1. Konfigurációs tér 808
4.2. Celladekompozíciós módszerek 811
4.3. Szkeletonizációs módszerek 813
5. Bizonytalan mozgások tervezése 814
5.1. Robusztus módszerek 815
6. Mozgás 817

Created by XMLmind XSL-FO Converter.

6.1. Dinamika és vezérlés 818
6.2. Potenciáltér-vezérlés 819
6.3. Reaktív irányítás 820
7. Szoftverarchitektúrák a robotikában 821
7.1. Alárendelt architektúra 822
7.2. Háromréteges architektúra 823
7.3. Robotprogramozási nyelvek 823
8. Alkalmazási területek 824
9. Összefoglalás 826
9.1. Irodalmi és történeti megjegyzések 827
9.2. Feladatok 829
26. Filozófiai alapok 834
1. Gyenge MI: Tudnak-e a gépek intelligensen cselekedni? 834
1.1. A képesség hiányából vett érv 835
1.2. A matematikai ellenvetés 836
1.3. A meghatározatlanságból vett érv 837
2. Erős MI: Tudnak-e ténylegesen gondolkodni a gépek? 838
2.1. A test–elme probléma 840
2.2. Az „agy a tartályban” kísérlet 840
2.3. Az agyprotézis-kísérlet 841
2.4. A kínai szoba 842
3. A mesterséges intelligencia fejlesztésének etikai kérdései és kockázatai 844
4. Összefoglalás 847
4.1. Irodalmi és történeti megjegyzések 847
4.2. Feladatok 849
27. MI: Jelen és jövő 851
1. Ágensösszetevők 851
2. Ágensarchitektúrák 853
3. Egyáltalán a jó irányba haladunk? 854
4. Mi van, ha az MI sikerrel jár? 856
A. Matematikai alapok 857
1. A1. Bonyolultságanalízis és az O() jelölés 857
1.1. Aszimptotikus analízis 857
1.2. NP és az inherensen nehéz problémák 858
2. A2. Vektorok, mátrixok és lineáris algebra 858
3. A3. Valószínűségi eloszlások 860
3.1. Irodalmi és történeti megjegyzések 861
B. Megjegyzések a nyelvekről és az algoritmusokról 863
1. B1. Nyelvek definiálása Backus–Naur-Formában (BNF) 863
2. B2. Az algoritmusok leírása pszeudokóddal 863
3. B3. Online segédnyújtás 864
C. Irodalomjegyzék 865
D. Magyar nyelvű szakirodalom 942
1. Mesterséges intelligencia általánosságban 942

Created by XMLmind XSL-FO Converter.

Az ábrák listája

1.1. Az MI néhány meghatározása négy kategóriába szervezve 1
1.2. Az idegsejt vagy neuron részei. Minden neuron sejttestből vagy szómából áll, amely sejtmagot tartalmaz. A sejttestből ágazik el néhány dendritnek nevezett és egy hosszú, axonnak nevezett szál. Az axon jó messzire elér, sokkal messzebbre, mint ahogy ezt a jelen ábra sugallja. Az axonok tipikusan 1 cm hosszúak (ez a sejttest átmérőjének 100-szorosa), de elérhetik az 1 m-t is. Egy neuron 10–100 000 más neuronnal tart kapcsolatot a szinapszisoknak nevezett csatlakozásokon keresztül. A jeleket neuronról neuronra egy összetett elektrokémiai reakció továbbítja. A jelek rövid távon szabályozzák az agy aktivitását, hosszú távon pedig befolyásolják a neuronok helyzetét és a kapcsolódási rendszerüket. Úgy tartják, e mechanizmusok képezik az agyi tanulás alapját. Az információfeldolgozás zöme az agykéregben történik, ami az agy külső rétege. A feldolgozás szervezeti egysége, úgy tűnik, egy kb. 0,5 mm-es átmérőjű szövetoszlop, amelynek magassága az egész kéregre kiterjed, ez az embereknél kb. 4 mm. Egy oszlop kb. 20 000 neuront tartalmaz. 9
1.3. A számítógépek számítási erőforrásainak (2003 körül) és az agy erőforrásainak durva összehasonlítása. A könyv első kiadása óta a számítógépekre jellemző számok legalább egy 10-es szorzóval megnőttek, és várható, hogy ez a növekedés ebben az évtizedben is folytatódik. Az agyra jellemző számok az utóbbi 10 000 évben nem változtak. 10
1.4. Egy konkrét probléma, amelyet Evans ANALOGY programja megoldott 16
1.5. Egy elrendezés a kockavilágban. A SHRDLU robot (Winograd, 1972) éppen sikeresen teljesítette a „keresd meg azt a blokkot, ami a kezedben tartottnál magasabb, és tedd a dobozba” utasítást. 16
2.1. Az ágensek környezetükkel érzékelőkön és beavatkozókon keresztül állnak kapcsolatban 27
2.2. A porszívóvilág mindössze két helyszínnel 28
2.3. A 2.2. ábrán szereplő porszívóvilág egy egyszerű ágensfüggvénye részleges táblázatos formában 28
2.4. Egy automatizált taxi feladatkörnyezetének TKBÉ-leírása 32
2.5. Példák ágenstípusokra és TKBÉ-leírásuk 33
2.6. Példák feladatkörnyezetekre és jellemzőik 35
2.7. A TÁBLÁZAT-VEZÉRLÉSŰ-ÁGENS program minden egyes új észlelésre meghívódik, és minden alkalommal visszaad egy cselekvést. Saját adatstruktúrájában követi nyomon az észlelési sorozatot. 37
2.8. Egy egyszerű reflexszerű ágens programja a kétállapotú porszívókörnyezetben. A 2.3. ábrán megadott ágensfüggvényt valósítja meg. 38
2.9. Egy egyszerű reflexszerű ágens sematikus diagramja 38
2.10. Egy egyszerű reflexszerű ágens. A jelenlegi szituációra (amit az észlelés határoz meg) illeszkedő feltételű szabály megkeresésével, majd a hozzá tartozó cselekvés végrehajtásával működik. 39
2.11. Belső állapottal rendelkező reflexszerű ágens 40
2.12. A modellalapú reflexszerű ágens. Nyomon követi a világ jelenlegi állapotát egy belső modellben. Ezek után a reflexszerű ágenshez hasonlóan választ egy cselekvést. 40
2.13. Egy modellalapú, célorientált ágens. Nyomon követi a világ állapotát és az elérendő célok halmazát is, és kiválaszt egy cselekvést, amely (végső soron) céljainak eléréséhez vezet. 41
2.14. Egy modellalapú, hasznosságalapú ágens. A világ modellje mellett egy hasznosságfüggvényt is alkalmaz, amely a világ állapotaihoz rendelt preferenciáit méri. Ezek után olyan cselekvést választ, amely a legjobb várható hasznossághoz vezet, amit az összes lehetséges végállapot előfordulási valószínűségével súlyozott átlagolásával számít ki. 43
2.15. Tanuló ágensek egy általános modellje 44
3.1. Egy egyszerű problémamegoldó ágens. Először a célt és a problémát fogalmazza meg, majd a problémát megoldó cselekvéssorozatot keres, végül a cselekvéseket egyenként végrehajtja. Amikor kész vele, egy másik célt fogalmaz meg, és az egészet újrakezdi. Jegyezzük meg, hogy a cselekvéssorozata végrehajtása alatt az ágens az érzékeléseivel nem foglalkozik. Feltételezi, hogy az általa megtalált megoldás mindig működőképes. 51
3.2. Románia egy részének sematikus országúti térképe 53
3.3. A porszívóvilág állapottere. Az élek cselekvéseket jelentenek: B = *Balra*, J = *Jobbra*, S = *Szív*. 54
3.4. A 8-as kirakójáték egy tipikus feladványa 55
3.5. Egy majdnem jó megoldás a 8-királynő problémára. (A megoldás megkeresését az olvasóra bízzuk.) 56
3.6. Az Arad és Bukarest közötti útkeresési probléma részleges keresési fái. A kifejtett csomópontokat árnyékoltuk, a legenerált, de még ki nem fejtett csomópontok vastagon keretezettek, a még le nem generált csomópontokat pedig halvány szaggatott vonalak jelzik. 59

Created by XMLmind XSL-FO Converter.

3.7. Az általános fakeresési algoritmus informális leírása 60
3.8. A keresési fa alapvető építő adatstruktúrái a csomópontok. Minden csomópontnak van szülője, állapota és számos adminisztráló adatmezeje. A nyilak a gyerekektől a szülőkig mutatnak. 60
3.9. Az általános fakeresési algoritmus. (Vegyük észre, hogy a *perem* argumentumnak egy üres sornak kell lennie, és a sor típusa befolyással lesz a keresés sorrendjére.) A MEGOLDÁS függvény a szülőmutatók gyökérig való követésével kinyert cselekvéssorozatot adja vissza. 61
3.10. Szélességi keresés egy egyszerű bináris fában. Minden lépésnél a következő kifejtendő csomópontot egy marker jelzi. 63
3.11. A szélességi keresés idő- és tárigénye. Az ábra adatai b = 10-es elágazási tényezőt, 10 000 csomópont/percet és 1000 bájt/csomópontot feltételeznek. 64
3.12. Mélységi keresés egy bináris keresési fában. A kifejtett csomópontok, amelyeknek a peremben nincsenek követőik, el is hagyhatók a memóriából. Ezeket feketével jelöltük meg. A 3-as mélységű csomópontokról feltételezzük, hogy nincsenek követőik, valamint azt is feltesszük, hogy M az egyetlen célcsomópont. 65
3.13. A mélységkorlátozott keresés rekurzív implementációja 66
3.14. Az iteratívan mélyülő keresési algoritmus mélységkorlátozott keresést alkalmaz ismételten, növekvő mélységkorláttal. Az algoritmus megáll, ha a megoldást megtalálja, vagy ha a mélységkorlátozott keresés kudarccal tér vissza, jelezve, hogy megoldás nem létezik. 67
3.15. Az iteratívan mélyülő keresési algoritmus négy iterációja egy bináris fán 68
3.16. A kétirányú szélességi keresés sematikus ábrája. Az ábrán a két keresési irány majdnem találkozik, amikor a kiinduló csomópontból kinyúló egyik ág összeér egy a célcsomópontból kinyúló másik ággal. 69
3.17. A keresési stratégiák értékelése. b az elágazási tényező, d a legsekélyebb megoldás mélysége, m a keresési fa maximális mélysége, ℓ a mélységkorlát. A felső indexszel jelzett kikötések a következők: [a] teljes, ha b véges; [b] teljes, ha a lépésköltség $\geq \varepsilon$, pozitív ε-ra; [c] optimális, ha a lépésköltségek mind azonosak; [d] ha mindkét irányban szélességi keresést használunk. 70
3.18. Egy exponenciálisan nagyobb keresési fát generáló állapottér. (a) Egy olyan állapottér, amelyben két lehetséges cselekvés vezethet A-ból B-be, kettő B-ből C-be és így tovább. Ez az állapottér d+1 állapotot tartalmaz, ahol d a maximális mélység. (b) Az (a) állapottér megfelelő keresési fája, 2^d ággal, amely az állapotér 2^d útjának felel meg. (c) Egy négyzetrács. A kezdeti (A) állapottól kétlépésnyire fekvő állapotokat szürke színnel jelöltük. 71
3.19. Az általános gráfkereső algoritmus. A *zárt* halmazt egy hash-táblával lehet implementálni, hogy az ismételt állapotokat hatékonyan ellenőrizni tudjuk. Ez az algoritmus feltételezi, hogy az s állapothoz vezető első út a legolcsóbb (lásd szöveg). 71
3.20. A porszívóvilág nyolc lehetséges állapota 73
3.21. A determinisztikus, szenzor nélküli porszívóvilág hiedelmi állapotterének hozzáférhető része. Minden árnyalt doboz egyetlenegy hiedelmi állapothoz tartozik. Minden pillanatban az ágens egy konkrét hiedelmi állapotban tartózkodik, azonban nem tudja, hogy melyik fizikai állapotban van. A kezdeti hiedelmi állapot (a teljes tudatlanság) legfelül középen helyezkedik el. A cselekvéseket a címkézett nyilak jelölik. Az önhurkokat a nagyobb áttekinthetőség érdekében az ábrából kihagytuk. 73
3.22. Egy elrendezés sokszögakadályokkal 79
4.1. A h_{LMT} értékei – a légvonalbeli távolságok Bukarestig 83
4.2. A mohó legjobbat-először keresés lépései Bukarest esetén a légvonalban mért távolságot (h_{LMT}) alkalmazva. A csomópontok a saját h-értékeikkel vannak felcímkézve. 83
4.3. Az A* keresés lépései Bukarest keresése során. A csomópontok az $f = g + h$ értékekkel vannak felcímkézve. A h-értékek a Bukaresttől légvonalban mért távolságokat jelölik, melyeket a 4.1. ábrából vettünk át. 86
4.4. Románia térképe. Az ábra Arad mint kiinduló állapot esetén az f = 380, f = 400 és f = 420 értékekhez tartozó határvonalat mutatja. Egy adott határvonalon belüli csomópontokhoz a határvonal értékénél kisebb f költség tartozik. 88
4.5. A rekurzív legjobbat-először keresés algoritmusa 90
4.6. Az RLEK lépései, miközben a legrövidebb utat keresi Bukarest felé. A mindenkori rekurzív hívás f-korlát értéke a mindenkori aktuális csomópont felett látható. (a) A Rimnicu Vilceán át vezető utat az algoritmus addig követi, amíg az aktuális legjobb levél (Piteşti) értéke nem lesz rosszabb, mint a legjobb alternatív út értéke (Fogaras). (b) A rekurzió visszalép és az elfelejtett alfa legjobb levélértékét (417) Rimnicu Vilceánál feljegyezzük. Majd Fogaras kifejtése következik 450-nel, mint a legjobb levélértékkel. (c) A rekurzió visszalép. Az elfelejtett alfa legjobb levélértékét (450) Fogarasnál jegyezzük fel. Következik Rimnicu Vilcea kifejtése. Ezúttal, mivel a legjobb alternatív út (Temesváron keresztül) 447-be kerül, a kifejtés folytatódik Bukarest felé. 91
4.7. A 8-as kirakójáték egy tipikus feladata. A megoldás 26 lépés hosszú. 94

Created by XMLmind XSL-FO Converter.

4.8. A keresési költség és az effektív elágazási tényező összehasonlítása ITERATÍVAN-MÉLYÜLŐ-KERESÉS és A* esetén h_1 és h_2 használatával. Az adatokat különböző hosszúságú megoldásokra a 8-as kirakójáték 100 egyedét megoldva átlagolva kaptuk. 94
4.9. A 4.7. ábrán látható 8-as kirakójáték eset egy részproblémája. A feladat az, hogy 1, 2, 3 és 4 lapkát a helyes pozíciókba juttassuk el, nem törődve azzal, hogy a többi lapkával mi fog történni. 97
4.10. Egy egydimenziós állapottérfelszín, ahol a magasság a célfüggvénynek felel meg. A cél a globális maximum megtalálása. A hegymászó keresés az aktuális állapotot módosítja a javulás irányában, ahogy ezt a nyíl is mutatja. A különböző topografikus jellemzőkkel a szövegben foglalkozunk. 98
4.11. A hegymászó keresési algoritmus (a **legmeredekebb emelkedő** (**steepest ascent**) változat) a lokális keresés alapvető módszere. Minden lépésben az aktuális csomópontot a legjobb szomszédjával cseréli le, ami ebben a változatban a legmagasabb ÉRTÉK-ű szomszédot jelenti. Ha azonban egy *h* heurisztikus költségbecslést alkalmazunk, ez akkor a legalacsonyabb *h*-jú szomszéd lenne. 99
4.12. (a) A 8-királynő probléma $h = 17$ heurisztikus költségbecslésű állapota, ahol minden olyan követő állapot *h* értékét megadtuk, amikor a királynőt a saját oszlopában mozgatjuk. A legjobb lépéseket bejelöltük. (b) Egy lokális minimum a 8-királynő térben. Az állapot *h* értéke egységnyi, azonban minden követőnek magasabb a költsége. 100
4.13. Annak illusztrálása, hogy a gerinc miért jelent nehézséget a hegymászó keresés számára. Az állapotok rácshálózata (sötét körök) rá van illesztve a balról jobbra emelkedő gerincre, egy olyan lokális maximum sorozatot eredményezve, ahol a maximumok nincsenek közvetlen módon egymással összekapcsolva. Minden egyes lokális maximumból az összes lehetséges cselekvés a lejtőn lefelé mutat. 101
4.14. A szimulált lehűtés keresési algoritmus. A hegymászó keresés egy olyan változata, ahol lefelé tartó lépések is megengedettek. A lefelé tartó lépéseket a lehűtési terv elején inkább, az idő múlásával egyre kevésbé fogadjuk el. A *lehűtési terv*, mint az eljárás bemenete, a *T* értékeit mint időfüggvényt adja meg. 102
4.15. A genetikus algoritmus. Az (a)-beli kezdeti populációt (b)-ben fitness-függvény alapján rangsoroljuk, aminek eredményét a (c)-ben látható reprodukáló párok adják. A létrehozott utódok a (d)-ben láthatók, melyekre még hat a mutáció (e). 104
4.16. A 4.15. (c) ábrán látható első két szülőnek és a 4.15. (d) ábrán látható két utódnak megfelelő 8-királynő állapotok. Az árnyalt oszlopok a kereszteződés során elvesznek, a nem árnyalt oszlopok megmaradnak. 104
4.17. Egy genetikus algoritmus. Ez az algoritmus ugyanaz, mint amelyet a 4.15. ábrán mutattunk, egy kivétellel. Ebben a jobban elterjedt változatban a két szülő reprodukciója nem két, hanem csak egy utódot hoz létre. 105
4.18. Egy egyszerű labirintusprobléma. Az ágens *S*-nél kezd és *C*-t kell elérnie, a környezetéről azonban semmit sem tud. 109
4.19. (a) Két olyan állapottér, amely az online kereső ágenst egy zsákutcába viheti. (b) Egy kétdimenziós környezet, amely arra késztetheti az online kereső ágenst, hogy a célhoz legkevésbé hatékony utat kövesse. Akármit is választ az ágens, az ellenség eltorlaszolja az útját egy másik hosszú, vékony fallal úgy, hogy a követett út sokkal hosszabb lesz, mint a legjobb lehetséges út. 110
4.20. Egy mélységi feltárást használó online kereső ágens. Az ágens csak a kétirányú keresési terekben alkalmazható. 111
4.21. Egy környezet, amelyben a véletlen vándorlásnak exponenciálisan sok lépésre lesz szüksége, hogy a célt megtalálja 113
4.22. A TRTA* öt iterációja egydimenziós állapottérben. Minden állapotot $H(s)$-sel, a cél elérésének aktuális becslésével, és minden élt a lépésköltséggel címkéztünk. Az árnyalt állapot az ágens helyét jelöli meg, és a frissített értékeket minden iterációban bekarikáztuk. 113
4.23. A TRTA* ágens a cselekvéseit a környező állapotok értékei alapján választja ki, mely az állapotokat az állapottérben mozogva folyamatosan frissíti 114
5.1. (a) Ausztrália államai és területei. A térkép kiszínezése tekinthető kényszerkielégítési problémának is. A cél az, hogy minden egyes részhez olyan színt találjunk, amely nem azonos egy szomszédos rész színével sem. (b) A térképszínezési probléma kényszergráfként reprezentálva. 123
5.2. (a) Egy betűrejtvény. Mindegyik betű különböző számjegyet jelöl, a rejtvény célja olyan számjegyeket helyettesíteni a betűk helyére, amelyekkel a kiadódó összeg aritmetikailag helyes lesz (azzal a külön megkötéssel, hogy nem kezdő nullák nem megengedettek). (b) A betűrejtvény kényszerhipergráfja a *MindKül* kényszer és az oszloponkénti összeadási kényszerek feltüntetésével. A kényszereket négyzet alakú dobozok jelölik. Minden doboz azzal a változóval van összekötve, amelyre a doboz által jelzett kényszer vonatkozik. 125
5.3. Egy egyszerű visszalépéses keresési algoritmus kényszerkielégítési problémákhoz. Az algoritmus a 3. fejezet rekurzív mélységi keresőjén alapul. A HOZZÁRENDELETLEN-VÁLTOZÓ-KIVÁLASZTÁSA és a

Created by XMLmind XSL-FO Converter.

TARTOMÁNY-ÉRTÉKEK-SORRENDEZÉSE függvények használhatók a szövegben bemutatott általános célú heurisztikák megvalósítására. 126
5.4. Részlet a keresési fából, amelyet az egyszerű visszalépéses keresés generált az 5.1. ábrán bemutatott térképszínezési problémához 126
5.5. Az egyes CSP-algoritmusok teljesítménye különböző problémák megoldásában. Az algoritmusok balról jobbra haladva a következők: egyszerű visszalépéses, MRV-heurisztikával bővített visszalépéses, előrenéző ellenőrzés, MRV-heurisztikával kiegészített előrenéző ellenőrzés és a minimális konfliktusok lokális keresése. Az egyes cellákban a probléma megoldása során elvégzett konzisztencia-ellenőrzések száma található (öt futtatás mediánértékéből meghatározva); figyeljünk fel arra, hogy a két jobb felső cella kivételével mindegyik érték ezrekben van megadva (ezt jelzi a K betű). Az első probléma az Amerikai Egyesült Államok 50 államát tartalmazó térkép négy színnel történő színezése. A többi probléma a (Bacchus és van Run, 1995)-ből származik (1. táblázat). A második probléma az *n*-királynő megoldásához szükséges ellenőrzések száma (*n* értékeire 2-től 50-ig összegezve). A harmadik az 5.13. feladatban is szereplő „zebrarejtvény”. A maradék kettő mesterséges véletlenszerűen generált problémák. (A min-konfliktusok algoritmus ezeken nem futott.) Az eredmények azt érzékeltetik, hogy az MRV-heurisztikával kombinált előrenéző ellenőrzés mindegyik probléma esetén jobban teljesít, mint a többi visszalépéses algoritmus, de nem minden esetben jobb a min-konfliktusokat használó helyi keresésnél. 126
5.6. A térképszínezés folyamata előrenéző ellenőrzéses keresés esetén. Az első hozzárendelés az *NyA = vörös,* majd az előrenéző ellenőrzés törli a *vörös* értéket a szomszédos *ÉT* és *DA* változók tartományaiból. A *Q = zöld* hozzárendelés után a *zöld* is kikerül az *ÉT, DA* és *ÚDW* változók tartományaiból. A *V = kék*-et követően a *kék* is törlődik a *DA* és *ÚDW* változók tartományaiból, és *DA* számára nem marad megengedett érték. 129
5.7. Az AC-3 élkonzisztencia algoritmus. Az AC-3 alkalmazását követően vagy mindegyik él élkonzisztens, vagy néhány változó tartománya üres, azaz a kényszerkielégítési probléma nem hozható élkonzisztens alakra (tehát nem oldható meg). Az „AC-3” nevet az algoritmus kitalálója (Mackworth, 1977) vezette be, mert ezt az algoritmust mutatta be cikkében harmadikként. 129
5.8. A MIN-KONFLIKTUSOK algoritmus kényszerkielégítési problémák megoldására lokális kereséssel. A kezdeti állapotot véletlenszerűen választjuk ki, vagy egy mohó hozzárendelési folyamat segítségével, amely minden egyes sorra kerülő változónak egy minimális-konfliktus értéket választ. A KONFLIKTUSOK függvény megszámolja az adott érték által megsértett kényszerek számát (adottnak tekintve az aktuális hozzárendelések fennmaradó részét). 133
5.9. Egy kétlépéses megoldás a 8-királynő problémára a MIN-KONFLIKTUSOK felhasználásával. Mindegyik fázisban egy királynőnek keresünk új oszlopot. A konfliktusok számát (esetünkben a támadó pozícióban lévő királynők számát) mindegyik négyzetben feltüntettük. Az algoritmus a királynőket a MIN-KONFLIKTUSOK négyzetre viszi, véletlenszerűen törve fel ezzel a kötéseket. 134
5.10. (a) Egy faszerkezetű kényszerkielégítési probléma kényszergráfja. (b) Az *A* csomópont gyökérnek tekintésével konzisztens változók egy lineáris rendezése. 135
5.11. (a) Az 5.1. ábrán szereplő eredeti kényszergráf. (b) A kényszergráf *DA* eltávolítását követően. 135
5.12. Az 5.11. ábra (a) részén szereplő kényszergráf egy fa dekompozíciója 136
6.1. A 3 × 3-as amőbajáték (részleges) keresési fája. A legfelső csomópont a kiinduló állapot. MAX lép először, egy X-et téve valamelyik üres négyzetbe. A keresési fa egy részét mutatjuk, MIN (O) és MAX váltakozó, egymást követő lépéseit megadva, amíg el nem érjük a végállapotokat, melyekhez a játék szabályai szerint lehet hasznossági értékeket hozzárendelni. 144
6.2. Egy lépésváltásos játékfa. A Δ csomópontok MAX lépéseit, míg a ∇ csomópontok MIN lépéseit jelölik. A végcsomópontok a hasznossági függvénnyel számított hasznossági értékeket MAX szemszögéből mutatják, míg a többi csomópontnál a minimax értékét jelöltük be. MAX legjobb lépése a gyökérben a_1, mert ez vezet a legmagasabb minimax értékű követőhöz. MIN legjobb válasza b_1, mert ez vezet a minimális minimax értékű követőhöz. 145
6.3. Egy algoritmus a minimax döntések kiszámítására. Az algoritmus a lehető legjobb lépéshez tartozó operátort adja vissza, vagyis ahhoz a lépéshez tartozó operátort, amelyik a legnagyobb hasznossági értékkel rendelkező eredményre vezet, feltételezve, hogy az ellenfél úgy játszik, hogy minimalizálja a hasznossági értéket. A MAX-ÉRTÉK és MIN-ÉRTÉK függvények végigmennek a teljes játékfán, le egészen a levélcsomópontokig, hogy meghatározzák a csomópont felfelé terjesztett értékét. 147
6.4. Három játékos (*A, B, C*) játékfája a három első fél lépés esetén. Minden csomópontot az összes játékos szemszögéből számított értékkel címkéztük meg. A legjobb lépést a gyökérnél jelöltük be. 148
6.5. Az optimális döntés kiszámításának lépései a 6.2. ábrán látható játékfa esetén. Minden ponton a lehetséges értékek terjedelmét mutatjuk meg minden egyes csomópont számára. (a) A *B* csomópont alatt az első levélnek 3 az értéke. *B* tehát, ami egy MIN csomópont, *legfeljebb* 3 értékű. (b) A *B* alatti második csomópontnak 12 az értéke. MIN ezt a lépést elkerüli, így *B* értéke még mindig legfeljebb 3. (c) A *B* alatti harmadik levélnek 8 az értéke. *B* összes követőit láttuk már, *B* értéke tehát pontosan 3. Most azt

Created by XMLmind XSL-FO Converter.

következtethetjük ki, hogy a gyökér értéke *legalább* 3, mert MAX-nak a gyökérben 3-as értékű választása van. (d) A *C* alatti első levélnek 2 az értéke. *C* tehát, ami egy MIN csomópont, *legfeljebb* 2 értékű. De mi tudjuk már, hogy *B*-nek 3 az értéke, MAX tehát *C*-t soha nem fogja választani. Azért nem is érdemes *C* további követőit megvizsgálni. Ez az alfa-béta nyesés egy példája. (e) A *D* alatti első levélnek 14 az értéke, így *D* értéke *legfeljebb* 14. Ez több mint MAX legjobb alternatívája (azaz 3), meg kell vizsgálni ezért *D* követőit. Jegyezzük meg, hogy most a gyökér minden követője esetén rendelkezünk értékkorláttal, a gyökér értéke legfeljebb 14. (f) *D* második követőjének értéke 5, így tovább kell folytatnunk a vizsgálatot. A harmadik követő értéke 2, *D* értéke tehát pontosan 2. MAX döntése a gyökérnél, hogy *B* felé kell lépni, 3-as értékkel. 149
6.6. Alfa-béta nyesés: általános eset. Ha *m* a Játékos számára jobb, mint *n*, akkor a játék során sosem fogunk elérni *n*-be. 150
6.7. Az alfa-béta keresési algoritmus. Ugyanazt a számítást hajtja végre, mint egy normál minimax algoritmus a 6.3. ábrán, kivéve két sort a MAX-ÉRTÉK és a MIN-ÉRTÉK hívásokban, ahol az α és a β értékeket kezeli (és adminisztrálja e paraméterek átadását). 152
6.8. Két, kissé eltérő sakktáblaállás. (a) Feketének egy huszár és két gyalog előnye van, és a játékot megnyeri. (b) Fekete veszít, miután Fehér a vezérét leüti. 154
6.9. A horizonthatás. A fekete bástyával adott sorozatos sakk a „horizonton túlra” kényszeríti az elkerülhetetlen vezérré változtató lépést, és ezt a táblaállást enyhén előnyös állásként tünteti fel Fekete számára, holott Fehér számára ez egy biztos nyerő állás. 155
6.10. Egy tipikus ostáblaállás. A játék lényege, hogy az egyik fél összes bábuját eltávolítsuk a tábláról. Fehér az óramutató járásával egyező irányban halad a 25-ös felé, míg Fekete az óramutató járásával ellentétes irányban, a 0 felé halad. Egy bábuval bármelyik helyre lehet lépni, kivéve azokat a helyeket, ahol az ellenfélnek már legalább két bábuja található. Ha egy olyan helyre lépünk, ahol az ellenfélnek egy bábuja található, akkor azt a bábut elfogtuk, és a lépkedést az elejéről kell újrakezdenie. Ennél a táblaállásnál Fehér épp most dobott 6–5-öt, és négy megengedett lépés közül választhat: (5–10,5–11), (5–11,19–24), (5–10,10–16) és (5–11,11–16). 157
6.11. Egy ostáblaállás sematikus játékfája 158
6.12. A levélértékeken végrehajtott sorrendőrző transzformáció megváltoztatja a legjobb lépést . 160
6.13. Egy olyan egylépésváltásos játékfa, amire a minimax alkalmatlannak tűnik 165
6.14. Egy egyszerű játék kiinduló állása. Az *A* játékos indul elsőnek. Mindkét játékos felváltva lép és a zsetonját a szomszédos szabad helyre helyezheti, mindkét irányban. Ha a szomszédos mező foglalt, akkor a játékos az ellenfél felett átugorhat a következő szabad helyre, ha van ilyen. (Például ha *A* a 3-n és *B* a 2-n van, akkor *A* visszaléphet 1-re.) A játéknak vége, ha az egyik játékos eléri a tábla ellentétes végét. Ha az *A* játékos elsőnek éri el a 4-et, a játék értéke +1, ha *B* játékos éri el elsőnek az 1-et, a játék értéke az *A* számára –1. 170
6.15. Az a helyzet, amikor az algoritmus eldönti, hogy lenyesheti-e az n_j csomópontot 171
7.1. Egy általános tudásbázisú ágens 176
7.2. Egy tipikus wumpus világ. Az ágens a bal alsó sarokban van. 177
7.3. Az ágens első lépése a wumpus világban. (a) A kezdeti helyzet a [*Nincs*, *Nincs*, *Nincs*, *Nincs*, *Nincs*] érzékelése után. (b) Az első lépés után, érzékelve a [*Nincs*, *Szellő*, *Nincs*, *Nincs*, *Nincs*]-et. 179
7.4. Két későbbi helyzet az ágens előrehaladása során. (a) A harmadik lépést követően, miután [*Bűz*, *Nincs*, *Nincs*, *Nincs*, *Nincs*]-et érzékelt. (b) Az ötödik lépés és a [*Bűz*, *Szellő*, *Csillogás*, *Nincs*, *Nincs*] érzékelése után 180
7.5. Lehetséges modelljei a csapda jelenlétének az [1, 2], [2, 2] és [3, 1]-ben, ha adott a megfigyelés, hogy az [1, 1]-ben semmi és a [2, 1]-ben szellő érezhető. (a) A tudásbázis és α1(nincs csapda[1, 2]-ben) modelljei. (b) A tudásbázis és *α*2(nincs csapda[2, 2]-ben) modelljei. 182
7.6. A mondatok az ágens fizikai konfigurációi, és a következtetés az a folyamat, amely új fizikai konfigurációkat hoz létre régiekből. A logikai következtetésnek biztosítania kell, hogy az új konfigurációk olyan aspektusait reprezentálják a világnak, amelyek ténylegesen is következnek azokból az aspektusokból, amelyeket a régi konfigurációk reprezentálnak. 183
7.7. Ítéletkalkulus-beli mondatok BNF (Backus–Naur-forma) nyelvtana 184
7.8. Az öt logikai összekötőjel igazságtáblája. Amikor a táblát használjuk például a $P \vee Q$ értékének számítására, ha *P* igaz és *Q* hamis, akkor először megkeressük azt a sort, amelyben *P igaz* és *Q hamis* (a harmadik sor). Ezután a sorban megkeressük a $P \vee Q$ alatti oszlopot, hogy megtaláljuk az eredményt: *igaz*. Tekinthetjük a táblázat úgy is, hogy minden sor egy modell, és az egyes oszlopbeli elemek az adott sorban azt mondják meg, hogy a megfelelő mondat igaz-e az adott modellben. 186
7.9. A tudásbázis alapján épített igazságtábla látható az ábrán. A *TB* igaz, ha Sz_1-től és Sz_5-ig igaz, amely a 128 sorból csak 3-ban fordul elő. Mind a 3 sorban $C_{1,2}$ hamis, tehát nincsen csapda az [1, 2]-ben. Viszont lehet, hogy van csapda [2, 2]-ben (bár lehet, hogy nincs). 187

Created by XMLmind XSL-FO Converter.

7.10. Egy igazságtábla felsoroló algoritmus ítéletkalkulus állítás vonzatának eldöntésére. Az IT az igazságtáblát jelöli. A IK-IGAZ? igazat ad vissza, ha a mondatot tartalmazza a modell. A *modell* változó reprezentál egy részleges modellt, egy hozzárendelést a változók egy részéhez. A KIEGÉSZÍT (*P*, *igaz*, *modell*) függvény egy új részleges modellt ad vissza, amelyben *P* értéke igaz. 187
7.11. Standard logikai ekvivalenciák. Az α, β, γ szimbólumok tetszőleges ítéletkalkulus mondatokat jelölnek. 188
7.12. Egy egyszerű rezolúciós algoritmus az ítéletkalkulushoz. Az IK-REZOLVÁLÁS a két bemenetként megkapott állítás rezolválásából származó összes lehetséges klóz halmazát adja vissza. 194
7.12. Az IK-REZOLÚCIÓ algoritmus részleges alkalmazása egy egyszerű wumpus világbeli következtetésre. A felső sor első négy klóza alapján származtatjuk a $\neg P_{1,2}$-t. 194
7.14. Előrefelé láncolás algoritmus az ítéletkalkulus számára. Az *agenda* tartalmazza azokat a szimbólumokat, amelyek ismerten igazak, de még nem „dolgozták fel” őket. A *számol* tábla követi, hogy az egyes implikációknak hány premisszája ismeretlen. Ha az agendáról egy új *p* szimbólumot feldolgozunk, minden olyan implikáció számlálója csökken eggyel, amelynek premisszájában *p* megjelenik. (Ez megoldható konstans időben, ha a *TB* megfelelően indexelt.) Ha a számláló eléri a nullát, azaz az implikáció minden premisszája ismert, akkor az implikáció konklúzióját hozzá lehet adni az agendához. Végül szükségünk van arra, hogy kövessük melyik szimbólumot dolgoztuk már fel; egy kikövetkeztetett szimbólumot nem kell hozzáadni az agendához, ha azt már korábban sikeresen feldolgoztuk. Így elkerülhető a redundáns munka, és megakadályozza a végtelen ciklusok kialakulását, amelyet olyan implikációk okozhatnak, mint a $P \Rightarrow Q$ és $Q \Rightarrow P$. 196
7.15. (a) Egyszerű Horn-klózokból álló tudásbázis. (b) Ugyanez a tudásbázis ÉS-VAGY gráfként. 197
7.16. A DPLL algoritmus ítéletkalkulus mondatok kielégíthetőségének ellenőrzése. A TISZTA-SZIMBÓLUM-KERESÉS és az EGYSÉG-KLÓZ-KERESÉS eljárásokat a szövegben elmagyaráztuk; mindkettő egy szimbólummal vagy nullával tér vissza, és a szimbólumhoz hozzárendelendő igazságértékkel. Mint az IT-VONZAT? eljárás, ez is részleges modelleken dolgozik. 200
7.17. A WALKSAT algoritmus, kielégíthetőségének ellenőrzése véletlenszerűen cserélgetetett változó értékekkel. Számos változata ismert az algoritmusnak. 201
7.18. (a) A grafikon a klóz/szimbólum arány függvényében annak valószínűségét mutatja, hogy $n = 50$ mondat közül véletlenszerűen választott 3-CNF mondat kielégíthető-e. (b) A grafikon a DPLL és a WALKSAT algoritmusok futási időinek középértékét mutatja 100 *kielégíthető* 3-CNF mondaton $n = 50$ mellett, a kritikus m/n arány melletti szűk sávban. 202
7.19. (Egy wumpusvilág-beli ágens programja, amely ítéletkalkulust használ a csapdák, a wumpusok és a biztonságos négyzetek azonosítására. Az ÚTVONAL-PROBLÉMA szubrutin létrehoz egy keresési problémát, aminek megoldása egy cselekvéssorozat, amely elvezet az [*x*, *y*] négyzetből az [*i*, *j*]-be, és csak korábban már látogatott négyzeteken vezet keresztül. 203
7.20. Egy áramköralapú, wumpusvilág-beli ágens egy része, a bemeneteket és a kimeneteket mutatva, az arany megragadását megvalósító áramkör, és az áramkör, amely meghatározza, hogy a wumpus él-e. A regisztereket négyzet jelöli, és a késleltetőket kis háromszögek mutatják. 205
7.21. Egy áramkör, amely meghatározza, hogy az ágens az [1, 1]-ben van-e. Minden pozíció és orientáció regiszterhez egy hasonló áramkör csatlakozik. 206
8.1. Formális nyelvek, valamint azok ontológiai és ismeretelméleti hozzárendelései 219
8.2. Egy öt objektumot – két bináris relációt, három egyelemű relációt (címkékkel jelezve az objektumokon) és egy egyargumentumú függvényt, bal láb – tartalmazó modell 220
8.3. Az elsőrendű logika (egyenlőséggel kiegészített) szintaxisa Backus–Naur-forma jelöléseivel. (Lásd az 1. szakasz - B1. Nyelvek definiálása Backus–Naur-Formában (BNF) részben, ha ez a jelölés ismeretlen.) A szintaxis szigorú a zárójelezés tekintetében; a zárójelekről és az operátori elsőbbségről a 3.1. szakasz - Szintaxis részben szereplő megjegyzések ugyanúgy vonatkoznak az elsőrendű logikára is. 221
8.4. Egy C_1-es digitális áramkör, amelynek az a célja, hogy egy egybites teljes összeadást végezzen. Az első két bemenet az a két bit, amit össze kell adni, míg a harmadik bemenet az átvitel. Az első kimenet az összeg, míg a második kimenet az átvitel a következő összeadó felé. Az áramkör két XOR, két AND és egy OR kaput tartalmaz. 235
8.5. Egy tipikus családfa. Az „=” szimbólum a házastársakat köti össze, a nyilak a gyerekekre mutatnak. 242
8.6. Egy négybites összeadó 243
9.1. Az egyesítési algoritmus. Az algoritmus összehasonlítja a bemenetek felépítését elemről elemre. A θ helyettesítés, amely az EGYESÍTÉS argumentuma, útközben épül fel, és arra használjuk, hogy meggyőződjünk arról, hogy a későbbi összehasonlítások konzisztensek lesznek az általunk előzőleg létrehozott lekötésekkel. Egy összetett kifejezésben, mint például az *F*(*A*, *B*), a VÁL függvény kiveszi az *F* függvényszimbólumot, és az ARGOK függvény kiveszi az (*A*, *B*) argumentumlistát. 248

Created by XMLmind XSL-FO Converter.

9.2. (a) A bennfoglalási háló, amelynek a legalacsonyabb csomópontja ez a mondat: *Alkalmaz(AIMA.org, Richárd)*. (b) A bennfoglalási háló arra a mondatra, hogy: *Alkalmaz(János, János)*. 250
9.3. Egy koncepcionálisan egyszerű, de kevéssé hatékony előrefelé láncolási algoritmus. Minden egyes iterációban hozzáadja a *TB*-hez az összes atomi mondatot, amelyet egy lépésben az implikációs mondatokból és a *TB*-ben már meglévő atomi mondatokból kikövetkeztethetünk. 252
9.4. Az előrefelé láncolás algoritmus által generált bizonyítási fa a bűntény példára. A kiinduló tények az alsó sorban jelennek meg, az első iterációban kikövetkeztetett tények a középső sorban, míg a második iterációban kikövetkeztetett tények a legfelső sorban. 253
9.5. (a) Ausztrália térképének kiszínezését bemutató kényszergráf (5.1. ábra). (b) A térképszínező kényszerkielégítési probléma egy határozott klózzal reprezentálva. Vegyük észre, hogy a változók tárgyterülete a KÜL-re megadott alaptények konstansai által implicit módon definiáltak. 255
9.6. Egy egyszerű hátrafelé láncolási algoritmus 257
9.7. Egy hátrafelé láncolással létrehozott bizonyítási fa, annak bizonyítására, hogy West egy bűnöző. A fát mélységi kereséssel kell kiértékelni, balról jobbra. Hogy bebizonyítsuk a *Bűnöző*(*West*) klózt, be kell bizonyítanunk az alatta elhelyezkedő négy konjunktot is. Néhányan közülük megtalálhatók a tudásbázisban, míg mások további hátrafelé láncolást igényelnek. Mindegy egyes sikeres egyesítés lekötéseit láthatjuk a megfelelő részcél mellett. Jegyezzük meg, hogy mihelyt egy konjunkcióban egy részcél sikeres, helyettesítéseit a következő részcélokhoz alkalmazzuk. Így tehát, mire az ERL-HL-KÉRDEZ eljut az utolsó konjunkcióig, az eredetileg *Ellenséges*(*z*)-ig, az már a *Nono*-hoz lesz lekötve. 258
9.8. A Csatol predikátum fordításának pszeudokódos reprezentációja. Az ÚJ-VÁLTOZÓ függvény egy új változót ad vissza, amely különbözik a korábban visszaadott változóktól. A HÍV (*folytatás*) eljárás folytatja a végrehajtást a megadott folytatással. 261
9.9. (a) Az *A*-ból a *C*-be vezető útvonal megtalálása a Prologot egy végtelen hurokba vezetheti. (b) Egy gráf, amelyben minden csomópontot két véletlenszerű következő rétegbeli csomóponthoz kapcsolunk a következő rétegben. Egy útvonal megtalálása az A_1-ből a J_4-be 877 következtetést igényel. 262
9.10. (a) Az *A*-ból a *C*-be vezető útvonal bizonyítása. (b) Végtelen mélységű bizonyítási fa, amely akkor generálódik, ha a klózok „rossz” sorrendben vannak. 262
9.11. Egy rezolúciós bizonyítás arra, hogy West bűnöző 266
9.12. Egy rezolúciós bizonyítása annak, hogy a Kíváncsiság ölte meg a macskát. Figyeljük meg a faktorálás használatát a *Szereti*(*G*(*Jankó*), *Jankó*) klóz származtatásánál. 268
9.13. A rezolúcióteljesség bizonyításának a szerkezete 269
9.14. Az OTTER tételbizonyítás vázlata. A heurisztikus kontrollt alkalmazzák a „legkönnyebb” klóz kiválasztására és a FILTER funkcióban, amely kitörli a jelentéktelen klózokat a további vizsgálatból. 274
10.1. A világ ontológiájának felsőbb szintjei, amelyek a fejezetben később tárgyalt témákat mutatják. Minden él azt jelzi, hogy az alsó fogalom a felső fogalom egy specializálódása. 287
10.2. Szituációkalkulusban minden szituáció (az S_0-t kivéve) valamilyen cselekvés eredménye ... 294
10.3. Általánosított események. Az univerzumnak térbeli és időbeli dimenziója van. Ezen az ábrán csak egy térbeli dimenziót mutatunk. Minden esemény az univerzum *Része*. Az olyan esemény, mint a *MásodikVilágHáború* a tér-idő egy tartományában történik, melynek határai kissé szabadon választhatók, és időben változók. Egy *Intervallum*, mint amilyen a *HuszadikSzázad*, rögzített és korlátos időbeli, valamint maximális térbeli kiterjedéssel rendelkezik. A *Hely*, amilyen például Ausztrália, nagyjából rögzített térbeli és maximális időbeli kiterjedéssel rendelkezik. 299
10.4. Összetett események ábrázolása. (a) *T*(*Mindkettő*(*p, q*), *i*), másképpen *T*(*p* o *q*, *i*), (*b*) *T*(*Egyik*(*p, q*), *i*), (*c*) *T*(*VagyVagy*(*p, q*), *i*). 301
10.5. Az időintervallumok predikátumai 302
10.6. Az *Elnök*(*USA*) objektum sémaszerű ábrázolása létezésének első 15 évében 303
10.7. Egy absztrakt online bolt weblapja, ahogy egy böngészőt használó ember látja (felül), és a hozzá tartozó HTML-füzér, ahogy azt a böngésző vagy a vásárló ágens látja (alul). A HTML-ben a < és a > közötti karakterek a jelölő direktívák, amelyek meghatározzák, hogy a lapot hogyan kell kijelezni. Az Válasszon füzér például jelzi, hogy dőlt betűre kell átkapcsolni, a *Válasszon* szót kijelezni és a dőlt betű használatát befejezni. Egy lapazonosító, mint amilyen a http://absz-bolt.com/zene az ún. **egységes erőforrás azonosító** (**uniform resource locator**, **URL**).Az *hivatkozás* bejelölés jelzi, hogy az *url* felé létesíteni kell egy hipertextkapcsolatot a **hivatkozó szöveg** (**anchor text**) *hivatkozással*. 307
10.8. (a) A termékkategóriák taxonómiája. (b) Ezen kategóriák számára a hivatkozó szövegek. .. 309
10.9. Egy szemantikus háló négy objektummal (János, Mária, 1 és 2) és négy kategóriával. A relációkat a címkézett élek jelölik 312
10.10. A szemantikus háló egy részlete, amely a *Repül*(*Shankar, NewYork, Újdelhi, Tegnap*) logikai állítás reprezentációját valósítja meg 313
10.11. A leírások szintaxisa a CLASSIC nyelv egy részhalmazában 315

Created by XMLmind XSL-FO Converter.

10.12. Horn-klózok egy halmazának Clark-lezárása. Az eredeti Horn-program (balra) négy tantárgyat említ explicit módon, és azt is állítja, hogy minden egészre a 101 és a 130 között létezik egy matematikai tárgy, meg azt is, hogy minden SZT tárgyhoz a 100-as (BSc) sorozatban létezik egy megfelelő tantárgy a 200-as (MSc) sorozatban. A Clark-lezárás (jobbra) azt mondja, hogy más tantárgy nincs is. A lezárással és az egyedi elnevezések feltételezéssel (valamint az *Egész* predikátum nyilvánvaló definíciójával) együtt eljutunk a kívánt konklúzióig, miszerint pontosan 36 tantárgy van: 30 matematikai és 6 SZT tárgy. 317
11.1. A *Strips* és az ADL nyelv összehasonlítása a tervkészítési feladatok reprezentációjának szempontjából. Mindkét esetben a célok úgy viselkednek, mint egy paraméterek nélküli cselekvés előfeltételei. 337
11.2. A STRIPS-probléma repülőterek közötti légi teherszállítási feladathoz 338
11.3. Az egyszerű pótkerék probléma 338
11.4. A kockavilág tervkészítési problémája: egy három kockából álló torony építése. A [*Mozgat*(*B, Asztal, C*), *Mozgat*(*A, Asztal, B*)] cselekvéssor egy lehetséges megoldás. 340
11.5. A tervkészítés két megközelítése. (a) Előrefelé (progresszív) állapottér-keresés, a kiinduló állapotból indulva és a probléma cselekvéseit használva halad a cél felé. (b) Visszafelé (regressziós) állapottér-keresés: valószínűségi állapot keresés (lásd 6.1. szakasz - Szenzor nélküli problémák részben) a célállapot(ok)ból indulva a cselekvések inverzét alkalmazva keresi visszafelé a kezdeti állapotot. 341
11.6. Egy részben rendezett terv a zokni és a cipő felvételéhez, és a hat lehetséges sorba rendezés a teljesen rendezett tervhez 344
11.7. Az egyszerű kerékcsere probléma 347
11.8. A nem teljes részben rendezett terv a kerékcsere problémához, miután megválasztottuk a cselekvéseket az első két nyitott előfeltételhez. A dobozok cselekvéseket testesítenek meg, a bal oldalon az előfeltételekkel, a jobb oldalon pedig a következményekkel. (Az *Indít* cselekvést kivéve a következményeket elhagytuk.) A fekete nyilak az okozati kapcsolatokat mutatják, melyek a nyíl fejénél szereplő állítást védik. 347
11.9. A terv alakulása, az *OtthagyÉjszakára* választása után, amit a ¬Ott(*LaposKerék, Tengely*) eléréséhez választottunk. Hogy elkerüljük az ütközést az *Ott*(*Pótkerék, Csomagtartó*)-t védő, az *Eltávolít*(*Pótkerék, Csomagtartó*)-ból induló okozati kapcsolattal, az *OtthagyÉjszakára* cselekvést egy rendezési megkötéssel az *Eltávolít*(*Pótkerék, Csomagtartó*) elé helyeztük, amit a szaggatott nyíllal jelöltünk. 348
11.10. A kerékcsere probléma végső megoldása. Vegyük észre, hogy az *Eltávolít*(*Pótkerék, Csomagtartó*) és az *Eltávolít*(*LaposKerék, Tengely*) tetszőleges sorrendben végrehajtható a *Felszerel*(*Pótkerék, Tengely*) előtt. 349
11.11. A „legyen süti és együnk is” probléma 351
11.12. A „legyen süti és együnk is” probléma tervkészítési gráfja az S_2 szintig. A téglalapok az akciókat jelentik (a kicsi négyzetek a megőrző cselekvések), az egyenesek az előfeltételeket és a következményeket jelölik. A kölcsönös kizárásokat ívelt szürke vonalak jelölik. 351
11.13. A GRAPHPLAN algoritmus. A GRAPHPLAN algoritmus egy megoldás kinyerő és egy gráf-bővítő lépés között alternál. A MEGOLDÁS-KINYERÉS a befejezéstől visszafelé keresve ellenőrzi, hogy található-e megoldás. A GRÁF-BŐVÍTÉS adja a cselekvéseket az adott szinthez és az állapotliterálokat a következőhöz. 353
11.14. A kerékcsere probléma tervkészítési gráfja az S_2 szintre bővítés után. A mutex kapcsolatokat szürke vonalak jelölik. Csak néhány fontos mutexet mutatunk, mert az ábra olvashatatlanná válna, ha az összeset jelölnénk. A megoldást megvastagított vonalak és kiemelések jelölik. 354
11.15. A SATPLAN algoritmus. A tervkészítési feladatot egy konjunktív normál formájú mondatra fordítjuk le, amelyben minden cél egy megadott *T* időpillanatban teljesül, és *T*-ig minden lépésre tartalmazza az axiómákat. (A fordítás részleteit a szövegben írjuk le.) Ha a kielégíthetőségi algoritmus talál modellt, akkor a tervet a modellben az *igaz* értéket kapott ítéletlogikai szimbólumokhoz tartozó cselekvések kinyerésével kapjuk meg. Ha nem létezik modell, akkor a folyamatot ismételjük, a célt egy lépéssel későbbre mozgatva. 356
11.16. A Sussman-anomália kockavilág tervkészítési probléma 365
11.17. Shakey világa. Shakey képes egy szobán belül mozogni, át tud menni a szobák közötti ajtókon, fel tud mászni tárgyakra, el tudja tolni a mozdítható tárgyakat, valamint a villanyt tudja kapcsolni. . 366
12.1. A két autó összeszerelését tartalmazó ütemezési feladat. A jelölésben az *Időtartam*(*d*) jelentése, hogy egy cselekvés *d* percet vesz igénybe. A *Motor*(E_1, C_1, 60) jelentése az, hogy az E_1 egy motor, ami a C_1 alvázba illeszthető és 60 perc alatt szerelhető be. 369
12.2. A 12.1. ábrán szereplő ütemezési probléma megoldása. Az ábra tetején a részben rendezett terv megoldása látható. Minden cselekvés hozza a téglalap alján látható, a legkorábbi és a legkésőbbi kezdési időkkel [*ES, LS*] a bal felső sarokban. A két szám között különbség a cselekvés mozgástere, a 0 mozgásterű cselekvések a kritikus útvonalon vannak, melyeket vastag vonallal jelölünk. Az ábra alján ugyanezen megoldás időrendjét láthatjuk. A szürke téglalapok azt az intervallumot jelölik, ami alatt a

Created by XMLmind XSL-FO Converter.

cselekvés végrehajtható, feltételezve, hogy a sorrendezési megkötéseket betartjuk. A szürke téglalapok nem felhasznált, nem kitöltött területei a mozgásteret jelölik. 370
12.3. Két autó összeszerelésének ütemezési feladata erőforrásokkal. Egy összeszerelő állomás, egy kerékszerelő-állomás és két vizsgálóbiztos az összes elérhető erőforrás. Az ERŐFORRÁS:*r* jelölés jelentése, hogy az *r* erőforrást egy cselekvés végrehajtása alatt használjuk, de a cselekvés befejezése után újra szabad. 372
12.4. A 12.3. ábra erőforrásokat is tartalmazó ütemezési feladatának megoldása. Az ábra bal széle az erőforrásokat sorolja fel, a cselekvések az általuk használt erőforrásokkal egy sorban jelennek meg. Attól függően, hogy melyik összeszerelés használja a motorszerelő-állomást először, két lehetséges ütemezés létezik. Mi az optimális megoldást ábrázoltuk, ami 115 percet vesz igénybe. 372
12.5. A *HázatÉpít* cselekvés egy lehetséges dekompozíciója 374
12.6. A házépítési probléma cselekvéseinek leírása és a *HázatÉpít* cselekvés részletes dekompozíciója. A leírások a pénzzel kapcsolatban egy egyszerűsített, az építőkkel kapcsolatban egy optimista nézetet alkalmaznak. 375
12.7. Egy magas szintű cselekvés dekompozíciója egy létező tervben. A *HázatÉpít* cselekvést a 12.5. ábra dekompozíciójával helyettesítjük. A *Telek* külső előfeltételt a már létező *TelketVesz*-ből kiinduló okozati kapcsolat biztosítja. A *Pénz* külső előfeltétel nyitott marad a dekompozíciós lépés után, ezért a *KölcsöntVeszFel* cselekvést szúrjuk be. 376
12.8. Az O. Henrik történetből kiragadott *Mágusok ajándéka* probléma egy inkonzisztens absztrakt tervet mutat, ami azonban dekomponálható egy konzisztens megoldásra. Az (a) ábra a problémát mutatja be: egy szegény házaspárnak csak két értékes tulajdona van. A férfinek egy aranyórája, a nőnek pedig a gyönyörű hosszú haja. Mindketten azt tervezik, hogy ajándékot vásárolnak a másiknak, hogy az boldog legyen. A férfi úgy dönt, hogy az óráját egy ezüstfésűre cseréli be, míg a nő eladja a haját, hogy aranyláncot vegyen az órához. (Feltételezzük, hogy a „Fésűt ad” cselekvés előfeltétele a *Haj*, mivel ha a feleségnek nincs hosszú haja, a cselekvés nem éri el a kívánt hatást, hogy boldoggá tegye; és hasonlóképpen a „Láncot ad” cselekvésre.) A (b) ábrán szereplő részleges terv inkonzisztens, mert a „Fésűt ad” és „Láncot ad” absztrakt lépések nem sorrendezhetők konfliktus nélkül. (c) Dekomponáljuk a „Fésűt ad” lépést egy „beilleszt terv” metódussal. A dekompozíció első lépésében a férj megszerzi a fésűt, és odaadja feleségének, miközben az órát egy későbbi időpontban adja oda fizetségül. A második lépésben az órát átadja, és a kötelezettséget teljesíti. Egy hasonló módszer dekomponálja a „Láncot ad” lépést. Amíg mindkét odaadó lépést a szállítási lépés elé sorrendezzük, ez a dekompozíció megoldja a feladatot. (Vegyük észre, hogy ez azon múlik, hogy a lánc használata az órához vagy a fésű használata a hajhoz boldogságot okoz még akkor is, ha a tulajdonjogot már elvesztették.) 378
12.9. A „dupla-Murphy” porszívóvilág keresési fájának első két szintje. Az állapotcsomópontokban és a VAGY csomópontokban cselekvéseket kell választani. A valószínűségi csomópontok, amelyeket körökkel jelöltünk ÉS csomópontok, ahol, ahogy azt a kimenő ágakon szereplő ív is jelöli, minden kimenetelt kezelni kell. A megoldást vastag vonallal jelöltük. 383
12.10. Egy algoritmus a nemdeterminisztikus környezetek által generált ÉS-VAGY gráfok keresésére. Feltételezzük, hogy az ÁLLAPOTÁTMENET függvény a cselekvések egy listáját adja vissza, melyek mindegyike egy lehetséges kimenetel *halmazhoz* tartozik. A cél egy feltételes terv megtalálása, ami bármilyen körülmények között elér egy célállapotot. 384
12.11. A „tripla-Murphy” porszívóvilág keresési gráfjának első szintje, ahol a ciklusokat explicit megjelöltük. A probléma összes megoldása ciklikus terv. 385
12.12. A „váltakozó dupla-Murphy” porszívóvilág ÉS-VAGY gráfjának egy részét mutatja be, melyben a tiszta négyzetet elhagyó ágens piszkot hagyhat maga után. Az ágens nem tudja érzékelni a más négyzetben levő piszkot. 387
12.13. Egy cselekvésmonitorozó és -újratervező ágens. Ez szubrutinként a TERVKÉSZÍTŐ teljes állapottér tervkészítő algoritmust használja. Ha a következő cselekvés előfeltételei nem teljesülnek, az ágens a *teljes_terv* lehetséges *p* pontjain iterál olyat keresve, amelyhez a TERVKÉSZÍTŐ útvonalat tud tervezni. Ezt az útvonalat hívjuk *javításnak*. Ha a TERVKÉSZÍTŐ sikeres a javításban, az új terv készítéséhez összefűzi a javítást és a terv *p* utáni részét. Az ágens ezután a terv első lépését adja vissza. 390
12.14. A végrehajtás előtt a tervkészítő egy *teljes_terv*-nek nevezett tervet készít, mely *S*-ből *G*-be vezet. Az ágens az *E*-vel jelölt pontig végrehajtja a tervet. A *terv* hátralevő részének végrehajtása előtt a szokásos módon ellenőrzi az előfeltételeket, és azt találja, hogy valójában az *O* állapotban van, és nem az *E*-ben. Ezután meghívja a tervkészítő algoritmust, hogy készítsen egy *javítás*-t, mely egy terv az *O*-ból az eredeti *teljes_terv* egy *P* pontjához. Az új *terv* ezután a *javítás* és a *folytatás* (az eredeti *teljes_terv* maradék része) összefűzéséből adódik. 391
12.15. Az állapotok sora, amint a folytonos tervkészítő ágens megpróbálja a (d)-nek megfelelő *Rajta*(*C*, *D*) ∧ *Rajta*(*D*, *B*) célállapotot elérni. A kiinduló állapot az (a). A (b) állapotnál egy másik ágens közbelépett és (d)-t (b)-re helyezte. A (c) állapotban az ágens a *Mozgat*(*C*, *D*) cselekvést hajtotta végre,

Created by XMLmind XSL-FO Converter.

ami sikertelen volt, (c)-t az (a)-ra ejtette. Újrapróbálja a *Mozgat*(*C*, *D*) cselekvést, amivel eléri a (d) célállapotot. 394
12.16. A folytonos tervkészítő ágens által készített kiinduló terv. A terv egyelőre nem különbözik egy átlagos részben rendezett tervkészítő által készítettől. 394
12.17. Miután valaki más elmozdította a *D* kockát a *B*-re, a *Tiszta*(*B*) és a *Rajta*(*D*, *G*) biztosítására szolgáló kapcsolatokat eldobjuk, ami ezt a tervet eredményezi 394
12.18. A *Mozgat*(*C*, *D*) által létrehozott kapcsolatot helyettesítettük az *Indít* kapcsolatával, a most már redundáns *Mozgat*(*D*, *B*) lépést pedig elhagytuk 395
12.19. A *Mozgat*(*C*, *D*) végrehajtása és a tervből való eltávolítása után az *Indít* lépés következményei azt a tényt tükrözik, hogy a *C* a várt *D* helyett az *A* kockára került. A *Rajta*(*C*, *D*) cél előfeltétel még minding nyitott. 395
12.20. A nyitott előfeltételt a *Mozgat*(*C*, *D*) ismételt hozzáadásával oldjuk fel. Vegyük észre az előfeltételekhez szükséges új kapcsolatokat. 395
12.21. A *Mozgat*(*C*, *D*) végrehajtása és tervből eltávolítása után a *Rajta*(*C*, *D*) megmaradt nyitott feltételt az *Indít* lépésből induló okozati kapcsolat hozzáadásával oldjuk fel. A terv most teljes. 395
12.22. A `FOLYTONOS-RRT-ÁGENS`, egy folytonos, részben rendezett tervkészítő ágens. Egy érzékelés után az ágens eltávolít egy lépéssorozatot a folytonosan javított tervéből, és visszaad egy cselekvést. Gyakran számos cselekvéssor-eltávolító tervkészítési lépésre van szükség, mialatt csak a *NoOp*-ot adja vissza, mielőtt készen állna egy valós cselekvés végrehajtására. 396
12.23. A páros tenisz probléma. Két ágens együtt játszik, és négy pozíció egyikében lehetnek: [*Bal*, *Alapvonal*], [*Jobb*, *Alapvonal*], [*Bal*, *Háló*] és [*Jobb*, *Háló*]. A labda visszaadható, ha pontosan egy játékos van a megfelelő helyen. 397
13.1. Döntéselméleti ágens, amely racionális cselekvéseket választ ki. A lépéseket a következő öt fejezetben részletezzük. 411
13.2. Mivel az 1. ágens meggyőződése nem következetes, ezért a 2. ágens tud úgy tételeket tenni, amely *a* és *b* kimenetelétől függetlenül biztosítja, hogy az 1. ágens veszítsen 418
13.3. A *Fogfájás*, *Lyuk*, *Beakadás* világ egy teljes együttes valószínűség-eloszlása 418
13.4. A valószínűségi következtetés egy olyan algoritmusa, amely valamely teljes együttes valószínűségi eloszlás bejegyzéseit veszi számba 420
13.5. Két példa nagyméretű együttes eloszlások kisebb eloszlásokra való felbontására az abszolút függetlenség alapján. (a) Az időjárás és a fogászati problémák függetlenek. (b) A pénzfeldobások függetlenek. 421
13.6. (a) Az ágens beragad, miután [1, 2]-ben és [2, 1]-ben is szellőt észlel – nincs feltárható biztonságos hely. (b) A négyzetek szétosztása *Ismert*, *Perem* és *Egyéb* csoportokba az [1, 3]-ra vonatkozó lekérdezéshez. 426
13.7. $C_{2,2}$ és $C_{3,1}$ peremváltozók, az egyes modellek C(*perem*) értékét mutató konzisztens modelljei: (a) három, két vagy három csapdát jelző modell $C_{1,3}$ = *igaz* mellett, és (b) két, egy vagy két csapdát jelző modell $C_{1,3}$ = *hamis* mellett. 427
14.1. Egy egyszerű Bayes-háló, amelyben az *Időjárás* független a többi három változótól, a *Fogfájás* és a *Beakadás* pedig feltételesen függetlenek a *Lyuk* ismeretében 435
14.2. Egy tipikus Bayes-háló, amely a topológiát és a feltételes valószínűségi táblákat (FVT) is mutatja. Az FVT-kben *B*, *F*, *R*, *J* és *M* szerepel *Betörés*, *Földrengés*, *Riasztás*, *JánosTelefonál* és *MáriaTelefonál* helyett. 435
14.3. A háló struktúrája függ a hozzáadás sorrendjétől. A csomópontokat mindegyik hálóhoz felülről lefelé haladva adtuk hozzá. 438
14.4. (a) Az *X* csomópont feltételesen független a nem leszármazottaitól (Z_{ij}-ktől) a szülei (a szürke területen látható U_i-k) ismeretében. (b) Az *X* csomópont feltételesen független a háló összes többi csomópontjától a Markov-takarójának ismeretében (a szürke terület). 440
14.5. Egy egyszerű háló diszkrét (*Támogatás* és *Vásárol*) és folytonos (*Termés* és *Ár*) változókkal 442
14.6. Az (a) és (b) grafikonok az *Ár* valószínűség-eloszlását mutatják a *Termés* függvényében és a *Támogatás* igaz és hamis értéke mellett. A (c) diagram a P(*Ár*| *Termés*) eloszlást mutatja, ami a két aleset összegzéseként adódik. 443
14.7. (a) A *Vásárlás* valószínűségének probit eloszlása az *Ár* ismeretében, $\mu = 6{,}0$ és $\sigma = 1{,}0$ mellett. (b) Logit eloszlás hasonló paraméterekkel. 443
14.8. A (14.3) egyenletben szereplő kifejezés struktúrája. A kiértékelés fentről lefelé halad, összeszorozva az értékeket az egyes útvonalak mentén, és összegezve a „+” csomópontoknál. Vegyük észre, a *j*-hez és az *m*-hez tartozó útvonalak megismétlődését. 445
14.9. A felsoroló algoritmus Bayes-hálós lekérdezések megválaszolására 446
14.10. A változó eliminálás algoritmus Bayes-hálós lekérdezések megválaszolására 448

Created by XMLmind XSL-FO Converter.

14.11. (a) Egy többszörösen összekötött háló feltételes valószínűségi táblákkal. (b) Egy többszörösen összekötött háló csoportosított ekvivalense. 450
14.12. Egy mintavételező algoritmus, amely eseményeket generál egy Bayes-hálóból 451
14.13. Az elutasító mintavétel algoritmusa, ami Bayes-hálós lekérdezéseket válaszol meg bizonyítékok esetén 452
14.14. A valószínűségi súlyozás algoritmusa Bayes-hálóban történő következtetéshez 454
14.15. Az MCMC algoritmus Bayes-hálóban történő közelítő következtetéshez 455
14.16. (a) Egy két osztályt (*Professzor* és *Diák*) leíró RVM. Két professzor és két diák van, mindkét diák konzulense *KovácsProf*. (b) Az (a)-beli RVM Bayes-háló ekvivalense. 459
14.17. Egy Bayes-háló részlete, ami egy olyan RVM-hez tartozik, amelyben a *Konzulens*(*János*) ismeretlen, de vagy *KovácsProf*, vagy *MórProf*. A konzulens választása az egyes professzorok finanszírozásától függ. Vegyük észre, hogy a *Siker*(*János*) most *mind a két* professzor *Hírnev*-étől függ, bár a *Konzulens*(*János*) értéke meghatározza, hogy valójában melyiknek is van hatása. 460
14.18. Egy gépkocsi elektromos és motorikus rendszereinek részeit leíró Bayes-háló. Minden változó bináris, és az *igaz* érték jelzi a megfelelő funkció helyes működését. 469
14.18. Három lehetséges háló a távcsőproblémára 470
15.1. (a) Egy elsőrendű Markov-folyamathoz tartozó Bayes-hálóstruktúra. A reprezentált Markov-folyamatban az állapotot az $\mathbf{X}_t$ változók definiálják. (b) Egy másodrendű Markov-folyamat. 475
15.2. Az esernyős világot leíró Bayes-hálóstruktúra és feltételes eloszlások. Az álapotátmenet-modell a $P(Eső_t|Eső_{t-1})$ feltételes valószínűség-eloszlás, az érzékelő modell a $P(Esernyőt_t|Eső_t)$. 476
15.3. A simítás egy múltbeli k időpontban az állapot a posteriori $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ eloszlását számítja ki egy teljes, 1-től t-ig terjedő megfigyelés sorozat ismeretében 479
15.4. Az előre-hátra algoritmus, ami állapotok egy sorozatának az a posteriori valószínűségeit számítja ki adott megfigyelési sorozat ismeretében. Az ELŐRE és a HÁTRA műveleteket rendre a (15.3) és (15.7) egyenletek definiálják. 480
15.5. (a) Az *Eső*$_t$ lehetséges állapotsorozatait tekinthetjük egy átvezető útnak egy gráfon, aminek csomópontjai a lehetséges állapotok az egyes időpontokban. (Az állapotokat négyzetes csomópont jelzi, hogy félreérthetetlenül megkülönböztessük őket egy Bayes-háló csomópontjaitól.) (b) A Viterbi-algoritmus működése az [*igaz, igaz, hamis, igaz, igaz*] megfigyelési sorozatra. Minden t időpontra feltüntettük az $\mathbf{m}_{1:t}$ üzenet értékeit, ami minden egyes t időpontbeli állapothoz megadja a legjobb, benne végződő sorozat valószínűségét. Minden egyes állapothoz egy vastag nyíl is vezet, ami a legjobb elődjét jelzi, a megelőző sorozat valószínűségének és az átmenet valószínűségének a szorzata szerint. A vastag nyilak visszafelé követése az $\mathbf{m}_{1:5}$-beni legvalószínűbb állapotból pedig megadja a legvalószínűbb sorozatot. 482
15.6. Egy állandó d lépésnyi időkülönbözettel simító algoritmus, folyamatos működésű (online) algoritmusként megvalósítva: egy új időpontbeli megfigyelésre kiadja az új simított becslést 484
15.7. Egy $\mathbf{X}_t$ hely, $\dot{\mathbf{X}}_t$ sebesség és $\mathbf{Z}_t$ helymegfigyelés alkotta lineáris dinamikus rendszerhez tartozó Bayes-hálóstruktúra 486
15.8. A Kalman-szűrés frissítési ciklusának a lépései egy véletlen bolyongás esetén. A véletlen bolyongás a priori eloszlása $\mu_0 = 0{,}0$ és $\sigma_0 = 1{,}0$, az átmenet bizonytalansága $\sigma_x = 2{,}0$, az érzékelő bizonytalansága $\sigma_z = 1{,}0$, az első megfigyelés pedig $z_1 = 2{,}5$ (az x tengelyen csillaggal jelölve).Vegyük észre, ahogy a $P(x_1)$ előrejelzés ellapul a $P(x_0)$-hoz képest az átmenet bizonytalansága miatt. Vegyük azt is észre, hogy az a posteriori $P(x_1|z_1)$ kissé balra helyezkedik el a z_1 megfigyeléstől, mivel az átlag az előrejelzés és a megfigyelés súlyozott átlaga. 488
15.9. (a) A Kalman-szűrés eredménye egy X–Y síkon mozgó objektumnál, feltüntetve a valódi (balról jobbra haladó) pályagörbét, a zajos megfigyelések egy sorozatát és a Kalman-szűrés alapján becsült pályagörbét. A helybecslések szórásait az oválisok jelzik. (b) A Kalman-simítás eredménye ugyanarra a megfigyelési sorozatra. 490
15.10. Egy fa felé repülő madár (felülnézetben). (a) Egy Kalman-szűrő előrejelzése a madár helyzetére, ami egyetlen Gauss-eloszlás az akadály közepére illesztve. (b) Egy valósághűbb modell számításba veszi a madár elkerülő manővereit, és azt jelzi előre, hogy az egyik vagy a másik oldalon fog elszállni. 491
15.11. (a) Az esernyős DBH-hoz tartozó a priori eloszlás, az állapotátmenet-modell és az érzékelő modell megadása. Az összes következő szelet feltevésünk szerint az 1. szelet másolata. (b) Egy X–Y síkon való robotmozgáshoz tartozó egyszerű DBH. 493
15.12. (a) Felső görbe: az Akkumulátort várható értékének a pályagörbéje egy olyan megfigyelési sorozatnál, ami csak 5-ösöket tartalmaz, kivéve a 0-kat a t = 21 és a t = 22 időpontokban, egy egyszerű Gauss-hibamodell felhasználásával. Alsó görbe: a pályagörbe, amikor a megfigyelések 0-val folytatódnak t = 21-től. (b) Ugyanez a kísérlet az átmenetihiba-modellel elvégezve. Vegyük észre, hogy az átmeneti hibát jól kezeli, de a végleges hiba túlzott pesszimizmust eredményez. 494

Created by XMLmind XSL-FO Converter.

15.13. (a) Az érzékelő állapotváltozóját bemutató DBH-részlet. Ez a változó szükséges az akkumulátormérő végleges hibájának a modellezésére. (b) Felső görbék: az Akkumulátort várható értékének a pályagörbéje az „átmeneti hibájú” és a „tartós hibájú” megfigyelési sorozatoknál. Alsó görbék: az AMHibás valószínűségének pályagörbéi a két megfigyelési sorozatnál. 495
15.14. Egy dinamikus Bayes-háló kibontása: a szeletek ismétlődnek befogadva a megfigyelési sorozatot (a halvány csomópontok). További szeleteknek már nincs hatása a megfigyelési perióduson belüli következtetésre. Ha a DBH-t kibontottuk, bármely következtetési algoritmus használható – változó eliminálás, egyesítési fa módszerek és így tovább –, ami a 14. fejezetben szerepel. 496
15.15. A részecskeszűrés algoritmus megvalósítása egy rekurzív frissítési művelettel és állapottal (ami a minták halmaza). A mintavételező lépések mindegyike az aktuális szelet változóinak a mintavételezését jelenti egy topológiai sorrendben, hasonlóan, mint a PRIOR-MINTA-ban. A SÚLYOZOTT-MINTA-VISSZATÉTELLEL művelet megvalósítható $O(N)$ várható futási idővel. 497
15.16. A részecskeszűrés frissítési ciklusa az esernyő DBH-nál $N = 10$ esetén, bemutatva minden egyes állapot mintapopulációját. (a) A t időpontban 8 minta jelez *Eső*-t és 2 ¬*Eső*-t. Mindegyiket továbbterjesztjük a következő állapotátmenet-modell szerinti mintavételezésével. A $t + 1$ időpontban 6 minta jelez *Eső*-t és 4 ¬*Eső*-t. (b) A $t + 1$ időpontban ¬*Esernyő* a megfigyelés értéke. Mindegyik mintát súlyozzuk a mintának a megfigyelésre vett feltételes valószínűségével, amit a körök nagysága jelez. (c) Egy új 10-es mintahalmaz generálódott az aktuális halmazból történő súlyozott sorsolással, 2 olyan mintát eredményezve, ami *Eső*-t jelez, és 8 olyat, ami ¬*Eső*-t. 498
15.17. A DARPA fonetikus ábécé, avagy **ARPAbet**, amely az amerikai angolban használt beszédhangokat sorolja fel. Számos alternatív felírás létezik, köztük az International Phonetic Alphabet (IPA), mely az összes ismert nyelv beszédhangját tartalmazza. 500
15.18. Az akusztikus jel átalakítása keretek sorozatává; mindegyik keretet három akusztikus jellemzővel írunk le 501
15.19. A „tomato” szó két kiejtési modellje. Mindegyik modellt egy átmenetdiagramként ábrázoljuk, amiben az állapotokat körrel jelöljük, a megengedett átmeneteket pedig nyilakkal, rajtuk a kapcsolódó valószínűségekkel. (a) Egy nyelvjárásbeli különbségeket is tekintetbe vevő modell. A 0,5-ös értékek a két szerző preferált kiejtésein alapuló becslések. (b) Egy olyan modell, ami az első magánhangzón egy koartikulációs hatást is figyelembe vesz, megengedve az [ow] vagy az [ah] beszédhangokat. 503
15. Az [m] háromállapotú beszédhang egy RMM-je. Mindegyik állapotnak számos lehetséges kimenetele lehet, különálló valószínűségekkel. A $C_1, \ldots, C_7$ VK-címkék önkényesen lettek megválasztva. ... 504
15.21. Az unigram és bigram számlálóknak egy részleges táblázata a jelen könyv (eredeti angol nyelvű kiadásának) szavai alapján. A „the” a leggyakoribb egyedülálló szó 33 508 előfordulással (513 893 összes szóból). A leggyakoribb bigram az „of the”, 3833 előfordulással. Néhány szám nagyobb, mint várható (például 4-szer fordul elő az „on is”), mivel a bigram számlálás figyelmen kívül hagyja az elválasztást: egy mondat végződhet „on”-ra és a következő kezdődhet „is”-zel. 505
15.22. Egy váltó Kalman-szűrő Bayes-háló reprezentációja. Az S_t váltó változó egy diszkrét állapotváltozó, aminek az értéke meghatározza az $\mathbf{X}_t$ folytonos állapotváltozók átmeneti modelljét. Minden i diszkrét állapotra a $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{X}_t, S_t = i)$ állapotátmenet-modell egy lineáris Gauss-modell, pontosan úgy, mint egy szabályos Kalman-szűrőben. A diszkrét állapotok közti $\mathbf{P}(S_{t+1}|S_t)$ állapotátmenet-modellt egy mátrixnak tekinthetjük, ahogyan egy rejtett Markov-modellben. 511
16.1. (a) A cserék egy teljes ciklusa, bemutatva, hogy a nemtranzitív $A \succ B \succ C \succ A$ preferenciák irracionális viselkedést eredményeznek. (b) A felbonthatósági axióma. 515
16.2. A pénz hasznossága. (a) Empirikus adat Beard úrtól egy korlátos tartományban. (b) Egy tipikus görbe a teljes tartományban. 518
16.3. Szigorú dominancia. (a) Determinisztikus: A-t szigorúan dominálja B, de sem C, sem D nem. (b) Bizonytalan: A-t szigorúan dominálja B, de C nem. 521
16.4. Sztochasztikus dominancia. (a) S_1 sztochasztikusan dominálja S_2-t a költségek vonatkozásában. (b) S_1 és S_2 negatív költségeinek eloszlásfüggvényei. 521
16.5. Egy egyszerű döntési hálózat a repülőtér-elhelyezési problémára 524
16.6. A repülőtér-elhelyezési probléma egy egyszerűsített reprezentációja. A kimeneti állapotokhoz tartozó véletlen csomópontok hatását kiszámítottuk, majd a csomópontokat elhagytuk. 526
16.7. Három általános eset az információ értékére. Az (a) esetben A_1 majdnem teljes bizonyossággal jobb marad A_2-höz képest, így az információ nem szükséges. A (b) esetben a választás nem egyértelmű, és az információ döntő fontosságú. A (c) esetben a választás bizonytalan, de igazából nem számít, így az információ is kevésbé értékes. 528
16.8. Egy egyszerű információgyűjtő ágens váza. Az ágens folyamatosan a legnagyobb információértékű megfigyelést választja, mindaddig, amíg a következő megfigyelés költsége kisebb, mint a várható haszna. 529
16.9. Az aortaszűkület hatásdiagramja (Peter Lucas hozzájárulásával) 531

Created by XMLmind XSL-FO Converter.

17.1. (a) Egy egyszerű 4 × 3-as környezet, ami az ágens számára szekvenciális döntési probléma. (b) A környezet állapotátmenet-modelljének illusztrálása: a „szándékolt” kimenetel 0,8 valószínűséggel következik be, de 0,2 valószínűséggel az ágens oldalra mozdul a szándékolt irányhoz képest. A fallal való ütközéskor nincsen mozgás. A két végállapot jutalma a jelzett +1 és –1, az összes többi állapot jutalma –0,04. 538
17.2. (a) Egy optimális eljárásmód a sztochasztikus környezetre $R(s) = –0,04$ esetén nem végállapotoknál. (b) Optimális eljárásmódok $R(s)$ négy különböző értéktartománya esetén. 540
17.2. Az állapotok hasznosságai a 4 × 3-as világban, $\gamma = 1$ és a nem végállapotoknál $R(s) = –0,04$ esetén 543
17.4. Az értékiteráció algoritmus az állapotok hasznosságának kiszámítására. A leállási feltétel a (17.8) egyenletből származik. 545
17.5. (a) A grafikonokon kiválasztott állapotok hasznosságainak a fejlődése látható az értékiteráció felhasználásánál. (b) A szükséges értékiterációk száma (k), hogy a hiba garantáltan legfeljebb $\varepsilon = cR_{max}$ legyen, c különböző értékeinél, mint a γ leszámítolási tényező függvénye. 545
17.6. A hasznosságbecslés maximális hibája $||U_i – U||$ és az eljárásmód vesztesége $\left\|U^{\pi_i} - U\right\|$ az optimális eljárásmódhoz viszonyítva az értékiteráció iterációszámának függvényében 547
17.7. Az eljárásmód-iterációs algoritmus egy optimális eljárásmód kiszámítására 549
17.8. (a) Az ágens elhelyezkedésének kezdeti valószínűségi eloszlása. (b) Ötszöri *Balra* mozgás után. (c) Ötszöri *Fel* mozgás után. (d) Ötszöri *Jobbra* mozgás után. 550
17.9. Egy dinamikus döntési háló általános struktúrája. Az ismert értékű változók árnyékoltak. A jelenlegi időpont t, és az ágensnek el kell döntenie, hogy mit tegyen – azaz választania kell egy értéket A_t számára. A háló három jövőbeli lépésre van kibontva, és a jövőbeli jutalmakat, valamint az előrelátó horizonton lévő állapot hasznosságát ábrázolja. 553
17.10. A 17.9. ábrán látható DDH előrenéző megoldásának egy része 554
17.11. (a) és (b): Minimax játékfák a kétujjas snóblijátékhoz, ha a játékosok a fordulókban tiszta stratégiát játszanak. (c) és (d): Parametrizált játékfák, ahol az első játékos kevert stratégiát játszik. A jutalmak függnek a kevert stratégia (p és q) valószínűségi paramétereitől. (e) és (f): A valószínűségi paraméter bármely konkrét értékénél a második játékos a két cselekvés közül a „jobbikat” fogja választani, így az első játékos kevert stratágiájának az értékét a vastag vonal jelöli. Az első játékos a kevert stratégia valószínűségi paraméterét a metszéspontnál fogja megválasztani. 560
18.1. (a) Példa (x, $f(x)$) értékpárok és egy velük konzisztens lineáris hipotézis. (b) Egy – ugyanazokkal az adatokkal konzisztens – hetedfokú polinom. (c) Egy másik adathalmaz egy pontosan illeszkedő hatodfokú polinommal, illetve egy közelítő lineáris illesztéssel. (d) Egy egyszerű szinuszos illesztés ugyanazokra az adatokra. 573
18.2. Egy döntési fa annak eldöntésére, hogy várjunk-e asztalra 575
18.3. Példák az étterem problématerületre 577
18.4. A példahalmaz attribútumteszteléssel történő szétosztása. (a) A *Konyha* alapján történő szétosztás nem visz közelebb ahhoz, hogy a pozitív és negatív példákat megkülönböztessük. (b) A *Vendégek* attribútum tesztje jól szeparálja a pozitív és negatív példákat. Miután a *Vendégek* tesztjét elvégeztük, az *Éhes* egy eléggé jó második tesztlehetőséget ad. 577
18.5. A döntési fa tanulásra szolgáló algoritmus 578
18.6. A 12 példás tanító minta halmazból származtatott döntési fa 579
18.7. Tanulási görbe az étterem problématérre véletlenszerűen generált 100 példa esetén. A görbe 20 kísérlet eredményét összegzi. 581
18.8. A együttes tanulás eredményeképpen nyert megnövekedett kifejezőerőt illusztráló példa. Három lineáris küszöb jellegű hipotézist vizsgálunk, mindegyik a nem vonalkázott félsíkon ad pozitív besorolást. Az együttesben csak arra adunk pozitív osztályozást, amelyre mindhárom pozitívat ad. Az eredményül kapott háromszögterület egy olyan hipotézisnek felel meg, amelyet az eredeti hipotézistér egyetlen eleme sem képes kifejezni. 585
18.9. A turbó algoritmus működése. Mindegyik árnyékolt téglalap egy példának felel meg, a téglalap magassága mutatja a példa súlyát. A pipák és keresztek mutatják, hogy a példát jól vagy rosszul osztályoztuk a pillanatnyi hipotézis alapján. A döntési fa mérete mutatja, hogy ennek a hipotézisnek mekkora a súlya a végső együttesben. 586
18.10. Az együttes tanulás turbó módszereinek `ADABOOST` nevű variánsa. Az algoritmus hipotézisek kombinációját hozza létre. A `SÚLYOZOTT-TÖBBSÉG` egy hipotézist generál. Ez az új hipotézis azt a kimeneti értéket adja vissza, amely a **h**-ban található hipotézisek közül, a **z**-ben számon tartott súlyozással a legtöbb szavazatot kapta. 586
18.11. (a) A döntési tönkök és a turbózott döntési tönkök teljesítményének az étterem példán történő összehasonlítása $M = 5$ esetén. (b) A tanító halmazon és a teszthalmazon kapott jó válaszok aránya M-nek

Created by XMLmind XSL-FO Converter.

– az együttesben található hipotézisek számának – a függvényében. Figyeljük meg, hogy a teszthalmazon mért pontosság még azután is javul egy kicsit, amikor a tanító halmazon elérte az 1-et, azaz miután az együttes pontosan illeszkedett az adatokra. 587
18.12. A hipotézistér sematikus ábrázolása a keresett *f* függvény körül felvett „ ε -gömb” ábrázolásával 589
18.13. Az éttermi problémára alkotott döntési lista 590
18.14. Egy döntési lista tanulásra használható algoritmus 591
18.15. A DÖNTÉSI-LISTA-TANULÁS algoritmusnak az éttermi adatokon felvett tanulási görbéje. Az összehasonlítás kedvéért feltüntettük a DÖNTÉSI-FA-TANULÁS algoritmus görbéjét is. 591
19.1. (a) Egy konzisztens hipotézis. (b) Egy hamis negatív példa. (c) A hipotézist általánosítottuk. (d) Egy hamis pozitív példa. (e) A hipotézist szűkítettük. 600
19.2. A pillanatnyilag-legjobb-hipotézis tanuló algoritmus. Konzisztens hipotézist keres és viszszalép, ha nem található konzisztens szűkítés/általánosítás. 600
19.3. Verziótér tanuló algoritmus. *V*-nek olyan részhalmazát tanulja meg, amelynek elemei a *példák*kal konzisztensek. 602
19.4. A verziótér a példákkal konzisztens összes hipotézist tartalmazza 602
19.5. A *G* és az *S* elemeinek kiterjesztései. A kettő között nem található ismert példa. 604
19.6. A kumulatív tanulási folyamat felhasználja és idővel folyamatosan bővíti a háttértudás készletét 606
19.7. Az egyszerűsítési probléma bizonyítási fái. Az első fa az eredeti problémapéldány bizonyítása, amiből az *AritmetikaiIsmeretlen*(*z*) ⇒ *Egyszerűsítés*(1 × (0 + *z*), *z*) vezethető le. A második fa az a bizonyítás, amikor az eredeti problémapéldányban szereplő összes konstanst változókkal helyettesítettük, amiből viszont további szabályok sokaságát vezethetjük le. 610
19.8. Algoritmus egy minimális konzisztens meghatározás megkeresésére 613
19.9. Az RADFT és a DÖNTÉSI-FA-TANULÁS hatékonyságának összehasonlítása véletlen módon generált adatok és egy olyan célfüggvény esetén, amely a lehetséges 16 attribútumból csak 5-öt használ . 614
19.10. Az (a) és (b) ábra a fehérjemolekula összehajtogatásának tárgyterületén a „négy-helikális fel-és-le köteg” fogalom pozitív és negatív példáját mutatja. Mindkét példa struktúráját egy olyan kb. 100 konjunktív tagot tartalmazó logikai kifejezésbe kódolták, mint amilyen például a *TeljesHossz*(*D2mhr*, 118) ∧ *HelikálisSzám*(*D2mhr*, 6) ∧ … kifejezés. 616
19.11. Egy tipikus családfa 616
19.12. A FOIL algoritmus váza, amely példákból elsőrendű Horn-klózokat tanul meg. Az ÚJ-LITERÁLOK és a LITERÁL-MEGVÁLASZTÁSA eljárásokat a szövegben magyarázzuk meg. 619
19.13. Az inverz rezolúciós eljárás kezdő lépései. Az árnyékolt klózokat az inverz rezolúciós eljárás generálja a jobb oldalon és az alatta lévő klózokból. A nem árnyékolt klózok a *Leírások*-ból és a *Besorolások*-ból származnak. 620
19.14. Egy új *P* predikátumot generáló inverz rezolúciós lépés 622
20.1. (a) A (20.1) egyenletből számított $P(h_i|d_1,\ldots,d_N)$ a posteriori valószínűségek. A megfigyelések száma 1-től 10-ig terjed, és mindegyik megfigyelés citromízű cukorka. (b) A (20.2) egyenlet alapján számított $P(d_{N+1} = citrom|d_1,\ldots,d_N)$ Bayes-predikció. 628
20.2. (a) Bayes-háló modell az ismeretlen arányban citrom-, illetve meggyízű cukorkák esetére. (b) Annak a modellje, amikor a csomagolópapír színe függ (valószínűségi alapon) a cukorka ízétől. 630
20.3. A 18. fejezet étterem problémájára alkalmazott naiv Bayes-tanulás tanulási görbéje. A döntési fa tanulás tanulási görbéjét összehasonlítás céljából ábrázoltuk. 633
20.4. (a) Egy $y = (\theta_1 + \theta_j + \theta_2)$ egyenlettel leírható lineáris Gauss-modell additív, rögzített varianciájú Gauss-zajjal. (b) E modell alapján generált 50 adatpontból álló halmaz. 634
20.5. Példák a béta[*a*,*b*] eloszlásra különböző [*a*,*b*] értékek esetén 635
20.6. Egy Bayes-tanulásnak megfelelő Bayes-háló. A Θ, Θ_1, Θ_2 változók a posteriori eloszlásai kikövetkeztethetők az a priori eloszlásokból és az *Íz*$_i$, *Csomagoló*$_i$ változókra vonatkozó tényekből. 636
20.7. (a) A rejtett változónak feltételezett szívbetegség egyszerű diagnosztikai hálója. Minden egyes változónak három lehetséges értéke van, a változókat a feltételes eloszlásukban szereplő független változók számával címkéztük, a végső számuk 78. (b) A *SzívBetegség* eltávolításával készült ekvivalens háló. Vegyük észre, hogy a tünetváltozók most már nem feltételesen függetlenek, ha a szülőket megadjuk. Ez a háló 708 paramétert igényel. 639
20.8. (a) 500 kétdimenziós adatpont, amely három osztály jelenlétét sugallja. (b) Egy három-komponensű kevert Gauss-modell, a súlyok (balról jobbra) 0,2, 0,3 és 0,5. Az (a) ábrán látható adatokat ebből a modellből generáltuk. (c) Az adatokból az EM algoritmussal rekonstruált modell 640
20.9. Az *L* log likelihood értéknek – az EM iterációszámának függvényében való – alakulását bemutató görbék. A vízszintes vonal a helyes modellnek megfelelő log likelihood értéket mutatja. (a) A 20.8. ábrán

Created by XMLmind XSL-FO Converter.

látható kevert Gauss-modellre vonatkozó görbe. (b) A 20.10. (a) ábrán látható Bayes-hálóra vonatkozó görbe. 641
20.10. (a) A cukorkák kevert modellje. A különböző ízű cukorkák, a csomagolások, valamint a lyukas cukrok aránya a zsáktól függ, amelyet nem figyelünk meg. (b) Egy kevert Gaussra vonatkozó Bayes-háló. Az **X** megfigyelhető változó átlaga és kovarianciája a *C* komponenstől függ. 642
20.11. Rejtett Markov-modellt reprezentáló kiterített dinamikus Bayes-háló (a 15.14. ábra megismétlése) 644
20.12. (a) A 20.8. (a) ábra adatainak 128 pontos részhalmaza, két kiválasztott ponttal, és 10 legközelebbi szomszédjukkal. (b) Az adatok előállításánál használt kevert Gauss-eloszlás 3D ábrája. 647
20.13. A 20.12. (a) ábrán látható adatokra végzett sűrűségbecslés, $k = 3$ (a); $k = 10$ (b) és $k = 40$ (c) esetén 647
20.14. A 20.12. (a) ábrán látható adatok sűrűségbecslése Gauss-kernelekkel, $w = 0{,}02$ (a); $w = 0{,}07$ (b) és $w = 0{,}20$ (c) esetén. 648
20.15. A neuron egyszerű matematikai modellje. Az egység kimeneti aktivációja $$a_i = g\left(\sum_{j=0}^{n} W_{j,i} a_j\right)$$ ahol a_j a j-edik egység kimeneti aktivációja és $W_{j,i}$ a j-től i-ig vezető összeköttetés súlya. 649
20.16. (a) A **küszöb** aktivációs függvény, amely 1-et ad a kimeneten, ha a bemenete pozitív, különben pedig 0-t. (Néha az előjelfüggvényt is használják ehelyett, amely ±1-et ad a bemenet előjelétől függően.) (b) A **szigmoid** függvény $1/(1+e^{-x})$. 650
20.17. Megfelelő bemeneti és eltolássúlyokkal rendelkező, küszöbaktivációjú egységek képesek logikai kapuként működni 651
20.18. Egy nagyon egyszerű, két bemeneti egységgel, egyetlen – két egységből álló – rejtett réteggel és egy kimeneti egységgel rendelkező háló 652
20.19. (a) Egy perceptronhálózat három kimeneti egységből, amelyeknek öt közös bemenete van. Ha kiválasztunk egy kimeneti egységet (mondjuk a másodikat, vastag vonallal kiemelve), akkor azt látjuk, hogy bemeneti összeköttetései nincsenek semmilyen hatással a többi kimeneti egységre. (b) Egy kétbemenetű szigmoid aktivációs függvényű perceptronegység kimenetének ábrázolása. 653
20.20. Lineáris szeparálhatóság küszöbperceptronok esetén. Fekete pötty jelzi a bemeneti tér olyan pontjait, amelyekre a függvény értéke 1, fehér pötty pedig az olyan pontokat, amelyre 0 ez az érték. A perceptronkimenet az egyenes nem árnyékolt oldalán 1. A (c) esetben nincs olyan egyenes, amely jól osztályozná a bemeneteket. 653
20.21. A gradiensalapú perceptron tanulási algoritmus, feltételezve, hogy a g aktivációs függvény differenciálható. Küszöbperceptronokra elhagyjuk a $g'(in)$ faktort a súlyfrissítésből. A NEURÁLIS-HÁLÓ-HIPOTÉZIS függvény egy olyan hipotézist ad vissza, amely bármely bemenetre kiszámítja a háló válaszát. 655
20.22. A perceptronok és döntési fák teljesítményének összehasonlítása. (a) A perceptronok jobbak a 11 bemenetű többségfüggvény tanulásában. (b) A döntési fák jobbak az étterem példában a *VárjunkE* predikátum tanulásában. 655
20.23. (a) Két – egymással szemben álló – lágy küszöbfüggvény kombinációjának eredménye: egy hegygerinc. (b) Két hegygerinc kombinációjának eredménye a dudor. 657
20.24. Többrétegű neurális háló egy rejtett réteggel és 10 bemeneti egységgel, alkalmas az étterem problémához 657
20.25. A többrétegű hálókra kidolgozott hiba-visszaterjesztéses tanulási algoritmus 658
20.26. (a) Az étterem probléma egy adott példahalmazán felvett tanulási görbe, ami a súlyok számos epoch során történő módosításával elért fokozatos hibacsökkenést mutatja. (b) Összehasonlító tanulási görbék, amelyek azt mutatják, hogy a döntési fa tanulás valamivel jobb teljesítményt ad, mint a többrétegű háló hiba-visszaterjesztéses tanulása. 659
20.27. (a) Egy kétdimenziós tanító halmaz, amelyben a pozitív példákat fekete, a negatívakat fehér körök jelölik. Az $x_1^2 + x_2^2 \leq 1$ valódi elválasztó határt is bejelöltük. (b) Ugyanazok az adatok az $x_1^2, x_2^2, \sqrt{2}x_1x_2$ háromdimenziós térbe való leképezés után. Az (a) ábrán látható kör alakú döntési határ a háromdimenziós térben lineáris döntési felületbe ment át. 661
20.28. Az első két dimenzióra vetített közelkép: a 20.27. (b) ábra optimális szeparátora. A szeparátort vastag vonallal jelöltük, a hozzá legközelebbi pontokat – a **szupport vektor**okat – bekarikáztuk. A **tartalék** nem más, mint a pozitív és negatív példák közti elválasztó sáv szélessége. 662
20.29. Néhány példa az NIST kézzel írt számjegyekre létrehozott adatbázisából. A felső sorban a 0–9 karakterek könnyebben felismerhető példáit mutatjuk be, az alsó sorban pedig ugyanezen számjegyekre nehezebben felismerhető példákat látunk. 664

Created by XMLmind XSL-FO Converter.

21.1. (a) A 4 × 3-as világban alkalmazott π stratégia. Ez optimális stratégia, ha nem alkalmazunk leértékelést, és minden állapotban, amely nem végállapot, $R(s) = -0{,}04$. (b) A 4 × 3-as világ állapotainak hasznossága a π stratégia esetén. 674
21.2. Egy adaptív lineáris programozás alapon tanuló passzív megerősítéses ágens. A kód egyszerűsítése érdekében feltettük, hogy minden egyes észlelés szétbontható egy észlelt állapotra és egy jutalomjelre. 676
21.3. A passzív ADP-tanuló tanulási görbéi a 4 × 3-as világ tanulása során, a 21.1. ábrán látható optimális stratégia használata esetén. (a) Az állapotok egy kiválasztott részhalmazára kiszámított hasznosságbecslések a kísérletek számának függvényében. Figyeljük meg a 78-adik kísérlet környékén a jelentős változásokat – ez az első eset, amikor az ágens a (4,2) végállapotba jut, melyben –1 jutalmat kap. (b) Az $U(1,1)$ hibájának RMS-értéke 20 – egyenként 100 kísérletből álló – futásra átlagolva. 677
21.4. Egy időbeli különbség alapon hasznosságot tanuló passzív megerősítéses ágens 678
21.5. Az IK-tanulás 4 × 3-as világon mért tanulási görbéi. (a) Néhány kiválasztott állapot hasznosságbecslése a kísérletszám függvényében. (b) Az $U(1,1)$ becslésének RMS-hibája, amelyet 20 futás átlagából számoltunk, mindegyik futás 500 kísérletet tartalmazott. Csak az első 100 kísérletből származó eredményt rajzoltuk fel, hogy a görbe a 21.3. ábrával összehasonlítható legyen. 678
21.6. Annak a mohó ADP-ágensnek a felmutatott teljesítménye, amely végrehajtja a tanult modell alapján optimálisnak tűnő cselekvést. (a) A kilenc nem végállapot alapján számolt átlagos RMS-hiba. (b) Az a szuboptimális stratégia, amely felé ebben a kísérletsorozatban a mohó ágens konvergál. 680
21.7. A felfedező ADP-ágens teljesítménye, $R^+ = 2$ és $N_e = 5$ esetén. (a) Néhány kiválasztott állapot hasznosságának becslése az idő függvényében. (b) A hasznosságértékek RMS-hibája, és az ehhez kapcsolható, a stratégiából származó veszteség. 683
21.8. Egy felfedező *Q*-tanuló ágens. Egy aktív tanuló, amely megtanulja minden cselekvésnek minden egyes állapotban a $Q(a, s)$ értékét. Ugyanazt az *f* felfedezési függvényt használja, amelyet a felfedező ADP-ágens is használt, de elkerüli az állapotátmenet-modell tanulását, mivel az állapot *Q*-értékét közvetlenül a szomszédos állapotok *Q*-értékeihez tudja kapcsolni. 684
21.9. A mozgó kocsi tetején álló hosszú rúd egyensúlyozásának vizsgálatára szolgáló berendezés. A kocsit az $x, \theta, \dot{x}, \dot{\theta}$ állapotváltozókat megfigyelő szabályzó jobbra vagy balra lökheti. 688
21.10. Időben eltolt képek egymásra másolása útján kapott eredő kép, amelyen egy nagyon bonyolult „körberepülés orral a kör középpontja fele” manővert hajtanak végre. A helikoptert egy Pegasus stratégiakereső algoritmussal fejlesztett stratégiával vezérlik. Egy szimulátormodellt fejlesztettek a valós helikopter egyes vezérlési beavatkozásokra adott válaszainak vizsgálatára. Ezek után egész éjszaka futtatták az algoritmust a szimulátoron. Egy sor vezérlőt fejlesztettek ki különböző manőverekre. Minden esetben, amikor távirányítást használtak, messze jobb volt az eredmény, mint a képzett humán pilótáké. (A képet Andrew Ng engedélyével közöljük.) 690
22.1. A kommunikációban részt vevő hét folyamat, a „The wumpus is dead” mondat felhasználásával 700
22.2. Egy kommunikáló ágens, amely elfogad parancsokat, kérdéseket és állításokat. Az ágens leírhatja az aktuális állapotot, valamint végrehajthat „hagyományos”, nem szólásaktus cselekvést is, ha nincs mit mondania. 701
22.3. $\mathcal{E}_0$ szókincse 702
22.4. Az $\mathcal{E}_0$ nyelvtana, minden egyes szabályra példa kifejezést adva 703
22.5. A „the wumpus is dead” lentről felfelé történő elemzési lépései. A szavakat tartalmazó csomópontok listájával kezdünk. Ezek után egy szabály jobb oldalára illeszkedő részsorozatokat felcserélünk egy új csomóponttal, amelynek gyökere a szabály bal oldala. Például a harmadik sorban az *Article* és *Noun* csomópontokat lecseréljük egy *NP* csomópontra, amelynek e két csomópont lesz a gyereke. A fentről lefelé elemzés hasonló lépéseket eredményezne, de épp a fordított irányban. 705
22.6. A „The agent feels a breeze” mondat diagramjának egy részlete. Mind a hat csomópont látható, de csak három él szerepel azok közül, melyek teljes elemzést eredményeznének. 706
22.7. A diagramelemző algoritmus. *S* a kezdő szimbólum és *S′* egy új nem záró szimbólum, *diagram*[*j*] pedig azon élek listája, melyek a *j* csomópontban végződnek. A görög ábécé betűi nulla vagy több szimbólumból álló füzérekre illeszkednek. 707
22.8. Az „$_0$ I $_1$ feel $_2$ it $_3$” mondat diagramelemzése. Az *m*:*S* jelölés azt jelenti, hogy az *m* élnek egy *S* áll a jobb oldalán, míg az *f*:*VP*/*Verb* azt, hogy az *f* élnek *VP* van a bal oldalán, de egy *Verb*-et vár. Öt teljes él van a csomópontok felett, és nyolc hiányos alattuk. 708
22.9. Az „$_0$ I $_1$ feel $_2$ it $_3$.” elemzési lépései. Az a–m élek mindegyikére megmutatjuk azt az eljárást, amit az él más, a diagramon már szereplő élekből történő származtatására használtunk. Bizonyos éleket a rövidség érdekében elhagytunk. 709
22.10. Egy balsarok-elemző algoritmussal elkerülhető a „ride” szóval kezdődő *VP* jóslása, de jósolni fog egy „was” szóval kezdődő *VP*-t, mivel a nyelvtan egy *NP*-t követő *VP*-t vár. A „the horse gave” feletti

Created by XMLmind XSL-FO Converter.

háromszög azt jelenti, hogy a szavaknak van egy *RelCause* elemzésük, amihez azonban közvetlenül további részek kapcsolódnak, amik nem láthatók. 710
22.11. Felül: Az $\mathcal{E}_1$ egy BNF nyelvtana, amely kezeli az alanyi és a tárgyas esetet a főnévi kifejezésekben, és ezért annyira nem is generál túl. Az $\mathcal{E}_1$ nyelvtanával megegyező részeket elhagytuk. Lent: Az $\mathcal{E}_1$ egy definit klóz nyelvtana (DCG). 712
22.12. Példák igékre az alkategória-listáikkal együtt 714
22.13. A „You give me the gold (Ideadod nekem az aranyat)” elemzési fája, amely bemutatja az ige és az igei kifejezés alkategóriáit 715
22.14. Szemantikával kibővített nyelvtan az aritmetikai kifejezésekre. Minden x_i változó egy összetevő szemantikáját reprezentálja. Vegye észre, hogy a {*teszt*} jelölést használjuk olyan logikai predikátumok definiálására, amelyeket ki kell elégíteni, de nem összetevők! 716
22.15. A „3 + (4 ÷ 2)” füzér elemzési fája szemantikai értelmezésekkel 717
22.16. Egy nyelvtan, amely képes egy elemzési fa és szemantikai értelmezés levezetésére a „John loves Mary” (és három másik) mondat számára 717
22.17. A „John loves Mary” füzér elemzési fája szemantikai értelmezésekkel 718
22.18. Minden szintaktikai kategória kvázilogikai formájú kifejezésének típusát mutató tábla. A $t \rightarrow r$ jelölés egy függvényt takar, amely egy t típusú argumentumot fogad, és r típusú eredménnyel tér vissza. Például a *Preposition* szemantikai típusa $object^2 \rightarrow sentence$, ami azt jelenti, hogy az elöljárószó szemantikája egy függvény, amelyet ha két logikai objektumra alkalmazunk, akkor egy logikai mondatot eredményez. 720
22.19. Az „Every agent smells a wumpus” mondat elemzési fája, amely mind a szintaktikai struktúrát, mind a szemantikai értelmezéseket mutatja 720
22.20. Szemantikával ellátott nyelvtan kvázilogikai formában 721
22.21. Koherenciarelációk listája (Hobbs, 1990) alapján. Mindegyik reláció két, egymást követő szövegrészlet, S_1 és S_2 között áll fenn. 727
22.22. Az „*abcdbcabcd*” bemeneti szöveget elemző nyelvtant tartalmazó Sequitur-rendszer futási lépései. Egy *S*-re vonatkozó szabályból indulunk ki, és ezen szabály végéhez sorban minden egyes szimbólumot hozzáadunk. Miután a hatodik szimbólumot is hozzáadtuk, megkapjuk egy ismétlődő pár első előfordulását: *bc*. Ezért a *bc* mindkét előfordulását lecseréljük egy új nem záróra, *A*-ra, és egy új szabályt veszünk fel: $A \rightarrow bc$. Miután három újabb szimbólumot hozzáraktunk a bemenethez, a kilencedik a *bc* egy újabb ismétlését eredményezi, így újra *A*-val helyettesítjük. Ez az *aA* két előfordulásához vezet, ezért egy új nem záróval, *B*-vel helyettesítjük. Miután a tizedik, utolsó szimbólumot is hozzáraktuk, *Bd* két előfordulását kapjuk, ezért lecseréljük őket egy új nem záróval, *C*-vel. De *B* immár csak egyszer szerepel, a *C* szabály jobb oldalán, ezért *B*-t a neki megfelelő bővítéssel, *aA*-val cseréljük le. 729
23.1. Viterbi-alapú szószegmentáló algoritmus. Egy szóközöket nem tartalmazó szófüzért feldolgozva megadja a legvalószínűbb szavakra történő szegmentációt. 739
23.2. Az $\mathcal{E}_0$ nyelvtan egy részletének valószínűségi környezetfüggetlen nyelvtana (PCFG) és szókincse. A szögletes zárójelben levő szám jelzi a valószínűségét annak, hogy az adott bal oldali szimbólumot a megfelelő szabály szerint írjuk át 740
23.3. A „Minden wumpus bűzlik” mondat elemzési fája, megadva minden egyes részfa valószínűségét. A teljes fa valószínűsége $1{,}0 \times 0{,}5 \times 0{,}05 \times 0{,}15 \times 0{,}60 \times 0{,}10 = 0{,}000225$. Mivel a mondatnak ez az egyetlen elemzése, ezért ennyi a mondat valószínűsége is. 740
23.4. A [Bayes information retrieval model] lekérdezés-valószínűségi IR-modellje a könyv első öt fejezetét tartalmazó dokumentumgyűjtemény felett. Megadjuk a szógyakoriságot mindegyik dokumentum-szó párra, és a szavak számát (N) az összes dokumentumra. Két dokumentummodellt alkalmazunk, D_i az i-edik dokumentumon alapuló simítatlan unigram szómodell, míg D'_i ugyanaz a modell adj-hozzá-egyet simítással, majd kiszámítjuk a lekérdezés valószínűségét minden dokumentumra mindkét modellel. A jelen (23.) fejezet az egyértelmű győztes, mindkét modell esetén több mint kétszázszor valószínűbb, mint bármely más dokumentum. 744
23.5. Egy gépi fordító rendszer választási lehetőségeit ábrázoló sematikus diagram. A fenn található angol szöveggel kezdjük. A köztes nyelven alapuló rendszer a folytonos vonalakat követi, az angol szöveget először szintaktikai elemzésnek veti alá, majd szemantikai és köztes nyelvi reprezentációt állít elő, végül szemantikai, szintaktikai és lexikai formákon keresztül francia szöveget állít elő. Az átvitelalapú rendszerek a szaggatott vonalak által jelölt rövidzárakat használják. Az egyes rendszerek eltérő szinteken végzik az átvitelt, egyes rendszerek több szinten is. 753
24.1. A képalkotás geometriája a sötétkamrában 763
24.2. Az emberi szem vízszintes keresztmetszete 765
24.3. (a) Egy asztalon lévő tűzőgép fényképe. (b) Az (a) egy 12 × 12 képpontból álló, nagyított részlete. (c) A részletnek megfelelő képfényességértékek egy 0-tól 255-ig tartó skálán. 765

Created by XMLmind XSL-FO Converter.

24.4. Különféle élek: (1) mélységi diszkontinuitás, (2) felületi orientáció diszkontinuitás, (3) viszszaverődési képesség diszkontinuitás, (4) fényességi diszkontinuitás (árnyékok) 768
24.5. (a) Egy asztalon nyugvó tűzőgép fényképe. (b) Az (a) alapján meghatározott élek. 768
24.6. Felül: Az *I(x)* intenzitásprofil egy egydimenziós részlet mentén egy lépcsőfok élén át. Középen: Az intenzitás deriváltja, *I'(x)*. A függvény nagy értékei éleknek felelnek meg, de a függvény zajos. Lent: Az intenzitás simított változatának deriváltja, $(I * G_\sigma)'$, amely $I * G'$-hoz hasonlóan egy lépésben számolható. Az $x = 75$ pontban levő zajos él jelölt eltűnt. 768
24.7. (a) Rubik-kocka egy forgó lemezkorongon. (b) Ugyanaz a kocka 19/30 másodperccel később (Richard Szeliski nyomán). (c) Az (a) és (b) képek összehasonlítása alapján számolt optikai folyam vektorok (Joe Weber és Jitendra Malik hozzájárulásával). 772
24.8. (a) Egy olyan videokép-sorozat négy kockája, amelyben a kamera a tárgyhoz képest mozgott és forgott. (b) A sorozat első képkockája, amelyen apró négyzetek jelzik a tulajdonságfelismerő által megtalált tulajdonságokat (Carlo Tomasi hozzájárulásával). 773
24.9. (a) A 24.8. ábrán látható kép tulajdonságai elhelyezkedésének háromdimenziós rekonstrukciója. (b) A valódi ház ugyanabból a perspektívából. 773
24. A térbeli látás alapgondolata: különböző kamerapozíciók ugyanannak a háromdimenziós jelenetnek kissé eltérő kétdimenziós nézeteit eredményezik 774
24.11. A diszparitás és a mélység összefüggése a sztereolátásban 774
24.12. (a) A textúragradienseket illusztráló jelenet. Feltéve, hogy a valós mintázat egyforma, lehetővé teszi a felület irányágának a meghatározását. A számított irányt a képen oly módon elhelyezett fehér kör és nyíl jelzi, mintha a kört a felületre festették volna abban a pontban. (b) Egy ívelt felület alakjának meghatározása a mintázat alapján. (A képeket Jitendra Malik és Ruth Rosenholtz hozzájárulásával közöljük [Malik és Rosenholtz, 1994].) 775
24.13. Egy felidéző vonalas rajz (Isha Malik hozzájárulásával) 777
24.14. Különféle vonalcímkék 778
24.15. A triéderes csúcsok négy fajtája 778
24.16. A 24.15. ábrán 1-gyel jelölt csúcs különböző előfordulásai 779
24.17. A Huffman–Clowes-címkekészlet 779
24.18. Két háromdimenziós tárgy többféle nézete 782
24.19. Egy arcmegtaláló algoritmus eredménye (Henry Schneiderman és Takeo Kanade hozzájárulásával) 783
24.20. Alakzatkontextus-számítás és -illesztés. (a, b) Két alakzat élének mintapontjai. (c) Az alakzatkontextus számítására használt log-polár hisztogram rekeszek ábrája. 5 rekeszt használunk a log r és 12 rekeszt θ számára. (d–f) Az (a) és (b) ábrán jelölt referenciaminták alakzatkontextusai: ○, ◊, ◅. Minden alakzatkontextus a maradék ponthalmaz koordinátáinak log-polár hisztogramja a referenciapontot mint origót használva. (A sötét cellák több pontot jelentenek a rekeszekben.) Vegye észre a ○ és ◊ pontok alakzatkontextusainak vizuális hasonlóságát, amelyeket a két alakzat relatíve hasonló pontjaira számoltunk! Ezzel ellentétben a ◅ alakzatkontextusa egészen eltérő. (g) Az (a) és (b) között kétoldali illesztéssel megtalált megfeleltetések, a hisztogramok χ^2 távolsága által definiált költséget használva. 784
24.21. Az illesztésalgoritmus informális leírása 786
24.22. (a) A tűzőgép fényképén talált sarkok. (b) Az eredeti képre ráhelyezett hipotetikus rekonstrukció (Clark Olson hozzájárulásával). 787
24.23. Az út képe egy autóban lévő kamerával lefényképezve. A vízszintes fehér csíkok jelzik azokat a keresési ablakokat, amiken belül a vezérlő az útvonali jeleket keresi. A gyenge képminőség nem szokatlan az alacsony felbontású fekete-fehér videók esetében. 787
24.24. A megcímkézendő objektum, amelynek minden sarokpontja triéderes 792
24.25. Felülnézeti kép egy kétkamerás látórendszerről, amely egy üveget figyel meg, amely mögött egy fal van 793
25.1. (a) A NASA Sojourner mobil robotja, amely a Mars felszínét derítette fel 1997 júliusában. (b) A Honda P3 és Asimo elnevezésű humanoid robotjai. 795
25.2. (a) A mobil robotoknál népszerű SICK LMS lézeres távolságmérő. (b) Egy horizontálisan felszerelt távolságmérő által mért visszaverődési kép 2D-s térképre vetítve. 796
25.3. (a) A Stanford manipulátor: egy korai robotkar öt rotációs (R) és egy transzlációs (prizmatikus) csuklóval (P), összesen hat szabadságfokkal. (b) Egy elsőkerék-kormányzású nemholonomikus négykerekű jármű mozgása. 798
25.4. (a) Marc Raibert egyik lépegető robotja mozgás közben. (b) A Sony AIBO robot focizás közben (copyright 2001, The RoboCup Federation). 799
25.5. A robotérzékelés felfogható úgy, mint cselekvések és mérések sorozatának időbeli egymásra hatása, ahogy ezt a dinamikus Bayes-háló is illusztrálja 800

Created by XMLmind XSL-FO Converter.

25.6. (a) Egy mobil robot egyszerűsített kinematikai modellje. A robotot a kör jelképezi, és a bevágás mutatja a haladási irányt. Külön-külön láthatjuk a t-beli és a $t + 1$-beli pozíciót és orientációt a $v_t \Delta_t$ és $\omega_t \Delta_t$ frissítési értékekkel. Szintén fel van tüntetve egy t időpontban megfigyelt referenciapont (tereptárgy) az (x_i, y_i) pontban. (b) A pásztázó távolságmérés modellje. Egy adott távolságú méréshez (z_1, z_2, z_3, z_4) tartozó két lehetséges robothelyzet látható. Sokkal valószínűbb, hogy a bal oldali helyzetből származnak a távolságmérések. 801
25.8. A Monte Carlo lokalizáció, egy részecskeszűrő algoritmus mobil robotok pozíciójának meghatározására. Felül: a kezdeti, teljes bizonytalanság. Középen: nagyjából kétpólusú bizonytalanság, miután a robot elindult a középső (szimmetrikus) folyosón. Alul: egyetlen pont köré szűkült bizonytalanság, miután bement az egyik irodába. 802
25.7. A Monte Carlo lokalizáló algoritmus, független zajjal terhelt pásztázó távolságméréses modellt használva 804
25.9. Egydimenziós, linearizált mozgási modell illusztrálása: (a) Az f függvény és $t + 1$ időpontbeli $\boldsymbol{\mu}_t$ középértéke és kovarianciája ($\boldsymbol{\Sigma}_t$ alapján). (b) A linearizált változat az f érintője a $\boldsymbol{\mu}_t$ pontban. A $\boldsymbol{\mu}_t$ középérték ugyanaz marad, ugyanakkor az előrevetített $\tilde{\boldsymbol{\Sigma}}_{t+1}$ kovariancia már különbözik a $\boldsymbol{\Sigma}_{t+1}$-től. 804
25.10. Példa a kiterjesztett Kalman-szűrő használatára lokalizációs problémához. A robot egy egyenes mentén mozog. Ahogy halad előre, a saját helyzetének ismeretéhez tartozó bizonytalanság fokozatosan nő, ahogy azt a hibaellipszisek mutatják. Amikor észrevesz egy ismert pozíciójú referenciapontot, a bizonytalanság csökken. 805
25.11. A KKSZ alkalmazása a robottérképezés problémájára. A robot pályája a szaggatott vonal, és a besatírozott ellipszisek jelentik helyzetére vonatkozó saját becsléseit. Nyolc pont jelenti az ismeretlen helyzetű referenciapontokat, míg a helyzetükre vonatkozó becsléseiket a fehér ellipszisek mutatják. Az (a), (b) és (c) ábrákon a robot helyzetére vonatkozó becslés bizonytalansága folyamatosan növekszik, ahogy azoké a referenciapontoké is, amelyek mellett elmegy. A (d) ábrán a robot újra elérkezik az első referenciaponthoz, így az összes többi pont helyzetének bizonytalansága is csökken, köszönhetően annak, hogy a becslések korreláltak. 806
25.12. (a) Egy két szabadságfokú robotkar munkatere. A munkatér egy doboz, amiben egy lapos akadály lóg a mennyezetről. (b) Ugyanennek a robotnak a konfigurációs tere. A fehérrel jelzett terület jelenti azokat a konfigurációkat, ahol a robot nem ütközik össze semmivel. A pont az ábrán a robot bal oldali konfigurációját mutatja. 809
25.13. Három különböző robotkonfiguráció a munkatérben és a konfigurációs térben 810
25.14. (a) A költségfüggvény és az előállított trajektória a konfigurációs térben, diszkrét hálós celladekompozíciós közelítés esetén. (b) Ugyanaz az útvonal a munkatérben ábrázolva. Érdemes megfigyelni, hogy a robot behajlítja a könyökét, hogy elkerülje a függőleges akadályt. 811
25.15. (a) A taszító potenciáltér távol tartja a robotot az akadályoktól. (b) A megoldásként kapott pálya, ha egyszerre minimalizáljuk az úthosszt és a potenciálfüggvényt. 812
25.16. (a) A Voronoi-diagram azon pontok halmaza a konfigurációs térben, amelyek egyenlő távolságra vannak két vagy több akadálytól. (b) A szabad tér 400 véletlenszerűen kiválasztott pontjából felépített valószínűségi úthálózat. 813
25.17. Egy kétdimenziós környezet, a sebesség vektor bizonytalanságát ábrázoló kúppal. A vezérlés bizonytalansága miatt az eredeti szándéknak megfelelő v sebesség valójában bármilyen vektor lehet a C_v kúpon belül. Ennek eredményeképpen a robot végső konfigurációja bárhol lehet a burkolón (a világosabban árnyalt területen), azaz nem tudjuk, hogy eljutottunk-e a lyukba, vagy sem. 816
25.18. Az első mozgásparancs és a lehetséges robotmozgások burkolója. A hibától függetlenül tudjuk, hogy a végső konfiguráció a lyuk bal oldalán lesz. 816
25.19. A második mozgásparancs és a lehetséges mozgások. Még hibák esetén is végül bele fogunk találni a lyukba. 817
25.20. Robotkarvezérlés (a) egységnyi erősítésű arányos szabályozóval; (b) 0,1-es erősítési tényezőjű arányos szabályozóval; (c) PD szabályozóval, 0,3-as erősítésű arányos taggal és 0,8-as differenciáló komponenssel. Mindhárom esetben a robotkar a szürkével jelölt pályát próbálta követni. 818
25.21. Potenciáltér-vezérlés. A robot egy potenciáltérben halad, amely az akadályoktól taszító erőkből és a célkonfigurációhoz vonzókból áll. (a) Sikeres útvonal. (b) Lokális optimum. 820
25.22. (a) Hexapod robot. (b) Kiterjesztett véges automata (KVA) (augmented finite state machine, AFSM) egyetlen láb vezérléséhez. Fontos, hogy ez a KVA-szenzoros visszacsatolásra reagál: ha a láb megakad az előremozgatás során, akkor magasabbra emeli. 821
25.23. (a) A Helpmate robot élelmiszert és egyéb orvosi felszerelést szállít több tucat kórházban, világszerte. (b) Sebészrobotok a műtőben (da Vinci Surgical Systems). 825
25.24. (a) Egy elhagyatott bánya feltérképezése egy robottal. (b) A bányának a robot által kapott 3D képe. 825
25.25. Diagram a 25.5. feladathoz 830

Created by XMLmind XSL-FO Converter.

25.26. Egyszerűsített robot egy labirintusban. Lásd 25.6. feladat. 832
27.1. Egy modell- és hasznosságalapú ágens, ahogyan először a 2.14. ábrán bemutattuk 851
27.2. A kompilálás célja a megfontolt (deliberatív) döntéshozatal hatékonyabb, reflexszerű mechanizmussá való átalakítása 853

Created by XMLmind XSL-FO Converter.

Előszó

A **mesterséges intelligencia** (**MI**) nagyon nagy terület, és ez a könyv is igen terjedelmes. A könyvben megkíséreltük e területet teljes szélességében feltárni: átölelve a logikát, a valószínűség-számítást, a folytonos matematikát, az érzékelést, a következtetést, a tanulást, a cselekvést, valamint mindent, a mikroelektronikától a világűrt kutató robotokig. A könyv azért is terjedelmes, mert bizonyos mértékig el is mélyedünk az eredmények tárgyalásában, miközben az egyes fejezetek központi részében csak a leglényegesebb gondolatokat igyekszünk bemutatni. További eredményekre az egyes fejezetek végén található irodalmi feljegyzésekben mutatunk rá.

A könyv alcíme „modern megközelítésben”. E talán önmagában üres szófordulat szándékolt jelentése az, hogy az ismert eredményeket egy közös keretbe szintetizálva igyekeztük bemutatni, ahelyett hogy az MI egyes részterületeit külön-külön a saját történelmi hátterükben tárgyaljuk. Elnézést is kérünk mindazoktól, akik a saját szakterületüket így a szokásosnál kevésbé ismerik fel.

A könyv fő egységesítő motívuma az **intelligens ágens** koncepciója. Felfogásunkban az MI egyszerűen a környezetüket észlelő és cselekvő ágensek tanulmányozása. Minden ilyen ágens egy érzékeléseket cselekvésekre leképező függvényt valósít meg. A könyvben e függvénykapcsolat többféle reprezentációjával foglalkozunk, olyan reprezentációkkal, mint például a produkciós rendszerek, a reaktív ágensek, a valós idejű feltételes tervkészítő rendszerek, a neurális hálók és a döntéselméleti rendszerek. A továbbiakban bemutatjuk a tanulás szerepét, mint a rendszertervezőnek azt a módszerét, amivel ismeretlen környezeteket is képes figyelembe venni, és megmutatjuk, hogy a tanulás milyen korlátozásokat jelent az ágens tervezésében, kiemelve az explicit tudásreprezentáció és a következtetés szerepét. A robotikát és a gépi látást nem függetlenül definiált problémaként tárgyaljuk, hanem mint a cél eléréséhez szükséges szolgáltatásokat. Külön hangsúlyozzuk a feladatkörnyezet fontosságát az ágens megfelelő megtervezése szempontjából.

A fő célunk azokat a *gondolatokat* átadni, amelyek az MI-kutatás elmúlt 50 évében és a kapcsolódó területeken az elmúlt kétezer évben születtek. E gondolatok bemutatásánál, miközben a precizitás megtartására törekedtünk, igyekeztünk kerülni a túlzott formalizmust. Ahol alkalmasnak találtuk, pszeudokód szintű algoritmusokat is közöltünk, hogy a gondolatok konkrét alakot öltsenek. Az általunk használt pszeudokódot röviden a B) függelékben írjuk le. A különböző programozási nyelveken implementált algoritmusokat az olvasó a **http://aima.cs.berkeley.edu** weboldalon találja meg.

A könyvet alapvetően egy alapképzésbeli tárgyhoz vagy tárgysorozathoz szántuk. Ugyanakkor használható a graduális képzés tárgyaiban is (esetleg az irodalmi megjegyzésekben javasolt források némelyikének hozzáadásával). A széles témalefedés és a nagyszámú részletezett algoritmus miatt a könyv alapreferenciaként szolgálhat az MI-szakos végzős hallgatók és a szűkebb kutatási területükből kitörni kívánó szakemberek számára. Egyetlen feltétel a számítógépes tudományok alapvető fogalomszintű ismerete (algoritmusok, adatstruktúrák, komplexitás) egy első-másodéves hallgató szintjén. Az analízis alsóbb éves szintű ismerete jól jön a neurális hálók és a statisztikai tanulás részleteinek megértéséhez. A szükséges matematikai háttér egyes részeit az A) függelékben adjuk meg.

A könyv áttekintése

A könyv nyolc részből áll. Az I. rész, a **Mesterséges intelligencia** az MI-kutatások olyan megközelítését adja, amely az intelligens ágens elképzelés köré szerveződik. Egy olyan rendszer köré, amely képes eldönteni, hogy mit cselekedjen és amely cselekedni is képes. A II. rész, a **Problémamegoldás** olyan döntési módszerekkel foglalkozik, amikor néhány lépésre előre kell gondolkodni. Ilyen pl. a sakk vagy a nyílt terepen történő navigálás. A III. rész, a **Tudás és következtetés** a világról alkotott tudás reprezentálásával foglalkozik, valamint azzal, hogy hogyan működik egy ilyen reprezentáció, milyen reprezentációkat ismerünk, hogyan befolyásolhatjuk ezeket a cselekvésünkkel, és hogy ezzel a tudással hogyan lehet logikusan következtetni. Ennek folytatásaként a IV. rész, a **Tervkészítés** e következtetési módszerek használatát mutatja a cselekvés eldöntésében, pontosabban a *tervek* készítésében. Az V. rész, a **Bizonytalan tudás és következtetés** hasonlít a III. és IV. részekre, de a *bizonytalanságok* melletti következtetésre és döntéshozatalra koncentrál. Ilyen problémákra pl. az orvosi diagnosztizáló és kezelő rendszerekben kell számítani.

A II.-tól az V. részig az anyag az intelligens ágenseknek azzal a komponensével foglalkozik, amely a döntéshozatalért felelős. A VI. rész, a **Tanulás** a döntéshozó komponens által igényelt tudás megszerzési módszereit írja le. A VII. rész, a **Kommunikáció, észlelés és cselekvés** olyan módszereket tár fel, amelyekkel az intelligens ágens a környezetét érzékelheti – látás, tapintás, hallás vagy akár nyelvmegértés révén – abból a célból, hogy megtudja, mi történik körülötte, továbbá olyan módszerekkel foglalkozik, melyek képesek az ágens

Created by XMLmind XSL-FO Converter.

terveit igazi cselekvéssé alakítani, ahol a cselekvés lehet egy robotmozgás, de akár nyelvi közlés is. Végül a VIII. rész, a **Konklúziók** az MI múltját és jövőjét, valamint a filozófiai és etikai következményeit elemzi.

Változások az első kiadáshoz képest

Az első kiadás óta az MI sok változáson ment át, és ez a könyv is sokat változott. Lényegében át is írtunk minden fejezetet abból a célból, hogy a könyv tükrözze a kutatás legújabb eredményeit, a korábbi munkáknak az új eredményekhez jobban illeszkedő értelmezését adhassa, valamint hogy a gondolatok folyamának oktató jellege még jobban érvényesülni tudjon. Az MI követőit bátorítani szeretnénk, hogy a mai módszerek sokkal praktikusabbak, mint az 1995-ösek. A tervkészítést példának véve az első kiadás algoritmusai néhány tíz lépésből álló tervek generálására voltak képesek, e kiadás algoritmusai több tízezer lépésnyi tervekre skálázhatók fel. Hasonló nagyságrendi javulás tapasztalható a valószínűségi következtetés, a nyelvfeldolgozás és más területek esetében is. A könyvben az alábbiak a legfontosabb változások:

- Az I. részben méltatjuk a szabályozáselmélet, a játékelmélet, a gazdaságtan és a neurális tudományok történelmi szerepét. Ez segíti, hogy e gondolatokat a következő fejezetekben integráltabb módon tudjuk tárgyalni.

- A II. részben az online keresési algoritmusokkal foglalkozunk, és egy új, a korlátozáskielégítésről szóló fejezetet is beiktattunk. Ez utóbbi természetes átmenetet képez a logikával foglalkozó anyagrészek felé.

- A III. részben az ítéletkalkulus, amit az első kiadásban az elsőrendű logika felé vezető lépcsőnek tekintettünk, itt mint egy önálló hasznos reprezentációs nyelv kerül bemutatásra, gyors következtetési eljárásokkal és áramköralapú ágensfejlesztéssel. Az elsőrendű logikával foglalkozó fejezetet átszerveztük, hogy az anyag áttekinthetőbb legyen és alkalmazási példaként az internetes vásárlás tárgyterülettel bővítettük.

- A IV. részben foglalkozunk olyan újabb módszerekkel, mint a `GRAPHPLAN` és a kielégítésalapú tervkészítés. Bővült az ütemezés, a feltételes tervkészítés, a hierarchikus tervkészítés és a többágenses tervkészítés bemutatása.

- Az V. részben a valószínűségi hálók anyagát új algoritmusokkal bővítettük, mint például a változóeliminálás és Markov lánc Monte Carlo algoritmusok. Egy, a bizonytalan temporális következtetésekkel foglalkozó új fejezetet írtunk, ahol rejtett Markov-modellekről, Kalman-szűrőkről és dinamikus valószínűségi hálókról van szó. Mélyebb a Markov döntési folyamat tárgyalása, új alfejezetek foglalkoznak a játékelmélettel és a mechanizmustervezéssel.

- A VI. részben kapcsoljuk össze a statisztikai, a szimbolikus és a neurális tanulást továbbá a boosting algoritmusokkal, az EM algoritmussal, a példányalapú tanulással és a kernelmódszerekkel (support vector machines, szupport vektor gépek) foglalkozó részekkel bővítjük anyagot.

- A VII. részben a nyelvfeldolgozás tárgyalását a párbeszédmegértéssel és a nyelvtanindukálással bővítjük. Új rész foglalkozik a nyelv valószínűségi modellezésével, az információ-visszakeresési és fordítási alkalmazásokkal. A robotika tárgyalásánál kiemeljük a bizonytalan szenzoradatok egyesítését. A látással foglalkozó anyagrész objektumfelismeréssel bővült.

- A VIII. részben egy új alfejezetben az MI etikai következményeivel foglalkozunk.

A könyv javasolt használata

A könyvnek 27 fejezete van, mindegyik nagyjából egyhetes előadásnyi anyagot jelent. Az egész könyv feldolgozása két félévet tenne így ki. A tanfolyamot azonban a szűkebb hallgatói és előadói érdeklődéshez is szabhatjuk. Mivel a könyv széles területet fed le, jó háttéranyag lehet egy rövid bevezető alapkurzushoz, de felhasználható egyes témaköröket magasabb szinten bemutató specializált graduális kurzusokhoz is. A könyv első kiadását felhasználó 600 egyetemi kurzus tematikájából egy ízelítő az *aima.cs.berkeley.edu* weblapon található. Itt néhány hasznos tanács is található arra nézve, hogy konkrét speciális igényekhez hogyan állíthatunk össze egy megfelelő témaszekvenciát.

A könyv 385 gyakorlatot tartalmaz. Azokat, amelyekhez nagyobb lélegzetű programozási munka is szükséges, egy klaviatúra ikon jelzi. Az ilyen gyakorlatokat legjobban úgy lehet megoldani, hogy igénybe vesszük az *aima.cs.berkeley.edu* programkódtár szolgáltatásait. Méreténél fogva néhány gyakorlat féléves házi feladatként is kiadható.

Created by XMLmind XSL-FO Converter.

Olvasnivaló. A gyakorlatok egy része irodalmazást is igényel. Ezeket egy nyitott könyv ikon jelzi.

Fontos

A könyvben a fontos részeket egy mutató kéz ikon jelzi. Hogy a könyvben az egyes témákat könnyű legyen megtalálni, kiterjedt tárgymutatót is készítettünk.

A Weboldal használatáról

Az *aima.cs.berkeley.edu* weboldalon megtalálhatók:

- a könyvbeli algoritmusok implementációja több különböző programozási nyelven;
- a könyvet használó több mint 600 oktatási intézmény listája, számos webes linkkel az online oktatási anyagokhoz;
- annotált linkek listája több mint 800 olyan weboldalhoz, ahol hasznos MI-tartalom található;
- a segédanyagok és linkek fejezetenkénti listája;
- útmutató, hogy hogyan csatlakozhatunk a könyvvel foglalkozó vitafórumhoz;
- útmutató, hogy a kérdéseinkkel és az észrevételeinkkel hogyan léphetünk kapcsolatba a szerzőkkel;
- útmutató, hogy a nagy valószínűséggel előforduló hibákat hogyan jelezzük;
- a könyvbeli ábrák, fóliák és más oktatáshoz hasznos anyagok.

Köszönetnyilvánítás

A 24. fejezet (Az észlelés) nagy részét Jitendra Malik, a 25. fejezet (Robotika) többségét az első kiadásban John Canny, a jelen kiadásban pedig Sebastian Thrun írta. Az Irodalmi és történeti megjegyzéseket az első kiadáshoz Doug Edwards kutatta fel. Tim Huang, Mark Paskin és Cynthia Bruyns segítettek az ábrák és az algoritmusok megfelelő formájú kialakításában. A Prentice Hall kiadónál Alan Apt, Sondra Chavez, Toni Holm, Jake Warde, Irwin Zucker és Camille Trentacoste minden tőlük telhetőt megtettek, hogy rákényszerítsenek bennünket a határidők betartására, és sok hasznos javaslattal segítették, hogy a könyv a megfelelő tartalommal és megfelelő köntösben jelenjen meg.

Stuart meg szeretné köszönni szüleinek a folyamatos támogatást és bátorítást, feleségének, Loy Sheflottnak a végtelen türelmét és a határtalan bölcsességét. Reméli, hogy Gordon és Lucy rövidesen már olvashatják is ezt. Igen segítőkésznek bizonyult az RKHC (Russell Kivételes Hallgatói Csoportja) is.

Peter szeretné megköszönni szüleinek (Torsten és Gerda) az indíttatást. Feleségének (Kris), gyerekeinek és barátainak köszöni a bátorítást és azt, hogy elviselték őt a könyv megírásának hosszú ideje és a könyv átírásának még hosszabb ideje alatt.

Adósak vagyunk a Berkeley, a Stanford, az MIT és a NASA könyvtárosainak és a CiteSeer, továbbá a Google fejlesztőinek, hogy a kutatás módját forradalmasították.

Nem tudunk külön-külön köszönetet mondani mindenkinek, aki a könyvet használta, és hasznos megjegyzésekkel állt elő, azonban az alább felsoroltak különösen hasznos megjegyzéseit meg szeretnénk köszönni: Eyal Amir, Krzysztof Apt, Ellery Aziel, Jeff Van Baalen, Brian Baker, Don Barker, Tony Barrett, James Newton Bass, Don Beal, Howard Beck, Wolfgang Bibel, John Binder, Larry Bookman, David R. Boxall, Gerhard Brewka, Selmer Bringsjord, Carla Brodley, Chris Brown, Wilhelm Burger, Lauren Burka, Joao Cachopo, Murray Campbell, Norman Carver, Emmanuel Castro, Anil Chakravarthy, Dan Chisarick, Roberto Cipolla, David Cohen, James Coleman, Julie Ann Comparini, Gary Cottrell, Ernest Davis, Rina Dechter, Tom Dietterich, Chuck Dyer, Barbara Engelhardt, Doug Edwards, Kutluhan Erol, Oren Etzioni, Hana Filip, Douglas Fisher, Jeffrey Forbes, Ken Ford, John Fosler, Alex Franz, Bob Futrelle, Marek Galecki, Stefan Gerberding, Stuart Gill, Sabine Glesner, Seth Golub, Gosta Grahne, Russ Greiner, Eric Grimson, Barbara Grosz, Larry Hall, Steve Hanks, Othar Hansson, Ernst Heinz, Jim Hendler, Christoph Herrmann, Vasant Honavar, Tim Huang, Seth Hutchinson, Joost Jacob, Magnus Johansson, Dan Jurafsky, Leslie Kaelbling, Keiji Kanazawa, Surekha Kasibhatla, Simon Kasif, Henry Kautz, Gernot Kerschbaumer, Richard Kirby, Kevin Knight, Sven Koenig,

Created by XMLmind XSL-FO Converter.

Daphne Koller, Rich Korf, James Kurien, John Lafferty, Gus Larsson, John Lazzaro, Jon LeBlanc, Jason Leatherman, Frank Lee, Edward Lim, Pierre Louveaux, Don Loveland, Sridhar Mahadevan, Jim Martin, Andy Mayer, David McGrane, Jay Mendelsohn, Brian Milch, Steve Minton, Vibhu Mittal, Leora Morgenstern, Stephen Muggleton, Kevin Murphy, Ron Musick, Sung Myaeng, Lee Naish, Pandu Nayak, Bernhard Nebel, Stuart Nelson, XuangLong Nguyen, Illah Nourbakhsh, Steve Omohundro, David Page, David Palmer, David Parkes, Ron Parr, Mark Paskin, Tony Passera, Michael Pazzani, Wim Pijls, Ira Pohl, Martha Pollack, David Poole, Bruce Porter, Malcolm Pradhan, Bill Pringle, Lorraine Prior, Greg Provan, William Rapaport, Philip Resnik, Francesca Rossi, Jonathan Schaeffer, Richard Scherl, Lars Schuster, Soheil Shams, Stuart Shapiro, Jude Shavlik, Satinder Singh, Daniel Sleator, David Smith, Brian So, Robert Sproull, Lynn Stein, Larry Stephens, Andrea Stolcke, Paul Stradling, Devika Subramanian, Rich Sutton, Jonathan Tash, Austin Tate, Michael Thielscher, William Thompson, Sebastian Thrun, Eric Tiedemann, Mark Torrance, Randall Upham, Paul Utgoff, Peter van Beek, Hal Varian, Sunil Vemuri, Jim Waldo, Bonnie Webber, Dan Weld, Michael Wellman, Michael Dean White, Kamin Whitehouse, Brian Williams, David Wolfe, Bill Woods, Alden Wright, Richard Yen, Weixiong Zhang, Shlomo Zilberstein és a Prentice Hall névtelen bírálói.

A könyv borítójáról

A borítón látható képet a szerzők szerkesztették és Lisa Marie Sardegna és Maryann Simmons kivitelezte SGI Inventor™ és az Adobe Photoshop™ segítségével. A címlap az MI történetéből az alábbiakat eleveníti fel:

1. Arisztotelész tervkészítő algoritmusa a *De Motu Animalium* c. műből (kb. i.e. 400).
2. Ramon Lull fogalomgenerátora az *Ars Magna* c. műből (1300 körül).
3. Charles Babbage Differenciál Gépe, az első univerzális számítógép prototípusa (1848).
4. Az elsőrendű logika Gottlob Frege-féle formalizmusa (1789).
5. A logikai következtetés Lewis Carroll-féle diagramjai.
6. A valószínűségi hálózat Sewall Wright-féle ábrázolása (1921).
7. Alan Turing (1912–1954).
8. Shakey a Robot (1969–1973).
9. Egy korszerű diagnosztikai szakértői rendszer (1993).

A Szerzőkről

Stuart Russell 1962-ben Portsmouth-ban, Angliában született. 1982-ben Oxfordban fizikából szerzett kitüntetéses oklevelet, majd 1986-ban a Stanfordon doktorált számítógépes tudományokból. Ettől kezdve a Kalifornai Berkeley Egyetemen (University of California at Berkeley) dolgozik, ahol jelenleg a számítógépes tudományok professzora, az Intelligens Rendszerek Központ igazgatója és a Smith–Zadeh mérnöktudományi tanszék vezetője. 1990-ben megkapta a Nemzeti Kutatási Alap Fiatal Kutatók Elnöki Díját, 1995-ben pedig másodmagával a Számítógép és a Gondolat Díjat. 1996-ban a Miller Intézet ösztöndíjas professzora volt a Kaliforniai Egyetemen és 2000-ben megkapta a tekintélyes hároméves, rektori kutatói-professzori ösztöndíjat. 1998-ban a Stanfordon Forsythe-emlékelőadásokat tartott. Az Amerikai MI Társaság tagja, korábban pedig a társaság vezetőségi tagja volt. Több mint 100 cikket publikált az MI különböző területeiről. További könyvei között megtaláljuk a *The Use of Knowledge in Analogy and Induction* és (Eric Wefalddal együtt) a *Do the Right Thing: Studies in Limited Rationality* c. könyveket.

Peter Norvig a Search Quality részleg vezetője a Google-nál. Az Amerikai MI Társaság vezetőségi tagja. Korábban a NASA Ames Research Centerben a Számítástudományi Osztály igazgatója volt, ahol felügyelte a NASA MI- és robotikai kutatásait és fejlesztéseit. Ezt megelőzően a Junglee-nél dolgozott tudományos főmunkatársként, ahol hozzájárult az első internetes információkinyerő szolgáltatás kifejlesztéséhez, valamint a Sun Microsystems Laboratoriesnál vezető kutatóként, ahol az intelligens információ-visszaszerzésen dolgozott. Diplomáját alkalmazott matematikában a Brown Egyetemen, PhD-fokozatát, számítógépes tudományokban, a Kaliforniai Egyetemen Berkeleyben kapta. A Dél-kaliforniai Egyetemen professzorként, a Berkeleyben, Kaliforniában pedig a tanszéki kutatócsoport tagjaként dolgozott. Több mint 50 publikációja van a számítógépes tudományok területén. Ezek között találjuk a *Paradigms of AI Programming: Case Studies in Common Lisp*, *Verbmobil: A Translation System for Face-to-Face Dialog* és *Intelligent Help System for UNIX* c. könyveket.

Created by XMLmind XSL-FO Converter.

1. fejezet - Bevezetés

Amelyben megkíséreljük megmagyarázni, hogy miért tartjuk a mesterséges intelligenciát olyan területnek, amit igenis érdemes tanulmányozni, és amelyben azt is megkíséreljük eldönteni, hogy vajon mi is ez a terület.

Az ember **Homo sapiens**nek – gondolkodó embernek – nevezte el a saját fajtáját, mert mentális képességeink annyira fontosak a számunkra. Évezredek óta próbáljuk megérteni azt, hogy *hogyan gondolkozunk*. Azt, hogy az anyag néhány maréknyi mennyisége hogyan képes a saját magánál sokkal nagyobb és bonyolultabb világot észlelni, megérteni, a világ alakulását megjósolni és manipulálni. A **mesterséges intelligencia** (**artificial intelligence**), röviden MI (AI), még tovább is megy: az intelligens entitások megértése mellett ilyen entitások *építésével* is próbálkozik.

Az MI az egyik legújabb tudományterület. A munka közvetlenül a második világháború befejeztével kezdődött. A mesterséges intelligencia elnevezés 1956-ban született meg. A molekuláris biológiával egyetemben az MI az a terület, amiről más tudományterületek kutatói úgy nyilatkoznak, hogy „én legszívesebben ezzel foglalkoznék”. Egy fizikushallgató teljes joggal gondolhatja, hogy Galilei, Newton, Einstein és mások már minden jó ötlettel rég előálltak. Ezzel szemben az MI-ben az Einstein-kaliberű egyéniségek előtt még szabad az út.

Az MI jelenleg az általános rendeltetésű területektől, mint az észlelés és a tanulás, egészen olyan specifikus feladatokig, mint a gépi sakk, a matematikai tételbizonyítás, a gépi költészet vagy az orvosi diagnózis, a legkülönbözőbb részterületek óriási választékát öleli át. Az MI az intellektuális képességeket igénylő feladatokat rendszerezi és automatizálja, és így potenciálisan releváns az értelmes emberi cselekvés minden területén. Ilyen értelemben az MI igazán univerzális tudomány.

1. Mi az MI?

Eddig arról beszéltünk, hogy miért izgalmas az MI, de nem mondtuk meg, hogy *mi is* valójában? Az MI definícióit, nyolc jelenlegi tankönyvet követve, az 1.1. ábra mutatja. Ezek a definíciók két dimenzió mentén értelmezhetők. Az ábra felső részében levők a *gondolati folyamatokat* és a *következtetést* célozzák, míg az alsó részben levők tárgya a *viselkedés*. A bal oldali definíciók a sikert az *emberi* teljesítményhez mérik, míg a jobb oldaliak mércéje az intelligencia egy *ideális* koncepciója, amit mi **racionalitás**nak (**rationality**) fogunk nevezni. Egy rendszer racionális, ha tudásához viszonyítva helyesen cselekszik.

1.1. ábra - Az MI néhány meghatározása négy kategóriába szervezve

Created by XMLmind XSL-FO Converter.

Emberi módon gondolkodó rendszerek	Racionálisan gondolkodó rendszerek
„Izgalmas újszerű kísérlet, hogy a számítógépet gondolkodásra késztessük… *tudatos gépek*, e fogalom teljes és szó szerinti értelmében" (Haugeland, 1985) „Az emberi gondolkodással asszociálható olyan aktivitások [automatizálása], mint pl. a döntéshozatal, a problémamegoldás, a tanulás,…" (Bellman, 1978)	„A mentális képességek tanulmányozása számítási modellek segítségével" (Charniak és McDermott, 1985) „Az észlelést, a következtetést és a cselekvést biztosító számítási mechanizmusok tanulmányozása" (Winston, 1992)
Emberi módon cselekvő rendszerek	**Racionálisan cselekvő rendszerek**
„Az olyan funkciókat teljesítő gépi rendszerek létrehozásának a művészete, amelyhez az intelligencia szükséges, ha azt emberek teszik" (Kurzweil, 1990) „Annak tanulmányozása, hogy hogyan lehet a számítógéppel olyan dolgokat művelni, amiben pillanatnyilag az emberek a jobbak" (Rich és Knight, 1991)	„Számítási intelligencia az intelligens ágensek tervezésének a tanulmányozása" (Poole és társai, 1998) „Az MI… a műtárgyak intelligens viselkedésével foglalkozik" (Nilsson, 1998)

Az idők folyamán mind a négy irányzat követőre talált. De ahogy ez várható volt, feszültség uralkodik az embercentrikus és a racionalitáscentrikus irányzatok között.[1] Az embercentrikus irányzat szükségképpen empirikus tudomány, hipotézisekkel és empirikus igazolással.

A racionalitáscentrikus megközelítés a matematikára és a mérnöki tudományokra támaszkodik. Mindegyik csoport rágalmazta is, de segítette is a többi csoport munkáját. Nézzük most a négyféle megközelítést kicsit részletesebben.

1.1. Emberi módon cselekedni: Turing-teszt megközelítés

A **Turing-teszt**et (**Turing-test**) Alan Turing javasolta azzal a céllal, hogy az intelligenciának egy kielégítő munkadefiníciót adjon (Turing, 1950). Ahelyett hogy az intelligenciára jellemző kvalitások hosszú és feltehetően vitás listáját megadná, Turing egy tesztet javasolt, amelynek alapja a vitathatatlanul intelligens entitástól – egy embertől – való megkülönböztethetetlensége. A számítógép akkor állja ki a próbát, ha az emberi kérdező néhány írásos kérdés feltevése után nem képes eldönteni, hogy az írásos válaszok egy embertől vagy egy géptől érkeznek-e. A tesztet a 26. fejezet tárgyalja részletesen, ahol azzal is foglalkozunk, hogy intelligensnek mondható-e a számítógép, ha teljesíti a tesztet. Jelenleg igen sok feladatot jelent egy gépet úgy programozni, hogy teljesítse a tesztet. A számítógépnek a következő képességekkel kellene rendelkeznie:

- **természetes nyelvfeldolgozás** (**natural language processing**) a sikeres angol (illetve más emberi) nyelvű párbeszédhez;
- **tudásreprezentáció** (**knowledge representation**) az ismert vagy hallott információ tárolására;
- **automatizált következtetés** (**automated reasoning**), hogy a tárolt információt kérdések megválaszolására és új következtetések levonására használjuk;
- **gépi tanulás** (**machine learning**) az új körülményekhez való adaptálódáshoz, a mintázatok detektálására és általánosítására.

[1] Meg kell jegyeznünk, hogy az *emberi* és a *racionális* viselkedés megkülönböztetése nem jelenti, hogy az emberek szükségszerűen „irracionálisak" „emocionálisan instabil", „elmebeteg" értelemben. Azonban fel kell figyelni arra, hogy nem vagyunk tökéletesek. Nem vagyunk valamennyien sakkmesterek, azok sem, akik a sakk minden szabályával tisztában vannak. Sajnálatos módon nem is vizsgázik mindenki jelesre. Az emberi következtetést terhelő szisztematikus hibák egy részét Kahneman sorolja fel (Kahneman és társai, 1982).

Created by XMLmind XSL-FO Converter.

A Turing-teszt a kérdező és a számítógép közötti fizikai kölcsönhatást szándékosan kerülte, mert egy személy *fizikai* szimulációja az intelligenciához nem szükséges. Az ún. **teljes Turing-teszt** (**total Turing-test**) azonban videojelet is fel kell dolgozzon, hogy a kérdező tesztelni tudja az illető érzékelési képességeit, valamint tartalmazza annak a lehetőségét is, hogy a kérdező az objektumokat átadhassa „egy nyíláson keresztül”. A teljes Turing-teszt teljesítéséhez a számítógépnek szüksége lesz:

- **gépi látás**ra (**computer vision**), az objektumok érzékeléséhez és
- **robotiká**ra (**robotics**) az objektumok mozgatásához.

Ez a hat terület nagyjából lefedi az MI-t. Turing becsületére válik, hogy egy olyan tesztet talált ki, amely 50 év múltával is releváns maradt. Az MI-n belül a kutatók sok erőfeszítést mégsem fejtettek ki a Turing-teszt teljesítése érdekében, abban a hiedelemben, hogy a mögötte sorakozó elvek tanulmányozása fontosabb, mint egy példány duplikálása. A „mesterséges repülés” kutatása akkor járt sikerrel, amikor a Wright testvérek abbahagyták a madarak utánzását, és az aerodinamikát kezdték tanulmányozni. A repüléssel foglalkozó könyvek nem azt a célt fogalmazzák meg, hogy olyan „gépeket kellene építeni, amelyek annyira hasonlóan repülnek a galambokhoz, hogy ezzel akár más galambokat is képesek megtéveszteni”.

1.2. Emberi módon gondolkodni: a kognitív modellezés

Ha azt szeretnénk kijelenteni, hogy egy program emberi módon gondolkodik, valamilyen módon meg kellene határoznunk, hogy az emberek hogyan gondolkodnak. Ehhez az emberi elme működési mechanizmusának *belsejébe* kellene tudnunk belenézni. Két módja van ennek: önelemzés révén – az átsuhanó gondolataink megragadásával – vagy pszichológiai kísérletekkel. Ha egyszer majd rendelkezünk az elme elegendően részletes elméletével, lehetségessé válik az elmélet számítógépes programmal való kifejezése. Ha a program bemenetei és kimenetei, valamint időzítése az emberi viselkedéssel megegyeznek, ez egyben bizonyíték arra, hogy a program bizonyos mechanizmusai feltehetően az emberben is megtalálhatók. Így például Newell és Simon, akik az „általános problémamegoldót (General Problem Solver, GPS)” (Newell és Simon, 1961) kifejlesztették, csupán azzal, hogy a programjuk helyesen oldja meg a problémákat, nem voltak elégedettek. Jobban érdekelte őket, hogy összehasonlítsák a program következtetési lépéseit az ugyanazon a feladaton dolgozó emberek lépéseivel. Az MI számítógépes modelljeit és a pszichológia kísérleti technikáit a **kognitív tudomány** (**cognitive science**) interdiszciplináris területe kapcsolja össze azáltal, hogy kísérletet tesz precíz és verifikálható elméletek megfogalmazására az emberi elme működéséről.

A kognitív tudomány önmagában is vonzó terület, annyira, hogy egy önálló enciklopédia létrehozásának is van értelme (Wilson és Keil, 1999). A könyvünkben meg sem kíséreljük leírni, hogy az emberi megismerésből mit tudunk már. Esetenként azonban kitérünk az MI-technikák és az emberi gondolkodás közötti hasonlóságokra és különbségekre. Az igazi kognitív tudomány – szükségszerűen – a valódi emberek, illetve állatok kísérleti kutatásán alapul, mi pedig feltételezzük, hogy az olvasó a kísérleteihez csupán számítógéppel rendelkezik.

Az MI-kutatás kezdeti szakaszában a megközelítéseket sokszor össze is keverték. Egy szerző állíthatta például, hogy ha egy algoritmus egy feladaton jól vizsgázik, *akkor* jó modellje az emberi képességeknek, és megfordítva. A mai szerzők a kétféle igényt elkülönítik. Ez a megkülönböztetés mind az MI, mind a kognitív tudomány gyorsabb fejlődését tette lehetővé. Az MI és a kognitív tudomány folyamatosan termékenyítően hatnak egymásra, különösképpen a látás és a természetes nyelv vonatkozásában. A látás a közelmúltban különösen sokat fejlődött a neurofiziológiai eredményeket és a számítási modelleket összefogó integrált megközelítés révén.

1.3. Racionálisan gondolkodni: a gondolkodás törvénye

Arisztotelész görög filozófus volt az elsők egyike, aki megkísérelte a „helyes gondolkodás”, azaz a megcáfolhatatlan következtetési folyamatok törvényekbe foglalását. Híres **szillogisztiká**ja (**syllogisms**) olyan mintákat szolgáltatott a következtetési sémákhoz, amelyek helyes premisszákból mindig helyes következményekre vezettek. Például „Szókratész egy ember; minden ember halandó; azaz Szókratész halandó.” Ezekről a gondolkodási törvényekről feltételezték, hogy az elme működését irányítják, és ezek indították el a **logika** (**logic**) tudományának kialakulását.

A 19. században a logikusok – a logika tudományát művelők – megadták a világ mindennemű objektumára és az azok közötti relációkra vonatkozó állításoknak a precíz megfogalmazását. (Hasonlítsuk ezt a közönséges aritmetikai jelölésrendszerhez, ami főleg a számokon értelmezett egyenlőségek és egyenlőtlenségek kifejezésére alkalmas.) 1965-ben léteztek már olyan programok, amelyek – legalább elvben – képesek voltak *tetszőleges,*

Created by XMLmind XSL-FO Converter.

logikai jelölésekkel kifejezett problémához megadni a probléma megoldását.[2] Az MI-n belül uralkodó **logicista** (**logicist**) hagyomány azt reméli, hogy ilyen programokra alapozva intelligens rendszereket képes létrehozni.

Ennek a megközelítésnek két alapvető baja van. Először is a logikai jelölésrendszer által igényelt formális elemekkel informális tudást kifejezni nem ilyen egyszerű, különösen ha a tudás nem 100%-osan biztos. A másik az, hogy nagy a különbség egy probléma „elvi", illetve gyakorlati megoldása között. Már a néhány tíz tényt kitevő problémák megoldása is kimerítheti egy tetszőleges számítógépes rendszer számítási erőforrásait, hacsak valamilyen módon nincs vezérelve, hogy melyik következtetési lépésekkel próbálkozzék először. Annak ellenére, hogy ez a két probléma a számítógépes következtető rendszer *bármilyen* elven történő fejlesztésénél felbukkan, először a logicista hagyományban jelent meg.

1.4. Racionálisan cselekedni: a racionális ágens

Egy **ágens** (**agent**) nem más, mint valami, ami cselekszik (az *ágens* szó forrása a latin *agere* – cselekedni). Számítógépes ágensektől azonban elvárjuk, hogy legyenek más jellemzői is, amelyekben különböznek a „mezei" programoktól. Ilyen jellemzők például az autonóm vezérlés felügyelte cselekvés, a környezet észlelése, a hosszabb idejű tartós létezés, a változásokhoz történő adaptáció és mások céljainak az átvétele. Egy **racionális ágens** (**rational agent**) a legjobb kimenetel érdekében vagy – bizonytalanság jelenlétében – a legjobb várható kimenetel érdekében cselekszik.

A „gondolkodás törvénye" megközelítésben a hangsúly teljes egészében a korrekt következtetéseken volt. Korrekt következtetések meghozatala egyes esetekben a racionális ágensek *részét* képezi, hiszen a racionális cselekvés egyik módja, hogy logikusan következtetve olyan következményekre jutunk, hogy adott cselekvés a céljaink elérését biztosítja. A korrekt következtetés azonban nem *egésze* a racionalitásnak, mert gyakran megesik, hogy akkor is cselekedni kell, amikor bizonyíthatóan korrekt cselekvés nem létezik. A racionális cselekvésnek olyan esetei is vannak, amikor következtetésnek nyoma sincs. Így például a forró kályhától a kezet elrántó reflexszerű cselekvés sikeresebb, mint a tudatos megfontolásból adódó lassabb mozdulat.

A Turing-teszthez szükséges minden képesség lényeges a racionális cselekvés megvalósítása szempontjából. A tudásreprezentálás és a következtetés képessége szükséges ahhoz, hogy a helyzetek széles spektrumában helyes döntésre juthassunk. Természetes nyelven megfogalmazott érthető szövegek kimondásával egy komplex társadalomban is elboldogulunk. Tanulásra nem a műveltség kedvéért van szükség. A világ működéséről alkotott jobb elképzelések hatékonyabb stratégiákhoz vezethetnek a környezetünkkel való kapcsolat kialakításában. A vizuális érzékelés nem azért szükséges, mert látni szórakoztató dolog, hanem mert így jobb elképzelésünk lesz arról, hogy egy cselekvéssel mit érhetünk el. Egy ízletes falat látványa például segít abban, hogy tudatosan érte nyúljunk.

Fontos

Emiatt az MI-nek mint racionális ágensek tervezésének a tanulmányozása két előnyt jelent. Először is ez a megközelítés a „gondolkodás törvénye" megközelítésnél általánosabb, hiszen a korrekt következtetés csupán egyike a racionalitást biztosító mechanizmusoknak. Másodszor, tudományosan jobban is kezelhető, mint azok a megközelítések, amelyek az emberi viselkedésen vagy emberi gondolkodáson alapulnak, mert a racionalitás mértéke jól definiált és teljesen általános. Az emberi viselkedés viszont egy bizonyos specifikus környezethez jól adaptálódott, és részben egy olyan bonyolult, zömében nem ismert evolúciós folyamat eredménye, amely valószínűleg még messze áll a tökéletességtől. *Könyvünk tehát a racionális ágensek általános elveire és a létrehozásukhoz szükséges komponensekre koncentrálódik.* Látni fogjuk, hogy a problémamegfogalmazás látszólagos egyszerűsége ellenére, hihetetlen sok következménnyel kell szembenéznünk, ha megkíséreljük a problémát megoldani. Néhány ilyen következményről részletesebben a 2. fejezet szól.

Egy fontos dologról nem szabad elfeledkezni: rövidesen meglátjuk, hogy összetett környezetben a tökéletes racionalitást – mindig helyesen cselekedni – lehetetlen elérni. A számítási szükségletek egyszerűen túl nagyok. A könyv nagyobb részében azonban azzal a munkahipotézissel fogunk élni, hogy a tökéletes döntéshozatal megértése jó kiindulópont. A probléma így egyszerűbbé válik, és megfelelő keretet nyújt a területhez tartozó alapozó anyag többségéhez. A **korlátozott racionalitás**sal (**limited rationality**) – azaz megfelelően cselekedni, miközben az összes kívánt számítás elvégzésére nincs elegendő idő – explicit módon a 6. és a 17. fejezet foglalkozik.

[2] Ha nincs megoldás, a program a keresést esetleg soha nem fejezi be.

Created by XMLmind XSL-FO Converter.

2. A mesterséges intelligencia alapjai

Ebben a részben azon tudományterületek rövid történetét mutatjuk be, amelyek ötleteikkel, nézőpontjukkal, módszereikkel hozzájárultak az MI-hez. Mint minden történetnél, most is csak kevés személyről és eseményről tudunk megemlékezni, másokat, akik, illetve amelyek szintén fontosak, figyelmen kívül hagyunk. Az eseményeket egy kérdéssorozat körül rendeztük sorba. Ugyanakkor azt a benyomást sem szeretnénk kelteni, mintha ezek a kérdések a terület által kizárólagosan vizsgált kérdések lennének, vagy, hogy ezek a tudományterületek célkitűzéseikben mind az MI irányába fejlődtek volna.

2.1. Filozófia (i. e. 428-tól napjainkig)

- Lehet-e formális szabályok révén helyes konklúziókhoz jutni?
- Hogyan emelkedik ki a mentális elme a fizikai agyból?
- Honnan jön a tudás?
- A tudás hogyan vezet a cselekvéshez?

Arisztotelész (i. e. 384–322) volt az első, aki megkísérelte precízen megfogalmazni az elme racionális részén uralkodó törvényszerűségeket. A korrekt következtetéseket leíró szillogizmusok informális rendszerét fejlesztette ki, ami elvben lehetővé tette a konklúziók gépies származtatását, ha a kezdeti premisszák egyszer adottak voltak. Sokkal később Ramon Lull (élt 1315-ig) javasolta, hogy a hasznos következtetést éppenséggel egy gépezetre rá lehetne bízni. Az általa javasolt „fogalmi kerekek” a könyv borítóján láthatók. Thomas Hobbes (1588–1679) felvetette, hogy a következtetés olyan, mint egy numerikus számítás, vagyis hogy „csendes gondolatainkban kivonunk és összeadunk”. Eközben a számítások automatizálása már a legjobb úton haladt. 1500 körül Leonardo da Vinci (1452–1519) megtervezett – bár meg nem épített – egy mechanikus kalkulátort. A kalkulátor közelmúltbeli rekonstrukciója bebizonyította, hogy a tervezet működőképes volt. Az első ismert számítógépet 1623 körül Wilhelm Schickard (1592–1635) német tudós konstruálta, bár a Blaise Pascal (1623–1662) által 1642-ben épített Pascaline ismertebb. Pascal azt írta, hogy „az aritmetikai gép olyan hatást produkál, ami közelebbinek tűnik a gondolathoz, mint az állatok összes cselekvése”. Gottfried Wilhelm Leibniz (1646–1716) egy mechanikus berendezést épített, amely nem a számokon, hanem a fogalmakon végzett műveleteket, bár működési köre igen korlátozott volt.

Most, hogy van már elképzelésünk arról, hogy az elme formális, racionális részét milyen elvek alapján lehet leírni, a következő lépésként az elmét fizikai rendszernek tekintjük. René Descartes (1596–1650) elemezte először világosan az elme és az anyag közötti különbséget és az ebből adódó problémákat. Az elme tiszta fizikai megközelítésének egyik problémája az, hogy kevés helyet hagy a szabad akarat számára. Ha az elmét csupán fizikai törvények uralják, nincs több szabad akarata mint egy sziklának, amely „eldönti”, hogy a föld középpontja felé fog esni. Annak ellenére, hogy Descartes a következtetés hatalmának szószólója volt, ő volt az is, aki a **dualizmus**t (**dualism**) javasolta. Azt tartotta, hogy az elmének (vagy a léleknek) van olyan része, amely nem része a természetnek és nem alanya a fizikai törvényeknek. Ezzel szemben úgy érezte, hogy az állatok nélkülözik ezt a dualista természetet, lényegében gépeknek tekintendők. A dualizmus alternatívája a **materializmus** (**materialism**), amely azt tartja, hogy éppen az agynak a fizikai törvények szerinti működése *valósítja meg* az elmét. A szabad akarat nem más, mint az a mód, ahogyan a választási lehetőségek észlelése a döntési folyamat számára megjelenik.

Ha az elme a benne levő ismereteken operáló fizikai eszköz, akkor a következő probléma a tudás forrásának a meghatározása. Francis Bacon (1561–1626) *Novum Organum*ától[3] induló **empiricista** (**empiricist**) mozgalmat John Locke (1632–1704) kijelentése jellemzi: „Nincs semmi a megértésben, ami előbb ne létezne az érzékszervekben.” David Hume (1711–1776) *A Treatise of Human Nature* c. művében (Hume, 1739) javasolja azt a fogalmat, amit ma **indukció**nak (**induction**) nevezünk, nevezetesen hogy az általános elveket a komponenseik ismétlődő kapcsolataiból emeljük ki. Ludwig Wittgenstein (1889–1951) és Bertrand Russell (1872–1970) munkájára alapozva a Rudolf Carnap (1891–1970) vezette híres Bécsi Kör kialakította a **logikai pozitivizmus** (**logical positivism**) doktrínáját. Ez a doktrína azt állítja, hogy minden tudást végső soron az érzékszervi bemeneteknek megfelelő **megfigyeléses állítások**on (**observation sentences**) alapuló logikai elméletekkel meg lehet magyarázni.[4] Rudolf Carnap és Carl Hempel (1905–1997) **igazoló elmélet**e

[3] Az Arisztotelész-féle *Organon*, azaz a gondolkodás eszközének egy újabb változata.
[4] Ebben a megközelítésben, megvizsgálva a szavak értelmét illetve kísérleteket végezve, vagy igazolni, vagy cáfolni lehet minden értelmes állítást. Mivel ez – a szándéknak megfelelően – a metafizika többségét kizárja, a logikai pozitivizmus bizonyos körökben nem volt népszerű.

Created by XMLmind XSL-FO Converter.

(**confirmation theory**) megkísérelte megvilágítani, miként gyarapítja a tudást a tapasztalat. Carnap könyve, a *The Logical Structure of the World* (Carnap, 1928) a tudás elemi tapasztalatból történő kinyerésére egy explicit számítási eljárást definiált. Ez volt talán az első olyan elmélet, amely az elmét számítási folyamatként ábrázolta.

Az elme filozófiai képének utolsó eleme a tudás és a cselekvés kapcsolata. Az MI szempontjából ez a kérdés elsődleges fontosságú, mert az intelligencia cselekvést is és következtetést is igényel. Továbbá ahhoz, hogy világos legyen előttünk, hogyan kell egy olyan ágenst építeni, amelynek cselekvései jogosak vagy racionálisak lesznek, meg kell értenünk, hogy hogyan igazolhatók a cselekvések. Arisztotelész a cselekvéseket azért tartotta jogosnak, mert a célok és a cselekvések kimenetelei között logikai kapcsolat lelhető fel (az alábbi idézet záró része a könyv borítóján is megtalálható):

> De hogyan lehetséges, hogy a gondolatot néha követi cselekvés, és néha nem, néha mozgással és néha anélkül? Úgy tűnik, mintha ugyanaz a dolog történne itt is, mint ami a változatlan objektumokra vonatkozó következtetések meghozatalánál. De ebben az esetben az eredmény egy spekulatív állítás … holott itt a premisszából következő konklúzió egy cselekvés. … Kellene egy ruha, a köpeny egy ruha. Kell egy köpeny. Ami kell, azt elő kell állítanom, és kell egy köpeny. Elő kell állítanom egy köpenyt. A konklúzió tehát, hogy „elő kell állítanom egy köpenyt” nem más, mint egy cselekvés.
>
> —(Nussbaum, 1978, 40. o.)

A *Nicomachean Ethics*ben (III. könyv, 3, 1112b) Arisztotelész a témát tovább folytatja, egy algoritmust javasolva:

> Nem a célokat, hanem az eszközöket mérlegeljük. Az orvos nem azt fontolgatja, hogy gyógyítson-e, a szónok sem, hogy meggyőzzön-e, az államférfi sem azon hezitál, hogy jogot és törvényt teremtsen, és senki sem a célt mérlegeli. Mindenki feltételezi a célt, és azt fontolgatja, hogy az hogyan, milyen eszközökkel érhető el, és hogy azáltal könnyen és a lehető legjobban érhető-e el. És ha valamilyen eszközzel elérhető a cél, azt fontolgatjuk, hogy azzal az eszközzel *hogyan*, milyen módon érhetjük el, amíg a legelső okhoz el nem jutunk. Ez az ok a felfedezés sorában a legutolsó… és ami az elemzés rendjében az utolsó, a megtörténés rendjében elsőnek tűnik. És ha lehetetlenséghez jutunk, a keresést feladjuk, például ha pénzre van szükségünk, azt viszont megszerezni nem lehet. Ha azonban valami lehetségesnek tűnik, akkor megpróbálkozunk vele.
>
> —Arisztotelész

Arisztotelész megközelítését 2300 évvel később Newell és Simon a GPS-programjukban implementálták. Ma ezt regressziós tervkészítő rendszernek neveznénk (lásd 11. fejezet).

A célalapú analízis hasznos, de nem segít akkor, amikor a célt több cselekvéssel is el lehet érni, vagy pedig, ha annak eléréséhez egy cselekvés sem elegendő. Antoine Arnauld (1612–1694) helyesen adta meg azt a kvantitatív formulát, hogy ilyen esetekben milyen cselekvéshez kellene folyamodni (lásd 16. fejezet). John Stuart Mill (1806–1873) *Utilitarianism* c. művében (Mill, 1863) a racionális döntés kritériumának a gondolatát az emberi cselekvés minden aspektusára kiterjesztette. A döntéshozatal egy jobban formalizált elméletéről a következő részben lesz szó.

2.2. Matematika (kb. 800-tól napjainkig)

- Melyek az érvényes következtetések formális szabályai?
- Mi az, ami kiszámítható?
- Hogyan vagyunk képesek a bizonytalan információ alapján következtetéseket levonni?

A filozófusok számba vették az MI fontosabb ötleteit, de ahhoz, hogy a formális tudományra rátérhessünk, szükséges volt legalább három alapvető területnek – a logikának, a számítástechnikának és a valószínűségnek – bizonyos szintű matematikai formalizálása.

A formális logika elképzelése egészen az ókori Görögország filozófusaihoz nyúlik vissza (lásd 7. fejezet), de a matematikai kialakulása George Boole (1815–1864) munkásságával kezdődött igazán, aki részleteiben kidolgozta az ítélet- vagy Boole-logikát (Boole, 1847). 1879-ben Gottlob Frege (1848–1925) kiterjesztette a Boole-logikát, hogy az objektumokat és relációkat is tartalmazhasson. Ezzel megalkotta azt az elsőrendű

Created by XMLmind XSL-FO Converter.

logikát, amely lényegében manapság a legalapvetőbb tudásreprezentációs rendszer.[5] Alfred Tarski (1902–1983) egy referenciaelméletet vezetett be, amely azt mutatja meg, hogy hogyan rendelhetők össze a logika és a valós világ objektumai. A következő lépés annak a kiderítése volt, hogy hol vannak a logikának és a számításoknak az alkalmazási határai.

Az első nem triviális **algoritmus**nak (**algorithm**) a legnagyobb közös osztót kiszámító euklideszi algoritmust tekintik. Az algoritmusok önmagában való tanulmányozása al-Hvárizmi, a 9. századi perzsa matematikus idejéig nyúlik vissza, akinek a művei az arab számokkal és az algebrával is megismertették Európát. Boole és mások a logikai dedukció algoritmusaival foglalkoztak, a 19. század végén pedig már erőfeszítések történtek az általános matematikai érvelés logikai dedukcióként való formalizálására. 1900-ban David Hilbert (1862–1943) 23 problémából álló listát publikált, amelyről – helyesen – azt vélte, hogy matematikus társait a század nagy részében le fogja kötni. Utolsó problémája azt firtatta, vajon létezik-e a természetes számokra vonatkozó tetszőleges logikai állítás igazságát eldöntő algoritmus. Ez a híres ún. *Entscheidungsproblem* (eldönthetőségi probléma). Hilbert lényegében azt a kérdést tette fel, hogy a hatékony bizonyítási eljárásoknak vannak-e elvi korlátjai. Kurt Gödel (1906–1978) 1930-ban megmutatta, hogy a Frege és Russell-féle elsőrendű logikában létezik hatékony eljárás tetszőleges igaz állítás bizonyítására. Az elsőrendű logika azonban nem elegendő a matematikai indukció kifejezéséhez, ez utóbbi viszont a természetes számok jellemzéséhez szükséges. 1931-ben Gödel meg is mutatta, hogy valóban léteznek elvi korlátok. **Nemteljességi tétel**e (**incompleteness theorem**) kimutatja, hogy minden, a természetes számok tulajdonságait kifejezni képes nyelvben léteznek igaz, de abban az értelemben eldönthetetlen állítások, hogy az igazságuk algoritmikusan nem mutatható ki.

Ez az alapvető eredmény úgy is interpretálható, hogy léteznek az egész számokon értelmezhető olyan függvények, amelyek algoritmikusan nem írhatók le, magyarán nem számíthatók ki. Ez motiválta Alan Turingot (1912–1954), hogy megkísérelje a *kiszámítható* függvények pontos jellemzését. Ez a megfogalmazás kissé problematikus, mert egy számítási procedúrának vagy egy hatékony eljárásnak a formális definícióját valójában lehetetlen megadni. Általánosan elfogadott azonban, hogy a Church–Turing-tézis, amely azt állítja, hogy a Turing-automata (Turing, 1936) képes minden kiszámítható függvényt kiszámítani, elegendő definíciót szolgáltat. Turing azt is kimutatta, hogy vannak olyan függvények, amelyeket semmilyen Turing-automata nem képes kiszámítani. Így például nincs olyan automata, amely *általánosságban* el tudná dönteni, hogy egy adott program egy adott bemenetre választ fog-e adni, vagy pedig örökké futni fog.

Ámbár az eldönthetetlenség és a kiszámíthatóság fontos fogalmak a számítások megértéséhez, a **kezelhetetlenség**nek (**intractability**) sokkal nagyobb hatása volt. Durván fogalmazva a problémák egy osztályát kezelhetetlennek mondjuk, ha az egyes problémapéldányok megoldásához szükséges idő legalább exponenciálisan nő a problémapéldányok nagyságától függően. A bonyolultság polinomiális és exponenciális növekedése közötti különbségre először az 1960-as évek közepén hívták fel a figyelmet (Cobham, 1964; Edmonds, 1965). A kétfajta növekedés közötti különbség fontos, mert az exponenciális növekedés következménye, hogy közepes nagyságú problémapéldányokat sem lehet értelmes időkorláton belül megoldani. Az intelligens viselkedés generálásának általános problémáját tehát inkább kezelhető, mint kezelhetetlen részproblémákra kellene lebontanunk.

Hogyan lehet egy kezelhetetlen problémát felismerni? A Stephen Cook és Richard Karp által útjára indított **NP-teljesség** (**NP-completness**) elmélete megadja rá a választ (Cook, 1971; Karp, 1972). Cook és Karp a kanonikus kombinatorikus keresési és következtetési problémák nagy osztályára belátta, hogy azok NP-teljesek. Bármilyen problémaosztály, amire egy NP-teljes problémaosztály visszavezethető, minden bizonnyal kezelhetetlen lesz (és bár még senkinek sem sikerült bebizonyítani, hogy az NP-teljes problémák szükségszerűen nem kezelhetők, kevés tudós van más véleményen). Ezek az eredmények élesen elütnek a számítógép megjelenését kísérő „elektronikus szuperagy – Einsteinnél gyorsabb” lelkesedéstől. Az egyre növekvő számítási sebesség ellenére az intelligens rendszerekre mindig jellemző lesz az erőforrások óvatos és kényes használata. Durván fogalmazva, a világ egy *rendkívül* nagy problémapéldány! Az utóbbi években az MI segített megérteni, hogy az NP-teljes problémák egyes példányai miért nehezek, holott mások könnyűnek bizonyulnak (Cheeseman és társai, 1991).

A logikán és a számításokon túlmenően a matematika harmadik nagy hozzájárulása az MI-hez a **valószínűség-elmélet** (**probability**). A valószínűség gondolatát – a szerencsejátékok lehetséges kimeneteleivel leírva – először az itáliai Girolamo Cardano (1501–1576) fogalmazta meg. A valószínűség hamarosan felbecsülhetetlen komponense lett az összes kvantitatív elméletnek, segítséget adva a bizonytalan mérések és hiányos elméletek kezeléséhez. Pierre Fermat (1601–1665), Blaise Pascal (1623–1662), James Bernoulli (1654–1705), Pierre Laplace (1749–1827) és mások az elméletet tovább fejlesztették, és új statisztikai módszereket vezettek be. A

[5] Az elsőrendű logika Frege javasolta felírása soha nem terjedt el. A nyilvánvaló okát könnyen megértjük, ha a borítón lévő példára pillantunk.

Created by XMLmind XSL-FO Converter.

valószínűségek felfrissítési szabályát – új tények hatására – Thomas Bayes (1702–1761) javasolta. A Bayes-szabály és az azon alapuló Bayes-analízis az MI-rendszerekben alkalmazott bizonytalan következtetés korszerű megközelítésének az alapja.

2.3. Gazdaságtan (1776-tól napjainkig)

- Hogyan kell döntenünk, hogy a hasznunk maximális legyen?
- Mit kell tennünk, ha mások esetleg nem segítőkészek?
- Hogyan kell döntenünk, ha a haszonhoz csak a távoli jövőben jutunk el?

A közgazdaságtan tudománya 1776-ban kapott indító lökést, amikor is Adam Smith skót filozófus (1723–1790) megjelentette az *An Inquiry into the Nature and Causes of the Wealth of Nations* c. művét. Amíg az ókori görögök és mások a közgazdasági gondolkodáshoz járultak hozzá, addig Smith volt az első, aki a gazdaságtant tudományként kezelte, a gazdaságokat a saját gazdasági jólétet maximálizáló különálló ágensekből állóknak képzelve. Az emberek többsége azt véli, hogy a gazdaság lényege a pénz. A közgazdák azonban azt fogják állítani, hogy ők igazából azt tanulmányozzák, hogy az emberek hogyan döntenek, hogy kívánatos eredményekhez jussanak. A „kívánatos eredmények”, avagy a **hasznosság** (**utility**) matematikai kezelését elsőként Léon Walras (1834–1910) formalizálta. Ezt fejlesztette tovább Frank Ramsey (Ramsey, 1931) és később a *The Theory of Games and Economic Behavior* c. művében Neumann János és Oskar Morgenstern (Neumann és Morgenstern, 1944).

A **döntéselmélet** (**decision theory**), amely a valószínűség-elméletet és a hasznossági elméletet kapcsolja össze, a bizonytalanság melletti (gazdasági vagy egyéb) döntéshozatalhoz ad formális és teljes keretet – azaz azokban az esetekben, amikor a valószínűségi leírás megfelelően írja le a döntéshozó környezetét. Ez a megközelítés a „nagy” gazdaságokhoz jó, ahol az ágenseknek nem kell más ágensek (mint entitások) cselekvéseit figyelembe venni. „Kis” gazdaságokban a helyzet inkább egy **játék**ra (**game**) hasonlít: az egyik játékos cselekvései lényeges befolyással lehetnek a másik játékos hasznosságára (akár pozitív, akár negatív értelemben). A Neumann és Morgenstern által kifejlesztett **játékelmélet** (**game theory**) (lásd még Luce és Raiffa, 1957) olyan meglepő eredményeket is tartalmaz, miszerint az egyes játékokban a racionális ágensnek véletlenszerűen kell cselekednie, vagy legalábbis olyan módon, hogy az az ellenfelek számára véletlennek tűnjön.

Harmadik kérdésünkkel, vagyis, hogy hogyan kell racionálisan dönteni, ha a haszon nem azonnali, hanem több egymást követő cselekvés *sorozatának* az eredménye, a közgazdászok többsége nem foglalkozott. Ezt a feladatot az **operációkutatás** (**operational research**) területén vizsgálták, mely Angliában látott napvilágot, amikor a II. világháborúban a radartelepítést igyekezték optimalizálni, és amely később a komplex menedzsment döntéshozatal területén talált polgári alkalmazásokra. Richard Bellman (Bellman, 1957) a **Markov döntési folyamat**oknak (**Markov decision process**es) nevezett szekvenciális döntési problémákat formalizálta. Mi ezeket a 17. és 21. fejezetben tanulmányozzuk.

A közgazdaságtan és az operációkutatás nagyban hozzájárult a racionális ágens felfogásunkhoz, mégis az MI-kutatás évekig egészen más pályán haladt. Az egyik ok a racionális döntéshozatal látszólag igen nagy **komplexitás**a (**complexity**) volt. Herbert Simon (1916–2001), az MI-kutatás egyik úttörője azzal nyerte meg 1978-ban a közgazdasági Nobel-díjat, hogy egy korai munkájában kimutatta, a **kielégítő** döntéshozatalon (**satisfying**), azaz az „elegendően jó” döntéseken alapuló modellek a tényleges emberi viselkedés jobb leírói, mint azok, amelyek hosszadalmas számításokkal meghatározott optimális döntések eredményeképp születtek (Simon, 1947). Az 1990-es években az ágensrendszerek területén a döntéselméleti technikák iránt fokozott érdeklődés mutatkozott (Wellman, 1995).

2.4. Neurális tudományok (1861-től napjainkig)

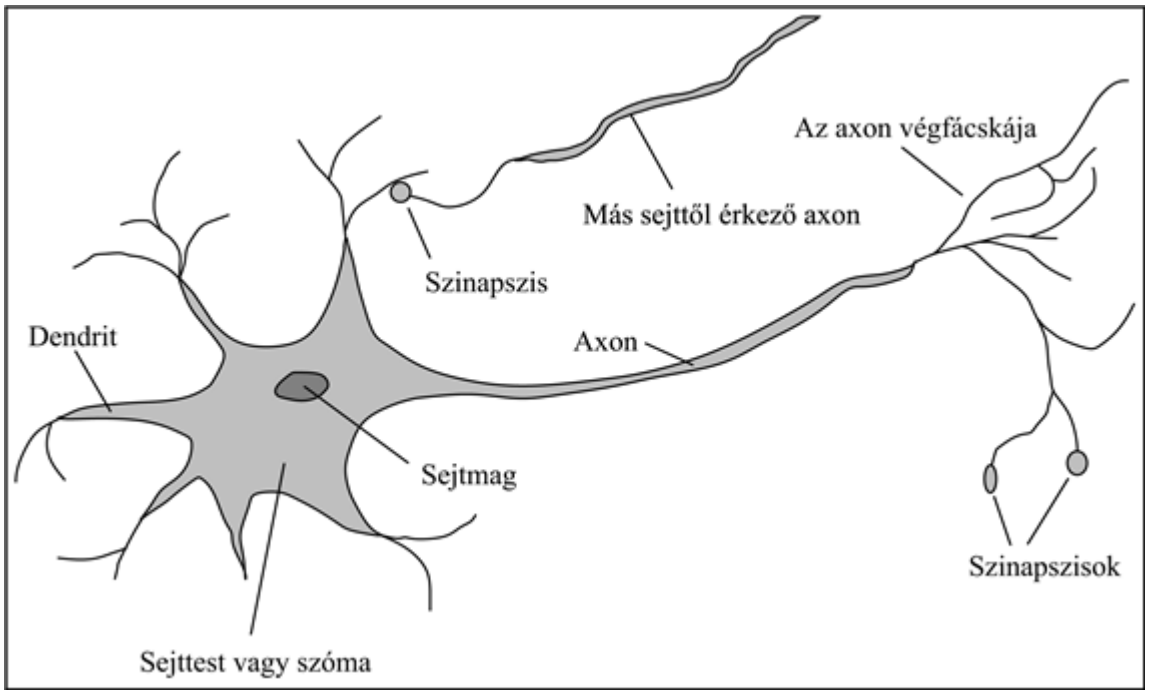
- Hogyan dolgozza fel az információt az agy?

A **neurális tudományok** (**neuroscience**) az idegrendszert, de különösen az agyat tanulmányozzák. Annak egzakt módja, hogy hogyan keletkeznek a gondolatok az agyban, a tudomány egyik nagy feladványa. Évezredek óta tudjuk, hogy az agy valahogy részese a gondolatnak, hiszen egy erős fejbeütés a mentális funkciók károsodásához vezethet. Hosszú ideje azt is tudjuk, hogy az emberi agy valahogy más. I. e. 335-ben

Created by XMLmind XSL-FO Converter.

Arisztotelész azt írta, hogy „az összes állat közül méretéhez képest az ember agya a legnagyobb”.[6] Ennek ellenére csak a 18. század közepétől ismerték fel széles körben, hogy a tudat az agyban lakozik. Korábbi jelöltek között szerepelt a szív, a lép és a tobozmirigy.

1.2. ábra - Az idegsejt vagy neuron részei. Minden neuron sejttestből vagy szómából áll, amely sejtmagot tartalmaz. A sejttestből ágazik el néhány dendritnek nevezett és egy hosszú, axonnak nevezett szál. Az axon jó messzire elér, sokkal messzebbre, mint ahogy ezt a jelen ábra sugallja. Az axonok tipikusan 1 cm hosszúak (ez a sejttest átmérőjének 100-szorosa), de elérhetik az 1 m-t is. Egy neuron 10–100 000 más neuronnal tart kapcsolatot a szinapszisoknak nevezett csatlakozásokon keresztül. A jeleket neuronról neuronra egy összetett elektrokémiai reakció továbbítja. A jelek rövid távon szabályozzák az agy aktivitását, hosszú távon pedig befolyásolják a neuronok helyzetét és a kapcsolódási rendszerüket. Úgy tartják, e mechanizmusok képezik az agyi tanulás alapját. Az információfeldolgozás zöme az agykéregben történik, ami az agy külső rétege. A feldolgozás szervezeti egysége, úgy tűnik, egy kb. 0,5 mm-es átmérőjű szövetoszlop, amelynek magassága az egész kéregre kiterjed, ez az embereknél kb. 4 mm. Egy oszlop kb. 20 000 neuront tartalmaz.



Paul Broca (1824–1880) afázia (beszédzavar) kutatása agysérült pácienseknél 1861-ben lökést adott e területnek, és meggyőzte az orvosi szakmát, hogy az agynak léteznek lokalizált, konkrét kognitív funkciókért felelős részei. Konkrétan azt mutatta ki, hogy a beszéd előállítása a bal félteke, most Broca-féle területnek nevezett részére korlátozódik.[7] Ekkor tudták már, hogy az agy idegsejtekből – **neuron**okból (**neuron**) – áll, de 1873-ig, amikor Camillo Golgi (1843–1926) kifejlesztett egy festési módszert, nem tudták az agyban az egyedi neuronokat megfigyelni (lásd 1.2. ábra). Ezt a módszert alkalmazta Santiago Ramon y Cajal (1852–1934) az agy neurális struktúráit feltáró úttörő munkájában.[8]

Ma már vannak adataink arról, hogy az agy egyes területei és az általuk kontrollált, vagy a szenzorikus bemeneteket generáló test részei között milyen leképezések vannak. Az ilyen leképezések hetek alatt képesek

[6] Azóta sikerült felfedezni, hogy az egyes delfin- és cetfajtáknál az agy relatív nagysága még nagyobb. Most úgy tartják, hogy az emberi agy tekintélyes nagyságát hűtőrendszerének fejlettsége teszi lehetővé.
[7] Sokan Alexander Hoodot (Hood, 1824) idézik, mint lehetséges elsődleges forrást.
[8] Golgi kitartott abban a hiedelmében, hogy az agyi funkciók egy folytonos közegben valósulnak meg, amibe a neuronok be vannak ágyazva. Ezzel ellentétben Cajal a „neurondoktrínát” képviselte. 1906-ban mindketten (megosztott) Nobel-díjban részesültek, de a díj átvételekor meglehetősen ellenséges hangnemű beszédeket tartottak.

Created by XMLmind XSL-FO Converter.

drasztikus változásokra, ráadásul egyes állatoknál úgy tűnik, többszörös leképezéseket találhatunk. Azt azonban még nem teljesen értjük, hogy az egyes területek miként veszik át más részek funkcióit, ha azok megsérülnek. Az egyedi emlékek tárolásáról kis híján semmilyen elméletünk nincs.

Az ép agy aktivitásának mérése 1929-ben kezdődött, amikor Hans Berger kidolgozta az elektroencefalográfia (EEG) elvét. A funkcionális mágneses rezonancia (fMRI) közelmúltbeli kifejlesztése (Ogawa és társai, 1990) eddig nem látott részletességgel tárja az idegsejtkutatók elé az agy aktivitását, olyan méréséket is lehetővé téve, amelyek érdekes módon korrelálnak a zajló kognitív folyamatokkal. Az ismereteket kiegészíti az egyedi sejt szintjén történő neuronaktivitás rögzítésének fejlődése. A nagymértékű fejlődés ellenére még messze vagyunk attól, hogy megértsük, hogy e kognitív folyamatok bármelyike valójában hogyan is működik.

1.3. ábra - A számítógépek számítási erőforrásainak (2003 körül) és az agy erőforrásainak durva összehasonlítása. A könyv első kiadása óta a számítógépekre jellemző számok legalább egy 10-es szorzóval megnőttek, és várható, hogy ez a növekedés ebben az évtizedben is folytatódik. Az agyra jellemző számok az utóbbi 10 000 évben nem változtak.

	Számítógép	Emberi agy
Számítási egységek	1 CPU, 10^8 kapu	10^{11} neuron
Tárolóegységek	10^{10} bit RAM	10^{11} neuron
	10^{11} bit diszk	10^{14} szinapszis
Ciklusidő	10^{-9} mp	10^{-3} mp
Sávszélesség	10^{10} bit/mp	10^{14} bit/mp
Memóriamódosítások száma másodpercenként	10^9	10^{14}

Fontos

Az igazán csodálatos észrevétel az, hogy *egyszerű sejtek gyűjteménye elvezethet a gondolathoz, a cselekvéshez és a tudathoz,* vagy más szavakkal: az *agytól eljutunk az elméhez* (Searle, 1992). Az egyedüli elvi alternatíva a miszticizmus: hogy létezik egy misztikus világ, ahol megvalósul az elme a fizikai tudományok hatáskörén túl.

Fontos

Az agy és a számítógép igen eltérő feladatokat végeznek és eltérő tulajdonságokkal rendelkeznek. Az 1.3. ábra azt mutatja, hogy az emberi agyban tipikusan 1000-szer több neuron van, mint ahány kapu található egy korszerű számítógép processzorában. A Moore-törvény[9] alapján meg lehet jósolni, hogy egy processzorban lévő logikai kapuk száma az agyi neuronok számát 2020 körül éri el. Az ilyen jóslatokból persze vajmi kevés következtetésre lehet jutni. Ráadásul a tárolókapacitásbeli különbség elenyésző a kapcsolási sebességben és a párhuzamosságban mutatkozó különbségekhez képest. Egy számítógépes chip nanomásodpercek alatt hajt végre egy utasítást, a neuronok ennél milliószor lassabbak. Az agy ennél többet is kompenzál. Minden neuron és szinapszis egyidejűleg aktív, holott a legkorszerűbb számítógépeknek is csak egy vagy legfeljebb néhány processzora van. Így *annak ellenére, hogy a számítógép a nyers kapcsolási sebességben milliószor gyorsabb, az agy végeredményben százezerszer gyorsabban oldja meg a feladatait.*

2.5. Pszichológia (1879-től napjainkig)

- Hogyan gondolkoznak és cselekszenek az emberek és az állatok?

Azt mondhatjuk, hogy a tudományos pszichológia Hermann von Helmholtz német fizikus (1821–1894) és tanítványa, Wilhelm Wundt (1832–1920) munkásságával kezdődött. Helmholtz tudományos módszereket

[9] A Moore-törvény azt állítja, hogy az egy négyzetcentiméterre eső tranzisztorok száma minden 1–1,5 évben megduplázódik. Az emberi agy kapacitása durván 2–4 millió évenként duplázódik.

Created by XMLmind XSL-FO Converter.

alkalmazott az emberi látás tanulmányozásánál, és a *Handbook of Physiological Optics* c. művét még ma is úgy tekintik, mint „az emberi látás fizikájának és fiziológiának máig egyedüli legfontosabb tanulmányát” (Nalwa, 1993, 15. o.). Wundt 1879-ben a Lipcsei Egyetemen nyitotta meg a kísérleti pszichológia első laboratóriumát. Wundt ragaszkodott azon kísérletek szigorú ellenőrzéséhez, amelyekben munkatársai érzékelési, illetve asszociációs feladatokat végeztek, hogy közben a kísérletet végzők gondolkodási folyamatait is meg lehessen figyelni. A gondos felügyelet révén a pszichológia inkább tudománnyá vált, de az adatok szubjektív volta miatt valószínűtlen volt, hogy egy kísérletező képes legyen valaha is a saját elméleteit megcáfolni. Ezzel szemben az állati viselkedést tanulmányozó biológusok nem önmagukra vonatkozó adatokkal dolgoztak, és egy objektív módszertant dolgoztak ki, amit H. S. Jenkins (Jenkins, 1906) a *Behavior of Lower Organisms* c. nagy hatású műve mutat be. Ezt a nézőpontot az emberekre kivetítve a John Watson (1878–1958) vezette **behaviorista** mozgalom (**behaviorism**) a mentális folyamatok *minden* elméletét elutasította, azzal érvelve, hogy önelemzésből lehetetlen megbízható bizonyítékot szerezni. A behavioristák ragaszkodtak ahhoz, hogy az állatot érő érzést (vagy *ingert*) és az eredményül kapott cselekvést (másképpen *választ*) szigorúan objektív mércék szerint tanulmányozzuk. Olyan gondolati konstrukciókat, mint a tudást, a hiedelmeket, a célokat és a következtetés lépéseit elutasították, áltudományos, „népi pszichológiának” tartva azokat. A behaviorizmus eredményesnek bizonyult a patkányoknál és a galamboknál, de az emberek megértésében kevésbé volt sikeres. A pszichológiára azonban – különösképpen 1920 és 1960 között az Egyesült Államokban – erős hatást gyakorolt.

Az a **kognitív pszichológiá**t (**cognitive psychology**) jellemző alapvető nézet, miszerint az agy egy információfeldolgozó eszköz, egészen William James (1842–1910)[10] munkáihoz vezethető vissza. Helmholtz is ragaszkodott ahhoz az elképzeléshez, hogy az érzékelésben egy tudat alatti logikai következtetésnek is van szerepe. A kognitív álláspontot az Egyesült Államokban háttérbe szorította a behaviorizmus, de a Frederic Bartlett (1886–1969) vezette Cambridge-i Alkalmazott Pszichológiai Intézetben (Cambridge’s Applied Psychology Unit) a kognitív modellezés virágozhatott tovább. Kenneth Craiknek, aki Bartlett hallgatója, majd utódja volt, a *The Nature of Explanation* c. műve (Craik, 1943) határozottan visszaállította az olyan „mentális” fogalmak legitimációját, mint a hiedelmek és a célok, azzal érvelve, hogy ezek ugyanúgy tudományos fogalmak, mint a nyomás vagy a hőmérséklet, amikor gázokról beszélünk, miközben tudjuk, hogy a gázok molekulákból állnak, és azoknak sem nyomásuk, sem hőmérsékletük nincs. Craik megfogalmazta a tudásalapú ágens működésének három kulcsfontosságú lépését: (1) az ingert egy belső reprezentációra le kell fordítani, (2) e reprezentációt kognitív folyamatok manipulálják, és új belső reprezentációkat származtatnak belőle, amelyek (3) ismét cselekvésre fordítódnak vissza. Craik világosan megmagyarázta, hogy ez a megközelítés miért jó elgondolása egy ágensnek:

> Abban az esetben, ha egy szervezet magában hordja a külső valóság és a saját lehetséges cselekvéseinek „kisméretű modelljét”, képes különféle alternatívákat kipróbálni, a számára legjobb mellett dönteni, a jövőbeli helyzetekre azok bekövetkezése előtt reagálni, a múltbeli események ismeretét a jelen és a jövő kezelésében felhasználni, és a felmerülő szükséghelyzetekre minden vonatkozásban kimerítőbb, biztonságosabb és kompetensebb módon reagálni.
>
> —(Craik, 1943)

Miután Craik 1945-ben egy biciklibalesetben meghalt, munkáját Donald Broadbent folytatta. Műve, a *Perception and Communication* (Broadbent, 1958) a pszichológiai jelenségek néhány első információfeldolgozó modelljét írja le. Közben, az Egyesült Államokban, a számítógépes modellezés fejlődése elvezetett a **kognitív tudomány** (**cognitive science**) kialakulásához. Ez a terület mondhatni egy tudományos találkozóval indult 1956 szeptemberében az MIT-n. (Látni fogjuk, hogy ez éppen két hónappal később követte azt a konferenciát, ahol az MI „megszületett”.) A találkozón George Miller a *The Magic Number Seven,* Noam Chomsky a *Three Models of Language*, Allen Newell és Herbert Simon pedig a *The Logic Theory Machine* c. előadásokat tartották. Ez a három befolyásos előadás azt mutatta, hogy a számítógépes modelleket hogyan lehet bevetni a memória, a nyelv és a logikai gondolkodás pszichológiájába. Manapság a pszichológusok közt teljesen elfogadott, hogy „a kognitív elméletnek olyannak kell lennie, mint egy számítógépes programnak” (Anderson, 1980). Ezen azt értik, hogy az elméletnek egy kognitív funkció megvalósításához az információfeldolgozó mechanizmust részletesen le kell írnia.

2.6. Számítógépes tudományok (1940-től napjainkig)

- Hogyan lehet hatékony számítógépet építeni?

[10] William James Henry James író testvére volt. Azt mondják, hogy Henry úgy írt irodalmat, mintha pszichológia lenne, William viszont a pszichológiát írta úgy, mintha irodalom lenne.

Created by XMLmind XSL-FO Converter.

A mesterséges intelligencia sikeréhez két dolog szükséges: intelligencia és valamilyen mesterségesen létrehozott termék, „műtermék”. A választott műtermék a számítógép lett. A korszerű digitális számítógépet, majdnem egy időben és egymástól függetlenül, három, a II. világháborúban harcoló országban találták fel. Az első *működőképes* számítógép az elektromechanikus Heath Robinson[11] volt, amit Alan Turing munkacsoportja 1940-ben épített azzal a kizárólagos céllal, hogy a német üzeneteket dekódolhassák. 1943-ban ugyanez a csoport fejlesztette ki a Colossust, egy elektroncsöves, nagy teljesítményű, általános célú gépet.[12] Az első működőképes *programozható* számítógép a Z–3, a német Konrad Zuse 1941-ből származó találmánya volt. A lebegőpontos számokat is Zuse találta ki, és ő volt az, aki az első magas szintű programozási nyelvet, a Plankalkült kifejlesztette. Az Egyesült Államokban az első *elektronikus* számítógépet, az ABC-t, John Atanasoff és végzős hallgatója, Clifford Berry állították össze 1940 és 1942 között az Iowai Állami Egyetemen. Munkájukat nemigen támogatták, elismerést sem kaptak érte. A mai számítógép előfutárai közül a legnagyobb hatású gép az ENIAC volt, amelyet egy titkos katonai projekt részeként John Mauchly és John Eckert részvételével a Pennsylvaniai Egyetem egy csoportja fejlesztett ki.

Az azóta eltelt fél évszázad során a számítógéphardver minden újabb generációja sebességben és kapacitásban növekedést, árban esést hozott. A teljesítőképesség mintegy 18 havonta megduplázódik, és ez így mehet még egy vagy két évtizedig. Utána molukuláris gépekre vagy valamilyen más technológiára lesz szükségünk.

Természetesen voltak számítóberendezések az elektronikus számítógépek előtt is. A 17. századból származó legkorábbi automatizált gépekről már a filozófiánál volt szó. Az első *programozható* gép Joseph Marie Jacquard 1805-ben feltalált szövőszéke volt, amely lyukasztott kártyákon tárolta a szövőmintához szükséges utasításokat. A 19. század közepe táján Charles Babbage (1792–1871) két gépet tervezett, de egyiket sem fejezte be. A könyvünk borítóján is látható Difference Engine rendeltetése matematikai táblázatok számítása lett volna mérnöki és tudományos feladatokhoz. Végül is ezt a gépet 1991-ben megépítették, és a londoni Science Museumban működés közben bemutatták (Swade, 1993). A Babbage-féle Analitikus Gép (Analytical Engine) sokkal ambiciózusabb terv volt. Címezhető memóriájával, tárolt programjával és feltételes ugrásaival ez volt az első, univerzális számításokra alkalmas műtermék. Ada Lovelace, Babbage kolléganője, Lord Byron költő lánya volt talán a világ első programozója. (Az Ada programozási nyelv róla kapta a nevét.) Ada programokat írt a befejezetlen Analitikus Gépre, sőt azon is spekulált, hogy a gép sakkjátékra vagy zenekomponálásra is képes lehet.

Az MI a számítógép-tudományok szoftveroldalának is adósa, hiszen ez szolgáltatja az operációs rendszereket, a programozási nyelveket és a korszerű programok írásához szükséges eszközöket (és a róluk szóló cikkeket). Ez azonban egy olyan terület, ahol az adósságot törlesztették is. Az MI-kutatásokban megjelent sok ötlet visszakerült a számítógépes tudományok „fő áramlatába”, beleértve az időosztásos (time-sharing) operációs rendszereket, az interaktív értelmezőket, a személyi számítógépeket ablakokkal és egérrel, a gyors fejlesztést lehetővé tevő fejlesztői környezeteket, a láncolt listás adatszerkezeteket, az automatikus tárolókezelést és az objektumorientált, szimbolikus, funkcionális és dinamikus programozás kulcsfontosságú fogalmait.

2.7. Irányításelmélet és kibernetika (1948-tól napjainkig)

- Hogyan működhet egy műtermék a saját irányítása mellett?

Az első önszabályozó gépezetet – egy vízi órát, amely olyan szabályozóval volt ellátva, hogy a víz átfolyását konstans, megjósolható értéken tartotta – az alexandriai Ktesibios (i. e. kb. 250) építette. Ez a találmány megváltoztatta annak definícióját, hogy egy műtermék mire képes. Korábban csak az élő dolgokról tartották, hogy képesek a viselkedésüket a környezeti változások hatására módosítani. Az önszabályozó visszacsatolt szabályozó rendszerek további példái a James Watt (1736–1819) építette gőzgépszabályozó vagy a Cornelis Drebbel (1572–1633) által feltalált termosztát. Cornelius Drebbel volt az is, aki a tengeralattjárót feltalálta. A stabil visszacsatolt rendszerek elméletét a 19. században fejlesztették ki.

A ma **irányításelmélet**nek (**control theory**) nevezett terület keletkezésében Norbert Wiener (1894–1964) játszott központi szerepet. Wiener zseniális matematikus volt, aki többek közt Bertrand Russell-lel is együttműködött, mielőtt az érdeklődését a biológiai és mechanikai szabályozó rendszereknek és ezek a kognitív mechanizmusokkal való kapcsolatának szentelte. Craikhoz hasonlóan (aki szintén használt szabályozási rendszereket mint pszichológiai modelleket), Wiener és kollégái, Arturo Rosenblueth és Julian Bigelow kihívást

[11] Heath Robinson karikatúrarajzoló volt, aki a mindennapi feladatokhoz, mint például a vajaskenyér-kenéshez, kitalált szeszélyes és abszurd módon bonyolult szerkezeteiről volt híres.
[12] A háború után Turing szerette volna ezeket a számítógépeket az MI-kutatáshoz – például az egyik legelső sakkprogramhoz (Turing és társai, 1953) – felhasználni. Erőfeszítéseinek azonban a brit kormány vetett gátat.

Created by XMLmind XSL-FO Converter.

jelentettek a behaviorista ortodoxia számára (Rosenblueth és társai, 1943). Nézetük szerint a célorientált viselkedés a „hibát” – az aktuális állapot és a célállapot közötti különbséget – minimalizáló szabályozó mechanizmusból fejlődik ki. Az 1940-es évek végén Wiener, Warren McCulloch-val, Walter Pittsszel és Neumann Jánossal együtt egy sor konferenciát szervezett, ahol a kognitív folyamatok új matematikai és számítási modelljeivel foglalkoztak, és a behaviorista tudományok területén sok tudósra voltak hatással. Wiener *Cybernetics* c. könyve (Wiener, 1948) bestseller lett, és ráébresztette a széles publikumot az intelligens mesterséges gépek lehetőségére.

A korszerű irányításelmélet, és különösen ennek a *sztochasztikus optimális szabályozás* nevű ága olyan rendszerek kifejlesztését tűzte ki célul, amelyek egy **célfüggvény**t (**objective function**) maximalizálnak az időben. Ez nagyjából megegyezik az MI-ről alkotott képünkkel: optimálisan viselkedő rendszereket fejleszteni. Akkor miért lett az MI és az irányításelmélet két különböző kutatási terület, hiszen megalkotói még szoros kapcsolatban is álltak egymással? A válasz forrása az a szoros kapcsolat, amely a résztvevők által ismert matematikai technikák és az ezeknek megfelelő, mindegyik világképben megjelenő problémák halmaza között alakult ki. Az irányításelmélet eszközei, a mátrixalgebra és az analízis folytonos változók rögzített halmazai által leírható rendszerekhez vezettek. Az egzakt analízis ráadásul tipikusan csak *lineáris* rendszerek esetében lehetséges. Az MI-t részben azért alkották meg, hogy az irányításelmélet 1950-es évekbeli matematikájától meg tudjanak szabadulni. A logikai következtetés és a számítás eszközei lehetővé tették az MI-kutatók számára, hogy olyan problémákat is figyelembe vegyenek, mint a nyelv, a látás és a tervkészítés, amelyek az irányításelméleti célkitűzéseken teljesen kívül estek.

2.8. Nyelvészet (1957-től napjainkig)

- Mi a nyelv és a gondolat kapcsolata?

B. F. Skinner 1957-ben publikálta a *Verbal Behavior* c. művét. A könyv e terület egyik legjobb szakértőjének tollából a nyelvtanulás behaviorista megközelítésének átfogó és részletes tárgyalását adta. Furcsamód a könyvismertető ugyanolyan híres lett, mint maga a könyv, és a behaviorizmus iránti érdeklődést csaknem kiirtotta. Az ismertető írója Noam Chomsky volt, aki épp akkor publikálta a saját elméletéről szóló *Syntactic Structures* c. könyvét. Chomsky kimutatta, hogy a behaviorista elmélet nem kezeli a nyelvben meglevő kreativitást. Az elmélet nem tudta megmagyarázni, hogy egy gyerek miként képes olyan mondatokat megérteni vagy megformálni, amilyeneket korábban soha nem hallott. Chomskynak, Pánini (kb. i. e. 350) indiai nyelvészig viszszatekintő, szintaktikai modellekre alapozó elmélete viszont magyarázatot tudott adni erre, és a korábbi elméletekkel ellentétben kellően formális volt ahhoz, hogy legalább elvben programozható is legyen.

A modern nyelvészet és az MI nagyjából ugyanabban az időben „született meg” és együtt fejlődött, a nyelv használatára összpontosító **számítógépes nyelvészet**nek (**computational linguistics**) vagy **természetes nyelvfeldolgozás**nak (**natural language processing**) nevezett hibrid területen találkozva egymással. A nyelv megértésének problémája rövidesen sokkal bonyolultabbnak bizonyult, mint ahogy ez 1957-ben látszott. A nyelv megértéséhez meg kell érteni a témát és a kontextust is, nem elegendő ehhez a mondat struktúrájának a megértése. Ez persze triviálisnak tűnhet, azonban az 1960-as évekig mégsem volt általánosan elfogadott. A **tudásreprezentáció**hoz (**knowledge representation**) – amely annak a kutatása, hogy a tudást hogyan fejezzük ki a számítógép által feldolgozható formában – tartozó kezdeti kutatások zöme a nyelvhez kötődött és a nyelvészeti kutatásokból táplálkozott, azok viszont a nyelv évtizedes filozófiai elemzéseivel voltak kapcsolatban.

3. A mesterséges intelligencia története

A bevezető anyaggal a hátunk mögött most már készek vagyunk arra, hogy felvázoljuk a szó szoros értelmében vett mesterséges intelligencia fejlődését.

3.1. A mesterséges intelligencia érlelődése (1943–1955)

Az első olyan eredményt, amit ma általánosan MI-eredménynek ismernek el, Warren McCulloch és Walter Pitts érte el (McCulloch és Pitts, 1943). Három forrásból merítettek: az alapszintű fiziológiai és az agyi neuronok működésére vonatkozó ismeretekből, az ítéletkalkulus Russell és Whitehead-féle formális elemzéséből és Turing számításelméletéből. Egy mesterséges neuron modellt javasoltak, ahol minden neuron vagy „bekapcsolt”, vagy „kikapcsolt” állapotban lehet, és ahol az átkapcsolás „be” állapotba akkor történik, amikor a neuront kellő számú szomszédos neuron stimulálja. A neuron állapotáról azt tartották, hogy „ténylegesen azzal a logikai állítással ekvivalens, amely a megfelelő ingert kiváltotta”. Kimutatták például, hogy összekapcsolt

Created by XMLmind XSL-FO Converter.

neuronok valamilyen hálózatával minden kiszámítható függvény előállítható, és hogy egyszerű hálóstruktúrákkal az összes logikai műveletet (ÉS, VAGY, NEM stb.) is elő lehet állítani. McCulloch és Pitts azt is felvetette, hogy egy megfelelően kialakított háló képes lehet tanulni is. Donald Hebb egy olyan egyszerű értékfrissítő szabályt mutatott be a neuronok közötti összeköttetések erősségének módosítására, amely lehetővé teszi a tanulást (Hebb, 1949). Tanulási szabálya, amit **Hebb-tanulás**nak (**Hebbian learning**) nevezünk, máig érvényes hatású modellnek bizonyult.

1951-ben a Princeton Egyetem matematika tanszékén két végzős hallgató – Marvin Minsky és Dean Edmonds – megépítette az első neurális számítógépet. A Snarc-nak elnevezett gépben 3000 elektroncső és a B–24 bombázó automatapilóta mechanizmusa egy 40 neuronból álló hálózatot szimulált. Minsky PhD-bizottsága szkeptikus volt, vajon egy ilyen munkát matematikának lehet-e nevezni, de Neumann János (aki tagja volt a bizottságnak) állítólag úgy nyilatkozott, hogy „ha ez nem is matematika most, valamikor az lesz”. Minsky volt később az, aki nagy hatású tételeivel kimutatta a neuronhálós kutatás korlátait.

Sok kezdeti eredményt lehetne MI-nek nevezni, azonban egy teljes elképzelést az MI-ről 1950-ben Alan Turing fogalmazott meg a *Computing Machinery and Intelligence* c. cikkében. Itt vezette be a Turing-teszt, a gépi tanulás, a genetikus algoritmusok és a megerősítéses tanulás fogalmakat.

3.2. A mesterséges intelligencia megszületése (1956)

Princeton volt az otthona az MI egy másik befolyásos személyiségének, John McCarthynak. Az egyetem befejezése után McCarthy a Dartmouth College-ba került, ami a témakör hivatalos szülőhelye lett. McCarthy meggyőzte Minskyt, Claude Shannont és Nathaniel Rochestert, hogy segítsenek neki azokat az amerikai kutatókat összehozni, akik érdekeltek az automataelméletben, a neurális hálókban és az intelligencia kutatásában. 1956 nyarán egy két hónapos munkatalálkozót szerveztek Dartmouthban. Összesen tíz résztvevő gyűlt össze, beleértve Trenchard More-t Princetonból, Arthur Samuelt az IBM-től, valamint Ray Solomonoffot és Oliver Selfridge-et az MIT-ből.

A pálmát a Carnegie Tech.[13] két kutatója, Allen Newell és Herbert Simon vitte el. Bár másoknak is voltak ötletei, és néhány esetben konkrét alkalmazásra – dámajátékra – voltak programjaik is, Newell és Simon már egy következtető programmal, a Logic Theorist (LT)-vel rendelkezett.[14] Erről Simon azt állította, hogy „egy olyan programot találtunk fel, amely képes nemnumerikusan gondolkodni, és ezzel meg is oldottuk a tiszteletre méltó anyag-szellem viszony problémáját”. Nem sokkal a munkatalálkozó után a program képes volt bebizonyítani a Russell és Whitehead *Principia Mathematica* c. művének 2. fejezetében foglalt tételek többségét. Azt mondják, Russell el volt ragadtatva, amikor Simon megmutatta neki, hogy az egyik tétel esetén a program rövidebb bizonyítással állt elő, mint amit a *Principiá*ban közöltek. A *Journal of Symbolic Logic* szerkesztőit kevésbé hatotta meg a dolog. A Newell, Simon és a Logic Theorist szerzői hármastól származó cikket elutasították.

A dartmouthi munkatalálkozó új áttöréshez ugyan nem vezetett, de a fontos személyiségek bemutatkoztak egymásnak. A következő húsz évben ők, továbbá hallgatóik és kollégáik az MIT-n, a CMU-n, a Stanfordon és az IBM-nél lesznek azok, akik meghatározó szerepet töltenek be az MI területén. A munkatalálkozó talán legtartósabb eredménye az volt, hogy elfogadták a terület McCarthy által kreált új nevét, azaz a **mesterséges intelligenciá**t (**artificial intelligence**). A „számítási racionalitás” talán jobb név lett volna, de az „MI” név azóta is megmaradt.

Ha bepillantunk a dartmouthi munkatalálkozót javasló anyagba (McCarthy és társai, 1955), látjuk, miért volt szükségszerű, hogy az MI egy külön területté váljon. De miért nem lehetett az MI-kutatást az irányításelmélet, az operációkutatás vagy a döntéselmélet keretein belül tartani, amikor ráadásul ezek célkitűzései nagyon hasonlók az MI célkitűzéseihez? Vagy az MI miért nem lett a matematika egyik ága? Az első válasz az, hogy az MI a kezdetek óta sajátjának tekintette az olyan emberi képességek duplikálását, mint a kreativitás, az önfejlesztés és a nyelv használata. Ezekkel a kérdésekkel semmilyen más terület nem foglalkozott. A másik válasz a módszertanban rejlik. Az említett területek közül tisztán csak az MI tekinthető a számítógépes tudományok egy ágának (bár az operációkutatás szintén súlyt helyez a számítógépes szimulációkra). Az MI az egyetlen olyan terület, ahol bonyolult, változó környezetben autonóm módon működő gépek építése a cél.

[13] Most Carnegie Mellon Egyetem (Carnegie Mellon University, CMU).

[14] Az LT megírásához Newell és Simon az IPL-t, a listakezelő nyelvet is feltalálta. Nem rendelkezvén fordítóval, programjukat kézzel fordították le gépi kódra. Hogy a hibákat elkerüljék, párhuzamosan dolgoztak, és minden utasítás megírásánál a bináris számokat egymásnak átkiabálták, hogy így biztosra menjenek.

Created by XMLmind XSL-FO Converter.

3.3. Korai lelkesedés, nagy elvárások (1952–1969)

Az MI korai évei – bizonyos kereteken belül – bővelkedtek a sikerekben. Ha figyelembe vesszük azoknak az időknek a primitív számítógépeit és programozási eszközeit, továbbá azt, hogy még néhány évvel korábban is csupán aritmetikai feladatok elvégzésére tartották alkalmasnak a számítógépet, megdöbbentő volt, hogy a számítógép akár csak távolról is okosnak tűnő dologra lehet képes. Értelmiségi körökben, összességében, inkább azt szerették volna hinni, hogy „a gép *X*-re soha nem lesz képes” (az *X*-ek, Turing által kigyűjtött hosszú listája a 26. fejezetben található). Az MI kutatói természetesen erre azzal válaszoltak, hogy egymás után demonstrálták az *X*-eket. A modern MI-kutatók közül néhányan úgy említik ezt az időszakot, mint a „Nézze uram, biz’ isten, magától megy!” idejét.

Newell és Simon kezdeti sikerét az általános problémamegoldó program, a General Problem Solver, GPS követte. A Logic Theoristtal ellentétben ezt a programot eleve úgy tervezték, hogy az emberi problémamegoldás protokolljait imitálja. Az derült ki, hogy a program által kezelhető feladványok osztályán belül, a részcélok és a lehetséges cselekvések megfontolásának sorrendje tényleg hasonlított ahhoz, mint ahogy a hasonló problémákon dolgozó emberek cselekszenek. Így, a GPS volt talán az első, az „emberi módon gondolkodni” megközelítést megtestesítő program. A GPS és az azt követő programok sikere arra késztette Newellt és Simont (Newell és Simon, 1976), hogy megfogalmazzák híres **fizikai szimbólumrendszer** hipotézisüket (**physical symbol system**), amely azt állítja, hogy „a fizikai szimbólumrendszerek az általános intelligens cselekvés szükséges és elégséges eszközeivel rendelkeznek”. Arra gondoltak, hogy minden, intelligenciát felmutató rendszernek (legyen az ember vagy gép) képesnek kell lennie arra, hogy szimbólumokból álló adatstruktúrákat manipuláljon. Később látni fogjuk, hogy e hipotézist több irányból is megtámadták.

Az első néhány MI-programot az IBM-nél Nathaniel Rochester és kollégái fejlesztették ki. Herbert Gelernter egy olyan geometriai tételbizonyító programot (Geometry Theorem Prover, Gelernter, 1959) írt, mely sok matematikus hallgató által trükkösnek talált tételt tudott bebizonyítani. 1952-től kezdve Arthur Samuel dámajátékot játszó programokat írt, amelyek végül megtanultak egy erős amatőr versenyzői szinten játszani. Eközben sikerült megcáfolnia, hogy a számítógép csak arra képes, amire utasítják, hiszen programja gyorsan megtanult nála is jobban játszani. A program tv-bemutatása 1956 februárjában igen nagy hatást keltett. Turinghoz hasonlóan Samuelnek is csak nehezen sikerült gépidőt szereznie. Éjszaka dolgozott, az IBM számítógépes üzemében végtesztelésre váró gépeket használva. A kétszemélyes játékokkal a 6. fejezet, a Samuel által használt tanuló technikával és annak továbbfejlesztésével a 21. fejezet foglalkozik.

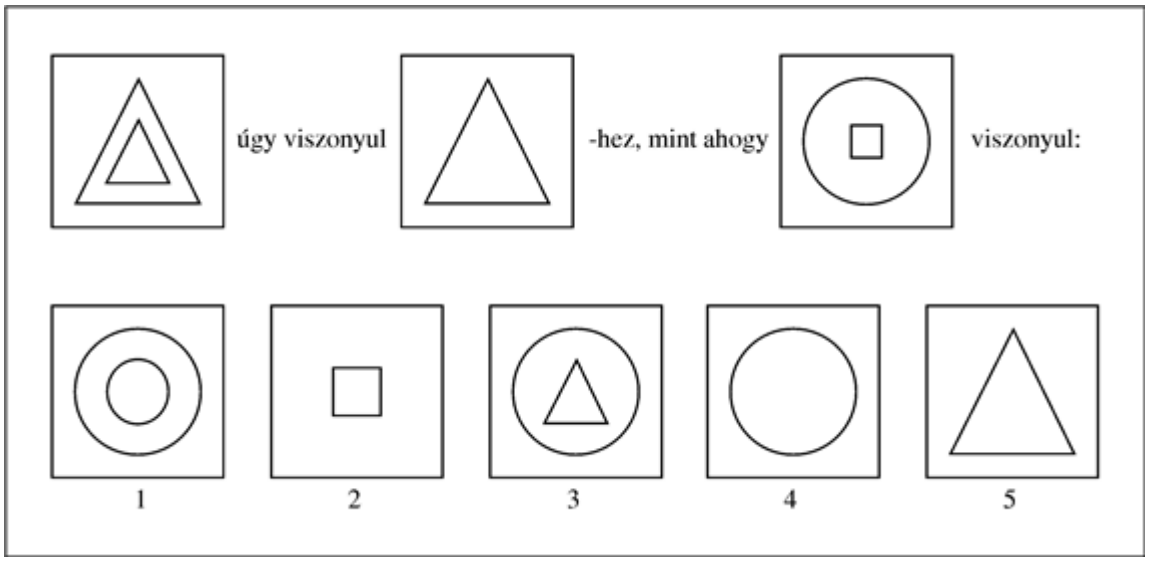
John McCarthy Dartmouth-ból átment az MIT-re, és ott egyetlen év alatt, a történelminek nevezhető 1958-as évben, három kulcsfontosságú eredményt ért el. Az 1. számú MIT AI Lab Memóban definiálta a **Lisp**et, amely elsődleges MI-programozási nyelvvé nőtte ki magát. A Lisp a második legrégebbi nyelv, amely még használatban van, a Fortrannál csak egy évvel fiatalabb. A Lisp esetén McCarthy rendelkezett már a szükséges eszközzel, de a ritka és drága számítógépes erőforrásokhoz való hozzáférés számára is komoly problémát jelentett. Így aztán az MIT-n McCarthy és mások kitalálták az időosztást. Szintén 1958-ban McCarthy *Programs with Common Sense* címen cikket publikált, amelyben az Advice Takert írta le. Ez egy hipotetikus program, amit az első teljes MI-rendszernek tekinthetünk. A Logic Theoristhez és a Geometry Theorem Proverhez hasonlóan McCarthy programja is tudást használt fel egy probléma megoldásának megtalálásához. Azonban másokkal ellentétben, ennek a programnak a világra vonatkozó általános tudással kellett rendelkeznie. McCarthy megmutatta például, hogy néhány egyszerű axióma elegendő ahhoz, hogy programja képes legyen terveket generálni arra vonatkozóan, hogyan kell a repülőtérre kimenni ahhoz, hogy a repülőgépet le ne késsük. A programot úgy tervezte, hogy képes legyen normális működés közben új axiómákat is elfogadni, és ennek eredményeként *átprogramozás nélkül* új területeken is kompetenciát mutatni. Az Advice Taker ily módon a tudásreprezentáció és a következtetés leglényegesebb elveit testesítette meg, miszerint hasznos, ha rendelkezünk a világot és az ágens cselekvéseinek eredményét leíró explicit és formális reprezentációval, és képesek vagyunk ezt a reprezentációt deduktív módon manipulálni. Figyelemre méltó, hogy 35 év múltával még mennyire releváns maradt az 1958-as cikk.

1958 volt az az év is, amikor Marvin Minsky az MIT-re ment át. Kezdeti együttműködése McCarthyval nem tartott sokáig. McCarthy a reprezentációra és a formális logikai következtetésre tette a hangsúlyt, Minskyt inkább az érdekelte, hogy a programok működőképesek legyenek, majd végül logikaellenes álláspontra helyezkedett. 1963-ban McCarthy a Stanfordon megalakította az ottani MI-labort. Kutatási programja – amely arra irányult, hogy a logikát felhasználja a legvégső Advice Taker építésében – lökést kapott, amikor J. A. Robinson felfedezte a rezolúciót, az elsőrendű logika teljes bizonyítási eljárását (lásd 9. fejezet). A Stanfordon a kutatás hangsúlyozottan a logikai következtetés általános módszereire irányult. A logika alkalmazásaihoz tartoztak Cordellnek a Green kérdését megválaszoló és tervkészítő rendszerei (Green, 1969b), továbbá Shakey

Created by XMLmind XSL-FO Converter.

robotikus projektje az új Stanfordi Kutatóintézetben (Stanford Research Institute, SRI).[15] Ez a projekt – amit részletesebben a 25. fejezetben tárgyalunk – volt az első, amely a logikai következtetést és a fizikai aktivitást teljes egészében integrálta.

1.4. ábra - Egy konkrét probléma, amelyet Evans ANALOGY programja megoldott



Minsky a hallgatók egész sorát irányította, akik a megoldásokhoz láthatóan intelligenciát igénylő, korlátos problémákkal foglalkoztak. Ezeket a korlátos problématerületeket később **mikrovilág**oknak (**microworld**s) nevezték el. James Slagle SAINT nevű programja (Slagle, 1963a) képes volt az elsőéves analízis tanfolyamra jellemző, zárt alakra hozható integrálszámítási feladatokat megoldani. Tom Evans ANALOGY programja (Evans, 1968) az IQ-tesztekben előforduló (lásd 1.4. ábra) geometriai analógia jellegű problémákat oldotta meg. Daniel Bobrow STUDENT programja (Bobrow, 1967) olyan algebrai feladványokat oldott meg, mint például az alábbi:

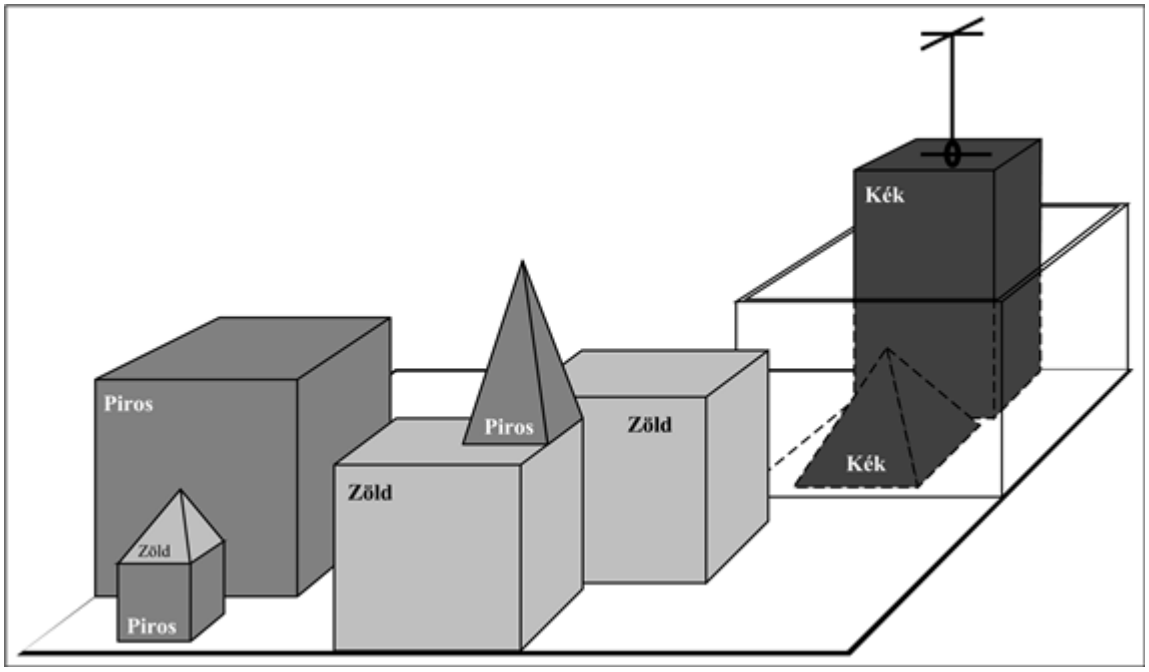
> Ha Tamás ügyfeleinek a száma kétszerese az általa közzétett hirdetések 20%-a négyzetének, és Tamás 45 hirdetést adott közzé, akkor hány ügyfele van Tamásnak?
>
> —Bobrow, 1967

A mikrovilágok legismertebbike a kockavilág lett, amely egy asztalra (vagy gyakrabban egy szimulált asztalra) helyezett tömör geometriai testekből áll (lásd 1.5. ábra). Az ilyen világban értelmezett feladat a kockák egy bizonyos átrendezése egy olyan robotkar segítségével, amely egyszerre egyetlenegy kockát képes megfogni. A kockavilág otthont adott David Huffman gépi látási projektjének (Huffman, 1971), David Waltz gépi látási és kényszerterjesztés-kutatásának (Waltz, 1975), Patrick Winston tanulási elméletének (Winston, 1970), Terry Winograd természetes nyelvfeldolgozási programjának (Winograd, 1972) és Scott Fahlman tervkészítő programjának (Fahlman, 1974).

1.5. ábra - Egy elrendezés a kockavilágban. A SHRDLU robot (Winograd, 1972) éppen sikeresen teljesítette a „keresd meg azt a blokkot, ami a kezedben tartottnál magasabb, és tedd a dobozba” utasítást.

[15] Az angol nyelvben a tervkészítő tervezés (planning) és a terméket készítő tervezés (design) szépen elkülönül. Hogy a pontos jelentést hangsúlyozzuk, az angol „planning” fordításánál a továbbiakban „tervkészítést” használunk. (*A ford.*)

Created by XMLmind XSL-FO Converter.



A McCulloch és Pitts neurális hálóin alapuló kezdeti kutatás szintén virágzott. Winograd és Cowan eredményei megmutatták, hogy a nagyszámú elem hogyan képes együttesen egy egyedi fogalmat reprezentálni, miközben növeli a párhuzamosságot és a robusztusságot (Winograd és Cowan, 1963). Hebb tanulási módszereit Bernie Widrow (Widrow és Hoff, 1960; Widrow, 1962) fejlesztette tovább, aki a hálózatait **adaline**-oknak nevezte, továbbá Frank Rosenblatt, aki a **perceptron**okat vezette be (Rosenblatt, 1962). Rosenblatt bebizonyította **perceptron konvergencia tétel**ét (**perceptron convergence theorem**), kimutatva ezzel, hogy tanulási algoritmusa képes a perceptron súlyait úgy módosítani, hogy az tetszőleges bemeneti adatokhoz illeszkedjen, feltéve, hogy ilyen illeszkedés egyáltalán lehetséges. Ezekről a kérdésekről a 20. fejezetben írunk majd.

3.4. Egy adag realitás (1966–1973)

Az MI kutatói már a kezdetekben sem voltak szégyenlősek a várható sikereiket illetően. Gyakran idézik Herbert Simonnak a következő, 1957-ből származó kijelentését:

> Sem meglepni, sem sokkolni senkit nem célom – de a legegyszerűbben összefoglalva azt mondhatom, hogy a világban léteznek ma már gondolkodó, tanuló és kreatív gépek. E képességük rohamosan fog fejlődni, és – a közeljövőben – az általuk feldolgozott problémák köre összemérhető lesz azokkal a problémákkal, amelyekkel az emberi elme eddig megküzdött.
>
> —Herbert Simon

Bár vitatható, hogy „közeljövőnek” mi tekinthető, Simon néhány más előrejelzése konkrétabb volt. Megjósolta, hogy tíz éven belül a számítógép sakkvilágbajnok lesz, és hogy a gép fontos új matematikai tételeket fog bebizonyítani. Ezek a jóslatok, ha nem is tíz, hanem inkább negyven év múltával (közelítőleg vagy teljesen), de beigazolódtak. Simon magabiztossága a korai MI-programok egyszerű példaproblémákon felmutatott sikereiből táplálkozott. Ezek a korai rendszerek azonban majdnem minden esetben szánalmasan csődöt mondtak, ha szélesebb körben vagy netán nehezebb problémákra akarták őket bevetni.

A nehézség egyik forrása az volt, hogy a korai programok az általuk kezelt problémákról sokszor kevés vagy szinte semmi tudást nem tartalmaztak, és csupán egyszerű szintaktikai manipulálással értek el sikereket. Egy tipikusnak mondható történet a korai gépi fordítással kapcsolatos. A gépi fordítást a Nemzeti Kutatási Alap (National Research Council) bőkezűen finanszírozta azért, hogy a Szputnyik 1957-es kilövését követően meggyorsítsák az orosz tudományos cikkek fordítását. Kezdetben azt vélték, hogy az angol és az orosz nyelvtanra alapozó egyszerű szintaktikai transzformációk és az elektronikus szótárra alapozó szóbehelyettesítés elegendő lesz a mondat pontos értelmének átadásához. Valójában a fordításhoz a téma általános ismerete szükséges, hogy feloldhassuk a kétértelműségeket, és a mondat jelentését megállapítsuk. „A szellem készséges, de a test gyenge” („the spirit is willing but the flesh is weak”) híres visszafordítása „a vodka jó, de a hús romlott”-ra („the vodka is good but the meat is rotten”) a tapasztalt nehézségeket világosan érzékeltette. Egy

Created by XMLmind XSL-FO Converter.

tanácsadó bizottság 1966-os jelentése azt állapította meg, hogy az „általános tudományos szöveg fordítása még nem megoldott, és a közeljövőben e téren gyors előrehaladás nem is várható”. Az egyetemi gépi fordítási projekteknek a kormány általi finanszírozását Amerikában teljesen megszüntették. Manapság a gépi fordítás egy nem tökéletes, ám széles körben alkalmazott eszköz műszaki, kereskedelmi, kormányzati és internetdokumentumok esetében.

Fontos

A másik nehézséget az jelentette, hogy sok olyan probléma, amelyeket az MI által kíséreltek megoldani, kezelhetetlen volt. A korai MI-programok többsége úgy dolgozott, hogy a problémára vonatkozó alapvető tényeket gépen reprezentálva, megoldó lépésszekvenciákat próbáltak ki, a különféle lépéskombinációkkal addig kísérletezve, amíg nem leltek rá a helyesre. A korai programok azért voltak használhatók, mert a mikrovilágok csak kevés objektumot, és ebből adódóan nagyon kevés lehetséges cselekvést és nagyon rövid megoldási sorozatokat tartalmaztak. Az NP-teljesség elméletének megfogalmazása előtt általában azt tartották, hogy a nagyobb problémákra „felskálázni” csupán gyorsabb hardver és nagyobb memória kérdése. A rezolúciós tételbizonyítás kifejlesztését kísérő optimizmus például hamarosan lelohadt, amikor a néhány tucat ténynél többet igénylő tételeket nem sikerült bebizonyítani. *Az a tény, hogy egy program egy megoldás megtalálására elvben alkalmas, nem jelenti azt, hogy a program bármi olyan mechanizmust is tartalmaz, amely a megoldás gyakorlati megvalósításához szükséges*.

A korlátlan számítási kapacitás illúziója nem csak a problémamegoldó programokra korlátozódott. A **gépi evolúció** (**machine evolution**), amelyet most **genetikus algoritmus**oknak (**genetic algorithm**s) nevezünk (Friedberg, 1958; Friedberg és társai, 1959), területén végzett korai kísérletek azon a kétségtelenül helyes feltevésen alapultak, amely szerint ha egy gépi kódú programot megfelelően kicsi mutációk révén változtatunk, tetszőleges, egyszerű feladatot jól megoldó programhoz juthatunk el. Az ötlet tehát az volt, hogy véletlen mutációkkal próbálkozva, a program viselkedését javító mutációkat tartsuk meg. Azonban ezernyi óra gépidő ellenére sem sikerült szinte semmilyen előrehaladást kimutatni. A korszerű genetikus algoritmusok jobb reprezentációkat használnak, és több sikerre is vitték.

Az volt az MI ellen irányuló legfontosabb kritika a Lighthill-tanulmányban (Lighthill, 1973), hogy nem képes leküzdeni a „kombinatorikus robbanást”. A tanulmány alapján a brit kormány, kettő kivételével az összes egyetemen minden MI-kutatási támogatást visszavont (a szóbeszéd kissé más és színesebb képet fest a nyomdafestéket nem tűrő politikai ambíciókról és a személyes ellenségeskedésről).

A harmadik nehézség forrását az intelligens viselkedés generálásához használt alapvető struktúrák fundamentális korlátai jelentették. Minsky és Papert *Perceptrons* c. könyve (Minsky és Papert, 1969) például azt bizonyította be, hogy bár a perceptron (a neurális háló egy egyszerű formája) megtanulhat mindent, amit képes reprezentálni, vajmi keveset képes reprezentálni. Így például a két bemenetű perceptront nem lehet megtanítani arra, hogy a bemeneteinek különbözőségét felismerje. És bár a szerzők eredményei bonyolultabb, többrétegű hálókra nem vonatkoztak, a neurális hálók kutatásának finanszírozása rövidesen majdnem nullára esett vissza. A sors iróniája, hogy a többrétegű neurális hálók későbbi, az 1980-as években történő óriási feltámadását hozó új visszaterjesztéses tanuló algoritmust éppen 1969-ben fedezték fel először (Bryson és Ho, 1969).

3.5. Tudásalapú rendszerek: a hatalom kulcsa? (1969–1979)

A problémamegoldásnak az a képe, amely az MI-kutatás első évtizedében alakult ki, egy olyan általános célú kereső mechanizmus volt, amely a teljes megoldás megtalálásának érdekében szekvenciába fűzte az elemi következtetési lépéseket. Az ilyen megközelítéseket **gyenge módszer**eknek (**weak method**s) nevezték, mert annak ellenére, hogy általánosak, a problémák nagy vagy nehéz példányaira nem skálázhatók fel. A gyenge módszerek alternatívája az erőteljesebb, területspecifikus tudás használata, amely lehetővé teszi a nagyobb granuláltságú következtetési lépések megvalósítását, és szűkebb szakértői tárgyterületeken a tipikus konkrét problémák megoldását. Ahhoz, hogy egy nehéz problémát megoldjunk, mondhatni majdnem kész válasszal kellene rendelkeznünk.

E megközelítés egyik korai példája a `DENDRAL` program volt (Buchanan és társai, 1969). A programot a Stanfordon fejlesztették ki, ahol Ed Feigenbaum (Herbert Simon volt hallgatója), Bruce Buchanan (a számítógépes szakemberből lett filozófus) és Joshua Lederberg (Nobel-díjas genetikus) összefogtak, hogy a tömegspektrométer által szolgáltatott adatokból a molekuláris struktúra kinyerésének problémáját megoldják. A program bemeneti adatai a molekula alapképlete (például $C_6H_{13}NO_2$) és a tömegspektrum voltak. A spektrum

Created by XMLmind XSL-FO Converter.

megadta a molekula bizonyos részeinek a tömegét, amikor a molekulát elektronsugárral bombázták. A tömegspektrum tartalmazhatott például m = 15-nél egy csúcsot, amit a metil (CH_3) molekularésszel lehetett azonosítani.

A program naiv verziója a molekula képletével konzisztens minden lehetséges struktúrát előállított. Ezt követően minden egyes struktúrához megjósolta a megfelelő megfigyelhető tömegspektrumot, és ezt hasonlította össze az aktuálisan megfigyelt spektrummal. Ahogy várható volt, nagyobb molekulák esetén az eljárás gyorsan kezelhetetlenné vált. A DENDRAL kutatói analitikus vegyészekhez fordultak segítségért. Azt találták, hogy a vegyészek a spektrumban a molekulában található elterjedt részstruktúrákra utaló, jól ismert csúcsmintákat keresik. Így például a keton (C = O) alcsoport (amely 28 súlyú) felismeréséhez az alábbi szabály volt használatos:

ha két olyan csúcs, x_1 és x_2 létezik, hogy

a. $x_1 + x_2 = M + 28$ (M a teljes molekula tömege);

b. $x_1 - 28$ egy magas csúcs;

c. $x_2 - 28$ egy magas csúcs;

d. x_1 és x_2 közül legalább egy csúcs magas,

akkor ketoncsoport van jelen.

Azzal, hogy felismerjük, hogy egy konkrét részstruktúra a molekula része, a lehetséges struktúrajelöltek száma nagyon nagy mértékben csökken. A DENDRAL-rendszer hatékony volt, mert:

> Az ilyen problémák megoldásához szükséges összes elméleti tudást sikerült (a rendszer spektrumjósló komponensében) leképezni az általános formáról („elsődleges ismeretek”) egy hatékony speciális formára („szakácskönyv”).
>
> —(Feigenbaum és társai, 1971)

A DENDRAL fontossága abban rejlik, hogy vitathatatlanul ez volt az első sikeres *tudásintenzív* rendszer. Szakértelmét a nagyszámú speciális rendeltetésű szabály biztosította. A későbbi rendszerekben szintén megjelent a McCarthy-féle Advice Taker egyik fő gondolata – a (szabályformájú) tudás és a következtető komponens határozott elkülönítése.

Okulva az ilyen leckén, Feigenbaum és mások a Stanfordon belekezdtek a heurisztikus programozási projektbe (Heuristic Programming Project, HPP) azzal a céllal, hogy megvizsgálják, a **szakértőrendszer**eknek (**expert system**s) ez az új módszertana milyen mértékben alkalmazható az emberi szakértelem más területein. A következő komoly erőfeszítés az orvosi diagnózis területén született meg. Feigenbaum, Buchanan és dr. Edward Shortliffe vérrel kapcsolatos fertőzések diagnosztizálására fejlesztették ki a MYCIN-rendszert. 450 szabályával a MYCIN elérte az egyes szakértők hatékonyságát és a kezdő orvosoknál lényegesen jobb teljesítményt nyújtott. A MYCIN két fő vonatkozásban különbözött a DENDRAL-tól. Először is, a DENDRAL szabályaival ellentétben, a MYCIN-szabályok származtatásához nem létezett semmilyen általános elméleti modell. A szabályokat a szakértők kiterjedt kikérdezése révén kellett beszerezni, akik viszont a szabályokat könyvekből, más szakértőktől és közvetlen tapasztalatokból merítették. A másik különbség abból eredt, hogy a szabályoknak tükrözniük kellett az orvosi ismeret bizonytalanságát. A MYCIN-ben **bizonyossági tényező**knek (**certainty factor**s) (lásd 14. fejezet) nevezett bizonytalanságkezelő mechanizmust alkalmaztak, amelyről akkortájt úgy tűnt, jól tükrözi a tényállás diagnózisra gyakorolt hatásának orvosi megítélését.

A tárgytartomány ismeretének fontossága nyilvánvaló volt a természetes nyelvfelismerés területen is. Bár Winograd természetes nyelvfelismerő rendszere, a SHRDLU igen élénk érdeklődést keltett, a szintaktikai elemzéstől való függősége néhány, a kezdeti gépi fordításban már tapasztalt problémához vezetett. A program képes volt a kétértelműségen felülkerekedni, és a névmási szerkezeteket megérteni. Ez azonban azért volt lehetséges, mert a programot kifejezetten egy adott tárgytartományhoz – a kockavilághoz – fejlesztették ki. Néhány kutató, köztük Eugene Charniak, Winograd végzős társa az MIT-ről, felvetette, hogy a természetes nyelv robusztus felismerése a világról szóló általános ismereteket és ezen ismeretek általános felhasználási módszereit igényli.

A Yale-en, Roger Schank, a nyelvésszé lett MI-kutató, ezt a nézetet még jobban hangsúlyozta, azt állítván, hogy „olyan dolog, mint a szintaxis pedig nincs”, ami ugyan sok nyelvészt felháborított, de egyben egy hasznos

Created by XMLmind XSL-FO Converter.

eszmecserét is elindított. Schank és hallgatói a természetes nyelvet felismerő programok egész sorát építették meg (Schank és Abelson, 1977; Wilensky, 1978; Schank és Riesbeck, 1981; Dyer, 1983). A hangsúlyt azonban kevésbé *magára* a nyelvre, sokkal inkább a nyelv megértéséhez szükséges tudás reprezentálására és a vele való következtetésre helyezték. E problémakörbe tartozott a sztereotip helyzetek reprezentálása (Cullingford, 1981), az emberi memória szervezésének leírása (Rieger, 1976; Kolodner, 1983) és a célok, tervek megértése (Wilensky, 1983).

A valós alkalmazások széles körű elterjedése a működőképes tudásreprezentációs sémák iránti igények növekedéséhez vezetett. Számos különböző reprezentációs és következtető nyelvet fejlesztettek ki. Egyes megoldások a logikán alapultak – például a Prolog nyelv, amely Európában, és a `PLANNER` nyelvcsalád, amely az Egyesült Államokban lett népszerű. Mások, a Minsky által bevezetett **keret**ek (**frame**s) (Minsky, 1975) ötletét követve, inkább strukturált megközelítést választottak. Egybegyűjtöttek bizonyos eseménytípusokra vagy objektumokra jellemző tényeket, majd azokat a biológiai taxonómiára hasonlító nagy taxonomikus típushierarchiákba rendezték.

3.6. Az MI iparrá válik (1980-tól napjainkig)

Az első üzletileg sikeres szakértőrendszert, az R1-et a Digital Equipment Corporationnél (McDermott, 1982) alkalmazták. A rendszer az új számítógépes rendszerek megrendeléseit segítette konfigurálni, és 1986-ra évi mintegy 40 millió dollár megtakarítást jelentett a cégnek. 1988-ra a DEC MI-csoportja már 40 szakértőrendszert állított üzembe, és több ilyen rendszer üzembe állítása folyamatban volt. A DuPont cégnél 100 ilyen rendszer üzemelt, és folyamatban volt további 500 rendszer fejlesztése. Az ezekből származó becsült megtakarítás elérte az évi 10 millió dollárt. Majdnem minden nagyobb amerikai cég saját MI-csoporttal rendelkezett, és vagy használta, vagy tanulmányozta a szakértőrendszer technológiát.

1981-ben a japánok meghirdették az „ötödik generációs” (Fifth Generation) projektjüket – egy 10 éves tervet a Prolog nyelvet gépi kódként használó, intelligens számítógépes rendszerek építésére. Válaszul az Egyesült Államokban létrehozták az MCC (Microelectronics and Computer Technology Corporation) kutatótársulatot, amelynek célja a nemzeti versenyképesség biztosítása volt. Mindkét esetben az MI egy olyan általánosabb erőfeszítés része lett, amely a chiptervezésre és az ember–gép interfész kutatására is irányult. Az MCC és az Ötödik Generáció MI-komponensei azonban az ambiciózus célkitűzéseket mégsem tudták elérni. Nagy-Britanniában az Alvey-jelentés visszaállította a Lighthill-jelentés következtében leállított finanszírozást.[16]

Mindent egybevéve az MI-iparnak az 1980-as néhány millió dolláros forgalma 1988-ra 2 milliárd dollárra nőtt. Rövidesen ezután az „MI tele” periódus következett be, amikor sok cég belebukott abba, hogy extravagáns ígéreteit nem tudta teljesíteni.

3.7. A neurális hálók visszatérése (1986-tól napjainkig)

Bár a számítógép-tudomány az 1970-es évek végén a neurális hálók témakörről megfeledkezett, a kutatás más területeken folytatódott. A fizikusok, mint például Hopfield, a statisztikus mechanika módszereit használták, hogy a hálók tárolási és optimalizálási tulajdonságait elemezzék (Hopfield, 1982), úgy kezelve az egyszerű neuronok együttesét, mint atomok együttesét. A pszichológusok, David Rumelhartot és Geoff Hintont is beleértve, folytatták a memória neurális hálós modelljének kutatását. Mint ezt a 20. fejezetben megmutatjuk, az igazi lökés az 1980-as évek derekán történt, amikor legalább négy különböző kutatócsoport újra feltalálta a visszaterjesztéses tanuló algoritmust, azt az algoritmust, amit először Bryson és Ho írtak le 1969-ben. Az algoritmust számos tanulóproblémára alkalmazták mind a számítógépes tudományokban, mind a pszichológiában. Az eredmények széles körű bemutatására a nagy érdeklődést keltő *Parallel Distributed Processing* c. (Rumelhart és McClelland, 1986) gyűjteményes kötetben került sor.

Az intelligens rendszereknek ilyen, ún. **konnekcionista** (**connectionist**) modelljeit egyesek a Newell és Simon javasolta szimbolikus modellek, valamint a McCarthy és mások által alkalmazott logicista megközelítés közvetlen versenytársának vélték (Smolensky, 1988). Hogy egy bizonyos szinten az ember szimbólumokkal operál, nyilvánvalónak tűnhet. Sőt Terrence Deakon *The Symbolic Species* c. művében (Deakon, 1997) ezt a képességet az embereket *definiáló jellemzőnek* javasolja. A legmegrögzöttebb konnekcionisták azonban kérdőre vonták a tekintetben, hogy a kognitív folyamatok részletes modelljében a szimbolikus manipulációnak van-e egyáltalán valamilyen valós magyarázó szerepe. Ez a kérdés megválaszolatlan maradt, és a jelenlegi álláspont az, hogy a konnekcionista és a szimbolikus megközelítések egymás kiegészítői, nem pedig versenytársak.

[16] A kínos helyzet elkerülésére intelligens tudásalapú rendszerek (Intelligent Knowledge-Based Systems, IKBS) néven új kutatási területet definiáltak, hiszen a mesterséges intelligenciát hivatalosan törölték.

Created by XMLmind XSL-FO Converter.

3.8. Az MI tudománnyá válik (1987-től napjainkig)

A legutóbbi években az MI-kutatásnak mind tartalmában, mind módszertanában lényeges változások álltak be.[17] Mostanság inkább megszokott létező elméletekre építeni, mint teljesen újakat javasolni, az állításokat az intuíció helyett inkább szigorúan vett tételekre, illetve komoly kísérleti bizonyítékokra alapozni, továbbá a lényeges eredményeket nem játékproblémákon, hanem valós feladatokon bemutatni.

Az MI-t részben a létező kutatási területek – mint az irányításelmélet és a statisztika – korlátaival szembeni kitörési vágyból alapították meg. Most azonban az MI ezeket a területeket igyekszik magában foglalni. David McAllester szavaival:

> Az MI korai szakaszában plauzíbilisnak tűnt, hogy a szimbolikus számítások új formái, például a keretek és szemantikus hálók, a klasszikus elméletek nagyobb részét elavulttá tették. Ez egyfajta elszigetelődéshez vezetett, ahol az MI a számítási tudomány többségétől el lett választva. Ezzel az izolacionizmussal most szakítunk. Fel kell ismerni, hogy a gépi tanulást nem szabad elszigetelni az információelmélettől, hogy a bizonytalanság melletti következtetést nem szabad elszigetelni a sztochasztikus modellezéstől, hogy a keresést nem szabad elszigetelni a klasszikus optimalizálástól és szabályozástól, és hogy az automatikus következtetést nem szabad elszigetelni a formális módszerektől és a statikus elemzéstől.
>
> —(McAllester, 1998)

Módszertanát tekintve az MI-ben végre a tudományos megközelítés uralkodott el. Hogy egy hipotézist elfogadhassunk, szigorú empirikus kísérleteknek kell alávetni, és az eredmények relevanciáját statisztikailag kell verifikálni (Cohen, 1995). Manapság a kísérletek reprodukálhatóságát az internet és a megosztott tesztadat- és programkódtárak szavatolják.

Ez a folyamat a beszédfelismerés területén jól látható. Az 1970-es években igen sok különböző architektúrát és megközelítést próbáltak ki. Ezek közül sok *ad hoc* jellegű és gyenge volt, amelyek működését csupán néhány, erre a célra megválasztott példán demonstrálták. A legutóbbi években a **rejtett Markov-modell**eken (**hidden Markov model**s, **HMM**) alapuló megközelítések uralják e területet. A HMM-ek két aspektusa lényeges. Először is szigorú matematikai elméleten alapulnak. Ez lehetővé tette, hogy a beszédkutatók a más területeken kifejlesztett több évtizedes matematikai eredményekre építsenek. Másodszor, e modelleket valós és nagyméretű beszédgyűjteményt felhasználó tanulási folyamat során hozzák létre. Ez biztosítja robusztus működésüket. A szigorú vaktesztek a rejtett Markov-modellek folyamatos javulását mutatják. A beszédtechnológia és a vele rokon kézírás-felismerés útban van a széles körű ipari és fogyasztói alkalmazások felé.

Ez a trend a neurális hálókra is igaz. Az 1980-as években a kutatás többsége arra irányult, hogy kitapasztalják, a hálókkal meddig mehetnek el, és hogy megtanulják, a hálók a „hagyományos” technikáktól miben különböznek. A jobb módszertan és az elméleti háttér révén eljutottak ahhoz, hogy most a hálókat össze lehet hasonlítani a megfelelő statisztikai, alakfelismerési és gépi tanulási technikákkal, és az adott alkalmazáshoz meg lehet választani a leginkább sikerrel kecsegtetőt. Az ilyen fejlődés eredményeképpen az **adatbányászat** (**data mining**) technológia virágzó új iparrá nőtte ki magát.

Judea Pearl *Probabilistic Reasoning in Intelligent Systems* c. műve a valószínűség- és a döntéselmélet MI-n belüli újbóli elfogadását jelezte (Pearl, 1988). Mindez azt követően történt, hogy Peter Cheeseman *In Defense of Probability* cikkében összefoglalta az érdeklődés újraéledését (Cheeseman, 1985). A **Bayes-hálók** (**Bayesian network**s) formalizmusát a bizonytalan tények hatékony ábrázolására és a velük történő szabatos következtetés céljára találták ki. Ez a megközelítés a valószínűségi következtető rendszerek 1960-as és 1970-es években tapasztalt problémáit nagyrészt megoldotta, és ma uralja a bizonytalan következtetésre és a szakértőrendszerekre irányuló MI-kutatásokat. Ez a megközelítés teszi lehetővé a tapasztalatból való tanulást és ez kapcsolja össze a klasszikus MI és a neurális hálók legfontosabb eredményeit. Judea Pearl, továbbá Eric Horvitz és David Heckerman munkája támogatta a *normatív* szakértőrendszer gondolatát, azaz egy olyan rendszerét, amely a döntéselméleti törvényeknek megfelelően racionálisan cselekszik, és nem kísérli meg az emberi szakértőket imitálni (Pearl, 1982a; Horvitz és Heckerman, 1986; Horvitz és társai, 1986). A Windows™ operációs rendszer

[17] Vannak, akik ezeket a változásokat úgy jellemezték, mint az **elegánsak** (**neats**) győzelmét a **szakadtak** (**scruffies**) felett. Az elegánsak úgy gondolják, hogy az MI-elméleteknek szigorú matematikai alapokon kell állniuk, míg a szakadtak inkább sok ötletet próbálnának ki, programokat írnának, és megnéznék, mi működőképes ezekből. Mindkét megközelítés fontos. Az elegancia felé történő eltolódás annak a jele, hogy a terület egy stabil és érett szintet ért el (más kérdés persze, hogy ezt a stabilitást új „szakadt” gondolatok mennyire bolygatják fel).

Created by XMLmind XSL-FO Converter.

tartalmaz néhány normatív szakértőrendszert a felmerülő hibák javítására. Erről az elgondolásról szól a 13–16. fejezet.

Hasonló szelíd forradalom következett be a robotika, a gépi látás és a tudásreprezentációk területén. A problémák és bonyolultságuk jobb megértése, a növekvő matematikai háttérrel összefonódva, robusztusabb módszerekhez és megvalósítható kutatási menetrendekhez vezetett. Sok esetben a formalizálás és a speciálizálódás felaprózódást eredményezett. Az olyan témák, mint a látás és a robotika az MI fő vonalától egyre jobban elszigetelődnek. Az MI-nek a racionális ágensben megtestesült egységesítő képe egy olyan megközelítés, amely e divergáló területeken újra egységet teremthet.

3.9. Az intelligens ágensek kialakulása (1995-től napjainkig)

Valószínűleg az MI részproblémáiban elért sikereken felbátorodva a kutatók elővették a „teljes ágens” problémakörét. A teljes ágensarchitektúra legismertebb esete a SOAR, Allen Newell, John Laird és Paul Rosenbloom munkája (Newell, 1990; Laird és társai, 1987). Az ún. beágyazott mozgalom célul tűzte ki a valós környezetbe ágyazott, folytonos szenzorikus adatokat fogadó ágensek működésének a megértését. Az intelligens ágensek szempontjából az egyik legfontosabb környezet az internet. A világhálós alkalmazásokban az MI-rendszerek annyira mindennaposak lettek, hogy a „-bot” szóvégződés már a mindennapi nyelvbe is beépült. Ráadásul az MI-technológiák sok olyan internetes eszköznek képezik az alapját, amilyenek a keresőgépek, az ajánló rendszerek és a weboldalszerkesztő rendszerek.

E könyv első kiadása (Russell és Norvig, 1995) mellett más kurrens könyv is átvette az ágensperspektívát (Poole és társai, 1998; Nilsson, 1998). A teljes ágenstervezés egyik következménye, hogy fel kell ismerni, az MI eddig elszigetelt területeit minden bizonnyal valamelyest át kell szervezni, ha az eredményeiket össze akarjuk kapcsolni. Ma már széles körben elfogadott, hogy az érzékelő rendszerek (látás, szonár, beszédfelismerés stb.) nem képesek a környezetről tökéletesen megbízható információt szolgáltatni. A következtetésnek és a tervkészítésnek így fel kell készülnie a bizonytalanság kezelésére. Az ágensperspektíva másik lényegi következménye, hogy az MI az ágensekkel foglalkozó más területekkel, például az irányításelmélettel és a gazdaságtannal, sokkal közelebbi kontaktusba került.

4. A mesterséges intelligencia jelenlegi helyzete

Mit tehet az MI manapság? Nehéz erre tömör választ adni, mert annyi minden történt, és olyan sok a művelt részterület. Az alábbiakban bemutatunk néhány alkalmazást, másokról a könyv további részeiben szó lesz.

Autonóm tervkészítés és ütemezés: Több száz millió mérföldre a Földtől a NASA Remote Agent programja lett az első fedélzeti autonóm tervkészítő program, amely egy űrhajó műveleteinek ütemezését felügyelte (Jonsson és társai, 2000). A Remote Agent a terveit a Földről küldött magas szintű célokból generálta, és a tervek végrehajtása közben monitorozta az űrhajó működését, hibákat detektált, diagnosztizált, és visszaállította a helyes működést, ha problémák léptek fel.

Kétszemélyes játékok: Az IBM Deep Blue rendszere lett az első számítógépes sakkprogram, amely legyőzte a világbajnokot, amikor egy bemutató mérkőzésen 3,5 : 2,5 arányban győzedelmeskedett Garri Kaszparov felett (Goodman és Keene, 1997). Kaszparov nyilatkozata szerint egy „újfajta intelligenciát” érzett a sakktábla mögött. A *Newsweek* újság a mérkőzést az „agy utolsó védelmi vonalának” titulálta. Az IBM részvényei 18 milliárd dollárral emelkedtek.

Autonóm szabályozás: Az ALVINN számítógépes látórendszert arra tanították, hogy egy gépkocsit egy közlekedési sávot követve vezessen. A rendszert a CMU NAVLAB számítógép-vezérelt kis tehergépkocsijára helyezték, és arra használták, hogy az Egyesült Államokon keresztül elnavigáljon. A 2850 mérföldes távból a rendszer az idő 98%-ban vezetett. A maradó 2%-ért, főleg a sztrádalejáratokon, az ember vállalta felelősséget. A NAVLAB videokamerákkal rendelkezik, amelyek az ALVINN számára közvetítik az út képeit. A tanulóutakból szerzett tapasztalatok felhasználásával az ALVINN számítja ki a kormánykerék legjobb beállítását.

Diagnózis: Valószínűségi elemzésen alapuló orvosi diagnosztizáló rendszerek az orvosi tudományok több területén szakértő orvosok szintjén voltak képesek helytállni. Heckerman (Heckerman, 1991) leír egy esetet, amikor a nyirokcsomó-patológia egyik vezető szakembere gúnyolódó megjegyzést tesz egy pokolian nehéz esetet diagnosztizáló program javaslatára. A rendszer fejlesztője azt javasolja, hogy kérdezze meg a rendszertől a diagnózis magyarázatát.

Created by XMLmind XSL-FO Converter.

A gép rámutatott a döntését befolyásoló fő tényezőkre, és megmagyarázta az adott esetben jelentkező tünetek bonyolult kölcsönhatását. A szakember végül egyetértett a programmal.

Logisztikai tervkészítés: 1991-ben, az Öböl-válság idején az amerikai haderő automatikus logisztikai tervkészítésre és a szállítás ütemezésére egy DART – Dynamic Analysis and Replanning Tool – nevű rendszert alkalmazott. A rendszer működése egyidejűleg 50 ezer (teher- és személyszállító-) járműre terjedt ki, figyelembe vette a kiindulási és célállomásokat, útvonalakat és az összes paraméter közötti konfliktusfeloldást is. MI-technikák révén a terv órák alatt kész volt, szemben a heteket igénylő korábbi megoldásokkal. A Védelmi Kutatási Ügynökség (Defense Advanced Research Project Agency, DARPA) közleménye szerint, csupán ezen egyetlen alkalmazás kapcsán megtérült a DARPA által 30 éven keresztül az MI-re fordított befektetés.

Robotika: A mikrosebészetben manapság sok sebész robotsegédekre támaszkodik. A HipNav rendszer, miután számítógépes látási technikák segítségével létrehozta a páciens belső anatómiájának háromdimenziós modelljét, robotszabályozással irányítja a csípőprotézis behelyezését (DiGioia és társai, 1996).

Nyelvmegértés és problémamegoldás: A `PROVERB` rendszer a legtöbb embernél jobb keresztrejtvényfejtő (Littman és társai, 1999). Ehhez rendelkezik a lehetséges megfejtő szavakra vonatkozó korlátozásokkal, a régebbi keresztrejtvények nagy adatbázisával és sokféle információs forrással, szótárakat és az olyan online adatbázisokat is beleértve, mint mozicímlisták a bennük szereplő színészekkel. Így például képes megállapítani, hogy a „Nice Story” meghatározás megoldása „ETAGE”, mert az adatbázisában „Story in France/ETAGE” meghatározás/megoldás pár szerepel, és felismeri, hogy a „Nice *X*” és „*X* in France” mintáknak sokszor azonos a megoldása. A program természetesen nem tudta, hogy Nice egy város Franciaországban, a keresztrejtvényt mégis képes volt megoldani.[18]

Fentiek csupán példák a ma létező mesterséges intelligencia rendszerekre. Nem mágia vagy sci-fi – inkább tudomány, technika és matematika, amihez bevezetőt nyújt ez a könyv.

5. Összefoglalás

Ebben a fejezetben definiáltuk az MI-t és áttekintettük az MI fejlődésének kulturális hátterét. A legfontosabb gondolatok közül néhány:

- Az egyes emberek különböző módon vélekednek az MI-ről. A két fontos felteendő kérdés az, hogy: a gondolkodás vagy a viselkedés az, ami Önt érdekli? Embereket akar modellezni, vagy egy idealizált megközelítést választ?

- Ebben a könyvben azt a nézetet fogadjuk el, hogy az intelligencia lényegében a **racionális cselekvés**sel kapcsolatos. Egy **intelligens ágens**, ideális esetben, az adott szituációban a legjobb cselekvéshez folyamodik. Az ilyen értelemben vett intelligens ágensek építési problémáit fogjuk tanulmányozni.

- A filozófusok (i. e. 400-ig visszamenőleg) tették lehetővé az MI kialakulását azáltal, hogy felvetették: az elme bizonyos értelemben gépszerű, hogy valamilyen belső nyelvezetben kódolt tudásanyagon operál, és hogy a gondolat a helyes cselekvés megválasztásának eszköze.

- A matematikusok megadták a logikailag biztos, valamint a bizonytalan valószínűségi állítások manipulálásának eszközét. Megadták annak az alapjait is, hogy megértsük a számításokat és az algoritmusokról következtethessünk.

- A közgazdászok formalizálták a döntéshozatal folyamatát, hogy a döntéshozónak maximális várható hasznot biztosítson.

- A pszichológusok megerősítették azt a gondolatot, hogy az ember és az állatok információprocesszáló gépezetnek tekinthetők. A nyelvészek azt mutatták ki, hogy a nyelvhasználat ezzel a modellel összhangban van.

- A számítógép-technika biztosította az MI alkalmazását lehetővé tevő „műterméket”. Az MI-programok nagyok, a működésük lehetetlen lenne a memóriának és a sebességnek a számítógépes ipar biztosította nagyfokú fejlődése nélkül.

[18] Az ilyen és hasonló helyeken, ahol a könyv szerzője igazán csak angolul értelmes szófordulatokkal illusztrálja a mondanivalóját, az üzenet érdekében meghagyjuk a szavak eredeti angol formáját. (*A lektor*)

Created by XMLmind XSL-FO Converter.

- Az MI történetében voltak sikeres időszakok, de megtalálhatók a téves optimizmus és a lelkesedés, valamint a finanszírozás elapadása következtében beálló hanyatlás ciklusai is. Voltak időszakok, amelyek során új kreatív megközelítések bevezetése és a legjobb ötletek szisztematikus finomítása történt meg.

- Az MI fejlődése az utolsó évtizedben a tudományos módszereknek a kísérletezésben és a megközelítések összehasonlításában való szélesebb körű alkalmazása következtében felgyorsult.

- Az intelligencia elméleti alapjainak megértésében bekövetkezett jelenlegi fejlődés kéz a kézben együtt járt a valós rendszerek képességeinek javulásával. Az MI egyes részterületei jobban integrálódtak, és az MI és más tudományágak megtalálták a közös alapjukat.

5.1. Irodalmi és történeti megjegyzések

Herb Simon a *The Science of the Artificial* c. műben tárgyalja az MI módszertani státusát (Simon, 1981). A mű azokat a kutatási területeket tárgyalja, amelyek bonyolult műtermékekkel kapcsolatosak. Azt is bemutatja, hogy hogyan lehet az MI-t mind matematikának, mind tudománynak látni. Cohen (Cohen, 1995) áttekinti az MI kísérleti technológiáját. Ford és Hayes (Ford és Hayes, 1995) pedig a Turing-teszt hasznosságáról fejtik ki a véleményüket.

Az MI filozófiai és gyakorlati problémáiról olvasmányos áttekintést ad John Haugeland *Artificial Intelligence: The Very Idea* c. műve (Haugeland, 1985). A kognitív tudományról néhány újkeletű forrás (Johnson-Laird, 1988; Stillings és társai, 1995; Thagard, 1996) és az *Encyclopedia of the Cognitive Sciences* (Wilson és Keil, 1999) adnak jó leírást. A modern nyelvészet szintaktikai aspektusát (Baker, 1989), szemantikai aspektusát pedig (Chierchia és McConnell-Ginet, 1990) mutatja be jól. A számítógépes nyelvészettel Jurafsky és Martin (Jurafsky és Martin, 2000) foglalkoznak.

Az MI-kutatások kezdetétől a Feigenbaum és Feldman *Computers and Thought* (Feigenbaum és Feldman, 1963) és a Minsky *Semantic Information Processing* (Minsky, 1968) című művekben, továbbá a Donald Michie által szerkesztett *Machine Intelligence* sorozatban lehet olvasni. Webber és Nilsson (Webber és Nilsson, 1981) és Luger (Luger, 1995) nagy hatású cikkekből állítottak össze antológiákat. A *Neurocomputing* (Anderson és Rosenfeld, 1988) a neurális hálókkal foglalkozó korai cikkek gyűjteménye. Az *Encyclopedia of AI* (Shapiro, 1992) majdnem minden MI-témáról tartalmaz egy áttekintő cikket. Ezek a cikkek általában jó kiindulópontok az egyes témakörök kutatásával foglalkozó irodalomhoz.

A legfrissebb munkák a nagyobb MI-konferenciák – a kétévente megrendezett International Joint Conference on AI (IJCAI), az évenkénti European Conference on AI (ECAI) és a támogató szervezetről AAAI-ként ismert évenkénti National Conference on AI – kiadványaiban találhatók meg. Az MI általános kérdéseivel foglalkozó fő folyóiratok az *Artificial Intelligence*, a *Computational Intelligence*, az *IEEE Transactions on Pattern Analysis and Machine Intelligence*, az *IEEE Intelligent Systems* és az elektronikus formában megjelenő *Journal of Artificial Intelligence Research*. Sok, speciális területekkel foglalkozó folyóirat és konferencia is létezik, ezekről azonban a megfelelő fejezetekben lesz szó. Az MI fő szakmai szervezetei az American Association for Artificial Intelligence (AAAI), az ACM Special Interest Group in Artificial Intelligence (SIGART) és a Society for Artificial Intelligence and Simulation of Behaviour (AISB). Az AAAI *AI Magazine*-ja sok áttekintő és tematikus cikket publikál, az aaai.org weboldala pedig híreket és háttérinformációkat tartalmaz.

5.2. Feladatok

Az alábbi gyakorlópéldáknak az a rendeltetése, hogy vitát stimuláljanak. Néhány közülük féléves házi feladatnak is alkalmas. E feladatokat már most megkísérelhetik megoldani és később, a könyv olvasását befejezve, e próbálkozásokat megismételhetik.

1.1.

Saját szavaival definiálja: (a) az intelligencia, (b) a mesterséges intelligencia, (c) az ágens fogalmakat.

1.2.

Olvasnivaló. Olvassa el Turing eredeti cikkét az MI-ről (Turing, 1950). A cikkben Turing a célkitűzésével és az intelligenciatesztjével kapcsolatos potenciális kifogásokat tárgyalja. Melyik kifogásnak van még ma is súlya? Érvényesek-e a cáfolatai? El tud képzelni más kifogásokat is, amelyek a cikk megírását követő fejlődésből adódnak? Turing a cikkében azt jósolja, hogy 2000-ben egy számítógépnek 30%-os esélye lesz, hogy teljesítsen

Created by XMLmind XSL-FO Converter.

egy 5 perces Turing-tesztet egy nem képzett kérdezővel szemben. Ésszerűnek gondolja ezt? És mondjuk 50 év múlva?

1.3.

Olvasnivaló. Minden évben Loebner-díjjal jutalmazzák azt a programot, amely a Turing-teszt egy bizonyos verzióját legjobban képes teljesíteni. Kutassa fel a Loebner-díj legújabb győztesét és számoljon be róla. Milyen technikákra támaszkodik ez a megoldás? Milyen hatással lehet ez az MI jelenlegi helyzetére?

1.4.

A problémáknak vannak olyan jól ismert osztályai, amelyek egy számítógép számára kezelhetetlenül nehezek, más osztályok viszont számítógépen bizonyíthatóan nem dönthetők el. Azt jelenti-e ez, hogy az MI lehetetlen?

1.5.

Tegyük fel, hogy Evans `ANALOGY` programját úgy fejlesztjük tovább, hogy a standard IQ-tesztnél 200 pontot érjen el. Lesz-e akkor egy olyan programunk, ami intelligensebb az embernél? Magyarázza meg a válaszát!

1.6.

Hogyan lehetséges, hogy az önelemzés – valakinek a saját gondolatairól szóló beszámolója – pontatlan? Tévedhetek arról, amit gondolok? Vitassa meg!

1.7.

Olvasnivaló.

Tanulmányozza az MI-irodalmat, hogy eldönthesse, az alábbi feladatokat jelenleg meg lehet-e oldani számítógépen:

a) Egy színvonalas tenisz- (pingpong-) játszma.
b) Autóvezetés Kairó központjában.
c) Heti élelmiszer-vásárlás a piacon.
d) Heti élelmiszer-vásárlás a világhálón.
e) Elfogadható bridzsparti versenyszinten.
f) Új matematikai tételek felfedezése és bizonyítása.
g) Szándékosan humoros történet írása.
h) Kompetens jogi tanácsadás egy specializált jogi területen.
i) Beszélt angol nyelv valós időben történő fordítása beszélt svéd nyelvre.
j) Egy bonyolult sebészeti beavatkozás levezetése.

A ma még nem megvalósítható feladatok esetén próbálja azonosítani a nehézségeket, és megbecsülni, hogy mikorra tervezhető azok megoldása.

1.8.

Néhány szerző azt állította, hogy az érzékelés és a motorikus képességek az intelligencia legfontosabb komponensei, és hogy a „magasabb szintű” képességek szükségszerűen parazita jellegűek – adalékok az alattuk levő képességekhez. Nyilvánvaló, hogy az evolúció zöme és az agy nagy része az érzékeléssel és a motorikus képességekkel foglalkozik. Az MI pedig a játék vagy logikai következtetés jellegű feladatokat a valós világban való érzékelésnél és cselekvésnél sok szempontból könnyebbnek találta. Nem gondolja-e, hogy az MI-nek a magasabb szintű kognitív képességekre irányuló hagyományos összpontosítása téves célkitűzés?

1.9.

Az evolúció miért eredményezett racionálisan cselekvő rendszereket? Milyen célokat hivatottak az ilyen rendszerek teljesíteni?

1.10.

Created by XMLmind XSL-FO Converter.

Racionálisak-e a reflexszerű cselekvések (mint például a forró tűzhelytől elrántani a kezet)? Intelligensek-e?

1.11.

„Egy számítógép nem lehet intelligens – csak azt teszi, amit a programozó mond neki.” Igaz-e az állítás második része, és vajon implikálja-e az elsőt?

1.12.

„Az állatok nem lehetnek intelligensek – csak azt teszik, amit a génjeik mondanak nekik.” Igaz-e az állítás második része, és vajon implikálja-e az elsőt?

1.13.

„Az állatok, az emberek és a számítógépek nem lehetnek intelligensek – csak azt teszik, amit a részüket alkotó atomoknak a fizika törvényei parancsolnak.” Igaz-e az állítás második része, és vajon implikálja-e az elsőt?

Created by XMLmind XSL-FO Converter.

2. fejezet - Intelligens ágensek

Ebben a fejezetben az ágensek természetét, legyen az tökéletes vagy sem, a környezetek különbözőségét és az ágenstípusok ezekből származó sokféleségét tárgyaljuk.

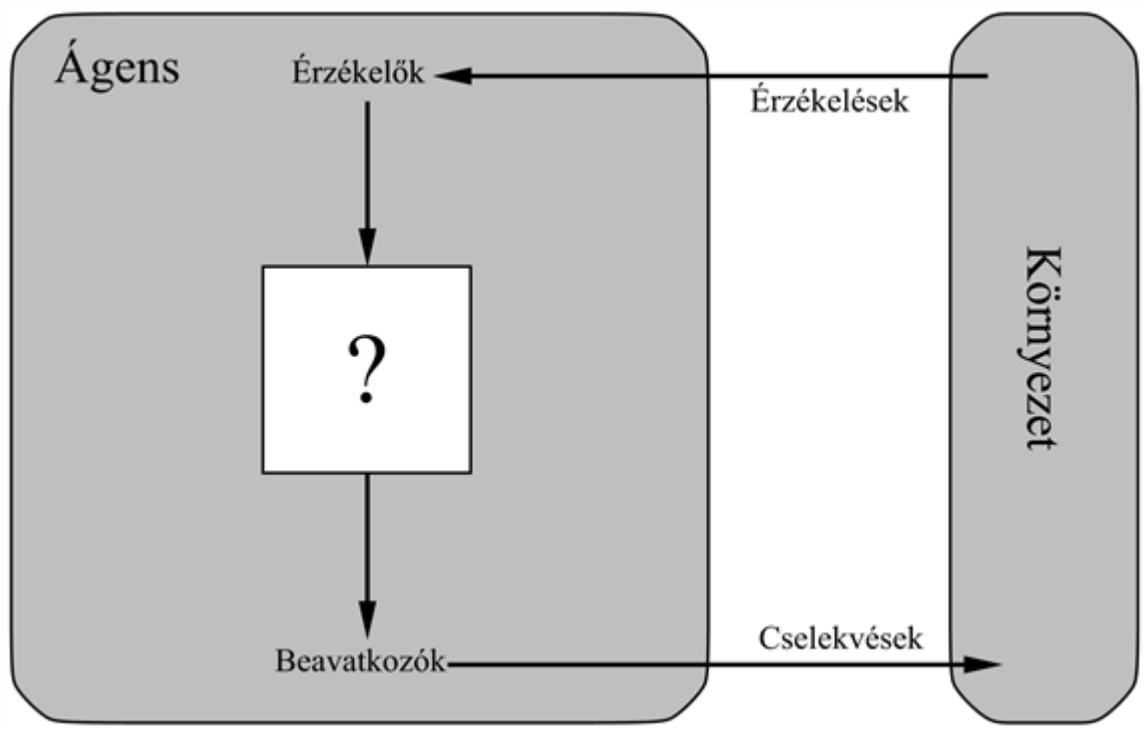
Az 1. fejezet a **racionális ágens** (**rational agent**) fogalmát úgy vezette be, mint a mesterséges intelligencia tárgyalásának általunk választott központi módszerét. Ebben a fejezetben pontosítjuk ezt a koncepciót. Látni fogjuk, hogy a racionalitás koncepciója az ágensek széles körére alkalmazható mindenféle elképzelhető környezetben. Az a tervünk a könyvben, hogy ezt a koncepciót néhány olyan tervezési elv megfogalmazásához használjuk, amelyekkel sikeres ágenseket építhetünk – olyan rendszereket, melyek joggal hívhatók **intelligens**nek.

Ágensek, környezetek és a köztük lévő kapcsolatok vizsgálatával kezdünk. Annak megfigyelése, hogy egyes ágensek jobban viselkednek, mint mások, a racionális ágens ötletéhez vezet majd – olyan ágenshez, amelyik olyan jól viselkedik, amennyire csak lehet. Az, hogy mennyire viselkedhet jól egy ágens, a környezet természetén múlik; bizonyos környezetek sokkal nehezebbek, mint mások. A környezetek nagyjából csoportosításával megmutatjuk, hogy egy környezet tulajdonságai hogyan befolyásolják a környezetnek megfelelő ágensek tervezését. Bemutatunk számos egyszerű ágenstervezési „csontvázat”, melyeket a könyv hátralevő fejezeteiben tovább részletezünk.

1. Bevezetés

Egy **ágens** (**agent**) bármi lehet, amit úgy tekinthetünk, mint ami az **érzékelői** (**sensors**) segítségével érzékeli a **környezet**ét (**environment**), és **beavatkozó**i (**actuator**s) segítségével megváltoztatja azt. A 2.1. ábra ezt az egyszerű elképzelést szemlélteti. Az emberi ágensnek van szeme, füle és egyéb szervei az érzékelésre, és keze, lába, szája és egyéb testrészei a beavatkozásra. A robotágens kamerákat és infravörös távolsági keresőket használ érzékelőként, és különféle motorokat beavatkozóként. A szoftverágens billentyűleütéseket, fájltartalmakat és hálózati adatcsomagokat fogad érzékelőinek bemeneteként, és képernyőn történő kijelzéssel, fájlok írásával, hálózati csomagok küldésével avatkozik be a környezetébe. Azzal az általános feltételezéssel fogunk élni, hogy minden ágens képes saját akcióinak érzékelésére (de nem mindig látja azok hatását).

2.1. ábra - Az ágensek környezetükkel érzékelőkön és beavatkozókon keresztül állnak kapcsolatban



Created by XMLmind XSL-FO Converter.

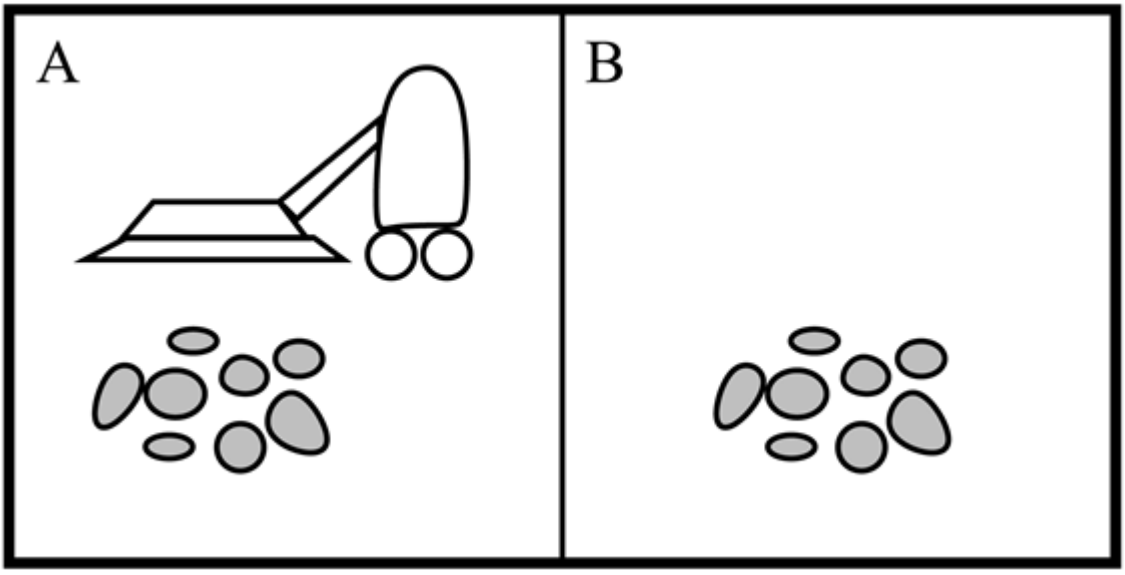
Fontos

Az **érzékelés** (**percept**) fogalmat használjuk az ágens érzékelő bemeneteinek leírására egy tetszőleges pillanatban. Egy ágens **érzékelési sorozat**a (**percept sequence**) az ágens érzékeléseinek teljes története, minden, amit az ágens valaha is érzékelt. Általánosságban, *egy adott pillanatban egy ágens cselekvése az addig megfigyelt teljes érzékelési sorozatától függhet.* Ha az összes lehetséges érzékelési sorozathoz meg tudjuk határozni az ágens lehetséges cselekvéseit, akkor lényegében mindent elmondtunk az ágensről. Matematikailag megfogalmazva azt mondhatjuk, hogy az ágens viselkedését az **ágensfüggvény** (**agent function**) írja le, ami az adott érzékelési sorozatot egy cselekvésre képezi le.

Elképzelhetjük *táblázatos formában* az ágensfüggvényt, amely forma mindenféle ágenst leír; az ágensek többségére ez igen nagy táblázatot jelentene – valójában végtelen nagyot, hacsak nem korlátozzuk a figyelembe veendő érzékelési sorozatok hosszát. Ha adott egy ágens, amivel kísérletezhetünk, akkor elméletileg megalkothatjuk ezt a táblázatot az összes lehetséges érzékelési sorozat kipróbálásával és az ágens válaszul végrehajtott cselekvéseinek feljegyzésével.[1] Természetesen a táblázat az ágens *külső* jellemzése. Egy mesterséges ágens *belsejében* az ágensfüggvényt egy **ágensprogram** (**agent program**) valósítja meg. Fontos, hogy megkülönböztessük e két dolgot. Az ágensfüggvény egy absztrakt matematikai leírás, az ágensprogram egy konkrét implementáció, amely az ágens architektúráján működik.

Ezen ötletek illusztrálására egy igen egyszerű példát fogunk használni – a 2.2. ábrán látható porszívóvilágot. A világ annyira egyszerű, hogy mindent le tudunk írni, ami megtörténik, továbbá ez egy kitalált világ, így sokféle variációt kitalálhatunk. A világban csak két helyszín van: az *A* és *B* négyzetek. A porszívóágens észleli, hogy melyik négyzetben van, valamint azt, hogy van-e ott piszok, vagy nincs. Lehetséges akciói: mozoghat balra vagy jobbra, felszívhatja a port, valamint nem csinál semmit. Egy egyszerű ágensfüggvény a következő: ha az aktuális négyzet koszos, szívd fel a koszt, egyébként menj át a másik négyzetbe. Ezen ágensfüggvény táblázatának egy részlete a 2.3. ábrán látható. Egy, az ágensfüggvényt megvalósító egyszerű ágensprogramot a fejezet későbbi részében, a 2.8. ábrán mutatunk be.

2.2. ábra - A porszívóvilág mindössze két helyszínnel



2.3. ábra - A 2.2. ábrán szereplő porszívóvilág egy egyszerű ágensfüggvénye részleges táblázatos formában

[1] Amennyiben az ágens cselekvésének kiválasztásában a véletlennek is szerepe van, úgy minden sorozatot többször kellene kipróbálnunk, hogy az egyes cselekvések valószínűségeit meghatározhassuk. Látszólag úgy tűnhet, hogy véletlenszerűen cselekedni botorság, a későbbiekben azonban látni fogjuk, hogy ez lehet nagyon intelligens viselkedés is.

Created by XMLmind XSL-FO Converter.

Érzékelési sorozat	Cselekvés
[*A*, *Tiszta*]	*Jobbra*
[*A*, *Piszkos*]	*Felszívás*
[B, *Tiszta*]	*Balra*
[*B*, *Piszkos*]	*Felszívás*
[*A*, *Tiszta*], [*A*, *Tiszta*]	*Jobbra*
[*A*, *Tiszta*], [*A*, *Piszkos*]	*Felszívás*
…	…
[*A*, *Tiszta*], [*A*, *Tiszta*], [*A*, *Tiszta*]	*Jobbra*
[*A*, *Tiszta*], [*A*, *Tiszta*], [*A*, *Piszkos*]	*Felszívás*
…	…

Fontos

A 2.3. ábrára nézve láthatjuk, hogy sokféle porszívóvilág-beli ágens definiálható egyszerűen a jobb oldali oszlop különböző kitöltésével. Ezek után a nyilvánvaló kérdés a következő: *Mi a táblázat helyes kitöltésének módja?* Más szavakkal, mitől lesz egy ágens jó vagy rossz, intelligens vagy buta. Ezeket a kérdéseket a következő részben fogjuk megválaszolni.

Mielőtt lezárnánk ezt a részt, fontos megjegyeznünk, hogy az ágens fogalom bevezetésének a célja nem a világ ágensekre és nem ágensekre történő felosztása, hanem egy olyan eszköz megteremtése, amivel rendszereket elemezhetünk. Egy kézi számológép is tekinthető ágensnek, amely azt a cselekvést választja, hogy „4”-et ír a kijelzőjére, ha „2 + 2 =” érzékelési sorozatot kap, de egy ilyen felfogás aligha segít minket a számológép megértésében.

2. Jó viselkedés: a racionalitás koncepciója

Egy **racionális ágens** (**rational agent**) olyan, amely helyesen cselekszik – elméletileg megfogalmazva az ágensfüggvény táblázatában minden bejegyzés helyesen van kitöltve. Helyesen cselekedni nyilvánvalóan jobb, mint helytelenül, de mit is jelent helyesen cselekedni? Első közelítésben azt mondjuk, hogy a helyes cselekedet az, amely az ágenst a legsikeresebbé teszi. Tehát szükségünk lesz egy módszerre, amivel a sikeresség mérhető. A környezetnek, valamint az ágens érzékelőinek és beavatkozóinak leírásával együtt ez az ágens feladatát teljes mértékben specifikálja. Ezáltal precízebben definiálhatjuk, mit is jelent racionálisnak lenni.

2.1. Teljesítménymértékek

A **teljesítménymérték** (**performance measure**) testesíti meg az ágens sikerességének kritériumát. Ha egy ágenst elhelyezünk a környezetében, akkor cselekvések sorozatával válaszol az érzékeléseire. A cselekvések sorozatának hatására a környezet állapotok sorozatán halad végig. Ha ez a sorozat kívánatos, akkor az ágens jól teljesített. Nyilvánvalóan nincs egyetlen olyan rögzített mérték, amely minden ágens számára megfelelő lenne. Megkérdezhetjük az ágens szubjektív véleményét, hogy mennyire érzi magát elégedettnek a saját teljesítményével, de egyes ágensek képtelenek lesznek válaszolni, míg mások becsapják magukat.[2] Ezért valamilyen objektív teljesítménymértékhez fogunk ragaszkodni, tipikusan egy olyanhoz, amit az ágens tervezője határozott meg.

Fontos

Példaként tekintsünk az előző részben megismert porszívó ágenst. Teljesítménymértékként javasolhatjuk az egy nyolcórás műszakban feltakarított por mennyiségét. Egy racionális ágenssel, természetesen, az ember azt kapja, amit kér. Azaz a racionális ágens azzal maximalizálhatja ezt a teljesítménymértéket, hogy feltakarítás után kiönti a szemetet, majd újra feltakarítja és így tovább. Egy finomított mérték a tiszta padló elérését díjazná. Például minden tiszta négyzet egy pontot érhet minden időlépésben (talán valamilyen levonással figyelembe véve az elfogyasztott elektromos energiát, illetve

[2] Az emberi ágensekre különösen jellemző a „savanyú a szőlő” effektus – azaz hogy nem is akarják igazán azt, amit nem sikerült megkapniuk, mint például: „Nos, sebaj, nem is akartam annyira azt a buta Nobel-díjat”.

Created by XMLmind XSL-FO Converter.

a keltett zaj mennyiségét is). *Általános szabályként megfogalmazható, hogy a teljesítménymértéket jobb aszerint megállapítani, hogy mit akarunk elérni a környezetben, mint aszerint, hogy miképp kellene az ágensnek viselkednie.*

A teljesítménymérték kiválasztása nem mindig könnyű. Például a „tiszta padló” előző paragrafusbeli fogalma a tisztaság időátlagán alapul. Még ugyanaz az átlagos tisztaság is elérhető két különböző ágenssel, az egyik közepes aktivitással dolgozik állandóan, míg a másik energikusan rövid ideig, aztán hosszú szüneteket tart. Hogy melyik a javasolt módszer, az látszólag a házvezetés tudományának kérdése, valójában egy mély filozófiai kérdés, messzibe mutató következményekkel. Mi a jobb: a túlzások vakmerő világa vagy a biztonságos, de unalmas létezés? Mi a jobb: egy gazdaság, ahol mindenki egyformán mérsékelten szegény, vagy egy másik, ahol vannak nagyon gazdagok és nagyon szegények? Ezen kérdések megválaszolását házi feladatként bízzuk a szorgalmas olvasóra.

2.2. Racionalitás

Négy dolgon múlik az, hogy egy adott pillanatban mi racionális. Ezek:

- A siker fokát mérő teljesítménymérték.
- Az ágens eddigi tudása a környezetről.
- A cselekvések, amiket az ágens képes végrehajtani.
- Az ágens érzékelési sorozata az adott pillanatig.

Fontos

Ez elvezet a **racionális ágens** (**rational agent**) definíciójához:

Az ideális racionális ágens minden egyes észlelési sorozathoz a benne található tények és a beépített tudása alapján minden elvárható dolgot megtesz a teljesítménymérték maximalizálásáért.

Vizsgáljuk meg az egyszerű porszívóágenst, amelyik megtisztítja a négyzetet, ha az koszos, és átmegy a szomszédos négyzetbe, ha nem az! Az ágensfüggvényt táblázatosan a 2.3. ábra adja meg. Ez egy racionális ágens? Attól függ! Először is, meg kell mondanunk mi a teljesítménymérték, mit tudunk a környezetről, és milyen érzékelői és beavatkozói vannak az ágensnek. Tegyük fel a következőket:

- A teljesítménymérték egy ponttal jutalmaz minden tiszta négyzetet minden időpillanatban egy 1000 időpillanatból álló „élettartam” alatt.
- A környezet „geográfiája” *a priori* ismert (2.2. ábra), de a piszok eloszlása és az ágens kezdeti pozíciója nem. A tiszta négyzetek tiszták maradnak, a felszívás pedig megtisztítja az aktuális négyzetet. A *Balra* és *Jobbra* cselekvések balra és jobbra mozgatják az ágenst, kivéve ha ezzel kikerülne a környezetből, mely esetben az ágens nem mozdul.
- A következő cselekvések léteznek: *Balra*, *Jobbra*, *Szív* és *Semmittevés* (azaz nem tesz semmit sem).
- Az ágens helyesen észleli jelenlegi helyzetét és azt, hogy van-e ott kosz.

Azt állítjuk, hogy *ezen feltételek mellett* az ágens igenis racionális: a várható teljesítménye legalább annyira jó, mint bármely más ágensé. A 2.4. feladatban azt kérjük majd, hogy ezt az állítást bizonyítsa be.

Könnyen belátható, hogy ugyanez az ágens más körülmények között irracionális lenne. Például ha minden koszt feltakarított, akkor szükségtelenül oszcillálni fog ide-oda; ha a teljesítménymérték tartalmaz egy büntetőpontot minden balra vagy jobbra történő mozgásért, akkor az ágens egészen gyengén teljesít. Egy jobb ágens abban az esetben, ha meggyőződött arról, hogy az összes négyzet tiszta, nem csinálna semmit. Ha a tiszta négyzetek ismét koszossá válhatnak, az ágensnek időnként ellenőriznie kellene őket, és kitakarítania, ha szükséges. Ha a környezet geográfiája ismeretlen, az ágensnek fel kell térképeznie ahelyett, hogy az *A* és *B* négyzetekhez köti magát. A 2.4. feladatban azt kérjük majd, hogy tervezzen ezekre az esetekre ágenseket.

Created by XMLmind XSL-FO Converter.

2.3. Mindentudás, tanulás és autonómia

Ügyelnünk kell a racionalitás és a **mindentudás** (**omniscience**) különválasztására. Egy mindentudó ágens tudja cselekedetei *valódi* kimenetelét, és ennek megfelelően cselekedhet, de a gyakorlatban a mindentudás lehetetlen. Vegyük a következő példát: egy nap a Nagykörúton sétálok és egy régi barátomat látom az út túloldalán közeledni. Az úton nincs forgalom, és ráérek, így – racionális lévén – elindulok a másik oldalra. Mindeközben 5000 kilométer magasan egy utasszállító repülőgép raktárajtaja leesik,[3] és mielőtt még átérnék az út túloldalára, rám esik és kilapít. Irracionális voltam, amikor át akartam kelni az úton? Valószínűtlen, hogy a gyászjelentésem így fog szólni: „egy idióta megpróbált keresztülmenni az utcán”.

Ez a példa azt mutatja, hogy a racionalitás nem azonos a tökéletességgel. A racionalitás az *elvárt* teljesítményt maximalizálja, míg a tökéletesség a *tényleges* teljesítményt. Az, hogy elállunk az ágensekkel szemben támasztott tökéletesség követelményétől, nem csak a velük szembeni méltányosság kérdése. Az a helyzet, hogy ha azt a cselekvést várjuk el egy ágenstől, ami a megtörténte után a legjobbnak bizonyul, akkor lehetetlen lesz egy ilyen ágens megalkotása – hacsak nem javítjuk a kristálygömbök vagy az időgépek teljesítményét.

Ezek alapján a racionalitás definíciónk nem követel mindentudást, hiszen a racionális választás csak *az adott pillanatig* felépített érzékelési sorozattól függ. Azt is biztosítanunk kell, hogy tudtunk nélkül se engedjük meg az ágensnek, hogy tudatosan unintelligens cselekvésbe kezdjen. Például ha egy ágens nem néz körül mindkét irányban, mielőtt egy forgalmas utat keresztez, akkor az érzékelési sorozata nem fogja neki megmondani, hogy egy hatalmas kamion közeledik nagy sebességgel. Azt mondja-e a racionalitás definíciónk, hogy ebben az esetben rendben átkelhet az úton? Egyáltalán nem! Először is, az útkeresztezés nem lenne racionális ezen információszegény észlelési sorozat mellett: az átkelés szétnézés nélkül túlságosan rizikós. Másodszor, egy ideális racionális ágensnek a „szétnézés” cselekvést kellene választania az útra lépés előtt, mivel ez segíti a teljesítmény maximalizálásában. *A hasznos információk beszerzése érdekében* végbevitt – általában **információgyűjtés**nek (**information gathering**) hívott – cselekedetek a racionalitás fontos részét jelentik, részletesebben a 16. fejezetben lesz szó róluk. Az információgyűjtés másik példáját a **felfedezés** (**exploration**) szolgáltatja, amit egy porszívóágensnek egy ismeretlen környezetben meg kell tennie.

Definíciónk nemcsak azt követeli meg a racionális ágenstől, hogy információt gyűjtsön, hanem azt is, hogy amennyit csak lehet **tanul**jon (**learn**) a megfigyeléseiből. Az ágens kezdeti konfigurációja valamilyen előzetes tudást tükrözhet a környezetről, de ahogy az ágens tapasztalatot szerez, ez a tudás módosulhat és átértékelődhet. Vannak szélsőséges esetek, amikor a környezet *a priori* teljesen ismert. Ilyen esetekben az ágensnek nem kell érzékelnie vagy tanulnia, egyszerűen csak helyesen cselekszik. Természetesen az ilyen ágensek nagyon sérülékenyek. Vegyük például az egyszerű ganajtúró bogarat. Miután kiásta a fészkét és elhelyezte a tojásait, a közeli halomból egy ganajlabdát készít, hogy elzárja a fészek bejáratát. Ha útközben a labdát eltávolítjuk a fogói közül, a bogár folytatja útját, és a nem létező labdával is eljátssza a fészek lezárását, miközben észre sem veszi, hogy a labda hiányzik. Az evolúció beépített egy feltételezést a bogár viselkedésébe, és amikor ez megsérül, sikertelen viselkedés lesz az eredménye. Kicsit intelligensebb a szöcskeölő darázs. A nőstény ás egy lyukat, keres és megcsíp egy hernyót, elcipeli a lyukhoz, bemegy a lyukba megnézni, hogy minden rendben van-e, majd bevonszolja a hernyót és lerakja a tojásait. A hernyó szolgál a tojások kikelése után táplálékul. Eddig jó, de ha egy rovarkutató pár centiméterrel odébb rakja a hernyót, miközben a darázs a lyukat ellenőrzi, az visszalép a hernyó odacipelése fázishoz, és a tervét változtatás nélkül folytatja onnan, akár tucatnyi hernyóelmozdító közbelépés után is. A darázs képtelen megtanulni, hogy öröklött terve hibás, így nem fogja megváltoztatni.

A sikeres ágensek három különböző periódusra bontják az ágensfüggvény kiszámításának feladatát: amikor az ágenst tervezik, a számítás egy részét tervezőik végzik; amikor megfontolja a következő cselekvését, az ágens további számításokat végez; és amikor tanul a tapasztalataiból, még további számításokat végez annak eldöntésére, hogy hogyan módosítsa a viselkedését.

Fontos

Addig a szintig, míg az ágens a tervezői által beépített tudásra épít és nem saját megfigyeléseire, azt mondjuk, hogy az ágens nem **autonóm** (**autonom**). Egy racionális ágensnek autonómnak kell lennie – mindent, amit csak megtanulhat, meg kell tanulnia ahhoz, hogy a hiányos vagy hibás előzetes tudását kompenzálja. Például az a porszívóágens, amelyik megtanulja előrelátni, hogy hol és mikor tűnik fel

[3] Lásd N. Henderson: Sürgősen új ajtózárakat a Boeing 747 jumbo jetek számára. *Washington Post,* 1989. augusztus 24.

Created by XMLmind XSL-FO Converter.

további szemét, jobban fog dolgozni, mint amelyik nem. Praktikus okból ritkán kell teljes autonómiát biztosítani már a kezdetektől: amikor egy ágensnek nincs tapasztalata, vagy csak kevés van, akkor véletlenszerűen kellene cselekednie, hacsak a tervezője nem ad valamilyen segítséget. Így, mint ahogy az evolúció ellátja az állatokat elegendő beépített reflexszel a túléléshez addig, míg saját maguk is megtanulják mindazt, ami a túléléshez szükséges, a mesterséges intelligens ágensek kezdő tudással és tanulási képességgel való ellátása is ésszerű lenne. Miután elegendő tapasztalatot szerzett a környezetéről, a racionális ágens viselkedése gyakorlatilag *függetlenné* válhat a beépített előzetes tudásától. Ily módon a tanulás alkalmazásával olyan ágens tervezhető, amely sokféle környezetben is sikeres lesz.

3. A Környezetek természete

Most, hogy már rendelkezünk a racionalitás definíciójával, már majdnem készen állunk arra, hogy racionális ágensek építéséről gondolkodjunk. Először azonban meg kell vizsgálnunk a **feladatkörnyezetek**et (**task environment**s), amelyek lényegében a „problémák”, amelyekre a racionális ágensek jelentik a „megoldásokat”. Először azt mutatjuk meg, hogyan kell a feladatkörnyezetet meghatározni, több példával illusztrálva a folyamatot. Ezek után megmutatjuk, hogy többféle feladatkörnyezet létezik. A környezet típusa közvetlenül befolyásolja az ágensprogram megfelelő tervezését.

3.1. A környezet meghatározása

Az egyszerű porszívóágens racionalitásának vizsgálatakor specifikálnunk kellett egy teljesítménymértéket, a környezetet és az ágens beavatkozóit és érzékelőit. Mindezeket a **feladatkörnyezet** (**task environment**) címszó alatt fogjuk egyesíteni. A betűszavakban gondolkodók számára ezt **TKBÉ – Teljesítmény**, **Környezet**, **Beavatkozók**, **Érzékelők** (**Performance**, **Environment**, **Actuators**, **Sensors**) leírásnak hívjuk. Egy ágens tervezése során az első lépésnek mindig a feladatkörnyezet lehető legteljesebb meghatározásának kell lennie.

2.4. ábra - Egy automatizált taxi feladatkörnyezetének TKBÉ-leírása

Ágenstípus	Teljesítménymérték	Környezet	Beavatkozók	Érzékelők
Taxisofőr	Biztonságos, gyors, törvényes, kényelmes utazás, maximális haszon	Utak, egyéb forgalom, gyalogosok, ügyfelek	Kormány, gáz, fék, index, kürt, kijelző	Kamerák, hangradar, sebességmérő, GPS (műholdas pozicionáló), kilométeróra, motorérzékelők, billentyűzet

A porszívóvilág egyszerű példa volt, vegyünk most egy bonyolultabbat: az automatizált taxisofőr példáját. Ezt a példát fogjuk a fejezet hátralevő részében használni. Ki kell emelnünk, mielőtt az olvasó megrémülne, hogy a teljesen automata taxisofőr jelenleg valamelyest túlmutat a létező technológia lehetőségein. (A 4. szakasz - A mesterséges intelligencia jelenlegi helyzete található egy létező sofőr robot leírása, de az „Intelligent Transportation Systems” konferencia legutóbbi kiadványait is érdemes megnézni.) A teljes autóvezetési feladat rendkívül *nyitott végű*. Nincs határa a lehetségesen felbukkanó körülmények újszerű kombinációinak – egy másik ok arra, hogy ezt válasszuk vizsgálatunk tárgyául. A 2.4. ábra összefoglalja a taxi feladatkörnyezetének TKBÉ-leírását. A következőkben részletesen tárgyaljuk az egyes elemeket.

Elsőként: mi az a **teljesítménymérték**, amit az automata taxisofőrünk elé állítunk? Elvárt jellemzők a helyes célállomás elérése; az üzemanyag-fogyasztás és az elhasználódás minimalizálása; az út költségének és/vagy idejének minimalizálása; a közlekedési szabályok megszegésének és más vezetők megzavarásának minimalizálása; a biztonság és utaskényelem maximalizálása; a haszon maximalizálása. Nyilvánvalóan ezen célok egy része konfliktusban van egymással, kompromisszumokra lesz szükség.

A következő: mi a taxi vezetési **környezet**e? Minden taxisofőrnek több típusú úttal kell szembenéznie, a vidéki és városi utaktól egészen a tizenkét sávos autópályákig. Az utak más forgalommal, gyalogosokkal, kóbor

Created by XMLmind XSL-FO Converter.

állatokkal, útkarbantartással, rendőrautókkal, tócsákkal és gödrökkel is rendelkeznek. A taxinak a lehetséges és a jelenlegi utasokkal is kapcsolatban kell állnia. Van továbbá néhány opcionális lehetőség. Lehet, hogy a taxinak Dél-Kaliforniában kell üzemelnie, ahol a hó ritkán jelent problémát, vagy Alaszkában, ahol ritkán nem. Közlekedhet mindig az út jobb oldalán, vagy szeretnénk, hogy elég rugalmas legyen ahhoz, hogy a bal oldalon is vezessen abban az esetben, ha Nagy-Britanniában vagy Japánban akarjuk működtetni. Nyilvánvalóan minél korlátozottabb a környezet, annál egyszerűbb a tervezési probléma.

Az automata taxisofőr rendelkezésére álló **beavatkozó**k lényegében ugyanazok lesznek, amelyek egy emberi vezető számára adottak: a motor vezérlése a gázpedál segítségével, kormányzás és fékezés. Ezek mellett szükséges egy képernyő vagy beszédszintetizátor a kimenet megjelenítéséhez az utasok számára, és valószínűleg valamilyen mód arra, hogy más járművekkel kommunikálhasson, udvariasan vagy másképp.

A vezetési környezetben levő céljai eléréséhez a taxinak tudnia kell, hogy hol van, mi más van az úton, és hogy milyen gyorsan halad. Alapvető **érzékelő**i között lenni kell ezért egy vagy több vezérelhető tv-kamerának, sebességmérőnek és kilométerórának. A jármű helyes irányításához – különösen a kanyarokban – szüksége lesz gyorsulásmérőre, ezenkívül ismernie kell a jármű mechanikai állapotát, így szüksége lesz a szokásos motor- és elektromosrendszer-érzékelők rendszerére is. Lehetnek olyan műszerei is, amelyek nem érhetők el az átlagos emberi sofőrök számára: egy műholdas globális pozicionáló rendszer (GPS), amely megadja a pontos pozícióját egy elektronikus térképen, vagy infravörös- és hangradar-érzékelők más autótól és tárgyaktól való távolságok érzékelésére. Végül szüksége lesz egy mikrofonra vagy billentyűzetre, aminek a segítségével az utasok az úti céljukat megadhatják.

A 2.5. ábrán felvázoltuk az alapvető TKBÉ-elemeket számos további ágenstípusra. További példák a 2.5. feladatban jelennek meg. Egyes olvasóink számára meglepő lehet, hogy az ágenstípusok listájában olyan programokat is szerepeltetünk, amelyek a billentyűzet bemenetének és a karakteres képernyő kimenetének teljesen mesterséges környezetében működnek. „Igen”, mondhatja valaki, „ez nem egy valódi környezet, ugye?”. A tény az, hogy nem a „valódi” és a „mesterséges” környezetek közötti különbségtétel számít, hanem az ágens viselkedése, a környezet keltette érzékelési sorozat, valamint a teljesítménymérték közötti viszony komplexitása. Egyes „valódi” környezetek valójában igen egyszerűek. Például egy robot, amit a futószalagon érkező alkatrészek vizsgálatára terveztek, sok egyszerűsítő feltevéssel élhet: a világítás állandó, a futószalagon csak általa ismert alkatrészek vannak, és csak két cselekvés létezik (elfogad vagy elutasít).

2.5. ábra - Példák ágenstípusokra és TKBÉ-leírásuk

Ágenstípus	Teljesítménymérték	Környezet	Beavatkozók	Érzékelők
Orvosi diagnosztikai rendszer	Egészséges páciens, költségek és perek minimalizálása,	Páciens, kórház, személyzet	Kérdések kijelzése, vizsgálatok, diagnózisok, kezelések, beszámolók	Tünetek, leletek, páciensek válaszainak bevitele billentyűzeten
Műholdas képfeldolgozó rendszer	Helyes képkategorizálás	Orbitális műhold földi kapcsolata	A kép kategorizálásának kijelzése	Színes képpontmátrixok
Alkatrész-felvevő robot	A helyes tartályokban levő alkatrészek aránya	Futószalag alkatrészekkel, tartályok	Csuklós kar és fogó	Kamera, csuklószög-érzékelők
Olajfinomító-vezérlő	A tisztaság, hozam és biztonság maximalizálása	Finomító, operátorok	Szelepek, pumpák, fűtők és kijelzők	Hőmérséklet, nyomás, kémiai érzékelők
Interaktív angol tanár	A tanulók tesztered-ményeinek maximalizálása	Tanulók egy csoportja, tesztelő hivatal	Gyakorlatok, javaslatok és javítások kijelzése	Billentyűzet bevitel

Ezzel ellentétben bizonyos **szoftverágens**ek (vagy szoftverrobotok, azaz **szoftbot**ok) részletgazdag, korlátok nélküli környezetekben léteznek. Képzeljünk el egy szoftbotot, amelyik egy nagy utasszállító gép szimulátorával repül. A szimulátor egy részletgazdag, komplex környezet, amelyben más repülőgépekkel kapcsolatos és földi műveletek is vannak, és a szoftverágensnek valós időben kell választania az akciók széles választékából. Vagy képzeljünk el egy szoftbotot, amelyet internetes hírforrások figyelésére terveztek, és az

Created by XMLmind XSL-FO Converter.

érdekes híreket megmutatja az ügyfeleinek. A helyes működéshez szüksége lesz valamilyen természetes nyelv feldolgozási képességre, meg kell tanulnia, hogy melyik ügyfelét mi érdekli, és szüksége lesz a tervei dinamikus megváltoztatására – például amikor az egyik hírforráshoz megszűnik a kapcsolata vagy amikor egy új forrás létesül. Az internet egy olyan környezet, amelynek komplexitása versenyez a fizikai világéval, és amelynek lakói között sok mesterséges ágens van.

3.2. A környezetek tulajdonságai

Az MI-ben felmerülő feladatkörnyezetek választéka nyilvánvalóan hatalmas. Mindazonáltal meghatározhatunk viszonylag kevés számú dimenziót, amelyek mentén a feladatkörnyezeteket kategorizálhatjuk. Ezek a dimenziók nagymértékben meghatározzák a helyénvaló ágenstervezést és az ágensimplementációk alaptechnikáinak alkalmazhatóságát. Először felsoroljuk a dimenziókat, azután több feladatkörnyezetet elemzünk az elképzelések bemutatására. Az itt szereplő definíciók informálisak – a későbbi fejezetek sokkal pontosabban fogalmazzák meg a környezeteket, és mindegyik fajtára példákat mutatnak majd.

- **Teljesen megfigyelhető** (**fully observable**) vagy **részlegesen megfigyelhető** (**partially observable**). Ha az ágens szenzorai minden pillanatban hozzáférést nyújtanak a környezet teljes állapotához, akkor azt mondjuk, hogy a környezet teljesen megfigyelhető.[4] Egy környezet ténylegesen teljesen megfigyelhető, ha az érzékelők minden olyan aspektusát észlelik, amelyek a cselekvés kiválasztásához *relevánsak* – a relevancia pedig a teljesítménymértéktől függ. A teljesen megfigyelhető környezetek kényelmesek, mivel az ágensnek nem kell semmilyen belső állapotot nyilvántartania a környezet nyomon követéséhez. Egy környezet lehet részlegesen megfigyelhető a zajos és pontatlan szenzorok miatt, vagy mivel az állapot egyes részei egyszerűen nem szerepelnek a szenzorok adatai között – például egy helyi koszérzékelő szenzorral rendelkező porszívóágens nem tudja megmondani, van-e piszok más négyzetekben, és egy automata taxi nem láthatja, hogy más vezetők mit gondolnak.

- **Determinisztikus** (**deterministic**) vagy **sztochasztikus** (**stochastic**). Amennyiben a környezet következő állapotát jelenlegi állapota és az ágens által végrehajtott cselekvés teljesen meghatározza, akkor azt mondjuk, hogy a környezet determinisztikus, egyébként sztochasztikus. Teljesen megfigyelhető, determinisztikus környezetben az ágensnek elvben nem kell a bizonytalansággal törődnie. Amennyiben azonban a környezet részlegesen megfigyelhető, úgy sztochasztikusnak *tűnhet*. Különösen igaz ez, ha a környezet összetett, nehezen teszi lehetővé nem megfigyelhető aspektusainak követését. Így gyakran jobb az *ágens szemszögéből* determinisztikusnak vagy sztochasztikusnak tekinteni egy környezetet. A taxivezetés ebben az értelemben nyilvánvalóan sztochasztikus, mivel senki sem tudja megjósolni a forgalmat pontosan, továbbá egy motor minden figyelmeztetés nélkül lerobbanhat, és a kerekek váratlanul kidurranhatnak. A porszívóvilág, ahogy leírtuk, determinisztikus, de egyes variációk tartalmazhatnak sztochasztikus elemeket, például véletlenszerűen megjelenő piszkot vagy megbízhatatlan szívási mechanizmust (2.12. feladat). Ha a környezet más ágensek cselekvéseit leszámítva determinisztikus, akkor azt **stratégiai**nak (**strategic**) nevezzük.

- **Epizódszerű** (**episodic**) vagy **sorozatszerű**[5] (**sequential**). Epizódszerű környezetben az ágens tapasztalata elemi „epizódokra” bontható. Minden egyes epizód az ágens észleléseiből és egy cselekvéséből áll. Nagyon fontos, hogy a következő epizód nem függ az előzőben végrehajtott cselekvésektől. Epizódszerű környezetekben az egyes epizódokban az akció kiválasztása csak az aktuális epizódtól függ. Sok osztályozási feladat epizódszerű. Például az összeszerelő soron levő hibás alkatrészeket észlelő ágens minden egyes döntését az aktuális alkatrész alapján hozza, függetlenül a korábbi döntésektől, továbbá az aktuális döntés nem befolyásolja, hogy a következő alkatrész hibás lesz-e. Másrészt, sorozatszerű környezetekben az aktuális döntés befolyásolhat minden továbbit. A sakk és a taxivezetés sorozatszerű: a rövid távú akciók mindkét esetben hosszú távú következményekkel járhatnak. Az epizódszerű környezetek sokkal egyszerűbbek a sorozatszerűeknél, hiszen az ágensnek nem kell előre gondolkodnia.

- **Statikus** (**static**) vagy **dinamikus** (**dynamic**). Ha a környezet megváltozhat, amíg az ágens gondolkodik, akkor azt mondjuk, hogy a környezet az ágens számára dinamikus; egyébként statikus. A statikus környezetekkel egyszerű bánni, mivel az ágensnek nem kell állandóan a világot figyelnie, miközben dönt a cselekvés felől, és nem kell az idő múlásával sem törődnie. Másrészt, a dinamikus környezetek állandóan azt kérdezik az ágenstől, hogy mit akar tenni; ha még nem döntötte el, az annak számít, hogy úgy döntött, hogy

[4] A könyv első kiadása a **hozzáférhető** (**accessible**) és a **nem hozzáférhető** (**inaccessible**) kifejezéseket használta a **teljesen** és a **részlegesen megfigyelhető** (**fully** és **partially observable**) helyett; a **nemdeterminisztikus** (**nondeterministic**) a **sztochasztikus** (**stochastic**) helyett, valamint a **nem epizódszerű** (**nonepisodic**) a **sorozatszerű** (**sequential**) kifejezés helyett szerepelt. Az új terminológia jobban illeszkedik az elterjedt gyakorlathoz.

[5] A „sorozatszerű” (sequential) szót a számítástudományban a „párhuzamos” (parallel) ellentéteként is használják. A két jelentés alapvetően független egymástól.

Created by XMLmind XSL-FO Converter.

nem tesz semmit. Ha a környezet nem változik az idő előrehaladtával, de az ágens teljesítménymértéke igen, akkor azt mondjuk, hogy a környezet **szemidinamikus** (**semidynamic**). A taxivezetés nyilvánvalóan dinamikus: a többi autó és a taxi továbbhalad, miközben a vezetési algoritmus azon bizonytalankodik, hogy mit is tegyen. Az órával játszott sakk szemidinamikus. A keresztrejtvények statikusak.

- **Diszkrét** (**discrete**) vagy **folytonos** (**continuous**). A diszkrét/folytonos felosztás alkalmazható a környezet *állapotára,* az *időkezelés* módjára, az ágens *észleléseire,* valamint *cselekvéseire*. Például egy diszkrét állapotú környezet, mint amilyen a sakkjáték, véges számú különálló állapottal rendelkezik. A sakkban szintén diszkrét az akciók és cselekvések halmaza. A taxivezetés folytonos állapotú és idejű probléma: a sebesség, a taxi és más járművek helye folytonos értékek egy tartományát járja végig a folytonos időben. A taxivezetés akciói szintén folytonosak (például kanyarodási szögek stb.). Szigorúan véve a digitális kameráktól érkező bemenet diszkrét, de tipikusan úgy kezeljük, mint amely folyamatosan változó mennyiségeket és helyeket reprezentál.

- **Egyágenses** (**single agent**) vagy **többágenses** (**multiagent**). Az egyágenses és többágenses környezetek közötti különbségtétel egyszerűnek tűnhet. Például a keresztrejtvényt megfejtő ágens önmagában nyilvánvalóan egyágenses környezetben van, míg egy sakkozó ágens egy kétágensesben. Vannak azonban kényes kérdések. Először is: leírtuk azt, hogy egy entitás hogyan tekinthető ágensnek, ugyanakkor nem magyaráztuk meg, mely entitások tekintendők ágensnek. Egy *A* ágensnek (például a taxisofőrnek) egy *B* objektumot (egy másik járművet) ágensnek kell tekintenie, vagy egyszerűen egy sztochasztikusan viselkedő dolognak, a tengerparti hullámokhoz vagy a szélben szálló falevelekhez hasonlatosan? A választás kulcsa az, hogy vajon *B* viselkedése legjobban egy *A* viselkedésétől függő teljesítménymérték maximalizálásával írható-e le. Például a sakkban a *B* ellenfél saját teljesítménymértékét próbálja maximalizálni, amely – a sakk szabályainak következtében – *A* teljesítménymértékét minimalizálja. Így a sakk egy **versengő** (**competitive**) többágenses környezet. Másrészről, a taxi vezetési környezetben az ütközések elkerülése az összes ágens teljesítménymértékét maximálja, így az részben **kooperatív** (**cooperative**) többágenses környezet. Emellett részben versengő is, hiszen például csak egy autó tud egy parkolóhelyet elfoglalni. A többágenses környezetekben felmerülő ágenstervezési problémák gyakran egészen mások, mint egyágenses környezetekben. Többágenses környezetekben például a **kommunikáció** (**communication**) gyakran racionális viselkedésként bukkan fel; egyes részlegesen megfigyelhető versengő környezetekben a **sztochasztikus viselkedés** racionális, hiszen így elkerülhetők a megjósolhatóság csapdái.

Ahogy várható, a legnehezebb a *részlegesen megfigyelhető, sztochasztikus, sorozatszerű, dinamikus, folytonos* és *többágenses* eset. Az is kiderül, hogy a valós helyzetek legtöbbje olyan bonyolult, hogy *valódi* determinisztikusságuk vitatott kérdés; gyakorlati okokból sztochasztikusként kezelendők. A taxivezetés mindezen szempontok szerint nehéz.

A 2.6. ábra ismerős környezetek tulajdonságait sorolja fel. Vegyük észre, hogy a válaszok nem mindig egyértelműek. Például a sakkot teljesen megfigyelhetőnek tüntettük fel; szigorúan véve ez hibás, mivel a rosálásra, menet közbeni ütésre és ismétléses döntetlenre vonatkozó egyes szabályok megkövetelik a játék menetével kapcsolatos adatok megjegyzését, amelyek a táblázat állapotának részeként nem figyelhetők meg. A megfigyelhetőség ezen kivételei persze csekélyek egy taxivezető, egy angol nyelvtanár vagy egy orvosi diagnosztikai rendszerhez képest.

A táblázatban bizonyos további válaszok a feladatkörnyezet definiálásától függnek. Az orvosi diagnosztikai feladatot egyágensesnek tüntettük fel, mivel a betegben zajló betegségi folyamat eredményesen nem modellezhető ágensként, de egy orvosi diagnosztikai rendszernek szükség esetén törődnie kell ellenszegülő betegekkel és szkeptikus munkatársakkal, így a környezetnek lehet többágenses aspektusa. Továbbá az orvosi diagnosztika epizódszerű, ha valaki a feladatot úgy tekinti, mint a tünetek listája alapján történő diagnózis kiválasztását; a probléma sorozatszerű, ha a feladat tartalmazhat egy vizsgálatsorozat-javaslatot, az eredmények kiértékelését a kezelés folyamán és így tovább. Továbbá sok környezet epizódszerű az ágensek egyéni akcióinál magasabb szinteken. Például egy sakktorna játszmák sorozatát tartalmazza, minden játék egy epizód, mivel (nagyjából) az ágens adott játszmabeli lépéseinek hozzájárulása az ágens teljesítményéhez nem függ a korábbi játszmákban választott lépéseitől. Másrészről, az egy játszmán belüli döntéshozatal biztosan sorozatszerű.

2.6. ábra - Példák feladatkörnyezetekre és jellemzőik

Created by XMLmind XSL-FO Converter.

Környezet	Megfigyelhető	Determinisztikus	Epizódszerű	Statikus	Diszkrét	Ágensek
Kereszt-rejtvény	Teljesen	Determinisztikus	Sorozat	Statikus	Diszkrét	Egy
Sakkjátszma időméréssel	Teljesen	Stratégiai	Sorozat	Szemi-dinamikus	Diszkrét	Több
Póker	Részben	Sztochasztikus	Sorozat	Statikus	Diszkrét	Több
Ostábla	Teljesen	Sztochasztikus	Sorozat	Statikus	Diszkrét	Több
Taxivezetés	Részben	Sztochasztikus	Sorozat	Dinamikus	Folytonos	Több
Orvosi diagnosztika	Részben	Sztochasztikus	Sorozat	Dinamikus	Folytonos	Egy
Képfeldolgozó rendszer	Teljesen	Determinisztikus	Epizód	Szemi-dinamikus	Folytonos	Egy
Alkatrész-felvevő robot	Részben	Sztochasztikus	Epizód	Dinamikus	Folytonos	Egy
Olajfinomító-vezérlő	Részben	Sztochasztikus	Sorozat	Dinamikus	Folytonos	Egy
Interaktív angol tanár	Részben	Sztochasztikus	Sorozat	Dinamikus	Diszkrét	Több

A könyvhöz tartozó példaprogramtár *(aima.cs.berkeley.edu)* tartalmaz számos környezetimplementációt, egy általános célú környezetszimulátorral együtt, amely egy vagy több ágenst helyez el egy szimulált környezetben, megfigyeli a viselkedésüket az időben, és kiértékeli őket egy adott teljesítménymérték szerint. Ilyen kísérleteket gyakran nem egy, hanem több, egy **környezetosztály**ból (**environment class**) származó környezetre hajtanak végre. Például egy taxisofőr szimulált forgalomban történő értékelésére több szimulációt futtatnánk különböző forgalommal, világítással, és időjárási viszonyokkal. Ha az ágenst egyedi esetre terveztük, kihasználhatjuk az adott környezet speciális tulajdonságait, de nem találhatjuk meg a vezetés általában jó módját. Ezért a programtár tartalmaz egy **környezetgenerátor**t (**environment generator**) minden környezetosztályra, amelyik kiválaszt bizonyos környezeteket (bizonyos valószínűséggel) az ágens futtatására. Például a porszívó környezetgenerátor véletlenszerűen állítja be a piszok mintázatát és az ágens helyét. Ezek után az ágens átlagos teljesítményére vagyunk kíváncsiak az adott környezetosztályra nézve. Egy racionális ágens az adott környezetosztályra maximalizálja ezt az átlagos teljesítményt. A 2.7. és 2.12. közötti feladatok végigvisznek egy környezetosztály kialakításán és abban többféle ágens kiértékelésén.

4. Az intelligens ágensek struktúrája

Az ágenseket eddig *viselkedésük* leírásán keresztül vizsgáltuk – azon cselekvés alapján, amelyet egy adott észlelési sorozat hatására végrehajtanak. Most neki kell esnünk a kemény diónak: hogyan működnek belül. A mesterséges intelligencia feladata az **ágensprogram** (**agent program**) megtervezése: egy függvényé, amely megvalósítja az észlelések és a cselekvések közötti leképezést. Feltételezzük, hogy ez a program valamiféle fizikai érzékelőkkel és beavatkozókkal ellátott számítóeszközön fog futni – ezt **architektúrá**nak (**architecture**) nevezzük:

```
ágens = architektúra + program
```

Nyilvánvalóan a kiválasztott programunknak olyannak kell lennie, ami megfelelő az architektúra számára. Ha a program például *Sétál* jellegű akciókat fog javasolni, akkor jobb, ha az architektúrának vannak lábai. Az architektúra lehet egyszerű számítógép, vagy lehet egy robotautó számos fedélzeti számítógéppel, kamerával és más érzékelőkkel. Általánosságban az architektúra a szenzoroktól érkező észleléseket elérhetővé teszi a program számára, futtatja a programot, és cselekvéseit létrejöttük pillanatában a beavatkozók felé továbbítja. E könyv túlnyomó része az ágensprogramok tervezéséről szól, bár a 24. és 25. fejezet kifejezetten az érzékelőkkel és a beavatkozókkal foglalkozik.

4.1. Ágensprogramok

Created by XMLmind XSL-FO Converter.

A könyvben tervezett ágensprogramok mindannyian azonos vázzal rendelkeznek: bemenetként fogadják az aktuális észleléseket a szenzoroktól, és visszaküldenek egy cselekvést a beavatkozókhoz.[6] Vegyük észre a különbséget az ágensprogram, amely az aktuális észlelést veszi bemenetként, és az ágensfüggvény között, amely a teljes észlelési történetet fogadja. Az ágensprogram azért fogadja csak az aktuális észlelést bemenetként, mert csak ez érkezik a környezettől – ha az ágens cselekedetei a teljes észlelési sorozattól függnek, az ágensnek emlékeznie kell az észlelésekre.

Az ágensprogramot egy egyszerű pszeudokódú nyelv segítségével fogjuk leírni, amit a B) függelék definiál. (Az interneten elérhető kódtár valóságos programozási nyelveken írt megvalósításokat tartalmaz.) Például a 2.7. ábra egy egészen egyszerű ágenst mutat, amely nyomon követi az észlelési sorozatot, és felhasználja egy akciókat tartalmazó táblázat megcímzésére annak eldöntésére, hogy mit csináljon. A táblázat reprezentálja az ágensfüggvényt, amit az ágensprogram testesít meg. Ahhoz, hogy ily módon egy racionális ágenst építsünk, nekünk, mint tervezőknek, olyan táblázatot kell megalkotnunk, amely minden lehetséges észlelési sorozathoz a megfelelő cselekvést tartalmazza.

2.7. ábra - A TÁBLÁZAT-VEZÉRLÉSŰ-ÁGENS program minden egyes új észlelésre meghívódik, és minden alkalommal visszaad egy cselekvést. Saját adatstruktúrájában követi nyomon az észlelési sorozatot.

```
function TÁBLÁZAT-VEZÉRLÉSŰ-ÁGENS(észlelés) returns egy cselekvés
  static: észlelések, egy sorozat, kezdetben üres
          táblázat, egy táblázat, az észlelési sorozat indexeli, kezdetben teljesen feltöltött

  csatold az észlelést az észlelések végére
  cselekvés ← KIKERESÉS(észlelések, táblázat)
  return cselekvés
```

Példaértékű annak megfontolása, hogy az ágensek tervezésének táblázatvezérelt megközelítése miért van kudarcra ítélve. Legyen P a lehetséges észlelések halmaza és T az ágens élettartama (az általa vett észlelések teljes száma). A táblázat $\sum_{t=1}^{T} |\mathbf{P}|^t$ elemet fog tartalmazni. Vegyük példaként az automata taxit: az egyetlen kamerától érkező vizuális bemenet körülbelül 27 megabájt/másodperc sebességgel érkezik (30 kocka másodpercenként, 640 x 480 képpont 24 bit színinformációval). Ez alapján egyórányi vezetéshez olyan táblázatot kapunk, amely több mint $10^{250\,000\,000\,000}$ bejegyzést tartalmaz. Még a sakkhoz tartozó táblázat is – amely a való világ egy kicsiny, jól viselkedő részlete – legalább 10^{150} bejegyzést tartalmazna. Ezen táblázatok ijesztő mérete (a megfigyelhető világegyetemben az atomok száma kevesebb mint 10^{80}) azt jelenti, hogy (a) ebben az univerzumban egyetlen fizikai ágensnek sem lesz elég helye a táblázat tárolására, (b) a tervezőnek nem lenne elég ideje a táblázat elkészítéséhez, (c) egyetlen ágens sem lenne képes a táblázat helyes bejegyzéseit megtanulni saját tapasztalatából, és (d) még ha a környezet eléggé egyszerű is egy megvalósítható méretű táblázathoz, a tervezőnek még akkor sincs segítsége ahhoz, hogyan töltse ki a táblázat bejegyzéseit.

Mindezek ellenére, a TÁBLÁZAT-VEZÉRLÉSŰ-ÁGENS *megteszi* azt, amit akarunk: megvalósítja a kívánt ágensfüggvényt. Az MI alapvető kihívása, hogy hogyan írjunk olyan programot, amely – a lehetőségek határain belül – nagyszámú táblázatbejegyzések helyett kisméretű programkóddal produkál racionális viselkedést. Sok példánk van arra, hogy ez más területeken megvalósítható: például az 1970-es évek előtt a mérnökök és az iskolás gyerekek által használt hatalmas négyzetgyöktáblázatokat ma már egy elektronikus kalkulátorokon futó ötsoros program váltotta fel, amely a Newton-módszert alkalmazza. A kérdés az, hogy megteheti-e az MI azt az általános intelligens viselkedésre, amit Newton tett meg a négyzetgyökökre? Abban hiszünk, hogy a válasz igen.

E fejezet hátralevő részében négy alapvető ágensprogramtípust vázolunk, amely megtestesíti a szinte minden intelligens rendszer mögött meghúzódó alapelveket:

- Egyszerű reflexszerű ágensek.
- Modellalapú reflexszerű ágensek.

[6] Vannak más lehetőségek is az ágensváz számára, például leírhatjuk az ágensprogramot **korutin**ok (**coroutine**s) formájában, amelyek a környezethez képest aszinkron módon futnak. Minden ilyen korutin rendelkezik egy bemeneti és egy kimeneti porttal, és egy olyan ciklust tartalmaz, amelyik beolvassa a bemeneti portról az észleléseket, és kiírja a cselekvéseket a kimeneti portra.

Created by XMLmind XSL-FO Converter.

- Célorientált ágensek.
- Hasznosságorientált ágensek.

Ezek után általánosságban elmagyarázzuk, hogyan alakíthatjuk át ezeket *tanuló ágensekké*.

4.2. Egyszerű reflexszerű ágensek

A legegyszerűbb fajtájú ágens az **egyszerű reflexszerű ágens** (**simple reflex agent**). Ezek az ágensek az *aktuális* észlelés alapján választják ki a cselekvéseket, figyelmen kívül hagyva az észlelési történet többi részét. Például a porszívó ágens, amelynek ágensfüggvényét a 2.3. ábra táblázatos formában mutatja, egy egyszerű reflexszerű ágens, mivel döntései csak a jelenlegi helyszínen és azon alapulnak, hogy ott van-e piszok. A 2.8. ábra mutatja ezen ágens programját.

Vegyük észre, hogy az ágensprogram igen kicsi a megfelelő táblázathoz képest. A legnyilvánvalóbb méretcsökkentés az észlelési történet figyelmen kívül hagyásából következik, amely a lehetőségek számát 4^T-ről 4-re csökkenti. Egy további, kisebb csökkenés abból a tényből fakad, hogy amikor az aktuális négyzet piszkos, az akció nem függ a helyszíntől.

2.8. ábra - Egy egyszerű reflexszerű ágens programja a kétállapotú porszívókörnyezetben. A 2.3. ábrán megadott ágensfüggvényt valósítja meg.

```
function REFLEXSZERŰ-PORSZÍVÓ-ÁGENS(helyszín, állapot) returns egy cselekvés

  if állapot = Piszkos then return Felszívás
  else if helyszín = A then return Jobbra
  else if helyszín = B then return Balra
```

Képzeljük magunkat az automata taxisofőr helyébe! Ha az előttünk haladó autó fékez, és a féklámpái kigyulladnak, akkor észre kell vennünk, és el kell kezdenünk fékezni. Más szavakkal, valamilyen feldolgozás zajlik a képi bemeneten, hogy megállapítsunk egy feltételt, amit az „előző autó fékez”-nek hívunk; azután ez kivált valamilyen kialakított kapcsolatot az ágens programjában a „kezdj fékezni” cselekvéshez. Ezt a kapcsolatot **feltétel–cselekvés szabály**nak (**condition–action rule**)[7] hívjuk, és

```
ha az-előző-autó-fékez akkor kezdj-fékezni
```

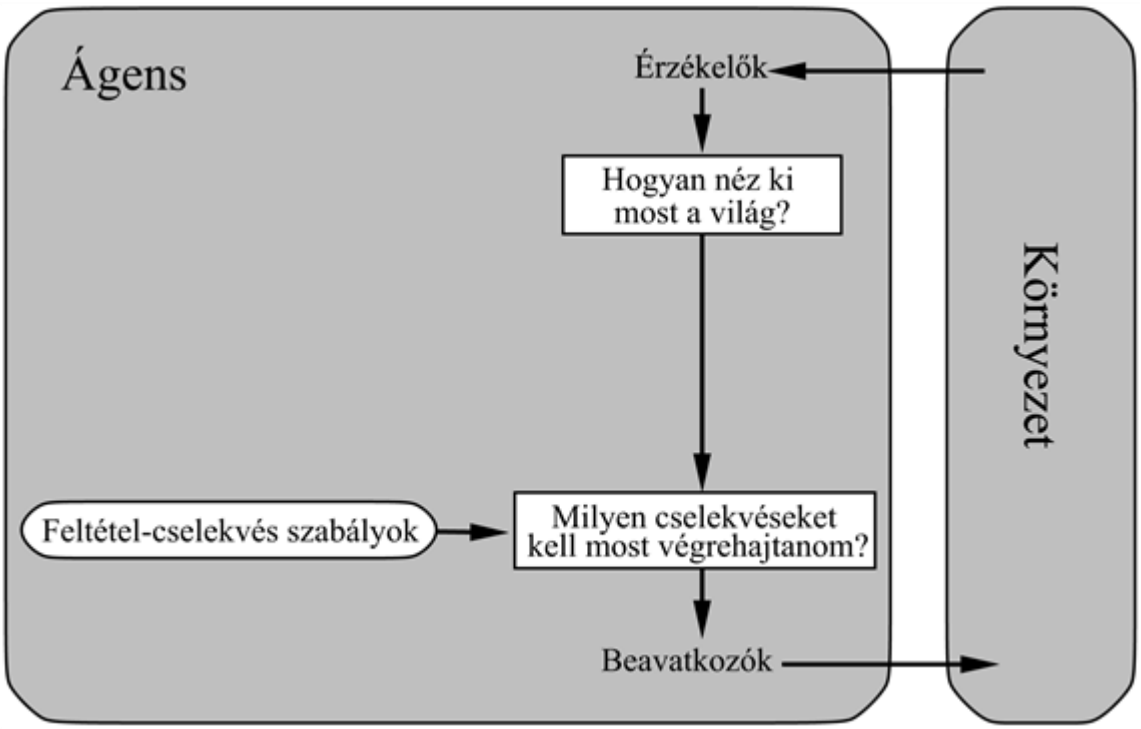
formában írjuk. Az embereknek sok hasonló kapcsolatuk van, egy részük tanult válasz (ami a vezetést illeti), míg más részük feltétlen reflex (mint például a pislogás, amikor valami megközelíti a szemünket). A későbbiekben több különböző módot is látni fogunk arra, hogy ilyen kapcsolatok hogyan tanulhatók és valósíthatók meg.

A 2.8. ábra programja egy egyedi porszívókörnyezetre specifikus. Egy általánosabb és rugalmasabb megközelítés az, hogy először egy általános célú értelmezőt építünk feltétel–cselekvés szabályokra, majd szabályhalmazokat hozunk létre specifikus feladatkörnyezetek számára. A 2.9. ábra ezen általános program struktúráját adja meg sematikus formában, megmutatva, hogy a feltétel–cselekvés szabályok hogyan teszik lehetővé az ágens számára az észlelések és cselekvések összekötését. (Ne aggódjon, ha ez túl egyszerűnek tűnik: hamarosan sokkal érdekesebbé válik.) Az ágens döntési folyamatának aktuális belső állapotát négyszögekkel jelöljük, a folyamatban felhasznált háttértudás reprezentálására pedig oválisokat használunk. Az ágensprogram, amely szintén igen egyszerű, a 2.10. ábrán látható. A `BEMENET-FELDOLGOZÁS` függvény a bemenetből állítja elő az aktuális állapot absztrakt leírását, a `SZABÁLY-ILLESZTÉS` függvény pedig az első olyan szabályt adja vissza a szabályok halmazából, amely illeszkedik az adott állapot leírására. Vegyük észre, hogy a „szabályok” és az „illeszkedés” fogalmakkal történő leírás tisztán koncepcionális – egyedi megvalósítások akár egy logikai áramkört megvalósító logikai kapu halmazból is állhatnak.

2.9. ábra - Egy egyszerű reflexszerű ágens sematikus diagramja

[7] A kapcsolatot **szituáció–cselekvés** (**situation–action rule**), **produkciós** (**production rule**) vagy **ha–akkor szabály**nak (**if–then rule**) is nevezik.

Created by XMLmind XSL-FO Converter.



Fontos

Az egyszerű reflexszerű ágenseknek megvan az az értékelendő tulajdonsága, hogy egyszerűek, ugyanakkor igen korlátozott intelligenciájúnak bizonyulnak. A 2.10. ábrán látható ágens *csak akkor* fog működni, *ha a helyes döntés kizárólag az aktuális észlelés alapján meghozható – azaz akkor, ha a környezet teljesen megfigyelhető*. Már a megfigyelhetőség legkisebb hiánya is komoly problémát okozhat. Például a korábban bemutatott fékezési szabály feltételezi, hogy az *előző-autó-fékez* feltétel megállapítható az aktuális észlelésből – az aktuális videoképből – amennyiben az előző autónak középre szerelt féklámpája van. Sajnálatos módon, a régebbi modelleken különbözőképpen helyezkednek el a hátsó helyzetjelzők, a féklámpák és az indexlámpák, így egyetlen kép alapján nem mindig lehetséges azt megállapítani, hogy az autó fékez-e. Egy egyszerű reflexszerű ágens, amely egy ilyen autó mögött halad, vagy folyamatosan szükségtelenül fékezne, vagy ami még rosszabb, sosem fékezne.

2.10. ábra - Egy egyszerű reflexszerű ágens. A jelenlegi szituációra (amit az észlelés határoz meg) illeszkedő feltételű szabály megkeresésével, majd a hozzá tartozó cselekvés végrehajtásával működik.

```
function EGYSZERŰ-REFLEXSZERŰ-ÁGENS(észlelés) returns egy cselekvés
  static: szabályok, feltétel–cselekvés szabályok halmaza

  állapot ← BEMENET-FELDOLGOZÁS(észlelés)
  szabály ← SZABÁLY-ILLESZTÉS(állapot, szabályok)
  cselekvés ← SZABÁLY-CSELEKVÉS[szabály]
  return cselekvés
```

Hasonló problémát láthatunk felbukkanni a porszívóvilágban. Tegyük fel, hogy egy egyszerű reflexszerű ágenst megfosztunk a helyzetérzékelő szenzorától, és csak piszokérzékelője van. Egy ilyen ágensnek csak két észlelése van: [*Koszos*] és [*Tiszta*]. *Felszívhat* válaszul a [*Koszos*]-ra; de mit tegyen, ha [*Tiszta*] az észlelés? A *Balra* mozgás (mindörökre) sikertelen lesz, ha például az *A* négyzetben kezd, a *Jobbra* mozgás pedig (mindörökre) sikertelen, ha például a *B* négyzetben kezd. A végtelen ciklusok gyakran elkerülhetetlenek részlegesen megfigyelhető környezetben működő egyszerű reflexszerű ágensek számára.

Created by XMLmind XSL-FO Converter.

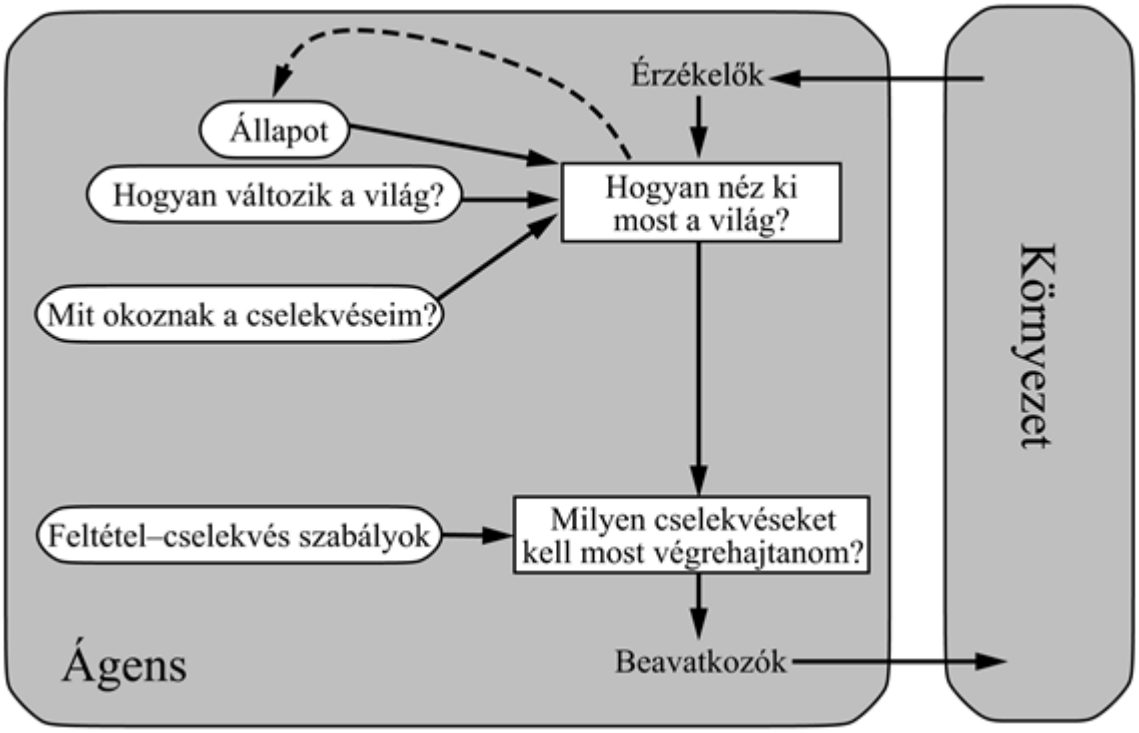
Elképzelhető menekvés a végtelen ciklusokból, ha az ágens **randomizál**hatja (**randomize**) cselekvéseit. Például ha a porszívó ágens [*Tisztá*]-t észlel, feldobhat egy érmét a *Bal* és *Jobb* közötti választáshoz. Könnyű megmutatni, hogy az ágens el fogja érni a másik négyzetet átlagosan két lépésen belül. Ezután, ha a négyzet piszkos, feltakarítja, és a tisztítási feladat befejeződik. Ily módon egy randomizált egyszerű reflexszerű ágens túlteljesíthet egy determinisztikus egyszerű reflexszerű ágenst.

A 2.3. alfejezetben említettük, hogy a helyes formájú randomizált viselkedés lehet racionális bizonyos többágenses környezetekben. Egyágenses környezetekben a randomizáció általában *nem* racionális. Egy hasznos trükk, amely segítheti az egyszerű reflexszerű ágenst bizonyos szituációkban, de a legtöbb esetben ennél sokkal jobbat érhetünk el kifinomultabb determinisztrikus ágensekkel.

4.3. Modellalapú reflexszerű ágensek

A részleges megfigyelhetőség kezelésének leghatékonyabb módja, ha az ágens *nyomon követi a világ jelenleg nem látható részét*. Azaz, az ágensnek nyilván kell tartania valamiféle **belső állapot**ot (**internal state**), amely az észlelési történeten alapul, és így a jelenlegi állapot nem megfigyelt aspektusainak legalább egy részét tükrözi. A fékezési probléma esetében a belső állapot nem túlságosan terjedelmes – csak a kamerától származó előző képkocka, amely lehetővé teszi az ágens számára annak megállapítását, hogy a jármű két szélén levő két piros lámpa egyszerre gyullad és alszik-e ki. Más vezetési feladatokhoz, mint például a sávváltáshoz, az ágensnek nyomon kell követnie hol tartózkodik a többi autó, amennyiben nem látja mindegyiket egyszerre.

2.11. ábra - Belső állapottal rendelkező reflexszerű ágens



2.12. ábra - A modellalapú reflexszerű ágens. Nyomon követi a világ jelenlegi állapotát egy belső modellben. Ezek után a reflexszerű ágenshez hasonlóan választ egy cselekvést.

Created by XMLmind XSL-FO Converter.

```
function REFLEXSZERŰ-ÁGENS-ÁLLAPOT(észlelés) returns egy cselekvés
  static: állapot, a világ jelenlegi állapotának leírása
          szabályok, feltétel–cselekvés szabályok halmaza
          cselekvés, a legutolsó cselekvés, kezdetben semmi

  állapot ← ÁLLAPOT-FRISSÍTÉS(állapot, cselekvés, észlelés)
  szabály ← SZABÁLY-ILLESZTÉS(állapot, szabályok)
  cselekvés ← SZABÁLY-CSELEKVÉS[szabály]
  return cselekvés
```

Ezen belső állapot frissítése az idő előrehaladtával kétféle tudás beépítését igényli az ágensprogramba. Először is szükségünk van olyan információra, hogy a világ hogyan alakul az ágenstől függetlenül – például egy előző autó általában közelebb lesz most, mint egy pillanattal azelőtt. Másodszor, szükségünk van valamilyen információra arról, hogy az ágens saját cselekvései hogyan változtatják meg a világot – például amikor az ágens a kormányt az óramutató járásának megfelelően tekeri, az autó jobbra fordul, vagy öt percig északnak tartó vezetés után egy autópályán általában körülbelül öt mérfölddel lesz északabbra, mint ahol öt perce volt. Ezt, a „világ működésének mikéntjéről” szóló tudást – akár egyszerű logikai áramkörökkel, akár teljes tudományos elméletekkel valósítjuk meg – a világ **modell**jének (**model**) hívjuk. Egy ilyen modellt használó ágenst pedig **modellalapú ágens**nek (**model-based agent**).

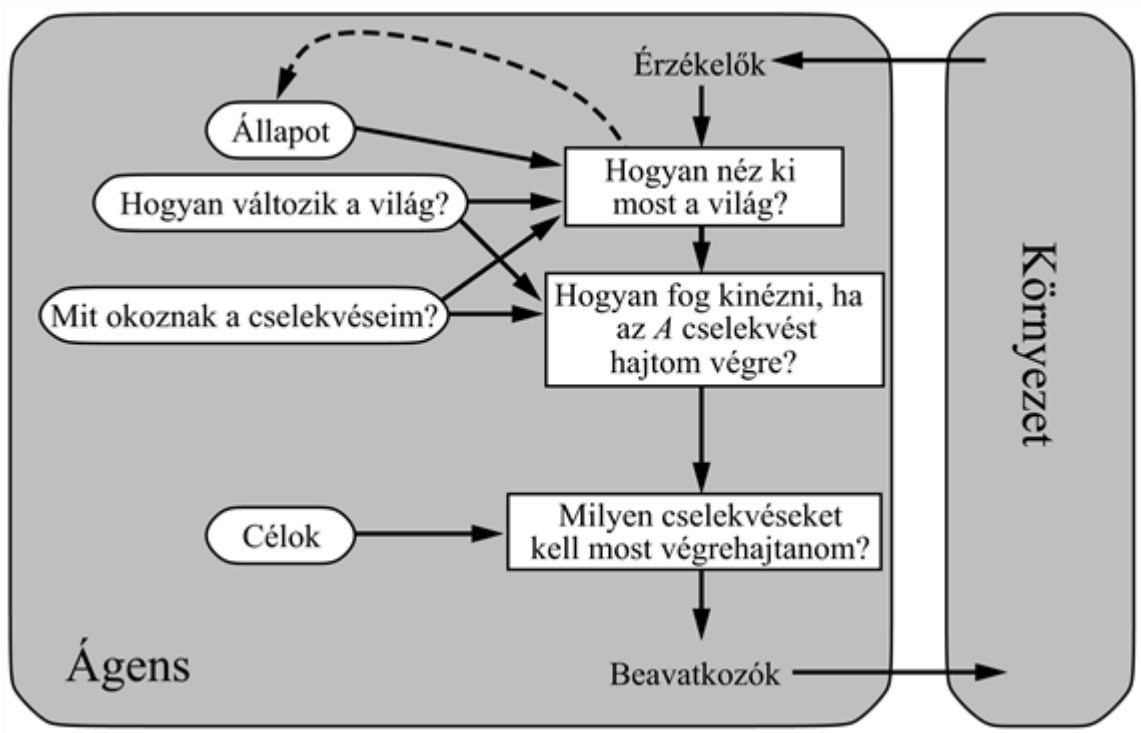
A 2.11. ábra szemlélteti a belső állapottal rendelkező reflexszerű ágens struktúráját, megmutatva azt, ahogyan a jelenlegi észlelés a régi belső állapottal kombinálódva létrehozza a jelenlegi állapot frissített leírását. Az ágensprogram a 2.12. ábrán látható. Az érdekes rész az `ÁLLAPOT-FRISSÍTÉS` eljárás, amely a belső állapot leírásának létrehozásáért felelős. Az új észleléseknek az állapotról szóló tudás tükrében történő értelmezése mellett az ágensprogram nyomon követi a világ láthatatlan részeit a világ fejlődését leíró információ segítségével, az ágensprogramnak pedig azt is tudnia kell, hogy az ágens cselekvései milyen hatással lesznek a világ állapotára. Részletesebb példák a 10. és 17. fejezetben találhatók.

4.4. Célorientált ágensek

A környezet jelenlegi állapotának ismerete nem mindig elég annak eldöntéséhez, hogy mit tegyünk. Például egy kereszteződésben a taxi mehet egyenesen, de fordulhat jobbra vagy balra is. A helyes döntés attól függ, hogy a taxi hova szeretne eljutni. Más szavakkal, a jelenlegi állapot leírása mellett az ágensnek valamiféle **cél**- (**goal**) információval is rendelkeznie kell, amely leírja a kívánatos helyzeteket – például azt, hogy az utas jusson el a célállomásig. Az ágensprogram ezt összevetheti a lehetséges cselekvések eredményeiről szóló információkkal (ugyanazokkal, amelyeket a reflexszerű ágens esetében a belső állapot frissítésekor használt) annak érdekében, hogy a céljához vezető cselekvést meghatározza. A 2.13. ábra bemutatja a célorientált ágens felépítését.

2.13. ábra - Egy modellalapú, célorientált ágens. Nyomon követi a világ állapotát és az elérendő célok halmazát is, és kiválaszt egy cselekvést, amely (végső soron) céljainak eléréséhez vezet.

Created by XMLmind XSL-FO Converter.



Néha a célorientált cselekvés választás egyszerű, amikor a cél teljesülése azonnal egyetlen cselekvéssel elérhető. Máskor sokkal trükkösebb, amikor az ágensnek befordulások és visszakanyarodások sorozatát kell megfontolnia a cél eléréséhez. A **keresés (search)** (lásd 3–6. fejezet) és a **tervkészítés (planning)** (lásd 11–12. fejezet) a mesterséges intelligencia azon területei, amelyek az ágens céljait elérő cselekvéssorozat megtalálásával foglalkoznak.

Vegyük észre, hogy ez a fajta döntéshozatal alapjaiban különbözik az előzőkben ismertetett feltétel–cselekvés szabályoktól, ezek magukban foglalják a jövő figyelembevételét – mind a „Mi fog történni, ha ezt és ezt teszem?”, mind a „Boldoggá tesz ez majd engem?” kérdéseket. Reflexszerű ágensek tervezésében ezeket explicit módon nem használjuk, mivel a tervező különböző esetekre előre meghatározta a helyes cselekvést. A reflexszerű ágens fékez, ha féklámpát lát kigyulladni. Célorientált elméletben gondolkodhat úgy, hogyha az előző autó féklámpái égnek, akkor az le fog lassulni. Az alapján, ahogy a világ általában működik, az egyetlen cselekvés, amely eléri azt a célt, hogy ne koccanjon neki más autóknak, a fékezés.

Bár a célorientált ágens kevésbé tűnik hatékonynak, sokkal rugalmasabb, mivel a döntéseit alátámasztó tudás explicit módon megjelenik, és így módosítható. Ha elkezd esni az eső, frissítheti a tudását arról, hogy a fékek mennyire lesznek hatékonyak – ez automatikusan az összes releváns viselkedés olyan megváltozását eredményezi, hogy azok az új körülményhez igazodjanak. Reflexszerű ágensek esetében ugyanakkor, a feltétel–cselekvés szabályok nagy számát kellene átírnunk. Természetesen a célorientált ágens különböző úti célok elérésében is rugalmasabb. Egyszerűen egy új cél megadásával a célorientált ágenstől egy új viselkedés megjelenését kaphatjuk. A reflexszerű ágens „mikor fordulj” és „mikor menj egyenesen” szabályai csak egy cél esetében fognak működni: ki kell őket cserélni ahhoz, hogy egy új helyre menjen.

4.5. Hasznosságorientált ágensek

A környezetek többségében a célok önmagukban nem elegendőek jó minőségű viselkedés létrehozásához. Például többféle cselekvéssorozat vezet a taxi végállomásának – és így a céljának – eléréséhez, de némelyik gyorsabb, biztonságosabb, megbízhatóbb vagy olcsóbb, mint mások. A célok csak durva különbséget tesznek „boldog” és „boldogtalan” állapotok között, egy általánosabb teljesítménymérték ezzel szemben a világ állapotainak (vagy állapotsorozatainak) megfelelő összehasonlítását tehetné lehetővé, azt megadva, hogy pontosan mennyire tennék boldoggá az ágenst, ha elérné azokat. Mivel a „boldogság” nem hangzik túl tudományosan, ezért a szokásos terminológia azt mondja, hogy a világ egyik állapota előnyösebb egy másikhoz képest, ha nagyobb a **hasznossága (utility)**[8] az ágens számára.

[8] A „hasznosság” szó itt a „hasznosság minősége” kifejezésre utal, nem az áram- vagy vízszolgáltatókra. (Az angol *utility* szó többes jelentésű, a szerző ezért magyarázza meg helyes értelmezését a lábjegyzetben. – *A ford.*)

Created by XMLmind XSL-FO Converter.

A **hasznosságfüggvény** (**utility function**) egy állapotot (vagy állapotok egy sorozatát) egy olyan valós számra képezi le, amelyik a hozzá rendelt boldogság fokát írja le. A hasznosságfüggvény teljes meghatározása kétféle olyan helyzetben tesz lehetővé racionális döntéseket, amikor a célok erre alkalmatlanok. Egyrészt, amikor egymásnak ellentmondó célok vannak, amelyeknek csak egy része érhető el (mint például a biztonságosság és a sebesség), akkor a hasznosságfüggvény meghatározza a helyes kompromisszumot. Másrészt, amikor több cél van, amelyek elérésére az ágens törekedhet, és egyikük sem érthető el teljes bizonyossággal, a hasznosság olyan módszert ad, amivel a siker valószínűsége a célok fontosságához mérhető.

A 16. fejezetben megmutatjuk, hogy minden racionális ágensnek úgy kell működnie, *mintha* rendelkezne hasznosságfüggvénnyel, amelynek várható értékét maximalizálni akarja. Egy ágens, amelynek van egy *explicit* hasznosságfüggvénye, ily módon racionális döntéseket hozhat, és ezt egy általános algoritmussal teheti meg, amely nem függ a maximalizálandó specifikus hasznosságfüggvénytől. Ezáltal a racionalitás „globális” definíciója – amely azon ágensfüggvényeket tekinti racionálisnak, amelyek a legnagyobb teljesítményt hozzák – átváltozik a racionáliságens-tervezés egy „helyi” kényszerévé, amely kifejezhető egy egyszerű programban.

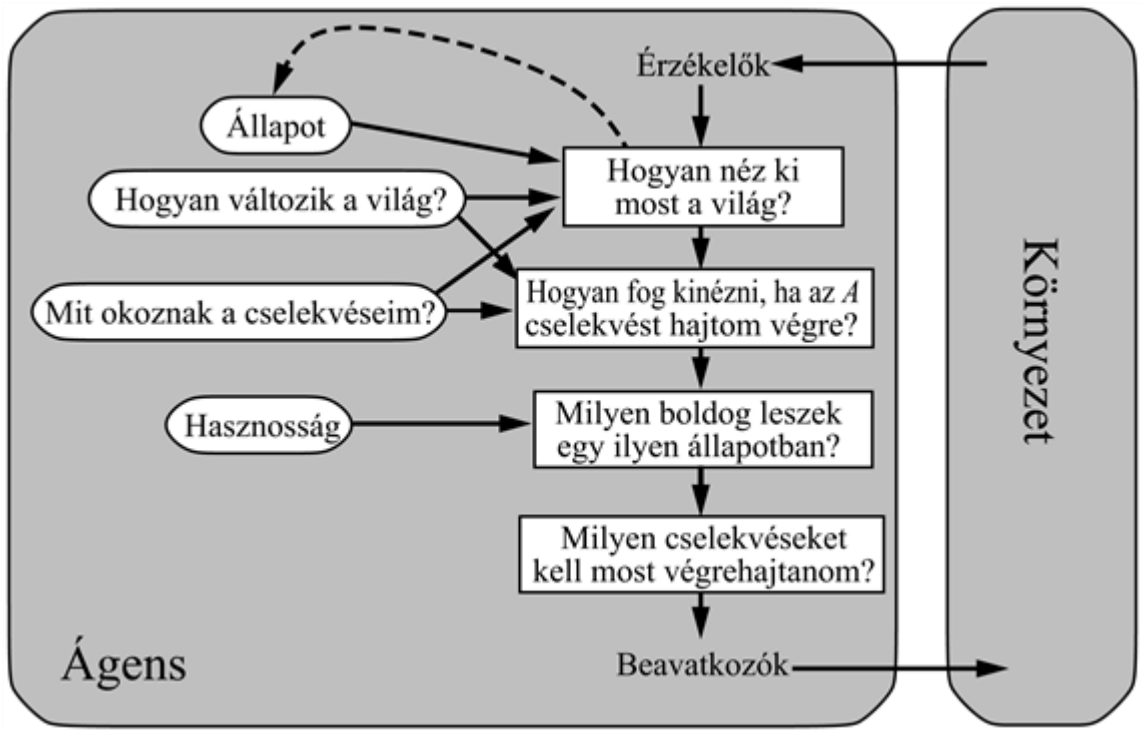
A hasznosságalapú ágens struktúrája a 2.14. ábrán látható. Valódi hasznosságalapú ágens programok az V. részben szerepelnek, ahol olyan döntéshozó ágenseket tervezünk, amelyeknek a részlegesen megfigyelhető környezetekben elkerülhetetlen bizonytalanságot is kezelniük kell.

4.6. Tanuló ágensek

Leírtunk többféle módszert arra, hogy ágensprogramok hogyan választják ki cselekvéseiket. Nem magyaráztuk meg azonban eddig, hogy *hogyan jön létre* az ágensprogram. Híres korai cikkében Turing (Turing, 1950) azzal az ötlettel foglalkozik, hogy intelligens gépeit ténylegesen kézzel programozza. Megbecsüli, mennyi munkát vehet ez igénybe, és megállapítja, hogy „valamilyen eredményesebb módszer kívánatosnak látszik”. Az általa javasolt módszer az, hogy építsünk tanuló gépeket, majd tanítsuk ezeket. Az MI sok területén ez ma a javasolt módszer korszerű rendszerek építésére. A tanulásnak van egy másik előnye is, ahogy megállapítottuk korábban: lehetővé teszi az ágens számára, hogy kezdetben ismeretlen környezetben működjön, és kompetensebbé váljon, mint ahogy azt kezdeti tudása lehetővé tette volna. Ebben a fejezetben röviden bemutatjuk a tanuló ágensek főbb elveit. A könyv majdnem minden fejezetében rámutatunk a tanulás lehetőségeire és módszereire az egyes ágensfajtákban. A VI. rész merül el mélyebben a különféle tanulási algoritmusokban.

2.14. ábra - Egy modellalapú, hasznosságalapú ágens. A világ modellje mellett egy hasznosságfüggvényt is alkalmaz, amely a világ állapotaihoz rendelt preferenciáit méri. Ezek után olyan cselekvést választ, amely a legjobb várható hasznossághoz vezet, amit az összes lehetséges végállapot előfordulási valószínűségével súlyozott átlagolásával számít ki.

Created by XMLmind XSL-FO Converter.

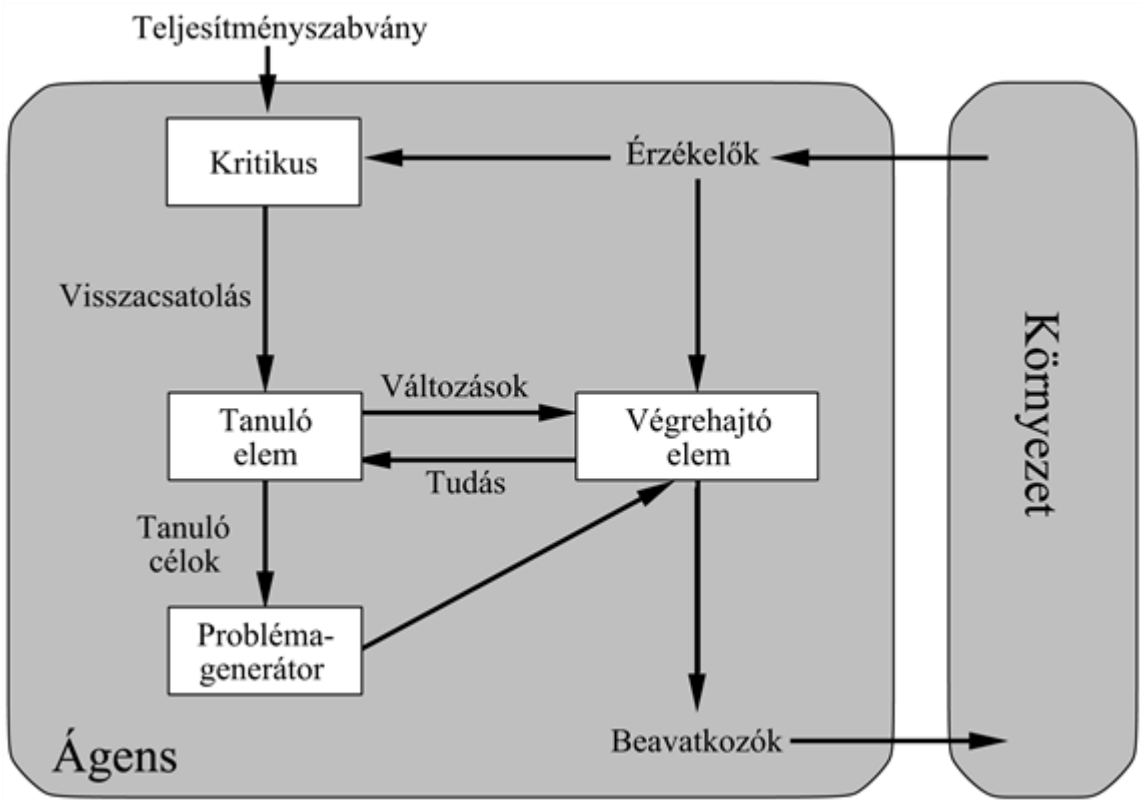


Egy tanuló ágens négy koncepcionális komponensre bontható fel, ahogy azt a 2.15. ábra mutatja. A legfontosabb különbségtétel a javításokért felelős **tanuló elem** (**learning element**) és a külső cselekvések kiválasztásáért felelős **végrehajtó elem** (**performance element**) között van. A végrehajtó elem az, amit eddig a teljes ágensnek tekintettünk: ez végzi az észleléseket, és ez dönt a cselekvésekről. A tanuló elem a **kritikus**tól (**critic**) kapott, az ágens működéséről szóló visszajelzést használja annak megállapítására, hogy a végrehajtó elemet hogyan kell módosítani annak érdekében, hogy a jövőben jobban működjön az ágens.

A tanuló elem tervezése nagyrészt a végrehajtó elem tervezésétől függ. Amikor egy olyan ágenst próbálunk tervezni, amely megtanul egy bizonyos képességet, az első kérdés nem a „hogyan fogom én ezt megtanulni?”, hanem a „milyen teljesítményelemre lesz az ágensemnek szüksége ennek végrehajtásához, ha már megtanulta, hogyan kell?”. Ha adott egy megtervezett ágens, tanuló eljárások az ágens minden részének javításához készíthetők.

2.15. ábra - Tanuló ágensek egy általános modellje

Created by XMLmind XSL-FO Converter.



A kritikus megmondja a tanuló elemnek, hogy az ágens milyen jól működik egy rögzített teljesítményszabványhoz viszonyítva. A kritikus szükséges, hiszen az észlelések önmagukban nem jelzik az ágens sikerességét. Például egy sakk program fogadhat egy olyan észlelést, hogy mattot adott az ellenfelének, de szüksége van egy teljesítményszabványra, hogy tudja, ez egy jó dolog. Az észlelés önmagában nem mondja meg ezt. Fontos, hogy a teljesítményszabvány rögzített legyen. Koncepcionálisan tekinthető úgy, mintha az ágensen teljesen kívül lenne, mivel az ágens nem módosíthatja azt saját viselkedésének megjavítása érdekében.

A tanuló ágens utolsó komponense a **problémagenerátor** (**problem generator**). Ennek feladata, hogy olyan cselekvéseket javasoljon, amelyek új és informatív tapasztalatokhoz vezetnek. Az a helyzet, hogy ha a végrehajtó elem a saját feje után menne, akkor adott tudása alapján mindig a legjobb cselekvéseket választaná. De ha az ágens hajlandó egy kis felfedezésre, és rövid távon talán néhány szuboptimális cselekvésre, hosszú távon sokkal jobb cselekvéseket fedezhet fel. A problémagenerátor feladata, hogy ilyen felfedező cselekvéseket javasoljon. Ezt teszik a tudósok is, amikor kísérleteket végeznek. Galilei nem gondolta, hogy a pisai torony tetejéből köveket leejteni önmagában értékes dolog. Sem a köveket nem próbálta összetörni, sem a szerencsétlen arra járók agyát nem akarta „megváltoztatni”. Célja saját agyának megváltoztatása volt azáltal, hogy az objektumok mozgásának egy jobb elméletét ismerje fel.

Hogy az általános tervet konkrétabbá tegyük, térjünk vissza az automata taxi példájához. A végrehajtó elem tartalmazza mindazon tudást és eljárások halmazát, amelyek a taxivezetés cselekvéseinek kiválasztásához szükségesek. A taxi ezen végrehajtó elem segítségével megy ki az utakra és közlekedik. A kritikus megfigyeli a világot, és továbbadja az információt a tanuló elemnek. Például miután a taxi három forgalmi sávon keresztül egy gyors balkanyart vett, a kritikus megfigyeli a többi vezető által használt sokkoló nyelvezetet. Ebből a kísérletből a tanuló elem képes egy szabályt formálni, miszerint ez egy rossz cselekvés volt, és a végrehajtó elem módosul az új szabály beillesztésével. A problémagenerátor megfigyelheti a viselkedés bizonyos területeit, melyek javításra szorulnak, és kísérleteket javasol, mint például a fékek kipróbálását különböző útfelszíneken és különböző viszonyok között.

A tanuló elem módosíthatja az ágensdiagramokon (2.9., 2.11., 2.13. és 2.14. ábrák) „tudás” komponensek bármelyikét. A legegyszerűbb esetek magukban foglalják az észlelési szekvenciából történő direkt tanulást. Az egymást követő környezeti állapot párok megfigyelése lehetővé teszi az ágens számára annak megtanulását, hogy a „világ hogyan változik”, és cselekvései eredményeinek megfigyelése lehetővé teszi a „mit okoznak a cselekvéseim” megtanulását. Például ha a taxi vezetője adott erővel fékez, mikor a taxi nedves úton halad, akkor hamarosan tapasztalni fogja, ténylegesen mekkora lassulást ért el. Világos, hogy ez a két tanulási feladat sokkal nehezebb, ha a környezet csak részlegesen figyelhető meg.

Created by XMLmind XSL-FO Converter.

Az előző bekezdésben szereplő tanulási formák nem igényelnek hozzáférést külső teljesítményszabványhoz – bizonyos értelemben a mérték univerzális: a kísérletekkel összhangban levő előrejelzések készítése. A helyzet némiképp bonyolultabb egy hasznosságalapú ágens számára, amelyik a hasznosságinformációt szeretné megtanulni. Például tegyük fel, hogy a taxisofőr ágens nem kap borravalót az utasoktól, akiket az út folyamán teljesen összerázott. A külső teljesítményszabványnak informálnia kell az ágenst, hogy a borravaló elvesztése egy negatív hozzájárulás az általános teljesítményéhez – ezek után az ágens megtanulhatja, hogy a hirtelen manőverek nem járulnak hozzá a hasznosságához. Bizonyos értelemben a teljesítményszabvány a beérkező észlelés egy részét elkülöníti **jutalom**ként (**reward**) (vagy **büntetés**ként – **penalty**), amely az ágens viselkedési minőségének direkt visszacsatolása. A rögzített teljesítményszabványok, mint például állatokban a fájdalom vagy éhség, ily módon értelmezhetők. Ezt a kérdést a 21. fejezetben tárgyaljuk.

Összegezve, az ágenseknek sokféle komponensük van, és ezen komponensek sokféleképpen reprezentálhatók az ágensprogramban, így úgy tűnik, sokféle tanulási módszer létezik. Van ugyanakkor egy egyesítő motívum. A tanulás az intelligens ágensekben összefoglalható úgy, mint egy folyamat, amely az ágens minden komponensét úgy módosítja, hogy az összhangba kerüljön a rendelkezésre álló visszacsatolt információval, ily módon növelve az ágens általános teljesítményét.

5. Összefoglalás

Ebben a fejezetben szélsebesen végigszaladtunk a mesterséges intelligencián, amelyet úgy tekintettünk, mint az ágenstervezés tudományát. A következő fontos pontokra kell emlékeznünk:

- Egy **ágens** (**agent**) olyan valami, amely észlel és cselekszik egy környezetben. Az **ágensfüggvény** (**agent function**) meghatározza, hogy egy tetszőleges észlelési sorozathoz milyen cselekvést hajtson végre az ágens.
- A **teljesítménymérték** (**performance measure**) értékeli az ágens viselkedését egy környezetben. Egy **racionális ágens** (**rational agent**) úgy cselekszik, hogy maximalizálja a teljesítménymérték várható értékét az addigi észlelési sorozat alapján.
- A **feladatkörnyezet** (**task environment**) leírása tartalmazza a teljesítménymérték, a külső környezet, a beavatkozók és a szenzorok meghatározását. Egy ágens tervezése során az első lépés mindig a feladatkörnyezet minél teljesebb leírása.
- A feladatkörnyezetek számos lényeges dimenzió mentén különbözhetnek. Lehetnek teljesen vagy részben megfigyelhetők, determinisztikusak vagy sztochasztikusak, epizódszerűek vagy sorozatszerűek, statikusak vagy dinamikusak, diszkrétek vagy folytonosak, valamint egyágensesek vagy többágensesek.
- Az **ágensprogram** (**agent program**) implementálja az ágensfüggvényt. Többféle alapvető ágensfelépítés létezik attól függően, hogy milyen információ jelenik meg és használódik fel a döntési folyamatban. A konstrukció különbözhet hatékonyságában, kompaktságában és rugalmasságában. Az ágensprogram megfelelő konstrukciója a környezet természetétől függ.
- Az **egyszerű reflexszerű ágens**ek (**simple reflex agent**s) azonnal válaszolnak az észlelésekre, míg a **modellalapú reflexszerű ágens**ek (**model-based reflex agent**s) a világ aktuális észlelésekből nem nyilvánvaló aspektusait belső állapotukban tartják nyilván. A **célorientált ágens**ek (**goal-based agent**s) céljaik elérése érdekében cselekszenek, a **hasznosságorientált ágens**ek (**utility-based agent**s) pedig saját „boldogságukat” próbálják meg maximalizálni.
- Minden ágens javíthatja a teljesítményét **tanulás** (**learning**) segítségével.

5.1. Irodalmi és történeti megjegyzések

A cselekvés központi szerepe az intelligenciában – a praktikus következtetés fogalma – egészen Arisztotelész *Nikomakhoszi Etiká*jáig vezethető vissza. A praktikus következtetés McCarthy (McCarthy, 1958) nagy hatású cikkének is tárgya volt: *Programs with Common Sense* (Józan ésszel bíró programok). A robotika és szabályozáselmélet területei természetüknél fogva alapvetően fizikai ágensek tervezésével foglalkoznak. A **vezérlő** (**controller**) koncepciója a szabályozáselméletben megegyezik az MI ágens fogalmával. Talán meglepő, hogy az MI történelmének nagy részében az ágensek elkülönített komponenseire – kérdés-felelet rendszerek, tételbizonyítók, látórendszerek és így tovább – koncentrált inkább, mint a teljes ágensekre. Az ágensek leírása Genesereth és Nilsson (Genesereth és Nilsson, 1987) cikkében egy nagy hatású kivétel volt. A teljes ágensk

Created by XMLmind XSL-FO Converter.

ma már széles körben elfogadott a szakmában, és ez lett az újabb írások központi témája (Poole és társai 1998; Nilsson 1998).

Az 1. fejezet a racionalitás koncepciójának gyökereit a filozófiára és közgazdaságtanra vezette vissza. Az MI-ben a nyolcvanas évek közepéig ez a koncepció csak periferiális érdeklődést kapott, amikor elkezdte elborítani a terület megfelelő technikai megalapozásának tárgyalásait. Jon Doyle cikke azt jósolta, hogy a racionális ágensek tervezése képezi majd az MI fő küldetését, míg más népszerű területek leválnak új tudományágakat formálva (Doyle, 1983).

A környezetek tulajdonságainak gondos figyelembevétele és hatásuk a racionális ágensek tervezésére leginkább a szabályozáselméleti tradícióban látható – például a klasszikus szabályozó rendszerek (Dorf és Bishop, 1999) teljesen megfigyelhető, determinisztikus környezeteket kezelnek; a sztochasztikus optimális vezérlés (Kumar és Varaiya, 1986) részlegesen megfigyelhető, sztochasztikus környezeteket kezel; és a hibrid szabályozó (Henzinger és Sastry, 1998) olyan környezetekkel foglalkozik, amelyek mind diszkrét, mind folytonos elemeket tartalmaznak. A teljesen és a részlegesen megfigyelhető környezetek közötti különbségtétel az operációkutatás területén kifejlődött **dinamikus programozás** (**dynamic programming**) irodalmában szintén központi kérdés (Puterman, 1994), amit a 17. fejezetben fogunk tárgyalni.

A reflexszerű ágensek voltak az elsődleges modellek a pszichológiai behavioristák, mint például Skinner (Skinner, 1953) számára, aki megpróbálta az organizmusok pszichológiáját kizárólag bemenet/kimenet, avagy inger/válasz leképezésekre visszavezetni. A pszichológiában a behaviorizmustól a funkcionalizmusig jutó fejlődés, amelyet legalább részben a számítógép-metafora ágensekre történő alkalmazása idézett elő (Putnam, 1960; Lewis, 1966), az ágens belső állapotának a megjelenését eredményezte. Az MI munkáinak nagy része az állapottal rendelkező tiszta reflexszerű ágensek ötletét túl egyszerűnek tartja ahhoz, hogy előrelépést jelentsen, de Rosenschein (Rosenschein, 1985) és Brooks (Brooks, 1986) munkája megkérdőjelezte ezt a feltételezést (25. fejezet). Az elmúlt években jelentős munka zajlott annak érdekében, hogy hatékony algoritmusokat találjanak komplex környezetek nyomon követésére (Hamscher és társai, 1992). A Remote Agent program, amely a Deep Space One űrhajót vezérelte (leírása a 4. szakasz - A mesterséges intelligencia jelenlegi helyzete található) egy különösen meggyőző példa (Muscettola és társai, 1998; Jonsson és társai, 2000).

A célorientált ágenseket mindenhol feltételezték, kezdve Arisztotelész gyakorlati gondolkodásáról alkotott nézetétől McCarthy logikai MI-ről írt korai cikkeiig. A Shakey the Robot (Fikes és Nilsson, 1971; Nilsson, 1984) volt egy logikai, célorientált ágens első robot formájú megtestesülése. A célorientált ágensek teljes logikai elemzése Genesereth és Nilsson (Genesereth és Nilsson, 1987) cikkében jelent meg, és egy célorientált programozási módszertant, az úgynevezett ágensorientált programozást Shoham (Shoham, 1993) dolgozott ki.

A célorientált nézőpont a kognitív pszichológia hagyományában is domináns a problémamegoldás területén, kezdve a jelentős mértékben meghatározó *Human Problem Solving* (Emberi problémamegoldás) (Newell és Simon, 1972) művel, és folytatódva Newell későbbi munkájában (Newell, 1990). A célok, tovább bontva *vágyakra* (általános) és *szándékokra* (jelenleg követett), központi témái a Bratman (Bratman, 1987) által kifejlesztett ágenselméletnek is. Ez az elmélet hatással volt mind a természetes nyelv megértésére, mind a többágenses rendszerekre.

Horvitz kifejezetten azt javasolja, hogy az MI alapja a racionalitás mint az elvárt hasznosság maximalizálása legyen (Horvitz és társai, 1988). Pearl írása (Pearl, 1988) volt az első az MI-ben, amely részletesen tárgyalta a valószínűséget és a hasznosságelméletet; az írás a bizonytalanság melletti következtetés és döntéshozatal praktikus módszereinek kifejtésével valószínűleg a legnagyobb önálló tényező volt a hasznosságorientált ágensek felé történő elmozdulásban a kilencvenes években (V. rész).

A 2.15. ábrán bemutatott általános tanuló ágens tervezés klasszikus a gépi tanulás irodalmában (Buchanan és társai, 1978; Mitchell, 1997). Ezen felépítés programokban megtestesülő példái legalább olyan messze nyúlnak vissza, mint Arthur Samuel dámajátékot játszó tanuló programja (Samuel, 1959, 1967). A tanuló ágenseket a VI. rész tárgyalja részletesen.

Az ágensek és ágenstervezés iránti érdeklődés gyorsan növekedett az elmúlt években, részben az internet, valamint az automatizált és mobil **szoftbot**ok iránti igény növekedésével (Etzioni és Weld, 1994). A releváns cikkeket a *Readings in Agents*ben (Huhns és Singh, 1998), valamint a *Foundations of Rational Agency*ben (Wooldridge és Rao, 1999) gyűjtötték össze. A *Multiagent Systems* (Weiss, 1999) nyújtja az ágenstervezés sok aspektusának egységes alapját. Az ágenseknek szentelt konferenciák között van az International Conference on Autonomous Agents, az International Workshop on Agent Theories, Architectures, and Languages, és az International Conference on Multiagent Systems. Végezetül, a *Dung Beetle Ecology* (Ganajtúró bogár ökológia)

Created by XMLmind XSL-FO Converter.

(Hanski és Cambefort, 1991) érdekes információk gazdag választékát kínálja a ganajtúró bogarak viselkedéséről.

5.2. Feladatok

2.1.

Definiálja saját világait a következő fogalmakkal: ágens, ágensfüggvény, ágensprogram, racionalitás, autonómia, reflexszerű ágens, modellalapú ágens, célorientált ágens, hasznosságorientált ágens, tanuló ágens!

2.2.

Mind a teljesítménymérték, mind a hasznosságfüggvény azt méri, hogy mennyire dolgozik jól egy ágens. Magyarázza el a köztük levő különbséget!

2.3.

Ez a gyakorlat felfedi az ágensfüggvények és a programok közötti különbségeket.

a. Lehetséges több mint egy ágensprogram, amely egy adott ágensfüggvényt valósít meg? Adjon meg egy példát vagy mutassa meg, miért nem lehetséges!

b. Vannak olyan ágensfüggvények, amelyeket nem lehet programmal megvalósítani?

c. Rögzítve egy géparchitektúrát, igaz-e, hogy minden egyes ágensprogram pontosan egy ágensfüggvényt valósít meg?

d. Lehetséges több mint egy ágensprogram, amely egy adott ágensfüggvényt valósít meg?

2.4.

Vizsgáljuk meg különböző porszívóvilág-beli ágensfüggvények racionalitását!

a. Mutassa meg, hogy a 2.3. ábrán látható egyszerű porszívó ágensfüggvény igenis racionális a 2.2. szakasz - Racionalitás felsorolt feltételezésekkel!

b. Írjon le egy racionális ágensfüggvényt arra a módosított teljesítménymértékre, amely levon egy pontot minden elmozdulásért! Szüksége van-e a függvénynek megfelelő ágensprogramnak belső állapotra?

c. Mutasson lehetséges ágensterveket olyan esetekre, amikor tiszta négyzetek piszkossá válhatnak, és a környezet geográfiája ismeretlen! Értelmes dolog-e, hogy ezekben az esetekben az ágens saját tapasztalataiból tanuljon? Ha igen, akkor mit kellene megtanulnia?

2.5.

Az alábbi ágensek mindegyikére dolgozza ki a feladatkörnyezetek TKBÉ-leírását!

a. Robot focijátékos;

b. Internetes könyvvásárló ágens;

c. Autonóm Mars terepjáró;

d. Matematikusok tételbizonyító segítője.

2.6.

A 2.5. példa minden egyes ágenstípusára jellemezze a környezeteket a 2.3. fejezetben megadott tulajdonságok alapján, majd válasszon ki megfelelő ágensterveket!

A következő feladatok mindegyike egy környezetnek és egy sor ágensnek a megvalósítására vonatkozik a porszívóvilágban.

Created by XMLmind XSL-FO Converter.

2.7.

Valósítson meg egy teljesítménymérő környezetszimulátort a 2.2. ábrán mutatott és 2.2. szakasz - Racionalitás specifikált porszívóvilágra! Implementációja legyen moduláris, hogy könnyen megváltoztathassa az érzékelőket, a beavatkozókat és a környezeti jellemzőket (méret, alak, piszok elhelyezkedése stb.). (*Megjegyzés:* Bizonyos operációs rendszerekre és programozási nyelvekre vannak kész implementációk az internetes kódtárban.)

2.8.

Tervezzen és valósítson meg egy tiszta reflexszerű ágenst a 2.7. feladat környezetére! Minden lehetséges kezdeti koszeloszlásra és ágenselhelyezkedésre futtassa le a környezetszimulátort ezzel az ágenssel! Jegyezze fel az ágens teljesítménymértékét minden egyes konfigurációra és az átlagos pontszámát is!

2.9.

Vizsgálja meg a 2.7. gyakorlat egy módosított környezetét, amelyben az ágenst egy ponttal büntetik minden egyes elmozdulásért.

a. Lehet-e tökéletesen racionális egy egyszerű reflexszerű ágens ebben a környezetben? Magyarázza meg, miért!

b. Mi a helyzet az állapottal rendelkező reflexszerű ágensekkel? Tervezzen egy ilyen ágenst!

c. Hogyan változna meg az (a) és (b) pontokra adott válasza, ha az ágens észlelései a környezet összes négyzetére megmondják neki, hogy azok tiszták vagy piszkosak?

2.10.

Vizsgálja meg a 2.7. feladat porszívókörnyezetének egy módosított változatát, amelyben a környezet alakja – mérete, határai és akadályai – ismeretlen, mint ahogy a kosz kezdeti elhelyezése is az! (Az ágens mehet *Fel* és *Le*, valamint *Balra* és *Jobbra* is.)

a. Lehet-e tökéletesen racionális egy egyszerű reflexszerű ágens ebben a környezetben? Magyarázza meg, miért!

b. Egy *randomizált* ágensfüggvénnyel rendelkező reflexszerű ágens jobban teljesíthet egy egyszerűnél? Tervezzen egy ilyen ágenst, és mérje meg a teljesítményét különböző környezetekben!

c. Tud olyan környezetet tervezni, ahol az Ön randomizált ágense igen rosszul fog teljesíteni? Mutassa meg az eredményeit!

d. Egy állapottal rendelkező reflexszerű ágens jobban teljesíthet egy egyszerűnél? Tervezzen egy ilyen ágenst, és mérje meg a teljesítményét különböző környezetekben! Tud ilyen típusú racionális ágenst tervezni?

2.11.

Ismételje meg a 2.10. feladatot egy olyan esetre, amikor az elhelyezkedés érzékelőt egy „ütközés” érzékelő váltja fel, ami érzékeli az ágens olyan elmozdulási kísérleteit, amikor nekiütközik egy akadálynak vagy átlépi a környezet határát! Tegye fel, hogy az ütközésérzékelő működése leáll! Hogyan kellene az ágensnek viselkednie?

2.12.

Az eddigi feladatok porszívókörnyezetei determinisztikusak voltak. Vizsgáljon meg lehetséges ágensprogramokat a következő sztochasztikus variációkra:

a. Murphy törvénye: az idő 25%-ában a *Felszívás* művelet nem tudja feltakarítani a padlót, ha az piszkos, illetve kiönti rá a koszt, ha az tiszta. Hogyan érinti az ágens programját az, ha a piszokérzékelő az idő 10%-ában rossz választ ad?

b. Kisgyerekek: minden időlépésben minden négyzetnek 10% esélye van bekoszolódni. Tud javasolni egy racionális ágenstervet erre az esetre?

Created by XMLmind XSL-FO Converter.

3. fejezet - Problémamegoldás kereséssel

Ebben a fejezetben megnézzük, hogy egy ágens hogyan tudja megtalálni azt a cselekvéssorozatot, amely elvezet a célhoz, ha erre egyetlen egyedi cselekvés sem elegendő.

A 2. fejezetben említett legegyszerűbb ágensek a reflexszerű ágensek voltak, amelyek cselekvései az állapotok és a cselekvések közötti közvetlen leképzésen alapulnak. Az ilyen ágensek nem képesek olyan környezetben tevékenykedni, ahol e leképzés tárolási igénye túl nagy lenne, vagy ahol a leképezés megtanulása túl sok időt venne igénybe. A célorientált ágensek azonban sikerrel járhatnak a jövőbeli cselekvéseknek és annak a számbavételével, hogy ezen cselekvések kimenetelei mennyire kívánatosak.

Ebben a fejezetben a célorientált ágensek egyik típusát, a **problémamegoldó ágenst (problem-solving agent)** ismertetjük. A problémamegoldó ágensek úgy határozzák meg, mit is kell tenniük, hogy olyan cselekvéssorozatokat keresnek, amelyek a kívánt állapotokba vezetnek. Azzal kezdjük, hogy pontosan megfogalmazzuk a „problémát” és a „megoldását” felépítő alkotóelemeket, és számos példával illusztráljuk ezen definíciókat. Ezek után néhány általános rendeltetésű keresési algoritmust mutatunk be, amelyek e problémák megoldására alkalmasak, majd az egyes algoritmusok előnyeit összehasonlítjuk. Az algoritmusok **nem informált**ak (**uninformed**) abban az értelemben, hogy a probléma definícióján túlmenően más információval a problémáról nem rendelkeznek. A 4. fejezet **informált** (**informed**) keresési stratégiákat tárgyal, amelyek rendelkeznek valamilyen elképzeléssel arról, hogy a megoldást merrefelé is kell keresni.

Ebben a fejezetben az algoritmuselmélet területén használt fogalmakat alkalmazzuk. Az aszimptotikus komplexitás (azaz az *O*() jelölésmód) és az NP-teljesség fogalomkörében nem járatos olvasó számára az A) függelék nyújt bővebb információkat.

1. Problémamegoldó ágensek

Az intelligens ágensekről feltesszük, hogy a teljesítménymértéküket maximalizálják. Mint azt a 2. fejezetben láttuk, a feladat valamelyest leegyszerűsödik, ha az ágens egy **célt** (**goal**) tud maga elé tűzni, és megpróbálja azt elérni. Először vizsgáljuk meg, hogy az ágens hogyan és miért teheti ezt meg.

Tegyük fel, hogy az ágensünk a romániai Arad városában tartózkodik, és országjáró túrája vége felé közeledik. Az ágens teljesítménymértéke számos tényezőt tartalmaz: például szépen le szeretne barnulni, tökéletesíteni szeretné a román nyelvtudását, látni szeretné az összes nevezetességet, élvezni az éjszakai életet (amilyen van), kerülni a másnaposságot stb. A döntési probléma igen bonyolult, sok kompromisszummal és útikönyvek gondos tanulmányozásával jár. Tételezzük most fel, hogy ágensünk egy Bukarestből érvényes, vissza nem váltható repülőjeggyel rendelkezik másnapra. Ebben a helyzetben értelmes **cél**nak (**goal**) tűnik, hogy eljusson Bukarestbe. Mindazon cselekvéseket további mérlegelés nélkül el lehet dobni, amelyekkel nem lehet időben Bukarestbe érni, amivel az ágens döntési problémája nagymértékben egyszerűsödik. A célok segítik megszervezni a viselkedést, mivel korlátozzák az ágens által elérendő dolgok számát. A pillanatnyi helyzeten és az ágens hasznosságmértékén alapuló **célmegfogalmazás** (**goal formulation**) a problémamegoldás első lépése.

Célnak a világ állapotainak egy halmazát tekintjük, pontosabban azon állapotok halmazát, amelyekben a cél teljesül. Az ágens feladata megkeresni, hogy amely cselekvések sorozata juttatja el őt egy célállapotba. Mielőtt azonban ezt megtehetné, el kell döntenie, hogy milyen cselekvéseket és állapotokat vizsgáljon meg. Amennyiben olyan szintű cselekvésekkel kellene dolgoznia, mint például „a bal lábadat tedd előrébb fél méterrel” vagy „a kormánykereket hat fokkal tekerd balra”, az ágens még a parkolóból sem lenne képes kihajtani, nemhogy időben eljutni Bukarestbe, mivel az ilyen részletek szintjén a világban túl sok a bizonytalanság, és a megoldás túl sok lépésből állna. A **problémamegfogalmazás** (**problem formulation**) az a folyamat, amely során eldöntjük, hogy amely cselekvéseket és állapotokat vegyünk figyelembe, ha egy cél adott. Ezt a folyamatot részletesebben is tárgyalni fogjuk. Egyelőre tételezzük fel, hogy az ágens az egyik nagyobb városból egy másik nagyobb városba való autózás szintjén szemléli a cselekvéseket. Ebből adódóan a figyelembe vett állapotok annak felelnek meg, hogy az ágens éppen egy adott városban van.[1]

[1] Vegyük észre, hogy ezek az állapotok valójában a világ állapotaiból képezett nagy halmazoknak felelnek meg, mivel a világ állapotai a valóság minden aspektusát meghatározzák. Fontos észben tartani a problémamegoldás állapotai és a világ állapotai közötti különbséget.

Created by XMLmind XSL-FO Converter.

Az ágensünk azt tűzte ki célul, hogy Bukarestbe megy, és azt fontolgatja, hogy Aradról hová menjen. Három út vezet ki Aradról: egy Nagyszeben felé, egy Temesvár felé és egy Nagyzerénd felé. Ezek egyike sem éri el közvetlenül a célt, így amennyiben ágensünk nem ismeri ki magát tökéletesen Romániában, nem fogja tudni, hogy melyik úton induljon el.[2] Más szóval az ágens nem fogja tudni eldönteni, hogy a lehetséges cselekvések közül melyik a legjobb, mert nincs elegendő információja a cselekvések hatására előálló állapotokról. Amennyiben az ágensnek nem áll rendelkezésére további tudás, akkor itt megakad. A legjobb amit tehet, hogy a cselekvésekből véletlenszerűen kiválaszt egyet.

Fontos

Tegyük fel azonban, hogy ágensünknek akár papíron, akár a fejében van egy Románia-térképe. A térkép jelentősége, hogy információt nyújthat az ágensnek azokról az állapotokról, ahova eljuthat, illetve a végrehajtható cselekvésekről. Az ágens ezen információt felhasználhatja egy hipotetikus utazás megtervezésére ezeken a városokon keresztül, hogy megtalálja a legjobb utat, amely végül elvezet Bukarestbe. Miután a térképen megtalált egy utat Aradról Bukarestbe, a célját el tudja érni, ha az utazás egyes állomásaihoz tartozó vezetési szakaszokat végrehajtja. Általánosságban: *amennyiben egy ágens előtt közvetlenül több ismeretlen értékű lehetőség áll, ki tudja választani, hogy mit tegyen, ha először megvizsgálja a különböző lehetséges cselekvéssorozatokat, amelyek ismert értékű állapotokba vezetnek, majd ezután kiválasztja a legjobbat*.

Az ilyen sorozat előállítási folyamatát **keresés**nek (**search**) nevezzük. A keresési algoritmus bemenete egy probléma, kimenete pedig egy cselekvéssorozat formájában előálló **megoldás** (**solution**). Miután előállt egy megoldás, az abban szereplő cselekvéseket végre lehet hajtani. Ezt **végrehajtás**i (**execution**) fázisnak szokás nevezni. Így előállt egy egyszerű „fogalmazd meg, keresd meg, hajtsd végre” ágenstervezési séma. Az előállt sémát a 3.1. ábra mutatja. A cél és a megoldandó probléma megfogalmazása után az ágens egy keresési eljárást hív meg. Ezek után a megoldást használja cselekvései vezérlésére, azt teszi, amit a megoldás a következő lépésnek javasol – ez tipikusan a cselekvéssorozat első lépése –, és a végrehajtott lépést törli a cselekvéssorozatból. Miután a megoldást végrehajtotta, az ágens új célt keres.

3.1. ábra - Egy egyszerű problémamegoldó ágens. Először a célt és a problémát fogalmazza meg, majd a problémát megoldó cselekvéssorozatot keres, végül a cselekvéseket egyenként végrehajtja. Amikor kész vele, egy másik célt fogalmaz meg, és az egészet újrakezdi. Jegyezzük meg, hogy a cselekvéssorozata végrehajtása alatt az ágens az érzékeléseivel nem foglalkozik. Feltételezi, hogy az általa megtalált megoldás mindig működőképes.

```
function EGYSZERŰ-PROBLÉMA-MEGOLDÓ-ÁGENS(érzékelés) returns egy cselekvés
  inputs: érzékelés, egy érzékelés
  static: sorozat, egy cselekvéssorozat, kezdetben üres
          állapot, a világ pillanatnyi állapotának valamilyen leírása
          cél, egy cél, kezdetben üres
          probléma, egy probléma megfogalmazása

  állapot ← ÁLLAPOT-FRISSÍTÉS(állapot, érzékelés)
  if sorozat üres then
    cél ← CÉL-MEGFOGALMAZÁS(állapot)
    probléma ← PROBLÉMA-MEGFOGALMAZÁS(állapot, érzékelés)
    sorozat ← KERESÉS(probléma)
  cselekvés ← AJÁNLÁS(sorozat, állapot)
  sorozat ← MARADÉK(sorozat, állapot)
  return cselekvés
```

[2] Feltételezzük, hogy legtöbb olvasónk ugyanebben a cipőben jár, és könnyen át tudja érezni ágensünk tanácstalanságát. Ezúton elnézést kérünk a romániai olvasóktól, akik számára ezen pedagógiai eszköz nem jelent segítséget.

Created by XMLmind XSL-FO Converter.

Először a problémamegfogalmazás folyamatát írjuk le, majd a fejezet további részeit a KERESÉS függvény különböző változatainak ismertetésére fordítjuk. Ebben a fejezetben nem tárgyaljuk részletesen az ÁLLAPOT-FRISSÍTÉS és a CÉL-MEGFOGALMAZÁS függvényeket.

Mielőtt a részletekbe belemerülnénk, egy pillanatra álljunk meg és nézzük meg, hogy a problémamegoldó ágens hogyan illik bele a 2. fejezetben tárgyalt ágensek és környezetek sokaságába. A 3.1. ábrán látható ágenstervezet feltételezi, hogy a környezet **statikus** (**static**), mivel a probléma megfogalmazása és megoldása semmilyen változásról nem vesz tudomást, amely esetleg a környezetben beáll. Az ágenstervezet azt is feltételezi, hogy a kezdeti állapot ismert. Ennek ismerete akkor a legkönnyebb, amikor a környezet **megfigyelhető** (**observable**). Az „alternatív cselekvések számontartása” tulajdonképpen azt tételezi fel, hogy a környezet **diszkrét** (**discrete**). Végül a legfontosabb: az ágenstervezet azt is feltételezi, hogy a környezet **determinisztikus** (**deterministic**). A problémamegoldások egyedi cselekvéssorozatok, így a váratlan eseményeket nem is tudják figyelembe venni. A megoldások végrehajtása ráadásul az érzékelésről nem is vesz tudomást! Annak az ágensnek, amely mondhatni csukott szemmel hajtja végre a terveit, eléggé biztosnak kell lennie a dolgában (szabályozáselméletben az ilyen rendszereket **nyílt hurkú**nak – **open-loop** – hívják, mert az érzékelések figyelmen kívül hagyása az ágens és a környezete közötti hurkot felbontja). E feltételezések azt jelentik, hogy a környezetek lehető legegyszerűbbikével foglalkozunk, és ez egyben indokolja azt is, hogy e fejezet miért a könyv elején található. A 3.6. alfejezet rövid bepillantást ad abba, hogy mi történik, amikor a megfigyelhetőség és a determinizmus feltételezéséről lemondunk. Ezt a témát sokkal mélyebben a 12. és a 17. fejezet tárgyalja.

1.1. Jól definiált problémák és megoldások

Egy **probléma** (**problem**) formális megragadásához az alábbi négy komponensre van szükség:

- A **kiinduló állapot** (**initial state**), amiből az ágens kezdi a cselekvéseit. A romániai ágensünk kezdeti állapotát például *Benn*(*Arad*)-ként lehetne leírni.
- Az ágens rendelkezésére álló lehetséges **cselekvés**ek (**action**s) halmaza. A leginkább használatos leírás[3] az **állapotátmenet-függvény**t (**successor function**) alkalmazza. Egy adott *x* állapot esetén az ÁLLAPOTÁTMENET-FV (*x*) visszaadja a rendezett ⟨ *cselekvés, utódállapot*⟩ párok halmazát, ahol minden cselekvés az *x* állapotban legális cselekvések egyike, és minden utódállapotot egy cselekvésnek az *x* állapotra való alkalmazásával nyerünk. A *Benn*(*Arad*) állapotban az állapotátmenet-függvény a romániai problémára a

{⟨ *Menj*(*Nagyszeben*), *Benn*(*Nagyszeben*)⟩, ⟨ *Menj*(*Temesvár*), *Benn*(*Temesvár*)⟩, ⟨ *Menj*(*Nagyzerénd*), *Benn*(*Nagyzerénd*)⟩ }

párokat adná vissza.

A kezdeti állapot és az állapotátmenet-függvény együttesen implicit módon definiálják a probléma **állapotter**ét (**state space**): azon állapotok halmazát, amelyek a kiinduló állapotból elérhetők. Az állapottér egy gráfot alkot, amelynek csomópontjai az állapotok és a csomópontok közötti élek a cselekvések. (Románia 3.2. ábrán látható térképe állapottérként értelmezhető, ha minden útját kétirányú gépkocsi-vezetési cselekvésként fogjuk fel). Az állapottér egy **út**ja (**path**) az állapotok egy sorozata, amely állapotokat a cselekvések egy sorozata köt össze.

- A **célteszt** (**goal test**), amely meghatározza, hogy egy adott állapot célállapot-e. Néha létezik a lehetséges célállapotok egy explicit halmaza, és a teszt egyszerűen megnézi, hogy az ágens elérte-e ezek egyikét. Romániában az ágens célja a {*Benn*(*Bukarest*)} szingleton. Néha a cél valamilyen absztrakt tulajdonsággal van definiálva, nem pedig explicit módon felsorolt állapothalmazzal. A sakkban például az úgynevezett „sakk-matt” állapot elérése a cél, amelyben az ellenfél királya az őt ért támadás elől nem tud elmenekülni.
- Egy **útköltség**- (**path cost**) függvény, amely minden úthoz hozzárendel egy költséget. A problémamegoldó ágens azt a költségfüggvényt fogja választani, amely a saját hatékonysági mértékének felel meg. A Bukarestbe sietős ágens számára az idő a lényeg, így az útköltség lehetne például az út a km-ben kifejezett hossza. Ebben a fejezetben az út költségének az utat alkotó egyes cselekvések költségének összegét fogjuk tekinteni az út mentén. Az *x* állapotból az *y* állapotba vezető *cs* cselekvés **lépésköltség**e (**step cost**) legyen *lk*(*x, cs, y*). Romániában a lépésköltségeket a 3.2. ábra mutatja úttávolságok formájában. Feltételezzük, hogy a lépésköltségek nemnegatívak.[4]

[3] Egy alternatív megfogalmazás az **operátor**ok (**operator**s) egy halmaza, amelyeket egy állapotra alkalmazva lehet az utódállapotokat generálni.

[4] A negatív költségek következményeivel a 3.17. feladatban foglalkozunk.

Created by XMLmind XSL-FO Converter.

Az előbbi elemek definiálják a problémát és egy közös adatstruktúrába foghatók, amit a problémamegoldó algoritmus bemenetének tekintünk. A probléma **megoldás**a (**solution**) nem más, mint a kiinduló állapotból a célállapotba vezető út. A megoldás kvalitását az útköltségfüggvény méri, és egy **optimális megoldás**nak (**optimal solution**) a megoldások közt a legkisebb lesz az útköltsége.

1.2. A problémák megfogalmazása

Az előbbi részben a „hogyan jutunk el Bukarestbe” probléma olyan megfogalmazását javasoltuk, amely a kezdeti állapotból, az állapotátmenet-függvényből, a célállapottesztből és az útköltségből áll. Ez a megfogalmazás értelmesnek tűnik, a valós világ számos aspektusát mégis figyelmen kívül hagyja. Hasonlítsuk csak össze az általunk választott egyszerű állapotleírást, *Benn*(*Arad*), egy tényleges országjáró kirándulással, ahol a világ állapota rengeteg mindent tartalmazhat: kivel utazunk, mit közvetít a rádió, milyen tájat látunk az ablakon át, van-e közelben rendőr, milyen messze van a következő pihenőhely, milyen az út állapota, milyen az időjárás és sok-sok más.

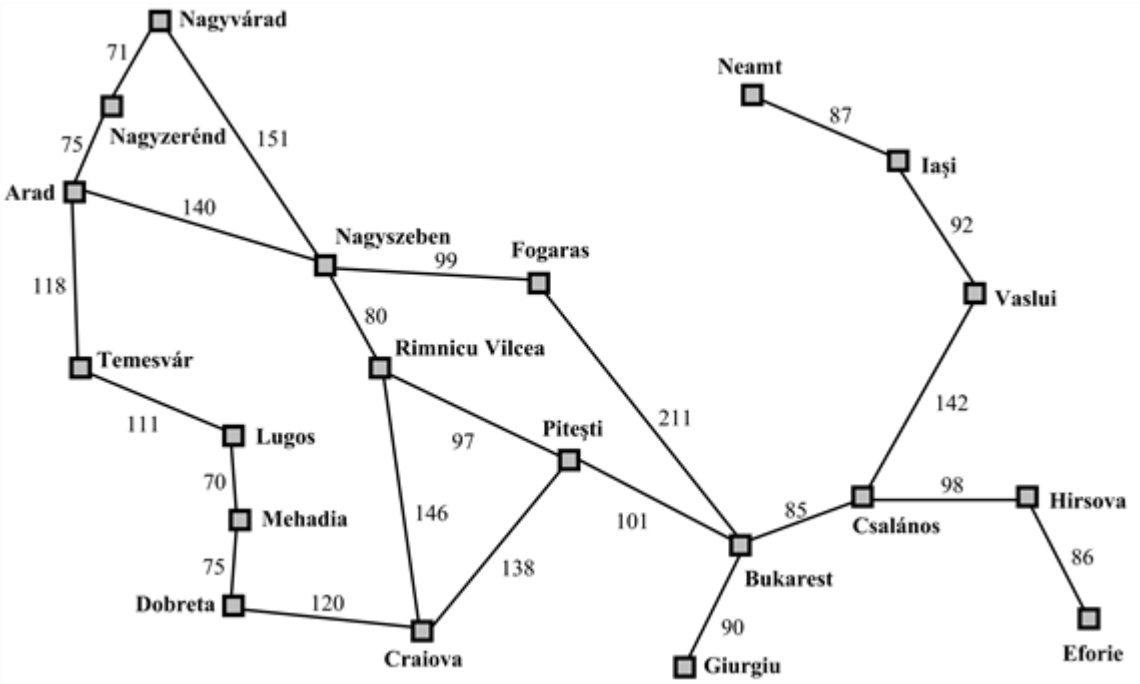
Ezeket a részleteket kihagytuk az állapotleírásokból, mert nem lényegesek a Bukarestbe vezető út megkeresésében. Egy reprezentációból a részletek eltávolítását **absztrakció**nak (**abstraction**) nevezzük.

Az állapotleírás absztrahálása mellett magukat a cselekvéseket is absztrahálni kell. Egy vezetési cselekvésnek számos hatása van. Amellett, hogy megváltoztatja a gépkocsi és utasai helyét, időbe telik, üzemanyagot fogyaszt, szennyezi a levegőt és megváltoztatja az ágenst (azt mondják, hogy az utazás kiszélesíti a látókört). A mi megfogalmazásunkban csak a helyváltoztatást vesszük figyelembe. Van számos olyan cselekvés, amit teljesen elhagyunk: például a rádió bekapcsolását, az ablakon való kinézést, a lassítást, mert rendőr van a közelben stb. És persze a „fordítsd a kormányt balra 3 fokkal” szintű cselekvésekkel sem foglalkozunk.

Lehetünk-e pontosabbak a megfelelő absztrakciós szint meghatározásában? A megválasztott absztrakt állapotokra és cselekvésekre úgy gondolunk, hogy azok a részletes valós állapotok és cselekvések egész halmazaihoz tartoznak. Most tekintsük az absztrakt probléma egy megoldását: például az Aradról Nagyszebenbe, majd Rimnicu Vilceába, Pites¸tibe és Bukarestbe vezető utat. Ez az absztrakt megoldás rengeteg részletesebb útnak felel meg. Például Nagyszeben és Rimnicu Vilcea között vezethettünk bekapcsolt rádióval, majd az utazás további részére kikapcsolhattuk a rádiót. Az absztrakció *érvényes,* ha az absztrakt megoldást megoldássá fejthetjük ki egy részletesebb világban is. Elégséges feltétel az, hogy minden olyan részletes állapothoz, mint az „Aradon van”, létezik egy részletes út valamelyik olyan állapothoz, mint a „Nagyszebenben van” stb. Az absztrakció *hasznos,* ha a megoldásbeli cselekvések végrehajtása az eredeti problémánál egyszerűbb. Ebben az esetben ezek eléggé egyszerűek ahhoz, hogy egy átlagos gépkocsivezető ágens végre tudja azokat hajtani minden további keresés vagy tervkészítés nélkül. Így egy jó absztrakció megválasztása magában foglalja az érvényesség megőrzése mellett a lehető legtöbb részlet törlését, és annak biztosítását, hogy az absztrakt cselekvéseket könnyű legyen véghezvinni. Ha nem lenne meg a hasznos absztrakciók megalkotásának képessége, akkor az intelligens ágensek a valós világban teljesen hasznavehetetlenek lennének.

3.2. ábra - Románia egy részének sematikus országúti térképe

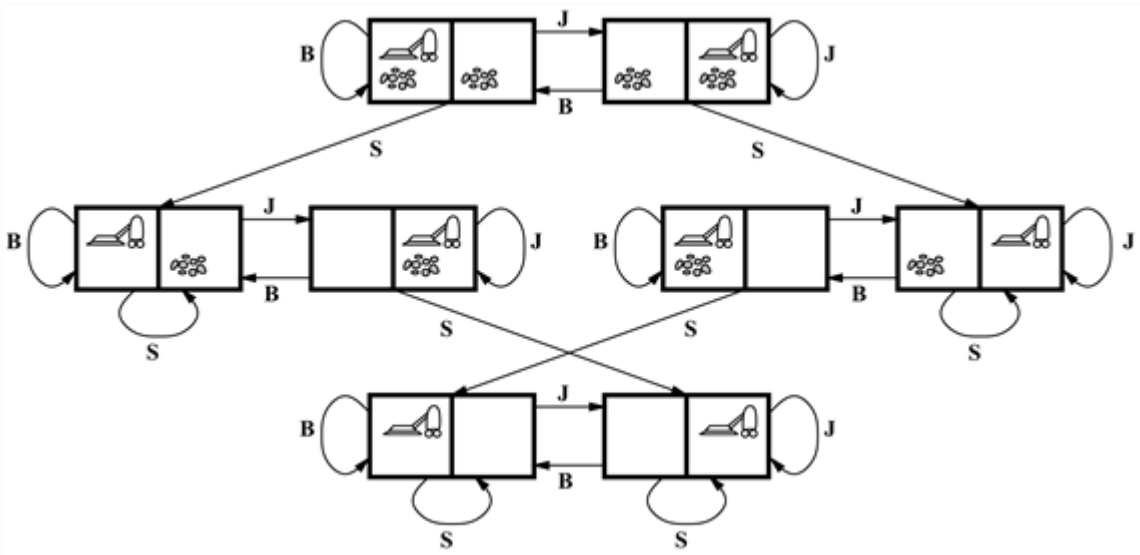
Created by XMLmind XSL-FO Converter.



2. Példaproblémák

A problémamegoldó megközelítést rengeteg feladatkörben alkalmazták. Néhányat az alábbiakban sorolunk fel, megkülönböztetve az úgynevezett játék- és a valósvilág-beli problémákat. A **játékproblémá**k (**toy problem**s) rendeltetése, hogy segítségükkel a különféle problémamegoldó módszereket illusztrálni tudjuk vagy ki tudjuk próbálni. Az ilyen problémák egzakt tömör leírása megadható. Ez azt jelenti, hogy különböző kutatók így könnyűszerrel használhatják az algoritmusok hatékonysági összehasonlításához A **valósvilág-beli problémá**k (**real-world problem**s) azok, amelyek megoldása tényleg érdekes az emberek számára. Ezeknek nincs egyetlen általánosan elfogadott megfogalmazásuk, mi azonban megpróbáltuk érzékeltetni az általános megfogalmazásuk jellegét.

3.3. ábra - A porszívóvilág állapottere. Az élek cselekvéseket jelentenek: B = *Balra,* J = *Jobbra,* S = *Szív.*



2.1. Játékproblémák

Created by XMLmind XSL-FO Converter.

Az első vizsgált példa a 2. fejezetben bevezetett **porszívóvilág** (**vacuum world**) (lásd 2.2. ábra). Ezt problémaként az alábbi módon definiálhatjuk:

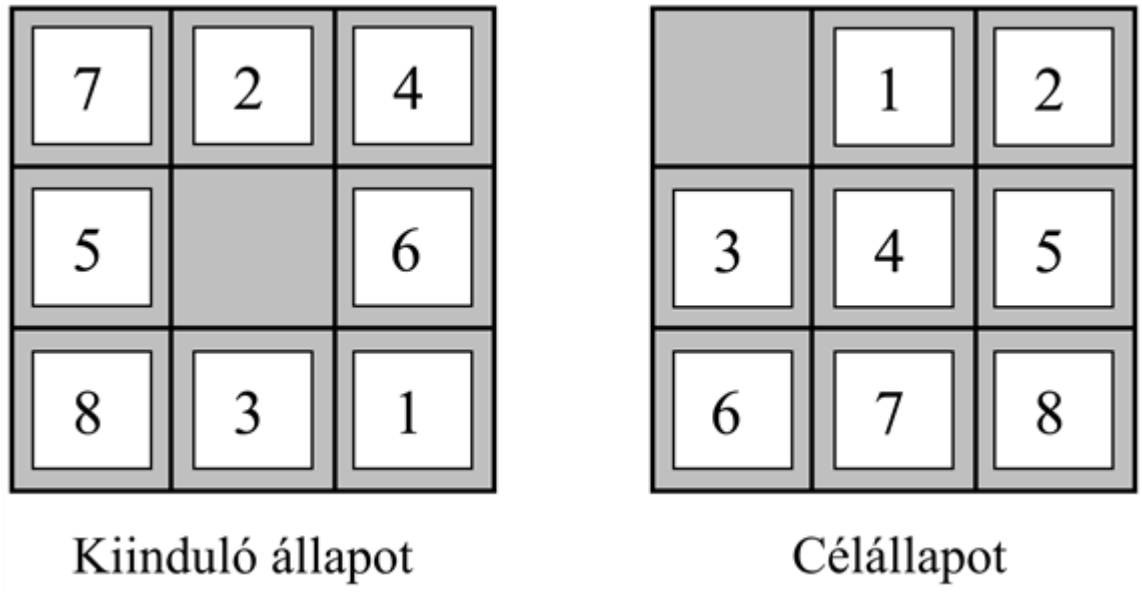
- **Állapot**ok: az ágens két hely egyikében lehet. Mindegyik lehet piszkos, de lehet tiszta is. Így $2 \times 2^2 = 8$ lehetséges állapotról beszélhetünk.
- **Kezdeti állapot**: akármelyik állapot lehet kezdeti állapot.
- **Állapotátmenet-függvény**: a három (*Balra*, *Jobbra*, *Szív*) cselekvés alkalmazásából adódó legális állapotokat generálja. A teljes állapottér a 3.3. ábrán látható.
- **Célteszt**: ellenőrzi, hogy minden négyzet tiszta-e.
- **Útköltség**: minden lépés költsége 1, így az út költsége megegyezik az út lépéseinek a számával.

A valós világgal összevetve e probléma jellemzői a diszkrét lokációk, a diszkrét kosz, a megbízható takarítás, valamint az, hogy takarítás után piszok soha nem keletkezik újra (a 3.6. alfejezetben ezeket a feltételezéseket feladjuk). Fontos megjegyezni, hogy az állapotot az ágens és a kosz helyzete együttesen határozza meg. Egy nagyobb, n helyű környezetben $n\,2^n$ állapot van.

A 3.4. ábrán bemutatott **8-as kirakójáték** (**8-puzzle**) egy 3 × 3-as táblából, 8 számozott kockából és egy üres helyből áll. Az üres hely melletti kockát be lehet csúsztatni az üres helyre. A cél egy meghatározott állás, mint például az ábra jobb oldalán látható állás elérése. A probléma szokásos megfogalmazása a következő:

- **Állapot**ok: az állapotleírás meghatározza mind a nyolc kocka és az üres hely pozícióját a kilenc lehetséges pozíció egyikében.
- **Kezdeti állapot**: akármelyik állás lehet kezdeti állapot. Figyeljük meg, hogy minden egyes célt pontosan az állapotok feléből lehet elérni (3.4. feladat).
- **Állapotátmenet-függvény**: a négy cselekvés (üres hely megy *Balra*, *Jobbra*, *Fel*, *Le*) alkalmazásából adódó legális állapotokat generálja.
- **Célteszt**: ellenőrzi, hogy az állapot megegyezik-e a 3.4. ábrán mutatott célállapottal (más célkonfiguráció is lehetséges).
- **Útköltség**: minden lépés költsége 1, így az út költsége megegyezik az út lépéseinek a számával.

3.4. ábra - A 8-as kirakójáték egy tipikus feladványa



Milyen absztrakciókhoz folyamodtunk itt? A cselekvéseket a kezdeti és végállapotuk erejéig absztraháltuk, elmozdítás közben egy kocka közbülső helyzeteit figyelmen kívül hagytuk. Az olyan cselekvésektől is

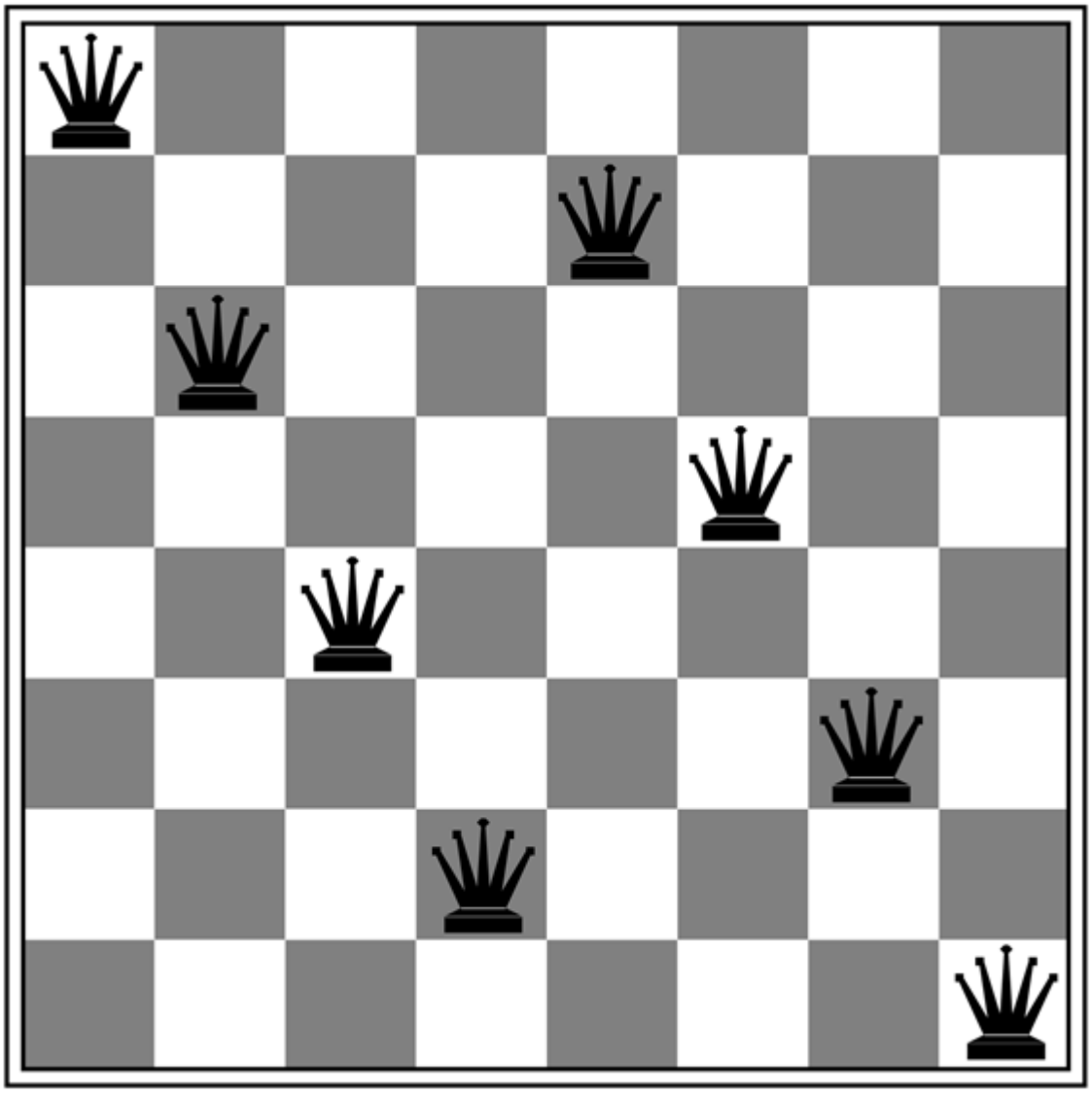
Created by XMLmind XSL-FO Converter.

elvonatkoztattunk, mint a tábla megrázása, ha a kockák megakadtak, vagy a kockák késsel való kiszedése és visszahelyezése. Amit kaptunk, az a játék szabálykészlete, a manipulációk fizikai részleteitől eltekintve.

A kirakójáték a **csúsztatós kirakójáték**ok (**sliding-block puzzle**s) családjába tartozik, amelyeket gyakran új kereső algoritmusok tesztelésére alkalmaznak az MI-ben. Ez az általános osztály NP-teljes, így nem várható, hogy lényegesen jobb módszert lehet találni az itt és a következő bekezdésben leírt keresési algoritmusoknál. A 8-as kirakójátéknak 9!/2 = 181 440 elérhető állapota van, és így e játék könnyen megoldható. A 15-ös kirakójátéknak (a 4 × 4-es táblán) kb. 10 billió állapota van, és a legjobb algoritmusok a véletlenül generált eseteket optimálisan néhány milliszekundum alatt oldják meg. A 24-es játéknak (az 5 × 5-ös táblán) kb. 10^{25} állapota van, és a véletlenszerűen kiválasztott esetek optimális megoldása még mindig kemény dió a jelenlegi gépek és algoritmusok számára.

A **8-királynő probléma** (**8-queens problem**) célja, hogy 8 királynőt úgy helyezzünk el egy sakktáblán, hogy egyáltalán ne támadják egymást. (Egy királynő egy vele azonos sorban, oszlopban vagy átlóban lévő bábut támad.) A 3.5. ábra egy sikertelen megoldási kísérletet mutat: a jobb szélső oszlopban lévő királynőt támadja a bal felső sarokban lévő királynő.

3.5. ábra - Egy majdnem jó megoldás a 8-királynő problémára. (A megoldás megkeresését az olvasóra bízzuk.)



Created by XMLmind XSL-FO Converter.

Bár hatékony célalgoritmusok léteznek ezen probléma és a teljes *n*-királynő probléma megoldására, a királynőfeladat mindamellett érdekes tesztproblémája marad a keresési algoritmusoknak. Két fő megfogalmazása létezik. Az **inkrementális megfogalmazás**ban (**incremental formulation**) az operátorok az állapotleírást *bővítik*, az üres állapottól kezdve. A 8-királynő probléma esetén ez azt jelenti, hogy a királynőket egyenként helyezzük el a sakktáblán. A **teljes állapot leírás**ban (**complete-state formulation**) először felhelyezzük mind a 8 királynőt, majd mozgatjuk őket. Az útköltség mindkét esetben érdektelen számunkra, mert csak a végső állapot számít. A probléma első inkrementális megfogalmazása lehet a következő:

- **Állapot**ok: egy állapot a 0 … 8 királynő, tetszőleges elrendezése a táblán.
- **Kezdeti állapot**: a táblán nincs egy királynő sem.
- **Állapotátmenet-függvény**: helyezz egy új királynőt egy üres mezőre.
- **Célteszt**: 8 királynő a táblán és egyik sincs támadás alatt.

Ebben a megfogalmazásban $64 \times 63 \times \ldots 57 \approx 1{,}8 \times 10^{14}$ lehetséges vizsgálandó sorozatunk van. Sokkal ésszerűbb választás lenne, ha figyelembe vennénk azt a tényt, hogy egy már eleve támadt mezőre nincs értelme letenni egy királynőt:

- **Állapot**ok: n ($0 \leq n \leq 8$) királynő olyan elrendezése a táblán, hogy az n bal oldali oszlopban oszloponként egy található úgy, hogy nem támadják egymást.
- **Állapotátmenet-függvény**: helyezz egy királynőt a bal szélső, még üres oszlopba úgy, hogy azt ne támadja egyetlen királynő sem.

Ez a megfogalmazás a 8-királynő probléma állapotterét $1{,}8 \times 10^{14}$-ről 2057 méretű térré csökkenti, minekutána megoldást könnyű megtalálni. Másrészt 100 királynő a korábbi megfogalmazás 10^{400} állapothoz vezet, míg a javítottnál csak kb. 10^{52} állapotunk lesz (3.5. feladat). Ez óriási redukció, a javított eset azonban még mindig túl nagy az ebben a fejezetben ismertetett algoritmusok számára. A teljes állapot leírásról a 4. fejezetben olvashatunk, az 5. fejezet viszont egy olyan egyszerű algoritmust közöl, amellyel a millió királynőt számláló problémát is könnyen meg lehet oldani.

2.2. Valósvilág-beli problémák

Már korábban láttuk, hogy az **útkeresési problémá**t (**route-finding problem**) hogyan definiálhatjuk a megadott helyek és a közöttük lévő összeköttetések segítségével. Az útkereső algoritmusokat számos területen alkalmazzák. Használják például számítógép-hálózatokban útvonalkeresésre, katonai műveletek tervezésénél és légi útvonaltervező rendszerekben is. Általában nehéz az ilyen problémákat specifikálni. Nézzük meg a légi útvonaltervezési probléma egy egyszerűsített példáját:

- **Állapot**ok: minden állapotot egy hely (például egy repülőtér) és az aktuális időpont azonosít.
- **Kezdeti állapot**: a probléma specifikálja.
- **Állapotátmenet-függvény**: azokat az állapotokat adja vissza, amelyek az aktuális repülőtérről valamely másikra olyan menetrendbeli járatok igénybevételével keletkeznek (esetleg tovább pontosítva, hogy mely osztályra és melyik konkrét ülésre vonatkozik az utazás), amelyek később indulnak, mint az aktuális időpont plusz a repülőtéri tranzit ideje.
- **Célteszt**: elértük-e a célállomást?
- **Útköltség**: függ a pénzügyi költségektől, várakozási időtől, repülési időtől, vámkezelési és útlevél-vizsgálati procedúráktól, az ülés minőségétől, a nap időszakától, a repülőgép típusától, az utaskedvezményektől stb.

A kereskedelmi útvonaljavasló rendszerek a probléma valami hasonló megfogalmazását használják, sok további bonyolítással, hogy a repülőtársaságok bonyolult viteldíjstruktúráit kezelni tudják. Minden tapasztalt utas tudja azonban, hogy nem minden repülőút zajlik a tervek szerint. Egy igazán jó rendszernek eshetőségi tervekkel – például az alternatív járatokra történő tartalék helyfoglalásokkal – is tudnia kell foglalkozni, ameddig persze ezeket a költség és az eredeti terv összeomlásának a valószínűsége igazolja.

Created by XMLmind XSL-FO Converter.

A **körutazási problémá**k (**touring problem**s) az útkeresési problémák rokonai egy fontos különbséggel. Tekintsük például azt a problémát, hogy „A 3.2. ábra minden egyes városába látogass el legalább egyszer úgy, hogy Bukarestből indulj, és az utat ott is fejezd be.” Az útkeresési problémához hasonlóan a cselekvés itt is két szomszédos város közötti utazásnak felel meg. De ebben a problémában az állapottér alapvetően más. Az ágens helye mellett minden egyes állapotban tárolni kell az *ágens által korábban meglátogatott városokat* is. Így a kiinduló állapot ebben az esetben a „Bukarestben vagyok, eddig meglátogattam {Bukarestet}” állapot lenne, míg egy tipikus közbenső állapot a „Vasluiban vagyok, eddig meglátogattam {Bukarestet, Csalánost, Vasluit}” állapot. A célteszt ellenőrizné, hogy az ágens Bukarestben van-e, és végiglátogatta-e mind a 20 várost.

Az **utazó ügynök probléma** (**Travelling Salesperson Problem** – **TSP**) egy körutazási probléma, amelyben minden várost pontosan egyszer kell meglátogatni. A cél a *legrövidebb* út megkeresése. A probléma NP-nehéz, de hatalmas erőfeszítések történtek a TSP-algoritmusok képességeinek javítására. Ezen algoritmusokat az utazó ügynökök útjainak megtervezése mellett nyomtatott áramkörök furatainak készítésénél a fúró automata mozgatásának megtervezésére, továbbá áruházak rakodógépeinek útvonaltervezésére is alkalmazták.

A **VLSI elhelyezés**i (**VLSI layout**) problémában komponensek és összeköttetések millióit egy chipen kell elhelyezni, a terület, az áramköri késleltetés és a szórt kapacitások minimalizálása és a gyártási kihozatal maximalizálása mellett. Az elhelyezési probléma a logikai tervezés fázisát követi, és általában két részből áll: **cellaelrendezés**ből (**cell layout**) és **huzalozás**ból (**channel routing**). A cellaelrendezés során az áramkör primitív komponenseit cellákba rendezik, amelyek mindegyike valamilyen jól definiált funkciót lát el. Minden egyes cellának rögzített nyomtatási képe (méret és alak) van, és adott számú összeköttetést igényel a többi cellához. A cél az, hogy az egyes cellákat úgy helyezzük el a chipen, hogy azok ne lapolódjanak át, és a cellák között legyen megfelelő hely az összekötő vezetékek számára. A huzalozás során minden egyes vezetéknek megfelelő vezetési útvonalat keresünk a cellák között. Ezek a keresési problémák hihetetlenül összetettek, de mindenképpen érdemes a megoldásukat megtalálni. A 4. fejezetben látni fogunk néhány algoritmust, amelyek képesek megoldani ezeket a problémákat.

A **robotnavigáció** (**robot navigation**) a korábban leírt útkeresési probléma általánosítása. Az utak diszkrét halmaza helyett a robot egy folytonos térben mozog (elvben) végtelen cselekvés- és állapothalmazzal. Egy egyszerű, sík felületen mozgó kör alakú robot esetén az állapot lényegében kétdimenziós. Amennyiben a robotnak karja és lába is van, akkor azokat is vezérelni kell, így a keresési tér sokdimenzióssá válik. Már a keresési tér végessé tételéhez is kifinomult technikákra van szükség. A 25. fejezetben megvizsgálunk néhány ilyen módszert. A probléma összetettsége mellett a valós robotoknak foglalkozniuk kell az érzékelőik által szolgáltatott jelek és a motorvezérlők hibáival is.

Összetett objektumok robotok általi **automatikus összeszerelés**ét (**automatic assembly sequencing**) először a `FREDDY` robot mutatta be (Michie, 1972). Azóta lassú, de biztos előrehaladás tapasztalható ezen a területen. Ma már például villanymotorok automatikus összeszerelése gazdaságossági szempontból is kivitelezhető. Az összeszerelési feladatoknál a probléma az alkatrészek összeszerelési sorrendjének megkeresése. Amennyiben rossz szerelési sorrendet választunk, valamikor a későbbiek folyamán egy adott alkatrészt nem tudunk egy másik alkatrész kiszerelése nélkül beszerelni. Annak eldöntése, hogy az összeszerelési sorrend egy lépése megvalósítható-e vagy sem, a robotnavigációhoz hasonló összetett geometriai keresési probléma. Így az összeszerelési sorrend megtervezésében a költséges feladat egy kivitelezhető következő lépés megtervezése. Minden gyakorlati fontosságú algoritmusnak kerülnie kell az állapottér egy kis töredékén túlmenő feltárását. Egy másik fontos összeállítási probléma a **fehérjetervezés** (**protein design**), ahol a célkitűzés olyan aminosav-szekvencia megkeresése, amely valamilyen betegség gyógyítása szempontjából lényeges tulajdonságokkal rendelkezik és háromdimenziós fehérjévé formálható.

Az utóbbi időben egyre nagyobb a kereslet az **interneten kereső** (**Internet searching**) szoftverrobotok iránt, amelyek kérdésekre választ, kapcsolódó információkat, vagy kereskedelmi üzletek lehetőségét kutatják fel. A keresési technikák szempontjából ez igazán jó alkalmazás, mert eléggé nyilvánvaló, hogy az internethez egy gráfszerű kép rendelhető, ahol a csomópontok (weboldalak) linkekkel össze vannak kapcsolva. Az internetkeresés teljes leírását a 10. fejezetre hagyjuk.

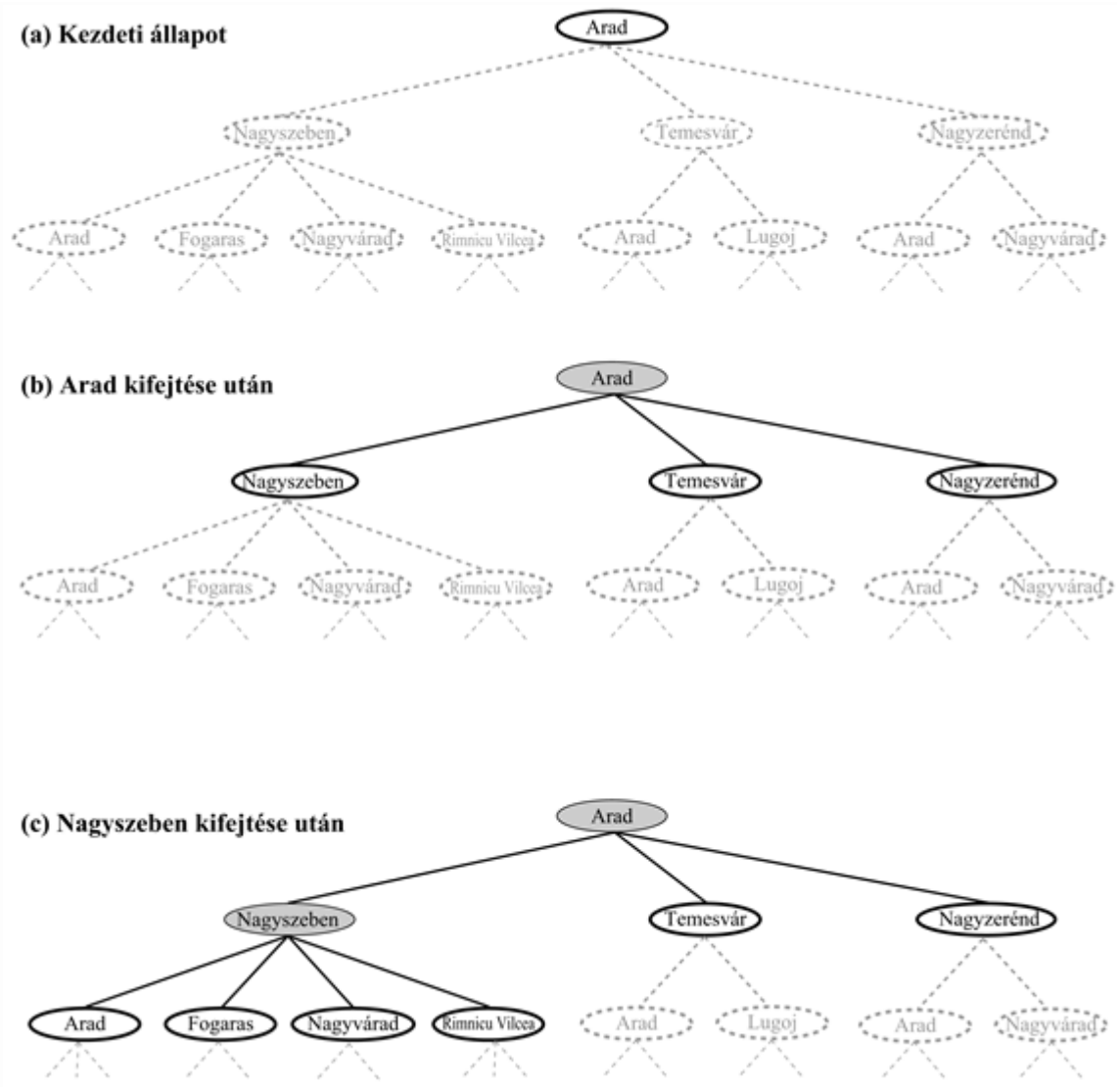
3. Megoldások keresése

Az előzőkben láttuk, hogyan kell egy problémát definiálni. A hátralévő lépés – a megoldás megkeresése – az állapottérben végrehajtott kereséssel történik. Ebben a részben olyan keresési technikákkal foglalkozunk, amelyek egy explicit **keresési fá**t (**search tree**) használnak, amelyet az állapotteret együttesen definiáló kezdeti állapotból és az állapotátmenet-függvényből generálnak. Általánosságban, ha egy állapotot több úton is

Created by XMLmind XSL-FO Converter.

elérhetünk, inkább keresési *gráfról,* mint keresési *fáról* beszélünk. Ezzel az igen fontos bonyolítással később a 3.5. alfejezetben foglalkozunk.

3.6. ábra - Az Arad és Bukarest közötti útkeresési probléma részleges keresési fái. A kifejtett csomópontokat árnyékoltuk, a legenerált, de még ki nem fejtett csomópontok vastagon keretezettek, a még le nem generált csomópontokat pedig halvány szaggatott vonalak jelzik.



A 3.6. ábrán láthatjuk az Arad és Bukarest közötti útkereső probléma keresési fájának egy kezdeti kifejtését. A keresési fa gyökere az a **keresési csomópont** (**search node**), amely a *Benn*(*Arad*) kezdeti állapotnak felel meg. Az első lépés annak ellenőrzése, hogy vajon ez célállapot-e. Nyilvánvalóan nem az, de fontos ellenőrizni, hogy meg tudjuk oldani az olyan beugró problémákat is, mint amilyen például az „Aradról indulva jussunk el Aradra". Mivel ez nem célállapot, egyéb állapotokat is meg kell vizsgálnunk. Ezt az aktuális állapot **kifejtés**ével (**expanding**) tesszük, azaz az állapotátmenet-függvénynek az aktuális állapotra történő alkalmazásával, amivel az állapotok egy új halmazát **generáljuk** (**generating**). Ebben az esetben három új állapotot kapunk: *Benn*(*Nagyszeben*), *Benn*(*Temesvár*) és *Benn*(*Nagyzerénd*). Most el kell döntenünk, hogy a három lehetőség közül melyik utat kövessük.

A keresés lényege a következő: egy lehetőséget kiválasztani, és a többit későbbre halasztani arra az eshetőségre, ha az első választás nem vezetne megoldásra. Tételezzük fel, hogy elsőnek Nagyszebent választjuk. Ellenőrizzük, hogy ez célállapot-e (nem az), majd kifejtjük, aminek hatására a *Benn*(*Arad*), *Benn*(*Fogaras*), *Benn*(*Nagyvárad*) és *Benn*(*RimnicuVilcea*) állapotokat kapjuk. Ezek után e négy közül bármelyiket választhatjuk, vagy akár vissza is mehetünk és választhatjuk Nagyzeréndet vagy Temesvárt. Folytatjuk a

Created by XMLmind XSL-FO Converter.

kiválasztást, a célállapot-ellenőrzést és a kifejtést, míg egy megoldást nem találunk, vagy amíg el nem fogynak a kifejtendő állapotok. A kifejtendő állapot kiválasztását a **keresési stratégia** (**search strategy**) határozza meg. Az általános fakereső algoritmus informális megfogalmazását a 3.7. ábra mutatja.

3.7. ábra - Az általános fakeresési algoritmus informális leírása

```
function FA-KERESÉS(probléma, stratégia) returns egy megoldás vagy kudarc
  a probléma kezdeti állapotából kiindulva inicializáld a keresési fát
  loop do
    if nincs kifejtendő csomópont then return kudarc
    a stratégiának megfelelően válassz ki kifejtésre egy levélcsomópontot
    if a csomópont célállapotot tartalmaz then return a hozzá tartozó megoldás
    else fejtsd ki a csomópontot és az eredményül kapott csomópontokat, és add a keresési fához
  end
```

Fontos megkülönböztetni az állapotteret és a keresési fát. Az útkeresési probléma esetében az állapottér csak 20 állapotból áll, minden egyes városhoz tartozik egy állapot. Ebben az állapottérben azonban végtelen sok út vezet, így a keresési fa végtelen sok csomópontból áll. Például az Arad–Nagyszeben, Arad–Nagyszeben–Arad, Arad– Nagyszeben–Arad–Nagyszeben három út az első három a végtelen számú útszekvenciából. (Egy jó keresési algoritmus nyilvánvalóan elkerüli az ilyen utak követését. A 3.5. alfejezetben megmutatjuk hogyan.)

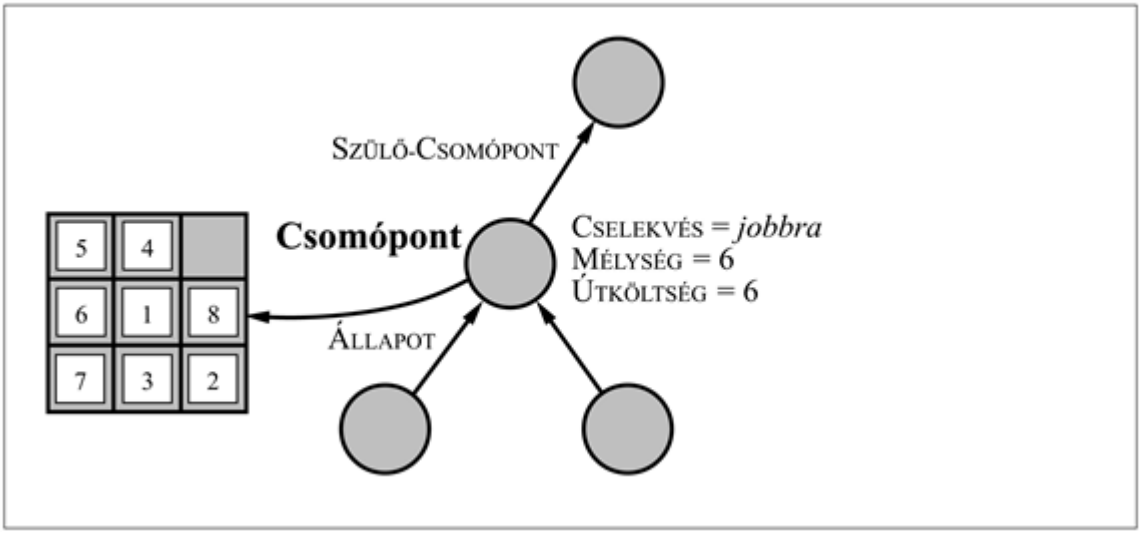
A csomópontokat sokféle módon lehet reprezentálni, de mi feltételezzük, hogy egy csomópont egy öt komponensből álló adatszerkezet:

- ÁLLAPOT: az állapottérnek a csomóponthoz tartozó állapota;
- SZÜLŐ-CSOMÓPONT: a keresési fa azon csomópontja, amely a kérdéses csomópontot generálta;
- CSELEKVÉS: a csomópont szülő-csomópontjára alkalmazott cselekvés;
- ÚT-KÖLTSÉG: a kezdeti állapotból a kérdéses csomópontig vezető út általában $g(n)$-nel jelölt költsége, ahogy ezt a szülőmutatók jelzik;
- MÉLYSÉG: a kezdeti állapotból vezető út lépéseinek a száma.

Fontos visszaidéznünk a csomópontok és az állapotok közti különbséget. A csomópont egy adatnyilvántartásra használt adatszerkezet, amit egy keresési fa leírására használunk. Egy állapot a világ egy konfigurációja. Így a csomópontok a SZÜLŐ-CSOMÓPONT mutatók által definiált meghatározott úton találhatók, míg az állapotok nem. Továbbá könnyen előfordulhat, hogy két különböző csomópont egyazon állapotot tartalmaz, ha ezt az állapotot két különböző cselekvéssorozattal generálták le. A csomópont-adatstruktúrát a 3.8. ábra mutatja.

3.8. ábra - A keresési fa alapvető építő adatstruktúrái a csomópontok. Minden csomópontnak van szülője, állapota és számos adminisztráló adatmezeje. A nyilak a gyerekektől a szülőkig mutatnak.

Created by XMLmind XSL-FO Converter.



Valahogy nyilván kell tartanunk a legenerált, kifejtésre váró csomópontokat is – ezt a gyűjteményt, listát **perem**nek (**fringe**) nevezik. A perem minden eleme egy **levélcsomópont** (**leaf node**), azaz egy olyan csomópont, amelynek a fában nincsenek követői. A 3.6. ábrán a fák pereme a vastagon bekeretezett csomópontokból áll. A perem legegyszerűbb reprezentációja egy csomóponthalmaz lenne. A keresési stratéga ekkor olyan függvény lenne, amely a következő lépésben kifejtendő csomópontot ebből a halmazból választaná ki. Bár elvi szempontból ez nyilvánvaló megoldás, számításigény szempontjából drága lenne, mert a keresési stratégia függvénynek a halmaz minden egyes elemét végig kellene néznie, hogy ki tudja választani a legjobb csomópontot. Ezért a továbbiakban feltesszük, hogy a csomópontgyűjtemény egy **várakozási sor**ként (**queue**) van megvalósítva. A soron végezhető műveletek az alábbiak:

- `SORT-LÉTREHOZ`(*elem, ...*) létrehoz egy az adott elemeket tartalmazó sort.
- `ÜRES?`(*sor*) csak akkor ad vissza igaz értéket, ha a sor üres.
- `ELSŐ-ELEM`(*sor*) visszaadja a sor első elemét.
- `TÁVOLÍTSD-EL-AZ-ELSŐ-ELEMET`(*sor*) visszaadja az `ELSŐ-ELEM`(*sor*)-t és ezt eltávolítja a sorból.
- `BESZÚR`(*elem, sor*) egy elemet szúr be a sorba.
- `BESZÚR-MIND`(*elemek, sor*) egy elemhalmazt beszúr a sorba.

Ezen definíciók felhasználásával az általános fakeresési algoritmus egy formálisabb definícióját adhatjuk meg. Ezt a 3.9. ábra mutatja.

3.9. ábra - Az általános fakeresési algoritmus. (Vegyük észre, hogy a *perem* argumentumnak egy üres sornak kell lennie, és a sor típusa befolyással lesz a keresés sorrendjére.) A `MEGOLDÁS` függvény a szülőmutatók gyökérig való követésével kinyert cselekvéssorozatot adja vissza.

Created by XMLmind XSL-FO Converter.

```
function FA-KERESÉS(probléma, perem) returns egy megoldás vagy kudarc

  perem ← BESZÚR(CSOMÓPONTOT-LÉTREHOZ(KIINDULÓ-ÁLLAPOT[probléma]), perem)
  loop do
    if ÜRES?(perem) then return kudarc
    csomópont ← TÁVOLÍTSD-EL-AZ-ELSŐ-ELEMET(perem)
    if CÉL-TESZT[probléma] alkalmazva az ÁLLAPOT(csomópont) állapotra igazat ad vissza
      then return MEGOLDÁS(csomópont)
    perem ← BESZÚR-MIND(KIFEJT(csomópont, probléma), perem)

function KIFEJT(csomópont, probléma) returns a csomópontok halmazát

  követők ← üres halmaz
  for each ⟨cselekvés, eredmény⟩ in ÁLLAPOTÁTMENET-FV[probléma](ÁLLAPOT[csomópont]) do
    s ← az új CSOMÓPONT
    ÁLLAPOT[s] ← eredmény
    SZÜLŐ-CSOMÓPONT[s] ← csomópont
    CSELEKVÉS[s] ← cselekvés
    ÚT-KÖLTSÉG[s] ← ÚT-KÖLTSÉG[csomópont] + LÉPÉS-KÖLTSÉG(csomópont, cselekvés, s)
    MÉLYSÉG [s] ← MÉLYSÉG[csomópont] + 1
    s hozzáadása a követők-höz
  return követők
```

3.1. A problémamegoldó hatékonyság mérése

A problémamegoldó algoritmus kimenete vagy *kudarc*, vagy egy megoldás (egyes algoritmusok végtelen hurokba kerülhetnek és soha nem térnek vissza válasszal). Mi az algoritmusok hatékonyságát négyféle módon fogjuk értékelni:

- **Teljesség** (**completeness**)**:** az algoritmus garantáltan megtalál egy megoldást, amennyiben létezik megoldás?
- **Optimalitás** (**optimality**): a stratégia megtalálja az optimális megoldást, ahogy azt a 1.1. szakasz - Jól definiált problémák és megoldások részben definiáltuk?
- **Időigény** (**time complexity**): mennyi ideig tart egy megoldás megtalálása?
- **Tárigény** (**space complexity**): a keresés elvégzéséhez mennyi memóriára van szükség?

Az idő- és tárigényről beszélve mindig a probléma nehézségének valamilyen mértékét tartjuk szem előtt. Az elméleti számítástudományban egy tipikus ilyen mérték az állapottér gráf nagysága, mivel a gráf a kereső program bemenetére adott explicit adatstruktúrának tekinthető (erre egy példa Románia térképe). Az MI-ben, ahol a gráfot implicit formában a kezdeti állapottal és az állapotátmenet-függvénnyel reprezentáljuk, és ahol a gráf sokszor végtelen, a komplexitást három tényezővel fejezzük ki. Ezek: b – az **elágazási tényező** (**branching factor**), vagyis a követők maximális száma minden csomópontban, d – a legsekélyebb célállapot mélysége és m – az állapottérben található utak maximális hossza.

Az időt gyakran a keresés közben generált[5] csomópontok számával, a tárat pedig a memóriában maximálisan tárolt csomópontok számával mérik.

A keresés hatékonyságának becslésénél gondolhatunk a **keresési költség**re (**search cost**), amely tipikusan az időigénytől függ, de tartalmazhat egy tárigényt jelző komponenst is; vagy használhatjuk a keresés **összköltség**ét (**total cost**), amely a megoldás útköltségét és a keresési költséget kapcsolja össze. Az Aradról Bukarestbe vezető útkeresési probléma esetén a keresési költség a keresés ideje, a megoldási költség pedig a teljes út hossza km-ben. Így amikor a teljes költséget akarjuk kiszámítani, kilométert és milliszekundumot kellene összeadnunk. Ez nem mindig egyszerű, mert nem létezik semmilyen „hivatalos váltószám” a kettő között, ennél a problémánál

[5] Egyes források ehelyett az időt a kifejtett csomópontok számával mérik. A két mérték legfeljebb egy b tényezőben tér el. Nekünk úgy tűnik, hogy a csomópontkifejtés végrehajtási ideje nő az adott kifejtésben generált csomópontok számával.

Created by XMLmind XSL-FO Converter.

azonban értelmesnek tűnhet a kilométereket milliszekundumokra átszámítani a gépkocsi átlagsebességét használva (mert az ágensnek az idő a fontos). Ez lehetővé teszi, hogy az ágens megtalálja azt a kompromisszumot, amikor a legrövidebb út keresését célzó minden további számítás improduktívvá válik. A különböző javak közötti kompromisszum általános problémájára a 16. fejezetben még visszatérünk.

4. Nem informált keresés

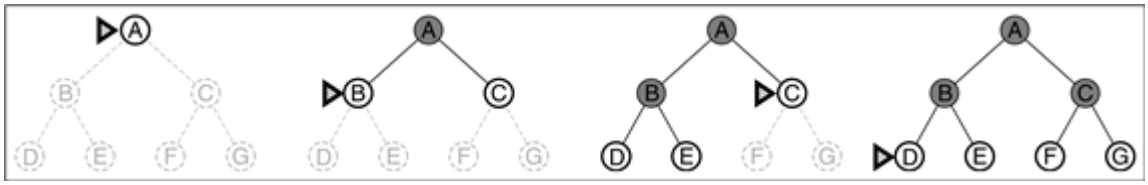
Ez az alfejezet öt keresési stratégiát tárgyal, amelyek a **nem informált** (**vak**nak is nevezett) **keresés** (**noninformed** (**blind**) **search**) cím alá sorolhatók. A kifejezés azt jelenti, hogy ezen stratégiáknak semmilyen információjuk nincs az állapotokról a probléma definíciójában megadott információn kívül. Működésük során mást nem tehetnek, mint a következő állapotok generálása és a célállapot megkülönböztetése a nem célállapottól. Azokat a stratégiákat, amelyek tudják, hogy az egyik közbülső állapot „ígéretesebb", mint egy másik közbülső állapot, **informált keresés**i (**informed search**) vagy **heurisztikus keresés**i (**heuristic search**) stratégiának nevezzük. Ezeket majd a 4. fejezet tárgyalja. A keresési stratégiákat a csomópontok kifejtési *sorrendje* különbözteti meg egymástól.

4.1. Szélességi keresés

A **szélességi keresés** (**breadth-first search**) egy egyszerű keresési stratégia, ahol először a gyökércsomópontot fejtjük ki, majd a következő lépésben az összes a gyökércsomópontból generált csomópontot, majd *azok* követőit stb. Általánosságban a keresési stratégia minden adott mélységű csomópontot hamarabb fejt ki, mielőtt bármelyik, egy szinttel lejjebbi csomópontot kifejtené.

A szélességi keresést meg lehet valósítani a `FA-KERESÉS` algoritmussal egy olyan üres peremmel, amely egy először-be-először-ki (first-in-first-out – FIFO) sor, biztosítva ezzel, hogy a korábban meglátogatott csomópontokat az algoritmus korábban fejti ki. Más szóval a `FA-KERESÉS` (*probléma*, FIFO-`SOR` ()) meghívása szélességi keresést eredményez. A FIFO sor az összes újonnan legenerált követőt a sor végére teszi, magyarán a sekélyebb csomópontok korábban kerülnek kifejtésre, mint a mélyebben fekvők. A 3.10. ábra illusztrálja a keresés előrehaladását egy egyszerű bináris fa esetén.

3.10. ábra - Szélességi keresés egy egyszerű bináris fában. Minden lépésnél a következő kifejtendő csomópontot egy marker jelzi.



A szélességi keresés elemzéséhez az előbbi részben tárgyalt négy jellemzőt fogjuk használni. Könnyű belátni, hogy ez a keresés *teljes*. Ha a legsekélyebb célcsomópont valamilyen véges d mélységben fekszik, a szélességi keresés eljut hozzá az összes nála sekélyebben fekvő csomópontot kifejtve (feltéve persze, hogy a b elágazási tényező véges). A *legsekélyebb* célcsomópont nem szükségképpen *optimális*. Pontosabban a szélességi keresés optimális, ha az útköltség a csomópont mélységének nem csökkenő függvénye (például ha minden cselekvésnek ugyanannyi a költsége).

Eddig a szélességi keresésnek csak a jó tulajdonságait láttuk. Ahhoz, hogy megértsük miért nem mindig ezt a stratégiát választjuk, meg kell vizsgálnunk a keresés végrehajtásához szükséges idő és memória mennyiségét. Ehhez egy olyan hipotetikus állapotteret veszünk alapul, amelyben minden egyes állapotot kifejtve b új állapot keletkezik. A keresési fa gyökércsomópontja b csomópontot generál az első szinten, amelyek mindegyike újabb b csomópontot, összesen b^2 csomópontot generál a második szinten. *Ezek* mindegyike újabb b csomópontot generál, összesen b^3 csomópontot a harmadik szinten és így tovább. Tételezzük fel, hogy ezen probléma megoldása d mélységben található. Ekkor a legrosszabb esetben a d-edik szinten az utolsót kivéve (mert a célt magát nem fejtjük ki) a csomópontok mindegyikét ki kell fejtenünk, a (d + 1)-edik szinten $b^{d+1} - b$ csomópontot generálva. A generált csomópontok össz-száma így:

$$b + b^2 + b^3 + \ldots + b^d + (b^{d+1} - b) = O(b^{d+1})$$

Created by XMLmind XSL-FO Converter.

Minden legenerált csomópontot a memóriában el kell tárolni, mert vagy a perem eleme, vagy egy perembeli csomópont őse. A tárigény így az időigénnyel azonos (meg egy további csomópont a gyökér számára).

Azok, akik járatosak a komplexitáselemzésben, kezdenek aggódni (vagy izgatottak lesznek, ha szeretik a kihívásokat), amikor exponenciális komplexitást látnak, mint amilyen például az $O(b^{d+1})$. A 3.11. ábra megmutatja, hogy miért. Az ábra egy $b = 10$ elágazási tényezővel rendelkező szélességi keresést mutat a d megoldás mélység több értékére. A táblában feltételezzük, hogy másodpercenként 10 000 csomópontot generálunk, illetve egy csomópont tárolásához 1000 bájtra van szükség. Számos feladvány jellegű probléma felel meg ezeknek a feltételezéseknek (egy 100-as tényezővel ide vagy oda), ha azokat modern személyi számítógépeken futtatjuk.

Fontos

A 3.11. ábra alapján két tanulságot vonhatunk le. Először is a *szélességi keresés esetén a tárigény nagyobb problémát jelent az időigénynél.* A legtöbb ember, amennyiben érdekli a válasz egy fontos problémára, türelmesen ki tud várni 31 órát, hogy egy 8 mélységű keresés lefusson, de csak kevés számítógépnek van a kereséshez szükséges Tbájtnyi memóriája. Szerencsére léteznek ennél kevesebb memóriát igénylő keresési algoritmusok is.

3.11. ábra - A szélességi keresés idő- és tárigénye. Az ábra adatai *b* = 10-es elágazási tényezőt, 10 000 csomópont/percet és 1000 bájt/csomópontot feltételeznek.

Mélység	Csomópontok	Időigény	Tárigény
2	1100	0,11 másodperc	1 Mbájt
4	111100	11 másodperc	106 Mbájt
6	10^7	19 perc	10 Gbájt
8	10^9	31 óra	1 Tbájt
10	10^{11}	129 nap	101 Tbájt
12	10^{13}	35 év	10 Pbájt
14	10^{15}	3523 év	1 Ebájt

Fontos

A második tanulság, hogy az időigény még mindig fontos tényező. Ha a problémánk 12 mélységű, akkor (a feltételezéseink mellett) a szélességi keresésnek (vagy akármlyik nem informált keresési algoritmusnak) 35 évbe telne a megoldás megtalálása. Általánosságban *az exponenciális komplexitású keresési problémák közül csak a legkisebb problémapéldányok oldhatók meg.*

4.2. Egyenletes költségű keresés

A szélességi keresés optimális, ha minden lépés költsége azonos, mert mindig a *legsekélyebb* ki nem fejtett csomópontot fejti ki. Egyszerű általánosítással egy olyan algoritmust találhatunk ki, amely tetszőleges lépésköltség mellett optimális. Az **egyenletes költségű keresés** (**uniform cost search**) mindig a *legkisebb útköltségű n* csomópontot fejti ki először, nem pedig a legkisebb mélységű csomópontot. Egyszerűen belátható, hogy a szélességi keresés is egyenletes költségű keresés, amennyiben minden lépésköltség azonos.

Az egyenletes költségű keresés nem foglalkozik azzal, hogy *hány* lépésből áll egy bizonyos út, hanem csak az összköltségükkel törődik. Emiatt mindig végtelen hurokba kerül, ha egy csomópont kifejtése zérus költségű cselekvéshez és ugyanahhoz az állapothoz való visszatérést eredményez (például a *NoOp* cselekvés). A teljességet csak úgy garantálhatjuk, hogy minden lépés költsége egy kis pozitív e konstansnál nagyobb, vagy azzal egyenlő. Ez a feltétel egyben az *optimalitás* elégséges feltétele is. Ez azt jelenti, hogy egy út költsége az út mentén mindig növekszik. Ebből a tulajdonságból látszik, hogy az algoritmus a csomópontokat mindig a növekvő útköltség függvényében fejti ki. Azaz az első kifejtésre kiválasztott célcsomópont egyben az optimális megoldás is (emlékezzünk arra, hogy a `FA-KERESÉS` a célállapottesztet csak a kifejtésre megválasztott csomópontokra alkalmazza). Javasoljuk, hogy próbálják ki az algoritmust, hogy a Bukarestbe vezető legrövidebb utat megtalálják.

Created by XMLmind XSL-FO Converter.

Az egyenletes költségű keresést nem a mélység, hanem az útköltség vezérli, így komplexitását a b és a d függvényében nehéz jellemezni. Helyette legyen C^* az optimális megoldás költsége, és $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$ hogy minden cselekvés költsége legalább e. Az algoritmus idő- és tárigénye legrosszabb esetben , ami sokkal több lehet, mint b^d. Ez azért lehetséges, mert az egyenletes költségű keresés képes (és sokszor ezt meg is teszi) a kis lépésekből álló nagy fákat felkuta $(b^{1+\lfloor C^*/\varepsilon \rfloor})$ feltehetően hasznos lépéseket tartalmazó utak előtt. Amikor minden lépés költsége ugyanannyi, az , persze azonos b^d-nel.

4.3. Mélységi keresés

A **mélységi keresés** (**depth-first search**) mindig a keresési fa aktuális peremében a *legmélyebben* fekvő csomópontot fejti ki. A keresés lefolyását a 3.12. ábra illusztrálja. A keresés azonnal a fa legmélyebb szintjére jut el, ahol a csomópontoknak már nincsenek követőik. Kifejtésüket követően kikerülnek a peremből és a keresés „visszalép” ahhoz a következő legmélyebben fekvő csomóponthoz, amelynek vannak még ki nem fejtett követői.

Ez a stratégia egy olyan `FA-KERESÉS` függvénnyel implementálható, amelynek a sorbaállító függvénye az utolsónak-be-elsőnek-ki (last-in-first-out, LIFO), más néven verem. A mélységi keresést szokás a `FA-KERESÉS` függvény alternatívájaként egy rekurzív függvénnyel is implementálni, amely a gyermekcsomópontokkal meghívja önmagát (mélységkorláttal dolgozó rekurzív mélységi keresés algoritmusát a 3.13. ábra mutatja).

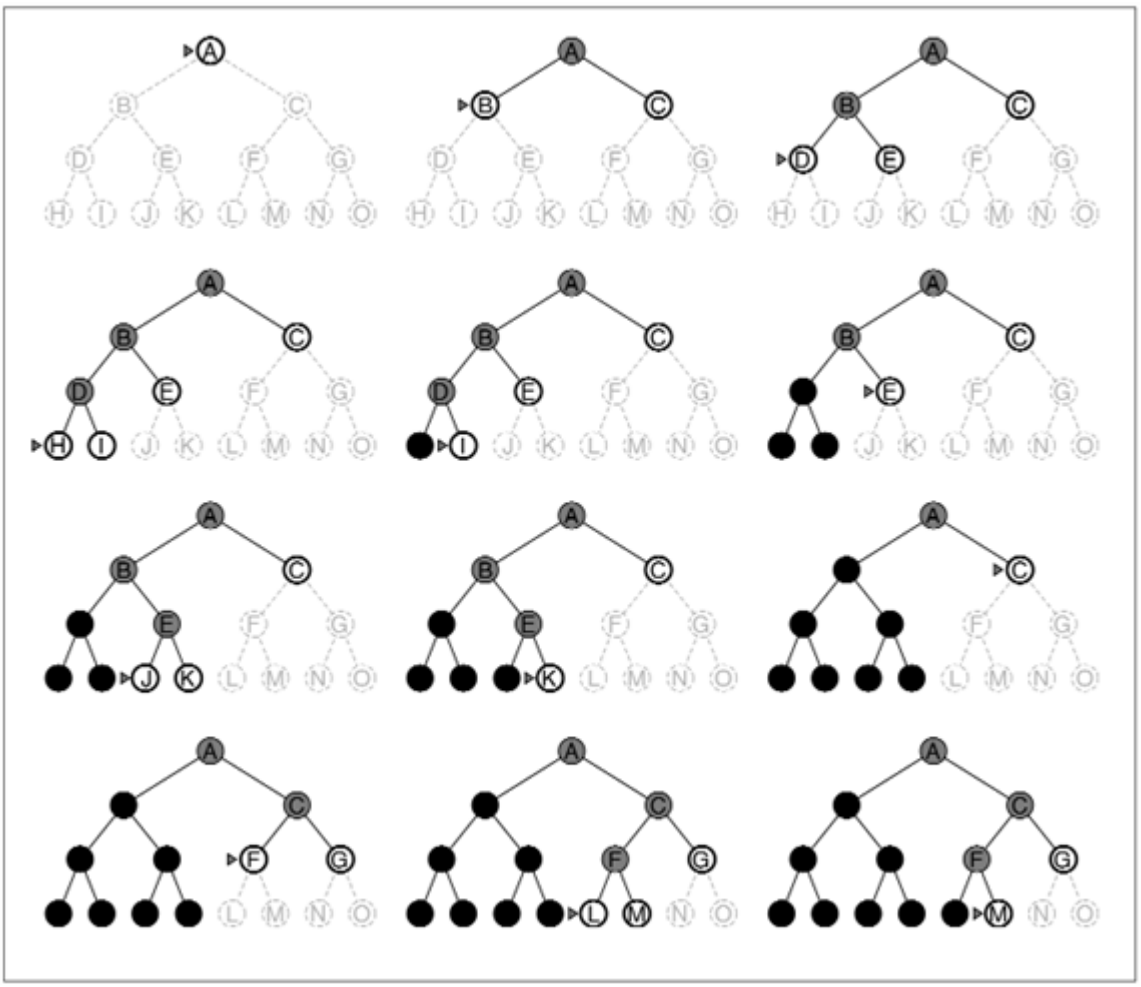
A mélységi keresés nagyon szerény tárigényű. Csak egyetlen, a gyökércsomóponttól egy levélcsomópontig vezető utat kell tárolnia, kiegészítve az út minden egyes csomópontja melletti kifejtetlen csomópontokkal. Egy kifejtett csomópont el is hagyható a memóriából, feltéve, hogy az összes leszármazottja meg lett vizsgálva. Egy b elágazási tényezőjű és m maximális mélységű állapottér esetén a mélységi keresés tárigénye $bm + 1$. A 3.11. ábra feltételezéseivel élve és feltételezve, hogy a célcsomópont mélységű csomópontoknak nincsenek követőik, azt találjuk, hogy például $d = 12$ mélység esetén a mélységi keresés 118 kbájtot igényelne a 10 Pbájttal szemben, ami tízmilliárdos redukciót jelent a tárigényben.

A mélységi keresés **visszalépéses keresés**nek (**backtracking search**) nevezett változata még kevesebb memóriát használ. A visszalépéses keresés az összes követő helyett egyidejűleg csak egy követőt generál. Minden részben kifejtett csomópont emlékszik, melyik követője jön a legközelebb. Ily módon csak $O(m)$ memóriára van szükség, $O(bm)$ helyett. A visszalépéses keresés még egy memória- (és idő-) spóroló trükkhöz folyamodik. Az ötlet a követő csomópont generálása az aktuális állapot *módosításával*, anélkül hogy az állapotot átmásolnánk. Ezzel a memóriaszükséglet egy állapotra és $O(m)$ cselekvésre redukálódik. Ahhoz, hogy az ötlet működjön, amikor visszalépünk, hogy a következő követőt generáljuk, mindegyik módosítást vissza kell tudnunk csinálni. Nagy állapottérrel rendelkező problémák esetén, mint például robot-összeszerelés esetén, az ilyen módszerek lényegesek a sikerességhez.

A mélységi keresés hátrányos tulajdonsága, hogy egy rossz választással egy hosszú (akár végtelen) út mentén lefelé elakadhat, miközben például egy más döntés elvezetne a gyökérhez közeli megoldáshoz. A 3.12. ábrán például a mélységi keresés kifejti az egész bal oldali részfát, annak ellenére, hogy a C csomópont a megoldás. Ha a J csomópont szintén megoldás lenne, a mélységi keresés azt adná vissza megoldásul, következésképpen a mélységi keresés nem optimális. Ha a bal oldali részfa korlátlanul mély lenne és nem tartalmazna megoldást, a mélységi keresés soha nem állna meg, következésképpen a mélységi keresés nem teljes. A legrosszabb esetben a mélységi keresés a keresési fában az összes $O(b^m)$ csomópontot generálni fogja, ahol m a csomópontok maximális mélysége. Jegyezzük meg, hogy m sokkal nagyobb lehet, mint d (a legsekélyebb megoldás mélysége), és korlátlan fák esetén értéke végtelen.

3.12. ábra - Mélységi keresés egy bináris keresési fában. A kifejtett csomópontok, amelyeknek a peremben nincsenek követőik, el is hagyhatók a memóriából. Ezeket feketével jelöltük meg. A 3-as mélységű csomópontokról feltételezzük, hogy nincsenek követőik, valamint azt is feltesszük, hogy *M* az egyetlen célcsomópont.

Created by XMLmind XSL-FO Converter.



4.4. Mélységkorlátozott keresés

A végtelen fák problémáját a mélységi keresés azáltal küszöböli ki, hogy az utak maximális mélységére egy ℓ korlátot ad. Az ℓ mélységben lévő csomópontokat úgy kezeli, mintha nem is lennének követőik. A módszer neve a **mélységkorlátozott keresés**, **MKK** (**depth-limited search**, **DLS**). A mélységkorlát a végtelen út problémáját ugyan megoldja, de a nemteljesség egy újabb forrását hozza be, ha $\ell < d$ -t választunk, azaz, ha a legsekélyebb célcsomópont a mélységkorláton túl van (ez nem is esélytelen, ha d eleve ismeretlen). A mélységkorlátozott keresés $\ell > d$ választással sem lesz optimális. A keresés időigénye $O(b^{\ell})$, tárigénye $O(b\ell)$. A mélységi keresés egy olyan speciális mélységkorlátozott keresésének tekinthető, amelynek mélységkorlátja $\ell = \infty$.

A mélységi korlátot néha a probléma ismeretére lehet alapozni. Például Románia térképén 20 város található, így tudjuk, hogy ha létezik egy megoldás, az maximálisan 19 lépés hosszú lehet, így az $\ell = 19$ egy lehetséges választás. Ha azonban a térképet tüzetesebben tanulmányoznánk, felfedeznénk, hogy minden város bármelyik másik városból legfeljebb 9 lépésben elérhető. Ez a szám, amit az állapottér **átmérő**jének (**diameter**) nevezünk, jobb mélységkorlátot ad, ami hatékonyabb mélységkorlátozott keresést eredményez. A legtöbb probléma esetén azonban mindaddig nem tudunk jó mélységkorlátot adni, amíg meg nem oldottuk a problémát.

A mélységkorlátozott keresést az általános fakeresési vagy a rekurzív mélységi keresési algoritmus egyszerű módosításával lehet implementálni. A 3.13. ábra mutatja a rekurzív mélységkorlátozott keresés pszeudokódját. Jegyezzük meg, hogy a mélységkorlátozott keresés kudarccal kétféle módon állhat le: a standard *kudarc* csomópont jelzi a megoldás hiányát, a *vágás* érték viszont jelzi a megoldás mélységkorláton belüli hiányát.

3.13. ábra - A mélységkorlátozott keresés rekurzív implementációja

Created by XMLmind XSL-FO Converter.

```
function MÉLYSÉG-KORLÁTOZOTT-KERESÉS(probléma, korlát) returns egy megoldás vagy kudarc/vágás
  return REKURZÍV-MKK(CSOMÓPONTOT-LÉTREHOZ(KIINDULÓ-ÁLLAPOT[probléma]), probléma, korlát)

function REKURZÍV-MKK(csomópont, probléma, korlát) returns egy megoldás vagy kudarc/vágás
  vágás-megtörtént? ← hamis
  if CÉL-TESZT[probléma](ÁLLAPOT[csomópont]) then return MEGOLDÁS(csomópont)
  else if MÉLYSÉG[csomópont] = korlát then return vágás
  else for each követő in KIFEJT(csomópont, probléma) do
       eredmény ← REKURZÍV-MKK(követő, probléma, korlát)
       if eredmény = vágás then vágás-megtörtént? ← igaz
       else if eredmény ≠ kudarc then return eredmény
  if vágás-megtörtént? then return vágás else return kudarc
```

4.5. Iteratívan mélyülő mélységi keresés

Az **iteratívan mélyülő keresés** (**iterative deepening search**) – vagy iteratívan mélyülő mélységi keresés – egy általános stratégia, amit sokszor a mélységi kereséssel együtt alkalmaznak a legjobb mélységkorlát megtalálására. Az algoritmus képes erre, mert fokozatosan növeli a mélységkorlátot – legyen az először 0, majd 1, majd 2 stb. – amíg a célt meg nem találja. Ez akkor következik be, ha a mélységkorlát eléri a *d*-t, a legsekélyebben fekvő célcsomópont mélységét. Az algoritmust a 3.14. ábra mutatja. Az iteratívan mélyülő keresés ötvözi a szélességi és a mélységi keresés előnyös tulajdonságait. A mélységi kereséshez hasonlóan szerény, pontosabban $O(bd)$ memóriaigénnyel rendelkezik. A szélességi kereséshez hasonlóan teljes, ha elágazási tényezője véges, és optimális, ha az útköltség a csomópontok mélységének nem csökkenő függvénye. A 3.15. ábra az `ITERATÍVAN-MÉLYÜLŐ-KERESÉS` első négy iterációját mutatja egy bináris fán, ahol az algoritmus a megoldást a negyedik iterációban találja meg.

Az iteratívan mélyülő keresés tékozlónak tűnhet, mert felettébb sok állapotot többször is kifejt. Kiderül azonban, hogy a költségtöbblet nem lényeges. Ennek az az oka, hogy egy olyan keresési fában, ahol minden szinten ugyanaz (vagy közel ugyanaz) az elágazási tényező, majdnem az összes csomópont a legmélyebb szinten található, így nem túl sokat számít, hogy a magasabb szinteket többször is kifejtjük. Az iteratívan mélyülő keresésben a legmélyebb szinten (*d* mélység) található csomópontokat csak egyszer fejtjük ki, egy szinttel feljebb kétszer stb. egészen a gyökér gyerekeiig, amelyeket *d*-szer fejtünk ki. Így a kifejtett csomópontok összszáma:

$$Cs(\text{IMK}) = (d)b + (d-1)b^2 + \ldots + (1)b^d$$

amely $O(b^d)$ időkomplexitást eredményez. Összehasonlításul nézzük meg a szélességi keresés által generált csomópontok számát:

$$Cs(\text{SZK}) = b + b^2 + \ldots + b^d + (b^{d+1} - b)$$

Vegyük észre, hogy a szélességi keresés $d + 1$ mélységben is generál csomópontokat, az iteratívan mélyülő kereséssel ellentétben. Ennek eredménye, hogy az iteratívan mélyülő keresés a szélességi keresésnél *gyorsabb*, annak ellenére, hogy a csomópontokat többször fejti ki. Konkretizálva, például $b = 10$ és $d = 5$ esetén ezek a számok:

$$Cs(\text{IMK}) = 50 + 400 + 3000 + 20\,000 + 100\,000 = 123\,450$$

$$Cs(\text{SZK}) = 10 + 100 + 1000 + 10\,000 + 100\,000 + 999\,990 = 1\,111\,100$$

Fontos

Általánosságban nagy keresési térrel rendelkező problémák esetén és ha a megoldás mélysége nem ismert, a nem informált módszerek köréből az iteratívan mélyülő keresés a javasolt.

3.14. ábra - Az iteratívan mélyülő keresési algoritmus mélységkorlátozott keresést alkalmaz ismételten, növekvő mélységkorláttal. Az algoritmus megáll, ha a megoldást

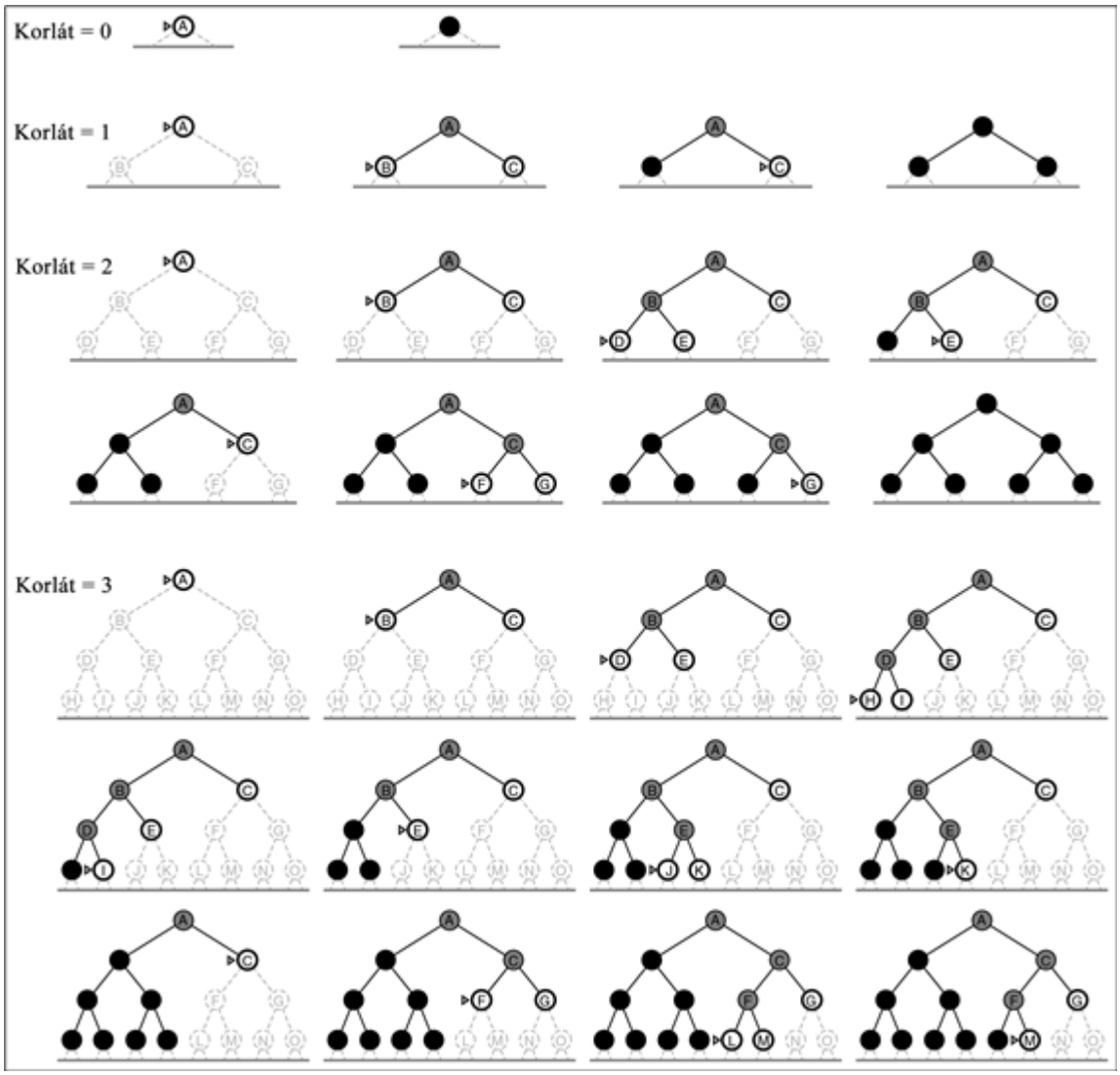
Created by XMLmind XSL-FO Converter.

megtalálja, vagy ha a mélységkorlátozott keresés kudarccal tér vissza, jelezve, hogy megoldás nem létezik.

```
function ITERATÍVAN-MÉLYÜLŐ-KERESÉS(probléma) returns egy megoldás vagy kudarc
  inputs: probléma, egy probléma

  for mélység ← 0 to ∞ do
    eredmény ← MÉLYSÉGKORLÁTOZOTT-KERESÉS(probléma, mélység)
    if eredmény ≠ vágás then return eredmény
```

3.15. ábra - Az iteratívan mélyülő keresési algoritmus négy iterációja egy bináris fán



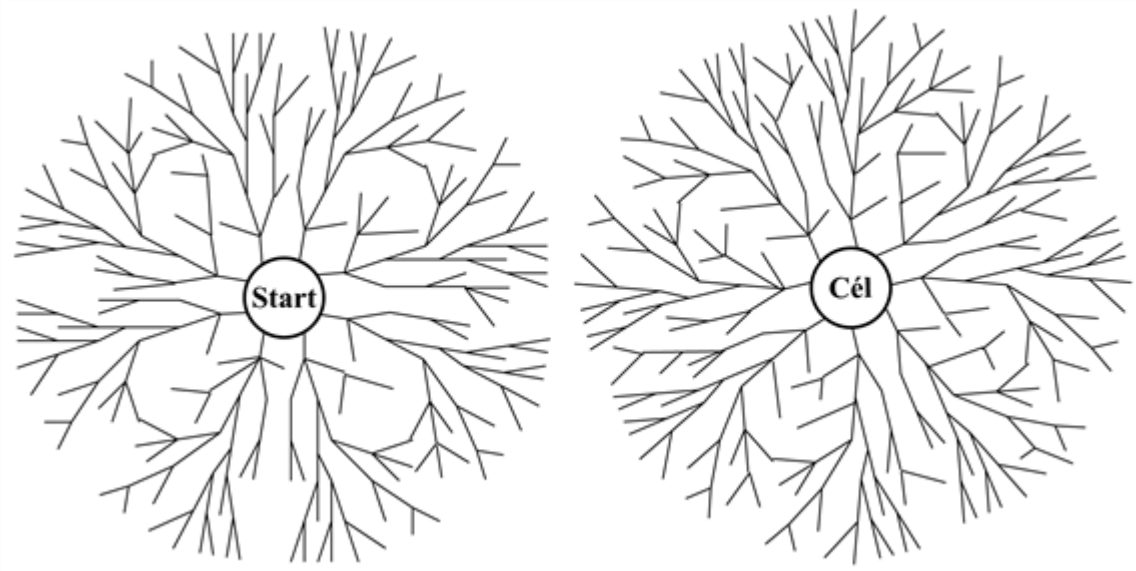
Az iteratívan mélyülő keresés a szélességi kereséssel abban rokon, hogy minden iterációban a csomópontok teljes rétegét megvizsgálja, mielőtt a következő rétegre térne rá. Hasznosnak látszik az egyenletes költségű keresés iteratív változatának kifejlesztése, amely örökölné ez utóbbi optimalitását, mellőzve annak a tárkövetelményeit. Az ötlet a növekvő útköltségkorlát használata a növekvő mélységkorlát helyett. Az eredményül kapott algoritmussal, amelynek neve **iteratívan megnyúló keresés (iterative lengthening search)** a 3.11. feladat foglalkozik. Sajnos az derül ki, hogy az iteratívan megnyúló keresés overheadje tekintélyes az egyenletes költségű kereséshez képest.

Created by XMLmind XSL-FO Converter.

4.6. Kétirányú keresés

A kétirányú keresés mögött az az ötlet húzódik, hogy egyszerre el lehet indítani egy keresést előrefelé a kiinduló állapotból, illetve hátrafelé a célállapotból, és a keresés akkor fejeződik be, ha a két keresés valahol találkozik (lásd 3.16. ábra). Az érv az, hogy $bd^{/2} + bd^{/2}$ sokkal kisebb, mint b^d, illetve az ábrán szemlélve, hogy a két kisebb kör összterülete kisebb, mint annak a nagy körnek a területe, amelynek középpontja a kiinduló állapot, és amely a peremével a célállapotot eléri.

3.16. ábra - A kétirányú szélességi keresés sematikus ábrája. Az ábrán a két keresési irány majdnem találkozik, amikor a kiinduló csomópontból kinyúló egyik ág összeér egy a célcsomópontból kinyúló másik ággal.



A kétirányú keresést úgy implementálják, hogy az egyik vagy mindkét keresés egy csomópont kifejtése előtt megvizsgálja, hogy az nem része-e a másik keresési fa peremének. Ha igen, megvan a cél. Ha a probléma például $d = 6$ megoldás mélységű, és mindegyik irányban a szélességi keresést futtatjuk csomópontonként, a két keresés a legrosszabb esetben akkor találkozik, ha mindegyik algoritmus a 3-as mélységben egy csomópont kivételével minden csomópontot kifejtett. A $b = 10$ esetén ez 22 200 csomópont generálását jelenti a standard szélességi keresés által generált 1 111 111 csomóponthoz képest. Annak ellenőrzését, hogy egy csomópont a másik keresési fához tartozik-e, egy hash-táblával konstans időben meg lehet oldani. A kétirányú keresés időkomplexitása így $O(bd^{/2})$. Legalább az egyik keresési fát a memóriában kell tartani, hogy a tartozás ellenőrzése kivitelezhető legyen, a tárkomplexitás tehát szintén $O(bd^{/2})$. Ez a tárkomplexitás a kétirányú keresés legnagyobb gyengéje. Ha mindkét keresés szélességi keresés, az algoritmus teljes és optimális (egyenletes költség esetén). Más módszerek kombinációja vagy a teljesség, vagy az optimalitás, vagy mindkettő elvesztéséhez vezethet.

Az időkomplexitás mérséklése a kétirányú keresést igen vonzóvá teszi, de mit is jelent a célállapotból hátrafelé keresni? Ez nem is olyan egyszerű, mint amilyennek hangzik. Legyenek x csomópont **elődcsomópont**jai (**predecessor**s), az *Előd*(x)-ek azon csomópontok, amelyek mindegyikének x a követő csomópontja. A kétirányú keresés feltételezi, hogy egy *Előd*(x) hatékonyan számítható. Legegyszerűbb az az eset, amikor az összes cselekvés az állapottérben reverzíbilis, így *Előd*(x) = *Követő*(x). Más esetben azonban igen nagy ötletességre lehet szükség.

Nézzük most, hogy mit is jelent az, hogy „cél”, ha a célállapotból hátrafelé kell keresni. A 8-as kirakójáték és a romániai útkeresés esetén egyetlenegy célállapot létezik csak, így a hátrafelé keresés és az előrefelé keresés igen hasonlítanak egymásra. Amennyiben létezik a célállapotoknak egy *explicit listája,* mint például a 3.3. ábra két koszmentes célállapota, akkor megkonstruálhatunk egy olyan ál-célállapotot, amelynek közvetlen követői az aktuális célállapotok. Más módon, néhány redundáns csomópont-generálás elkerülhető azzal, hogy a célállapotok halmazát egyetlenegy célállapotnak tekintjük, amelynek minden elődje szintén egy állapothalmaz – konkrétan azon állapotok halmaza, amelyek követője a célállapothalmaz eleme (lásd még 3.6. alfejezet).

Created by XMLmind XSL-FO Converter.

A kétirányú keresés szempontjából a legnehezebb eset, amikor a célállapottesztnél a feltehetően nagy célállapothalmazról csak implicit *leírás* áll rendelkezésünkre, például a sakkban az összes állapot, ami kielégíti a „matt" célt. A hátrafelé keresésnek az „m_1 cselekvés révén a »matt« állapotba vezető összes állapot" tömör leírását kellene tudnia megkonstruálni, és hasonló módon folytatni. E leírásokat az előrefelé haladó keresés által generált állapotokkal kellene tesztelni. Ennek nincs általánosan hatékony módja.

4.7. A keresési stratégiák összehasonlítása

A 3.17. ábra a 3.4. alfejezetben megfogalmazott négy kiértékelési kritérium tükrében összehasonlítja a keresési stratégiákat.

3.17. ábra - A keresési stratégiák értékelése. *b* az elágazási tényező, *d* a legsekélyebb megoldás mélysége, *m* a keresési fa maximális mélysége, *ℓ* a mélységkorlát. A felső indexszel jelzett kikötések a következők: [a] teljes, ha *b* véges; [b] teljes, ha a lépésköltség ≥ ε, pozitív ε-ra; [c] optimális, ha a lépésköltségek mind azonosak; [d] ha mindkét irányban szélességi keresést használunk.

Kritérium	Szélességi	Egyenletes költségű	Mélységi korlátozott	Mélység-mélyülő	Iteratívan	Kétirányú (amennyiben alkalmazható)
Teljes?	Igen[a]	Igen[a, b]	Nem	Nem	Igen[a]	Igen[a, d]
Időigény	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Tárigény	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Optimális?	Igen[c]	Igen	Nem	Nem	Igen[c]	Igen[c, d]

5. Az Ismételt állapotok elkerülése

Eddig figyelmen kívül hagytuk a keresés egyik legfontosabb megoldandó problémáját: a már korábban egy másik úton megtalált és kifejtett állapotok ismételt kifejtéséből adódó időpazarlást. Néhány probléma esetén ezen lehetőség soha nem merül fel, mert az állapottér egy fa, és minden állapotba csak egyetlen módon lehet eljutni. A 8-királynő probléma hatékony megfogalmazása – amikor minden új királynőt a bal szélső szabad oszlopba helyezzük – nagyrészt épp ennek köszönheti hatékonyságát, vagyis hogy minden egyes állapotba csak egyetlen úton lehet eljutni. Ha a 8-királynő problémát úgy fogalmazzuk meg, hogy egy királynőt báramelyik oszlopban el lehet helyezni, akkor az *n* királynőt tartalmazó minden állapotot *n*! különböző úton el lehet érni.

Számos problémánál azonban elkerülhetetlenek a megismételt állapotok. Ezek közé tartozik az összes olyan probléma, amelyben az operátorok reverzíbilisek. Többek között ebbe a csoportba tartoznak az útkeresési problémák és a csúszó lapka fejtörő játékok. Az ezen problémákhoz tartozó keresési fák végtelenek, de ha a megismételt állapotok egy részét levágjuk, akkor a keresési fát véges méretűre vághatjuk, ezáltal a keresési fának csak az állapottér gráfot kifeszítő részét generálva. Szélsőséges esetben egy $d + 1$ méretű állapottérből (lásd 3.18. (a) ábra) egy 2^d levelű fa lesz (lásd 3.18. (b) ábra). Egy élethűbb példa a **négyzetrács** (**rectangular grid**) (lásd 3.18. (c) ábra). A rácsban minden állapotnak négy követője van. Az ismétlődő állapotokat tartalmazó fának így 4^d levele van. Minden adott állapottól *d* lépésnyire azonban csak kb. $2d^2$ különböző állapot van. $d = 20$ esetén ez kb. egybillió csomópontot, de csak 800 különböző állapotot jelent.

Az ismétlődő állapotok tehát a megoldható problémákat megoldhatatlan problémákká alakítják, amennyiben az algoritmus nem képes ezeket az állapotokat detektálni. A detektálás általában azt jelenti, hogy az új kifejtendő csomópontot a már kifejtett csomópontokkal hasonlítjuk össze. Egyezés esetén az adott csomóponthoz az algoritmus két utat talált, és valamelyiket eldobhatja.

Fontos

A mélységi keresés csak a gyökeret és az aktuális csomóponttal összekötő úton fekvő állapotokat tárolja. E csomópontoknak az aktuális csomóponttal való összehasonlítása lehetővé teszi a hurkok felismerését, amiket ezek után azonnal el lehet dobni. Ez meg is véd attól, hogy a hurkok révén véges állapotterekből ne keletkezzenek végtelen keresési fák. Sajnos azonban nem tud megvédeni a nem

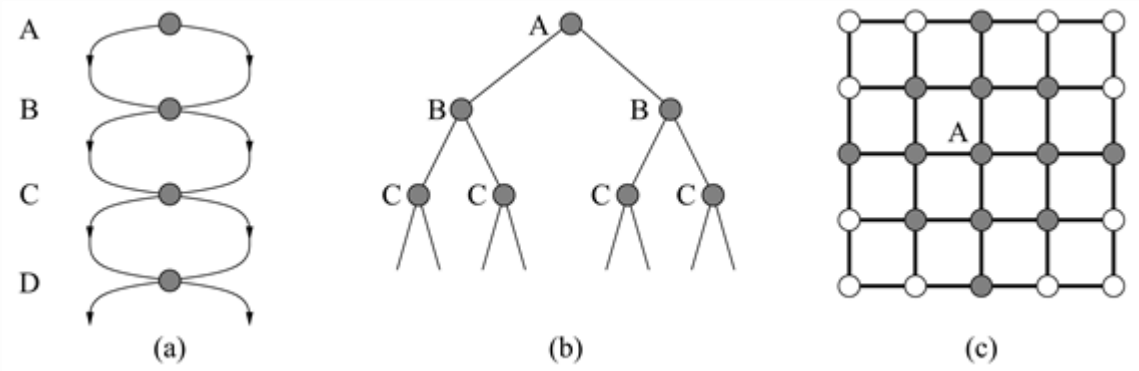
Created by XMLmind XSL-FO Converter.

hurkos utak exponenciális megsokszorozódásától az olyan problémákban, mint amilyenek a 3.18. ábrán látható. Ezt elkerülni csak úgy lehet, hogy több csomópontot tartunk a memóriában. Az idő és a tárigény között egy alapvető kompromisszum létezik. *Az az algoritmus, amely elfelejti a történetét, kénytelen azt megismételni*.

Ha egy algoritmus minden meglátogatott állapotra emlékszik, akkor közvetlen módon fedezi fel az állapotteret. A FA-KERESÉS algoritmust módosíthatjuk úgy, hogy egy **zárt listá**nak (**closed list**) nevezett adatszerkezetet tartalmazzon, amely minden kifejtett csomópontot tárol. (A még ki nem fejtett csomópontokból álló peremet néha **nyitott listá**nak [**open list**] nevezik.) Ha az aktuális állapot egybeesik a zárt listán lévő állapotok egyikével, akkor eldobható, ahelyett hogy a kifejtésével kellene foglalkozni. Az új algoritmus neve GRÁF-KERESÉS (3.19. ábra). Az ismétlődő állapotokat tartalmazó problémák esetén a GRÁF-KERESÉS sokkal hatékonyabb, mint a FA-KERESÉS. Az algoritmus a legrosszabb esetre számított tár- és időkomplexitása arányos az állapottér méretével, ami sokkal kisebb lehet, mint $O(b^d)$.

A gráfkeresés optimalitása már trükkös dolog. Korábban azt mondtuk, hogy egy ismételt állapot detektálásával az algoritmus két utat talált ugyanahhoz az állapothoz. A 3.19. ábrán látható GRÁF-KERESÉS algoritmus mindig az *újonnan felfedezett* utat dobja el. Nyilvánvaló, hogy ha az újonnan felfedezett út rövidebb, mint az eredeti, a GRÁF-KERESÉS elvéthet egy optimális megoldást. Szerencsére ki tudjuk mutatni (lásd 3.12. feladat), hogy az egyenletes költségű és a konstans lépésköltségű szélességi keresés esetén ez nem történhet meg. E két optimális fakeresési stratégia egyben optimális gráfkeresési stratégia is. Az iteratívan mélyülő keresés azonban mélységi kifejtést használ és könnyen kerülhet egy csomópont felé vezető szuboptimális útra, mielőtt megtalálná a hozzá vezető optimális utat. Az iteratívan mélyülő gráfkeresésnél tehát egy újonnan felfedezett útnál meg kell vizsgálni, hogy ez nem jobb-e, mint az eredeti. És ha igen, szükség lehet az adott csomópont követőinél a mélységek és az útköltségek revíziójára.

3.18. ábra - Egy exponenciálisan nagyobb keresési fát generáló állapottér. (a) Egy olyan állapottér, amelyben két lehetséges cselekvés vezethet A-ból B-be, kettő B-ből C-be és így tovább. Ez az állapottér *d*+1 állapotot tartalmaz, ahol *d* a maximális mélység. (b) Az (a) állapottér megfelelő keresési fája, 2^d ággal, amely az állapotér 2^d útjának felel meg. (c) Egy négyzetrács. A kezdeti (A) állapottól kétlépésnyire fekvő állapotokat szürke színnel jelöltük.



3.19. ábra - Az általános gráfkereső algoritmus. A *zárt* halmazt egy hash-táblával lehet implementálni, hogy az ismételt állapotokat hatékonyan ellenőrizni tudjuk. Ez az algoritmus feltételezi, hogy az *s* állapothoz vezető első út a legolcsóbb (lásd szöveg).

Created by XMLmind XSL-FO Converter.

```
function GRÁF-KERESÉS(probléma, perem) returns egy megoldás vagy kudarc

  zárt lista ← egy üres halmaz
  perem ← BESZÚR(CSOMÓPONTOT-LÉTREHOZ(KIINDULÓ-ÁLLAPOT[probléma]), perem)
  loop do
    if ÜRES?(perem) then return kudarc
    csomópont ← VEDD-AZ-ELSŐ-ELEMET(perem)
    if CÉL-TESZT[probléma](ÁLLAPOT[csomópont]) then return MEGOLDÁS(csomópont)
    if ÁLLAPOT[csomópont] nem eleme a zárt listának then
      adjuk hozzá az ÁLLAPOT[csomópont]-ot a zárt listához
      perem ← BESZÚR-MIND(KIFEJT(csomópont, probléma), perem)
```

Jegyezzük meg, hogy a zárt lista használata azt jelenti, hogy a mélységi és az iteratívan mélyülő keresés tárigénye már nem lineáris. Mivel a GRÁF-KERESÉS algoritmus minden csomópontot a memóriában megtart, egyes keresési fajták kivitelezhetetlenek a memóriakorlátok miatt.

6. Keresés részleges információ mellett

A 3.3. alfejezetben azt tételeztük fel, hogy a környezet teljesen megfigyelhető és determinisztikus, valamint hogy az ágens tisztában van minden cselekvésének következményével. Ebből kifolyólag az ágens pontosan ki tudja számítani, hogy egy tetszőleges cselekvésszekvenciának milyen állapot az eredménye, és azt is tudja, hogy ő maga melyik állapotban van. Az érzékelései nem nyújtanak új információt az egyes cselekvések után. Mi történik, ha a cselekvésekről és az állapotokról alkotott tudás nem teljes? Azt találjuk, hogy a nemteljesség különböző formái három elkülönülő problématípushoz vezetnek:

1. **Szenzor nélküli** (**sensorless** vagy **conformant**) problémák: ha az ágensnek egyáltalán nincsenek szenzorai, akkor (a tudásának megfelelően) a több lehetséges kezdeti állapot egyikében lehet, és minden cselekvése a lehetséges követő állapotok egyikéhez vezethet.

2. **Eshetőségi** (**contingency**) problémák: ha a környezet részben megfigyelhető, vagy ha a cselekvések bizonytalanok, akkor az ágens érzékszervei új információt nyújtanak minden cselekvés után. Minden lehetséges érzékelés egy eshetőséget határoz meg, amire az ágensnek terveznie kell. A problémát **ellenfél problémá**nak (**adversarial**) hívják, ha a bizonytalanságot egy másik ágens cselekvései okozzák.

3. **Felfedezéses** (**exploratorial**) probléma: ha a környezet állapotai és cselekvései nem ismertek, az ágensnek külön cselekednie kell, hogy azokat felderíthesse. A felfedezéses problémákat az eshetőségi problémák szélsőséges változatának lehet tekinteni.

Példaként a porszívóvilág környezetét vesszük. Emlékezzünk vissza, hogy az állapottér nyolc állapotból áll (lásd 3.20. ábra). Három cselekvés van – *Jobbra*, *Balra* és *Szív* –, és a cél az összes kosz felszedése (a 7. és 8. állapot). Ha a környezet megfigyelhető, determinisztikus és teljesen ismert, akkor a problémát triviálisan bármelyik ismertetett algoritmussal meg lehet oldani. Így például ha a kezdeti állapot az 5. állapot, akkor a [*Jobbra*, *Szív*] cselekvésszekvencia a 8. célállapotot éri el. A fejezet további részében e probléma szenzor nélküli és eshetőségi változatával foglalkozunk. A felfedezéses problémákat a 4.5. alfejezet, az ellenfélproblémákat a 6. fejezet tárgyalja.

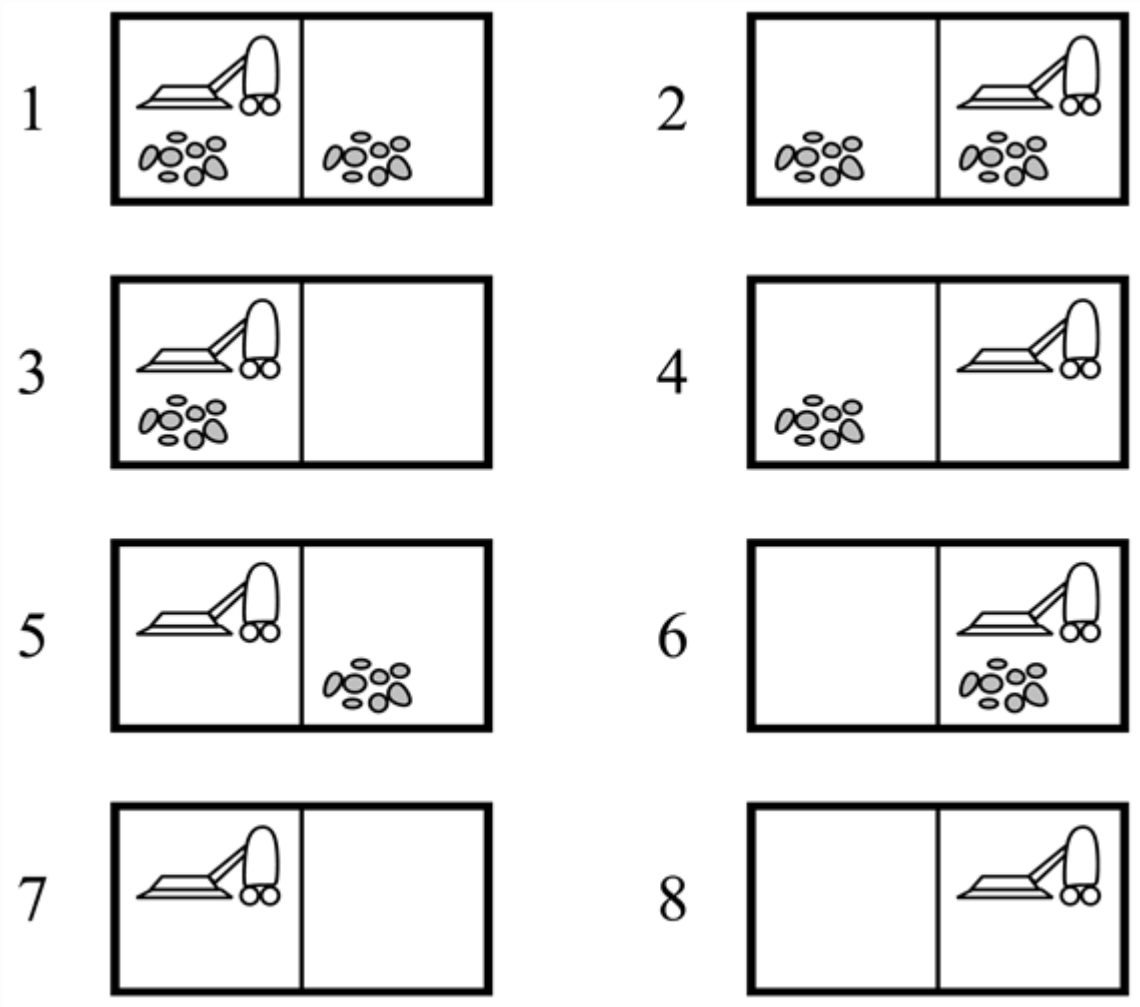
6.1. Szenzor nélküli problémák

Tegyük fel, hogy a porszívóágens ismeri az összes cselekvésének a hatását, de szenzorokkal nem rendelkezik. Ekkor csak azt tudja, hogy kezdeti állapota az {1, 2, 3, 4, 5, 6, 7, 8} halmaz egyik eleme. Azt lehetne gondolni, hogy az ágens sorsa reménytelen, azonban ágensünk igen jól feltalálja magát. Mivel tudja, hogy cselekvései mire képesek, kiszámíthatja például, hogy a *Jobbra* cselekvés a {2, 4, 6, 8} állapotok egyikébe viszi, és a *Jobbra*, *Szív* cselekvéssorozat eredményeképpen a {4, 8} egyikében köt ki. Végül a *Jobbra*, *Szív*, *Balra*, *Szív* sorozat garantáltan eléri a 7. célállapotot, függetlenül a kezdeti állapottól. Azt mondjuk, hogy az ágens a világot a 7. állapotba **bele** tudja **kényszerí**teni (**coerce**), akkor is, amikor nem is tudja, honnan indult. Összegezve: ha a világ nem teljesen megfigyelhető, az ágensnek az egyedi állapotokkal szemben azon *állapothalmazokra* kell tudnia következtetni, amely halmazok állapotaiba eljuthat. Minden ilyen állapothalmazt egy **hiedelmi**

Created by XMLmind XSL-FO Converter.

állapotnak nevezünk (**belief state**), amely az ágens pillanatnyi hiedelmét fejezi ki, hogy ő maga vajon milyen fizikai állapotban lehet most. (A teljesen megfigyelhető környezetben minden hiedelmi állapot egyetlen fizikai állapotból áll.)

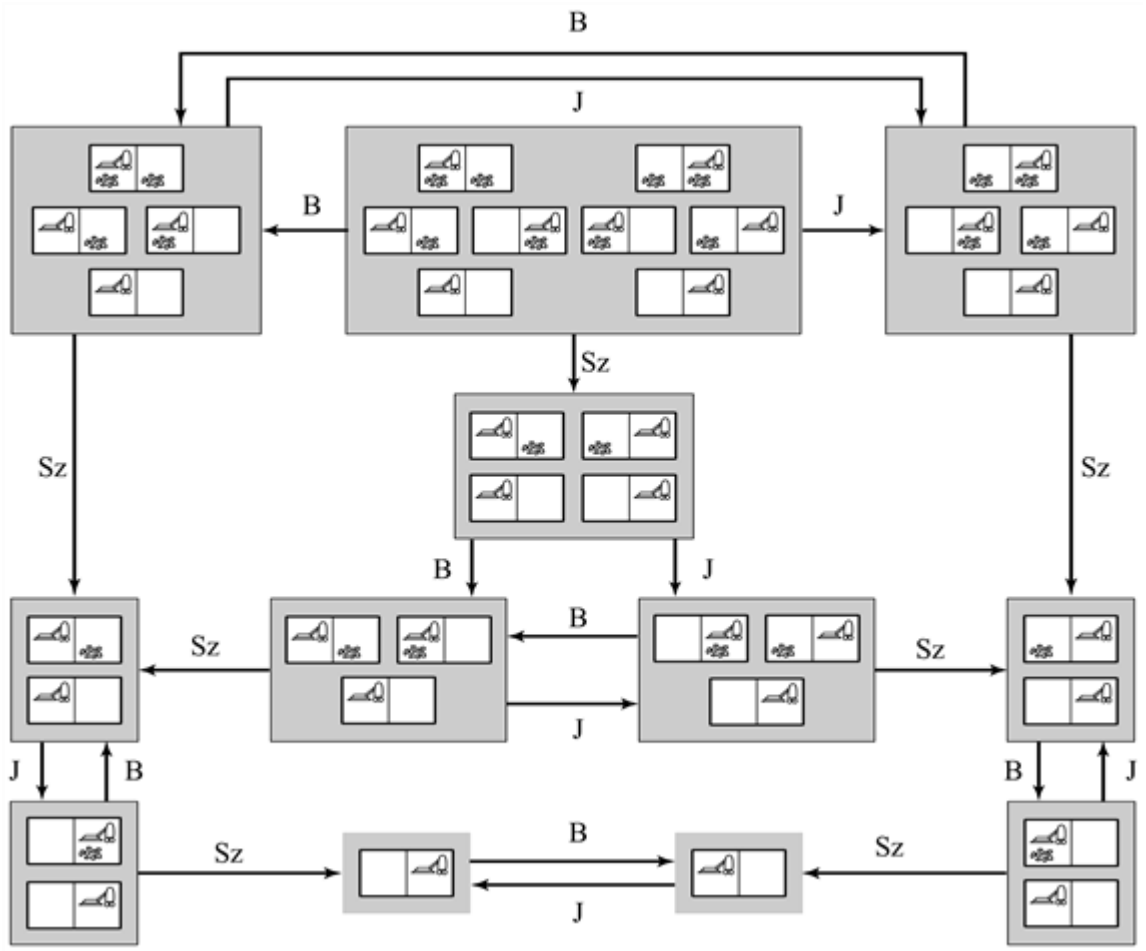
3.20. ábra - A porszívóvilág nyolc lehetséges állapota



Ahhoz, hogy egy szenzor nélküli problémát megoldhassunk, a fizikai állapotok terében történő keresés helyett a hiedelmi állapotok terében kell keresni. A kezdeti állapot egy hiedelmi állapot, és minden cselekvés egy hiedelmi állapotból egy hiedelmi állapotba képez le. Egy cselekvést úgy kell alkalmazni egy hiedelmi állapotra, hogy a hozzá tartozó minden fizikai állapotra kell azt alkalmazni, és az eredmények unióját kell képezni. Egy út most több hiedelmi állapotot köt össze, a megoldás pedig egy olyan utat jelent, amely *csak* célállapotokból álló hiedelmi állapotba vezet. A 3.21. ábra mutatja a determinisztikus szenzor nélküli porszívóvilág hozzáférhető hiedelmi állapotterét. Csak 12 hozzáférhető hiedelmi állapot létezik, az egész hiedelmi állapottér azonban tartalmazza a fizikai állapotok minden lehetséges halmazát, azaz $2^8 = 256$ hiedelmi állapotot. Általában, ha a fizikai állapottérnek S állapota van, a hiedelmi állapottér állapotainak száma 2^S.

3.21. ábra - A determinisztikus, szenzor nélküli porszívóvilág hiedelmi állapotterének hozzáférhető része. Minden árnyalt doboz egyetlenegy hiedelmi állapothoz tartozik. Minden pillanatban az ágens egy konkrét hiedelmi állapotban tartózkodik, azonban nem tudja, hogy melyik fizikai állapotban van. A kezdeti hiedelmi állapot (a teljes tudatlanság) legfelül középen helyezkedik el. A cselekvéseket a címkézett nyilak jelölik. Az önhurkokat a nagyobb áttekinthetőség érdekében az ábrából kihagytuk.

Created by XMLmind XSL-FO Converter.



A szenzor nélküli problémák eddigi elemzésében determinisztikus cselekvéseket tételeztünk fel. Az elemzés lényegében érvényes marad, ha a környezet nemdeterminisztikus, azaz ha a cselekvéseknek néhány lehetséges kimenetele lehet. Ennek az az oka, hogy szenzorok nélkül az ágens nem képes megmondani, hogy milyen kimenetel történt valójában. A különböző lehetséges kimenetelek így pótlólagos fizikai állapotok a követő hiedelmi állapotban. Tételezzük fel például, hogy a környezetben Murphy-törvény uralkodik: a *Szív* cselekvés *néha* koszt is hagy a szőnyegen, *de csak akkor, ha ott előzetesen a kosznak nyoma sem volt*.[6] Ha a *Szív* cselekvést tehát a 4. fizikai állapotban alkalmazzuk (lásd 3.20. ábra), a két lehetséges kimenetel a 2. és a 4. állapot. A kezdeti {1, 2, 3, 4, 5, 6, 7, 8} hiedelemállapotra alkalmazva a *Szív* olyan hiedelemállapothoz vezet, ami a cselekvés eredményeinek uniója a nyolc fizikai állapotra. Kiszámítva, azt kapjuk, hogy az új hiedelmi állapot az {1, 2, 3, 4, 5, 6, 7, 8}. A *Szív* cselekvés a Murphy-törvény uralta világban egy szenzor nélküli ágens hiedelmi állapotát tehát nem változtatja meg! A probléma valójában megoldhatatlan (lásd 3.18. feladat). Ennek intuitív magyarázata az, hogy az ágens nem képes megmondani, hogy az aktuális mező koszos-e, ebből kifolyólag nem képes megmondani, hogy a *Szív* cselekvés takarítani fog-e, vagy még több koszt hagy maga után.

6.2. Eshetőségi problémák

Ha a környezet jellege olyan, hogy az ágens a cselekvéseit követően szerezhet új információt a szenzorai révén, **eshetőségi problémá**ról (**contingency problem**) beszélünk. Egy eshetőségi probléma megoldása sokszor egy *fa* alakját veszi fel, ahol egy-egy ágnak a kiválasztása azon múlik, hogy odáig az ágens milyen érzékelésekhez jutott. Tegyük fel, hogy az ágens a Murphy-törvény világában rendelkezik helyzetérzékelővel és lokális koszérzékelővel, nincs azonban olyan szenzora, amely képes lenne koszt detektálni más négyzeteken. Így a [*B, Kosz*] érzékelés azt jelenti, hogy az ágens az {1, 3} állapotok egyikében van. Az ágens most a [*Szív*, *Jobbra*, *Szív*] cselekvési szekvenciát fogalmazhatta meg. A szívó cselekvés az {5, 7} állapotok egyikébe viszi az ágenst, és jobbra elmozdulva az állapota a {6, 8} állapotok egyikére változik. A szívó cselekvés alkalmazása a 6.

[6] Feltételezzük, hogy már minden olvasó került hasonló helyzetbe, így át tudják érezni ágensünk frusztráltságát. Elnézést kérünk a modern háztartási gépek tulajdonosaitól, akiknek ez a pedagógiai példa nem jelent segítséget.

Created by XMLmind XSL-FO Converter.

állapotban, a 8. állapotot – a célt – eredményezi. Ha azonban a szívó cselekvést a 8. állapotban alkalmazzuk, a 6. állapotba térünk vissza (a Murphy-törvény jóvoltából), amely esetben tervünk kudarcba fullad.

A hiedelmi teret a probléma e változatánál elemezve könnyen megállapíthatjuk, hogy semmilyen rögzített cselekvésszekvencia sem garantálja a probléma megoldását. De lesz megoldásunk, ha a cselekvések *szigorú sorrendjéről* lemondunk:

```
[Szív, Jobbra, ha [Jobbra, Kosz] akkor Szív]
```

A megoldások terét így azzal a lehetőséggel bővítjük, hogy a cselekvéseket a végrehajtás alatt megjelenő eshetőségek alapján választjuk. A valós fizikai világ számos problémája eshetőségi probléma, hiszen pontos előrejelzés lehetetlen. Sok ember ezért tartja mindig nyitva a szemét séta és vezetés közben.

Az eshetőségi problémák *néha* rögzített cselekvésszekvenciával is megoldhatók. Tekintsük például a *teljesen megfigyelhető* Murphy-törvény világot. Eshetőséggel állunk szemben, ha az ágens a *Szív* cselekvést egy üres négyzet felett hajtja végre, hiszen a kosz akkor vagy meg fog jelenni, vagy sem. Ameddig az ágens soha nem teszi ezt, eshetőségek fel sem merülnek, és minden kezdeti állapotból létezik egy rögzített megoldási szekvencia (lásd 3.18. feladat).

Eshetőségi problémák esetén az algoritmusok az e fejezetben található standard keresési algoritmusoknál bonyolultabbak. Ezekkel a 12. fejezet foglalkozik. Az eshetőségi problémák egy kissé eltérő ágens tervezési sémát is támogatnak, amelyekben az ágens egy garantált terv megtalálása *előtt* is cselekedhet. Ez hasznos, mert ahelyett, hogy a végrehajtás során *előforduló* minden eshetőséget előre figyelembe venne, gyakran jobb elkezdeni a végrehajtást, és megnézni, hogy mely eshetőségek *következnek be* valójában. A járulékos információ megadása után az ágens folytathatja a probléma megoldását. A keresés és a végrehajtás ily módon történő **összefésülése** (**interleaving**) a felderítéses problémák (lásd 4.5. alfejezet) vagy a játékok (6. fejezet) esetén is hasznos.

7. Összefoglalás

Ez a fejezet olyan módszereket ismertetett, amelyekkel az ágens a cselekvéseit választhatja meg determinisztikus, megfigyelhető, statikus és teljesen ismert környezetben. Ilyen esetekben az ágens olyan cselekvéssorozatokat konstruálhat, amelyekkel a céljait elérheti. Ezt a folyamatot **keresés**nek (**search**) nevezzük.

- A megoldás keresésének megkezdése előtt az ágensnek előbb meg kell fogalmaznia a **cél**t (**goal**), majd a célt fel kell használnia a **probléma** megfogalmazására.
- Egy **probléma** (**problem**) négy részből: a **kiinduló állapot**ból (**initial state**), egy **cselekvés-** (**action**) készletből, egy **célteszt-** (**goal test**) függvényből és egy **útköltség-** (**path cost**) függvényből áll. A probléma környezetét az **állapottér** (**state space**) írja le. Az állapottérben a kiinduló állapotból a célállapotba vezető **út** (**path**) a probléma egy **megoldás**a (**solution**).
- Egyetlen általános `FA-KERESÉS` algoritmus felhasználható bármilyen probléma megoldására. Az algoritmus konkrét változatai eltérő stratégiákat valósítanak meg.
- A keresési algoritmusokat azok **teljessége** (**completeness**), **optimalitás**a (**optimality**), **időigény**e (**time complexity**) és **tárigény**e (**space complexity**) alapján értékeljük. A komplexitást a probléma b elágazási tényezője és a legsekélyebben fekvő megoldás d mélysége határozza meg.
- A **szélességi keresés** (**breadth-first search**) először a legsekélyebben fekvő csomópontot fejti ki a keresési fában. Ez a keresés teljes, azonos költségű operátorok esetén optimális és $O(b^d)$ idő- és tárigénnyel rendelkezik. Tárigénye miatt a legtöbb esetben nem célszerű alkalmazni. Az **egyenletes költségű keresés** (**uniform-cost search**) hasonló a szélességi kereséshez, de először a legkisebb $g(n)$ útköltségű csomópontot fejti ki. Teljes és optimális, ha minden lépés költsége valamilyen pozitív ε korlátnál nagyobb.
- A **mélységi keresés** (**depth-first search**) először a keresési fa legmélyebben fekvő csomópontját fejti ki. Se nem teljes, se nem optimális, és időigénye $O(b^m)$, tárigénye pedig $O(bm)$, ahol m az állapottérbeli utak maximális mélysége.

Created by XMLmind XSL-FO Converter.

- A **mélységkorlátozott keresés** (**depth-limited search**) egy korlátot ad a mélységi keresés keresési mélységére.
- Az **iteratívan mélyülő keresés** (**iterative deepening search**) a cél megtalálásáig egyre növekvő mélységkorláttal hívja meg a mélységkorlátozott keresést. Teljes és egységnyi lépésköltség esetén optimális, időigénye $O(b^d)$, tárigénye pedig $O(bd)$.
- A **kétirányú keresés** (**bidirectional search**) radikálisan csökkentheti az időigényt, de nem mindig alkalmazható, és a tárigénye túl nagy lehet.
- Ha az állapotér inkább egy gráf, mint egy fa, kifizetődő lehet az ismétlődő állapotokat megvizsgálni. A GRÁF-KERESÉS algoritmus az összes duplikált állapotot kiküszöböli.
- Ha a környezet csak részben megfigyelhető, az ágens a keresési algoritmust a **hiedelmi állapot**ok (**belief state**s) terében alkalmazhatja, vagyis olyan lehetséges állapotok terében, amelyekben az ágens tartózkodhat. Egyes esetekben egyetlen megoldási szekvencia konstruálható, más esetekben az ágensnek **eshetőségi terv**re (**contingency plan**) van szüksége, hogy a felmerülő ismeretlen helyzetekkel megbirkózzon.

7.1. Irodalmi és történeti megjegyzések

Az ebben a fejezetben elemzett állapottér-keresési problémák többségének hosszú története van az irodalomban, és e problémák sokkal kevésbé triviálisak, mint azt első ránézésre gondolnánk. A 3.9. feladatban említett hittérítők és kannibálok problémáját Amarel elemezte részletesen (Amarel, 1968). A mesterséges intelligencia területén Simon és Newell (Simon és Newell, 1961), míg a számítástudomány és az operációkutatás területén Bellman és Dreyfus már korábban foglalkoztak e problémával (Bellman és Dreyfus, 1962). Az ilyen tanulmányok és Simon és Newell Logic Theoristra vonatkozó munkája (Simon és Newell, 1957), valamint a GPS (Simon és Newell, 1962) hatására váltak a keresési algoritmusok az 1960-as években az MI-kutatók fegyvertárának elsődleges eszközeivé, illetve ezek hatására vált a problémamegoldás kanonikus MI-feladattá. Sajnos eddig nagyon kevés történt a problémamegfogalmazás lépéseinek automatizálása irányában. A problémareprezentáció és -absztrakció frissebb tárgyalását Knoblock adta meg (Knoblock, 1990), beleértve olyan MI-programok leírását is, amelyek (részben) maguk végzik el ezeket a feladatokat.

A 8-as kirakójáték a bonyolultabb 15-ös kirakójáték kisebb rokona, amit a híres amerikai játéktervező, Sam Loyd talált ki az 1870-es években (Loyd, 1959). A játék akkoriban gyorsan óriási, Rubik bűvös kockájához mérhető népszerűségre tett szert Amerikában. A matematikusok is felfigyeltek a problémára (Johnson és Story, 1879; Tait, 1880). Az *American Journal of Mathematics* szerkesztői megállapították: „A 15-ös kirakójáték az elmúlt hetekben teljesen lebilincselte az amerikai embereket – 10 emberből 9 nemre, korra és származásra való tekintet nélkül a játék bűvöletébe került. Ez azonban még nem hatott volna a szerkesztőkre olyan mértékben, hogy rávegye őket, erről a témáról cikkeket jelentessenek meg az *American Journal of Mathematics* folyóiratban, de figyelembe véve azt a tényt, hogy…” (itt a 15-ös kirakójáték által felvetett érdekes matematikai kérdések rövid taglalása következett). A 8-as játék esetén P. D. A. Schofield számítógép segítségével elvégezte a probléma kimerítő elemzését (Schofield, 1967). Ratner és Warmuth megmutatta, hogy a legrövidebb megoldás megkeresése a 15-ös játék $n \times n$-es általánosításánál az NP-teljes problémák osztályába tartozik (Ratner és Warmuth, 1986).

A 8-királynő problémát először 1848-ban a *Schach* német sakkszaklapban névtelenül publikálták, később a problémát Max Bezzel nevéhez kapcsolták. A problémát 1850-ben újra publikálták, ami akkor felkeltette a kiváló matematikus, Karl Friedrich Gauss figyelmét, aki megkísérelte felsorolni az összes megoldást, azonban csak 72 megoldást talált meg. Az összes, 92 megoldást Nauck publikálta 1850-ben. A problémát Netto n királynő esetére általánosította (Netto, 1901), Abramson és Young pedig találtak egy $O(n)$ algoritmust (Abramson és Young, 1989).

A fejezetben említett valósvilág-beli keresési problémák közül mindegyik komoly kutatási erőfeszítések tárgya volt. Az optimális repülőjárat megválasztásának módszereit általában bizalmasan kezelik, azonban Carl de Marcken kimutatta (magánközlés), hogy a repülőtársaságok által alkalmazott jegytarifák és egyéb korlátozások annyira összekuszálódtak, hogy az optimális járat megválasztása formálisan *eldönthetetlen*. Az elméleti számítástudományban standard kombinatorikus probléma az utazó ügynök problémája (TSP) (Lawler, 1985; Lawler és társai, 1992). A problémáról Karp bizonyította be, hogy NP-nehéz (Karp, 1972), azonban megoldására hatékony heurisztikus közelítő módszereket dolgoztak ki (Lin és Kernighan, 1973). Az euklideszi térben zajló TSP-re Arora dolgozta ki a teljesen polinomiális közelítő sémát (Arora, 1998). A VLSI-elrendezés

Created by XMLmind XSL-FO Converter.

módszereit Shahookar és Mazumder (Shahookar és Mazumder, 1991) tekintették át, számos optimalizációs cikk VLSI-újságokban jelent meg. Robotnavigálási és szerelési problémákkal a 25. fejezet foglalkozik.

A problémamegoldás nem informált keresési algoritmusai központi szerepet töltenek be a klasszikus számításelméletben (Horowitz és Sahni, 1978), az operációkutatásban (Dreyfus, 1969). Deo és Pang, továbbá Gallo és Pallottino egy frissebb áttekintést adnak erről a területről (Deo és Pang, 1984; Gallo és Pallottino, 1988). A szélességi keresést labirintusok megoldására Moore (Moore, 1959) fogalmazta meg. A **dinamikus programozás** (**dynamic programming**) módszere (Bellman és Dreyfus, 1962), amely szisztematikusan jegyzi meg az összes növekvő nagyságú részprobléma megoldását, egy gráfokon futó szélességi keresésnek tekinthető. A Dijkstra-féle kétpontos legrövidebb út algoritmus (Dijkstra, 1959) az egyenletes költségű keresés kiindulópontja.

Az iteratívan mélyülő algoritmus egy változatát Slate és Atkin használta először a CHESS 4.5 sakkprogramjában (Slate és Atkin, 1977), hogy a sakkórát hatékonyabban lehessen használni. De az ötletet gráfokban a legrövidebb út keresésére először Korf használta (Korf, 1985a). A Pohl bevezette kétirányú keresés (Pohl, 1969; 1971) bizonyos esetekben szintén igen hatékony lehet.

A részben megfigyelhető és a nemdeterminisztikus környezeteket a problémamegoldási megközelítésen belül nem tanulmányozták behatóan. A hiedelmi állapotok terében történő keresés bizonyos hatékonysági problémáival Genesereth és Nourbakhsh foglalkoztak (Genesereth és Nourbakhsh, 1993). Koenig és Simmons (Koenig és Simmons, 1998) az ismeretlen kezdeti pozícióból induló robotnavigálást tanulmányozták. Erdmann és Mason (Erdmann és Mason, 1988) pedig a szenzorok nélküli robot manipuláció problémáját vizsgálták a hiedelem állapottérbeli keresés folytonos változatát használva. Eshetőségi kereséssel a tervkészítés területen belül foglalkoztak (lásd 12. fejezet). A bizonytalan információt felhasználó tervkészítést és cselekvést a valószínűség-elmélet és döntéselmélet eszköztárával oldották meg (lásd 17. fejezet).

Nilsson könyvei a klasszikus keresési algoritmusokkal kapcsolatban hasznos információkat tartalmaznak (Nilsson, 1971; 1980). Korf egy átfogó, naprakészebb áttekintést ad a klasszikus keresési algoritmusokról (Korf, 1988). Az új keresési algoritmusokról – amelyek feltárása figyelemre méltó módon folytatódik – olyan folyóiratokban jelennek meg cikkek, mint például az *Artificial Intelligence*.

7.2. Feladatok

3.1.

Definiálja saját szavaival a következő fogalmakat: állapot, állapottér, keresési fa, keresési csomópont, cél, cselekvés, állapotátmenet-függvény és elágazási tényező.

3.2.

Magyarázza meg, hogy a problémamegfogalmazásnak miért kell a célmegfogalmazást követnie.

3.3.

Tegyük fel, hogy a LEGÁLIS-CSELEKVÉS(s) azon cselekvések halmazát jelöli, amelyek legálisak az s állapotban, és az EREDMÉNY(a, s) pedig azt az állapotot, ami az s állapotban a végrehajtott legális cselekvés eredménye. Definiálja az ÁLLAPOTÁTMENET-FV-t a LEGÁLIS-CSELEKVÉS és az EREDMÉNY függvényében, és *megfordítva*.

3.4.

Mutassa meg, hogy a 8-as játék állapottere két diszjunkt részből áll, ahol egyik részhez tartozó állapotot semmilyen véges lépésszekvenciával sem lehet a másik részhez tartozó állapotba áttranszformálni. (*Segítség:* Berlekamp és társai, 1982). Dolgozzon ki egy eljárást, amely el tudja dönteni, hogy az állapot melyik részhez tartozik, és magyarázza meg, hogy ez miért hasznos, ha az állapotokat véletlen módon generáljuk.

3.5.

Gondolja át az n-királynő problémát a 2.1. szakasz - Játékproblémák részben ismertetett „hatékony” inkrementális megfogalmazást felhasználva. Magyarázza meg, hogy az állapottér mérete miért lesz legalább $\sqrt[3]{n!}$ és becsülje meg azt a legnagyobb n-et, amire a kimerítő feltárás még elfogadható. (*Segítség*: számolja ki az

Created by XMLmind XSL-FO Converter.

elágazási tényező alsó korlátját figyelembe véve, hogy mekkora a királynő által támadható sakktáblahelyek maximális száma egy tetszőleges oszlopban.)

3.6.

Egy véges állapottér mindig véges keresési fához vezet-e? Mi a helyzet egy olyan véges állapottérrel, amely maga is egy fa? Meg tudja-e precízebben fogalmazni, hogy milyen állapottérfajták vezetnek mindig véges keresési fákhoz? Átvéve (Bender, 1996)-ből.

3.7.

Az alábbiak mindegyikéhez adja meg a kiinduló állapotot, a céltesztet, az állapotátmenet-függvényt és az útköltségfüggvényt. Válasszon meg egy olyan megfogalmazást, amely elegendően precíz, hogy azt implementálni lehessen.

a. Négy szín felhasználásával ki kell színeznie egy síkbeli térképet úgy, hogy minden szomszédos tartomány különböző színű legyen.

b. Egy 1 m magas majom egy szobában tartózkodik, ahol van két 1 m magas láda (amelyeket egymásra lehet rakni, lehet mozgatni, és meg lehet mászni), és néhány banán lóg a 2,5 m magas mennyezetről. A majom el szeretné érni a banánt.

c. Van egy olyan programja, amely „illegális input rekord” üzenetet ad, ha az input rekordok egy bizonyos állományát dolgozza fel. Ön tudja, hogy egy-egy rekord feldolgozása a többitől független. Szeretné az illegális rekordot megtalálni.

d. Van három kancsója, egy 12 l-es, egy 8 l-es és egy 3 l-es és egy vízcsapja. A kancsókat megtöltheti, kiöntheti belőlük a vizet a földre, vagy az egyikből áttöltheti a vizet egy másikba. Pontosan 1 l vizet kell kimérnie.

3.8.

Tekintsen egy olyan állapotteret, ahol a kezdő állapot az 1-es számú és az állapotátmenet-függvény az n sorszámú állapotra két állapottal, a $2n$ és a $2n + 1$ sorszámú állapotokkal tér vissza.

a. Rajzolja fel az állapottérnek az 1-es állapottól a 15-ös állapotig terjedő részét.

b. Tételezze fel, hogy a célállapot a 11-es. Listázza ki a csomópontok kifejtési sorrendjét szélességi, 3-as mélységkorláttal rendelkező mélységkorlátos és iteratívan mélyülő keresés esetén.

c. Jó lenne-e a probléma megoldására a kétirányú keresés? Ha igen, magyarázza meg részletesen, hogyan fog ez működni.

d. Mennyi a kétirányú keresés elágazási tényezője mindkét irányban?

e. Vajon a (c) pontra adott válasz azt sugallja-e, hogy a problémának lehetséges olyan átfogalmazása, hogy az 1-es állapotból egy adott célállapothoz majdnem keresés nélkül el tudnánk jutni?

3.9.

A hittérítők és kannibálok problémáját általában az alábbi módon szokás megadni. Három hittérítő és három kannibál egy folyó azonos oldalán tartózkodik. Rendelkeznek csónakkal, amely egy vagy két embert bír el. Találjuk meg annak a módját, hogy mindenkit a folyó másik oldalára átjuttassunk, arra ügyelve, hogy a hittérítők soha, sehol ne legyenek kisebbségben a kannibálokkal szemben. Ez a probléma híres az MI-ben, mert ez volt témája az első olyan publikációnak, ahol a problémamegfogalmazást analitikus szemszögből kísérelték meg (Amarel, 1968).

a. Fogalmazza meg precízen a problémát. Csak azokat a megkülönböztetéseket tegye meg, amelyek a helyes megoldás érdekében szükségesek. Rajzolja fel a teljes állapottér diagramját.

b. Implementálja és oldja meg optimális módon a problémát, megfelelő keresési algoritmust választva. Jó ötlet-e az ismétlődő állapotok ellenőrzése?

Created by XMLmind XSL-FO Converter.

c. Véleménye szerint miért nehéz e feladvány az emberek számára, annak ellenére, hogy az állapottér ilyen egyszerű?

3.10.

Implementálja az állapotátmenet-függvény két változatát a 8-as játék számára. Az egyik az összes követőt egyszerre generálja a 8-as játék adatstruktúrájának másolásával és editálásával. A másik hívásakor, csak egy követőt generál, a szülő állapot közvetlen módosításával (és szükség esetén a módosítások visszaállításával). Írja meg az iteratívan mélyülő keresés olyan változatait, amelyek a függvény fenti változatait használják, és hasonlítsa össze a megoldások hatékonyságát.

3.11.

A 4.5. szakasz - Iteratívan mélyülő mélységi keresés részben az **iteratívan megnyúló keresést** (**iterative lengthening search**) említettük, amely az egyenletes költségű keresés iteratív változata. Az ötlet az útköltségre vonatkozó növekvő korlát használata. Egy újonnan generált csomópontot eldobunk, ha az útköltsége a korlátnál nagyobb. A következő iterációban a korlátot az előző iterációban eldobott csomópontok közül a legkisebb útköltségre állítjuk be.

a. Mutassa meg, hogy ez az algoritmus általános útköltségek esetén optimális.

b. Tekintsen egy homogén fát b elágazási tényezővel, d megoldásmélységgel és egységnyi lépésköltséggel. Hány iterációra lesz szüksége az iteratívan megnyúló keresésnek?

c. Most tekintsen egy olyan lépésköltséget, amit a [0, 1] folytonos tartományból sorsolnak, egy minimális pozitív ε költséggel. Hány iterációra lesz szükség a legrosszabb esetben?

d. Implementálja az algoritmust és alkalmazza a 8-as játék és az utazó ügynök probléma konkrét eseteire. Hasonlítsa össze az algoritmus hatékonyságát az egyenletes költségű algoritmus hatékonyságával, és értelmezze az eredményeit.

3.12.

Bizonyítsa be, hogy az egyenletes költségű keresés és a szélességi keresés konstans lépésköltség mellett optimálisak, ha azokat a `GRÁF-KERESÉS` algoritmusával együtt alkalmazzuk. Mutasson egy olyan állapotteret változó lépésköltséggel, ahol a `GRÁF-KERESÉS` algoritmus, iteratív mélyülést használva, csak egy szuboptimális megoldást talál.

3.13.

Adjon meg egy olyan keresési teret, amelyben az iteratívan mélyülő keresési algoritmus a mélységi keresésnél lényegesen rosszabb teljesítményt nyújt (például $O(n^2)$ az $O(n)$-nel szemben).

3.14.

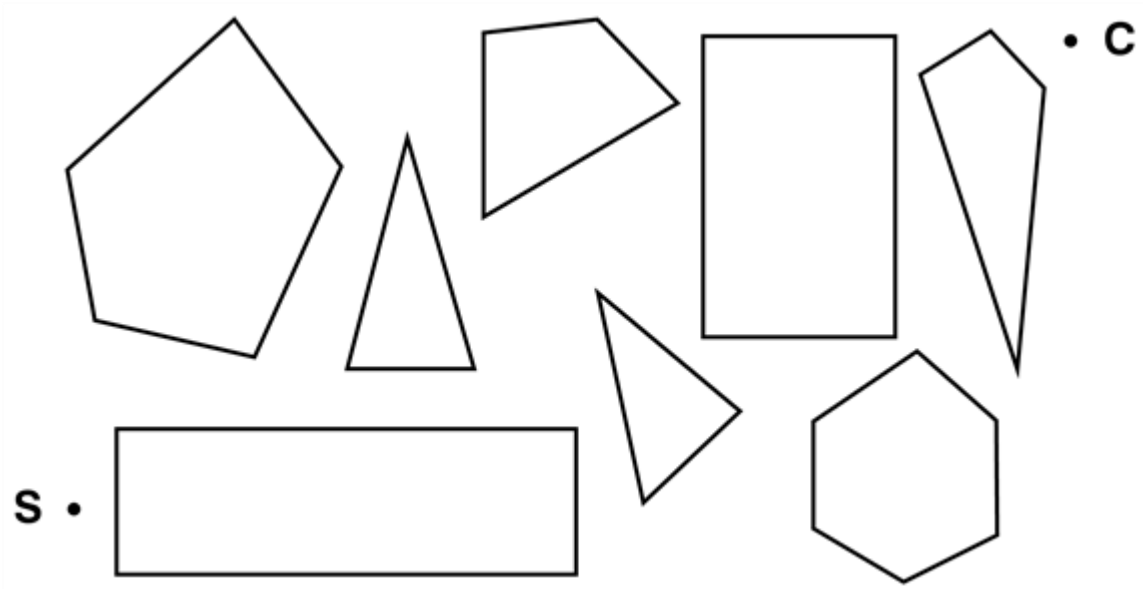
Írjon egy olyan programot, amely bementként két weblap URL-jét kapja meg, és megoldásul megtalálja az azokat összekapcsoló hivatkozási utat (linkeket). Mi a megfelelő keresési stratégia? Jó ötlet-e a kétirányú keresés? Alkalmazható-e egy keresőgép az elődcsomópont függvény megvalósítására?

3.15.

Gondolja végig egy sík két pontja közötti legrövidebb út megtalálásának a problémáját, ha a síkon konvex sokszög akadályok találhatók (lásd 3.22. ábra). Ez annak a problémának egy idealizált változata, amivel egy robotnak kell szembesülnie, amikor egy zsúfolt környezetben kell navigálnia.

3.22. ábra - Egy elrendezés sokszögakadályokkal

Created by XMLmind XSL-FO Converter.



a. Tegyük fel, hogy az állapottér a sík összes (x, y) pozíciójából áll. Hány állapotról beszélhetünk ekkor? Hány út vezet a célig?

b. Magyarázza el röviden, hogy egy sokszög csúcsát egy másikkal a síkban összekötő legrövidebb útnak miért kell olyan egyenes vonalszakaszokból állnia, amelyek az egyes sokszögek csúcsait kapcsolják össze? Adjon most egy jó állapottér definíciót. Milyen nagy ez a tér?

c. Definiálja a keresési probléma implementálásához szükséges függvényeket, beleértve az állapotátmenet-függvényt, amely bemenetként egy csúcsot kap, és a belőle egyenes vonalban elérhető csúcsok halmazát adja vissza (ne felejtsük ki a szomszédos csúcsokat ugyanazon a sokszögön sem). Heurisztikus függvénynek az egyenes vonalbeli távolságot használja.

d. Alkalmazzon a fejezetben ismertetett algoritmusok közül egyet vagy többet a tárgyterülethez tartozó problémák megoldására, és értékelje azok teljesítményét.

3.16.

A 3.15. feladat navigációs problémáját a következőképpen transzformálja át egy környezetbe:

- Az érzékelés az ágens által látható sokszögcsúcsok az ágenshez viszonyított helyzetének listája. Az érzékelés a robot pozícióját nem tartalmazza! A robotnak a saját pozícióját a térkép alapján kell megtanulnia. Azt is tételezzük fel ideiglenesen, hogy minden lokációnak eltérő a „panorámája”.

- Minden cselekvés egy a követendő egyenes utat leíró vektor lesz. Ha az úton nincsenek akadályok, a cselekvés sikeres. Egyébként a robot azon a ponton megáll, ahol az út az első akadályt metszi. Ha az ágens a zérus mozgás vektorát adja vissza és a célpozícióban van (ami rögzített és ismert), akkor a környezetnek teleportálnia kell az ágenst egy véletlen lokációba (de nem egy akadályon belülre).

- A hatékonysági mérték 1 pontot kér az ágenstől a megtett távolság minden egységtávjáért és 1000 ponttal díjazza a cél mindenkori elérését.

a. Implementálja ezt a környezetet és egy erre a környezetre vonatkozó problémamegoldó ágenst. Minden teleportálás után az ágensnek új célt kell megfogalmaznia, beleértve a saját pozíció felfedezését.

b. Dokumentálja az ágens teljesítményét (kommentálja az ágens mozgását), és írjon egy jelentést az ágens 100 epizódra vonatkozó teljesítményéről.

c. Módosítsa a környezetet úgy, hogy az ágens az esetek 30%-ban ne a szándékolt helyen találja magát (a helyet a látható sokszögcsúcsokból sorsoljuk, ha van ilyen egyáltalán, különben nincs mozgás). Ez a valós robot hibás mozgásának egy durva modellje. Módosítsa ágensét úgy, hogyha az ilyen hibát detektál, derítse ki, hol van, és készítsen tervet, hogy az eredeti pozícióba visszakerülhessen, ahonnan az eredeti tervét folytatja.

Created by XMLmind XSL-FO Converter.

Emlékezzen arra, hogy néha az eredeti pozícióba való visszatérés is kudarccal végződhet! Mutasson egy olyan ágenst, amely két egymás utáni hibás mozgással is sikeresen megbirkózik, és eléri a célt.

d. Most két különböző visszaállítási sémával kísérletezzen egy hiba megtörténte után: (1) induljon az eredeti út legközelebb eső csúcsa felé, és (2) tervezze át az utat a cél felé az új pozícióból kiindulva. Hasonlítsa össze a három visszaállítási séma teljesítményét. Meg fogja-e változtatni az összehasonlítás eredményét a keresési költségek figyelembevétele?

e. Most tételezze fel, hogy vannak lokációk azonos „panorámával” (tételezzük fel például, hogy a világ egy négyzetrács négyzetes akadályokkal). Milyen problémákkal kell most az ágensnek szembenéznie? Milyenek most a megoldások?

3.17.

A 1.1. szakasz - Jól definiált problémák és megoldások részben azt mondtuk, hogy nem tárgyaljuk a negatív útköltséggel rendelkező problémákat. Ennek a feladatnak a kapcsán részletesebben megvizsgáljuk a kérdést.

a. Tegyük fel, hogy a cselekvéseknek tetszőlegesen nagy negatív költségük lehet. Magyarázza meg, hogy ez a lehetőség miért kényszerít minden optimális algoritmust a teljes állapottér felkutatására.

b. Segíteni fog-e, ha ragaszkodunk ahhoz, hogy a lépésköltség egy negatív c konstansnál nagyobb vagy azzal egyenlő legyen? Mind a fákra, mind a gráfokra gondoljon.

c. Tételezzük fel, hogy cselekvések egy halmaza ciklust alkot úgy, hogy a halmaz cselekvéseit valamilyen sorrendben végrehajtva az állapot nem változik. Milyen következményekkel jár egy ilyen környezetben tevékenykedő ágens optimális viselkedésére, ha ezen cselekvések mindegyike negatív költségű?

d. Könnyen el tudunk képzelni nagy negatív költségű operátorokat még az útkeresési probléma esetén is. Például néhány útszakasz olyan gyönyörű tájakon halad, hogy az idő és üzemanyag tekintetében messze túlszárnyalja a normál költségeket. Pontos megfogalmazásban magyarázza el, hogy az emberek miért nem kocsikáznak a végtelenségig a szép tájak körül, és hogyan lehetne az útkeresési problémához az állapotteret és az operátorokat definiálni, hogy a mesterséges ágens is el tudja kerülni a ciklusokat.

e. Tud olyan valódi problémakört mondani, amelyben a lépések költsége ciklust okoz?

3.18.

Gondoljon a Murphy-törvény uralma alatt álló kétlokációjú, szenzor nélküli porszívóvilágra. Rajzolja fel az {1, 2, 3, 4, 5, 6, 7, 8} kezdeti hiedelmi állapotból elérhető hiedelem-állapotteret, és magyarázza meg, hogy a probléma miért megoldhatatlan. Mutassa meg azt is, hogy ha a világ teljesen megfigyelhető, minden kezdeti állapotból létezik egy megoldási szekvencia.

3.19.

Tekintse a 2.2. ábrán látható porszívóvilágot.

a. A fejezetben ismertetett algoritmusok közül amelyik lenne alkalmas e probléma megoldására? Szükséges-e, hogy az algoritmus ellenőrizze az ismételt állapotokat?

b. A 3 × 3 világban alkalmazza a megválasztott algoritmust az optimális cselekvési szekvencia kiszámítására egy olyan kezdeti állapottal, ahol a felső három négyzet mindegyikében található piszok, és az ágens a középső négyzetben tartózkodik.

c. Építsen kereső ágenst, és értékelje a teljesítményét egy olyan 3 × 3 porszívóvilág halmaz számára, ahol minden négyzetben 0,2 a kosz valószínűsége. A teljesítmény mérésénél mind az útköltséget, mind a keresési költséget vegye figyelembe, egy értelmes átszámítási tényezőt használva.

d. Hasonlítsa össze az eddigi legjobb kereső ágens viselkedését egy olyan egyszerű, randomizált reflexszerű ágensével, amely koszt szív, ha azt megtalálja, különben véletlen módon mozog.

e. Értékelje, hogy mi történne, ha a világot n × n-re nagyítanánk. Hogyan változna a kereső és a reflexszerű ágens teljesítménye n növelésével?

Created by XMLmind XSL-FO Converter.

4. fejezet - Informált keresési és felfedező módszerek

Ebben a fejezetben látni fogjuk, hogy az állapottérrel kapcsolatos információ segítségével az algoritmusok hogyan kerülhetik el a sötétben való tapogatózást.

A 3. fejezetben láttuk, hogy a nem informált keresési stratégiák oly módon képesek problémák megoldásait megtalálni, hogy szisztematikusan új állapotokat generálnak és összehasonlítják azokat a célállapottal. Sajnos ezek a stratégiák a legtöbb esetben hihetetlenül rossz hatékonysággal dolgoznak. Ezen fejezet megmutatja, hogy egy – problémaspecifikus tudást alkalmazó – informált keresési stratégia hatékonyabban képes a megoldást megtalálni. A 4.1. alfejezet bemutatja a 3. fejezetben tanulmányozott algoritmusok informált változatait, a 4.2. alfejezet pedig elmagyarázza, hogy a szükséges problémaspecifikus információ hogyan szerezhető meg. A 4.3. és a 4.4. alfejezet olyan algoritmusokkal foglalkozik, amelyek az állapottérben tisztán **lokális keresést** (**local search**) hajtanak végre, egy vagy több aktuális állapotot értékelve és módosítva ahelyett, hogy szisztematikusan tárnák fel az utat a kezdeti állapottól indulva. Ezek az algoritmusok olyan problémák esetén jók, ahol az útköltség közömbös, és az egyetlen, ami számít, hogy megtaláljuk-e a megoldást. A lokális keresési algoritmusok családjába a statisztikai fizika inspirálta módszerek (**szimulált lehűtés**, **simulated annealing**) és az evolúciós biológia sugallta módszerek (**genetikus algoritmus**ok, **genetic algorithm**s) is beletartoznak. A 4.5. alfejezet végül az **online keresés**sel (**online search**) foglalkozik, ahol az ágens egy teljesen ismeretlen állapottérrel találja magát szembe.

1. Informált (heurisztikus) keresési stratégiák

Ez a fejezet megmutatja, hogy az **informált keresés**i (**informed search**) stratégia – amely a probléma definícióján túlmenően problémaspecifikus tudást is felhasznál – hogyan képes hatékonyabban megtalálni a megoldást.

Az általunk vizsgált általános megközelítést a **legjobbat-először keresés**nek (**best-first search**) nevezzük. A legjobbat-először keresés az általános `FA-KERESÉS` vagy `GRÁF-KERESÉS` algoritmusok olyan speciális esete, ahol egy csomópont kifejtésre való kiválasztása egy $f(n)$ **kiértékelő függvény**től (**evaluation function**) függ. Hagyományosan a *legkisebb* értékű csomópontot választjuk kifejtésre, mert a kiértékelő függvény a céltól való távolságot méri. A legjobbat-először keresés az eddigi általános keresési eljárások keretein belül egy prioritási sor segítségével implementálható, ami egy olyan adatstruktúra, mely a peremet a növekvő f-értékek szerint rendezi.

A legjobbat-először keresés egy nagy múltú, azonban pontatlan elnevezés. Amenynyiben *valóban* képesek lennénk a legjobb csomópontot kifejteni, akkor egyáltalán nem kellene keresnünk, nyílegyenesen elmasíroznánk a célhoz. Ezzel szemben csak a kiértékelő függvény szerint legjobbnak *tűnő* csomópontot tudjuk kiválasztani. Ha kiértékelő függvényünk mindentudó, akkor a kiválasztott csomópont egyben a legjobb csomópont is. A valóságban azonban a kiértékelő függvény néha pontatlan, és félrevezetheti a keresést. Azonban a továbbiakban is ragaszkodni fogunk a legjobbat-először keresés elnevezéshez, mert a legjobbnak tűnőt először keresés egy kicsit furcsán hangzana.

A `LEGJOBBAT-ELŐSZÖR-KERESÉS` algoritmus tulajdonképpen egy keresési algoritmus család, amelynek az elemeit az eltérő kiértékelő függvények[1] különböztetik meg. Ezeknek az algoritmusoknak a kulcseleme a $h(n)$-nel jelölt **heurisztikus függvény**[2] (**heuristic function**):

`h(n) = az n csomóponttól a célig vezető legolcsóbb út becsült útköltsége`

Például Romániában az Arad és Bukarest közötti legolcsóbb út költségét az Arad és Bukarest közötti légvonaltávolsággal meg lehetne becsülni.

A heurisztikus függvény a leginkább megszokott módja annak, hogy a problémára vonatkozó pótlólagos tudást a keresési algoritmusba be tudjuk injektálni. A heurisztikus függvényekkel részletesen a 4.2. alfejezetben foglalkozunk. Egyelőre tetszőleges, ám problémaspecifikus függvényeknek fogjuk őket tekinteni, egy

[1] A 4.3. feladatban azt kérjük Öntől, hogy mutassa ki, hogy ez a család néhány jól ismert nem informált keresést is tartalmaz.
[2] A $h(n)$ heurisztikus függvény bemenete ugyan egy *csomópont*, a függvény értéke azonban a csomóponthoz tartozó *állapottól* függ.

Created by XMLmind XSL-FO Converter.

kikötéssel: ha n egy célállapot, akkor $h(n) = 0$. A jelen fejezet hátralévő része két olyan utat mutat be, ahol a heurisztikus információt a keresés irányítására használjuk fel.

1.1. A mohó legjobbat-először keresés

A **mohó legjobbat-először keresés**[3] (**greedy best-first search**) azt a csomópontot fejti ki a következő lépésben, amelyiknek az állapotát a legközelebbinek ítéli a célállapothoz, abból kiindulva, hogy így gyorsan megtalálja a megoldást. A csomópontokat az algoritmus tehát az $f(n) = h(n)$ heurisztikus függvénnyel értékeli ki.

Nézzük meg, hogy hogyan működik ez a romániai útkeresésnél a **légvonalban mért távolság**ot (**straight-line distance**) felhasználva, amit h_{LMT}-vel fogunk jelölni. Ha a cél Bukarest, szükségünk lesz Bukarest légvonalbeli távolságaira, amelyeket a 4.1. ábra ad meg. Például $h_{LMT}(Benn(Arad)) = 366$. Vegyük észre, hogy a h_{LMT} értékeit magának a problémának a leírásából kiszámítani nem lehet. Továbbá egy kis tapasztalat is szükséges ahhoz, hogy rájöjjünk, hogy a h_{LMT} az úton megtett tényleges távolsággal korrelál, és így számunkra hasznos heurisztika lehet.

4.1. ábra - A h_{LMT} értékei – a légvonalbeli távolságok Bukarestig

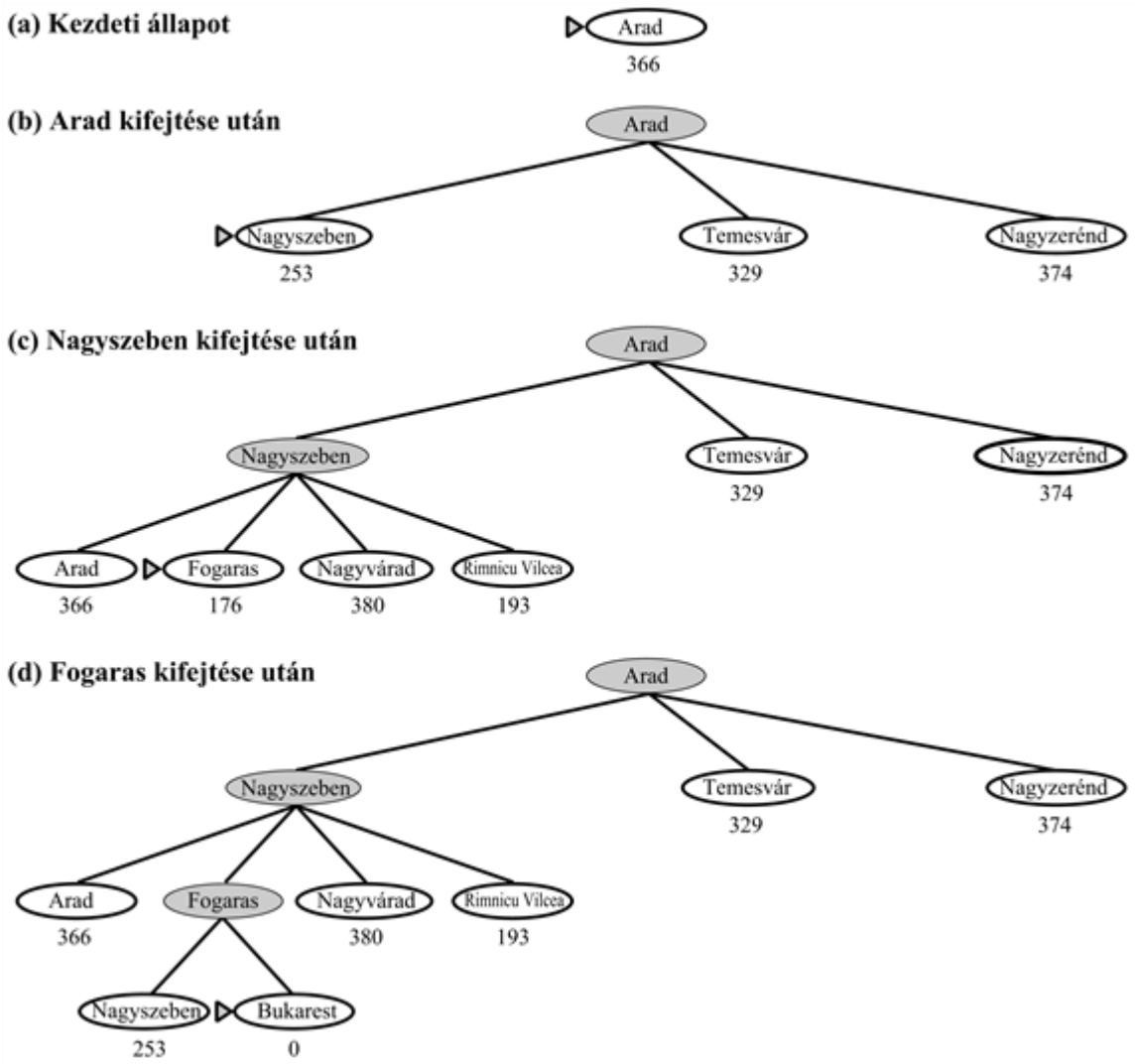
Arad	366	**Mehadia**	241
Bukarest	0	**Neamt**	234
Craiova	160	**Nagyvárad**	380
Dobreta	242	**Piteşti**	100
Eforie	161	**Rimniu Vilcea**	193
Fogaras	176	**Nagyszeben**	253
Giurgiu	77	**Temesvár**	329
Hirsova	151	**Csalános**	80
Iasi	226	**Vaslui**	199
Lugos	244	**Nagyzerénd**	374

A 4.2. ábra egy Aradról Bukarestbe vezető út mohó legjobbat-először keresését mutatja, h_{LMT}-t alkalmazva. Aradból az első kifejtett csomópont Nagyszeben, mert ez közelebb van Bukaresthez, mint Nagyzerénd vagy Temesvár. A következő kifejtendő csomópont pedig Fogaras, mert az van a legközelebb. Fogaras majd generálja Bukarestet, ami egyben a célállapot. Erre a konkrét problémára a h_{LMT} -t alkalmazó mohó legjobbat-először keresés úgy talál megoldást, hogy soha sem fejt ki olyan csomópontot, ami nem a megoldási úton fekszik. Következésképpen minimális a keresési költsége. Azonban nem optimális: a Nagyszebenen és Fogarason keresztül Bukarestbe vezető út 32 kilométerrel hosszabb a Rimnicu Vilceán és Piteştin keresztül vezető útnál. Ebből látszik, hogy az algoritmus miért „mohó” – minden lépésben igyekszik annyira közel kerülni a célhoz, ahogy csak lehet.

4.2. ábra - A mohó legjobbat-először keresés lépései Bukarest esetén a légvonalban mért távolságot (h_{LMT}) alkalmazva. A csomópontok a saját h-értékeikkel vannak felcímkézve.

[3] A könyv első kiadásában ennek **mohó keresés** (**greedy search**) volt a neve. Más szerzők ezt **legjobbat-először keresés**nek (**best-first search**) nevezik. A második változat általunk választott általánosabb használata Pearlt (Pearl, 1984) követi.

Created by XMLmind XSL-FO Converter.



A $h(n)$ minimalizálása érzékeny a hibás kezdő lépésekre. Tekintsük például azt az esetet, amikor Iaşiról Fogarasra akarunk eljutni. A heurisztika alapján Neamtot kellene először kifejteni, hiszen ez fekszik Fogarashoz a legközelebb, azonban ez zsákutca. A megoldás, hogy először elmegyünk Vasluira – egy olyan lépést teszünk, ami a heurisztika szerint a céltól távolabb visz – és aztán elmegyünk Csalánosra, Bukarestre, majd Fogarasra. Ebben az esetben a heurisztika szükségtelen csomópontok kifejtését eredményezi. Továbbá, ha nem ügyelünk arra, hogy felismerjük az ismétlődő állapotokat, akkor soha nem találjuk meg a megoldást – a keresés Neamt és Iaşi között fog oszcillálni.

A mohó legjobbat-először keresés a mélységi keresésre hasonlít abból a szempontból, hogy egyetlen út végigkövetését preferálja a célig, azonban zsákutcába jutva visszalép. Ugyanazokkal a problémákkal küszködik, mint a mélységi keresés – nem optimális és nem teljes (mert elindulhat egy végtelen úton és soha sem tér vissza újabb lehetőségeket kipróbálni). A mohó keresés legrosszabb esetre számított (worst-case) idő- és tárigénye $O(b^m)$, ahol m a keresési tér maximális mélysége. Jól megválasztott heurisztikus függvénnyel a komplexitás azonban jelentősen csökkenthető. A csökkenés mértéke az adott problémától és a heurisztikus függvény minőségétől függ.

1.2. A* keresés: a teljes becsült útköltség minimalizálása

A legjobbat-először keresés leginkább ismert változata az **A* keresés** (a kiejtése 'A csillag'). A csomópontokat úgy értékeli ki, hogy összekombinálja $g(n)$ értékét – az aktuális csomópontig megtett út költsége – és $h(n)$ értékét – vagyis az adott csomóponttól a célhoz vezető út költségének becslőjét:

$$f(n) = g(n) + h(n)$$

Created by XMLmind XSL-FO Converter.

Mivel $g(n)$ megadja a kiinduló csomóponttól az n csomópontig számított útköltséget, és $h(n)$ az n csomóponttól a célcsomópontba vezető legolcsóbb költségű út költségének becslője, így az alábbi összefüggést kapjuk:

$f(n)$ = a legolcsóbb, az n csomóponton keresztül vezető megoldás becsült költsége.

Így amennyiben a legolcsóbb megoldást keressük, ésszerű először a legkisebb $g(n) + h(n)$ értékkel rendelkező csomópontot kifejteni. Ezen stratégia kellemes tulajdonsága, hogy ez a stratégia több mint ésszerű: amennyiben a h függvény eleget tesz bizonyos feltételeknek, az A* keresés teljes és optimális.

Az A* optimalitását könnyű elemezni, ha az algoritmust a FA-KERESÉS-sel együtt alkalmazzuk. Ilyenkor A* optimális lesz, ha $h(n)$ egy **elfogadható heurisztika** (**admissible heuristic**), azaz ha $h(n)$ *soha nem becsüli felül* a cél eléréséhez szükséges költséget. Az elfogadható heurisztikák természetükből adódóan optimisták, mivel úgy gondolják, hogy a probléma megoldása kisebb költséggel jár, mint amekkora költséget a megoldás valójában igényel. Mivel $g(n)$ az n csomópont elérésének pontos költsége, azonnali következményként adódik, hogy $f(n)$ soha sem becsüli túl az n csomóponton keresztül vezető legjobb megoldás valódi költségét.

Az elfogadható heurisztikus függvények talán egyik legnyilvánvalóbb példája a Bukarestbe történő utazás során felhasznált h_{LMT} légvonalban mért távolság. A légvonalban mért távolság elfogadható, mert bármely két pont között a legrövidebb távolság a légvonalban mért távolság, így a légvonalbeli táv soha nem becsülhet túl. A 4.3. ábra a Bukarestet kereső A* fakeresés előrehaladását mutatja. A g értékeket a 3.2. ábrán látható lépésköltségekből számítjuk ki, a h_{LMT} értékei a 4.1. ábrán adottak.

Fontos

Vegyük észre, hogy Bukarest először az (e) lépés peremében jelent meg, azonban kifejtésre nem került, mert Bukarest f-értéke (450) magasabb, mint Piteştié (417). Ezt úgy lehetne megmagyarázni, hogy *lehet*, hogy Piteştin keresztül létezik egy 417 költségű olcsó megoldás, így az algoritmus egy 450 költségű megoldást nem fog választani. Ebből a példából megalkothatjuk annak az általános bizonyítását, hogy a FA-KERESÉS-t használó A* algoritmus *optimális, ha* $h(n)$ *elfogadható*. Tegyük fel, hogy a peremen egy G_2 szuboptimális célcsomópont jelenik meg, és az optimális megoldás költsége legyen C^*. Így mivel G_2 szuboptimális és $h(G_2) = 0$ (ami minden célállapotra igaz), tudjuk, hogy:

$f(G_2) = g(G_2) + h(G_2) = g(G_2) > C^*$

Gondoljunk most egy perembeli n csomópontra, amely a megoldási útvonalon fekszik (ilyennek mindig léteznie kell, ha a megoldás létezik). Ha $h(n)$ nem becsüli túl a megoldáshoz vezető út folytatását, akkor tudjuk, hogy:

$f(n) = g(n) + h(n) \leq C^*$

Kimutattuk tehát, hogy $f(n) \leq C^* < f(G_2)$, így G_2 nem kerül kifejtésre, és az A* egy optimális megoldással tér vissza.

A bizonyítás összeomlik, ha a FA-KERESÉS helyett a 3.19. ábra szerinti GRÁF-KERESÉS algoritmust használjuk. Az algoritmus visszatérhet szuboptimális megoldással, mert a GRÁF-KERESÉS algoritmus elvetheti az ismétlődő állapothoz vezető optimális utat, ha az nem elsőnek került kiszámításra (lásd 4.4. feladat). A problémát kétféle módon lehet megoldani. Az első megoldás a GRÁF-KERESÉS olyan kiterjesztése, hogy az az ugyanahhoz a csomóponthoz vezető két út közül a drágábbat fogja elvetni (lásd az értékelést a 3.5. alfejezetben). A pótlólagos adminisztrálás nem egyszerű, de az optimalitást garantálni fogja. A második megoldásnál azt kell biztosítani, hogy a bármelyik ismétlődő csomóponthoz vezető optimális út mindig az, amit az algoritmus elsőnek követ – mint ahogy ez az egyenletes költségű keresésénél volt. Ez a tulajdonság akkor áll fenn, ha a $h(n)$ függvényre extra követelményeket fogalmazunk meg, megkövetelve annak **konzisztenciá**ját (**consistency**), másképpen **monotonitás**át (**monotonicity**). A $h(n)$ heurisztikus függvény konzisztens, ha minden n csomópontra és annak egy tetszőleges a cselekvéssel generált minden n' utódcsomópontjára az n csomóponttól elért cél becsült költsége nem kisebb, mint az n'-be kerülés lépésköltsége és az n' csomóponttól elért cél becsült költsége:

$h(n) \leq c(n, a, n') + h(n')$

Ez az általános **háromszög egyenlőtlenség** (**triangle inequality**) egy formája, amely azt fejezi ki, hogy egy háromszög egy oldala sem lehet hosszabb, mint a két másik oldal összege. Itt a háromszöget az n, az n' és az n-hez legközebbi cél határozza meg. Könnyű megmutatni (4.7. feladat), hogy minden konzisztens heurisztika egyben elfogadható is.

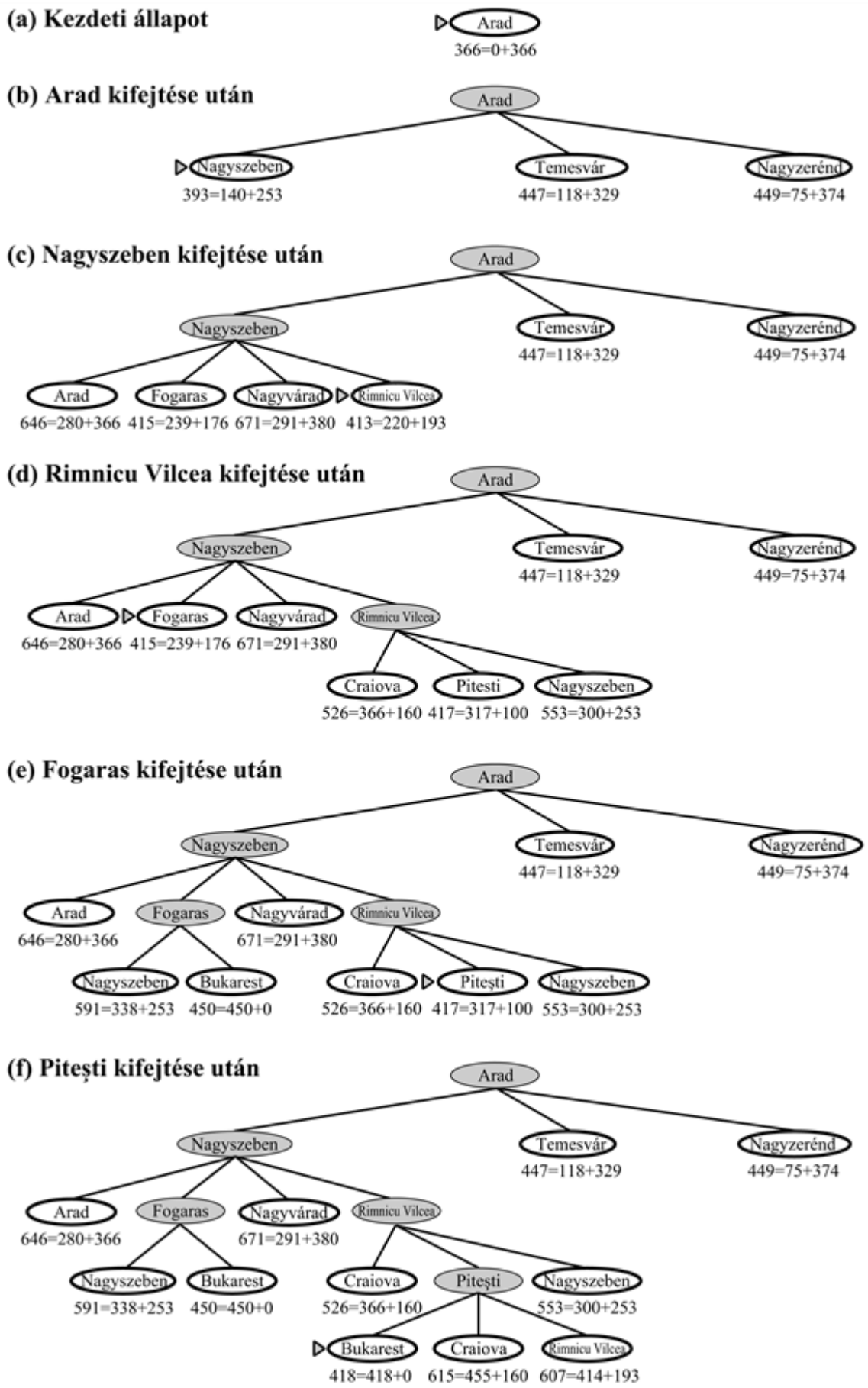
Created by XMLmind XSL-FO Converter.

Fontos

A konzisztencia legfontosabb következménye az, hogy: *a* `GRÁF-KERESÉS`*-t használó* A^* *algoritmus optimális, ha* $h(n)$ *konzisztens*.

4.3. ábra - Az A* keresés lépései Bukarest keresése során. A csomópontok az *f* = *g* + *h* értékekkel vannak felcímkézve. A *h*-értékek a Bukaresttől légvonalban mért távolságokat jelölik, melyeket a 4.1. ábrából vettünk át.

Created by XMLmind XSL-FO Converter.



Created by XMLmind XSL-FO Converter.

Bár a konzisztencia az elfogadhatóságnál szigorúbb követelmény, igazán nehéz olyan elfogadható heurisztikát találni, ami nem lenne egyben konzisztens. Az ebben a fejezetben tárgyalt összes elfogadható heurisztika mind konzisztens. Vegyük például a h_{LMT}-t. Tudjuk, hogy az általános háromszög egyenlőtlenség teljesül, ha az oldalakat egyenes vonalú távolságokkal mérjük, és hogy az n és n' közötti egyenesvonalú távolság $c(n, a, n')$-nél nem nagyobb. A h_{LMT} így egy konzisztens heurisztika.

Fontos

A konzisztencia egy másik fontos következménye az, hogy *ha $h(n)$ konzisztens, akkor az $f(n)$ értékek akármilyen út mentén nem csökkennek*. A bizonyítás bizonyos a mellett közvetlenül következik a konzisztencia definíciójából. Tegyük fel, hogy n' az n utódja, ekkor:

$$g(n') = g(n) + c(n, a, n')$$

és

$$f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$$

A GRÁF-KERESÉS-t használó A* algoritmus által kifejtett csomópontok sorozata tehát $f(n)$-ben nem csökkenő. A kifejtésre választott első célcsomópont így egy optimális megoldás is egyben, mert minden utána következő csomópont legalább ilyen költséges lenne.

Ha az f-költségek sohasem csökkenek, bármilyen utat választunk, akkor az állapottérben **határvonal**akat (**contour**) húzhatunk be, hasonlóan egy topografikus térkép kontúrjaihoz. A 4.4. ábra erre mutat egy példát. A 400-zal felcímkézett határvonalon belül az összes csomópont $f(n)$ értéke nem nagyobb 400-nál, és hasonló érvényes a többire is. Mivel az A* keresési algoritmus a legkisebb f értékkel rendelkező levélcsomópontot fejti ki először, láthatjuk, hogy az A* keresési algoritmus a gyökércsomópontból legyezőszerűen halad kifelé, növekvő f értékekhez tartozó koncentrikus sávokban hozzáadva a csomópontokat.

Az egyenletes költségű keresés esetén (A* keresés $h = 0$ mellett) a csomópontsávok a kiinduló csomópont köré húzott koncentrikus „köröket” alkotnak. Pontosabb heurisztikus függvény alkalmazásával a sávok a célállapot felé elnyúlnak, és keskenyebben fókuszálódnak az optimális út körül. Ha C^* az optimális megoldási út költségét jelöli, akkor az alábbiakat jelenthetjük ki:

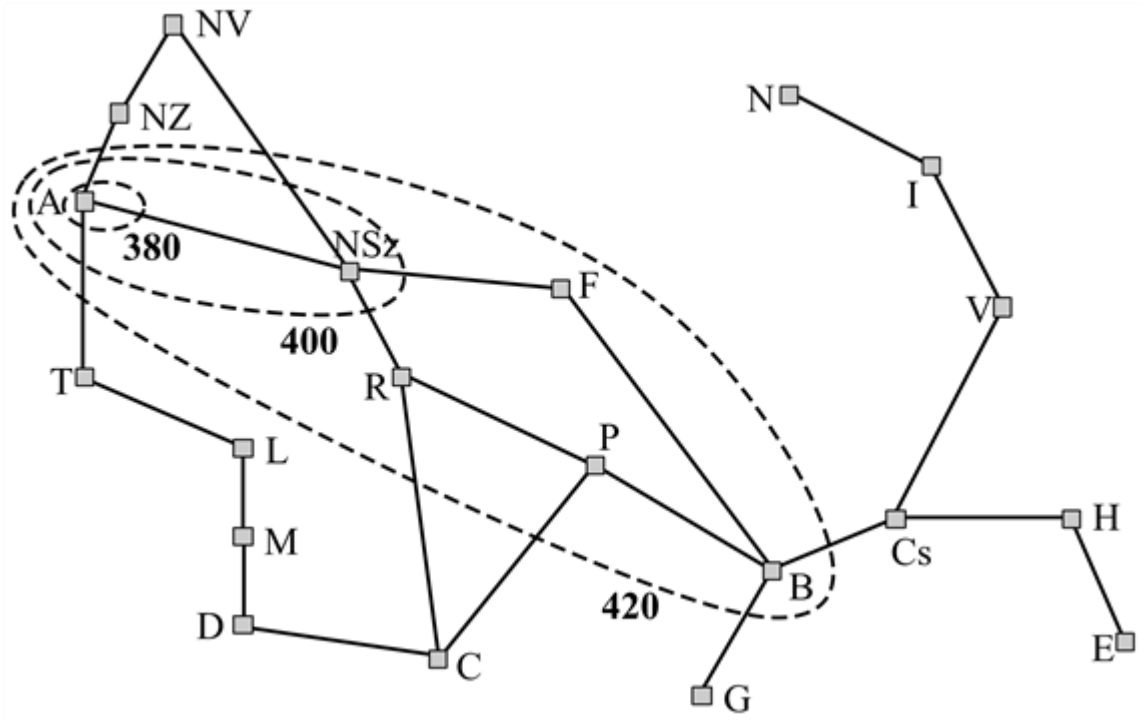
- Az A* keresési algoritmus kifejti az összes $f(n) < C^*$ értékkel rendelkező csomópontot.
- Ezek után az A* keresési algoritmus egy célcsomópont kiválasztása előtt még kifejthet néhány csomópontot a „célhatárvonalon”, amelyekre $f(n) = C^*$.

Intuitíven nyilvánvaló, hogy az első megtalált megoldásnak optimális megoldásnak kell lennie, hiszen a következő határvonalakon az összes csomóponthoz nagyobb f költség, ebből adódóan nagyobb g költség tartozik (mivel minden célállapotra $h(n) = 0$). Intuitíven az is nyilvánvaló, hogy az A* keresési algoritmus teljes. Ahogy egyre növekvő f értékű sávokat adunk a kereséshez, előbb-utóbb elérünk egy sávot, amelyhez tartozó f érték megegyezik egy célállapothoz vezető út költségével.[4]

4.4. ábra - Románia térképe. Az ábra Arad mint kiinduló állapot esetén az $f = 380$, $f = 400$ és $f = 420$ értékekhez tartozó határvonalat mutatja. Egy adott határvonalon belüli csomópontokhoz a határvonal értékénél kisebb f költség tartozik.

[4] A teljesség megkívánja, hogy a C^* költségnél kisebb vagy azzal egyenlő költségű csomópontokból véges sok legyen. Ez a feltétel akkor lesz igaz, ha minden lépésköltség egy véges ε-nál nagyobb és b véges.

Created by XMLmind XSL-FO Converter.



Vegyük észre, hogy az A* nem fejt ki $f(n) > C^*$ tulajdonságú csomópontokat. A 4.3. ábrán például Temesvár nem kerül kifejtésre annak ellenére, hogy ez a gyökérnek egy utódcsomópontja. Azt mondjuk, hogy a Temesvár alatti fát **lenyestük (pruned).** Mivel h_{LMT} elfogadható, az algoritmus biztonságosan figyelmen kívül hagyhat egy ilyen fát, miközben az optimalitást garantálja. A nyesés gondolata – a lehetőségek eliminálása anélkül, hogy megvizsgálnánk azokat – igen fontos az MI sok területén.

Egy végső észrevétel, az ilyen típusú – a gyökérből kiinduló utakat bővítő – optimális algoritmusok közül az A* keresési algoritmus bármely adott heurisztikus függvény mellett **optimális hatékonyságú (optimally efficient).** Ez azt jelenti, hogy egyetlen más optimális algoritmus sem fejt ki garantáltan kevesebb csomópontot, mint az A* (kivéve talán az $f(n) = C^*$ típusú csomópontok körét, ahol holtverseny alakulhat ki). Ez azért van, mert az összes olyan algoritmus, amelyik *nem* fejti ki az összes csomópontot, melyre $f(n) < C^*$, kockáztatja az optimális megoldás elkerülését.

Felettébb örömteli hír számunkra, hogy az A* keresési algoritmus teljes, optimális és az összes ilyen jellegű algoritmus között optimálisan hatékony. Sajnos ez azonban nem jelenti azt, hogy az A* algoritmus megoldja az összes kereséssel kapcsolatos problémánkat. A buktató a dologban az, hogy a legtöbb probléma esetén a csomópontok száma a keresési tér célhatárvonalon belüli részén a megoldás hosszának még mindig exponenciális függvénye. Bár az eredmény bizonyítása túlmutat ezen könyv keretein, megmutatható, hogy az exponenciális növekedéssel mindenképp szembe kell néznünk, kivéve, ha a heurisztikus függvényünk hibája legfeljebb az aktuális útköltség logaritmusával nő. Az exponenciálisnál lassabb növekedés feltétele matematikai megfogalmazásban:

$$|h(n) - h^*(n)| \leq O(\log h^*(n))$$

ahol $h^*(n)$ az n csomópontból a célcsomópontba való eljutás *valódi* költségét jelöli. Majdnem minden, a gyakorlatban használt heurisztikus függvény esetén a hiba legalább arányos az útköltséggel, és az ebből adódó exponenciális növekedéssel egyetlen számítógép sem tud megbirkózni. Ezért ahhoz ragaszkodni, hogy egy optimális megoldást találjunk, gyakran nem is praktikus. Használhatjuk az A* olyan változatát, amely a szuboptimális megoldásokat gyorsan megtalálja, vagy pedig dolgozhatunk pontosabb, de nem elfogadható heurisztikákkal. Egy jól megválasztott heurisztikus függvény ettől függetlenül a nem informált keresési algoritmusokhoz képest jelentős megtakarítást eredményezhet. A 4.2. alfejezetben megvizsgáljuk, hogy hogyan is lehet jó heurisztikus függvényt tervezni.

Az A* algoritmusnak azonban nem a szükséges számítási idő a nagy problémája. Mivel az összes legenerált csomópontot a memóriában tárolja (ahogy ezt az összes `GRÁF-KERESÉS` algoritmus teszi), ezért az algoritmus általában lényegesen hamarabb felemészti a rendelkezésére álló memóriát, mintsem kifutna az időből. Ezért az A* sok nagyméretű problémához nem praktikus. A közelmúltban kifejlesztett algoritmusok a végrehajtási idő

Created by XMLmind XSL-FO Converter.

kismértékű növekedése mellett a memóriaproblémát megoldották, anélkül hogy feláldoznák az optimalitást vagy a teljességet. Ezekkel az algoritmusokkal fogunk most foglalkozni.

1.3. Memóriakorlátozott heurisztikus keresés

Az A* memóriaigényének mérséklésére a legegyszerűbb módszer az iteratívan mélyülő algoritmus adaptálása heurisztikus keresés környezetre. Ennek eredménye az **iteratívan mélyülő A* algoritmus** – **IMA*** – (**iterative deepening A***, **IDA***). Az IMA* és a közönséges iteratívan mélyülő algoritmus közötti fő különbség az, hogy a vágási mechanizmus nem a mélységen, hanem az *f* költségen ($g + h$) alapul. Ezáltal minden egyes iterációban a vágási érték az a legkisebb *f* költség, ami az előbbi iterációban használt vágási értéknél nagyobb. Az IMA* praktikus megoldás számos olyan probléma esetén, ahol egységnyi a lépésköltség és elkerüli a rendezett csomópontsor memóriában való tartásának jelentős overheadjét. Sajnos, az IMA* algoritmus a valós értékű költségektől ugyanúgy szenved, mint az egyenletes költségű keresés iteratív változata, amit a 3.11. feladatban írtunk le. Ebben a fejezetben megvizsgálunk a memóriakorlátozott algoritmusok köréből két frissebb ötletet – az RLEK-t és az MA*-t.

A **rekurzív legjobbat-először keresés**, az **RLEK** (**recursive best-first search**, **RBFS**) egy egyszerű rekurzív algoritmus, amely megkísérli a rendes legjobbat-először algoritmus működését mímelni, de csak lineáris tárat használva. Az algoritmust a 4.5. ábra mutatja. A struktúrája hasonlít a rekurzív mélységi keresésre, azonban ahelyett, hogy az algoritmus egy utat a végtelenségig folytatna az aktuális pálya mentén, figyeli az aktuális csomóponthoz az elődeitől vezető eddigi legjobb alternatív út *f*-értékét. Ha az aktuális csomópont ezt az értéket túlhaladja, a rekurzió az alternatív útra lép vissza. Ahogy a rekurzió visszalép, az RLEK minden csomópont *f*-értékét a pálya mentén a gyerekeinek legjobb *f*-értékével helyettesíti. Ily módon az RLEK emlékszik a legjobb levélértékre az elfelejtett alfában, és eldöntheti, vajon érdemes-e ezt a fát valamikor később újra kifejteni. A 4.6. ábra azt mutatja, hogy hogyan éri el Bukarestet az RLEK.

Az RLEK valamivel hatékonyabb, mint az IMA*, azonban még mindig túlságosan szenved a csomópontok túlzott újbóli generálása miatt. A 4.6. ábrán látható példában az RLEK először a Rimnicu Vilcean átmenő utat követi, majd „meggondolja magát” és Fogarassal kísérletezik, majd újra meggondolja magát. Ezek a meggondolások azért történnek meg, mert amikor az aktuális legjobb utat tovább fejtjük, nagy az esély arra, hogy az *f*-érték nőni fog – *h* általában kevésbé optimista a célhoz közeli csomópontok esetén. Amikor ez megtörténik, különösképpen nagy keresési terekben, a második legjobb út a legjobb úttá válhat, és a keresésnek vissza kell lépnie, hogy ezt az utat tudja követni. Minden meggondolás megfelel az IMA* egy-egy iterációjának és az elfelejtett csomópontok számos újbóli kifejtését teszi szükségessé, hogy a keresés képes legyen a legjobb utat visszaállítani, és azt egy csomóponttal meghosszabbítani.

4.5. ábra - A rekurzív legjobbat-először keresés algoritmusa

```
function REKURZÍV-LEGJOBBAT-ELŐSZÖR-KERESÉS(probléma) returns egy megoldás vagy kudarc
    RLEK(probléma, CSOMÓPONTOT-LÉTREHOZ(KIINDULÓ-ÁLLAPOT[probléma], ∞)

function RLEK(probléma, csomópont, f_korlát) returns egy megoldás, vagy kudarc,
    és egy új f-költség korlát
    if CÉL-TESZT[probléma](ÁLLAPOT[csomópont]) then return csomópont
    követők ← KIFEJT(csomópont, probléma)
    if követők egy üres lista then return kudarc, ∞
    for each s in követők do
        f[s] ← max(g(s) + h(s), f[csomópont])
    repeat
        legjobb ← a legkisebb f-értékű csomópont a követők közül
        if f[legjobb] > f_korlát then return kudarc, f[legjobb]
        alternatíva ← a második legkisebb f-értékű csomópont a követők közül
        eredmény, f[legjobb] ← RLEK(probléma, legjobb, min(f_korlát, alternatíva))
        if megoldás ≠ kudarc then return megoldás
```

Az A*-hoz hasonlóan az RLEK is optimális algoritmus, feltéve, hogy a $h(n)$ heurisztikus függvény elfogadható. Tárkomplexitása $O(bd)$, az időkomplexitását azonban nehezebb meghatározni. Ez függ a heurisztika pontosságától és attól is, mennyire gyakran változik a csomópontok kifejtése közben a legjobb út. Mind az IMA*, mind az RLEK elvileg ki vannak téve a gráfokban való kereséssel kapcsolatos exponenciális

Created by XMLmind XSL-FO Converter.

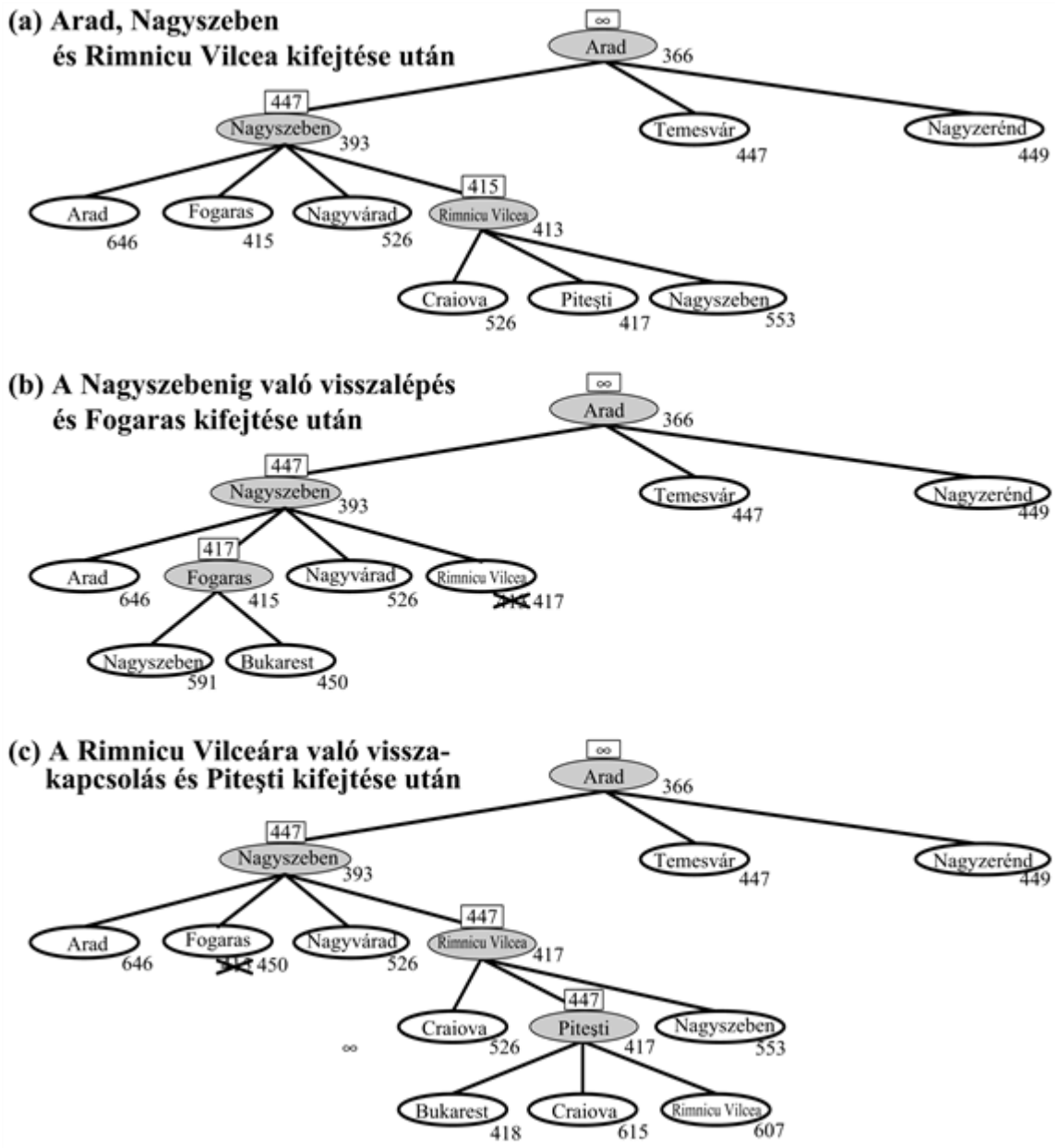
komplexitásnövekedésnek (lásd 3.5. alfejezet), tekintettel arra, hogy az ismétlődő állapotok jelenlétét csak az aktuális úton képesek ellenőrizni. Előfordulhat így, hogy ugyanazt az állapotot többször is kifejtik.

Az IMA*-nak és az RLEK-nek az a baja, hogy *túl kevés* memóriát használ. Az egyes iterációk között az IMA* egyetlen számot, az aktuális *f*-költség korlátot tárolja el. Az RLEK több információt tárol el a memóriában, azonban csak $O(bd)$ memóriát használ. Még ha több memória is állna a rendelkezésére, az RLEK-nek nincs módja ezt kihasználni.

Ésszerűnek tűnik tehát az összes, rendelkezésre álló memóriát használni. Az erre képes két algoritmus az **MA*** (**memóriakorlátozott A***) és az **EMA*** (**egyszerűsített MA***). Itt az EMA*-t írjuk le, ami – mitagadás – az egyszerűbb. Az EMA* az A* módjára halad a legjobb levelet kifejtve, amíg a memória be nem telik. Ezen a ponton a keresési fához új csomópontot hozzáadni nem képes, hacsak egy régit nem töröl ki. Az EMA* mindig a *legrosszabb* – a legmagasabb *f*-értékű – csomópontot hagyja ki. Majd, mint az RLEK, az elfelejtett csomópont értékét a szülőjéhez továbbítja. Ily módon egy elfelejtett részfa elődje tudja a részfa legjobb útjának az értékét. Ezzel az információval az EMA* csak akkor fejti ki újra a fát, ha kimutatta, hogy *minden más út* rosszabbnak tűnik, mint az elfelejtett út. Más szóval, ha az *n* csomópont minden utódját elfelejtjük, nem tudjuk, hogy *n*-ből merrefelé lehetne menni, tudni fogjuk azonban, hogy mennyire érdemes egyáltalán bárhová is menni az *n*-ből kiindulva.

4.6. ábra - Az RLEK lépései, miközben a legrövidebb utat keresi Bukarest felé. A mindenkori rekurzív hívás *f*-korlát értéke a mindenkori aktuális csomópont felett látható. (a) A Rimnicu Vilceán át vezető utat az algoritmus addig követi, amíg az aktuális legjobb levél (Piteşti) értéke nem lesz rosszabb, mint a legjobb alternatív út értéke (Fogaras). (b) A rekurzió visszalép és az elfelejtett alfa legjobb levélértékét (417) Rimnicu Vilceánál feljegyezzük. Majd Fogaras kifejtése következik 450-nel, mint a legjobb levélértékkel. (c) A rekurzió visszalép. Az elfelejtett alfa legjobb levélértékét (450) Fogarasnál jegyezzük fel. Következik Rimnicu Vilcea kifejtése. Ezúttal, mivel a legjobb alternatív út (Temesváron keresztül) 447-be kerül, a kifejtés folytatódik Bukarest felé.

Created by XMLmind XSL-FO Converter.



A teljes algoritmus túlságosan bonyolult, hogy itt írjunk róla,[5] azonban egy finom vonását érdemes megemlíteni. Azt mondtuk, hogy az EMA* a legjobb levelet fejti ki és a legrosszabb levelet felejti el. Mi van akkor, ha *minden* levélnek ugyanaz az *f*-értéke? Az algoritmus ekkor ugyanazt a csomópontot kifejtésre is, és elhagyásra is kiválaszthatná. Az EMA* ezt a problémát úgy oldja meg, hogy a kifejtésre a *legújabb* csomópontot választja, és a *legrégebbi* csomópontot törli. Ez a kettő ugyanaz a csomópont csak akkor lehet, ha csak egy levél van. Ebben az esetben az aktuális keresési fa egyetlen útból áll a gyökértől a levélig, ami kitölti a teljes memóriát. Ha a levél nem egy célcsomópont, akkor *még ha a célhoz vezető optimális úton fekszik is*, ez a cél az adott memóriával nem érhető el. Következésképpen a csomópontot ugyanúgy el lehet dobni, mintha nem is lenne követője.

Az EMA* teljes, ha van egyáltalán elérhető megoldás – azaz ha *D*, a legsekélyebb célcsomópont mélysége kevesebb, mint a memória nagysága (csomópontokban kifejezve). Optimális, ha van elérhető optimális megoldás, másképpen az algoritmus a legjobb elérhető megoldással tér vissza. Gyakorlatilag az EMA*-t messze a legjobb általánosan használatos algoritmusnak lehet tekinteni az optimális megoldások megkeresésére, különösképpen ha az állapottér egy gráf, a lépésköltség nem egyenletes, és a csomópontkifejtés drága a nyitott és zárt listák karbantartásának pótlólagos overheadjéhez képest.

Fontos

[5] Az algoritmus vázlatos leírása a könyv első kiadásában megtalálható.

Created by XMLmind XSL-FO Converter.

Nagyon nehéz problémák esetén azonban az EMA* sokszor kénytelen folyamatosan oda-vissza kapcsolgatni a lehetséges megoldási utak között, amelyekből csak kevés fér be a memóriába (hasonló ez a diszkalapú memória lapozó rendszer **vergődés**i (**thrashing**) problémájára). Az ugyanazon csomópontok ismételt újrakifejtéséhez szükséges extra idő azt jelenti, hogy azok a problémák, amelyeket a végtelen memóriájú A* gyakorlatilag meg tudna oldani, az EMA* számára kezelhetetlenek. Ez azt jelenti, hogy *memóriakorlát a problémát a számítási idő szempontjából kezelhetetlenné teheti.* Bár a memória és az idő közötti kompromisszum magyarázatára nincs elmélet, úgy tűnik, hogy ettől a problémától megmenekülni nem lehet. Egyetlen kiút elvetni az optimalitás követelményét.

1.4. Tanuljunk, hogy jobban keressünk!

Az eddigiekben néhány rögzített stratégiát mutattunk be – szélességi, mohó legjobbat-először stb. keresés –, amelyeket számítógépes szakemberek terveztek. Meg tudná-e *tanulni* egy ágens, hogy jobban keressen? A válasz igen, és a módszer alapját egy fontos fogalom, a **metaszintű állapottér** (**metalevel state space**) adja. A metaszintű állapottér minden állapota az **objektumszintű állapottér**beli (**object-level state space**) – amilyen például Románia – keresőprogram egy belső (számítási) állapotának felel meg. Az A* algoritmus belső állapota például az aktuális keresési fa. A metaszintű állapottér minden cselekvése egy számítási lépés, amely a belső állapotot változtatja meg. Az A* esetén például minden számítási lépés kifejt egy levelet, és a követőit a fához adja hozzá. A 4.3. ábra, amely az egyre növekvő keresési fák sorozatát mutatja, értelmezhető lehetne úgy, hogy a metaszintű állapottérben egy utat mutat, ahol az út minden állapota egy objektumszintű keresési fa.

A 4.3. ábrán az út öt lépésből áll, Fogaras kifejtését is beleszámítva, ami nem volt valami hasznos. Nehezebb problémák esetén sok ilyen téves lépésre lehet számítani, és a **metaszintű tanulási algoritmus** (**metalevel learning**) ilyen tapasztalatokból tanulhat, hogy elkerülje a haszontalan részfák kifejtését. Az ilyen tanuláshoz alkalmas módszereket a 21. fejezetben tárgyaljuk. A tanulás célja a problémamegoldás **totális költség**ének (**total cost**) a minimalizálása, kompromisszumot kötve a számítási kiadások és az útköltség között.

2. Heurisztikus függvények

Ebben a részben a 8-as kirakójátékhoz keresünk heurisztikus függvényeket, hogy rávilágítsunk a heurisztikus függvények általános természetére.

A 8-as kirakójáték a legrégebbi heurisztikus keresési feladatok egyike. Mint azt a 3.2. alfejezetben láttuk, a játék lényege, hogy a számozott lapkákat vízszintesen és függőlegesen az üres helyre tolva a kiindulóállásból a célállásba jussunk (lásd 4.7. ábra).

Egy átlagos megoldás véletlen módon generált 8-as kirakójáték példányok esetén kb. 22 lépésből áll. Az elágazási tényező nagyjából 3 (amikor az üres lapka középen van, akkor négy, amikor a sarokban van, akkor kettő, és amikor valamelyik szélső középső pozícióban van, akkor pedig három mozgatás lehetséges). Ebből adódóan egy 22 mélységig menő kimerítő keresés közelítőleg $3^{22} \approx 3{,}1 \times 10^{10}$ állapotot vizsgálna meg. Az ismétlődő állapotok nyilvántartásával ezt a számot 170 000-ed részére le lehet csökkenteni, mert csak 9!/2 = 181 440 különböző elérhető elrendezés létezik (lásd 3.4. feladat). Ez egy kezelhető szám, azonban ugyanez a szám a 15-ös kirakójáték esetén már durván 10^{13}, így a következő teendő egy jó heurisztika megkeresése. Amennyiben a legrövidebb megoldásokat akarjuk megtalálni A* bevetésével, olyan heurisztikus függvényre van szükségünk, ami soha sem becsüli túl a célállapot eléréséhez szükséges lépések számát. Az ilyen heurisztikák keresésének a 15-ös kirakójáték esetén nagy a múltja. Íme két lehetséges függvény:

- h_1 = a rossz helyen lévő lapkák száma. A 4.7. ábrán a 8 lapkából egyik sincs a helyén, így a kiinduló állapotban $h_1 = 8$ lenne. h_1 elfogadható heurisztikus függvény, mivel nyilvánvaló, hogy minden rossz helyen lévő lapkát legalább egyszer mozgatni kell.

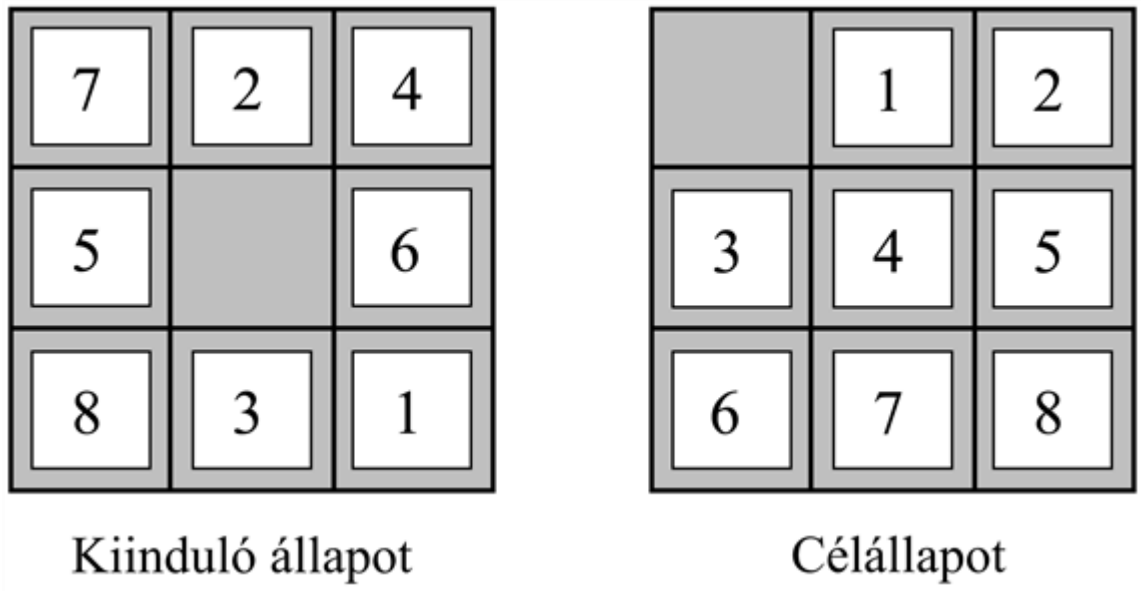
- h_2 = a lapkáknak a saját célhelyeiktől mért távolságaik összege. Mivel a lapkákat nem lehet átlók mentén mozgatni, az általunk kiszámított távolság a vízszintes és függőleges távolságok összege lesz. Ezt néha **háztömb**- (**city block distance**) vagy **Manhattan-távolság**nak (**Manhattan distance**) is szokás nevezni. h_2 szintén elfogadható heurisztikus függvény, mivel minden egyes mozgatással egy lapkát csak egy lépéssel lehet közelebb vinni a célhoz. A kiinduló állapotban az 1–8 lapkákra számított Manhattan-távolság:

$h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$

Ahogy reméltük, egyik sem becsüli túl a megoldás igazi költségét, ami 26.

Created by XMLmind XSL-FO Converter.

4.7. ábra - A 8-as kirakójáték egy tipikus feladata. A megoldás 26 lépés hosszú.



2.1. A heurisztikus függvény pontosságának hatása a megoldás hatékonyságára

A heurisztikus függvény minősítésének egyik lehetséges módja a b^* **effektív elágazási tényező** (**effective branching factor**) megadása. Amennyiben az A* algoritmus által kifejtett összes csomópont száma egy adott problémára N, és a megoldás mélysége d, akkor b^* annak a d mélységű kiegyensúlyozott fának az elágazási tényezőjével egyezik meg, amely $N + 1$ csomópontot tartalmazna. Ebből adódóan:

$$N + 1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$

Például ha az A* algoritmus egy 5 mélységben fekvő megoldást 52 csomópont kifejtésével talál meg, akkor az effektív elágazási tényező 1,92. Az effektív elágazási tényező a problémaesetek függvényében változhat, megfelelően nehéz problémák esetén azonban általában nagyjából állandó. Így a b^* kisszámú problémahalmazon végzett kísérleti mérése jó fogódzót adhat a heurisztikus függvény általánosságban vett használhatóságáról. Egy jól megtervezett heurisztikus függvény effektív elágazási tényezője 1 körüli érték, ami lehetővé teszi felettébb nagy problémák megoldását is.

A h_1 és a h_2 heurisztikus függvények teszteléséhez véletlenszerűen generáltunk 1200 problémapéldányt, 2–24 mélységű megoldással (100-100 példányt minden páros szám esetére), és megoldottuk azokat a h_1 és a h_2 heurisztikus függvényeket alkalmazva az A* fakereső algoritmussal, illetve a nem informált iteratívan mélyülő keresési algoritmussal is. A 4.8. ábra minden egyes stratégiára megadja az átlagosan kifejtett csomópontok számát és az effektív elágazási tényezőt. Az eredmények azt mutatják, hogy h_2 jobb, mint h_1, és hogy a nem informált keresés mindkettőnél sokkal rosszabb. A 14 hosszúságú megoldások esetén az A* h_2-vel 30 000-szer hatékonyabb, mint a nem informált iteratívan mélyülő keresés.

4.8. ábra - A keresési költség és az effektív elágazási tényező összehasonlítása `ITERATÍVAN-MÉLYÜLŐ-KERESÉS` és A* esetén h_1 és h_2 használatával. Az adatokat különböző hosszúságú megoldásokra a 8-as kirakójáték 100 egyedét megoldva átlagolva kaptuk.

Created by XMLmind XSL-FO Converter.

	Keresési költség			Effektív elágazási tényező		
d	IMK	A*(h_1)	A*(h_2)	IMK	A*(h_1)	A*(h_2)
2	10	6	6	2,45	1,79	1,79
4	112	13	12	2,87	1,48	1,45
6	680	20	18	2,73	1,34	1,30
8	6384	39	25	2,80	1,33	1,24
10	47127	93	39	2,79	1,38	1,22
12	364404	227	73	2,78	1,42	1,24
14	–	539	113	–	1,44	1,23
16	–	1301	211	–	1,45	1,25
18	–	3056	363	–	1,46	1,26
20	–	7276	676	–	1,47	1,27
22	–	18 094	1219	–	1,48	1,28
24	–	39 135	1641	–	1,48	1,26

Feltehetjük magunknak a kérdést, hogy vajon a h_2 mindig jobb-e, mint a h_1? A válasz: igen. A két heurisztikus függvény definíciójából jól látszik, hogy minden n csomópontra $h_2(n) \geq h_1(n)$. Azt mondjuk, hogy h_2 **dominál**ja h_1-et. A dominációt közvetlenül átvihető a hatékonyságra: a h_2-t használó A* algoritmus kevesebb csomópontot fog kifejteni, mint a h_1-et használó (talán csak az $f(n) = C^*$ tulajdonságú néhány csomópontot kivéve). Ezt az alábbi egyszerű gondolatmenettel mutathatjuk meg. Idézzük fel a 1.2. szakasz - A* keresés: a teljes becsült útköltség minimalizálása részben tett észrevételünket, miszerint minden csomópont kifejtésre kerül, amelyre $f(n) < C^*$. Ezzel egyenértékű az az állítás, mely szerint minden csomópont kifejtésre kerül, amelyre $h(n) < C^* - g(n)$. Mivel azonban h_2 minden csomópontra legalább akkora, mint h_1, így minden olyan csomópontot, amit kifejt a h_2-t alkalmazó A* algoritmus, kifejti a h_1-et alkalmazó A* algoritmus is, de h_1 alkalmazása más csomópontok kifejtését is okozhatja. Ebből adódóan mindig jobb nagyobb értékeket adó heurisztikus függvényeket alkalmazni, amíg nem becsüljük túl a valódi költséget, és a heurisztika számítási ideje nem túlságosan nagy.

2.2. Elfogadható heurisztikus függvények kitalálása

Láttuk, hogy mind h_1 (a nem a helyükön lévő lapkák száma), mind pedig h_2 (Manhattan-távolság) egész jó heurisztikus függvények a 8-as kirakójáték problémájához, és hogy h_2 jobbnak bizonyult. De hogyan is állt elő a h_2? Egy számítógép számára vajon lehetséges-e mechanikusan megalkotni ilyen heurisztikus függvényeket?

A h_1 és h_2 a 8-as kirakójátékban a fennmaradó út hosszát becsülik, azonban a játék *egyszerűsített* változatánál a tökéletesen pontos úthossz értékét adják meg. Ha a játék szabályait úgy módosítanánk, hogy egy lapka bárhová áthelyezhető legyen, nemcsak a szomszédos mezőkre, akkor h_1 pontosan megadná a legrövidebb megoldáshoz vezető lépések számát. Hasonlóan, ha egy lapkát bármelyik szomszédos mezőre átmozgathatnánk, még akkor is, ha az adott mezőn már van egy másik lapka, akkor h_2 megadná a legrövidebb megoldás pontos lépésszámát.

Fontos

Az olyan problémát, amelyben az operátorokra kevesebb megkötést teszünk, mint az eredeti problémában, **relaxált problémá**nak (**relaxed problem**) nevezzük. *A relaxált probléma optimális megoldásának költsége egy elfogadható heurisztika az eredeti problémára.* A heurisztika elfogadható, mert az eredeti probléma optimális megoldása definíciószerűen megoldása a relaxált problémának is, és így legalább olyan költséges, mint a relaxált probléma optimális megoldása. Mivel a számított heurisztika a relaxált problémára egy pontos költség, teljesítenie kell a háromszög egyenlőtlenséget, és ebből kifolyólag **konzisztens** 1.2. szakasz - A* keresés: a teljes becsült útköltség minimalizálása részben.

Created by XMLmind XSL-FO Converter.

Amennyiben a probléma megfogalmazása formális nyelven adott, akkor a relaxált problémákat automatikusan is elő lehet állítani.[6] Például ha a 8-as kirakójáték operátorait az alábbi módon írjuk le:

Egy lapka az *A* mezőről a *B* mezőre mozgatható,

ha *A* és *B* szomszédosak, és a *B* mező üres

egy vagy több feltétel törlésével három relaxált problémát hozhatunk létre:

a. Egy lapka az *A* mezőről a *B* mezőre mozgatható, ha *A* és *B* szomszédosak.

b. Egy lapka az *A* mezőről a *B* mezőre mozgatható, ha a *B* mező üres.

c. Egy lapka az *A* mezőről a *B* mezőre mozgatható.

Az (a)-ból a h_2 (Manhattan-távolság) vezethető le. A magyarázat az, hogy h_2 helyes eredményt adna, ha minden lapkát sorra elmozgatnánk a saját célhelyére. A (b)-ből származtatott heurisztikával a 4.9. feladatban foglalkozunk. A (c)-ből a h_1 (a nem a helyükön lévő lapkák száma) vezethető le, mert ez lenne a helyes eredmény, ha a lapkákat a célpozíciójukba egy lépésben el lehetne mozgatni. Vegyük észre, hogy lényeges, hogy e módszer által generált relaxált problémákat lényegében keresés nélkül meg lehet oldani, mert a relaxált szabályok a probléma 8 független részproblémává történő dekompozícióját teszik lehetővé. Ha a relaxált problémát nehéz megoldani, akkor a kapcsolatos heurisztikus értékek számítása drágának fog bizonyulni.[7]

Az ABSOLVER nevű program a „relaxált probléma” módszer és más egyéb módszer alkalmazásával (Prieditis, 1993), képes a probléma definíciójából automatikusan heurisztikus függvényeket generálni. Az ABSOLVER egy, az eddigieknél jobb heurisztikus függvényt hozott létre a 8-as kirakójáték megoldására, és ez a program találta meg az első használható heurisztikus függvényt a híres Rubik-kocka kirakásához.

Az új heurisztikus függvények előállításának egyik problémája, hogy felettébb nehéz felismerni a „nyilvánvalóan legjobb” heurisztikus függvényt. Ha egy problémához adottak a h_1, ..., h_m elfogadható heurisztikus függvények, és egyik sem dominálja a többit, melyiket kell választanunk? Mint az majd kiderül, nem kell választanunk. Az alábbi formulával a lehető legjobbat kaphatjuk meg:

$$h(n) = \max\{h_1(n), \ldots, h_m(n)\}$$

Az így megkonstruált összetett heurisztikus függvény mindig azt a függvényt használja, amelyik az adott csomópontra a legpontosabb. Mivel az alkotóelemként felhasznált heurisztikus függvények mind elfogadhatók, ezért *h* is elfogadható. Ugyancsak könnyű bizonyítani, hogy *h* konzisztens. Továbbá *h* dominálja az összes, benne alkotóelemként felhasznált heurisztikus függvényt.

Elfogadható heurisztikus függvényeket származtathatunk az adott probléma **részproblémá**jának (**subproblem**) megoldási költségéből is. A 4.9. ábra például a 4.7. ábrán látható 8-as kirakójáték egy részproblémáját mutatja. A részprobléma lényege az 1, 2, 3 és 4 lapkák helyükre való mozgatása. Világos, hogy e részprobléma optimális megoldásának költsége a teljes probléma megoldásának költségét alulról korlátozza. Bizonyos esetekben ez a függvény a Manhattan-távolságnál sokkal pontosabb.

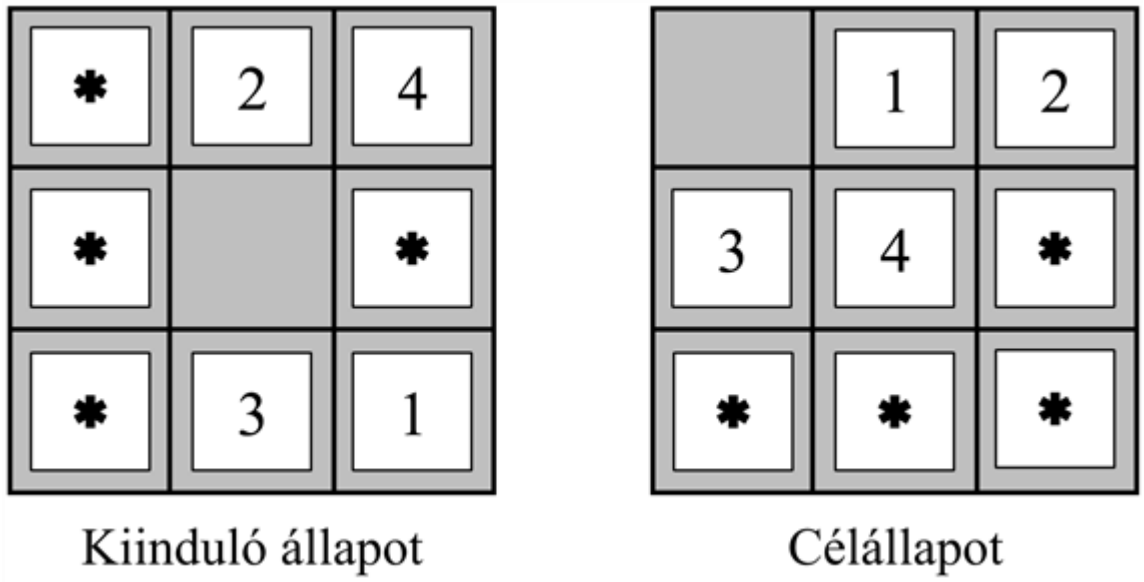
A **mintaadatbázis**ok (**pattern database**s) hátterében húzódó ötlet az, hogy tároljuk el az egyes részproblémaesetekhez tartozó pontos megoldási költségeket – ebben az esetben a négy lapka és az üres hely minden konfigurációjához (jegyezzük meg, hogy a másik négy lapka helye a részprobléma megoldása szempontjából közömbös, mozgásaik azonban a költségbe bele fognak számítani). A probléma teljes állapotához tartozó elfogadható h_{AB} heurisztikus függvényt ezek után keresés közben úgy számítjuk ki, hogy a megfelelő részprobléma-konfigurációt az adatbázisból kikeressük. Maga az adatbázis úgy lett megtervezve, hogy a célállapottól visszafelé keresve minden új minta költségét feljegyeztük. Ennek a keresésnek a költsége a sok egymás után jövő problémaesetre kivetítve amortizálódik.

[6] A 8. és a 11. fejezetben erre a feladatra alkalmas formális nyelveket írunk le. Manipulálható formális leírásokkal a relaxált problémák konstruálása automatizálható. Egyelőre természetes nyelvet használunk.

[7] Vegyük észre, hogy a tökéletes heurisztikát könnyűszerrel megkaphatnánk, ha megengednénk *h*-nak a teljes szélességi keresés lefuttatását „stikában”. A heurisztikus függvény pontossága és számítási ideje között így kompromisszumot kell kötni.

Created by XMLmind XSL-FO Converter.

4.9. ábra - A 4.7. ábrán látható 8-as kirakójáték eset egy részproblémája. A feladat az, hogy 1, 2, 3 és 4 lapkát a helyes pozíciókba juttassuk el, nem törődve azzal, hogy a többi lapkával mi fog történni.



Az 1-2-3-4 lapka megválasztása nyilván tetszőleges, az adatbázist az 5-6-7-8 és a 2-4-6-8 stb. lapkákra is meg tudnánk konstruálni. Minden adatbázis egy elfogadható heurisztikát szolgáltat, és ahogy korábban elmagyaráztuk, ezen heurisztikákat kombinálni lehet, maximális értékükkel számolva. Egy ilyen típusú kombinált heurisztika a Manhattan-távolságnál sokkal pontosabb. A 15-ös kirakójáték véletlenszerűen generált egyedeire a kifejtett csomópontok számát egy 1000-es tényezővel lehet csökkenteni.

El lehetne tűnődni azon, hogy az 1-2-3-4 és az 5-6-7-8 adatbázisokból kinyert heurisztikus függvényeket nem lehetne-e *összeadni,* mivel úgy tűnik, a két részproblémában nincs fedés. A válasz nemleges, mert egy adott állapotban az 1-2-3-4 és az 5-6-7-8 részproblémák megoldásai majdnem biztosan osztoznak közös lépéseken – igen valószínűtlen, hogy az 1-2-3-4 lapkákat a helyükre el lehet mozdítani, anélkül hogy az 5-6-7-8-hoz ne nyúlnánk hozzá, és *fordítva*. És mi lenne, ha ezeket a lépéseket nem is számolnák be? Azaz nem az 1-2-3-4 részprobléma teljes költségével számolnánk, hanem csak az 1-2-3-4 lapka lépéseit vennénk figyelembe. Könnyű ilyenkor rájönni, hogy a két költség összege még mindig a teljes probléma megoldási költségének egy alsó korlátja. Ez a **diszjunkt mintaadatbázis** (**disjoint pattern database**) hátterében húzódó ötlet. Az ilyen adatbázisok használatával lehetségessé válik a 15-ös kirakójátékot milliszekundumok alatt megoldani, a Manhattan-távolsághoz képest a generált csomópontok száma egy 10 000-es tényezővel kisebb. A 24-es kirakójáték esetén kb. egymilliós gyorsítást lehet elérni.

A diszjunkt mintaadatbázisok jól működnek a csúszólapka-játékok esetén, mert a problémát úgy lehet felbontani, hogy egy-egy lépés csak egy részproblémára van hatással, hiszen egyszerre csak egy lapkát mozgatunk. A Rubik-kocka problémára ilyen bontást elvégezni nem lehet, mert minden mozgás 8, 9 vagy 26 kockát érint. Nem tudjuk egyelőre, hogy az ilyen problémák számára a diszjunkt mintaadatbázisokat hogyan lehetne definiálni.

2.3. A heurisztikus függvény tanulása tapasztalatból

A $h(n)$ heurisztikus függvénytől elvárjuk, hogy az n csomópontbeli állapottól kezdve becsülje a megoldás költségét. Hogyan lenne képes egy ágens egy ilyen függvény megalkotására? Egy megoldást az előbbi részben adtunk meg – ami nevezetesen egy olyan relaxált problémának a kitalálása, amihez egy optimális megoldást könnyű találni. Egy másik megoldás a tapasztalatból való tanulás. A „tapasztalaton” itt azt értjük, hogy nagyon sok 8-as kirakójátékot kell megoldani. A 8-as játék minden optimális megoldása egy példát jelent, amiből $h(n)$ tanulható. Minden példa egy, a megoldási úton elhelyezkedő állapotból és az onnan számított valós megoldási költségből áll. Ilyen példák alapján **induktív tanulási algoritmus** (**inductive learning**) segítségével egy olyan $h(n)$ függvényt konstruálhatunk, amely (szerencsével) képes megjósolni a megoldás költségét a keresés során felbukkanó más állapotok esetén is. Ennek módszertanát, legyenek ezek neurális hálók, döntési fák vagy más módszerek, a 18. fejezet mutatja be (a 21. fejezetben leírt megerősítéses tanulás szintén alkalmazható).

Created by XMLmind XSL-FO Converter.

Az induktív tanulási módszerek akkor működnek a legjobban, ha nem az állapot nyers leírását, hanem inkább az állapot kiértékeléséhez releváns **jellemző**ket (**feature**) kapják bemenetként. A „nem a helyén lévő lapkák száma” jellemző segítség lehet, ha az aktuális állapotnak a céltól vett távolságát szeretnénk megjósolni. Hívjuk ezt a jellemzőt $x_1(n)$-nek. Vehetnénk a 8-as kirakójáték 100 véletlenszerűen generált konfigurációját, és gyűjthetnénk statisztikákat a megoldás aktuális költségéről. Esetleg azt találnánk, hogy ha $x_1(n)$ értéke 5, akkor a megoldás átlagos költsége 14 és így tovább. Ilyen adatok birtokában $x_1(n)$ értékéből jósolni lehetne $h(n)$ értékét. Persze több jellemzőt is lehetne használni. Egy másik jellemző, $x_2(n)$ lehetne például „azon szomszédos lapkapárok száma, melyek a célállapotban is szomszédosak”. Hogyan lehetne $x_1(n)$-et és $x_2(n)$-et összekombinálni $h(n)$ jóslása érdekében? Egy szokásos ötlet a lineáris kombináció használata:

$$h(n) = c_1 x_1(n) + c_2 x_2(n)$$

A c_1 és c_2 konstansokat módosítani lehet, hogy a megoldási költség aktuális adataira legjobban illeszkedjenek. Feltételezhetően c_1-nek pozitívnak, c_2-nek pedig negatívnak kellene lennie.

3. Lokális kereső algoritmusok és optimalizációs problémák

Az eddig látott keresési algoritmusokat arra tervezték, hogy a keresési tereket szisztematikusan tárják fel. A szisztematikusságot úgy érik el, hogy egy vagy több utat tartanak a memóriában, és azt is feljegyzik, hogy minden pontban az út mentén melyik alternatívát vizsgálták már meg, és melyiket nem. Amikor megtalálják a célt, a célhoz vezető *út* egyben a probléma *megoldását* is jelenti.

Számos problémában azonban a célhoz vezető út érdektelen. A 8-királynő problémában például 2.1. szakasz - Játékproblémák a királynők végleges konfigurációja számít, és nem az a sorrend, ahogy az újabb királynőket felhelyezzük. A problémák ezen osztályába olyan fontos problémák tartoznak, mint a VLSI-tervezés, a gyári gépelrendezés, a gyártási műveletek ütemezése, az automatikus programozás, a hírközlési hálózatok optimalizálása, a gépkocsiútvonal-tervezés és a portfóliómenedzsment.

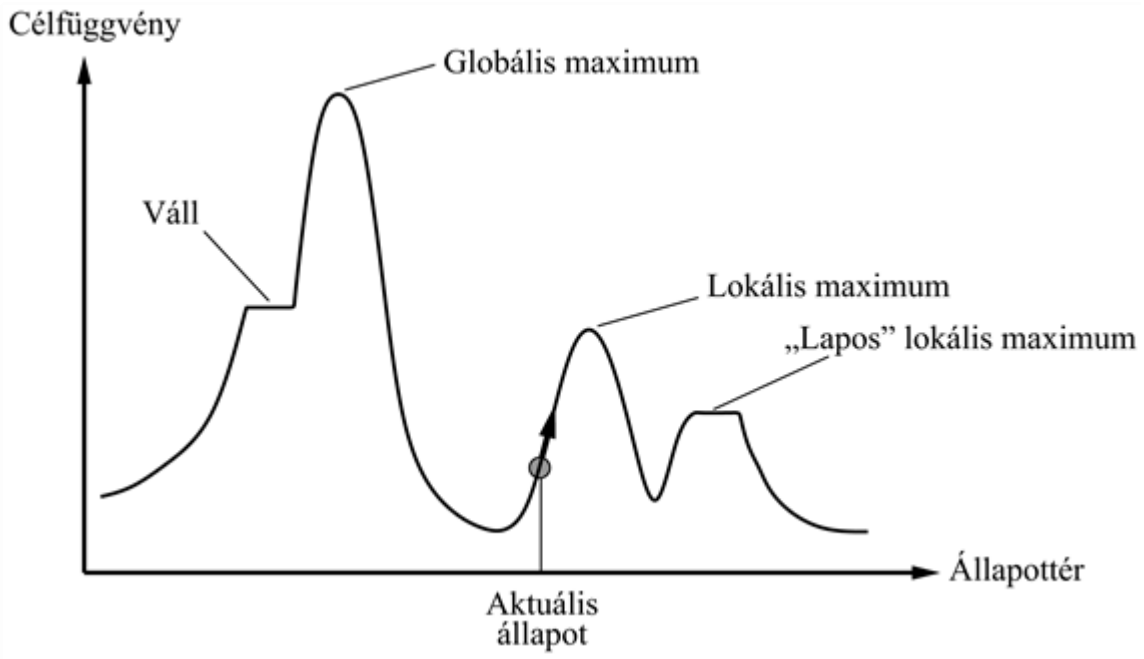
Ha a célhoz vezető út nem számít, más algoritmusokra is gondolhatunk, amelyek az utakkal egyáltalán nem foglalkoznak. A **lokális keresési algoritmus**ok (**local search**) csak egy **aktuális állapot**ot (**current state**) vesznek figyelembe (a többszörös utak helyett) és általában csak ennek az állapotnak a szomszédjaira lépnek tovább. A keresés által követett utat tipikusan nem is tárolják el. Bár a lokális keresési algoritmusok nem szisztematikusak, két kulcsfontosságú előnyük van: (1) igen kevés – általában konstans mennyiségű – memóriát használnak, és (2) sokszor nagy vagy végtelen (folytonos) keresési térben elfogadható megoldást produkálnak ott, ahol a szisztematikus algoritmusok alkalmatlanok lennének.

A cél megfogalmazásán túl, a lokális keresési algoritmusok a tisztán **optimalizációs problémá**k (**optimization problems**) megoldásában is hasznosak, ahol a cél a legjobb állapot megtalálása egy **célfüggvény** (**objective function**) értelmében. Számos optimalizációs probléma nem felel meg a 3. fejezetben bevezetett „standard” keresési modellnek. Így például a természet előállít egy célfüggvényt – a szaporodási fitnesst – amit úgy tűnik, a darwini evolúció igyekszik optimalizálni, e probléma esetén nem beszélhetünk azonban sem „céltesztről”, sem „útköltségről”.

A lokális keresés megértéséhez igen hasznosnak találjuk az **állapottérfelszín**t (**state space landscape**) (mint amilyen a 4.10. ábrán látszik). A felszínnek van „pontja” (amit az állapot definiál) és „magassága” (amit a heurisztikus vagy a célfüggvény értéke határoz meg). Ha a magasság a költséggel arányos, akkor a cél a legalacsonyabban fekvő völgyet – a **globális minimum**ot – megtalálni. Ha a magasság a célfüggvénynek felel meg, akkor a cél a legmagasabb csúcs – a **globális maximum** – megtalálása (az egyikről a másikra előjel váltásával könnyen áttérhetünk). A lokális keresés ezt a felületet vizsgálja. Egy **teljes** lokális keresés mindig talál megoldást, ha az egyáltalán létezik. Egy **optimális** algoritmus mindig megtalálja a globális minimumot vagy maximumot.

4.10. ábra - Egy egydimenziós állapottérfelszín, ahol a magasság a célfüggvénynek felel meg. A cél a globális maximum megtalálása. A hegymászó keresés az aktuális állapotot módosítja a javulás irányában, ahogy ezt a nyíl is mutatja. A különböző topografikus jellemzőkkel a szövegben foglalkozunk.

Created by XMLmind XSL-FO Converter.



3.1. Hegymászó keresés

A **hegymászó keresési algoritmus**t (**hill-climbing**) a 4.11. ábra mutatja. A keresés egyszerűen csak egy ciklus, ami mindig javuló értékek felé – azaz felfelé – lép. Az algoritmus megáll, amikor felér a csúcsra, ahol nincsenek már magasabb értékű szomszédjai. Az algoritmus nem tart nyilván keresési fát, ezért a csomópontot leíró adatszerkezetnek csak az állapotot és a célfüggvény értékét kell nyilvántartania. A hegymászó keresés nem néz előre az aktuális állapotot közvetlenül követő szomszédokon túl. Egy kicsit arra hasonlít, mintha a Mount Everest csúcsát szeretnénk megtalálni sűrű ködben és emlékezetkihagyásban szenvedve.

4.11. ábra - A hegymászó keresési algoritmus (a legmeredekebb emelkedő (steepest ascent) változat) a lokális keresés alapvető módszere. Minden lépésben az aktuális csomópontot a legjobb szomszédjával cseréli le, ami ebben a változatban a legmagasabb ÉRTÉK-ű szomszédot jelenti. Ha azonban egy *h* heurisztikus költségbecslést alkalmazunk, ez akkor a legalacsonyabb *h*-jú szomszéd lenne.

```
function HEGYMÁSZÁS(probléma) returns egy lokális maximumot jelentő állapot
    inputs: probléma, egy probléma
    local variables: aktuális, egy csomópont
                     szomszéd, egy csomópont

    aktuális ← CSOMÓPONTOT-LÉTREHOZ(KIINDULÓ-ÁLLAPOT[probléma])
    loop do
        szomszéd ← az aktuális legnagyobb értékű követő csomópontja
        if ÉRTÉK[szomszéd] ≤ ÉRTÉK[aktuális] then return ÁLLAPOT[aktuális]
        aktuális ← szomszéd
    end
```

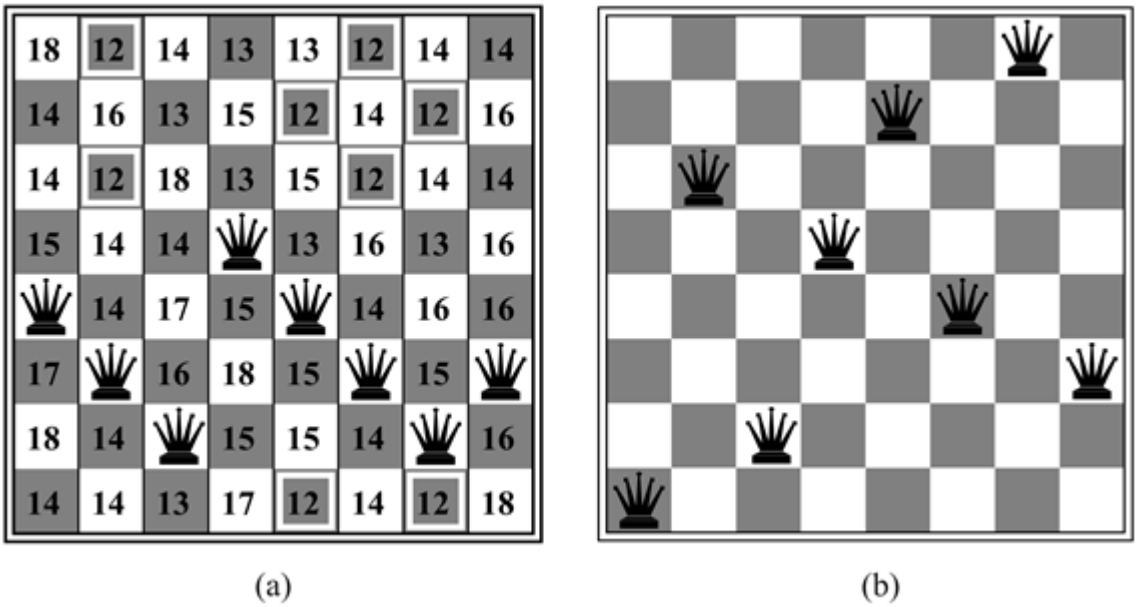
A hegymászó keresés illusztrálására a 2.1. szakasz - Játékproblémák bevezetett **8-királynő problémá**t (**8-queens problem**) fogjuk használni. A lokális keresési algoritmusok tipikusan a **teljes állapot leírás**sal (**complete-state formulation**) élnek, ahol minden állapotban a táblán 8 királynő helyezkedik el, oszloponként egy. Az állapotátmenet-függvény minden olyan lehetséges állapotot visszaad, amit úgy kapunk, hogy egy királynőt ugyanabban az oszlopban egy másik mezőre mozgatunk (így minden állapotnak $8 \times 7 = 56$ követője van). A *h* heurisztikus függvény a közvetlenül vagy közvetett módon egymást támadó királynőpárok száma. E függvény globális minimuma 0, ami csak a tökéletes megoldásban érhető el. A 4.12. (a) ábra egy $h = 17$ értékű

Created by XMLmind XSL-FO Converter.

állapotot mutat. Az ábra az összes követőnek az értékét is mutatja, ahol a legjobb követőé $h = 12$. Ha a legjobból több van, a hegymászó algoritmusok tipikusan véletlen módon sorsolnak belőlük egyet.

A hegymászó keresést néha **mohó lokális keresés**nek (**greedy local search**) is hívják, mert egy jó követő állapotot megragad, anélkül hogy megfontolná, merre lenne érdemes továbbmenni. Bár a mohóság a hét főbűn egyike, a mohó algoritmusok sokszor igen jól teljesítenek. A hegymászás gyakran igen gyorsan halad a megoldás felé, mert egy rossz állapoton általában nagyon könnyű javítani. Így például a 4.12. (a) ábrán látható állapotból indulva öt lépés elegendő, hogy elérjük a 4.12. (b) ábrán látható állapotot, melynek költsége $h = 1$, és amely igen közel van egy megoldáshoz. Sajnos a hegymászás gyakran megakad az alábbi problémák miatt:

4.12. ábra - (a) A 8-királynő probléma $h = 17$ heurisztikus költségbecslésű állapota, ahol minden olyan követő állapot h értékét megadtuk, amikor a királynőt a saját oszlopában mozgatjuk. A legjobb lépéseket bejelöltük. (b) Egy lokális minimum a 8-királynő térben. Az állapot h értéke egységnyi, azonban minden követőnek magasabb a költsége.



- **Lokális maximum**ok: egy lokális maximum egy csúcs, amely minden szomszédjánál magasabb, de a globális maximumnál alacsonyabb. A hegymászó algoritmusok, ha egy lokális maximum közelébe érnek, kénytelenek a csúcs felé tartani, ott azonban nincs tovább, és megakadnak. Ezt a problémát sematikusan a 4.10. ábra mutatja. Konkrétabban, a 4.12. (b) ábra állapota egy tényleges lokális maximum (azaz egy lokális minimum a h költségre nézve), bármely királynő bármilyen lépése csak ront a helyzeten.

- **Hegygerinc**ek (**ridge**s): egy hegygerincet a 4.13. ábra mutat. A hegygerinc egy olyan lokális maximum sorozatot eredményez, ahol egy mohó algoritmusnak igen nehéz navigálnia.

- **Fennsík** (**plateaux**): a fennsík az állapottérnek egy olyan területe, ahol a kiértékelő függvény gyakorlatilag lapos. Lehet ez egy lapos lokális maximum, amelyből nincs tovább felfelé, de lehet egy **váll** (**shoulder**), ahonnan még lehetséges az előrehaladás (lásd 4.10. ábra). Egy hegymászó keresés képtelen lehet arra, hogy egy fennsíkról megtalálja a kivezető utat.

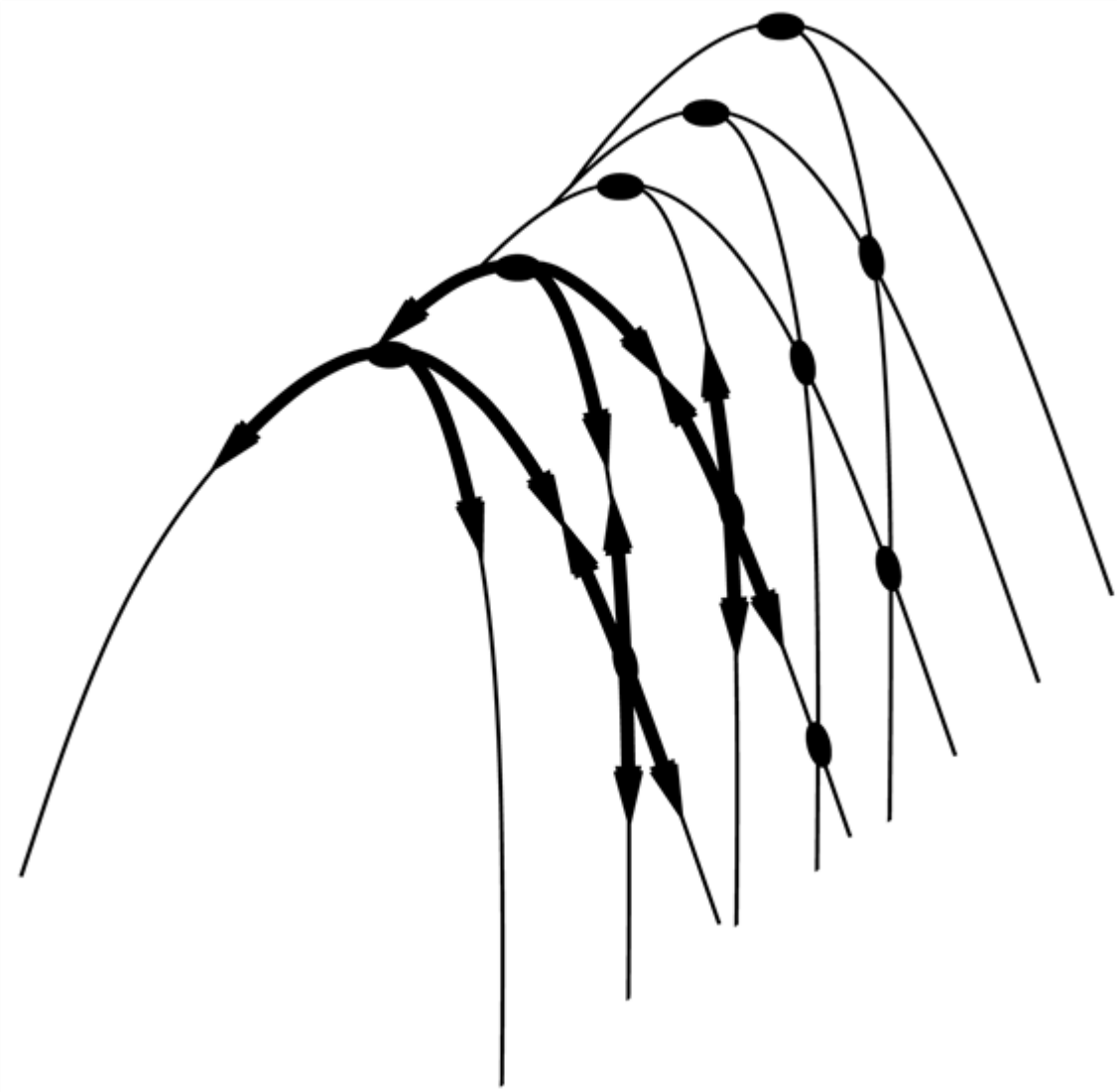
Az algoritmus minden esetben elér egy pontot, ahonnan már nem tud továbblépni. Egy véletűlen módon generált 8-királynő állapotból kiindulva a legmeredekebb emelkedő hegymászó algoritmus az esetek 86%-ában megakad, a problémaeseteknek csupán 14%-át oldja meg. Az algoritmus gyorsan dolgozik, átlagosan 4 lépést tesz, amikor sikerrel jár, és 3 lépést, amikor megakad. Ez egyáltalán nem is olyan rossz a $8^8 \approx 17$ millió állapotot tartalmazó állapottérben.

A 4.11. ábra algoritmusa megáll, ha fennsíkra ér, ahol a legjobb követőnek is ugyanaz az értéke, mint az aktuális állapotnak. Nem lenne-e jó ötlet mozgásban maradni – **oldallépések**et (**sideways move**) megengedve annak reményében, hogy a fennsík igazából egy váll, mint amilyent a 4.10. ábra mutat? A válasz általában igen, azonban óvatosan kell eljárnunk. Ha oldallépéseket mindig megengedünk, akkor ha felfelé haladás nincs, végtelen hurokba kerülünk minden olyan esetben, amikor az algoritmus olyan lapos lokális maximumot talál,

Created by XMLmind XSL-FO Converter.

amely nem váll. Gyakori megoldás az egymás után alkalmazott oldallépések számát korlátozni. A 8-királynő problémában például megengedünk 100 egymás utáni oldallépést. Ezzel a hegymászás által megoldott esetek arányát 14%-ról 94%-ra emelhetjük. A sikernek azonban ára van. Átlagosan 21 lépés hosszú minden sikerrel megoldott eset, és 64 lépés hosszú a kudarc.

4.13. ábra - Annak illusztrálása, hogy a gerinc miért jelent nehézséget a hegymászó keresés számára. Az állapotok rácshálózata (sötét körök) rá van illesztve a balról jobbra emelkedő gerincre, egy olyan lokális maximum sorozatot eredményezve, ahol a maximumok nincsenek közvetlen módon egymással összekapcsolva. Minden egyes lokális maximumból az összes lehetséges cselekvés a lejtőn lefelé mutat.



A hegymászó algoritmus számos változatát fejlesztették ki. A **sztochasztikus hegymászó keresés** (**stochastic hill climbing**) a felfelé mutató irányokból véletlen módon választ. A választás valószínűsége a felfelé mutató irány meredekségével változhat. Az algoritmus általában a legmeredekebb emelkedő módszernél lassabban konvergál, egyes állapotfelszíneken azonban képes jobb megoldást találni. Az **elsőnek-választott hegymászó algoritmus** (**first-choice hill climbing**) sztochasztikus hegymászó keresést használ, a követőket véletlen módon addig generálva, amíg az az aktuális állapotnál nem lesz jobb. Ez jó stratégia, ha egy állapotnak sok (például több ezer) követője van. Ennek vizsgálatára invitál a 4.16. feladat.

Az eddig leírt hegymászó algoritmusok nem teljesek – sokszor kudarcot vallanak a cél megkeresésében, mert egy lokális maximumba beragadnak. A **véletlen újraindítású hegymászás** (**random-restart hill-climbing**) az

Created by XMLmind XSL-FO Converter.

ismert közmondás szerint jár el: „Ha nem megy elsőre, csináld újra.” Véletlenül generált kiinduló állapotokból[8] hegymászó keresést végez, amíg célba nem ér. Az algoritmus 1-hez tartó valószínűséggel teljes annál a triviális oknál fogva, hogy előbb-utóbb a célállapotot kezdőállapotként is fogja generálni. Ha minden hegymászó keresés p valószínűséggel sikeres, a véletlen újraindítások várható száma $1/p$. A 8-királynő problémában, ha oldallépéseket nem engedünk meg, $p \approx 0{,}14$, így a cél megtalálásához átlagosan 7 iterációra van szükség (6 kudarc és 1 siker). A várható lépésszám a sikeres iteráció költsége valamint $(1 - p)/p$-szerese a kudarc költségének, durván 22 lépés. Ha oldallépéseket is engedünk, átlagosan $1/0{,}94 \approx 1{,}06$ iterációra és $(1 \times 21) + (0{,}06/0{,}94) \times 64 \approx 25$ lépésre van szükség. A 8-királynő probléma számára a véletlen újraindítású hegymászó keresés valóban hatékony. Még hárommillió királynő mellett is ez a megközelítés egy percen belül talál megoldást.[9]

A hegymászás sikere nagyban függ az állapottér „felszínének” alakjától: ha azon csak néhány lokális maximum és fennsík található, akkor a véletlen újraindítású hegymászó algoritmus gyorsan meg fog találni egy jó megoldást. Egy valódi problémához egy olyan felszín tartozik, ami leginkább egy sündisznócsaládhoz hasonlít egy sima padlón, ahol minden sündisznótüske csúcsát egy további miniatűr sündisznó lakja és így a *végtelenségig*. Ha a probléma NP-teljes, akkor minden bizonnyal exponenciálisan sok lokális maximummal rendelkezik, melyekben beragadhatunk. Ennek ellenére általában kisszámú újraindítás után már elfogadhatóan jó megoldást lehet találni.

3.2. Szimulált lehűtés

A hegymászó keresés, amely soha nem indul „lefelé a lejtőn” a kisebb értékű (vagy nagyobb költségű) állapotok felé, garantáltan nem teljes, mert egy lokális maximumban beragadhat. Ezzel ellentétben a tisztán véletlen vándorlás – azaz a követők halmazából egyenletesen véletlen módon sorsolt követőre való átlépés – teljes, de hihetetlenül nem hatékony. Értelmes dolognak tűnik a hegymászás és a véletlen vándorlás valamiféle ötvözése, hogy mind a teljességet, mind a hatékonyságot megtarthassuk. Egy ilyen algoritmus a **szimulált lehűtés** (**simulated annealing**). A kohászatban a **lehűtés** (**annealing**) a fémeket, illetve az üveget edző, keményítő folyamat, amikor azokat magas hőmérsékletre felmelegítjük, majd fokozatosan lehűtjük, lehetővé téve, hogy az anyag alacsony energiájú kristályos állapotba kerüljön. Hogy a szimulált lehűtést megértsük, a hegymászásról térjünk át a **gradiens leereszkedés**re (**gradient descent**) (azaz a költség minimalizálására), és képzeljük el, hogy az a feladatunk, hogy egy hepehupás asztalon egy pingponglabdát a legmélyebb szakadékba juttassunk. Ha a labdát gurulni hagyjuk, egy lokális minimumba kerül. Ha a felületet megrázzuk, a labdát kiugraszthatjuk a lokális minimumból. A trükk az, hogy olyan erősen kell megrázni a felületet, hogy a labda a lokális minimumból kikerüljön, de mégsem annyira erősen, hogy a labda a globális minimumból kiugorjon. A szimulált lehűtés olyan megoldás, hogy először erősen rázunk (azaz egy magas hőmérsékleten), majd fokozatosan csökkentjük a rázás intenzitását (vagyis csökkentjük a hőmérsékletet).

A szimulált lehűtés legbelső ciklusa (lásd 4.14. ábra) nagyon hasonlít a hegymászáshoz. A *legjobb* lépés megtétele helyett azonban egy *véletlen* lépést tesz. Ha a lépés javítja a helyzetet, akkor az mindig végrehajtásra kerül. Ellenkező esetben az algoritmus a lépést csak valamilyen 1-nél kisebb valószínűséggel teszi meg. A valószínűség exponenciálisan csökken a lépés „rosszaságával” – azzal a ΔE mennyiséggel, amivel a kiértékelő függvény értéke romlott. A valószínűség a T „hőmérséklet” csökkenésével is csökken. A „rossz” lépések az indulásnál T magasabb értékeinél valószínűbbek, T csökkenésével egyre valószínűtlenebbé válnak. Be lehet bizonyítani, hogy ha a *hűtési karakterisztika* T értékeit kellően lassan csökkenti, az algoritmus 1-hez tartó valószínűséggel a globális minimumban köt ki.

4.14. ábra - A szimulált lehűtés keresési algoritmus. A hegymászó keresés egy olyan változata, ahol lefelé tartó lépések is megengedettek. A lefelé tartó lépéseket a lehűtési terv elején inkább, az idő múlásával egyre kevésbé fogadjuk el. A *lehűtési terv*, mint az eljárás bemenete, a T értékeit mint időfüggvényt adja meg.

[8] Véletlen állapotot generálni egy implicit módon specifikált állapottérben azonban önmagában is igen kemény probléma lehet.
[9] Luby (Luby és társai, 1993) azt bizonyította be, hogy egyes esetekben az a legjobb, ha a véletlen keresést egy konkrét rögzített időtartam letelte után újraindítjuk, és ez sokkal hatékonyabb lehet annál, mintha mindegyik keresést a végtelenségig hagynánk folytatódni. Az oldallépések kitiltása vagy a számuk korlátozása ennek egy példája.

Created by XMLmind XSL-FO Converter.

```
function SZIMULÁLT-LEHŰTÉS(probléma, lehűtési terv) returns egy megoldási állapot
    inputs: probléma, egy probléma
            lehűtési terv, egy leképzés időről „hőmérsékletre”
    local variables: aktuális, egy csomópont
                     következő, egy csomópont
                     T, egy „hőmérséklet”, ami a lefelé lépések valószínűségét szabályozza

    aktuális ← CSOMÓPONTOT-LÉTREHOZ(KIINDULÓ-ÁLLAPOT[probléma])
    for t ← 1 to ∞ do
        T ← lehűtési terv[t]
        if T = 0 then return aktuális
        következő ← az aktuális egy véletlenszerűen kiválasztott követő csomópontja
        ΔE = ÉRTÉK[következő] – ÉRTÉK[aktuális]
        if ΔE > 0 then aktuális ← következő
        else aktuális ← következő csak e^(ΔE/T) valószínűséggel
```

A szimulált lehűtési algoritmust először elterjedten VLSI-elrendezési problémák megoldására használták a ’80-as évek elején. Azóta széles körben alkalmazzák ipari termelés ütemezésére és egyéb nagy volumenű optimalizációs feladatokra. A 4.16. feladat a szimulált lehűtési algoritmus és a véletlen újraindítású hegymászó algoritmus teljesítményének összehasonlítását kéri az *n*-királynő problémán.

3.3. Lokális nyaláb keresés

Mindössze egyetlen csomópontot tartani a memóriában eléggé extrém reagálásnak tűnik a memóriakorlát problémájára. A **lokális nyaláb keresés** (**local beam search**) algoritmus[10] nem egy, hanem *k* állapotot követ nyomon. Az algoritmus *k* véletlen módon generált állapottal indul. Minden lépésben a *k* állapot mindegyikének összes követőit kifejti. Ha ezek valamelyike egy cél, az algoritmus leáll. Egyébként a teljes listából kiválasztja a legjobb *k* követőt, és ezt az eljárást ismétli.

Fontos

Első látásra a *k* állapotú lokális nyaláb keresés nem tűnik másnak, mint a *k* véletlen újraindítás parallel futtatása a szekvenciális futtatás helyett. Valójában a két algoritmus igen különböző. A véletlen újraindítású algoritmusban minden keresési folyamat a többitől függetlenül fut le. *A lokális nyaláb keresési algoritmusban a* k *parallel keresési szál megosztja az információt*. Például ha az egyik állapot számos jó követőt generál, és a többi *k* – 1 mind rosszabbat, akkor az eredmény az, mintha az első állapot megüzenné a többinek: „Gyertek át ide, itt zöldebb a fű!” Az algoritmus gyorsan abbahagyja az eredménytelen kereséseket, és az erőforrásait oda viszi, ahol a legnagyobb előrehaladás érzékelhető.

A lokális nyaláb keresés a legegyszerűbb formájában a *k* állapot közötti változatosság hiányától szenvedhet. Az állapotok gyorsan koncentrálódhatnak a tér egy kicsi részében, amitől a keresés csak kevéssel lesz több, mint a hegymászó keresés egy drága változata. A problémát a **sztochasztikus nyaláb keresés** (**stochastic beam search**) segíti megoldani, ami analóg a sztochasztikus hegymászó kereséssel. A *k* legjobb követő megválasztása helyett az algoritmus a *k* követőt véletlen módon választja ki, ahol egy adott követő kiválasztásának valószínűsége az állapot értékének növekvő függvénye. A sztochasztikus nyalábkeresés egy kicsit hasonlít a természetes kiválasztódásra, ahol egy „állapot” (szervezet) „követői” (utódjai) a következő generációt az állapot „értéke” (fitness) alapján népesítik be.

3.4. Genetikus algoritmusok

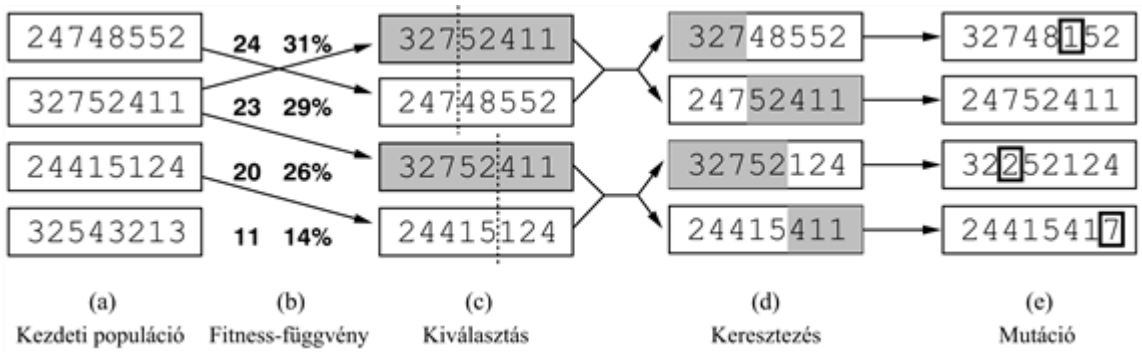
A **genetikus algoritmus**, **GA** (**genetic algorithm**) a sztochasztikus nyaláb keresés egy olyan variánsa, ahol a követő állapotokat nem egy állapot módosításával, hanem *két* szülő állapot összekombinálásával állítjuk elő. A természetes kiválasztódás analógiája itt is ugyanaz, mint a sztochasztikus nyaláb keresés esetén, azzal a különbséggel, hogy most az utódlétrehozásnak a szexuális és nem az aszexuális mechanizmusáról van szó.

[10] A lokális nyaláb keresés a **nyalábkeresés** (**beam search**) algoritmus egy adaptációja, ami viszont egy útalapú algoritmus.

Created by XMLmind XSL-FO Converter.

A nyalábkereséshez hasonlóan, a GA is *k* véletlen módon generált állapottal indul, aminek **populáció** (**population**) a neve. Minden állapotot vagy **egyed**et (**individual**) egy véges ábécé fölött értelmezett füzér képvisel – leggyakrabban egy 0-ból és 1-ekből álló füzér. A 8-királynő állapotnak például a 8 királynő pozícióját kell specifikálnia, mindegyik egy 8 négyzetet tartalmazó oszlopban, így $8 \times \log_2 8 = 24$ bitre van szükség. Az állapotot 8 számjeggyel is lehetne jellemezni, melyek mindegyike az 1 − 8 tartományból való (később látni fogjuk, hogy a két kódolás viselkedése eltérő). A 4.15. (a) ábra egy olyan populációt mutat be, mely négy, a 8-királynő állapotait reprezentáló 8 számjegyes füzérből áll.

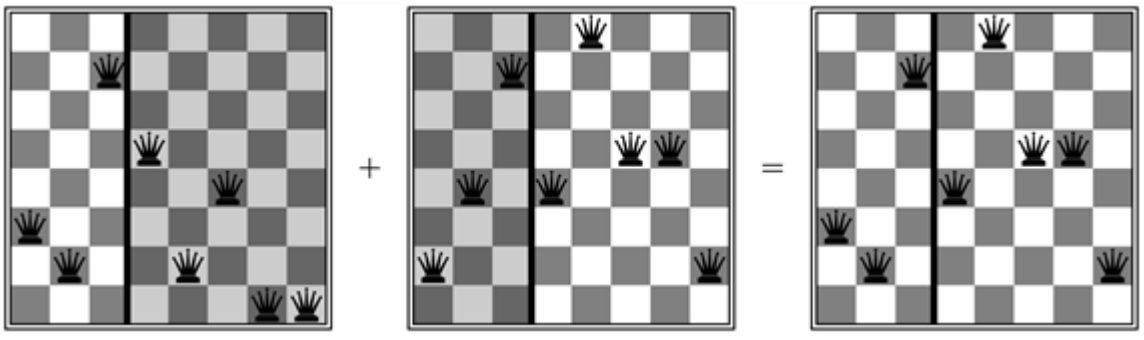
4.15. ábra - A genetikus algoritmus. Az (a)-beli kezdeti populációt (b)-ben fitness-függvény alapján rangsoroljuk, aminek eredményét a (c)-ben látható reprodukáló párok adják. A létrehozott utódok a (d)-ben láthatók, melyekre még hat a mutáció (e).



Az állapotok következő generációjának az előállítása a 4.15. (a)–(e) ábrán követhető. A (b)-ben minden állapotot a kiértékelő függvény vagy (GA-terminológiában) a **fitness-függvény** (**fitness function**) alapján rangsoroljuk. Egy fitness-függvénynek a jobb állapotokra magasabb értékeket kell visszaadnia, így a 8-királynő problémában fitness-függvényként a *nemtámadó* királynőpárok számát használjuk. Ennek értéke egy megoldás esetén 28. A négy állapot értéke 24, 23, 20 és 11. A genetikus algoritmus e konkrét változatában a szaporodásra való kiválasztás valószínűsége a fitness-értékkel egyenesen arányos. A százalékos mennyiségeket a fitness-értékek mellett adjuk meg.

A (c)-ben két párt véletlenszerűen választunk ki reprodukcióra, a (b)-beli valószínűségeknek megfelelően. Figyeljük meg, hogy egy egyedet kétszer, egy másikat viszont egyszer sem választottunk ki.[11] Minden keresztezendő párnál egy **keresztezés**i (**crossover**) pontot választunk a füzérbeli pozíciók közül. A 4.15. ábrán a kereszteződési pontok az első párnál a harmadik számjegy után, a második párnál az ötödik számjegy után vannak.[12]

4.16. ábra - A 4.15. (c) ábrán látható első két szülőnek és a 4.15. (d) ábrán látható két utódnak megfelelő 8-királynő állapotok. Az árnyalt oszlopok a kereszteződés során elvesznek, a nem árnyalt oszlopok megmaradnak.



A (d)-ben az utódok generálását látjuk, a szülő füzéreket a keresztezési pontoknál keresztezve. Az első pár első gyereke például az első szülőtől az első három számjegyét kapja és a másik szülőtől a többit. A második gyerek

[11] Ennek a kiválasztási szabálynak számos változata van. Kimutatható, hogy a **selejtezés** (**culling**) módszere, ahol egy küszöb alá eső minden egyedet eldobunk, a véletlen változatnál gyorsabban konvergál (Baum és társai, 1995).
[12] Itt számít a kódolás. Ha a 8 számjegyes kódolás helyett a 24 bites kódolást használjuk, a kereszteződési pontnak 2/3-ad esélye van, hogy egy számjegy belsejébe essen, ami ennek a számjegynek lényegében tetszőleges mutációját eredményezi.

Created by XMLmind XSL-FO Converter.

pedig az első három számjegyét a második szülőtől kapja és az első szülőtől a többit. Az ennél a reprodukciós lépésnél szereplő 8-királynő állapotokat a 4.16. ábra mutatja. A példa azt a tényt szemlélteti, hogy ha a két szülő állapot igen különböző, a keresztezés eredménye mindkét szülőtől igen távol eshet. Sokszor előfordul, hogy a keresési folyamat elején a populáció eléggé változatos, és a keresztezés (a szimulált lehűtéshez hasonlóan) nagy léptekkel halad előre, majd később, ha az egyedek többsége már igen hasonlít egymásra, kisebb lépések jönnek.

Végül az (e)-ben a füzér minden elemét valamilyen kis független valószínűséggel **mutáció**nak (**mutation**) vetjük alá. Az első, a harmadik és az ötödik utódban egy-egy számjegyet mutáltunk. A 8-királynő problémában ez annak felel meg, hogy egy királynőt véletlen módon kiválasztunk, és egy, az oszlopban szintén véletlen módon kiválasztott mezőre áthelyezünk. Az ezeket a lépéseket implementáló algoritmust a 4.17. ábra mutatja.

4.17. ábra - Egy genetikus algoritmus. Ez az algoritmus ugyanaz, mint amelyet a 4.15. ábrán mutattunk, egy kivétellel. Ebben a jobban elterjedt változatban a két szülő reprodukciója nem két, hanem csak egy utódot hoz létre.

```
function GENETIKUS-ALGORITMUS(populáció, FITNESS-FV) returns egy egyed
    inputs: populáció, az egyedek halmaza
            FITNESS-FV, egy egyed fitness-értékét mérő függvény
    repeat
      új_populáció ← üres halmaz
      loop for i from 1 to MÉRET(populáció) do
            x ← VÉLETLEN-KIVÁLASZTÁS(populáció, FITNESS-FV)
            y ← VÉLETLEN-KIVÁLASZTÁS(populáció, FITNESS-FV)
            utód ← REPRODUKCIÓ (x, y)
            if (kis véletlen valószínűség) then utód ← MUTÁCIÓ(utód)
            utód hozzáadása az új_populáció-hoz
      populáció ← új_populáció
    until valamelyik egyednek kellően magas a fitness-értéke, vagy már lejárt az idő
    return a populáció legjobb egyede, a FITNESS-FV-nek megfelelően

function REPRODUKCIÓ(x, y) returns egy egyed
    inputs: x, y a szülőegyedek

    n ← HOSSZ(x)
    c ← 1 és n közötti véletlen szám
    return HOZZÁFŰZ(RÉSZFÜZÉR(x, 1, c), RÉSZFÜZÉR(y, c + 1, n))
```

A sztochasztikus nyaláb kereséshez hasonlóan a genetikus algoritmus is kombinálja a hegymászó tendenciát, a véletlen feltárást és a parallel keresési szálak közötti információcserét. A genetikus algoritmus elsődleges előnye, ha ilyen egyáltalán létezik, a keresztezéstől származik. Matematikailag azonban kimutatható, hogy ha kezdetben a genetikus kód pozícióit véletlen sorrendben permutáljuk, a keresztezés semmilyen előnyt nem jelent. Intuitíve, az előny a keresztezés azon képességéből jön, hogy a keresztezés képes hasznos funkciókat teljesítő nagy betűblokkok kombinálására, melyek egymástól függetlenül alakultak ki, emelve ezzel a keresésnél a granuláltság szintjét. Az első három királynőt például a 2, 4 és 6 pozícióba helyezve (ahol nem támadják egymást), egy hasznos blokk alakul ki, amit más blokkokkal kombinálva egy megoldás nyerhető.

Ennek működését a genetikus algoritmusok elmélete a **séma** (**schema**) fogalommal magyarázza. A séma egy olyan részfüzér, amelyben bizonyos pozíciók nem specifikáltak. Így például a 246***** séma az összes olyan 8-királynő állapotot leírja, ahol az első három királynő a 2., 4. és 6. pozícióban van. A sémára illeszkedő füzérek, mint például a 24613578 füzér, a séma **példány**ai (**instance**s). Kimutatható, hogy ha egy séma példányainak átlagos fitness-értéke az átlag felett van, akkor idővel a populációban a sémához tartozó példányok száma nőni fog. Világos, hogy ez a hatás elenyésző lesz, ha a szomszédos biteknek semmi közük egymáshoz, mert kevés olyan tömör blokk lesz, ami konzisztens módon előnyt fog jelenteni. A genetikus algoritmus a legjobban akkor működik, ha a sémák a megoldás értelmes komponenseinek felelnek meg. Ha például a füzér egy antenna reprezentációja, akkor a sémák képviselhetik az antenna egyes komponenseit, például a reflektorokat és a deflektorokat. Egy jó komponensnek nagy valószínűséggel különböző konstrukciók

Created by XMLmind XSL-FO Converter.

sokaságában is jónak kell lennie. Ez azt sugallja, hogy a genetikus algoritmusok sikeres alkalmazása a reprezentáció gondos kialakítását igényli.

A gyakorlatban a genetikus algoritmusoknak nagy hatása volt az olyan optimalizációs problémákra, mint például az áramkör elrendezéstervezés és a gyártósor-ütemezés. Jelenleg nem világos, hogy a genetikus algoritmusok vonzereje a hatékonyságuk vagy az evolúció elméletében rejlő esztétikus eredetük eredménye. Sok munka van még hátra, hogy azokat a körülményeket azonosítsuk, amikor a genetikus algoritmusok igazán jól teljesítenek.

4. Lokális keresés folytonos terekben

A 2. fejezetben bemutattuk a diszkrét és a folytonos környezet közötti különbséget, arra is rámutatva, hogy a valósvilág-beli környezetek többsége folytonos. Az eddig leírt algoritmusok közül azonban egyik sem képes a folytonos állapottereket kezelni. Az állapotátmenet-függvény az esetek többségében végtelen számú állapottal térne vissza! Ez a részfejezet egy *nagyon rövid* bevezetőt ad néhány olyan lokális keresési technikához, melyek folytonos térben keresik az optimális megoldást. A téma irodalma óriási. Sok alaptechnika már a 17. században napvilágot látott, Newton és Leibniz kalkulusának kifejlődését[13] követően. E technikákhoz a könyvben több helyen fogunk folyamodni, beleértve a tanulásról, a látásról és a robotikáról szóló fejezeteket. Röviden, mindenhol, ahol a valós világgal foglalkozunk.

Fontos

Evolúció és keresés

Az **evolúció** elméletét Charles Darwin az *On the Origin of Species by Means of Natural Selection* c. művében (Darwin, 1859) dolgozta ki. A központi gondolat igen egyszerű: a reprodukcióban (**mutáció**ként ismert) eltérések fordulnak elő, és azokat a következő generációk nagyjából a reprodukciós fitnessre gyakorolt hatásuk arányában megtartják.

A darwini elméletet annak ismerete nélkül dolgozták ki, hogy a szervezetek tulajdonságai hogyan öröklődnek és módosulnak. E folyamatokat irányító valószínűségi törvényeket első ízben Gregor Mendel szerzetes azonosította (Mendel, 1866), aki borsóval kísérletezett, saját szavaival mesterséges megtermékenyítést alkalmazva. Sokkal később Watson és Crick feltárták a DNS-molekula szerkezetét és ábécéjét – AGTC (adenin, guanin, timin, citozin) – (Watson és Crick, 1953). A standard modellben a betűszekvenciában változás pontmutáció és „keresztezés” révén áll be (ahol az utód DNS-e a szülői DNS-ek hosszú részleteinek kombinálásával jön létre).

A lokális keresési algoritmus analógiájáról írtunk már. A sztochasztikus nyaláb keresés és az evolúció közötti alapvető különbség a *szexuális* reprodukció használata, ahol az utódokat *több* egyedből hozzuk létre, és nem csak egyből. Az evolúció tényleges mechanizmusai azonban sokkal gazdagabbak, mint amit a genetikus algoritmus lehetővé tenne. A mutációhoz például az átfordítás, a duplikálás, a DNS nagy szegmenseinek a mozgatása is hozzátartozik. Egyes vírusok a DNS-t az egyik szervezetből veszik és egy másik szervezetbe beillesztik. Vannak átvihető gének is, amelyek nem tesznek mást, csak egy génállományon belül többezerszer lemásolják magukat. Olyan gének is vannak, amelyek a potenciális reprodukciós partnereknél ezeket a géneket nem tartalmazó sejteket megmérgezik, növelve így a sokszorozódás esélyét. A legfontosabb tény, hogy *maguk a gének tartalmazzák annak a mechanizmusnak a kódját,* amely által a génállomány reprodukálódik, és egy szervezetté alakul át. A genetikus algoritmusokban ezek a mechanizmusok különálló programok részei, vagyis a manipulált füzérekben nem jelennek meg.

A darwini evolúció igencsak kis hatékonyságú mechanizmusnak tűnhet, hiszen vakon létrehozott kb. 10^{45} szervezetet, anélkül hogy a keresési heurisztikáin egy csöppet is javított volna. Darwin előtt 50 évvel a különben neves francia természettudós Jean Lamarck egy olyan evolúció elméletet javasolt (Lamarck, 1809), ahol egy szervezet az *élete során, adaptációja révén megszerzett tulajdonságokat* is képes az utódoknak átadni. Az ilyen mechanizmus hatékony lenne, de úgy tűnik, a természetben nem fordul elő. Sokkal később James Baldwin látszólag hasonló elmélettel állt elő (Baldwin, 1896), hogy a szervezet élete alatt megtanult viselkedés növelhetné az evolúció sebességét. Lamarck elméletével ellentétben a Baldwin-elmélet a darwini evolúcióval teljesen konzisztens, hiszen alapja egy szelekciós

[13] A többváltozós kalkulus és a vektoraritmetika ismerete hasznos ennek a részfejezetnek az olvasásához.

Created by XMLmind XSL-FO Converter.

nyomás, amely olyan egyedekre hat, amelyek lokális optimumokat találtak a genetikus felépítésük által engedélyezett viselkedések között. A korszerű számítógépes szimulációk alátámasztják a „Baldwin-effektus" valós voltát, feltéve, hogy a „közönséges" evolúció olyan szervezeteket képes kialakítani, amelyeknél a belső hatékonysági mérték a saját tényleges fitness-értékükkel valahogy korrelál.

Kezdjük egy egyszerű példával. Tegyük fel, hogy valahol Romániában három új repülőteret szeretnénk létesíteni úgy, hogy a városoknak (lásd 3.2. ábra) a hozzájuk legközelebb eső repülőtérhez való távolságösszegük legyen minimális. Az állapotteret ekkor a repülőterek (x_1, y_1), (x_2, y_2) és (x_3, y_3) koordinátái definiálják. Ez egy *hatdimenziós* tér, azt is mondhatjuk, hogy az állapotokat hat **változó** (**variable**) definiálja (általánosságban az állapotokat a változók n-dimenziós **x** vektora definiálja). Ebben a térben való mozgás a térképen a repülőterek áthelyezésének felel meg. Az $f(x_1, y_1, x_2, y_2, x_3, y_3)$ célfüggvényt, ha a legközelebbi városok már megvannak, bármely állapot esetére viszonylag könnyű kiszámítani, általánosságban azonban igen nehéz ezt leírni.

A folytonosság problémái egyszerűen elkerülhetők, ha az egyes állapotok szomszédságát diszkretizáljuk. Egyszerre például csak egy repülőteret, csak x vagy y irányban egy rögzített $\pm\delta$ lépéssel lehet áthelyezni. Ez 6 változó mellett, minden állapotban 12 követőt eredményez. Ehhez az előbb megismert bármelyik lokális keresési algoritmust alkalmazhatjuk. A sztochasztikus hegymászó keresést és a szimulált lehűtést közvetlenül, a tér diszkretizálása nélkül is alkalmazhatjuk. Ezek az algoritmusok a követőket véletlen módon választják ki, amelyek megvalósítása δ hosszúságú véletlen vektorok generálásával lehetséges.

Sok módszer található, melyek megkísérlik a felszín **gradiens**ét (**gradient**) felhasználni a maximum megtalálásához. A célfüggvény gradiense egy ∇f vektor, amely a legmeredekebb emelkedő nagyságát és irányát adja meg. Problémánk esetén:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

Egyes esetekben a maximumot a $\nabla f = 0$ egyenlet megoldásával találhatjuk meg (ezt megtehetnénk például akkor, ha csak egy repülőteret helyeznénk el; a megoldás az összes város koordinátáinak számtani közepe lesz). Sok esetben azonban ezt az egyenletet zárt alakban nem lehet megoldani. Három repülőtér esetén például a gradiens kifejezése attól függ, hogy az adott állapotban mely városok esnek legközelebb az egyes repülőterekhez. Ez azt jelenti, hogy a gradienst *lokálisan,* és nem *globálisan* tudjuk számítani. Ennek ellenére még mindig folyamodhatunk a legmeredekebb emelkedő hegymászáshoz a pillanatnyi állapotot az

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla(\mathbf{x})$$

képlettel frissítve, ahol α egy kis konstans. Más esetekben nem biztos, hogy a célfüggvény differenciálható formában áll a rendelkezésünkre – például a repülőterek helyeit egy konkrét esetben esetleg egy nagyméretű gazdasági szimulációs szoftvercsomag futtatásával határozhatjuk meg. Az ilyen esetekben az ún. **empirikus gradiens** (**empirical gradient**) meghatározásához folyamodhatunk, minden koordináta mentén egy kis pozitív és negatív változáshoz kiszámítva a választ. Az empirikus gradiens keresés ugyanaz, mint a legmeredekebb emelkedő hegymászás az állapottér diszkretizált változatában.

Az „α egy kis konstans" kifejezés mögött az α-t beállító módszerek óriási választéka rejlik. Az alapprobléma az, hogyha α túl kicsi, túlságosan sok lépésre van szükség. Ha α túl nagy, a keresés könnyűszerrel túllő a maximumon. A **vonalkeresés** (**line search**) módszere e problémát úgy kísérli megoldani, hogy a pillanatnyi gradiens irányát – általában α ismételt megduplázásával – meghosszabbítja, amíg f nem kezd újra csökkeni. Az a pont, ahol ez megtörténik, lesz az új aktuális állapot.

Sok probléma esetén a legjobb algoritmus a jó öreg **Newton–Raphson-módszer** (Newton, 1671; Raphson, 1690). Ez a gyökhelykeresés, azaz a $g(x) = 0$ alakú egyenletek megoldásának általános módszere. Működésének alapja az x gyök új becslésének számítása a Newton-formula szerint:

$$x \leftarrow x - g(x)/g'(x)$$

f maximumának vagy minimumának megkereséséhez olyan **x**-et kell megkeresni, amire a *gradiens* nulla (azaz $\nabla f(\mathbf{x}) = 0$). A Newton-formula $g(x)$-e így $\nabla f(\mathbf{x})$ lesz, és a frissítési egyenlete az alábbi mátrixos formában

$$\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x}) \nabla f(\mathbf{x})$$

Created by XMLmind XSL-FO Converter.

írható fel, ahol $\mathbf{H}_f(\mathbf{x})$ a második deriváltak mátrixa (**Hesse-mátrix**), melynek H_{ij} elemeit a $\partial^2 f / \partial x_i \partial x_j$ második deriváltak adják meg. Mivel a második deriváltak mátrixának n^2 eleme van, a Newton–Raphson-módszer sokdimenziós terekben drága lesz, ami számos közelítő módszer kifejlesztéséhez vezetett.

A lokális keresési módszerek számára a lokális maximumok, a gerincek és a fennsíkok folytonos terekben ugyanúgy gondot jelentenek, mint diszkrét terekben. A véletlen újraindítás és a szimulált lehűtés alkalmazható (itt is), és gyakran segít. A sokdimenziós folytonos terek azonban terebélyes helyek, ahol könnyű elveszni.

Az utolsó téma, amivel érdemes futó ismeretséget kötni a **korlátozott optimalizálás** (**constrained optimization**). Egy optimalizálási probléma korlátozott, ha a megoldásnak minden változójának értékeire nézve valamilyen kemény korlátozást kell teljesítenie. A repülőteres problémánkban korlátozhatjuk például a helyszíneket, hogy a repülőterek Romániában belül és szárazföldön (nem tavak közepén) helyezkedjenek el. A korlátozott optimalizálás nehézségei a korlátozások és a célfüggvény természetén múlnak. A legismertebb kategóriát a **lineáris programozás**i (**linear programming**) problémák jelentik, ahol a korlátozások lineáris egyenlőtlenségek, amelyek egy *konvex* régiót képeznek, és ahol a célfüggvény szintén lineáris. A lineáris programozási problémákat változó számban polinomiális időben meg lehet oldani. Olyan problémákat is tanulmányoztak, ahol más típusú korlátozások és célfüggvények fordulnak elő. Ilyen problémák például a kvadratikus programozási feladat, a másodrendű kónikus programozási feladat stb.

5. Online kereső ágensek és ismeretlen környezetek

Eddig olyan ágensekre összpontosítottunk, amelyek **offline keresési** (**offline search**) algoritmusokat használnak. Egy teljes megoldást számítanak ki mielőtt a valós világba beteszik a lábukat (lásd 3.1. ábra), majd a megoldást az érzékelések megvizsgálása nélkül végrehajtják. Ezzel ellentétben az **online kereső** (**online search**)[14] ágensek működésében a számítás és a végrehajtás **átlapolódik** (**interleaving**): először végrehajtanak egy cselekvést, majd megfigyelik a környezetüket és kiszámítják a következő cselekvést. Az online keresés jó ötlet a dinamikus és a szemidinamikus környezetekben – olyan környezetekben, ahol büntetik a semmittevést és a túlságosan hosszú számításokat. Az online keresés még jobb ötlet sztochasztikus környezetekben. Általánosságban egy offline keresésnek egy exponenciálisan nagy, minden lehetséges történést figyelembe vevő eshetőségi tervvel kellene előállnia, míg az online keresés csak az aktuálisan megtörténteket veszi figyelembe. Például ajánlatos, ha egy sakkozó ágens előbb megteszi az első lépést és csak ezután számítja ki a játszma teljes lefolyását.

Az online keresés szükségszerű ötlet a **felfedezési problémá**k (**exploration problem**s) esetén, ahol az állapotok és a cselekvések ismeretlenek az ágens számára. Egy ilyen tudatlan állapotban az ágensnek a cselekvéseit kísérletekként kell használnia, hogy megállapíthassa, mit tegyen a következő pillanatban, így a számítás és a cselekvés szükségszerűen átlapolódik.

Az online keresés kanonikus példája egy új épületben elhelyezett robot, melynek fel kell tárnia a környezetét, hogy felépíthesse azt a térképet, amit majd arra használ, hogy *A*-tól *B*-ig eljusson. A labirintusból való menekülési módszerek – az ókori hősök elengedhetetlen képessége – szintén példái az online keresésnek. A térbeli feltárás azonban nem az egyedüli formája a feltárásnak. Gondoljunk egy újszülöttre: sok lehetséges cselekvéssel rendelkezik, de egyiknek sem tudja a hatását, és csak néhány közvetlenül elérhető állapotról van tudomása. Az, ahogy egy újszülött fokozatosan felfedezi, hogy hogyan működik a világ, részben egy online keresési folyamat.

5.1. Online keresési problémák

Egy online keresési problémát egy tisztán csak számítást végző folyamat helyett csak egy olyan ágens tud megoldani, amely a cselekvéseit végrehajtja. Feltételezzük, hogy az ágens csak az alábbiakat tudja:

- `CSELEKVÉSEK` (*s*), amely az *s* állapotban engedélyezett cselekvések listáját adja vissza,
- A lépésköltség $c(s, a, s')$ függvény – jegyezzük meg, hogy ez nem használható addig, amíg az ágens nem tudja, hogy s' az eredmény, és
- `CÉL-TESZT` (*s*).

[14] Az „online” kifejezést sűrűn használják a számítógépes tudományokban olyan folyamatok megjelölésére, amelyek a bemeneti adatokat azok bejövetelekor dolgozzák fel, és nem várnak addig, amíg a teljes bemeneti adathalmaz hozzáférhetővé válik.

Created by XMLmind XSL-FO Converter.

Vegyük észre azt is, hogy az ágens *nem képes* egy állapot követőit másképpen elérni, mint úgy, hogy az adott állapotban az összes cselekvését kipróbálja. A 4.18. ábrán látható labirintusproblémában az ágens nem tudja, hogy a *Fel* cselekvés (1,1)-ből elvezeti őt az (1,2)-be, valamint ennek végeztével, hogy a *Le* cselekvés visszaviszi őt az (1,1)-be. Bizonyos alkalmazásokban a tudatlanság ezen szintje mérsékelhető – egy felfedező robot ismerheti a mozgási cselekvéseinek mechanizmusát, és csak az akadályok hollétéről nincs tudomása.

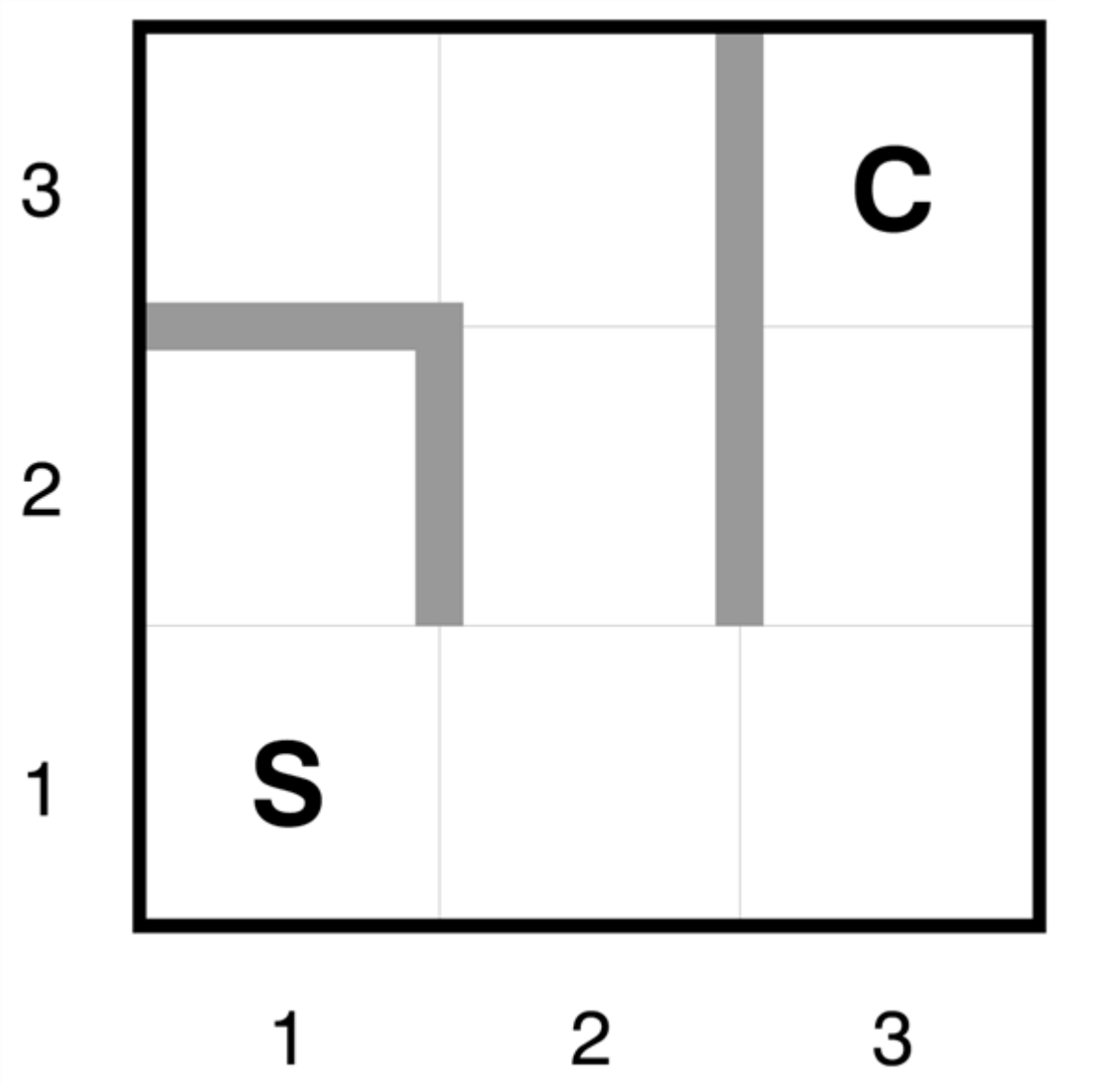
Feltételezzük, hogy az ágens mindig képes felismerni azt az állapotot, amiben már járt, és hogy a cselekvései determinisztikusak (e két feltételezéstől a 17. fejezetben eltekintünk). Végül az ágens hozzáférhet egy $h(s)$ elfogadható heuriszitikus függvényhez, amely a pillanatnyi állapot és a célállapot távolságát becsüli. A 4.18. ábrán például az ágens tudhatja a célpozícióját, és képes lehet a Manhattan-távolsági heurisztika használatára.

Az ágens célja tipikusan az, hogy elérjen egy célállapotot, és eközben minimalizálja a költségeket (egy másik lehetséges cél egyszerűen az egész környezet feltárása). A költség az ágens által megtett tényleges út teljes költsége. Szokásos ezt a költséget azzal az útköltséggel összehasonlítani, amit az ágens követne, *ha az egész keresési teret előre ismerné* – azaz az aktuális legrövidebb úttal (vagy a legrövidebb teljes feltárással). Az online algoritmusok nyelvén ezt **kompetitív arány**nak (**competitive ratio**) nevezzük, és azt szeretnénk, ha ez a lehető legkisebb lenne.

Bár ez ésszerű követelménynek tűnik, könnyű belátni, hogy egyes esetekben a legjobb elérhető kompetitív arány a végtelen. Ha például bizonyos cselekvések irreverzíbilisek, az online keresés esetleg egy olyan zsákutcába kerülhet, ahonnan a célállapot nem elérhető.

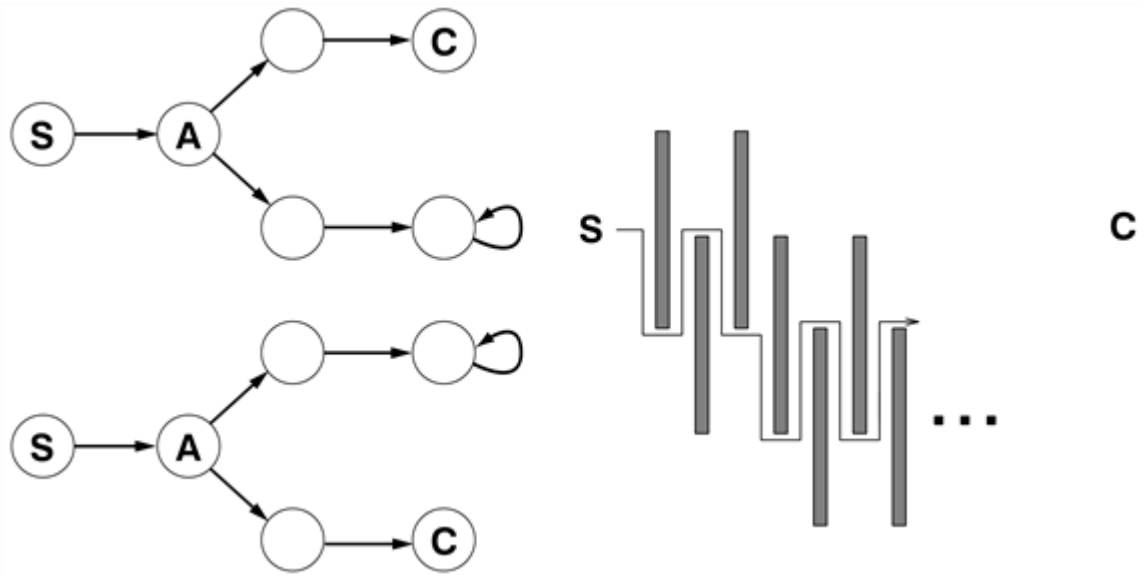
4.18. ábra - Egy egyszerű labirintusprobléma. Az ágens *S*-nél kezd és *C*-t kell elérnie, a környezetéről azonban semmit sem tud.

Created by XMLmind XSL-FO Converter.



4.19. ábra - (a) Két olyan állapottér, amely az online kereső ágenst egy zsákutcába viheti. (b) Egy kétdimenziós környezet, amely arra késztetheti az online kereső ágenst, hogy a célhoz legkevésbé hatékony utat kövesse. Akármit is választ az ágens, az ellenség eltorlaszolja az útját egy másik hosszú, vékony fallal úgy, hogy a követett út sokkal hosszabb lesz, mint a legjobb lehetséges út.

Created by XMLmind XSL-FO Converter.



Fontos

Lehet, hogy az „esetleg” megfogalmazást nem találja kellően meggyőzőnek – végül is létezhetne olyan algoritmus, amely feltárás közben a zsákutcákban nem köt ki. A kijelentésünk pontosabban megfogalmazva az, hogy *nincs olyan algoritmus, amely bármilyen állapottérben el tudná kerülni a zsákutcákat*. Nézzük meg közelebbről a 4.19. (a) ábra két zsákutcás állapotterét. Egy olyan online kereső algoritmus számára, amely az *S* és az *A* állapotokat már meglátogatta, a két állapottér *azonosnak* tűnik, tehát mindkettőben ugyanolyan döntéshez kell folyamodnia. Ez az **ellenség érv** (**adversary argument**) egy példája – el tudunk képzelni egy ellenséget, amely módosítja az állapotteret, miközben az ágens feltárja azt, és a célokat és a zsákutcákat tetszés szerint átrendezi.

A zsákutcák a robotfeltárás igazi nehézségei – lépcsőházak, felhajtók, szakadékok és a természetes terep minden fajtája irreverzíbilis cselekvésekhez vezethetnek. Hogy előrehaladhassunk, egyszerűen feltételezzük, hogy az állapottér **biztonságosan feltárható** (**safely explorable**) – bizonyos célállapotok az összes elérhető állapotból elérhetők. A visszafordítható cselekvéseket tartalmazó állapottereket, mint amilyenek a labirintusok és a kirakójátékok, irányítatlan gráfoknak lehet tekinteni, és nyilván biztonságosan feltárhatók.

Még a biztonságosan feltárható környezetekben sem biztosítható korlátozott kompetitív arány, ha léteznek benne korlátlan költségű utak. Ezt könnyű kimutatni irreverzíbilis cselekvéseket tartalmazó környezetekben, de igaz marad visszafordítható cselekvések esetén is, ahogy ezt a 4.19. (b) ábra mutatja. Emiatt általános, hogy az online algoritmusok hatékonyságát nem a legsekélyebben fekvő célállapot mélységével, hanem az egész állapottérrel jellemezzük.

5.2. Online kereső ágensek

Egy online kereső ágens minden cselekvés után érzékeli, hogy milyen állapotba került. Ebből az információból felépítheti környezetének térképét. Az aktuális térkép alapján eldönti, hogy legközelebb merre menjen. Ez az átlapolódó tervkészítés és végrehajtás azt jelenti, hogy az online kereső algoritmusok egészen mások, mint az előbb megismert offline algoritmusok. Az olyan offline algoritmusok, mint például az A* képesek egy csomópont követőit kiszámítani a tér egy részében, majd azonnal egy másik csomóponttal foglalkozni a tér egy másik részében, mert a csomópontkifejtés inkább szimulált, mint valódi cselekvéseket takar. Egy online algoritmus ezzel szemben csak azt a csomópontot fejtheti ki, amelyben fizikailag tartózkodik. Hogy a következő csomópont kifejtése érdekében a fa keresztül-kasul való végignézését elkerüljük, célszerűbbnek tűnik a csomópontokat *lokális* sorrendben kifejteni. Ezzel a tulajdonsággal éppen a mélységi keresés rendelkezik, mert (a visszalépést kivéve) a következő kifejtendő csomópont az előbb kifejtett csomópont gyereke.

4.20. ábra - Egy mélységi feltárást használó online kereső ágens. Az ágens csak a kétirányú keresési terekben alkalmazható.

Created by XMLmind XSL-FO Converter.

```
function ONLINE-MÉLYSÉGI-ÁGENS(s') returns egy cselekvés
    inputs: s', egy érzékelés, amely az aktuális állapotot azonosítja
    static: eredmény, cselekvésekkel és állapotokkal indexelt táblázat, kezdetben üres
            nemfeltárt, egy táblázat, amely minden meglátogatott állapotra listázza
              a még nem próbált cselekvéseket
            nemvisszalépett, egy táblázat, amely minden meglátogatott állapotra listázza
              a még nem próbált visszalépéseket
            s, a, az előbbi állapot és cselekvés, kezdetben nulla

    if CÉL-TESZT(s') then return stop
    if s' egy új állapot then nemfeltárt[s'] ← CSELEKVÉSEK(s')
    if s nem nulla then do
            eredmény[a, s] ← s'
            s hozzáadása a nemvisszalépett[s'] elejéhez
    if nemfeltárt[s'] üres then
    if nemvisszalépett[s'] üres then return stop
    else a ← egy olyan b cselekvés, hogy eredmény[b, s'] = POP(nemvisszalépett[s'])
    else a ← POP(nemfeltárt[s'])
    s ← s'
    return a
```

Egy online mélységi kereső ágenst a 4.20. ábra mutat. Ez az ágens a térképét egy *eredmény*[*a*, *s*] táblázatban tárolja, amely az *s* állapotban végrehajtott *a* cselekvés hatására előálló állapotot tartalmazza. Ha az aktuális állapotban marad még ki nem használt cselekvés, az ágens ezzel próbálkozik. A nehézség akkor áll be, ha az ágens egy állapotban már minden cselekvést kipróbált. Offline mélységi keresésnél ezt az állapotot a sorból egyszerűen elhagyjuk. Online keresésnél az ágensnek fizikailag kell visszalépnie. A mélységi keresésnél ez azt jelenti, hogy abba az állapotba kell visszakerülnie, amelyből az ágens legutóbb belépett a vizsgált állapotba. Ezt egy olyan táblázat karbantartásával lehet megoldani, amely minden állapot számára azokat az ősállapotokat listázza, melyekhez az ágens még nem lépett vissza. A keresés teljes, ha az ágens a viszszaléphető állapotokból kifut.

Azt javasoljuk, hogy az olvasó kövesse végig az ONLINE-MÉLYSÉGI-ÁGENS előrehaladását a 4.18. ábrán mutatott labirintus esetén. Könnyű észrevenni, hogy az ágens a térben minden csatlakozást legrosszabb esetben pontosan kétszer fog végigpásztázni. Feltárás esetében ez optimális eredmény. Egy cél megtalálása szempontjából azonban az ágens kompetitív aránya tetszőlegesen rossz lehet, ha egy hosszú túrára indul, holott a cél a kezdeti állapot közvetlen szomszédságában fekszik. Ezt a problémát az iteratívan mélyülő algoritmus online változata oldja meg. Egy homogén fát eredményező környezetben egy ilyen ágens kompetitív aránya egy kis konstans.

Az alkalmazott visszalépési módszer miatt az ONLINE-MÉLYSÉGI-ÁGENS csak olyan terekben működik, ahol minden cselekvés visszafordítható. Az általános állapotterekhez egy kicsit bonyolultabb algoritmusok léteznek, azonban ezek egyike sem garantál korlátozott kompetitív arányt.

5.3. Online lokális keresés

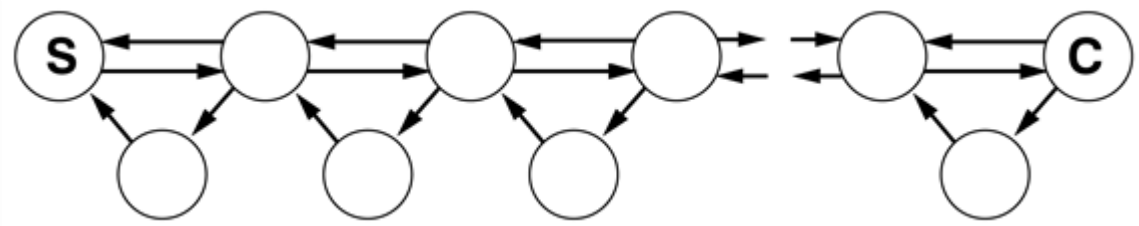
A mélységi kereséshez hasonlóan a **hegymászó keresés**re (**hill-climbing search**) jellemző a csomópontkifejtés lokalitása. A tény az, hogy mivel a hegymászó keresés a memóriában csak egy aktuális állapotot tart, ez az algoritmus *már* egy online keresési algoritmus! A legegyszerűbb formájában azonban nemigen használatos, mert az ágenst a lokális maximumokban bennrekedve hagyja, ahonnan nincs hová mennie. Ráadásul a véletlen újraindítást sem lehet használni, mert az ágens nem képes magát egy új állapotba áthelyezni.

A véletlen újraindítások helyett, a környezet feltárására a **véletlen vándorlás**t (**random walk**) használhatjuk. A véletlen vándorlás egyszerűen véletlen módon választ egy lehetséges cselekvést az aktuális állapotban. Elsőbbséget élvezhetnek az eddig még nem kipróbált cselekvések. Könnyű bebizonyítani, hogy a véletlen

Created by XMLmind XSL-FO Converter.

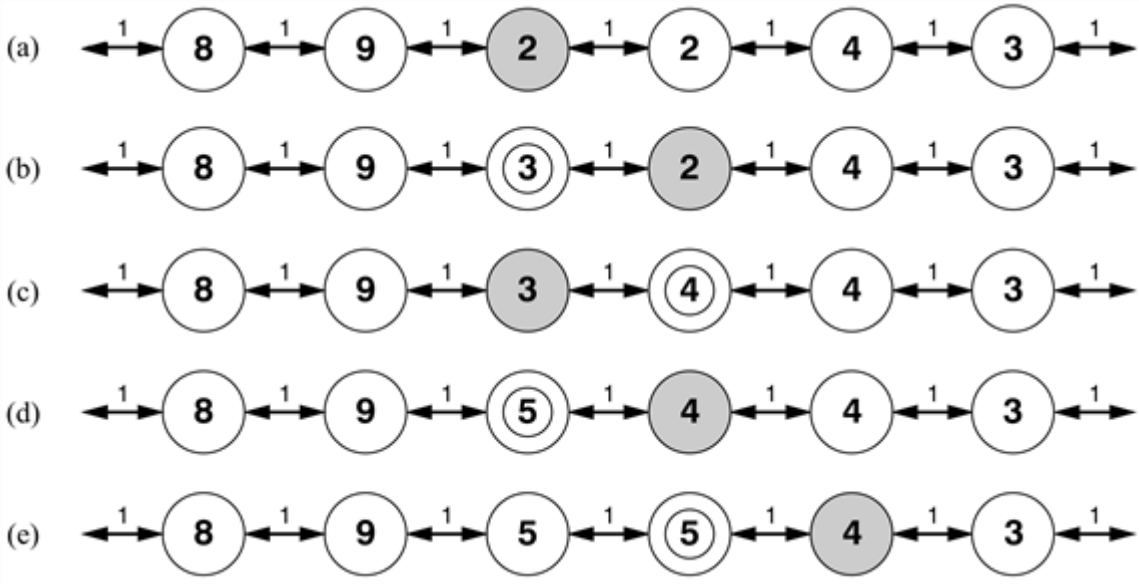
vándorlás *valamikor* megtalálja a célt, vagy feltárja a környezetet, feltéve, hogy a tér véges.[15] Másfelől a folyamat igen lassú lehet. A 4.21. ábra egy olyan környezetet mutat, ahol a véletlen vándorlásnak exponenciálisan sok lépésre van szüksége a cél megtalálásához, mert minden lépésnél a visszafelé haladás kétszer olyan valószínű, mint az előrehaladás. A példa persze kitalált, azonban sok olyan valós állapottér létezik, melynek topológiája ilyenfajta „csapdát” állít a véletlen vándorlás elé.

4.21. ábra - Egy környezet, amelyben a véletlen vándorlásnak exponenciálisan sok lépésre lesz szüksége, hogy a célt megtalálja



Sokkal hatékonyabb megközelítés, ha a hegymászást nem véletlen működéssel, hanem *memóriával* látjuk el. Az alapötlet az, hogy a cél minden meglátogatott állapotból való elérési költségének egy $H(s)$ „aktuális legjobb becslését” eltároljuk. $H(s)$ a $h(s)$ heurisztikus becslésből indul, és ahogy az ágens egyre több tapasztalatot gyűjt az állapottérben, értéke folyamatosan frissül. A 4.22. ábra egy egyszerű példát mutat egydimenziós állapottérre. Az (a)-ban úgy tűnik, az ágens egy lapos lokális minimumban, az árnyalt állapotban bennragadt. Ahelyett hogy ott maradna, ahol volt, az ágensnek követnie kellene azt az utat, ami a szomszédos állapotok aktuális költségbecslései alapján a célhoz vezető legjobb útnak néz ki. Annak becsült költsége, hogy a célt az s' szomszédon keresztül érjük el az s' elérésének költsége, valamint a cél elérésének becsült költsége az s' állapotból, azaz $c(s, a, s') + H(s')$. A példában két cselekvés van, 1 + 9 és 1 + 2 becsült költségekkel, legcélszerűbbnek tűnik tehát jobbra menni. Most világos, hogy az árnyékolt állapot költségének becslésére a kettes érték túl optimista. Mivel a legjobb cselekvés költsége 1, és ez elvezet a céltól legalább kétlépésnyire lévő állapothoz, az árnyalt állapot legalább háromlépésnyire van a céltól. A H függvényt így ennek megfelelően frissíteni kell, ahogy a 4.22. (b) ábra mutatja. Ezt a folyamatot folytatva az ágens még kétszer mozdul el előre-hátra, a H-t frissítve és a lokális minimumot „laposítva”, amíg jobbra nem tud elmenekülni.

4.22. ábra - A TRTA* öt iterációja egydimenziós állapottérben. Minden állapotot $H(s)$-sel, a cél elérésének aktuális becslésével, és minden élt a lépésköltséggel címkéztünk. Az árnyalt állapot az ágens helyét jelöli meg, és a frissített értékeket minden iterációban bekarikáztuk.



[15] A végtelen eset sokkal trükkösebb. A véletlen vándorlás teljes egy- és kétdimenziós végtelen rácsban, de nem a háromdimenziósban! Ebben az utóbbi esetben annak a valószínűsége, hogy a vándorlás valamikorra a kiindulópontba visszatér, csak 0,3405 körüli. Egy általános bevezető: (Hughes, 1995).

Created by XMLmind XSL-FO Converter.

Ezt a sémát, aminek **tanuló valós idejű A*** (**TRTA***) a neve, a 4.23. ábrán látható ágens implementálja. Ez az ágens az ONLINE-MÉLYSÉGI-ÁGENS-hez hasonlóan az *eredmény* táblázat felhasználásával környezetének egy térképét építi meg. Felfrissíti az éppen elhagyott állapot költségbecslését, és az aktuális költségbecslés szerint a „látszólag legjobb" cselekvést választja. Fontos részlet, hogy az *s* állapotban még ki nem próbált cselekvésekről mindig azt tételezi fel, hogy a lehető legkisebb $h(s)$ költséggel közvetlenül a célhoz vezetnek. Ez a **bizonytalanság melletti optimizmus** (**optimism under uncertainty**) felbátorítja az ágenst, hogy új, feltehetően gyümölcsöző utakat tárjon fel.

Egy TRTA* ágens garantáltan megtalálja a célt akármilyen véges, biztonságosan feltárható környezetben. Az A*-gal ellentétben azonban nem teljes a végtelen állapotterekben – vannak olyan esetek, amikor az ágenst reménytelenül félre lehet vezetni. Egy *n* állapotból álló környezetet legrosszabb esetben $O(n^2)$ lépésben tár fel, de sokszor jóval hatékonyabb ennél. A TRTA* ágens a cselekvéskiválasztási és a frissítési szabályokkal különböző módon definiálható online ágensek nagy családjának egy partikuláris tagja. Ezzel a családdal, melyet eredetileg sztochasztikus környezetekhez találtak ki, a 21. fejezetben foglalkozunk.

5.4. Tanulás online keresés során

Az online kereső ágens kezdeti tudatlansága több lehetőséget teremt a tanulásra. Először az ágens a környezetének „térképét" – pontosabban minden állapotban minden cselekvés kimenetelét – tanulja meg egyszerűen regisztrálva a tapasztalatát. (Jegyezzük meg, hogy a determinisztikus környezet feltételezése azt jelenti, hogy egy-egy cselekvés esetén egy tapasztalás elég lesz.) Másodszor a lokálisan kereső ágensek lokális frissítési szabályok segítségével pontosabb értékbecslésekre tesznek szert minden állapot esetén, ahogy ez a **TRTA***-nál történt. Látni fogjuk a 21. fejezetben, hogy ezek a frissítések előbb-utóbb minden állapot *egzakt* értékéhez fognak konvergálni, feltéve, hogy az ágens az állapotteret megfelelő módon tárja fel. Amikor az egzakt értékek már ismertek, az optimális döntések egyszerűen a legmagasabb értékű utódállapotba való átmenetekkel hozhatók meg – ilyenkor tehát a tiszta hegymászás az optimális stratégia.

4.23. ábra - A TRTA* ágens a cselekvéseit a környező állapotok értékei alapján választja ki, mely az állapotokat az állapottérben mozogva folyamatosan frissíti

```
function TRTA*-ÁGENS(s′) returns egy cselekvés
    input: s′, egy érzékelés, amely az aktuális állapotot azonosítja
    static: eredmény, cselekvésekkel és állapotokkal indexelt táblázat, kezdetben üres
            H, költségbecslések táblázata állapotokkal indexelve
            s, a, az előbbi állapot és cselekvés, kezdetben nulla

    if CÉL-TESZT(s′) then return stop
    if s′ egy új állapot (nincs H-ban) then H[s′] ← h(s′)
    unless s nulla
        eredmény[a, s] ← s′
        H[s] ← min_{b∈CSELEKVÉSEK(s)} TRTA*-KÖLTSÉG(s, b, eredmény[b, s], H)
    a ← egy olyan b cselekvés a CSELEKVÉSEK(s′)-ben, ami TRTA*-KÖLTSÉG(s′, b, eredmény[b, s′], H)-t
        minimalizálja
    s ← s′
    return a

function TRTA*-KÖLTSÉG(s, a, s′, H) returns egy költségbecslés
    if s′ nem definiált then return h(s)
    else return c(s, a, s′) + H[s′]
```

Ha elfogadta a javaslatunkat, hogy az ONLINE-MÉLYSÉGI-ÁGENS viselkedését kövesse nyomon a 4.18. ábrán bemutatott környezetben, észreveheti, hogy az ágens nem valami lángész. Így például, miután már látta, hogy a *Fel* cselekvés az (1,1)-ből az (1,2)-be visz, még mindig fogalma sincs, hogy a *Le* cselekvés az (1, 1)-be visz vissza, vagy hogy a *Fel* cselekvés a (2,1)-ből a (2,2)-be, a (2,2)-ből a (2,3)-ba stb. visz. Általánosságban azt szeretnénk, ha az ágensünk megtanulná, hogy a *Fel* növeli az *y* koordinátát, hacsak egy fal nincs útban, a *Le* a koordinátát csökkenti stb. Hogy ez megtörténhessen, két dologra van szükség. Először az ilyen általános

Created by XMLmind XSL-FO Converter.

szabályokhoz szükségünk van egy formális és manipulálható reprezentációra. Az információt egyelőre az állapotátmenet-függvény nevű fekete dobozba rejtettük. Ezzel a témával a III. rész foglalkozik. Másodszor szükségünk van egy algoritmusra, amely képes a megfelelő szabályokat az ágens által megtett konkrét megfigyelésekből konstruálni. Ezzel pedig a 18. fejezet foglalkozik.

6. Összefoglalás

Ebben a fejezetben áttekintettük, hogyan lehet a keresési költséget **heurisztikus függvény**ek használatával csökkenteni. Megvizsgáltunk számos heurisztikus függvényt alkalmazó algoritmust, és láttuk, hogy az optimalitásért a keresési költség viszonylatában drága árat kell fizetnünk, még akkor is, ha sikerült jó heurisztikus függvényt találnunk.

- A **legjobbat-először keresés** (**best-first search**) egyszerűen egy olyan `GRÁF-KERESÉS`, ahol először (valamilyen mérték szerint) a legkisebb költségű, még ki nem fejtett csomópontokat fejtjük ki. A legjobbat-először algoritmusok tipikusan egy $h(n)$ **heurisztikus** (**heuristics**) függvényt használnak, amely a cél költségét becsüli az n állapotból kiindulva.

- A **mohó legjobbat-először keresés** (**greedy best-first search**) a minimális $h(n)$ értékű csomópontokat fejti ki. Nem optimális, azonban sokszor hatékony.

- Az **A* keresési algoritmus** (**A* search**) a minimális $f(n) = g(n) + h(n)$ értékű csomópontokat fejti ki. Az A* algoritmus teljes és optimális, feltéve, hogy garantálni tudjuk, hogy $h(n)$ elfogadható (a `FA-KERESÉS` számára) vagy konzisztens (a `GRÁF-KERESÉS` számára). Az A* algoritmus tárkomplexitása még mindig elfogadhatatlan.

- A heurisztikus algoritmusok hatékonysága a heurisztikus függvény minőségétől függ. Jó heurisztikus függvények készíthetők néha például a probléma definíciójának relaxálásával, a mintaadatbázis részproblémáihoz tartozó megoldási költségek előzetes kiszámításával vagy a problémaosztályon belül a tapasztalatból való tanulással.

- Az **RLEK** (**RBFS**) és az **EMA*** (**SMA***) robusztus, optimális keresési algoritmusok, amelyek korlátozott mennyiségű memóriát használnak. Ha elegendő idő áll rendelkezésre, olyan problémákat is megoldanak, melyeket az A* algoritmus nem képes megoldani, mert elfogy a memóriája.

- A *lokális keresési módszerek*, mint például a **hegymászó keresés** (**hill climbing**) teljes állapotleírásokkal dolgoznak, de a memóriában csak csekély számú csomópontot tartanak. Több sztochasztikus algoritmust is kifejlesztettek, beleértve a **szimulált lehűtés**t (**simulated annealing**), amely megfelelő hűtési karakterisztika esetén optimális megoldással tér vissza. A folytonos térbeli problémákra számos lokális keresési algoritmus is használható.

- A **genetikus algoritmus** (**genetic algorithm**) egy olyan sztochasztikus hegymászó keresés, ahol az állapotok nagy populációjával dolgozunk. Új állapotokat **mutáció**val (**mutation**) és a populációbeli állapotpárokat összekombináló **keresztezés**sel (**crossover**) generálunk.

- **Felfedezési problémá**król (**exploration problem**s) akkor van szó, ha az ágensnek fogalma sincs környezete állapotairól és cselekvéseiről. Biztonságosan feltárható környezetek esetén az **online kereső** (**online search**) ágens felépítheti a környezetek térképét, és megtalálja a célt, ha az létezik. A heurisztikus becslések tapasztalat alapján történő frissítése hatékony módszer, az ágens lokális minimumokból való kimeneküléséhez.

6.1. Irodalmi és történeti megjegyzések

A heurisztikus információ alkalmazása a problémamegoldásban Simon és Newell egy korai írásában jelenik meg (Simon és Newell, 1958). A „heurisztikus keresés” frázis és a célhoz való távolságot becsülő heurisztikus függvény használata azonban valamivel későbbről származik (Newell és Ernst, 1965; Lin, 1965). Doran és Michie részletesen, kísérleti alapon tanulmányozták a heurisztikus keresési algoritmusok alkalmazását számos problémára, nagy hangsúlyt fektetve a 8-as és a 15-ös kirakójátékra (Doran és Michie, 1966). Habár Doran és Michie a heurisztikus keresésnél elméleti úthossz és „behatolás” (az úthossz és az eddig vizsgált csomópontok számának aránya) elemzéseket végzett, úgy tűnik, figyelmen kívül hagyták az aktuális úthossz nyújtotta információt. A heurisztikus keresésbe az aktuális úthosszt is beszámító A* algoritmust Hart, Nilsson és Raphael

Created by XMLmind XSL-FO Converter.

dolgozták ki (Hart és társai, 1968), néhány későbbi korrekcióval (Hart és társai, 1972). Az A* algoritmus optimális hatékonyságát Dechter és Pearl (Dechter és Pearl, 1985) mutatták ki.

Az eredeti A* algoritmust ismertető cikk bevezette a heurisztikus függvények konzisztenciájának feltételét. A heurisztikák monotonitási feltételét a konzisztenciafeltételnek egy egyszerűbb feltétellel való kiváltására Pohl vezette be (Pohl, 1977), de Pearl megmutatta, hogy a két feltétel ekvivalens (Pearl, 1984). Az A*-ot megelőző számos algoritmus használta a nyitott és a zárt listával ekvivalens fogalmakat. Ezek közé tartozik a szélességi, a mélységi és az egyenletes költségű keresés (Bellman, 1957; Dijkstra, 1959). Különösképpen Bellman munkája mutatta meg az additív útköltség fontosságát az optimalizálási problémák egyszerűsítésében.

Pohl elsőként tanulmányozta az A* algoritmus heurisztikahibája és időigénye közötti összefüggést (Pohl, 1970; 1977). Annak bizonyítása, hogy az A* algoritmus lineáris időben fut, ha a heurisztikus függvény hibája egy állandó korlát alatt van, Pohl és Gaschnig cikkében található (Pohl, 1977; Gaschnig, 1979). Pearl ennek az eredménynek egy erősebb megfogalmazását adta meg, amely megengedte a hiba logaritmikus növekedését (Pearl, 1984). A heurisztikus keresés hatékonyságát jellemző „effektív elágazási tényező" mértéket Nilsson javasolta (Nilsson, 1971).

Az A* algoritmusnak rengeteg változata létezik. Pohl javasolta a dinamikus súlyozási technikát, amelyben kiértékelő függvényként az A*-ban használatos egyszerű $f(n) = g(n) + h(n)$ helyett az aktuális úthossz és a heurisztikus függvény alábbi súlyozott összege szerepel: $f_w(n) = w_g g(n) + w_h h(n)$ (Pohl, 1973). Ez a módszer a keresés előrehaladtával dinamikusan állítja a w_g és w_h súlyokat. Kimutatható, hogy a Pohl-féle algoritmus ε-*elfogadható* – azaz garantálja a megoldás megtalálását az optimális megoldáshoz képest 1 + ε tényezőn belül. Itt ε az algoritmus egy bemeneti paramétere. Ugyanilyen tulajdonságú a A^*_ε algoritmus (Pearl, 1984), amely a perem minden olyan csomópontját kiválaszthatja, amely a perembeli minimális *f*-költségű csomóponttól *f*-költségben (1 + ε)-os tényezőn belül van. A kiválasztást úgy lehet lebonyolítani, hogy ez a keresési költségeket minimalizálja.

Az A* keresési algoritmus és más állapottérben kereső algoritmusok nagyban hasonlítanak az operációkutatás területén elterjedten alkalmazott *elágazik-és-korlátoz (branch-and-bound)* technikákhoz (Lawler és Wood, 1966). Az állapottérbeli keresés és az elágazik-és-korlátoz algoritmus közötti kapcsolatot részletesen vizsgálták Dana Nau, Laveen Kanal és Vipin Kumar (Kumar és Kanal, 1983; Nau és társai, 1984; Kumar és társai, 1988). Martelli és Montanari megmutatták a kapcsolatot a dinamikus programozás (lásd 17. fejezet) és bizonyos típusú állapottérbeli keresések között (Martelli és Montanari, 1978). Kumar és Kanal CDP – „összetett döntési folyamat" – néven megpróbálta „egységes alapokra hozni" a heurisztikus keresést, a dinamikus programozást és az elágazik-és-korlátoz technikákat (Kumar és Kanal, 1988).

Mivel az 1950-es évek végén, az 1960-as évek elején a számítógépek még csak legfeljebb pár száz szónyi memóriával rendelkeztek, így a memóriakorlátozott keresés már a kezdeti időkben is intenzíven kutatott terület volt. Doran és Michie Graph Traverser programja (Doran és Michie, 1966) az egyik legkorábbi keresőprogram, a memóriakorlát által megengedett mértékben legjobbat-először keresést hajt végre, majd kiválaszt egy operátort. Az IMA* algoritmus (Korf, 1985a, 1985b) volt az első széles körben alkalmazott optimális, memóriakorlátozott heurisztikus keresési algoritmus; ennek számos változatát ki is dolgozták. Az IMA* algoritmus hatékonyságának részletes elemzése és az algoritmus valós értékű heurisztikákkal kapcsolatos nehézségeinek tárgyalása a (Patrick és társai, 1992)-ben jelent meg.

Az RLEK (Korf, 1991, 1993) valójában egy kicsit bonyolultabb, mint a 4.5. ábrán bemutatott algoritmus, amely közelebb áll a függetlenül kifejlesztett ún. **iteratívan kifejtő** (**iterative expansion**) vagy **IK** (**IE**) (Russell, 1992) algoritmushoz. Az RLEK felső és alsó korlátot használ. A két algoritmus viselkedése azonos elfogadható heuriszitikák esetén. Az RLEK azonban a csomópontokat a legjobbat-először sorrendben akkor is kifejti, ha a heurisztika nem elfogadható. A legjobb alternatív út számontartásának gondolata korábban jelent meg Bratkónál (Bratko, 1986) az A* elegáns Prolog implementációjában és a DTA* algoritmusban (Russell és Wefald, 1991). Ez utóbbi a metaállapotterekkel és a metaszintű tanulással is foglalkozik.

Az MA* algoritmus először Chakrabartinál jelent meg (Chakrabarti és társai, 1989). Az EMA* vagy Egyszerűsített MA* az MA* implementációs kísérleteiből született meg, mint az IK egy összehasonlító algoritmusa (Russell, 1992). Kaindl és Khorsand az EMA* felhasználásával megalkottak egy kétirányú keresési algoritmust, amely jelentősen gyorsabb, mint a korábbi algoritmusok (Kaindl és Khorsand, 1994). Korf és Zhang az oszd-meg-és-uralkodj megközelítést írják le (Korf és Zhang, 2000). Zhou és Hansen pedig bevezették a memóriakorlátozott A* gráfkeresést (Zhou és Hansen, 2002). Korf áttekintést ad a memóriakorlátozott keresési technikákról (Korf, 1995).

Created by XMLmind XSL-FO Converter.

Held és Karp (Held és Karp, 1970) nagy hatású cikke a minimális-feszítőfa alkalmazását (lásd 4.8. feladat) taglalja az utazó ügynök problémára, megmutatva, hogy a relaxált probléma vizsgálatával hogyan lehet elfogadható heurisztikákra jutni.

Prieditis a Jack Mostow-val végzett korábbi kutatásokra építve (Mostow és Prieditis, 1989) sikeresen automatizálta a probléma relaxálási folyamatát (Prieditis, 1993). A mintaadatbázisok használatát az elfogadható heurisztikák előállítására Gasser, valamint Culberson és Schaeffer kezdeményezték (Gasser, 1995; Culberson és Schaeffer, 1998). A diszjunkt mintaadatbázisokat Korf és Felner írják le (Korf és Felner, 2002). A heurisztikák valószínűség-alapú értelmezését Pearl és Hansson és Mayer mélyrehatóan vizsgálták (Pearl, 1984; Hansson és Mayer, 1989).

A heurisztikus keresési algoritmusok messze legátfogóbb irodalma Pearl *Heuristics* c. könyve (Pearl, 1984). Ez a könyv különösen jó áttekintést nyújt számos oldalhajtásról és az A* különféle változatairól, és azok tulajdonságainak szigorú bizonyításait is tartalmazza. Kanal és Kumar elkészítették az alapvető és lényeges heurisztikus kereséssel foglalkozó cikkek antológiáját (Kanal és Kumar, 1988). A keresési algoritmusokkal kapcsolatos új eredmények rendszeresen az *Artificial Intelligence* folyóiratban jelennek meg.

A lokális keresési technikáknak a matematikában és a számítógépes tudományokban hosszú történetük van. Valóban, a Newton–Raphson-módszert (Newton, 1671; Raphson, 1690) egy igen hatékony lokális keresési módszernek lehet tekinteni folytonos terekben, ahol a gradiens információ rendelkezésre áll. Az ilyen információt nem igénylő optimalizáló algoritmusok klasszikus forrása Brent munkája (Brent, 1973). A nyalábkeresés, melyet lokális keresési algoritmusként mutattunk be, a beszédfelismerési célokra alkalmazott dinamikus programozás korlátozott szélességű változataként indult a HARPY rendszerben (Lowerre, 1976). A megfelelő algoritmust részletesen Pearl vizsgálta (Pearl, 1984, 5. fejezet).

A lokális keresés témaköre a nagy kényszerkielégítési problémák, mint például az *n*-királynő (Minton és társai, 1992) és a logikai következtetés (Selman és társai, 1992) területén elért meglepően jó eredményeknek és a véletlenség, a többszörös egyidejű keresés és más javítás beépítésének köszönhetően az utóbbi években megélénkült. Az ilyen algoritmusok reneszánsza, melyeket Christos Papadimitriou „New Age" algoritmusoknak nevez, felkeltette az elméleti számítógép-tudományokkal foglalkozó kutatók érdeklődését is (Koutsoupias és Papadimitriou, 1992; Aldous és Vazirani, 1994). Az operációkutatás területén népszerűségnek örvend a hegymászó keresés egy változata, amit **tabukeresés**i algoritmusnak (**tabu search**) hívnak (Glover, 1989; Glover és Laguna, 1997). Az emberi rövid távú memóriamodellre alapozva ez az algoritmus karbantartja az előbb meglátogatott *k* állapot tabulistáját, melyeket újra meglátogatni nem szabad. Ezáltal az algoritmus hatékonysága megnőtt a gráfkeresésben, és képes egyes lokális minimumokból kimenekülni. A hegymászó keresés másik hasznos javítása a STAGE algoritmus (Boyan és Moore, 1998). Az ötlet az, hogy használjuk a véletlen újraindítású hegymászó keresés révén nyert lokális maximumokat a tájfelszín általános alakjának felderítésére. Az algoritmus a lokális maximumokra sima felületet illeszt, és a globális maximumot analitikusan számítja ki. Ez lesz az új újraindítási pont. Az algoritmus a gyakorlatban működőképesnek bizonyult nehéz problémák esetén. Gomes (Gomes és társai, 1998) azt mutatta meg, hogy a futási idő eloszlása a szisztematikusan visszalépő algoritmusok esetén sokszor **lassan lecsengő eloszlás** (**heavy-tail distribution**), ami azt jelenti, hogy a nagyon hosszú futási idők valószínűsége nagyobb, mint amit a normális eloszlás alapján jósolni lehetne. Ez megadja a véletlen újraindítási mechanizmus elméleti igazolását.

A szimulált lehűtést először Kirkpatrick és szerzőtársai írták le (Kirkpatrick és társai, 1983), akik az algoritmust közvetlenül a statisztikus fizika területén összetett rendszerek szimulálására használt **Metropolis-algoritmus**tól kölcsönözték (Metropolis és társai, 1953). A Metropolis-algoritmust állítólag Los Alamosban egy estély során találták ki. A szimulált lehűtés mára már különálló kutatási részterületet alkot, ahol évente több száz cikket publikálnak.

Optimális megoldások keresése folytonos térben több terület alapfeladata, az **optimalizálás elmélet**et, az **optimális szabályozás elmélet**et és a **variációkalkulus**t (**optimization theory**, **optimal control theory**, **calculus of variations**) beleértve. Ezekről alkalmas (és gyakorlati) áttekintést Press (Press és társai, 2002) és Bishop (Bishop, 1995) adnak. A **lineáris programozás** (**LP**, **linear programming**) a számítógépek egyik első alkalmazása volt. A **szimplex algoritmus**t (**simplex-algorithm**) (Wood és Dantzig, 1949; Dantzig, 1949) még mindig alkalmazzák annak ellenére, hogy a legrosszabb esetben exponenciális komplexitású. Az LP egy gyakorlati polinomiális idejű algoritmusát Karmarkar (Karmarkar, 1984) dolgozta ki.

A genetikus algoritmusok fejlődésének fontos előfutára volt Sewall Wright munkája a **fitness tájfelszín** (**fitness landscape**) fogalmáról. Néhány statisztikus, Boxot (Box, 1957) és Friedmant (Friedman, 1959) is beleértve, az '50-es években használt már evolúciós technikákat optimalizálási problémákra, e megközelítés azonban csak

Created by XMLmind XSL-FO Converter.

akkor kezdett népszerűvé válni, amikor Rechenberg (Rechenberg, 1965; 1973) az **evolúciós stratégiá**kat (**evolution strategies**) a szárnyprofilok optimalizálási problémáinak megoldásába bevezette. Az 1960-as és 1970-es években John Holland (Holland, 1975) kiállt a genetikus algoritmusok mellett, azt tartva, hogy a genetikus algoritmusok hasznos eszköznek bizonyulnak és segítenek abban, hogy jobban megértsük a biológiai és egyéb adaptációt (Holland, 1995). A **mesterséges élet** (**artificial life**) mozgalom (Langton, 1995) ezt a gondolatot egy lépéssel tovább viszi, a genetikus algoritmusokkal létrehozott eredményeket nem problémamegoldások, hanem inkább *szervezeteknek* tekintve. Hinton és Nowlan (Hinton és Nowlan, 1987), valamint Ackley és Littman (Ackley és Littman, 1991) munkája ezen a területen sokban hozzá járult, hogy a Baldwin-hatás következményeit feltárjuk. Smith és Szathmáry (Smith és Szathmáry, 1990) munkáját mint az evolúció általános hátterét bemutató munkát ajánljuk.

A legtöbb összehasonlítás a genetikus algoritmusok és más megközelítések (különösképpen a hegymászó keresés) között azt találta, hogy a genetikus algoritmusok lassabban konvergálnak (O'Reilly és Oppacher, 1994; Mitchell és társai, 1996; Juels és Wattenberg, 1996; Baluja, 1997). Az ilyen eredmények a GA közösségben nem nagyon népszerűek, azonban e közösségen belül a legutóbbi olyan kísérletek, hogy a populációalapú keresést a Bayes-tanulás egy formájaként értelmezzük (lásd 20. fejezet) talán segít, hogy e terület és a kritikusai közötti szakadék eltűnjön (Pelikan és társai, 1999). A GA hatékonyságára magyarázatot adhat a **kvadratikus dinamikus rendszer**ek (**quadratic dynamical system**s) elmélete (Rabani és társai, 1998) is, lásd például Lohn és társainak (Lohn és társai, 2001) a GA antennatervezésre alkalmazott példáját és Larrañaga (Larrañaga és társai, 1999) munkáját, ahol a GA-t az utazó ügynök problémára alkalmazták.

A **genetikus programozás** (**genetic programming**) a genetikus algoritmusokkal közeli rokonságban van. Az elsődleges különbség az, hogy a mutált és az összekombinált reprezentációk nem bitfüzérek, hanem programok. A programokat kifejezésfák alakjában reprezentáljuk. A kifejezéseket valamilyen standard nyelvből meríthetjük, mint például a Lispből, de lehetnek egy olyan nyelv elemei, melyet kifejezetten áramkörök, robotszabályozók stb. reprezentálására terveztünk. A keresztezés a részfák és nem a részfüzérek összeillesztését jelenti. A mutáció ezen formája garantálja, hogy az utódok jól definiált kifejezések lesznek, ami nem biztos, hogy sikerülne, ha a programok manipulációját füzérszinten oldanánk meg.

A genetikus programozást övező jelenlegi érdeklődést John Koza (Koza, 1992) munkája váltotta ki, a módszer azonban visszavezethető Friedbergnek (Friedberg, 1958) a gépi kóddal folytatott korai kísérleteihez és Fogelnek (Fogel és társai, 1996) a véges állapotú automatákkal folytatott vizsgálataihoz. Mint a genetikus algoritmusok esetén itt is vita folyik a módszer hatékonyságát illetően. Koza (Koza és társai, 1999) sok olyan kísérletet ír le, amelyeket genetikus programozás felhasználásával az automatizált áramkörtervezés területén végzett.

A genetikus algoritmusokról és a genetikus programozásról az *Evolutionary Computation* és az *IEEE Transactions on Evolutionary Computation* folyóiratok írnak. Idevágó cikkeket még a *Complex Systems*, az *Adaptive Behavior* és az *Artificial Life* folyóiratokban is találunk. A legfontosabb konferenciák az *International Conference on Genetic Algorithms* és a *Conference on Genetic Programming*, amelyek a közelmúltban *Genetic and Evolutionary Computational Conference* néven fuzionáltak. A terület jó áttekintését adják Melanie Mitchell és David Fogel írásai (Mitchell, 1996; Fogel, 2000).

Az ismeretlen állapottereket feltáró algoritmusokkal évszázadok óta foglalkoztak. Egy labirintusban való mélységi keresés implementálható például úgy, hogy bal kezünket folyamatosan a falon tartjuk. A hurkokat elkerülhetjük, ha a keresztezéseket megjelöljük. A mélységi keresés kudarcba fullad, ha a cselekvések nem visszafordíthatók. Az **Euler-gráf**ok (**Eulerian graph**s) (azaz olyan gráfok, ahol minden csomópontban ugyanannyi él megy be, mint ki) feltárásának általánosabb problémáját Hierholzer (Hierholzer, 1873) algoritmusa oldotta meg. A tetszőleges gráfok feltárásának első részletes algoritmikus vizsgálatát Deng és Papadimitriou (Deng és Papadimitriou, 1990) végezték, akik egy teljesen általános algoritmushoz jutattak el, azonban kimutatták, hogy az általános gráf feltárására nem adható korlátos kompetitív arány. Papadimitriou és Yannakakis (Papadimitriou és Yannakakis, 1991) azt a kérdést vizsgálták, hogy hogyan található meg a célhoz vezető út geometriai úttervező környezetben (ahol minden cselekvés visszafordítható). Kimutatták, hogy négyzet alakú akadályok esetén kis kompetitív arány érhető el, az általános téglalap alakú akadályok esetén azonban korlátos arány nem érhető el (lásd 4.19. ábra).

A TRTA* algoritmust, a **valós idejű keresés** (**real-time search**) vizsgálatának részeként olyan környezetekben, ahol az ágensnek cselekednie kell egy rögzített idejű keresést követően (a kétjátékos játékokban egy megszokottabb helyzet) Korf (Korf, 1990) fejlesztette ki. A TRTA* valójában a megerősítéses tanulási algoritmus egy speciális esete sztochasztikus környezetekben (Barto és társai, 1995). Az optimizmus bizonytalanság melletti stratégiája – mindig a legközelebbi még nem meglátogatott állapotok felé tartani – olyan feltárási mintát eredményezhet, amely a nem informált esetben kevésbé hatékony, mint az egyszerű mélységi

Created by XMLmind XSL-FO Converter.

keresés (Koenig, 2000). Dasgupta (Dasgupta és társai, 1994) kimutatta, hogy egy kiegyensúlyozott fában a cél megkeresésére, amikor heurisztikus információ nincs, az online iteratívan mélyülő keresés optimálisan hatékony. A TRTA* számos informált változatát is kifejlesztették, különböző módszereket alkalmazva, a gráf már ismert részein belüli keresésre és frissítésre (Pemberton és Korf, 1992). Egyelőre még nem sikerült megérteni, hogy heurisztikus információ alkalmazása esetén hogyan kell a célokat optimális hatékonysággal megkeresni.

A fejezet azért nem foglalkozott a **párhuzamos keresési** (**parallel search**) algoritmusokkal, mert ezek tárgyalásához a párhuzamos architektúrák terjedelmes ismertetésére lett volna szükség. A párhuzamos keresés egyre fontosabb témakörré válik mind az MI, mind pedig az elméleti számítógép-tudomány területén. Egy az MI idevágó irodalmába történő rövid bevezető Mahanti és Daniels cikkében (Mahanti és Daniels, 1993) található.

6.2. Feladatok

4.1.

Kövesse végig az A* működését a Lugasból Bukarestbe utazás problémájára alkalmazva, légvonalban mért távolság heurisztikát alkalmazva. Azaz adja meg az algoritmus által megvizsgált csomópontok szekvenciáját az *f*-, *g*- és *h*-értékükkel együtt.

4.2.

A **heurisztikus útalgoritmus** (**heuristic path algorithm**) egy olyan legjobbat-először keresés, ahol a célfüggvény $f(n) = (2 - w)g(n) + wh(n)$. Milyen w értékekre garantálható az algoritmus optimalitása? (Feltételezheti, hogy h elfogadható.) Milyen keresésről van szó, amikor $w = 0$? Amikor $w = 1$? És amikor $w = 2$?

4.3.

Bizonyítsa be az alábbi állításokat:

a. A szélességi keresés az egyenletes költségű keresés egy speciális esete.

b. A szélességi keresés, a mélységi keresés és az egyenletes költségű keresés a legjobbat-először keresés speciális esetei.

c. Az egyenletes költségű keresés az A* keresés egy speciális esete.

4.4.

Tervezzen meg egy olyan állapotteret, ahol az A* a `GRÁF-KERESÉS` felhasználásával egy szuboptimális megoldással tér vissza egy elfogadható, de nem konzisztens $h(n)$ függvény mellett.

4.5.

A 1.1. szakasz - A mohó legjobbat-először keresés részben láttuk, hogy a légvonalban mért távolságheurisztika a mohó legjobbat-először keresést félrevezeti, amikor Iaşiról Fogarasra szerettünk volna eljutni. Az ellenkező problémára azonban, vagyis amikor Fogarasról Iaşira szeretnénk eljutni, a heurisztika tökéletes. Léteznek olyan problémák, amelyek esetében a heurisztika mindkét irányban félrevezet bennünket?

4.6.

Találjon ki a 8-as kirakójátékra olyan heurisztikus függvényt, ami néha túlbecsül, és mutassa meg, hogy egy konkrét probléma esetén ez hogyan vezethet szuboptimális megoldáshoz (használhat számítógépet, ha ez segít). Bizonyítsa be, hogy ha h soha nem becsül túl c-nél jobban, az A* algoritmus, h-t felhasználva, olyan megoldással tér vissza, melynek költsége az optimális megoldás költségét legfeljebb c-vel haladja meg.

4.7.

Bizonyítsa be, hogy ha a heurisztikus függvény konzisztens, akkor elfogadható. Szerkesszen egy elfogadható heurisztikus függvényt, amely nem konzisztens.

4.8.

Created by XMLmind XSL-FO Converter.

Az utazó ügynök problémát (TSP) meg lehet oldani a minimális feszítő fa (MFF) heurisztika alkalmazásával, ami feltételezve, hogy már egy részútvonalat megterveztünk, megbecsüli az útvonal befejezésének költségét. A városok egy halmazának MFF-költsége az összes várost összekötő fák minimális élköltségösszege.

a. Mutassa meg, hogy ez a heurisztikus függvény hogyan származtatható a TSP relaxált változatából!

b. Mutassa meg, hogy az MFF-heurisztika dominálja a légvonalban mért távolságheurisztikát!

c. Írjon egy problémagenerátort a TSP-problémára, amely olyan problémaegyedeket generál, amelyben a városokat az egységnégyzeten belüli véletlen módon generált pontok reprezentálják!

d. Keressen hatékony módszert az irodalomban az MFF megszerkesztésére, és használja azt egy elfogadható keresési algoritmussal a TSP-probléma egyedeinek a megoldására!

4.9.

A 2.1. szakasz - A heurisztikus függvény pontosságának hatása a megoldás hatékonyságára részben a 8-as kirakójáték olyan relaxációját definiáltuk, ahol a lapka az *A* mezőről a *B* mezőre mozdulhat el, ha *B* üres. E probléma egzakt megoldása az ún. **Gaschnig-heurisztikát** (**Gaschnig's heuristics**) definiálja (Gaschnig, 1979). Magyarázza meg, hogy miért lesz a Gaschnig-heurisztika legalább olyan pontos, mint h_1 (a rossz helyen lévő lapkák száma), és mutasson olyan eseteket, amikor mind h_1-nél mind h_2-nél pontosabb (Manhattan-távolság). Tud ajánlani a Gaschnig-heurisztikára egy hatékony számítási módszert?

4.10.

A 8-as kirakójátékra adtunk két egyszerű heurisztikus függvényt: a Manhattan-távolságot és a rossz helyen lévő lapkák számát. Az irodalomban számos heurisztikáról azt állítják, hogy azok mindkét előbbinél jobbak. Lásd például Nilsson (Nilsson, 1971), Mostow (Mostow és Prieditis, 1989) és Hansson (Hansson és társai, 1992) cikkét. Ellenőrizze ezeket az állításokat a heurisztikák implementálásával és a futási eredmények összevetésével.

4.11.

Nevezze meg azokat az algoritmusokat, amelyek az alábbi speciális esetekből származnak:

a. Lokális nyaláb keresés $k = 1$ mellett.

b. Lokális nyaláb keresés egy kezdeti állapottal és korlátlan számú visszatartott állapottal.

c. Szimulált lehűtés, ha $T = 0$ minden időpillanatra (és kihagyva a leállási tesztet).

d. Genetikus algoritmus $N = 1$ nagyságú populációval.

4.12.

Néha egy adott problémához nem létezik jó kiértékelő függvény, azonban létezik jó összehasonlítási módszer: vagyis el lehet dönteni, hogy egy csomópont egy másik csomópontnál jobb-e anélkül, hogy valamilyen számértéket rendelnénk hozzájuk. Mutassa meg, hogy ez elégséges a legjobbat-először (best-first) keresés végrehajtásához. Létezik-e ilyen analógia az A* esetére is?

4.13.

Hozza kapcsolatba a TRTA* időkomplexitását az algoritmus tárkomplexitásával.

4.14.

Tegyük fel, hogy egy ágens a 4.18. ábrán látható 3 × 3 labirintus környezetben tartózkodik. Az ágens tudja, hogy a kezdeti pozíciója az (1,1), a cél a (3,3)-n van, és hogy a négy *Fel*, *Le*, *Balra*, *Jobbra* cselekvésnek a szokásos hatása van, hacsak egy fal nincs útban. Az ágens *nem* tudja a belső falak helyét. Minden egyes állapotban az ágens a legális cselekvések halmazát képes érzékelni. Azt is meg tudja mondani, hogy az állapotot meglátogatta-e már korábban, vagy pedig egy új állapotban van.

Created by XMLmind XSL-FO Converter.

a. Magyarázza meg, hogy ezt az online keresési problémát hogyan lehet offline keresésnek tekinteni egy olyan hiedelmi állapotérben, ahol a kezdeti hiedelmi állapot a környezet minden lehetséges konfigurációját tartalmazza. Milyen nagy a kezdeti hiedelmi állapot? Milyen nagy a hiedelmi állapottér?

b. Hányféle különböző megfigyelés lehetséges a kezdeti állapotban?

c. Írja le e probléma esetén az eshetőségi terv néhány első ágát. Milyen nagy (nagyjából) az egész terv?

Jegyezze meg, hogy ez az eshetőségi terv az adott leírást kielégítő *minden lehetséges környezet* számára megoldás. A keresés és a végrehajtás átlapolódása tehát egy ismeretlen környezetben nem szigorúan szükséges.

4.15.

Ebben a feladatban a 4.8. feladatban leírt típusú TSP-probléma megoldására lokális keresési módszerekkel kísérletezünk.

a. Tervezze meg a TSP megoldására a hegymászó keresést. Az eredményeket az MFF-heurisztikával (4.8. feladat) dolgozó A* algoritmus optimális megoldásaival hasonlítsa össze.

b. Tervezzen meg a TSP megoldására egy genetikus algoritmust. Az eredményeket más megközelítésekkel hasonlítsa össze. Esetleg szüksége lehet a (Larrañaga és társai, 1999) cikk egyes reprezentációs javaslataira.

4.16.

Generáljon sok 8-as kirakójáték- és 8-királynő feladat-példányt, és oldja meg ezeket (ha lehetséges) a hegymászó kereséssel (legmeredekebb emelkedő és az első-választás változatok), véletlen újraindítású hegymászó kereséssel és szimulált lehűtéssel. Mérje meg a megoldási költségeket, és százalékosan adja meg a megoldások arányát. Mindezeket ábrázolja egy grafikonon az optimális költség függvényében. Fűzzön kommentárt az eredményekhez.

4.17.

Ebben a feladatban a hegymászó keresést a robotnavigáció kontextusában fogjuk vizsgálni, példának használva a 3.22. ábrán látható környezetet.

a. Ismételten oldja meg a 3.16. feladatot a hegymászó keresés segítségével. Le fog-e ragadni az ágens valamikor egy lokális maximumban? *Lehetséges,* hogy leragadjon az ágens konvex akadályok esetén?

b. Hozzon létre olyan konkáv környezetet, amiben az ágens leragad!

c. Módosítsa úgy a hegymászó algoritmust, hogy az az 1 mélységű keresés helyett *k* mélységű keresést használjon annak eldöntésére, hogy a következő lépésben hová menjen. Meg kell keresnie a legjobb *k* lépéses utat, e mentén egy lépést megtenni, majd az eljárást megismételni.

d. Létezik *k*-nak olyan értéke, ami mellett az új algoritmus garantáltan elkerüli a lokális maximumokat?

e. Magyarázza meg, hogy ebben az esetben a TRTA* algoritmus hogyan teszi lehetővé az ágens számára, hogy a lokális minimumokból kimeneküljön.

4.18.

Hasonlítsa össze az A* és a RLEK keresés hatékonyságát véletlen módon generált problémahalmazokon a 8-as kirakójáték (Manhattan-távolsággal) és a TSP-(MFH-vel, lásd 4.8. feladat) problémakörben. Értelmezze az eredményeket. Mi történik az RLEK hatékonyságával, ha a 8-as kirakójáték problémakörben a heurisztika értékeihez egy kis véletlen számot adunk hozzá?

Created by XMLmind XSL-FO Converter.

5. fejezet - Kényszerkielégítési problémák

Ebben a fejezetben megmutatjuk azt, hogy ha az állapotokat nem egyszerűen kis fekete dobozoknak tekintjük, akkor új hatékony keresési módszerek egész sorához jutunk, valamint jobban megértjük a probléma struktúráját és komplexitását.

A 3. és a 4. fejezetben azt az elképzelést jártuk körbe, hogy a problémákat **állapottér**beli kereséssel lehet megoldani. A tárgyterületre jellemző heurisztikák segítségével értékelhetjük ezeket az állapotokat és ellenőrizhetjük, hogy nem járunk-e célállapotban. A keresési algoritmus szempontjából azonban mindegyik állapot egy megkülönböztetés nélküli belső struktúrájú **fekete doboz** (**black box**). Az állapotokat egy tetszőleges adatstruktúra reprezentálja, amelyhez *a problémára jellemző* rutinokkal lehet hozzáférni: az állapotátmenet- és a heurisztikafüggvénnyel, valamint a célállapotteszttel.

A fejezetben a **kényszerkielégítési problém**ákkal (**constraint satisfication problem**s) foglalkozunk, amikor az állapotok és a célteszt illeszkedik egy szabályos, strukturált és elég egyszerű **reprezentáció**hoz (lásd 5.1. alfejezet). Az állapotstruktúra segítségével keresési algoritmusokat lehet definiálni, és nem *problémaspecifikus,* hanem *általános célú* heurisztikák használatával nagy problémák megoldása is lehetővé válik (lásd 5.2. és 5.3. alfejezet). Talán a legfontosabb azonban az, hogy a célteszt szabályos reprezentációja feltárja magának a problémának a struktúráját (lásd 5.4. alfejezet). Ez pedig módszereket ad a kezünkbe a probléma dekompozíciójára, és lehetővé teszi, hogy megértsük a probléma struktúrája és a megoldás nehézsége közötti szoros kapcsolatot.

1. Kényszerkielégítési problémák

A **kényszerkielégítési problém**ák (angol rövidítéssel **CSP**) formális definícióját **változó**k (**variable**s), X_1, X_2, …, X_n, és **kényszer**ek (**constraint**s), C_1, C_2, …, C_m halmazaival adhatjuk meg. Minden egyes X_i változó esetén adott a lehetséges **érték**ek egy nem-üres D_i **tartománya**. Minden egyes C_i kényszer a változók valamely részhalmazára vonatkozik, és meghatározza a részhalmaz megengedett értékkombinációit. Egy problémaállapotot az definiál, hogy vagy néhány, vagy mindegyik változóhoz értékeket **rendelünk hozzá** $\{X_i = v_i, X_j = v_j, ...\}$. Egy hozzárendelést **konzisztens**nek (vagy megengedettnek) nevezünk, ha egyetlen kényszert sem sért meg. Teljes az a hozzárendelés, amelyben mindegyik változó szerepel, és egy teljes hozzárendelés a kényszerkielégítési problémának **megoldása**, ha mindegyik kényszert kielégíti. Néhány kényszerkielégítési probléma azt is igényli, hogy a megoldás egy **célfüggvény**t maximalizáljon.

Mit is jelent mindez? Tegyük fel, hogy Romániába beleunva Ausztrália térképét tanulmányozzuk, és a térképen az 5.1. (a) ábrán feltüntetett államokat és területeket látjuk, feladatunk pedig az, hogy vörössel, zölddel és kékkel minden egyes részt színezzünk ki úgy, hogy a szomszédos részeknek ne legyen azonos a színe. Ahhoz, hogy ezt kényszerkielégítési problémaként fogalmazhassuk meg, változókat kell bevezetnünk az egyes részekhez: *NyA*, *ÉT*, *Q*, *ÚDW*, *V*, *DA* és *T*. Mindegyik változó tartománya a {*vörös*, *zöld*, *kék*} halmaz. Kényszerek írják elő, hogy a szomszédos részek különböző színűek legyenek, például *NyA* és *ÉT* esetén a megengedhető kombinációk az alábbi párok lehetnek:

```
{(vörös, zöld), (vörös, kék), (zöld, vörös), (zöld, kék), (kék, vörös), (kék, zöld)}
```

(A kényszert tömörebben is ki lehet fejezni a $NyA \neq ÉT$ egyenlőtlenséggel, feltéve persze, hogy a kényszermegoldó algoritmus valamiképpen ki tudja értékelni az ilyen kifejezéseket.) Sok lehetséges megoldás adódik, mint például az

```
{NyA = vörös, ÉT = zöld, Q = vörös, ÚDW = zöld, V = vörös, DA = kék, T = vörös}
```

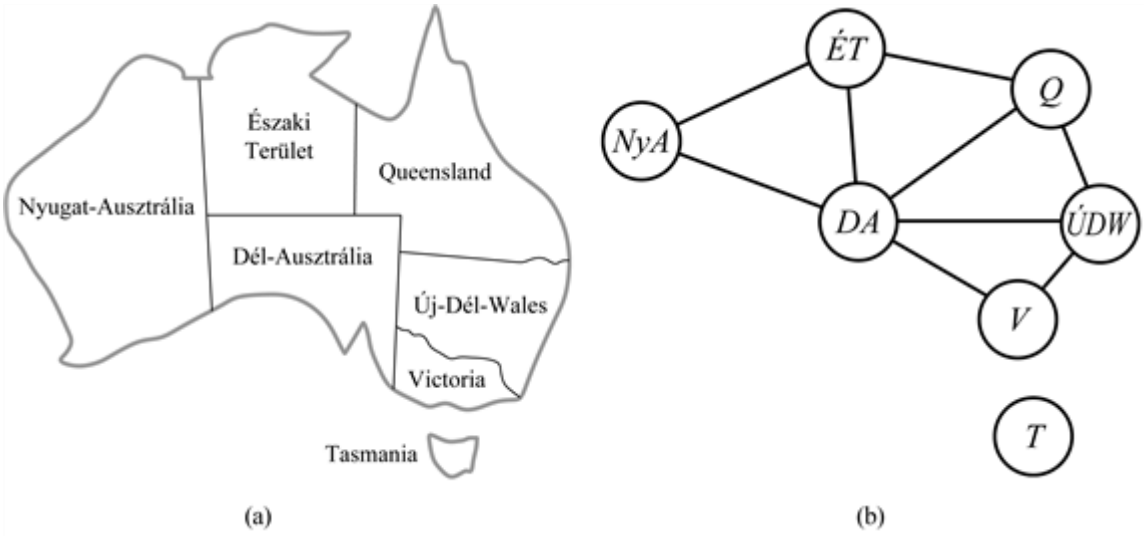
Gyakran hasznos lehet, ha felrajzoljuk a kényszerkielégítési probléma **kényszergráf**ját (**constraint graph**), amint az az 5.1. (b) ábrán is látható. A gráf csomópontjai a probléma változóinak, élei pedig a kényszereknek felelnek meg.

Sok előnnyel járhat, ha egy problémát kényszerkielégítési problémaként tekintünk. Az állapotok reprezentációja miatt a kényszerkielégítési problémák egy standard mintára illeszkednek (ez a minta a hozzárendelt értékekkel rendelkező változók halmaza), az állapotátmenet-függvényt és a célállapottesztet pedig az összes

Created by XMLmind XSL-FO Converter.

kényszerkielégítési problémára érvényes általános módon meg lehet írni. Sőt létrehozhatók hatékony, általános heurisztikák minden kiegészítő, tárgyterület-specifikus szakértelem nélkül. Végül pedig a kényszergráf struktúrájának segítségével lerövidíthető a megoldási folyamat, és ez néhány esetben exponenciálisan csökkenti a probléma komplexitását. A kényszerkielégítési probléma reprezentációja a legelső (és a legegyszerűbb) a könyv menete során bemutatott reprezentációs sémák sorában.

5.1. ábra - (a) Ausztrália államai és területei. A térkép kiszínezése tekinthető kényszerkielégítési problémának is. A cél az, hogy minden egyes részhez olyan színt találjunk, amely nem azonos egy szomszédos rész színével sem. (b) A térképszínezési probléma kényszergráfként reprezentálva.



Könnyű észrevenni, hogy egy kényszerkielégítési problémára adható egy **inkrementális megfogalmazás** (**incremental formulation**), amely a kényszerkielégítési problémát szabályos keresési problémaként tekinti:

- **Kiinduló állapot** (**initial state**): az üres hozzárendelés {}, ahol egyik változónak sincs értéke.
- **Állapotátmenet-függvény** (**successor function**): bármelyik hozzárendelés nélküli változó értéket kaphat, amennyiben ez nem ütközik a korábbi értékadásokkal.
- **Célteszt** (**goal test**): az aktuális hozzárendelés teljes.
- Az **út költsége** (**path cost**): egy konstans költség (például 1) mindegyik lépésre.

Fontos

Mindegyik megoldásnak egy teljes hozzárendelésnek kell lennie, tehát *n* változó esetén az *n*-edik mélységi szinten jelenik meg. A keresési fa pedig csak *n* mélységű. Emiatt a korlátkielégítési problémák esetén a mélységi keresési algoritmusok népszerűek (lásd 5.2. alfejezet). Az is igaz, hogy *a megoldáshoz vezető út nem lényeges.* Ezért használható a **teljes állapotleírás** (**complete-state formulation**) is, amelyben minden egyes állapot egy teljes változó-hozzárendelés, akár kielégíti a kényszereket, akár nem. Ebben a felírásban a lokális keresési eljárások is használhatók.

A kényszerkielégítési problémák legegyszerűbb esetében a változók **diszkrét**ek és **véges tartományú**ak. A térképszínezési problémák is ehhez az esethez tartoznak. A 3. fejezetben bemutatott 8-királynő problémát is felfoghatjuk véges tartományú kényszerkielégítési problémaként, ahol a Q_1, …, Q_8 változók a királynők 1, …, 8 oszlopokban betöltött pozícióit jelölik, és mindegyik változó tartománya {1, 2, 3, 4, 5, 6, 7, 8}. Amennyiben maximum d a kényszerkielégítési probléma bármely változójához tartozó tartomány mérete, akkor a lehetséges teljes hozzárendelések száma $O(d^n)$, azaz a változók számának exponenciális függvénye. A véges tartományú kényszerkielégítési problémák közé tartoznak a Boole-problémák is, ahol a változók értéke vagy *igaz*, vagy *hamis*. A **Boole kényszerkielégítési problémá**k speciális esetként tartalmaznak néhány NP-teljes problémát, például a 3SAT-ot (lásd 7. fejezet). Legrosszabb esetben tehát nem lehetséges, hogy exponenciálisnál kevesebb idő alatt meg tudunk oldani egy véges tartományú kényszerkielégítési problémát. A legtöbb gyakorlati

Created by XMLmind XSL-FO Converter.

alkalmazásban azonban az általános célú CSP algoritmusok megbirkóznak a 3. fejezetben bemutatott általános célú keresési algoritmusokkal megoldhatóknál *több nagyságrenddel nagyobb* problémákkal is.

A diszkrét változók lehetnek **végtelen tartományú**ak is: például ilyen az egész számok halmaza vagy a füzérek halmaza. Például amikor egy építkezési munka naptárát ütemezzük, minden feladat kezdési ideje egy változó, amelyeknek lehetséges értékei az aktuális dátum napjait megadó egész számok. Végtelen tartományok esetén már nem lehet a kényszereket a megengedett értékkombinációk felsorolásával leírni. Ehelyett egy **kényszernyelv**et (**constraint language**) kell használni. Például ha $Munka_1$ öt napig tart, és meg kell előznie $Munka_3$-t, akkor kényszernyelvként az algebrai egyenlőtlenségek nyelvére van szükségünk, mint például $KezdMunka_1 + 5 \leq KezdMunka_3$. Továbbá az ilyen kényszereket többé már nem lehet megoldani az összes lehetséges hozzárendelés felsorolásával, mert végtelen sok van belőlük. Különleges megoldó algoritmusok (melyekre itt nem térünk ki) léteznek egész értékű változók **lineáris kényszer**eire, azaz olyan kényszerekre, mint a fenti, ahol mindegyik változó csak lineáris műveletekben jelenik meg. Megmutatható, hogy nem létezhet algoritmus egész értékű változók általános **nemlineáris kényszer**einek megoldására. Néhány esetben az egész értékű kényszerproblémák egyszerűen azzal visszavezethetők véges tartományúakra, hogy korlátozzuk az összes változó értékeit. Egy ütemezési problémában például egy felső korlátot szabhatunk az összes ütemezendő munka teljes hosszára.

A **folytonos tartományú** kényszerkielégítési problémák elég gyakoriak a valós alkalmazásokban, és az operációkutatáson belül is tanulmányozták ezeket a problémákat. Például a Hubble űrteleszkóp kísérleteinek ütemezése a megfigyelések nagyon pontos időzítését igényli, a megfigyelések és a manőverek kezdeti és befejezési időpontjai folytonos értékű változók és csillagászati, precedencia-, valamint teljesítménykényszerek sokasága vonatkozik rájuk. A folytonos tartományú kényszerkielégítési problémák legismertebb válfaja a **lineáris programozás**i problémák csoportja, ahol a kényszerek *konvex* tartományt alkotó lineáris egyenlőtlenségek. A lineáris programozási feladatokat meg lehet oldani a változók száma szerinti polinomiális időben. Más típusú kényszerekkel és célfüggvényekkel rendelkező problémákat is tanulmányoztak, például ilyennel foglalkozik a kvadratikus programozás, a másodrendű kónikus programozás és hasonló társaik.

Azon túl, hogy megvizsgálhatjuk a kényszerkielégítési problémákban megjelenő változók típusát, hasznos lehet a kényszerfajtákat is tanulmányozni. A legegyszerűbb fajta az **unáris kényszer** (**unary constraint**), amely egy változó értékére tesz megkötést. Elképzelhető például, hogy a délausztrálok élénken tiltakoznak a *zöld* szín ellen. A hivatkozott változó tartományának előfeldolgozásával mindegyik unáris kényszer kiküszöbölhető: ki kell venni a kényszert sértő összes értéket. A **bináris kényszer** (**binary constraint**) két változót köt össze. Például a $DA \neq ÚDW$ egy bináris kényszer. A csak bináris kényszerekkel rendelkező kényszerkielégítési problémát binárisnak nevezzük, amit az 5.1. (b) ábrán láthatóhoz hasonló kényszergráffal lehet reprezentálni.

A magasabb rendű kényszerek három vagy több változóra vonatkoznak. Ennek egy ismerős példája a **betűrejtvény** (lásd 5.2. (a) ábra). A betűrejtvényben általában mindegyik betű különböző számot jelöl. Ez a megkötés az 5.2. (a) ábra betűrejtvénye esetén például a hatváltozós $MindKül(F,T,U,W,R,O)$ kényszerrel adható meg. Egy másik megoldás lehet, ha a megkötést bináris kényszerek (mint például $F \neq T$) gyűjteményével adjuk meg. A rejtvény négy oszlopában az összeadáskényszerek több változót is tartalmazhatnak, amelyek a következőképpen írhatók fel:

$$O + O = R + 10 \cdot X_1$$

$$X_1 + W + W = U + 10 \cdot X_2$$

$$X_2 + T + T = O + 10 \cdot X_3$$

$$X_3 = F$$

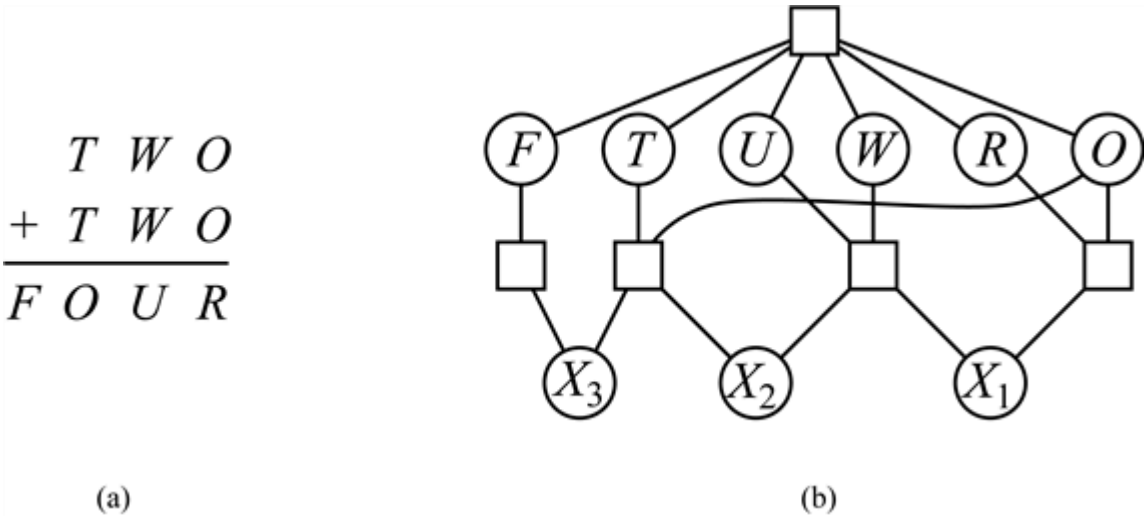
ahol X_1, X_2 és X_3 a következő oszlopba átvitt számjegyet (0 vagy 1) reprezentáló **segédváltozó**k (**auxiliary variable**s). A magasabb rendű kényszereket egy **kényszerhipergráf**fal (**constraint hyprergraph**) lehet ábrázolni, amelynek egy példája az 5.2. (b) ábrán látható. Az éles szemű olvasónak feltűnhet, hogy a *MindKül* kényszer felbontható bináris kényszerekre: $F \neq T$, $F \neq U$ stb. Az 5.11. feladat éppen annak bizonyítását kéri, hogy elegendő segédváltozó bevezetésével mindegyik magasabb rendű véges tartományú kényszer átírható bináris kényszerek halmazává. Ebből kiindulva ebben a fejezetben csak bináris kényszerekkel foglalkozunk.

Az eddig említett kényszerek mind **abszolút kényszer**ek voltak: egy kényszer megszegése kizár egy megoldásjelöltet. A valós alkalmazások során azonban sok kényszerkielégítési probléma tartalmaz **preferenciakényszer**eket, melyek jelzik, hogy mely megoldások preferáltak. Egy egyetemi órarend meghatározásakor például X professzor talán inkább reggel tanítana, míg Y professzor a délutáni órákat

Created by XMLmind XSL-FO Converter.

részesítené előnyben. Egy olyan órarend, amely szerint X professzornak délután kettőkor kellene tanítania, még megoldás lenne (persze csak ha X professzor éppenséggel nem a tanszékvezető), de ez nem lenne optimális megoldás. A preferenciakényszerek gyakran az egyedi változó-hozzárendelések költségeként ábrázolhatók: például ha X professzor egy délutáni időpontot kap, az két pontot ront az általános célfüggvényen, míg a reggeli óraidőpont csak egy pontba kerül. Így megfogalmazva a preferenciákkal kiegészített kényszerkielégítési problémák (útvonalalapú vagy lokális) optimalizációs keresési módszerekkel megoldhatók. Az ilyen kényszerkielégítési problémákkal a fejezetben többet nem foglalkozunk, de az irodalomjegyzékben megadunk néhány kiindulási pontként használható hivatkozást.

5.2. ábra - (a) Egy betűrejtvény. Mindegyik betű különböző számjegyet jelöl, a rejtvény célja olyan számjegyeket helyettesíteni a betűk helyére, amelyekkel a kiadódó összeg aritmetikailag helyes lesz (azzal a külön megkötéssel, hogy nem kezdő nullák nem megengedettek). (b) A betűrejtvény kényszerhipergráfja a *MindKül* kényszer és az oszloponkénti összeadási kényszerek feltüntetésével. A kényszereket négyzet alakú dobozok jelölik. Minden doboz azzal a változóval van összekötve, amelyre a doboz által jelzett kényszer vonatkozik.



2. A visszalépéses keresés alkalmazása kényszerkielégítési problémákra

Az előző alfejezetben keresési problémaként fogalmaztuk meg a kényszerkielégítési problémákat. Ezt a megfogalmazást felhasználva a 3. és a 4. fejezetben bemutatott bármely keresési eljárással megoldhatók a kényszerkielégítési problémák. Tegyük fel, hogy alkalmazzuk a szélességi keresést az előző fejezetben megadott általános CSP-megfogalmazásra. Hamarosan valami szörnyűség tűnik fel: a fa tetején az elágazási tényező nd, mert a d érték bármelyike hozzárendelhető az n változó bármelyikéhez. A következő szinten az elágazási tényező $(n - 1)d$ és így tovább az n szint során. Egy $n! \cdot d^n$ levelű fát építünk fel, noha csupán d^n lehetséges teljes hozzárendelés van!

Fontos

Látszólag racionális, valójában naiv problémamegfogalmazásunk elhanyagolta az összes kényszerkielégítési probléma egy döntő közös tulajdonságát: a **kommutativitás**t (**commutativity**). Egy probléma akkor kommutatív, ha a végeredmény szempontjából közömbös, hogy cselekvések egy adott sorozatát milyen sorrendben alkalmazzuk. Ilyenek a kényszerkielégítési problémák is, mert a változók hozzárendelése során a sorrendtől függetlenül ugyanazt a parciális hozzárendelést kapjuk. Ezért tehát *mindegyik kényszerkielégítési problémamegoldó a következő állapot generálásakor a keresési fa minden csomópontjában csak* egyetlen *változó lehetséges hozzárendeléseit veszi tekintetbe.* Az ausztráliai térképszínezési probléma keresési fájának gyökércsomópontjában választhatunk a *DA* =

Created by XMLmind XSL-FO Converter.

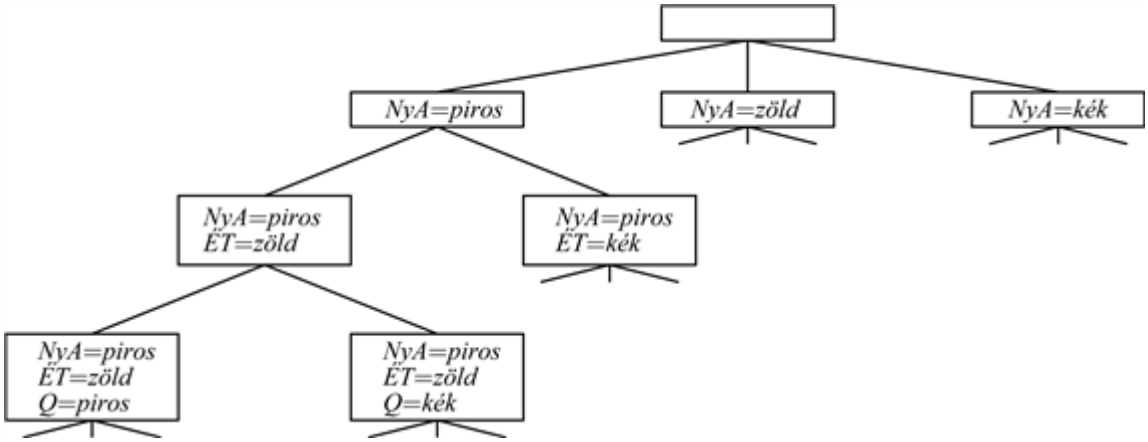
vörös, a *DA = zöld* és a *DA = kék* közül, de sohasem merül fel választási lehetőségként a *DA = vörös* vagy az *NyA = kék*. A levelek száma ezzel a megszorítással már a remélt d^n.

5.3. ábra - Egy egyszerű visszalépéses keresési algoritmus kényszerkielégítési problémákhoz. Az algoritmus a 3. fejezet rekurzív mélységi keresőjén alapul. A `HOZZÁRENDELETLEN-VÁLTOZÓ-KIVÁLASZTÁSA` és a `TARTOMÁNY-ÉRTÉKEK-SORRENDEZÉSE` függvények használhatók a szövegben bemutatott általános célú heurisztikák megvalósítására.

```
function VISSZALÉPÉSES-KERESÉS(csp) returns egy megoldás vagy kudarc
    return REKURZÍV-VISSZALÉPÉSES({}, csp)

function REKURZÍV-VISSZALÉPÉSES(hozzárendelések, csp) returns egy megoldás vagy kudarc
    if hozzárendelések teljes then return hozzárendelések
    var ← HOZZÁRENDELETLEN-VÁLTOZÓ-KIVÁLASZTÁSA(VÁLTOZÓK[csp], hozzárendelések, csp)
    for each érték in TARTOMÁNY-ÉRTÉKEK-SORRENDEZÉSE(var, hozzárendelések, csp) do
        if érték konzisztens a hozzárendelések-kel KÉNYSZEREK(csp) szerint then
            hozzáad {var = érték} hozzárendelések-hez
            eredmény ← REKURZÍV-VISSZALÉPÉSES(hozzárendelések, csp)
            if eredmény ≠ meghiúsulás then return eredmény
            kivesz {var = érték} hozzárendelések-ből
    return kudarc
```

5.4. ábra - Részlet a keresési fából, amelyet az egyszerű visszalépéses keresés generált az 5.1. ábrán bemutatott térképszínezési problémához



A **visszalépéses keresés** (**backtracking search**) kifejezést olyan mélységi keresésekre használjuk, melyek egyszerre csak egy változóhoz rendelnek értéket, és visszalépnek, ha már nincs megengedett hozzárendelési lehetőség. Az algoritmus az 5.3. ábrán látható. Vegyük észre, hogy az algoritmus lényegében a 115–116. oldalon leírt egyenkénti módszert használja az inkrementális következő állapot generálásra. Továbbá a következő állapot generálásakor kifejti az aktuális állapotot, nem pedig egyszerűen másolja. A kényszerkielégítési problémák szokásos reprezentációja miatt nincs szükség a `VISSZALÉPÉSES-KERESÉS` algoritmus kiegészítésére tárgyterület-specifikus kezdeti állapottal, állapotátmenet-függvénnyel vagy céltesztel. Az 5.4. ábrán látható egy részlet az ausztráliai színezési problémához generált keresési fából (a változó-hozzárendelések sorrendje *NyA, ÉT, Q,* …).

Az egyszerű visszalépéses keresés a 3. fejezet terminológiája szerint nem informált algoritmus, tehát nem is várjuk el, hogy nagy problémák esetén is hatékony legyen. Az 5.5. ábra első oszlopában bemutatott néhány eredmény megerősíti várakozásainkat.

5.5. ábra - Az egyes CSP-algoritmusok teljesítménye különböző problémák megoldásában. Az algoritmusok balról jobbra haladva a következők: egyszerű visszalépéses, MRV-heurisztikával bővített visszalépéses, előrenéző ellenőrzés, MRV-

Created by XMLmind XSL-FO Converter.

heurisztikával kiegészített előrenéző ellenőrzés és a minimális konfliktusok lokális keresése. Az egyes cellákban a probléma megoldása során elvégzett konzisztencia-ellenőrzések száma található (öt futtatás mediánértékéből meghatározva); figyeljünk fel arra, hogy a két jobb felső cella kivételével mindegyik érték ezrekben van megadva (ezt jelzi a K betű). Az első probléma az Amerikai Egyesült Államok 50 államát tartalmazó térkép négy színnel történő színezése. A többi probléma a (Bacchus és van Run, 1995)-ből származik (1. táblázat). A második probléma az *n*-királynő megoldásához szükséges ellenőrzések száma (*n* értékeire 2-től 50-ig összegezve). A harmadik az 5.13. feladatban is szereplő „zebrarejtvény". A maradék kettő mesterséges véletlenszerűen generált problémák. (A min-konfliktusok algoritmus ezeken nem futott.) Az eredmények azt érzékeltetik, hogy az MRV-heurisztikával kombinált előrenéző ellenőrzés mindegyik probléma esetén jobban teljesít, mint a többi visszalépéses algoritmus, de nem minden esetben jobb a min-konfliktusokat használó helyi keresésnél.

A probléma	Visszalépéses	BT+MRV	Előrenéző ellenőrzés	FC+MRV	Min-Konfliktusok
USA	(> 1000K)	(> 1000K)	2K	60	64
n-királynő	(> 40000K)	13500K	(> 40000K)	817K	4K
Zebra	3859K	1K	35K	0,5K	2K
Véletlen (1.)	415K	3K	26K	2K	
Véletlen (2.)	942K	27K	77K	15K	

A 4. fejezetben a nem informált keresési algoritmusok gyenge teljesítményét a probléma ismeretéből származó tárgyterület-specifikus heurisztika bevezetésével orvosoltuk. Az derül azonban ki, hogy a kényszerkielégítési problémák hatékonyan megoldhatók ilyen tárgyterület-specifikus tudás nélkül is. Olyan általános célú eljárások alakíthatók ki, amelyek az alábbi kérdéseket vizsgálják:

1. A következő lépésben melyik változóhoz rendeljünk értéket, és milyen sorrendben próbálkozzunk az értékekkel?

2. Milyen következményei vannak a jelenlegi változó-hozzárendeléseknek a még hozzárendeletlen változók számára?

3. Ha egy út sikertelennek bizonyul (azaz egy olyan állapothoz jutunk, ahol egy változónak nincs megengedhető értéke), a következő utak során el tudja-e kerülni a keresés ezt a hibát?

A következő pontokban megpróbáljuk sorra megválaszolni ezeket a kérdéseket.

2.1. Változó- és értékrendezés

A visszalépéses algoritmus tartalmazza az alábbi sort:

```
var ← Hozzárendeletlen-Változó-Kiválasztás(Változók[csp], hozzárendelések, csp)
```

Alapértelmezés szerint a HOZZÁRENDELETLEN-VÁLTOZÓ-KIVÁLASZTÁS egyszerűen kiválasztja a VÁLTOZÓK[*csp*] lista sorrendje szerinti következő hozzárendeletlen változót. A statikus változó-hozzárendelés ritkán vezet a leghatékonyabb kereséshez. Például miután kiválasztottuk az *NyA* = *vörös* és az *ÉT* = *zöld* hozzárendeléseket, csak egyetlen értéket rendelhetünk *DA*-hoz, tehát ésszerű lenne inkább a *DA* = *kék* hozzárendelést elvégezni, mintsem *Q* számára keresni értéket. Sőt miután *DA* értéket kapott, kényszerítve vagyunk *Q, ÚDW* és *V* értékeinek kiválasztásakor. Ezt az intuitív ötletet (a legkevesebb „megengedett" értékkel rendelkező változó kiválasztását) nevezik **legkevesebb fennmaradó érték** (**minimum remaining values**, **MRV**) heurisztikának. Nevezték már „legkorlátozottabb változó" vagy „meghiúsulást előre" heurisztikának is (az utóbbi névnek az a magyarázata, hogy azt a változót emeli ki, amelyik legvalószínűbben fog hamarosan hibához vezetni, ezáltal megnyesve a keresési fát). Ha van egy *X* változó, amelynek egyetlen megengedett értéke sincs, akkor az MRV-heurisztika ki fogja választani *X*-et, és azonnal kideríti a hibát, elkerülve ezzel a többi változó közötti értelmetlen keresgélést, ami mindig kudarcra vezet, amikor *X* végül is kiválasztódik. Az 5.5. ábra második, BT+MRV címkéjű oszlopa ennek a heurisztikának a teljesítményét mutatja. Az egyszerű visszalépéses

Created by XMLmind XSL-FO Converter.

keresésnél a teljesítmény a problémától függően mintegy 3…3000-szer jobb lehet. Vegyük észre, hogy a teljesítménymértékünk nem tartalmazza a heurisztikus értékek kiszámításának többletköltségét. A következő pontban bemutatunk egy módszert, amellyel kezelhetővé válik ez a költség.

Az MRV-heurisztika semmit sem segít abban, hogy melyik régiót válasszuk ki elsőként Ausztrália kiszínezésekor, mert a kiinduláskor mindegyik régiónak három megengedett színe van. Ebben az esetben segíthet a **fokszám-heurisztika** (**degree heuristics**). Ez azáltal kísérli meg a későbbi választások elágazási tényezőjét csökkenteni, hogy azt a változót választja ki, amely a legtöbbször szerepel a hozzárendeletlen változókra vonatkozó kényszerekben. Az 5.1. ábrán *DA* a legnagyobb fokszámú változó (fokszáma 5), az összes többi változó fokszáma 2 vagy 3, leszámítva *T*-t, amelyé nulla. Valójában, ha már egyszer *DA*-t kiválasztottuk, a fokszám-heurisztika alkalmazása a problémát egyetlen hibás lépés nélkül oldja meg: mindegyik választási pontnál bármelyik konzisztens színt is választjuk, mindenképpen visszalépés nélkül jutunk egy megoldáshoz. A legkevesebb fennmaradó érték heurisztika általában jobban irányít, de a patthelyzetek eldöntéséhez hasznos segítség lehet a fokszám-heurisztika.

Miután az algoritmus már kiválasztott egy változót, el kell döntenie, hogy milyen sorrendben vizsgálja meg ennek értékeit. Erre néha a **legkevésbé-korlátozó-érték** (**least-constraining-value**) heurisztika lehet hasznos. Ez a heurisztika előnyben részesíti azt az értéket, amely a legkevesebb választást zárja ki a kényszergráfban a szomszédos változóknál. Például tegyük fel, hogy az 5.1. ábra esetében már létrehoztuk az *NyA = vörös* és az *ÉT = zöld* parciális hozzárendelést, és most *Q* számára akarunk értéket találni. A kék rossz választás volna, mert ez kizárja *Q* szomszédjának, *DA*-nak utolsó megengedett értékét. A legkevésbé korlátozó heurisztika ezért a kékkel szemben a vöröset részesíti előnyben. Általában véve a heurisztika megpróbálja a későbbi változó-hozzárendelések számára a lehető legnagyobb szabadságot meghagyni. Természetesen ha egy probléma összes megoldását meg szeretnénk találni, és nem elégszünk meg csupán eggyel, akkor a sorrend közömbös, hiszen úgyis minden értéknek sorra kell kerülnie. Ugyanez igaz akkor is, ha a problémának nincsen megoldása.

2.2. Az információ terjesztése a kényszereken keresztül

A keresési algoritmusunk eddig csak akkor foglalkozott egy változóra vonatkozó kényszerrel, ha a változót a `HOZZÁRENDELETLEN-VÁLTOZÓ-KIVÁLASZTÁS` kiválasztotta. De ha néhány kényszert a keresés folyamán korábban vagy akár még a keresés megkezdése előtt megvizsgálunk, akkor drasztikusan csökkenthető a keresési tér mérete.

2.2.1. Előrenéző ellenőrzés

A kényszerek keresés közbeni jobb felhasználásának egyik módszerét **előrenéző ellenőrzés**nek (**forward checking**) nevezzük. Az előrenéző ellenőrzési folyamat minden egyes alkalommal, amikor egy *X* változó értéket kap, minden, az *X*-hez kényszerrel kötött, hozzárendeletlen *Y*-t megvizsgál, és *Y* tartományából törli az *X* számára választott értékkel inkonzisztens értékeket. Az 5.6. ábra bemutatja az előrenéző ellenőrzéses keresés menetét a térképszínezési probléma során. Két fontos dolgot kell megemlítenünk ezzel a példával kapcsolatban. Először is vegyük észre, hogy miután elvégeztük az *NyA = vörös* és a *Q = zöld* hozzárendeléseket, *ÉT* és *DA* tartományai egyetlen elemre szűkültek, és azzal, hogy az *NyA*-ból és a *Q*-ból származó információt terjesztettük, teljesen megszüntettük az ezen változók szerinti elágazást. Az MRV-heurisztika, amely az előrenéző ellenőrzés nyilvánvaló partnere, automatikusan *DA*-t és *ÉT*-t választja következőnek. (Tulajdonképpen még hatékonyabb eljárássá tehetjük az előrenéző ellenőrzést, ha az MRV-heurisztika munkájához szükséges információt inkrementálisan számítjuk.) A másik dolog, amire érdemes felfigyelni, az az, hogy a *V = kék* hozzárendelés után *DA* tartománya üres lett. Ezért az előrenéző ellenőrzés megállapítja, hogy a {*NyA = vörös*, *Q =zöld*, *V = kék*} részleges hozzárendelés inkonzisztens a probléma kényszereivel, és az algoritmus azonnal visszalép.

2.2.2. A kényszerek terjesztése

Az előrenéző ellenőrzés ugyan sok inkonzisztenciát észrevesz, de nem mindet. Például nézzük csak az 5.6. ábra harmadik sorát. Az látható itt, hogy amikor *DA vörös* és *Q zöld*, mind *ÉT*-nek, mind *DA*-nak kéknek kell lennie. De ezek szomszédosak, és nem lehet azonos az értékük. Az előrenéző ellenőrzés nem veszi észre ezt az inkonzisztenciát, mert nem néz elég messze előre. A **kényszer**ek **terjesztése** (**constraint propagation**) általános kifejezés jelöli azt, ha az egyik változó kényszerének a többi változót érintő következményeit terjesztjük: esetünkben *NyA*-ról *Q*-ra és *ÉT*-ről *DA*-ra kell terjeszteni (ahogy ezt már az előrenéző ellenőrzés is megtette), majd az *ÉT* és *DA* közti kényszerre, hogy észrevehessük az inkonzisztenciát. Ezt pedig gyorsan szeretnénk tenni: semmi értelme a keresés méretét csökkenteni, ha több időt töltünk a kényszerek terjesztésével, mint az egyszerű kereséssel tennénk.

Created by XMLmind XSL-FO Converter.

5.6. ábra - A térképszínezés folyamata előrenéző ellenőrzéses keresés esetén. Az első hozzárendelés az *NyA = vörös*, majd az előrenéző ellenőrzés törli a *vörös* értéket a szomszédos *ÉT* és *DA* változók tartományaiból. A *Q = zöld* hozzárendelés után a *zöld* is kikerül az *ÉT*, *DA* és *ÚDW* változók tartományaiból. A *V = kék*-et követően a *kék* is törlődik a *DA* és *ÚDW* változók tartományaiból, és *DA* számára nem marad megengedett érték.

	NyA	*ÉT*	*Q*	*ÚDW*	*V*	*DA*	*T*
Kezdeti tartományok	V Z K	V Z K	V Z K	V Z K	V Z K	V Z K	V Z K
Miután NyA = vörös	Ⓥ	Z K	V Z K	V Z K	V Z K	Z K	V Z K
Miután Q = zöld	Ⓥ	K	Ⓩ	V K	V Z K	K	V Z K
Miután V = kék	Ⓥ	K	Ⓩ	V	Ⓚ		V Z K

Az **élkonzisztencia** (**arc consistency**) alapul szolgálhat egy gyors, az előrenéző ellenőrzésnél lényegesen erősebb kényszerterjesztéshez. Az „él” itt a kényszergráf *irányított* éleit jelenti, amilyen például a *DA*-ból az *ÚDW*-be mutató él. Ha adott *DA* és *ÚDW* aktuális tartománya, akkor ez az él konzisztens, ha *DA mindegyik x* értékéhez található egy *x*-szel konzisztens *valamilyen y* érték *ÚDW*-ben. Az 5.6. ábra harmadik sorában *DA* és *ÚDW* pillanatnyi tartománya rendre a {*kék*}, illetve a {*vörös, kék*}. A *DA = kék* esetén található egy konzisztens hozzárendelés *ÚDW*-hez, nevezetesen az *ÚDW = vörös*; ezért a *DA*-ból *ÚDW*-be mutató él konzisztens. Viszont az *ÚDW*-ből a *DA*-ba mutató él nem konzisztens, mert az *ÚDW* = {*kék*} hozzárendelésnél a *DA*-hoz nem tudunk színt találni. Az él úgy tehető konzisztenssé, hogy az *ÚDW* tartományából töröljük a *kék* értéket.

A *DA*-ból *ÚT*-be mutató él mentén is alkalmazhatjuk a keresési folyamat ugyanazon fázisában az élkonzisztenciát. Az 5.6. ábra harmadik sora szerint mindkét változó értéktartománya {*kék*}. Ez azt eredményezi, hogy törölnünk kell a *kék*-et *DA* tartományából, amely ezzel kiürül. Az élkonzisztencia ellenőrzés tehát lehetővé teszi, hogy korábban észrevegyük az egyszerű előrenéző ellenőrzés által fel nem fedett inkonzisztenciát.

Az élkonzisztencia-ellenőrzés alkalmazható előfeldolgozó lépésként a keresés megkezdése előtt, vagy a keresési folyamat minden egyes hozzárendelését követő terjesztési lépésként (az előrenéző ellenőrzéshez hasonlóan). (Az utóbbi algoritmust *Élkonzisztencia fenntartásának* is, angol rövidítéssel MAC-nak nevezik.) Mindkét esetben addig kell *ismételve* alkalmazni a folyamatot, amíg nem marad inkonzisztencia. Erre azért van szükség, mert amikor egy élinkonzisztenciát eltávolítandó egy érték kikerül egy változó tartományából, az ehhez a változóhoz mutató éleknél új inkonzisztencia jöhet létre. Az AC-3, az élkonzisztencia teljes algoritmusa egy sort használ annak nyilvántartására, hogy mely élek inkonzisztenciáját kell még ellenőriznie (lásd 5.7. ábra). Minden egyes (X_i, X_j) élet sorban egyenként levesszük a tennivalók listájáról és ellenőrizzük; ha pedig X_i tartományának bármely változóját törölni kell, akkor minden X_i -be mutató (X_k, X_i) élet visszateszünk ellenőrzésre a sorba. Az élkonzisztencia-ellenőrzés komplexitása az alábbiak szerint vizsgálható: egy bináris kényszerkielégítési problémában legfeljebb $O(n^2)$ él van; minden egyes (X_k, X_i) él csak d alkalommal kerülhet napirendre, mert X_i-ben összesen d törölhető érték van; egy él konzisztenciájának ellenőrzése elvégezhető $O(d^2)$ időben; tehát a legrosszabb esetben vett teljes idő $O(n^2d^3)$. Ez ugyan lényegesen költségesebb, mint az előretekintő ellenőrzés, de az extra költség általában kifizetődik.[1]

5.7. ábra - Az AC-3 élkonzisztencia algoritmus. Az AC-3 alkalmazását követően vagy mindegyik él élkonzisztens, vagy néhány változó tartománya üres, azaz a kényszerkielégítési probléma nem hozható élkonzisztens alakra (tehát nem oldható meg). Az „AC-3” nevet az algoritmus kitalálója (Mackworth, 1977) vezette be, mert ezt az algoritmust mutatta be cikkében harmadikként.

[1] A Mohr és Henderson (Mohr és Henderson, 1986) bemutatta AC-4 algoritmus $O(n^2d^2)$ alatt fut. (Lásd 5.10. feladat.)

Created by XMLmind XSL-FO Converter.

```
function AC-3(csp) returns a CSP-t, lehetőleg redukált tartományokkal
    inputs: csp, egy bináris CSP {X1, X2, …, Xn } változókkal
    local variables: sor, a sorra következő élek, kiindulásként csp összes éle

    while sor nem üres do
        (Xi, Xj) ← ELSŐT-KIVESZ(sor)
        if INKONZISZTENS-ÉRTÉKEKET-KIVESZ(Xi, Xj) then
            for each Xk in SZOMSZÉDAI[Xi] do
                az (Xk, Xi) belekerül a sor-ba

function INKONZISZTENS-ÉRTÉKEKET-KIVESZ(Xi, Xj) returns igaz, a.cs.a. ha kiveszünk egy értéket
    eltávolítva ← hamis
    for each x in TARTOMÁNY[Xi] do
        if a TARTOMÁNY[Xj] egyetlen y értéke esetén sem elégíti ki (x, y) az Xi és Xj közti kényszert
            then x törlése TARTOMÁNY[Xi]-ből; eltávolítva ← igaz
    return eltávolítva
```

Mivel a kényszerkielégítési problémák speciális esetként tartalmazzák a 3SAT-ot, ezért nem várhatjuk, hogy polinom idejű algoritmust találunk annak eldöntésére, hogy egy adott kényszerkielégítési probléma konzisztens-e. Ebből arra következtethetünk, hogy az élkonzisztencia nem tár fel minden lehetséges inkonzisztenciát. Például az 5.1. ábrán az {*NyA = vörös, ÚDW = vörös*} részleges hozzárendelés inkonzisztens, de az AC-3 nem találja meg. Definiálható a terjesztés erősebb formája is, amit **k-konzisztenciá**nak (**k-consistency**) nevezünk. Egy kényszerkielégítési probléma akkor *k*-konzisztens, ha bármely *k* – 1 változóból álló halmaz és ezen változók bármely konzisztens hozzárendelése esetén mindig lehet a *k*-adik változónak konzisztens értéket találni. Az 1-konzisztencia például azt jelenti, hogy mindegyik individuális változó önmagában véve is konzisztens; ezt **csomópont-konzisztenciá**nak (**node consistency**) is nevezik. A 2-konzisztencia ugyanaz, mint az élkonzisztencia. A 3-konzisztencia azt jelenti, hogy egymás melletti változók bármely párja mindig kiterjeszthető egy szomszédos változóra; ezt **útvonal-konzisztenciá**nak (**path consistency**) is nevezik.

Egy gráf **erősen k-konzisztens** akkor, ha *k*-konzisztens, valamint (*k* – 1)-konzisztens, (*k* – 2)-konzisztens és így tovább le egészen az 1-konzisztenciáig. Tegyük fel most, hogy van egy *n* csomópontból álló kényszerkielégítési problémánk, és tegyük erősen *n*-konzisztenssé (azaz *k*-konzisztenssé *k* = *n* esetében). Ekkor visszalépés nélkül megoldhatjuk a problémát. Először egy konzisztens értéket választunk X_1 számára. Ezután garanciánk van arra, hogy tudunk X_2 számára értéket választani, mert a gráf 2-konzisztens, X_3 számára, mert a gráf 3-konzisztens és így tovább. Minden egyes X_i változó esetén csak *d* értéket kell ellenőriznünk a tartományból, hogy egy $X_1,..., X_{i-1}$-gyel konzisztens értéket találjunk. Garanciánk van tehát arra, hogy $O(nd)$ időben megoldást találunk. Persze semmit sem adnak ingyen: az *n*-konzisztenciát biztosító bármely algoritmus legrosszabb esetben *n*-ben exponenciális lesz.

Az *n*-konzisztencia és az élkonzisztencia között egy széles köztes mező húzódik: az erősebb konzisztencia-ellenőrzések futtatása tovább fog tartani, de több eredménnyel jár az elágazási tényező csökkentésében és az inkonzisztens részleges hozzárendelések felderítésében. Ki lehet számolni a legkisebb olyan *k* értéket, melyre a *k*-konzisztencia ellenőrzésének futtatása visszalépés nélküli problémamegoldást tesz lehetővé (lásd 5.4. alfejezet), de ez gyakran nem célszerű. A legmegfelelőbb konzisztencia-ellenőrzési szint kiválasztása valójában leginkább empirikus tudomány.

2.2.3. Speciális kényszerek kezelése

A kényszerek bizonyos típusai gyakran fordulnak elő a valós problémákban és speciális célú algoritmusokkal jóval hatékonyabban kezelhetők, mint az eddig leírt általános célú módszerekkel. Például a *MindKül* kényszer azt állítja, hogy a benne szereplő összes változóknak mind különböző értéket kell felvennie (ahogy a betűrejtvényes példában láttuk). A *MindKül* kényszer inkonzisztencia-ellenőrzésére egy egyszerű módszer az, hogy ha *m* változó szerepel a kényszerben, és ha ezeknek együttesen *n* különböző értéke lehet, akkor *m* > *n* esetén a kényszert nem lehet kielégíteni.

Ez a következő egyszerű algoritmushoz vezet: először vegyünk ki a kényszerből minden változót, amelynek egyelemű tartománya van, és vegyük ki ezeknek a változóknak az értékeit a megmaradó változók tartományaiból. Ismételjük ezt a lépést mindaddig, amíg van ilyen változónk. Ha bármely ponton egy üres

Created by XMLmind XSL-FO Converter.

tartomány jön létre, vagy több változónk marad, mint értékünk a tartományban, akkor egy inkonzisztenciát derítettünk fel.

Ezt a módszert használhatjuk arra, hogy az 5.1. ábra {*NyA = vörös, ÚDW = vörös*} részleges hozzárendelésében észrevegyük az inkonzisztenciát. Figyeljünk fel arra, hogy a *DA*, az *ÚT* és a *Q* változók lényegében egy *MindKül* kényszerrel vannak összekötve, mert mindegyik párnak különböző színűnek kell lennie. Miután alkalmaztuk az AC-3-at a részleges hozzárendelésre, az egyes változók tartománya a {*zöld, kék*} tartományra szűkült le. Azaz három változónk van és csupán két színünk, tehát megszegtük a *MindKül* kényszert. Tehát a magasabb szintű kényszerekre alkalmazott egyszerű konzisztencia-ellenőrzések néha hatékonyabbak, mintha az ekvivalens bináris kényszerek halmazára vizsgáltuk volna az élkonzisztenciát.

A legfontosabb magasabb rendű kényszer talán az **erőforráskényszer** (**resource constraint**), amit néha *legfeljebb* kényszernek is neveznek. Jelölje például PA_1, ..., PA_4 rendre négy feladathoz hozzárendelt személyzet számát. Azt a kényszert, miszerint összesen nem lehet 10-nél több embernek feladatot adni, így írhatjuk fel: *legfeljebb*(10, PA_1, PA_2, PA_3, PA_4). Az inkonzisztenciát egyszerűen ki lehet mutatni az aktuális tartományok minimális értékeinek összegzésével: ha például mindegyik változó tartománya a {3, 4, 5, 6}, akkor a *legfeljebb* kényszert nem lehet kielégíteni. A konzisztenciát azáltal is érvényesíthetjük, hogy töröljük bármely tartomány maximális értékét, amely nem konzisztens az összes többi tartomány minimális értékével. Így ha példánkban mindegyik változónak {2, 3, 4, 5, 6} a tartománya, akkor az 5 és a 6 értékeket mindegyik tartományból törölni lehet.

A nagy erőforrás-korlátozott egész értékű problémáknál – ilyenek például az emberek ezreit és járművek százait mozgató logisztikai problémák – általában nem tehető meg az, hogy mindegyik változó tartományát egy nagy egészhalmazzal ábrázoljuk, majd ezt a halmazt a konzisztencia-ellenőrző módszerek segítségével fokozatosan csökkentjük. Ehelyett a tartományokat az alsó és a felső határaikkal ábrázoljuk, és határterjesztés segítségével kezeljük. Tegyük fel például, hogy két repülőjárat van, a 271-es és a 272-es, amelyekre a repülőgépek kapacitása rendre 165 és 385. A repülőjárat utasszámának kezdeti tartománya tehát

*Járat*271 ∈ [0, 165] és *Járat*272 ∈ [0, 385]

Tegyük fel most, hogy van egy külön korlát, miszerint a két járatnak együtt 420 embert kell szállítania. *Járat*271 + *Járat*272 ∈ [420, 420]. A határkényszereket terjesztve a tartományokat a következőkre csökkentjük:

*Járat*271 ∈ [35, 165] és *Járat*272 ∈ [225, 385]

Akkor nevezünk egy kényszerkielégítési problémát határkonzisztensnek, ha minden *X* változóra ennek a változónak mind az alsó, mind a felső határértékére található minden *Y* változóhoz olyan érték, amely kielégíti az *X* és *Y* közti kényszereket. Ezt a fajta **kényszerterjesztés**t (**bound propagation**) széles körben használják a gyakorlati kényszerkielégítési problémák során.

2.3. Intelligens visszalépés: visszanézni

Az 5.3. ábra `VISSZALÉPÉSES-KERESÉS` algoritmusa egy elég egyszerű intézkedést alkalmaz akkor, amikor egy keresési ág meghiúsul: visszalép az előző változóra, és megpróbál számára egy másik értéket találni. Ezt **időrendi visszalépés**nek (**chronological backtracking**) nevezik, mert a *legutolsó* döntési pontot keresi fel újra. Ebben az alrészben látni fogjuk, hogy adódik erre sokkal jobb módszer is.

Nézzük meg, mi történik, amikor az egyszerű visszalépéses algoritmust az 5.1. ábrán látható problémára alkalmazzuk rögzített változósorrenddel (*Q, ÚDW, V, T, DA, NyA, ÚT*). Tegyük fel, hogy már létrehoztuk a {*Q = vörös, ÚDW = zöld, V = kék, T = vörös*} hozzárendelést. Amikor a következő változóval, *DA*-val próbálkozunk, azt látjuk, hogy mindegyik érték sérti a kényszert. Visszalépünk *T*-re, és egy új színt keresünk Tasmania számára. Ez nyilvánvalóan butaság: Tasmania átszínezése nem oldja meg a Dél-Ausztráliával kapcsolatos problémát.

A visszalépéses megközelítés intelligens alkalmazása lenne, ha ahhoz a változóhalmazhoz mennénk vissza, amely a *meghiúsulást okozta*. Ezt a halmazt **konfliktushalmaz**nak (**conflict set**) nevezik; esetünkben a *DA*-hoz tartozó konfliktushalmaz a {*Q, ÚDW, V*}. Általános esetben egy *X* változó konfliktushalmaza a korábban értéket kapott változók egy halmaza, melyeket *X*-hez kényszerek kötnek. A **visszaugrás** (**backjumping**) a konfliktushalmaz-beli legutolsó változóhoz lép vissza; esetünkben a visszaugrás átugorná Tasmaniát, és *V* számára próbálna új értéket keresni. Ezt egyszerű implementálni a `VISSZALÉPÉSES-KERESÉS` olyan

Created by XMLmind XSL-FO Converter.

módosításával, hogy az összegyűjtse a konfliktushalmazt, miközben hozzárendelhető értékeket keres. Ha nem talál ilyen értéket, akkor a konfliktushalmaz legutolsó elemét kell visszaadnia (a meghiúsulás jelzésével együtt).

Az éles szemű olvasónak feltűnhetett, hogy az előrenéző ellenőrzés a konfliktushalmazt minden külön munka nélkül előállíthatja: mindig amikor az előrenéző ellenőrzés *X* egy hozzárendelésén alapulva *Y* tartományából kitöröl egy elemet, *X*-et hozzá kell adnia *Y* konfliktushalmazához. Ezenkívül még minden esetben, amikor az utolsó értéket törli *Y* tartományából, az *Y* konfliktushalmazában szereplő változókat hozzá kell adnia *X* konfliktushalmazához. Ezek után mihelyt *Y*-hoz jutunk, már azonnal ismerni fogjuk, hogy hova lépjünk vissza, ha erre szükség van.

Fontos

A sasszemű olvasónak valami furcsaság is feltűnhetett: a visszaugrás akkor következik be, amikor egy tartomány minden értéke konfliktusban van az aktuális hozzárendelésekkel, de az előrenéző ellenőrzés felismeri ezt az eseményt, és megakadályozza, hogy valaha is ilyen csomóponthoz jussunk! Valóban, meg lehet mutatni, hogy *a visszaugrás által levágott minden ágat az előrenéző ellenőrzés is levágja*. Tehát az egyszerű visszaugrás redundáns az előrenéző ellenőrzés során, vagy pontosabban szólva, egy olyan keresésnél, amely erősebb konzisztencia-ellenőrzést, például MAC-ot használ.

Az előző bekezdés megfigyelése ellenére a visszaugrás mögötti ötlet továbbra is használható marad: lépjünk vissza a hiba helyére. A visszaugrás akkor veszi észre a kudarcot, amikor egy változó értéktartománya üressé válik, de sok esetben egy ág már sokkal korábban levágásra lett ítélve. Tekintsük ismét az {*NyA* = *vörös, ÚDW* = *vörös*} részleges hozzárendelést (amelyik, korábbi megjegyzéseink értelmében, inkonzisztens). Tegyük fel, hogy a *T* = *vörös*-t próbáljuk, majd az *ÚT, Q, V, DA*-kra alkalmazzunk hozzárendeléseket. Tudjuk, hogy az utóbbi négy változó esetén semmilyen hozzárendelés sem lesz jó, így végül *ÚT* próbálgatásánál kifogyunk az értékekből. A kérdés most az, hogy hova lépjünk vissza? A visszaugrás nem fog működni, mert *ÚT*-nek *igenis* van az előző hozzárendelésekkel konzisztens értéke: *ÚT* esetén nincsen teljes konfliktushalmaz a megelőző változók között, amely felelős lenne a kudarcért. Tudjuk azonban, hogy a négy változó, *ÚT, Q, V,* és *DA együttvéve* a megelőző változók egy halmaza miatt hiúsul meg, és ezek azok a változók, amelyek közvetlenül konfliktusban állnak a kérdéses néggyel. Ennek alapján *ÚT* konfliktushalmazának egy mélyebb megértéséhez jutottunk: a konfliktushalmaz a megelőző változók azon halmaza, ami *bármely következő változóval együtt* felelős *ÚT* konzisztens megoldásának kudarcáért. Esetünkben ez a halmaz az *NyA* és az *ÚDW*, tehát az algoritmusnak *ÚDW*-hez kell visszalépnie átugorva Tasmaniát. A konfliktushalmaz ilyen definíciójára támaszkodó visszaugrási algoritmust nevezzük **konfliktusvezérelt visszaugrás**nak (**conflict-directed backjumping**).

Magyarázatot kell most adnunk arra, hogy miként lehet kiszámítani ezeket az újfajta konfliktushalmazokat. A módszer valójában nagyon egyszerű. Egy keresési ág „terminálási” kudarca mindig azért következik be, mert egy változó tartománya kiürül, ennek a változónak pedig egy szokásos konfliktushalmaza van. Példánkban *DA* hiúsul meg, és konfliktushalmaza (mondjuk) az {*NyA, ÚT, Q*}. Visszaugrunk *Q*-ra, és *Q beépíti* saját közvetlen konfliktushalmazába (ami az {*ÚT, ÚDW*}) *DA* konfliktushalmazát (leszámítva persze *Q*-t magát), és így előáll az {*NyA, ÚT, ÚDW*} konfliktushalmaz. Azaz *Q*-tól számítva tovább már nincsen megoldás feltételezve {*NyA, ÚT, ÚDW*} korábbi hozzárendeléseit. Tehát visszalépünk *ÚT*-re, amelyik a legutóbbi ezek közül. *ÚT* felveszi {*NyA, ÚT, ÚDW*} – {*ÚT*}-t a saját közvetlen konfliktushalmazába, {*NyA*}-ba, és előáll az {*NyA, ÚDW*} konfliktushalmaz (ahogy az előző bekezdésben állítottuk). Foglaljuk össze az eddig mondottakat: jelölje X_j az aktuális változót és $conf(X_j)$ a saját konfliktushalmazát. Ha X_j minden lehetséges értéke meghiúsul, ugorjunk vissza a $conf(X_j)$-beli legutolsó X_i értékre és legyen

$$conf(X_i) \leftarrow conf(X_i) \cup conf(X_j) - \{X_i\}$$

A konfliktusvezérelt visszaugrás a keresési fa megfelelő helyére visz vissza, de nem akadályozza meg, hogy ugyanazt a hibát elkövessük a fa egy másik ágán. A **kényszertanulás** (**constraint learning**) a konfliktusból kiemelt új kényszer megtanulásával módosítja a kényszerkielégítési problémát.

3. Lokális keresés kényszerkielégítési problémáknál

A lokális keresési algoritmusok (lásd 4.3. alfejezet) nagyon hatékonynak bizonyulnak sok kényszerkielégítési probléma megoldásában: a kiinduló állapot minden változóhoz értéket rendel, és az állapotátmenet-függvény működése során általában egyszerre csak egy változó értékét módosítja. A 8-királynő problémában például a

Created by XMLmind XSL-FO Converter.

kezdeti állapot lehet a 8 királynő véletlenszerű elhelyezése a 8 oszlopban, az állapotátmenet-függvény pedig kiemel egy királynőt, és megpróbálja a saját oszlopán belül máshova helyezni. Egy másik lehetőség lehetne, ha úgy indulnánk, hogy a 8 királynő mindegyikét egy külön oszlopba a nyolc sor permutációjával helyezzük el, és az a következő állapotokat pedig két királynő sorának felcserélésével generálnánk.[2]Tulajdonképpen már láttunk egy példát lokális keresés alkalmazására egy CSP-probléma megoldásánál: ilyen volt a hegymászás alkalmazása a 8-királynő probléma megoldására 3.1. szakasz - Hegymászó keresés. Egy másik alkalmazás a WALKSAT 5.2. szakasz - Lokális keresés algoritmus a kielégíthetőségi probléma megoldására, mely szintén egy speciális esete a kényszerkielégítési problémáknak.

Amikor egy változónak új értéket választunk, a legnyilvánvalóbban kínálkozó heurisztika annak az értéknek kiválasztása, amelyik a legkevesebb konfliktust eredményezi más változókkal. Ezt nevezik **min-konfliktusok** (**min-conflicts**) heurisztikának. Az algoritmus az 5.8. ábrán látható, a 8-királynő problémára történő alkalmazását az 5.9. ábrán mutatjuk be, a kiértékelés eredménye pedig az 5.5. ábrán szerepel.

A min-konfliktusok heurisztika meglepően hatékony sok kényszerkielégítési probléma megoldásában, különösen amikor adott egy viszonylag jó kezdeti állapot. Teljesítményét az 5.5. ábra utolsó oszlopa mutatja. Meglepő módon az *n*-királynő probléma esetén a min-konfliktusok algoritmus futási ideje a kezdeti elhelyezéseket nem számítva nagyjából *független a probléma méretétől.* Akár a *millió*-királynő problémát is megoldja átlagosan ötven lépésben (a kezdeti értékadást követően). Ez a figyelemre méltó megfigyelés a kilencvenes években komoly kutatómunkához vezetett a lokális keresés, valamint a könnyű és nehéz problémák megkülönböztetése terén (ezzel a 7. fejezetben még foglalkozunk). Durván szólva az *n*-királynő azért könnyű a lokális keresés számára, mert a megoldások az állapottérben sűrűn helyezkednek el. A min-konfliktusok algoritmus nehéz problémák esetén is működik. Alkalmazták például a Hubble-űrtávcső megfigyeléseinek ütemezésére, és az egy hétre tervezett megfigyelések ütemezésének elkészítéséhez szükséges három hetet (!) körülbelül tíz percre rövidítette le.

5.8. ábra - A MIN-KONFLIKTUSOK algoritmus kényszerkielégítési problémák megoldására lokális kereséssel. A kezdeti állapotot véletlenszerűen választjuk ki, vagy egy mohó hozzárendelési folyamat segítségével, amely minden egyes sorra kerülő változónak egy minimális-konfliktus értéket választ. A KONFLIKTUSOK függvény megszámolja az adott érték által megsértett kényszerek számát (adottnak tekintve az aktuális hozzárendelések fennmaradó részét).

```
function MIN-KONFLIKTUSOK(csp, max_lépések) returns egy megoldás vagy kudarc
    inputs: csp, egy kényszerkielégítési probléma
            max_lépések, a lépések megengedett száma, mielőtt feladnánk

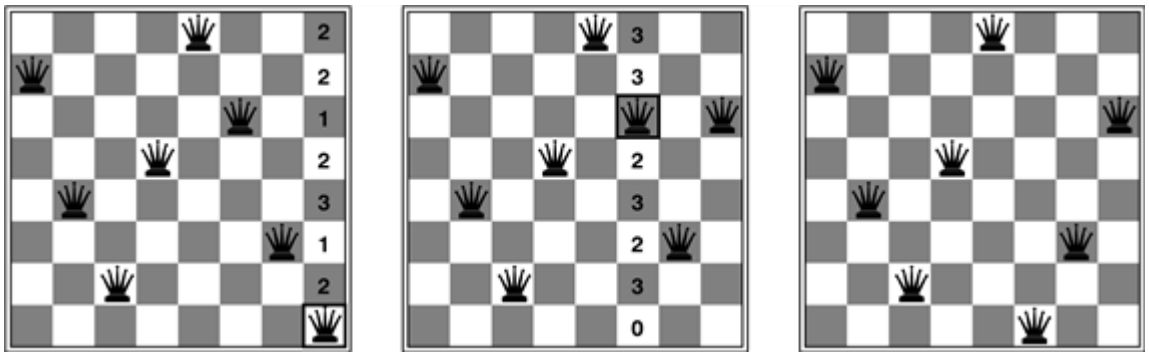
    aktuális ← csp egy kezdeti teljes hozzárendelése
    for i = 1 egészen max_lépések-ig do
        if aktuális egy megoldása csp-nek then return aktuális
        vált ← egy véletlenszerűen kiválasztott, konfliktusban lévő változó VÁLTOZÓK[csp]-ből
        érték ← azon v értéke vált-nak, amely minimalizálja KONFLIKTUSOK(vált, v, aktuális, csp)-t
        aktuális-ban állítsa vált-ot érték-re
    return kudarc
```

A lokális keresés másik előnye az, hogy alkalmazható online elrendezésben is, amikor a probléma változik. Ez különösen fontos az ütemezési problémákban. Egy légitársaság heti ütemezése járatok ezreit és emberekhez rendelt feladatok tízezreit tartalmazhatja, de egy rossz időjárás az egyik repülőtéren lehetetlenné teheti az ütemezést. Ezt a lehető legkevesebb változtatással szeretnénk helyrehozni, amit a jelenlegi ütemezésből kiinduló lokális keresési algoritmussal végezhetünk el könnyen. Egy visszalépéses keresés a kényszerek egy új halmazával általában jóval több időbe telik, és olyan megoldást is találhat, amely jelentősen különbözik az aktuálistól.

[2] A lokális keresés a célfüggvény bevezetésével könnyen kiterjeszthető egy kényszerkielégítési problémává. Ebben az esetben a hegymászásra és a szimulált lehűtésre használt összes technika alkalmazható a célfüggvény optimalizálására.

Created by XMLmind XSL-FO Converter.

5.9. ábra - Egy kétlépéses megoldás a 8-királynő problémára a MIN-KONFLIKTUSOK felhasználásával. Mindegyik fázisban egy királynőnek keresünk új oszlopot. A konfliktusok számát (esetünkben a támadó pozícióban lévő királynők számát) mindegyik négyzetben feltüntettük. Az algoritmus a királynőket a MIN-KONFLIKTUSOK négyzetre viszi, véletlenszerűen törve fel ezzel a kötéseket.



4. A problémák struktúrája

Ebben az alfejezetben azt vizsgáljuk meg, miként lehet a probléma *struktúráját,* ahogy azt a kényszergráf megmutatja, felhasználni a megoldások gyors keresésére. Az itt tárgyalt megközelítések java része nagyon általános és a kényszerkielégítési problémákon kívül más eseteknél is alkalmazható, például a valószínűségi következtetésnél. Végül is a részproblémákká történő dekompozíció az egyetlen mód, mellyel remélhetjük, hogy megbirkózhatunk a valósvilág-beli problémákkal. Ismét ránézve az 5.1. (b) ábrára, miközben a probléma struktúráját keressük, egy tény ötlik a szemünkbe ki: Tasmania nincs összeköttetésben a nagy szárazfölddel.[3] Napnál világosabb, hogy Tasmania színezése és a szárazföld színezése **független részproblémá**k (**independent subproblem**s): a szárazföld színezésének bármely megoldása és Tasmania színezésének bármely megoldása kombinálva egyben a teljes térkép színezésének is megoldása lesz. A függetlenségről könnyű megbizonyosodni a kényszergráf **összefüggő komponens**eit (**connected component**s) vizsgálva. Mindegyik komponens egy CSP_i részproblémának felel meg. Ha az S_i hozzárendelés egy megoldás a CSP_i részproblémára, akkor az $\cup_i S_i$ hozzárendelés megoldása lesz az $\cup_i CSP_i$-nek. Miért fontos ez? Gondoljunk bele a következőbe: tegyük fel, hogy minden egyes CSP_i-nek c változója van az n-ből, ahol c konstans. Ekkor n/c részproblémánk van, amelyek mindegyike legfeljebb d^c komplexitású. Tehát a teljes komplexitás $O(d^c n/c)$, ami lineáris n-ben, míg dekompozíció nélkül $O(d^n)$, ami exponenciális n-ben. Tegyük ezt még konkrétabbá: egy n = 80-nal jellemzett Boole CSP-t négy részproblémává osztva (c = 20) a megoldás futási idejét a legrosszabb esetben is az univerzum élethosszáról a másodperc törtrészére rövidítjük le.

Fontos

A teljesen független részproblémák ínyencfalatok, de ritkák. A legtöbb esetben a kényszerkielégítési problémák részproblémái kapcsolatban vannak egymással. A legegyszerűbb eset az, amikor a kényszergráf egy **fá**t alkot: bármely két változót legfeljebb egy út köt össze. Az 5.10. (a) ábra egy sematikus példát mutat erre.[4] Meg fogjuk mutatni, hogy bármely *fastruktúrájú kényszerkielégítési probléma megoldható a változók száma szerinti lineáris időben.* Az algoritmusnak a következő lépései vannak:

1. Válasszuk ki bármelyik változót a fa gyökércsomópontjául, és rendezzük a többi változót a gyökértől a levelekig úgy, hogy mindegyik csomópontot a sorrendezésben megelőzzön a szülője (lásd 5.10. (b) ábra). Címkézzük ezeket a változókat sorban X_1, …, X_n-nel. Most, a gyökércsomópontot leszámítva, mindegyik változónak pontosan egy szülője van.

2. Alkalmazzuk az élkonzisztenciát (X_i, X_j)-re, ahol X_i szülője X_j-nek (j pedig fusson visszafelé n-től 2-ig), és szükség esetén vegyünk ki értékeket a TARTOMÁNY[X_i]-ből.

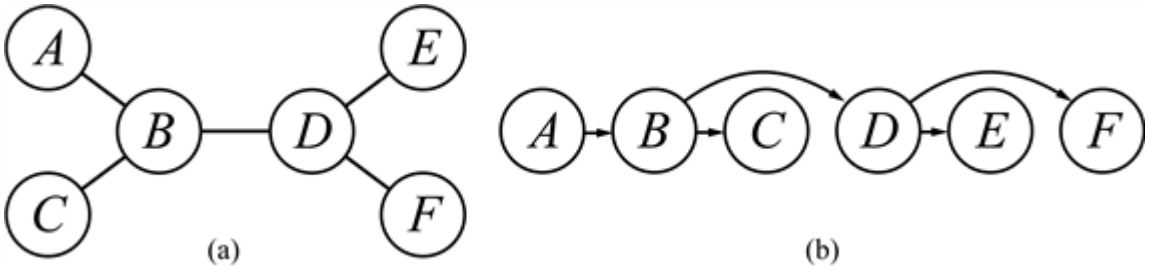
[3] Egy nagyon gondos térképész vagy egy tasmaniai patrióta ellenvethetné, hogy Tasmaniát nem lehet ugyanolyan színnel színezni, mint a legközelebbi szárazföldi szomszédját, nehogy úgy tűnjön, *mintha* annak az államnak a része lenne.

[4] Sajnos nagyon kevés olyan területe van a világnak (talán Celebesz ilyen), amelynek faszerkezetű térképe van. (Celebesz egy közelítőleg csillagstruktúrájú sziget Közép-Indonéziában – *A ford.*)

Created by XMLmind XSL-FO Converter.

3. Adjunk X_j-nek bármilyen, X_i hozzárendelt értékével konzisztens értéket ahol X_i szülője X_j-nek, és j 1-től n-ig halad.

5.10. ábra - (a) Egy faszerkezetű kényszerkielégítési probléma kényszergráfja. (b) Az *A* csomópont gyökérnek tekintésével konzisztens változók egy lineáris rendezése.



Két dolog érdemes említésre. Egyrészt a 2. lépés után a kényszerkielégítési probléma irány szerint élkonzisztens, tehát a 3. lépés hozzárendeléseiben nincsen szükség visszalépésre (lásd a *k*-konzisztencia tárgyalását a 2.2.2. szakasz - A kényszerek terjesztése részben). Másrészt, miután a 2. lépésben fordított sorrendben alkalmaztuk az élkonzisztencia-ellenőrzéseket, elértük az algoritmussal, hogy a törölt értékek ne veszélyeztessék a már feldolgozott élek konzisztenciáját. A teljes algoritmus $O(nd^2)$ időben fut.

Most, hogy fákra már van egy hatékony algoritmusunk, megvizsgálhatjuk, miként lehet az általánosabb kényszergráfokat valahogyan fákra *visszavezetni*. Alapvetően két mód van erre: az egyik a csomópontok eltávolításán, a másik a csomópontok összevonásán alapul.

Az első megközelítés úgy jár el, hogy néhány változónak értéket ad, a maradékok pedig fát fognak alkotni. Vegyük ismét az 5.11. (a) ábrán látható kényszergráfot az Ausztrália-példához. Ha törölni tudnánk Dél-Ausztráliát, akkor a gráf fává válhatna (ahogy az ábra (b) részén látható). Szerencsére meg tudjuk tenni ezt (a gráfban, nem a kontinensen) azzal, hogy *DA* értékét rögzítjük, és a többi változó tartományából töröljük azokat az értéket, melyek inkonzisztensek a *DA* számára választottal.

Most tehát hogy mind *DA*-t, mind a rá vonatkozó kényszereket eltávolítottuk, a kényszerkielégítési probléma bármely megoldása konzisztens lesz a *DA* számára választott értékkel. (Ez bináris kényszerkielégítési problémák esetén működik; a helyzet jóval bonyolultabb magasabb rendű kényszerek esetén.) Tehát a keletkező fa a fenti algoritmussal megoldható, és így az egész problémát is megoldottuk. Általános esetben persze (nem úgy, mint a térképszínezésnél) a *DA* számára választott érték lehet rossz is, és ekkor egyesével végig kell próbálgatni őket. Az általános algoritmus az alábbi:

1. Válasszunk ki egy *S* részhalmazt a VÁLTOZÓK[*csp*]-ből úgy, hogy a kényszergráf *S* eltávolítása után fa legyen. *S*-et **ciklikusság-vágóhalmaz**nak (**cycle cutset**) nevezzük.

2. *S* minden egyes változójának minden egyes, az *S*-re vonatkozó összes kényszert kielégítő lehetséges hozzárendelésére:

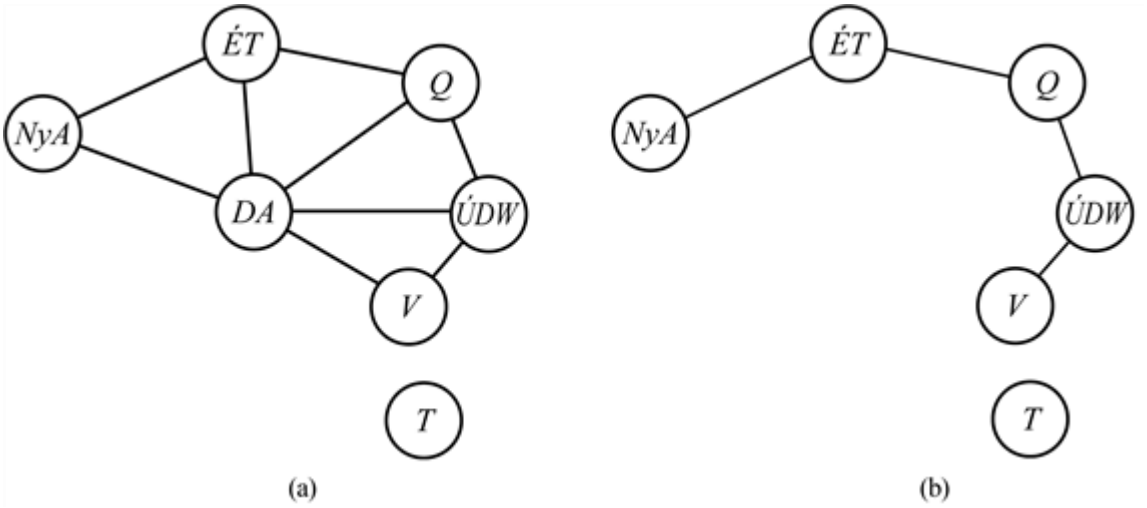
   a. vegyük ki a fennmaradó változók tartományaiból az *S* számára választott hozzárendeléssel inkonzisztens értékeket, és

   b. ha a fennmaradó kényszerkielégítési problémának van megoldása, akkor adjuk vissza ezt az *S* hozzárendelésével együtt.

Ha a ciklikusság-vágóhalmaz mérete c, akkor a teljes futási idő $O(d^c \cdot (n-c)d^2)$ lesz. Ha a gráf „közel fa”, akkor c kicsi lesz, a megtakarítás pedig tetemes a gondolkodás nélküli visszalépéses megoldáshoz képest. A legrosszabb esetben azonban c akár $(n-2)$ méretű is lehet. A *legkisebb* ciklikusság-vágóhalmaz megtalálása NP-nehéz probléma, de sok hatékony algoritmust ismerünk erre a feladatra. Az általános algoritmikus megközelítés a **vágóhalmaz-kondicionálás** (**cutset conditioning**); még találkozni fogunk ezzel a 14. fejezetben, ahol a valószínűségekről történő következtetésekhez használjuk.

5.11. ábra - (a) Az 5.1. ábrán szereplő eredeti kényszergráf. (b) A kényszergráf *DA* eltávolítását követően.

Created by XMLmind XSL-FO Converter.



A második megközelítés a kényszergráf több, egymással kapcsolatban lévő részproblémából álló **fadekompozíció** (**tree decomposition**) előállításán alapul. Mindegyik részproblémát függetlenül oldjuk meg, és a keletkező megoldásokat összekapcsoljuk. A legtöbb „oszd meg és uralkodj” típusú megközelítéshez hasonlóan ez is akkor működik, ha egyik részprobléma sem túl nagy. Egy fadekompozíciónak az alábbi követelményeket kell kielégítenie:

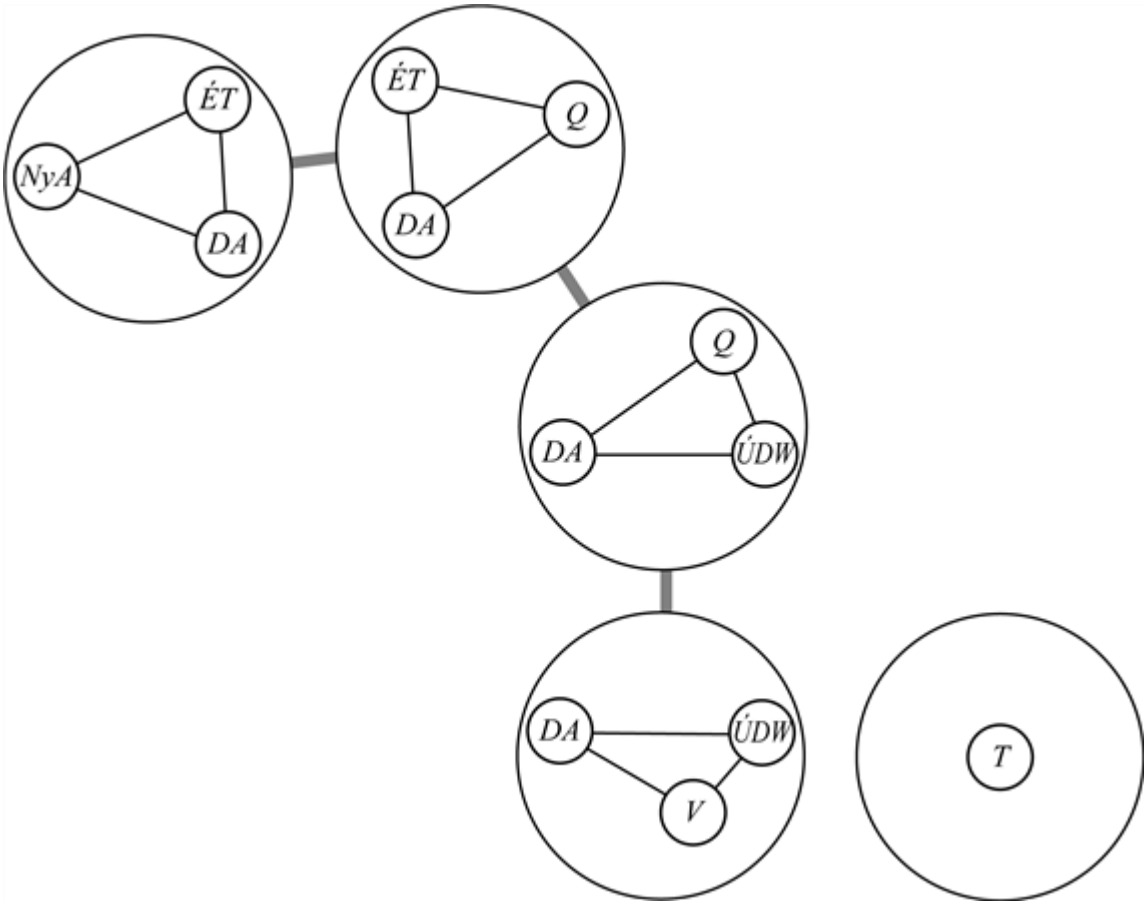
- Az eredeti probléma mindegyik változója szerepeljen a részproblémák legalább egyikében.
- Ha bármely két változót az eredeti problémában egy kényszer köt össze, akkor a részproblémák legalább egyikében együtt is elő kell fordulniuk (és természetesen a kényszernek is).
- Ha egy változó a fa két részproblémájában is előfordul, akkor a változónak az ezeket a részproblémákat összekötő út minden részproblémájában is elő kell fordulnia.

Az első két kikötés biztosítja, hogy az összes változó és kényszer előforduljon a dekompozícióban. A harmadik kikötés eléggé technikai ízűnek tűnik, de egyszerűen csak arról szól, hogy bármely adott változónak ugyanazzal az értékkel kell rendelkeznie az összes részproblémában, ahol előfordul – a részproblémákat a fában összekötő élek kényszerítik ezt ki. Például *DA* az 5.12. ábra mindegyik összekötött részproblémájában előfordul. Az olvasó az 5.11. ábra alapján igazolhatja, hogy van értelme ennek a dekompozíciónak.

Külön-külön megoldhatjuk az egyes részproblémákat, és ha bármelyiknek is nincs megoldása, akkor az egész problémának nincs. Ha az összes részproblémát meg tudjuk oldani, akkor a következők szerint megpróbálhatunk összeállítani egy globális megoldást. Először is tekintsünk minden egyes változót egy „megaváltozónak”, amelynek tartománya a részprobléma összes megoldásának halmaza. Például az 5.12. ábra bal szélső részproblémája az a térképszínezési probléma, amelynek három változója és – ezek szerint – hat megoldása van (ezek egyike az {*NyA = vörös, DA = kék, ÚT = zöld*}). Ezután a részproblémákat összekötő kényszereket meg tudjuk oldani a fákra adott fent bemutatott hatékony algoritmussal. A részproblémák közti kényszerek egyszerűen csak azért vannak, hogy a részproblémák megoldásai megegyezzenek az osztott változókban. Például, ha adott az {*NyA = vörös, DA = kék, ÚT = zöld*} megoldás az első részproblémára, akkor a következő részprobléma egyetlen konzisztens megoldása a {*DA = kék, ÚT = zöld, Q = vörös*} lehet.

5.12. ábra - Az 5.11. ábra (a) részén szereplő kényszergráf egy fa dekompozíciója

Created by XMLmind XSL-FO Converter.



Fontos

Egy adott kényszergráfnak több dekompozíciója is lehetséges; a dekompozíció kiválasztásakor az a cél, hogy a részproblémák a lehető legkisebbek legyenek. Egy gráf fadekompozíciójának **faszélessége** (**tree width**) eggyel kisebb, mint a legnagyobb részprobléma mérete; magának a gráfnak a faszélessége pedig definíció szerint a legkisebb faszélesség az összes fadekompozíciója között. Ha egy gráfnak w a faszélessége, és adott a megfelelő fadekompozíció, akkor a problémát meg lehet oldani $O(nd^{w+1})$ időben. Tehát *egy felülről korlátos faszélességű kényszergráffal rendelkező kényszerkielégítési probléma polinomiális időben megoldható*. Sajnálatos módon egy minimális faszélességű fadekompozíció megtalálása NP-nehéz probléma, de vannak olyan heurisztikus módszerek, amelyek jól működnek a gyakorlatban.

5. Összefoglalás

- A **kényszerkielégítési problémá**k (**CSP**-k) változókból állnak, melyekre kényszerek vonatkoznak. Nagyon sok fontos valósvilág-beli probléma írható le kényszerkielégítési problémaként. A kényszerkielégítési problémák struktúrája egy kényszergráffal reprezentálható.

- A **visszalépéses keresés** (**backtracking search**), a mélységi keresés egyik formája, a kényszerkielégítési problémák megoldásának gyakran alkalmazott eszköze.

- A **legkevesebb fennmaradó érték** (**least-remaining value**) **heurisztika** és a **fokszám-** (**degree-**) **heurisztika** tárgyterület-független módszerek annak eldöntésére, hogy a visszalépéses keresés során melyik változót válasszuk ki következőnek. A **legkevésbé korlátozó érték** (**least-constraining value**) **heurisztika** segítségül szolgálhat a változóértékek sorrendezésében.

- A visszalépéses algoritmus nagyban csökkenteni tudja a probléma elágazási tényezőjét a létrehozott részleges hozzárendelések következményeinek terjesztésével. Erre a legegyszerűbb módszer az **előrenéző ellenőrzés** (**forward checking**). Az **élkonzisztencia** (**arc consistency**) kikényszerítés egy jóval nagyobb teljesítőképességű technika, de tovább is tart a futása.

Created by XMLmind XSL-FO Converter.

- Visszalépésre akkor kerül sor, amikor egy változóhoz már nem találunk hozzárendelhető értéket. A **konfliktusvezérelt visszaugrás** (**conflict-directed backjumping**) közvetlenül a probléma okához ugrik vissza. A **min-konfliktusok** (**min-conflicts**) heurisztikát használó lokális keresést komoly sikerrel alkalmazták a kényszerkielégítési problémákra.

- A kényszerkielégítési probléma komplexitása szorosan kötődik a saját kényszergráfjának struktúrájához. A fastruktúrájú problémák megoldhatók lineáris időben. A **vágóhalmaz-kondicionálás** (**cutset conditioning**) az eredeti kényszerkielégítési problémát fastruktúrájúvá alakíthatja, és nagyon hatékonynak bizonyul, ha sikerül kis vágóhalmazt találnunk. A **fadekompozíció** (**tree decomposition**) technikák a kényszerkielégítési problémát részproblémák fájává alakíthatják, és hatékonyak, ha a kényszergráf **faszélessége** (**tree width**) kicsi.

5.1. Irodalmi és történeti megjegyzések

A kényszerkielégítéssel kapcsolatos legkorábbi munka a numerikus kényszerekhez kötődik. Az egész értékű egyenlőségi kényszereket Brahmagupta indiai matematikus tanulmányozta a 7. században. Ezeket gyakran **diofantoszi egyenlet**eknek is nevezik Diofantosz görög matematikus (kb. 200–284) nyomán, aki valójában a pozitív racionális számok esetét vizsgálta. A lineáris egyenlőségek változókiküszöböléssel történő megoldásának szisztematikus módszereit Gauss tanulmányozta (Gauss, 1829), a lineáris egyenlőtlenség alakú kényszerek megoldása Fourier-ig vezethető vissza (Fourier, 1827).

A véges tartományú kényszerkielégítési problémáknak hosszú történetük van. A **gráfszínezés** például (amelynek a térképszínezés csak egy speciális esete) a matematika régi problémáinak egyike. Biggs és társai (Biggs és társai, 1986) szerint a négyszín-sejtést (minden síkbeli gráf kiszínezhető legfeljebb négy színnel) először Francis Guthrie, de Morgan egyik tanítványa fogalmazta meg 1852-ben. A feladat ellenállt a megoldási kísérleteknek – annak ellenére, hogy néhányan publikációkban az ellenkezőjét állították –, mígnem Appel és Haken (Appel és Haken, 1977) előállt egy számítógépre is támaszkodó bizonyítással.

A kényszerkielégítési problémák egyes osztályai gyakran felmerültek a számítógép-tudomány történetében. Az egyik legkorábbi nagy hatású példa a SKETCHPAD rendszer volt (Sutherland, 1963), amely geometriai kényszereket oldott meg diagramokban, és a modern rajzolóprogramok és CAD-programok előfutárának tekinthető. A kényszerkielégítési problémák *általános problémaosztályként* történő azonosítása Ugo Montanari (Montanari, 1974) nevéhez fűződik. A magasabb rendű kényszerkielégítési problémák visszavezetése segédváltozók felvételével tisztán bináris esetre (lásd 5.11. feladat) eredetileg a 19. századi logikushoz, Charles Sanders Peirce-hez fűződik. A CSP-irodalomba Dechter (Dechter, 1990b) vezette be, majd Bacchus és Van Beek (Bacchus és van Beek, 1998) dolgozták ki. A megoldásokra vonatkozó preferenciákkal kiegészített kényszerkielégítési problémákat széles körben tanulmányozza az optimalizáció irodalma; lásd (Bistarelli és társai, 1997)-et a CSP-keretrendszer preferenciákat is megengedő általánosításáról. A vödör-eliminációs algoritmus (Dechter, 1999) szintén alkalmazható az optimalizációs problémákra.

A kényszerkielégítési problémák visszalépéses keresése Bitnertől és Reingoldtól (Bitner és Reingold, 1975) származik, noha ők az alapalgoritmust a 19. századig követik vissza. Bitner és Reingold az MRV-heurisztikát is bevezették (ők ezt *leginkább-korlátozott-érték* heurisztikának nevezték). Az MRV-heurisztika utáni eldöntetlen helyzetek megoldására Brelaz (Brelaz, 1979) a fokszám-heurisztikát alkalmazta. Az így létrejövő algoritmus, minden egyszerűsége ellenére, máig a leghatékonyabb tetszőleges gráfok *k*-színezésére. A *legkevésbé-korlátozó-érték* heurisztikát Haralick és Elliot javasolták (Haralick és Elliot, 1980).

A kényszerterjesztési módszereket Waltz (Waltz, 1975) sikere tette népszerűvé, amelyet a számítógépes látásnál felmerülő poliéder-élcímkézési problémán ért el. Waltz megmutatta, hogy sok probléma esetén a kényszerterjesztés teljesen kiküszöböli a visszalépést. Montanari (Montanari, 1974) bevezette a kényszerhálózat és az útvonalkonzisztencia-terjesztés fogalmát. Alan Mackworth (Mackworth, 1977) javasolta az AC-3 algoritmust az élkonzisztencia betartatására, csakúgy, mint annak általános lehetőségét, hogy valamilyen fokú konziszentencia-ellenőrzést építsünk be a visszalépéses algoritmusba. Az AC-4, egy jóval hatékonyabb algoritmus, melyet Mohr és Henderson fejlesztettek ki (Mohr és Henderson, 1984). Nem sokkal Mackworth cikkének megjelenése után a kutatók elkezdtek kísérletezni a konzisztencia-ellenőrzések költsége és a kereséslevágásban jelentkező előny csereviszonyának feltérképezésével. Haralick és Elliot (Haralick és Elliot, 1980) a McGregor által leírt (McGregor, 1979) minimális előrenéző ellenőrzés mellett álltak ki, míg Gaschnig (Gaschnig, 1979) minden egyes változó-hozzárendelés után élkonzisztencia-ellenőrzéseket javasolt (ezt az algoritmust nevezte később Sabin és Freuder (Sabin és Freuder, 1994) MAC-nak). Az utóbbi cikk valamennyire meggyőző bizonyítékot hoz fel amellett, hogy nehezebb problémáknál kifizetődik a teljes élkonzisztencia-ellenőrzés. Freuder (Freuder 1978, 1982) megvizsgálta a *k*-konzisztenciát és ennek kapcsolatát a

Created by XMLmind XSL-FO Converter.

kényszerkielégítési problémák megoldásának komplexitásával. Apt (Apt, 1999) egy általános algoritmikus keretrendszert mutat be a konzisztenciaterjesztési algoritmusok vizsgálatára.

A magasabb rendű kényszerek kezelésének külön módszerei elsősorban a **kényszerlogikai-programozás** (**constraint logic programming**) keretein belül alakultak ki. Marriott és Stuckey (Marriott és Stuckey, 1998) nagyszerű összefoglalást nyújt erről a kutatási területről. A *MindKül* kényszert Regin (Regin, 1994) tanulmányozta. Az alsó és felső határokból álló kényszereket Van Hentenryck és társai vezették be a kényszerlogikai programozásba (Van Hentenryck és társai,1998).

Az alapvető visszaugró módszer John Gaschnigtől (Gaschnig, 1977; 1979) származik. Kondrak és van Beek (Kondrak és Van Beek, 1997) megmutatták, hogy ezt az algoritmust lényegében magában foglalja az előrenéző ellenőrzés. A konfliktusvezérelt visszaugrást Prosser (Prosser, 1993) alakította ki. Az intelligens visszalépés legáltalánosabb és legerősebb formáját tulajdonképpen már nagyon korán kifejlesztette Stallman és Sussman (Stallman és Sussman, 1977). Technikájuk, a **függőségvezérelt visszalépés** (**dependency-directed backtracking**) az **igazság-karbantartó rendszer**ek (**truth maintenance system**s) kifejlesztéséhez vezetett (Doyle, 1979), amelyekkel a 10.8. alfejezetben foglalkozunk. A két terület közti kapcsolatot De Kleer (De Kleer, 1989) vizsgálta.

Stallman és Sussman munkája a **kényszerfeljegyzés** (**constraint recording**) elképzelését is bevezette: a keresés által elért részleges eredményeket elmentjük és a keresés során később felhasználjuk. Ezt az elképzelést a visszalépéses keresésbe formális módon Dechter (Dechter, 1990a) vezette be. A **visszajegyzés** (**backmarking**) (Gaschnig, 1979) egy különlegesen egyszerű módszer, amelyben a konzisztens és inkonzisztens páronkénti hozzárendeléseket elmentjük, hogy elkerüljük a kényszerek későbbi újraellenőrzését. A visszajegyzés ötvözhető a konfliktusvezérelt visszaugrással; Kondrak és Van Beek (Kondrak és Van Beek, 1997) bemutatnak egy hibrid algoritmust, amely bizonyíthatóan tartalmazza mindkét külön módszert. A **dinamikus visszalépés** (**dynamic backtracking**) (Ginsberg, 1993) megőrzi a változók későbbi részhalmazaiból származó sikeres parciális hozzárendeléseket, amikor egy olyan korábbi választási pontra ugrik vissza, amely a későbbi sikert nem teszi érvénytelenné.

A kényszerkielégítési problémák lokális keresését Kirkpatrick és társai (Kirkpatrick és társai, 1983) **szimulált lehűtés**ről (**simulated annealing**) (lásd 4. fejezet) szóló munkája tette népszerűvé, és ezt széles körben alkalmazták az ütemezési problémáknál. A *min-konfliktusok* heurisztikát először Gu (Gu, 1989) javasolta, és tőle függetlenül Minton és társai (Minton és társai, 1992) is kifejlesztették. Sosic és Gu (Sosic és Gu, 1994) megmutatta, hogyan lehet ennek a heurisztikának az alkalmazásával a 3 000 000-királynő problémát kevesebb, mint egy perc alatt megoldani. A bámulatos siker, amit a min-konfliktusokat használó lokális keresés ért el az *n*-királynő problémában, a „könnyű” és a „nehéz” problémák természetének és elterjedtségének újraértékeléséhez vezetett. Peter Cheesman és társai (Cheesman és társai, 1991) feltérképezték a véletlenszerűen generált kényszerkielégítési problémák nehézségét, és azt találták, hogy majdnem minden ilyen probléma vagy triviálisan könnyű, vagy megoldhatatlan. Csak akkor találunk „nehéz” probléma példányokat, ha a problémagenerátor paramétereit egy bizonyos szűk tartományba állítjuk be, melyen belül a problémák közelítőleg fele megoldható. Ezzel a jelenséggel a 7. fejezetben foglalkozunk részletesebben.

A kényszerkielégítési problémák struktúrájának és nehézségének kapcsolatával foglalkozó kutatást Freuder (Freuder, 1985) indította el, aki megmutatta, hogy az élkonzisztens fák esetében a keresés visszalépések nélkül fut le. Egy hasonló eredmény az aciklikus hipergráfokra való kiterjesztésével együtt jött létre az adatbázisokkal foglalkozó kutatói közösségben (Beeri és társai, 1983). Ezen cikkek publikálása óta komoly haladás történt a kényszergráf struktúrája és a kényszerkielégítési probléma megoldási komplexitásának kapcsolatát illetően. A faszélesség fogalmát a gráfelmélettel foglalkozó Robertson és Seymour (Robertson és Seymour, 1986) vezették be. Freuder munkásságára építve Dechter és Pearl (Dechter és Pearl, 1987, 1989) ugyanezt a fogalmat – amit ők **indukált szélesség**nek (**induced width**) hívtak – alkalmazták a kényszerkielégítési problémákra és kifejlesztették az 5.4. alfejezetben felvázolt fadekompozíciót. Az adatbázis-elméletre és erre az eredményre alapozva Gottlob és társai (Gottlob és társai, 1999a, 1999b) kialakították a kényszerkielégítési probléma hipergráfként történő felfogásán alapuló **hiperfaszélesség** (**hypertree width**) fogalmat. Annak megmutatásán túl, hogy minden w szélességű hiperfa-CSP megoldható $O(n^{w+1}\log n)$ időben, azt is bebizonyították, hogy a hiperfaszélesség az összes korábbi „szélesség”-mértéket magában foglalja (abban az értelemben, hogy bizonyos esetekben a hiperfaszélesség korlátozott, bizonyos esetekben pedig nem).

Sok jó áttekintés létezik a CSP-technikákhoz, például Kumar (Kumar, 1992), Dechter és Frost (Dechter és Frost, 1999) és Bartak (Bartak, 2001); továbbá a Dechter-től (Dechter, 1992) és Mackworthtől (Mackworth, 1992) származó enciklopédiacikkek. Pearson és Jeavons (Pearson és Jeavons, 1997) a könnyen kezelhető CSP-osztályokat veszik számba, beleértve mind a strukturális dekompozíciós módszereket, mind a maguknak a

Created by XMLmind XSL-FO Converter.

tartományoknak vagy a kényszereknek a tulajdonságaira támaszkodó módszereket. Kondrak és Van Beek (Kondrak és Van Beek, 1997) a visszalépéses keresési algoritmusok analitikus áttekintését adják, Bacchus és Van Run (Bacchus és Van Run, 1998) pedig empirikusabb képet festenek. Tsang (Tsang, 1993), valamint Marriott és Stuckey (Marriott és Stuckey, 1998) szövegei jóval mélyebbre hatolnak a témában, mint azt a fejezet korlátai számunkra lehetővé tették. Sok érdekes alkalmazást mutat be a Freuder és Mackworth szerkesztésében megjelent gyűjtemény (Freuder és Mackworth, 1994). Kényszerkielégítéssel foglalkozó cikkek rendszeresen jelennek meg az *Artificial Intelligence*-ben és egy specialistáknak szóló újságban, a *Constraints*ben. A legfőbb konferenciakiadvány az International Conference on Principles and Practice of Constraint Programming, amit gyakran *CP*-nek hívnak.

5.2. Feladatok

5.1.

Fogalmazza meg a saját szavaival a kényszerkielégítési problémák, a kényszerek, a visszalépéses keresés, az élkonzisztencia, a visszaugrás és a min-konfliktusok definícióját.

5.2.

Hány megoldása van az 5.1. ábra térképszínezési problémájának?

5.3.

Magyarázza el, miért jó heurisztika a *leginkább* korlátozott változót és a *legkevésbé* korlátozott értéket választani a kényszerkielégítési probléma megoldásának keresése közben.

5.4.

Tekintsük a keresztrejtvények készítésének (nem megoldásának) problémáját:[5] szavakat kell illeszteni egy négyszögletes rácsba. A rács, amely része a probléma specifikációjának, megadja, hogy mely négyzetek legyenek üresek és melyek sötétek. Tegyük fel, hogy adott a szavak egy listája (például egy szótár), és az a feladat, hogy az üres négyzeteket a lista tetszőleges részhalmazát használva kitöltsük. Fogalmazza meg pontosan ezt a problémát kétféle módon:

a. Mint általános keresési problémát. Válasszon egy megfelelő keresési algoritmust, és specifikálja a heurisztikus függvényt (amennyiben elképzelése szerint szüksége van rá). A fehér kockákba egyszerre egy betűt vagy egész szavakat érdemes-e beírni?

b. Mint kényszerkielégítési problémát. A változók betűk vagy szavak legyenek? Melyik megfogalmazást tartja jobbnak? Miért?

5.5.

Adjon precíz megfogalmazást az alábbiakra mint kényszerkielégítési problémákra:

a. **Négyszögletes kirakó**: találjon nemátfedő helyeket egy nagy négyszögben kisebb négyszögek számára.

b. **Órarend-ütemezés**: adott számú professzor és terem van, valamint rögzített az órarendi órák listája is a lehetséges időablakokkal együtt. Mindegyik professzorhoz adott az általa tartott órák halmaza.

5.6.

Oldja meg az 5.2. ábra betűrejtvényét kézzel, visszalépéses kereséssel, előrenéző ellenőrzéssel, valamint az MRV-, illetve a legkevésbé korlátozó érték heurisztikával.

5.7.

Az 5.5. ábra a különböző algoritmusokat az *n*-királynő problémán teszteli. Próbálja meg ugyanezt egy véletlenszerűen generált térképszínezési problémával is: osszon el az egységsíkon véletlenszerűen *n* pontot,

[5] Ginsberg és társai (Ginsberg és társai, 1990) több módszert tárgyalnak keresztrejtvények készítésére. Littman és társai pedig a nehezebb problémát, megoldásukat veszik célba (Littman és társai, 1999).

Created by XMLmind XSL-FO Converter.

válasszon ki véletlenszerűen egy *X* pontot, kösse *X*-et a legközelebbi olyan *Y* ponthoz, amelyikkel *X* még nincs összekötve, és a vonal semelyik más vonalat nem metsz; ismételje a fenti lépést mindaddig, amíg újabb összeköttetés már nem lehetséges. Számítsa ki a teljesítménytáblázatot a legnagyobb *n*-re, amire csak tudja (mind $d = 3$-at mind, $d = 4$-et használva). Fűzzön magyarázatokat a kapott eredményhez.

5.8.

Az AC-3 algoritmus felhasználásával mutassa meg, hogy az élkonzisztencia alkalmas arra, hogy kimutassa az 5.1. ábra {*NyA = vörös, V = kék*} parciális hozzárendelésének inkonzisztenciáját.

5.9.

Mi a fastruktúrájú kényszerkielégítési problémán futtatott AC-3 legrosszabb esetbeli komplexitása?

5.10.

Az AC-3 visszarak a sorba *minden* (X_k, X_i) élet, amikor X_i tartományából *bármely* értéket töröltek, akkor is, ha X_k minden értéke konzisztens X_i több fennmaradó értékével. Tegyük fel, hogy minden egyes (X_k, X_i) élhez nyilvántartjuk a fennmaradó X_i értékek számát, amelyek az X_k minden egyes értékével konzisztensek. Magyarázza el, hogyan lehet hatékonyan frissíteni ezeket az értékeket, és hogyan lehet ennek segítségével az élkonzisztenciát $O(n^2d^2)$ lépésben elérni.

5.11.

Mutassa meg, hogy egy ternáris kényszer, mint például az „*A + B = C*” egy segédváltozó bevezetésével három bináris kényszerré alakítható. Feltételezheti, hogy a tartományok végesek. (*Segítség:* gondoljon egy olyan új változóra, amelynek értékei más értékekből álló párok, és gondoljon olyan kényszerekre, mint „*X* az első eleme az *Y* párnak”.) Ezután mutassa meg, hogyan lehet hasonlóan kezelni a háromnál több változót tartalmazó kényszereket. Végül mutassa meg, miként lehet kiküszöbölni az unáris kényszereket a változók tartományának megváltoztatásával. Ez teljessé teszi annak bizonyítását, hogy bármely CSP átalakítható olyan problémákká, melyek csak bináris kényszereket tartalmazhatnak.

5.12.

Olvasnivaló. Tegyük fel, hogy ismerjük egy gráfról, hogy van egy legfeljebb *k* csomópontot tartalmazó ciklikusság-vágóhalmaza. Írjon le *n* változós CSP-k esetén egy egyszerű algoritmust a minimális ciklikusság-vágóhalmaz megkeresésére, ahol a futási idő maximuma $O(n^k)$. Végezzen irodalomkutatást olyan vágóhalmaz-keresési eljárások után, amelyek a vágóhalmaz méretében közelítőleg polinomiális időben találnak közelítőleg minimális ciklikusság-vágóhalmazt. Praktikussá teszi az ilyen módszerek létezése a ciklikusság-vágóhalmaz módszereket?

5.13.

Tekintsük a következő logikai rejtvényt: Öt különböző színű házban öt különböző nemzetiségű személy él, és mindegyikük más márkájú cigarettát, más italt és más háziállatot szeret. Az alábbi tények alapján a megválaszolandó kérdés a következő: „Hol lakik a zebra, és melyik házban isznak vizet?”

Az angol a vörös házban lakik.

A spanyolnak kutyája van.

A norvég balról az első házban lakik.

A Kools cigarettát a sárga házban szívják.

A Chesterfieldset szívó ember a rókás ház mellett lakik.

A norvég a kék ház mellett lakik.

A Winstont szívó ember kígyókat tart.

A Lucky Strike-ot szívó narancslevet iszik.

Created by XMLmind XSL-FO Converter.

Az ukrán teát iszik.

A japán Parliamentset szív.

A Koolsot abban a házban szívják, amely mellett lovat tartanak.

Kávét a zöld házban isznak.

A zöld ház közvetlenül jobbra (Ön felől nézve) van az elefántcsontszínű háztól.

Tejet a középső házban isznak.

Vizsgálja meg a probléma különböző CSP-reprezentációit. Milyen okokból részesítené előnyben az egyiket a másikkal szemben?

Created by XMLmind XSL-FO Converter.

6. fejezet - Keresés ellenséges környezetben

Ebben a fejezetben azokat a problémákat vizsgáljuk meg, amelyek akkor merülnek fel, amikor egy ágens előre tervez ellenséges ágenseket tartalmazó világban.

1. Kétszemélyes játékok[1]

A 2. fejezetben **többágenses környezet**eket (**multiagent environment**s) vezettünk be, ahol minden ágensnek számolnia kell más ágensek cselekvéseivel és azzal is, hogy azok hogyan befolyásolják a jólétét. Más ágensek nem megjósolható viselkedése számos lehetséges **eshetőség**et (**contingencies**) visz be az ágens problémamegoldásába, ahogy ezzel a 3. fejezetben foglalkoztunk. A 2. fejezetben bevezettük a több ágensből álló **kooperatív** (**cooperative**) és **verseny**- (**competitive**) környezeteket is. A versenykörnyezetek, ahol az ágensek céljai konfliktusban vannak, elvezetnek az **ellenségek melletti keresés**hez (**adversarial search**) – amit sokszor **kétszemélyes játék**oknak (**game**s) nevezünk.

A matematikai **játékelmélet** (**game theory**) a gazdaságtan egyik ága, mely a többágenses környezeteket játéknak tekinti, feltéve, hogy egy-egy ágens hatása másokra „szignifikáns”, függetlenül attól, hogy az ágensek kooperatívak vagy versengők.[2] Az MI-ben a „játékok” általában igen specializáltak – amit a játékelméleti szakemberek determinisztikus, váltott lépésű, kétszemélyes, **zérusösszegű teljes információjú játék**oknak (**zero-sum game**s **of perfect information**) neveznek. A mi nyelvezetünkben ez azt jelenti, hogy két ágens helyezkedik el egy determinisztikus és teljesen megfigyelhető környezetben, a cselekvéseik váltják egymást, és a játék végén a hasznosságértékeik mindig azonosak és ellentétes előjelűek. A sakkban például, ha az egyik játékos győz (+1), akkor a másik szükségszerűen veszít (–1). Éppen a hasznosságértékekben tapasztalt ellentéttől lesz a helyzet ellenséges. Ebben a fejezetben röviden foglalkozunk a többjátékos játékokkal, a nem zérusösszegű játékokkal és a sztochasztikus játékokkal, de a tényleges játékelmélettel csak a 17. fejezetben fogunk foglalkozni.

A játékok a civilizáció kezdete óta – néha már ijesztő mértékben – foglalkoztatják az emberek intellektuális képességeit. A játékok absztrakt természetük miatt vonzó területet jelentenek az MI-kutatók számára. Egy játék állását könnyű reprezentálni, és az ágensek képességei általában kisszámú, jól definiált eredményre vezető cselekvésre korlátozódnak. Az olyan fizikai játékoknak, mint a krikett vagy a jégkorong, sokkal bonyolultabb a leírása. Sokkal több lehetséges cselekvéssel rendelkeznek, és a cselekvések legális voltát éppenséggel nem túl precíz szabályok határozzák meg. A robotfutballt kivéve e fizikai játékok az MI-közösségben sok érdeklődést nem keltettek.

A kétszemélyes játékok az egyik legrégebbi, az MI által vizsgált területet jelentik. 1950-ben, alighogy a számítógépek programozhatóvá váltak, Konrad Zuse (az első programozható számítógép és az első programozási nyelv megalkotója), Claude Shannon (az információelmélet atyja), Norbert Wiener (a korszerű szabályozáselmélet megteremtője) és Alan Turing elkezdtek sakkprogramokkal foglalkozni. Azóta a játékok színvonala sokat fejlődött, addig a szintig, hogy a gépek túlszárnyalták az embert dámajátékban és Othellóban, megverték (bár nem minden alkalommal) ostáblában és sakkban az emberi bajnokokat, és versenyképesek más játékokban is. Kivétel a go, ahol a számítógép csak amatőr szinten játszik.

A 3. fejezetben tanulmányozott játékproblémákkal ellentétben a kétszemélyes játékok azért érdekesek, *mert* nagyon nehéz őket megoldani. A sakknál például az átlagos elágazási tényező 35, és egy játék során gyakran előfordul, hogy mindkét fél 50–50 lépést is megtesz, vagyis a keresési fának 35^{100}, illetve 10^{154} csomópontja van (bár a keresési gráfnak ebből „csak” 10^{40} különböző csomópontja lesz). A játékok, éppúgy, mint a valós világ, azt a képességet igénylik, hogy *valamilyen* döntést hozzunk, akkor is, ha az *optimális* döntés kiszámítása kivitelezhetetlen. A játékok nagyon komolyan büntetik a rossz hatékonyságot. Míg az A* algoritmus azon implementációja, ami csak fele olyan hatékony, egyszerűen csak kétszer annyi ideig fut, hogy megkapja az eredményt, addig egy olyan sakkprogramot, amely feleolyan hatékonyan gazdálkodik az idejével, feltéve, hogy minden más szempontból egy másik implementációval azonos, valószínűleg a földbe döngölnének. A

[1] Annak ellenére, hogy az angol „game” szó magyar szó szerinti fordítása „játék”, úgy éreztük, hogy az ilyen fordítás magyarul összemossa a „game” és a „toy” közötti különbséget. A „kétszemélyes játékok” szóhasználat alkalmas kompromisszum, annak ellenére, hogy az ebben a fejezetben tárgyalt módszerek olyan „kétszemélyes játékokra” is kiterjeszthetők, ahol több játékos van. (*A ford.*)

[2] A nagyon sok ágensből álló környezeteket jobb **gazdaság**oknak (**economies**) és nem játékoknak nézni.

Created by XMLmind XSL-FO Converter.

játékelméleti kutatás ezért számos érdekes ötlethez vezetett, hogy a rendelkezésre álló időt hogyan használjuk a lehető legjobb módon.

A probléma tárgyalását az elméletileg lehetséges legjobb lépés definíciójával és ennek egy keresőalgoritmusával kezdjük. Ezek után olyan technikákat nézünk meg, amelyek korlátozott idő alatt is alkalmasak egy jó lépés megválasztására. A **nyesés** vagy **metszés** (**pruning**) lehetővé teszi számunkra, hogy figyelmen kívül hagyjuk a keresési fa azon részeit, amelyek nincsenek befolyással a végső választásra. A heurisztikus **kiértékelő függvény**ek (**evaluation function**s) lehetővé teszik, hogy kimerítő keresés nélkül meg tudjuk becsülni egy adott állapot valódi hasznosságát. A 6.5. alfejezet olyan kétszemélyes játékokat tárgyal, amelyekben a véletlen is megjelenik, mint például az ostábla.[3] Foglalkozunk a briddzsel is, amely tartalmazza a **hiányos információ** (**imperfect information**) elemeit, hiszen egy-egy játékos számára az összes kártya nem ismert. Megnézzük végül, hogy a jelenlegi legfejlettebb játékprogramok hogyan győzik le az erős emberi ellenfeleket, és milyenek a jövőbeli trendek.

2. Optimális döntések kétszemélyes játékokban

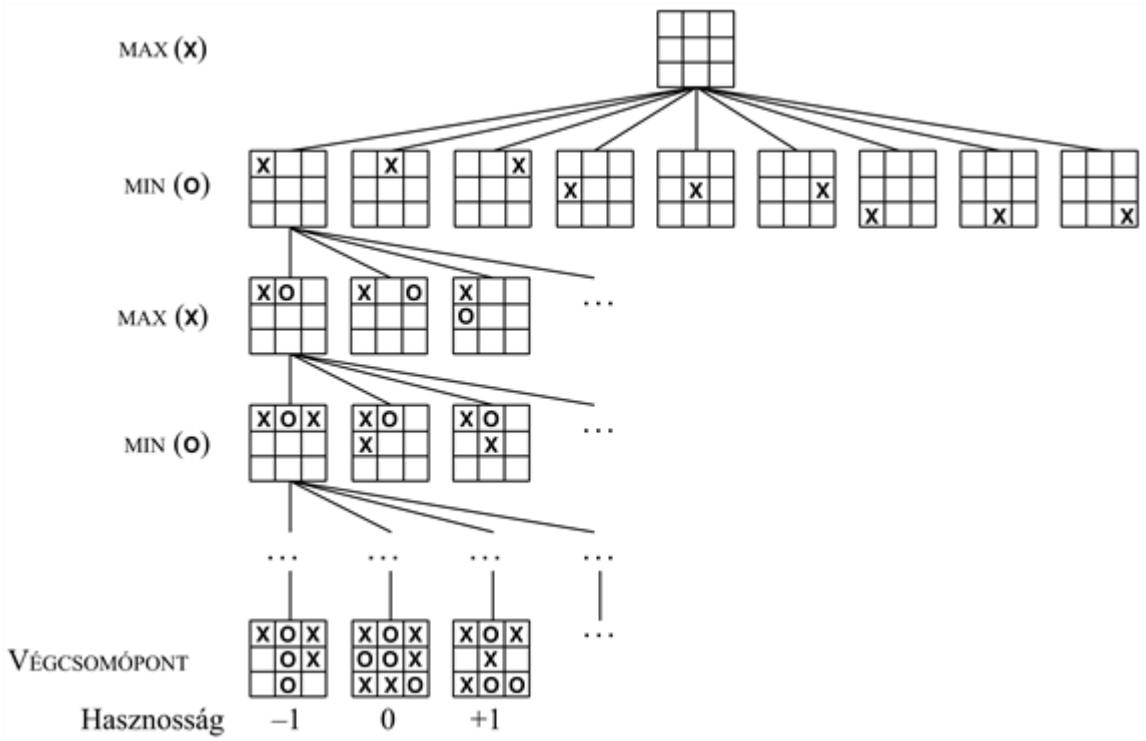
A kétszemélyes játékokkal fogunk foglalkozni, ahol a két játékost – a hamarosan nyilvánvalóvá váló okból – `MAX`-nak és `MIN`-nek fogjuk hívni. `MAX` lép először, majd a játékosok felváltva lépnek, amíg a játék véget nem ér. A játék végén a győztes játékos pontokat kap (vagy néha a vesztes kap büntetőpontokat). A játékot formálisan egyfajta keresési problémaként lehet definiálni az alábbi komponensekkel:

- A **kiinduló állapot** (**initial state**), ami magában foglalja a táblaállást, valamint azt, hogy ki fog lépni.
- Egy **állapotátmenet-függvény** (**successor function**), amely (*lépés*, *állapot*) párok listájával tér vissza, megadva a legális lépéseket és az azokból következő állapotokat.
- Egy **végteszt** (**terminal test**), ami meghatározza, hogy a játéknak mikor van vége. Azok az állapotok, ahol a játék befejeződött, a **végállapot**ok (**terminal state**s).
- Egy **hasznosságfüggvény** (**utility function**, amit **nyereségfüggvény**nek – **payoff function** – is neveznek) a játék végeredményéhez egy számértéket rendel. A sakkban a végeredmény győzelem, vereség vagy döntetlen lehet, amit a +1, –1 és 0 értékekkel ábrázolhatunk. Néhány játék ennél több végeredményre vezethet. Például az ostáblában a nyereség +192 és –192 között változhat. Ebben a fejezetben főleg a zérusösszegű játékokkal foglalkozunk, bár a nem zérusösszegű játékokat is megemlítjük.

6.1. ábra - A 3 × 3-as amőbajáték (részleges) keresési fája. A legfelső csomópont a kiinduló állapot. `MAX` lép először, egy `X`-et téve valamelyik üres négyzetbe. A keresési fa egy részét mutatjuk, `MIN` (`O`) és `MAX` váltakozó, egymást követő lépéseit megadva, amíg el nem érjük a végállapotokat, melyekhez a játék szabályai szerint lehet hasznossági értékeket hozzárendelni.

[3] Az ostábla (backgammon) Magyarországon nem annyira elterjedt, bár számos vidéken ismert. A lényege, hogy egy speciális táblán, felváltva lépve, a bábukat az ellenfél térfelére juttassuk. Fő érdekessége, ami a nehézségek forrása is egyben, hogy a legális lépések mindenkori választéka véletlenszerűen – egy kockadobásnak megfelelően – alakul. (*A ford.*)

Created by XMLmind XSL-FO Converter.



A kezdeti állapot és mindkét fél legális lépései a játék **játékfá**ját (**game tree**) definiálják. A 6.1. ábra a 3 × 3-as amőbajáték keresési fájának egy részét mutatja. Kezdeti állapotban `MAX`-nak kilenc lehetséges lépése van. A játék során `MAX` és `MIN` felváltva tesznek `X`-et illetve `O`-t, míg egy végállapotnak megfelelő levélcsomópontba el nem jutnak, ahol az egyik játékosnak egy sorban, egy oszlopban vagy az egy átló mentén három `O`-ja vagy `X`-e lesz, vagy minden négyzet ki lesz töltve. Az egyes levélcsomópontok alatt található számok a végcsomópontnak a `MAX` szempontjából mért hasznosságát jelölik. A nagy értékekről feltételezzük, hogy jók `MAX` számára és rosszak `MIN` számára (amiből a játékosok neve is ered). `MAX` feladata, hogy a játékfát (és főleg a végállapotok hasznosságát) a legjobb lépés meghatározására használja fel.

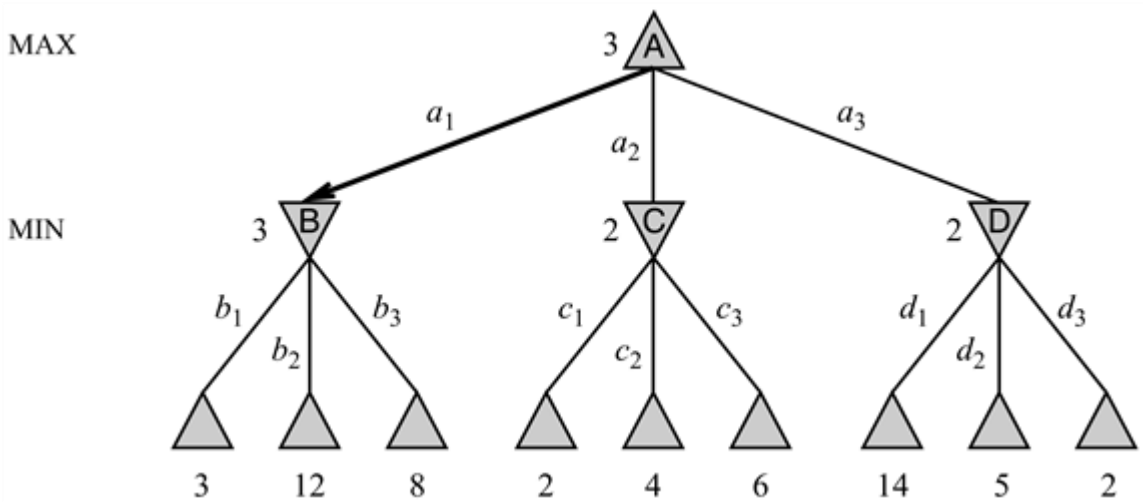
2.1. Optimális stratégiák

Egy normális keresési problémánál, az optimális megoldás nem lenne más, mint a célállapothoz vezető lépések szekvenciája – azaz egy olyan végállapothoz vezető lépésszekvencia, amely a győzelmet jelenti. Egy játékban azonban `MIN`-nek is van beleszólása a dologba. Ezért `MAX`-nak egy olyan **stratégiá**t (**strategy**) kell találnia, amely meghatározza `MAX` lépését a kezdeti állapotban, majd a `MIN` lehetséges válaszaiból keletkező állapotokban, majd ismét `MAX` lépéseit a `MIN` erre vonatkozó lehetséges válaszaiból keletkező állapotokban és így tovább. Nagyjából azt lehet mondani, hogy egy optimális stratégia olyan kimenetelekhez vezet, amelyek legalább olyan jók, mintha bármilyen más stratégiával egy tévedhetetlen opponens ellen játszanánk. Először megmutatjuk, hogyan kell megkeresni az optimális stratégiát, bár ennek kiszámítására `MAX`-nak általában nem lesz elegendő ideje az amőbánál bonyolultabb játékokban.

Még egy olyan egyszerű játék, mint az amőba is túl bonyolult ahhoz, hogy megmutassuk a teljes keresési fát, ezért áttérünk a 6.2. ábrán látható abszolút triviális játékra. `MAX` lehetséges lépéseit a_1-gyel, a_2-vel és a_3-mal címkéztük meg. Az a_1-re `MIN` lehetséges válaszait a b_1, b_2, b_3 stb. jelölik. Ez a konkrét játék `MAX` és `MIN` egy-egy lépése után véget ér. (A játékok nyelvén azt mondjuk, hogy ez a fa egy lépés mély és két fél lépésből vagy **lépésváltás**ból (**ply**) áll.) A végcsomópontok hasznossága ebben a játékban 2 és 14 közé esik.

6.2. ábra - Egy lépésváltásos játékfa. A Δ csomópontok `MAX` lépéseit, míg a ∇ csomópontok `MIN` lépéseit jelölik. A végcsomópontok a hasznossági függvénnyel számított hasznossági értékeket `MAX` szemszögéből mutatják, míg a többi csomópontnál a minimax értékét jelöltük be. `MAX` legjobb lépése a gyökérben a_1, mert ez vezet a legmagasabb minimax értékű követőhöz. `MIN` legjobb válasza b_1, mert ez vezet a minimális minimax értékű követőhöz.

Created by XMLmind XSL-FO Converter.



Adott játékfa mellett az optimális stratégia meghatározásához az egyes csomópontok **minimax értéké**t kell megvizsgálni, amit `MINIMAX-ÉRTÉK` (*n*)-ként írunk le. Egy csomópont minimax értéke a csomópont hasznossága `MAX` szemszögéből, feltéve, hogy innen kezdve egészen a játék befejezéséig *mindkét játékos optimálisan lép*. Egy végállapot minimax értéke természetesen a saját hasznossága. Továbbá, adott minimax értékek mellett, `MAX` szeretne a maximális értékű, `MIN` pedig a minimális értékű állapotba jutni. Rendelkezünk tehát az alábbi függvénnyel:

`MINIMAX-ÉRTÉK` (*n*) =

`HASZNOSSÁG` (*n*)	ha *n* egy végállapot,
$\max_{s \in Követők(n)}$ `MINIMAX-ÉRTÉK`(*s*)	ha *n* egy `MAX` csomópont,
$\min_{s \in Követők(n)}$ `MINIMAX-ÉRTÉK`(*s*)	ha *n* egy `MIN` csomópont.

Alkalmazzuk ezeket a definíciókat a 6.2. ábrán látható játékfára. Az ábra alján lévő célállapotokat már felcímkéztük hasznossági értékükkel. Az első, *B* címkéjű, `MIN` csomópontnak három követője van 3, 12 és 8 értékkel, a *B* csomópont minimax értéke tehát 3. Hasonlóan a két másik `MIN` csomópontnak 2 a minimax értéke. A gyökér egy `MAX` csomópont; a követőinek minimax értékei 3, 2, és 2, így a gyökér minimax értéke 3. Azonosíthatjuk a gyökér **minimax döntésé**t (**minimax decision**) is. `MAX` számára az a_1 cselekvés az optimális választás, mert maximális értékű követőhöz vezet.

Ezen optimális játékdefiníció `MAX` számára feltételezi, hogy `MIN` is optimálisan játszik – hiszen a dolgok kimenetelét `MAX` számára a legrosszabb esetre kivetítve maximalizálja. Mi van azonban, ha `MIN` nem játszik optimálisan? Ekkor könnyű belátni (6.2. feladat), hogy `MAX` még jobban jár. Szuboptimális ellenféllel szemben sok stratégia elképzelhető, amely az optimálisnál jobb, azonban ezek a stratégiák rosszabbnak fognak bizonyulni optimális ellenfelekkel szemben.

2.2. A minimax algoritmus

A **minimax algoritmus** (**minimax algorithm**) (6.3. ábra) az optimális döntést az aktuális állapotból számítja ki, felhasználva az egyes követő állapotok minimax értékeinek kiszámítására a definiáló egyenletekből közvetlenül származtatott, egyszerű rekurzív formulát. A rekurzió egészen a falevelekig folytatódik, majd a minimax értékeket a fa mentén **visszafelé terjeszt**jük (**back-up**), ahogy a rekurzió visszalép. A 6.2. ábrán például az algoritmus először rekurzív módon leereszkedik a három bal alsó csomóponthoz, a `HASZNOSSÁG` függvénnyel kiszámítva, hogy az értékek rendre 3, 12, és 8. Majd az algoritmus előveszi ezen értékek minimumát, azaz 3-t, és ezt adja vissza, ahogy a *B* csomóponthoz visszatér. Hasonló procedúra eredményezi a további visszaadott értékeket: 2-t a *C* és 2-t a *D* csomópont számára. Végül vesszük a 3, 2 és 2 értékek maximumát, hogy a gyökér által visszaadott 3-as értéket megkaphassuk.

Created by XMLmind XSL-FO Converter.

A minimax algoritmus a játékfa teljes mélységi feltárását végzi. Ha a fa maximális mélysége *m*, és minden csomópontban *b* legális lépés létezik, akkor a minimax algoritmus időkomplexitása $O(b^m)$. A tárkomplexitása $O(bm)$ egy olyan algoritmus számára, amely az összes követőt egyszerre számítja ki, és $O(m)$ egy olyan algoritmus esetében, amely a követőket egyenként generálja 4.3. szakasz - Mélységi keresés. Valós játékok esetén ez az időkomplexitás az algoritmust teljesen haszontalanná teszi, az algoritmus azonban jó alap a játékok matematikai elemzéséhez és a gyakorlati szempontból alkalmasabb algoritmusokhoz.

6.3. ábra - Egy algoritmus a minimax döntések kiszámítására. Az algoritmus a lehető legjobb lépéshez tartozó operátort adja vissza, vagyis ahhoz a lépéshez tartozó operátort, amelyik a legnagyobb hasznossági értékkel rendelkező eredményre vezet, feltételezve, hogy az ellenfél úgy játszik, hogy minimalizálja a hasznossági értéket. A `MAX-ÉRTÉK` és `MIN-ÉRTÉK` függvények végigmennek a teljes játékfán, le egészen a levélcsomópontokig, hogy meghatározzák a csomópont felfelé terjesztett értékét.

```
function MINIMAX-DÖNTÉS(állapot) returns egy cselekvés
    inputs: állapot, a játék aktuális állapota

    v ← MAX-ÉRTÉK(állapot)
    return v értékű cselekvés KÖVETŐÁLLAPOTOK(állapot)-ban

function MAX-ÉRTÉK(állapot) returns egy hasznossági érték
    if VÉG-TESZT(állapot) then return HASZNOSSÁG(állapot)
    v ← −∞
    for minden s ∈ KÖVETŐÁLLAPOTOK(állapot) do
    v ← MAX(v, MIN-ÉRTÉK(s))
    return v

function MIN-ÉRTÉK(állapot) returns egy hasznossági érték
    if VÉG-TESZT(állapot) then return HASZNOSSÁG(állapot)
    v ← ∞
    for minden s ∈ KÖVETŐÁLLAPOTOK(állapot) do
    v ← MIN(v, MAX-ÉRTÉK(s))
  return v
```

2.3. Optimális döntések többszemélyes játékokban

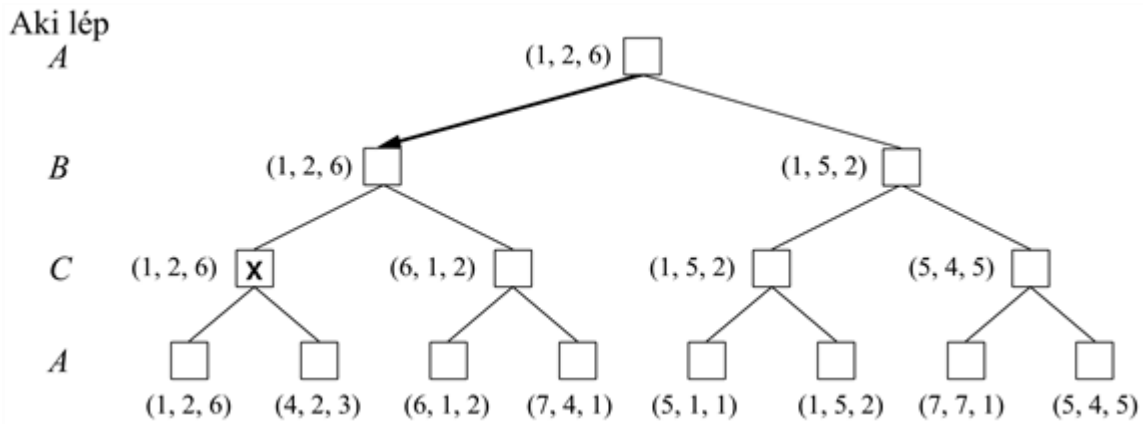
Számos elterjedt játékban több játékos is részt vehet, nem csupán kettő. Vizsgáljuk meg, hogy a minimax ötletet hogyan terjeszthetjük ki többszemélyes játékok esetére. Technikai szempontból a dolog egyszerű, azonban felmerül néhány érdekes koncepcionális kérdés.

Először is egy csomópontokhoz rendelt egyetlen értéket egy értékvektorral kell felváltani. Például egy háromszemélyes játékban, ahol három játékos, *A*, *B* és *C* vesz részt, minden csomóponttal egy $\langle v_A, v_B, v_C \rangle$ vektort társítunk. Végállapotok esetén ez a vektor megadja az állapot hasznosságát minden játékos szemszögéből (kétszemélyes zérusösszegű játékokban a kételemű vektort egy értékre le lehet egyszerűsíteni, mert az értékek mindig ellentétesek). Ezt a kibővítést legjobb úgy implementálni, hogy a `HASZNOSSÁG` függvény adja vissza a hasznosságok vektorát.

Most a nem terminális állapotokkal foglalkozunk. Nézzük meg a 6.4. ábrán látható játékfában az *X* jelzésű csomópontot. Ebben az állapotban a *C* játékos dönti el, hogy mit csináljon. Egyik választása a $\langle v_A = 1, v_B = 2, v_C = 6 \rangle$, míg a másik a $\langle v_A = 4, v_B = 2, v_C = 3 \rangle$ vektorokkal rendelkező végállapothoz vezet. Mivel 6 több, mint 3, *C*-nek az első lépést kellene választania. Ez azt jelenti, hogy ha a játék az *X* csomópontot eléri, a következő lépés a $\langle v_A = 1, v_B = 2, v_C = 6 \rangle$ hasznosságú végállapothoz fog vezetni. *X* visszaadott értéke így ez a vektor. Általánosságban egy *n* csomópont visszaadott értéke annak a követőnek a hasznosságvektora, amely követőnek az *n* csomópontnál választó játékos szempontjából legnagyobb az értéke.

Created by XMLmind XSL-FO Converter.

6.4. ábra - Három játékos (*A*, *B*, *C*) játékfája a három első fél lépés esetén. Minden csomópontot az összes játékos szemszegéből számított értékkel címkéztük meg. A legjobb lépést a gyökérnél jelöltük be.



Mindenki, aki olyan többszemélyes játékokat játszik, mint például a Diplomacy™, gyorsan meggyőződhet, hogy a kétszemélyes játéknál sokkal többről van itt szó. Többszemélyes játékban a játékosok között általában lehetségesek formális vagy informális **szövetség**ek (**aliance**s). A játék előrehaladtával szövetségek köttetnek és bontatnak fel.

Hogyan is kellene értelmezni egy ilyen viselkedést? Természetes következménye-e a szövetség az egyes játékosok optimális stratégiáinak egy többjátékos játékban? Úgy tűnik, hogy ez igaz lehet. Tegyük fel például, hogy *A* és *B* gyengén, míg *C* erősebben áll. Akkor néha optimális mind *A*, mind *B* számára, ha nem egymást, hanem *C*-t támadják meg, hogy az egyenként ne végezzen velük. Ily módon az együttműködés tisztán egoista viselkedésből is kialakulhat. Persze ahogy az együttes támadásnak kitett *C* gyengül, a szövetség értéke csökken, és vagy *A*, vagy *B* a megegyezést megszegheti. Egyes esetekben az explicit szövetségek az úgyis bekövetkezendő eseményeket rögzítik konkrét módon. Más esetekben szociális megbélyegzés jár a szövetség megszegéséért, így a játékosnak mérlegelnie kell a szövetség megszegésének rövid idejű előnyét és a szavahihetetlenként való megbélyegzés hosszú távú hátrányát. (Az ilyen bonyodalmakról többet a 17.6. alfejezetben.)

Ha a játék nem zérusösszegű, akkor együttműködés két játékos esetén is létrejöhet. Tegyük fel például, hogy létezik olyan végállapot, amelynek hasznossága $\langle v_A = 1000, v_B = 1000 \rangle$, és 1000 mindkét játékos számára az elérendő legnagyobb hasznosság. A két játékos optimális stratégiája akkor az, hogy ennek az állapotnak az elérése érdekében mindent megtesznek – azaz a két játékos automatikusan kooperálni fog, hogy a kölcsönösen előnyös célt elérjék.

3. Alfa-béta nyesés

A minimax keresés problémája, hogy a játékban a megvizsgálandó állapotok száma exponenciális a lépések számában. A kitevőtől sajnos megszabadulni nem tudunk, ám lényegében megfelezhetjük. A trükk az, hogy lehetséges a korrekt minimax döntés kiszámítása anélkül, hogy a játékfában minden csomópontra rá kelljen nézni. Ehhez kölcsönözhetjük a 4. fejezetben megismert **nyesés** (**prunning**) gondolatát, és a játékfa nagyobb részét a megfontolásokból kihagyhatjuk. A konkrét vizsgált technika az **alfa-béta nyesés** (**alpha-beta prunning**). Ha ezt egy standard minimax fára alkalmazzuk, ugyanazt az eredményt adja vissza, mint a minimax, a döntésre hatással nem lévő ágakat azonban lenyesi.

Tekintsük ismét a 6.2. ábrán látható egylépésváltásos játékfát. Kövessük még egyszer az optimális döntés kiszámításának menetét, most azonban kísérjük figyelemmel, hogy mit is tudunk valójában a folyamat minden pontjában. A lépésekhez a 6.5. ábra ad magyarázatot. Az eredmény az, hogy meg tudjuk határozni a minimax döntést anélkül, hogy a két levélcsomópontot bármikor is megnéznénk.

A módszert értelmezhetjük másképpen is, mint a `MINIMAX-ÉRTÉK` függvény egy egyszerűsítését. Legyen a *C* csomópontot a 6.5. ábrán látható, ki nem értékelt követő csomópontok értéke *x* és *y*, és legyen *z* a kettő minimuma. A gyökércsomópont értéke ekkor:

Created by XMLmind XSL-FO Converter.

MINIMAX-ÉRTÉK (*gyökér*) = max(min(3, 12, 8), min(2, *x*, *y*), min(14, 5, 2)

= max(3, min(2, *x*, *y*), 2)

= max(3, *z*, 2) ahol $z \leq 2$

= 3

Más szóval a gyökér értéke és így a minimax döntés *független* a lenyesett levelek *x* és *y* értékeitől.

Fontos

Az alfa-béta nyesést tetszőleges mélységű fákra lehet alkalmazni, és sokszor lehetséges, hogy levelek helyett teljes részfákat nyessünk le. Az általános elv az alábbi: tekintsünk egy olyan *n* csomópontot valahol a fa belsejében (lásd 6.6. ábra), hogy a Játékosnak lehetősége legyen e csomópontba lépni. Ha a Játékosnak van az *n* szülőcsomópontjánál vagy ettől feljebb bármelyik döntési pontban egy jobb választása, *m*, akkor *az n-t az aktuális játékban soha nem érjük el*. Ha az *n*-ről tehát (néhány követőjének megvizsgálásával) már eleget tudunk, hogy ehhez a konklúzióhoz jussunk, a csomópontot nyugodtan lenyeshetjük.

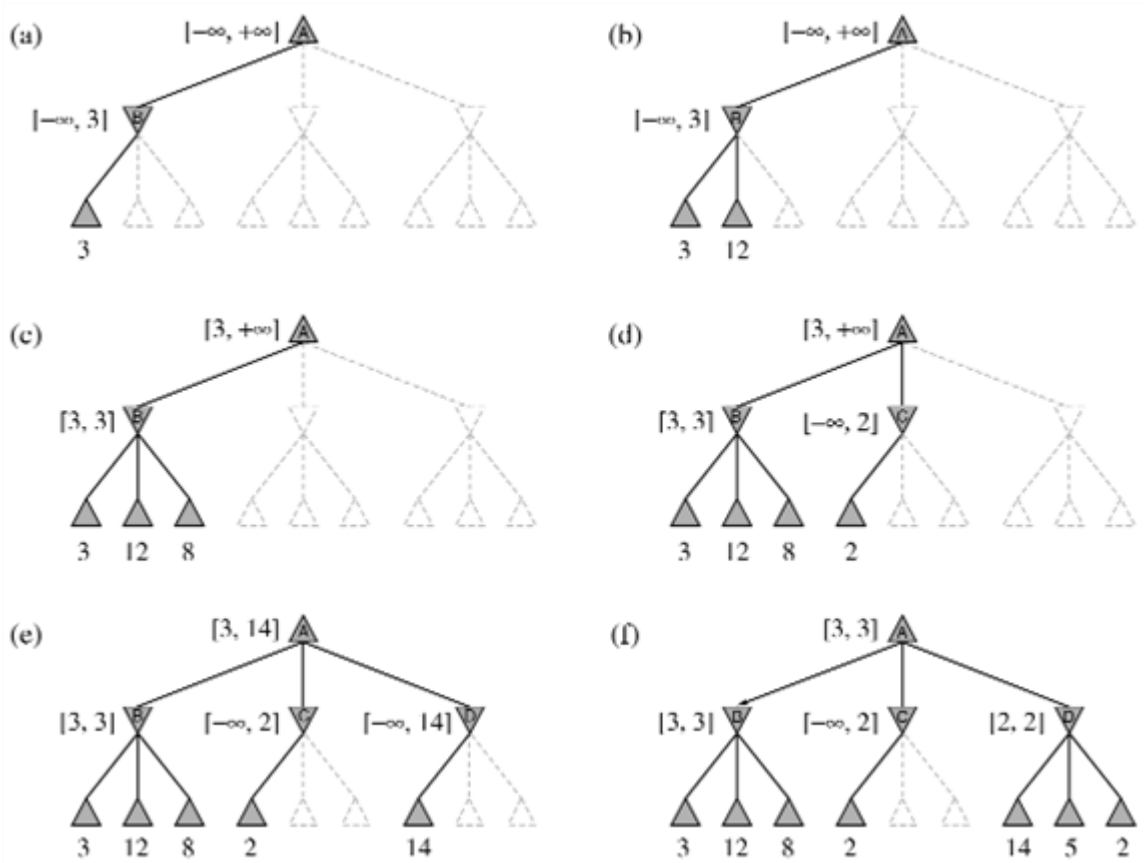
Emlékezzünk, hogy a minimax keresés mélységi keresés, egy adott időben tehát elegendő a fában egy egyedi út menti csomópontokkal foglalkozni. Az alfa-béta nyesés megnevezése két paramétertől származik, mely paraméterek az út mentén megjelenő visszaléptetett értékek korlátjaira vonatkoznak:

α = az út mentén tetszőleges döntési pontban a MAX számára eddig megtalált legjobb (azaz a legmagasabb értékű) választás értéke.

β = az út mentén tetszőleges döntési pontban a MIN számára eddig megtalált legjobb (azaz a legkisebb értékű) választás értéke.

6.5. ábra - Az optimális döntés kiszámításának lépései a 6.2. ábrán látható játékfa esetén. Minden ponton a lehetséges értékek terjedelmét mutatjuk meg minden egyes csomópont számára. (a) A *B* csomópont alatt az első levélnek 3 az értéke. *B* tehát, ami egy MIN csomópont, *legfeljebb* 3 értékű. (b) A *B* alatti második csomópontnak 12 az értéke. MIN ezt a lépést elkerüli, így *B* értéke még mindig legfeljebb 3. (c) A *B* alatti harmadik levélnek 8 az értéke. *B* összes követőit láttuk már, *B* értéke tehát pontosan 3. Most azt következtethetjük ki, hogy a gyökér értéke *legalább* 3, mert MAX-nak a gyökérben 3-as értékű választása van. (d) A *C* alatti első levélnek 2 az értéke. *C* tehát, ami egy MIN csomópont, *legfeljebb* 2 értékű. De mi tudjuk már, hogy *B*-nek 3 az értéke, MAX tehát *C*-t soha nem fogja választani. Azért nem is érdemes *C* további követőit megvizsgálni. Ez az alfa-béta nyesés egy példája. (e) A *D* alatti első levélnek 14 az értéke, így *D* értéke *legfeljebb* 14. Ez több mint MAX legjobb alternatívája (azaz 3), meg kell vizsgálni ezért *D* követőit. Jegyezzük meg, hogy most a gyökér minden követője esetén rendelkezünk értékkorláttal, a gyökér értéke legfeljebb 14. (f) *D* második követőjének értéke 5, így tovább kell folytatnunk a vizsgálatot. A harmadik követő értéke 2, *D* értéke tehát pontosan 2. MAX döntése a gyökérnél, hogy *B* felé kell lépni, 3-as értékkel.

Created by XMLmind XSL-FO Converter.

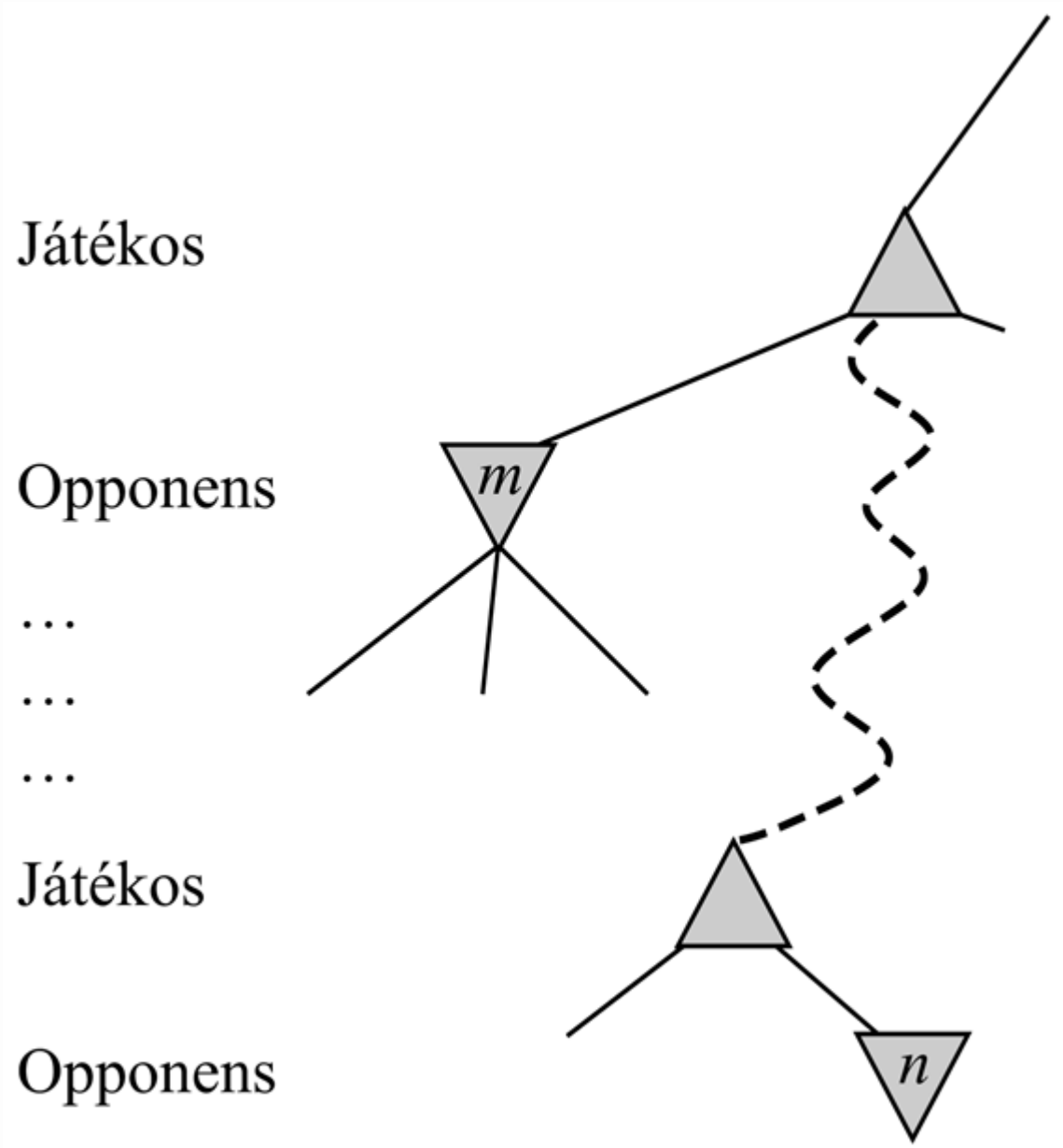


Az alfa-béta keresés az α és a *β* értékeit munka közben frissíti, és a csomópontnál a megmaradó ágakat lenyesi (a rekurzív hívást terminálja), amint csak biztossá válik, hogy az aktuális csomópont értéke rosszabb lesz, mint az aktuális α és *β* érték, `MAX`-ra, illetve `MIN`-re. A teljes algoritmust a 6.7. ábra mutatja. Bátorítjuk az olvasót, hogy a 6.5. ábrán látható fára alkalmazva, kövesse végig az algoritmus működését.

Az alfa-béta nyesés hatékonysága erősen függ a követő csomópontok vizsgálatának sorrendjétől. Például a 6.5. (e) és (f) ábrán *D* egyetlen követőjét sem lehetne lenyesni, mert először a legrosszabb követőt (`MIN` szemszögéből) generáltuk. Ha a harmadik követőt elsőnek generálnánk, a maradó kettőt le lehetne nyesni. Ez azt sugallja, hogy érdemes lenne először a legjobbnak tűnő csomópontokat megvizsgálni.

6.6. ábra - Alfa-béta nyesés: általános eset. Ha *m* a Játékos számára jobb, mint *n*, akkor a játék során sosem fogunk elérni *n*-be.

Created by XMLmind XSL-FO Converter.



Ha feltételezzük, hogy ez megtehető,[4] akkor az alfa-béta nyesésnek elég az $O(b^{m/2})$ csomópontot megvizsgálni, a legjobb lépés kiszámításához, a minimax $O(b^m)$ értékével szemben. Ez azt jelenti, hogy az effektív elágazási tényező b helyett $\sqrt{b}$ lesz – például a sakk esetében 35 helyett 6. Más szóval, az alfa-béta nyesés ugyanannyi idő alatt kétszer olyan messzire néz, mint a minimax. Ha a követő csomópontokat véletlen sorrendben vizsgálnánk a legjobbat-először helyett, a vizsgált csomópontok össz-száma $O(b^{3m/4})$ lenne, mérsékelt értékű b-kre. A sakk esetében egy igen egyszerű rendezőfüggvény (mint például az, hogy először a leütésekkel, majd a támadásokkal, ezt követően az előrelépésekkel és végül a hátralépésekkel próbálkozunk) a legjobb esetre vonatkozó $O(b^{m/2})$ eredményt egy 2-es tényező erejéig közelíti meg. Dinamika hozzáadása a lépésrendező sémákhoz, mint például azokkal a lépésekkel próbálkozni először, amelyek legutóbb a legjobbnak bizonyultak, az elvi korláthoz egészen közel visz.

A 3. fejezetben megjegyeztük, hogy a keresési fában az ismétlődő állapotok a keresési költségekben exponenciális növekedéshez is vezethetnek. Játékokban gyakran előfordulnak ismétlődő állapotok a **transzpozíció**k (**transpositions**) – a lépési szekvenciának az ugyanahhoz az álláshoz vezető különböző permutációi – miatt. Ha például Fehér lépése a_1, amire Fekete b_1 lépéssel válaszolhat, és a tábla más részén van egy vele nem kapcsolatos a_2 lépése, amire Fekete b_2-vel válaszolhat, akkor az $[a_1, b_1, a_2, b_2]$ és az $[a_1, b_2, a_2, b_1]$

[4] Természetesen nem tökéletes módon, különben a sorbarendező-függvény segítségével egy hibátlan játékot tudnánk játszani.

Created by XMLmind XSL-FO Converter.

szekvenciák mind ugyanabban az állásban végződnek (ahogy ebben az állásban végződik az előbbi lépések minden a_2-vel kezdődő permutációja). Érdemes ennek az állásnak a kiértékelését az első előforduláskor egy hash-táblában eltárolni, mert így a későbbi előfordulásoknál a kiszámítás megismétlését megspóroljuk.

Az előbb látott pozíciók hash-tábláját hagyományosan **transzpozíciós táblá**nak (**transposition table**) hívják. Ez lényegében azonos a GRÁF-KERESÉS *zárt* listájával 5. szakasz - Az Ismételt állapotok elkerülése. A transzpozíciós tábla használata drámai hatású lehet és a sakkban néha az elérhető mélység megduplázásához vezet. Másrészt ha másodpercenként a csomópontok millióit értékeljük ki, nem praktikus ezeket *mind* a transzpozíciós táblában tartani. A legértékesebb csomópontok megválasztására különböző stratégiákat dolgoztak ki.

6.7. ábra - Az alfa-béta keresési algoritmus. Ugyanazt a számítást hajtja végre, mint egy normál minimax algoritmus a 6.3. ábrán, kivéve két sort a MAX-ÉRTÉK és a MIN-ÉRTÉK hívásokban, ahol az α és a β értékeket kezeli (és adminisztrálja e paraméterek átadását).

```
function ALFA-BÉTA-KERESÉS(állapot) returns egy cselekvés
  inputs: állapot, a játék jelenlegi állása

  v ← MAX-ÉRTÉK(állapot, −∞, +∞)
  return egy cselekvés a KÖVETŐÁLLAPOTOK(állapot)-ban v értékkel

function MAX-ÉRTÉK(állapot, α, β) returns egy hasznossági érték
  inputs: állapot, a játék aktuális állása
          α, az állapot-ig vezető út mentén a legjobb érték MAX számára
          β, az állapot-ig vezető út mentén a legjobb érték MIN számára

  if VÉG-TESZT(állapot) then return HASZNOSSÁG(állapot)
  v ← −∞
  for minden s ∈ KÖVETŐÁLLAPOTOK(állapot) do
    v ← MAX(v, MIN-ÉRTÉK(s, α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-ÉRTÉK(állapot, α, β) returns egy hasznossági érték
  inputs: állapot, a játék aktuális állása
          α, az állapot-ig vezető út mentén a legjobb érték MAX számára
          β, az állapot-ig vezető út mentén a legjobb érték MIN számára

  if VÉG-TESZT(állapot) then return HASZNOSSÁG(állapot)
  v ← +∞
  for minden s ∈ KÖVETŐÁLLAPOTOK(állapot) do
    v ← MIN(v, MAX-ÉRTÉK(s, α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

4. Nem tökéletes, valós idejű döntések

A minimax algoritmus a játék egész keresési terét állítja elő, az alfa-béta nyesés viszont lehetőséget ad annak nagy részét lenyesni. Az alfa-béta nyesésnek mégis a végállapotokig kell keresnie legalább a keresési tér egy részében. Ez a mélység általában nem praktikus, mert a lépéseket elfogadható idő alatt – tipikusan legfeljebb percek alatt – kell megtenni. Shannon az 1950-es *Programming a computer for playing chess* c. cikkében azt javasolta, hogy a programnak a keresést korábban kell abbahagynia, és a csomópontokra egy heurisztikus **kiértékelő függvény**t (**evaluation function**) kell alkalmaznia, a nem végállapotokat lényegében végállapotoknak tekintve. Más szóval a minimaxot vagy az alfa-bétát kétféle módon kellene módosítani: a

Created by XMLmind XSL-FO Converter.

hasznossági függvényt egy heurisztikus KIÉRTÉKEL kiértékelő függvény helyettesíti, amely az állás hasznosságának egy becslését adja meg, valamint a célállapottesztet egy **vágási teszt** (**cutoff test**) váltja fel, amely eldönti, hogy KIÉRTÉKEL-t kell-e alkalmazni.

4.1. Kiértékelő függvények

Egy kiértékelő függvény egy adott állásból kiindulva a játék várható hasznosságának *becslését* adja vissza, éppúgy, ahogy a 4. fejezet heurisztikus függvénye a céltól való távolság becslését adja. A becslés ötlete, amikor Shannon felvetette, nem számított újnak. A sakkozók (és természetesen más játékok szerelmesei) az évszázadok során kifejlesztettek olyan módszereket, amelyekkel mérlegelni tudták egy állás értékét, hiszen a keresés mennyiségének tekintetében az emberek még inkább korlátozottak, mint a számítógépes programok. Nyilvánvaló, hogy egy játékprogram teljesítménye függ az alkalmazott kiértékelő függvény minőségétől. Egy pontatlan függvény a programot olyan állásokhoz vezeti, amelyek valójában vesztes pozíciók. Milyen pontosan tudunk jó kiértékelő függvényeket tervezni?

Először is a kiértékelő függvénynek a *végállapotokat* ugyanúgy kellene sorba rendeznie, mint az igazi hasznosságfüggvénynek, máskülönben az azt használó ágens még akkor is szuboptimális lépéseket választhatna, ha történetesen a játék végéig előre látna mindent. Másodsorban a kiértékelő függvény kiértékelése nem tarthat túl sokáig! (A kiértékelő függvény szubrutinként meg tudná hívni a MINIMAX-DÖNTÉS-t, és ki tudná számítani a pozíció pontos értékét, ez azonban az egész gyakorlat értelmét – az időmegtakarítást – megkérdőjelezné.) Harmadsorban a nem végállapotok tekintetében a kiértékelő függvénynek pontosan kell tükröznie a nyerés valódi esélyét.

El lehet merengeni a „nyerés esélye” kifejezésen. Végül is a sakk nem szerencsejáték, az aktuális állapotot biztosan tudjuk, és nincs a dologban kockavetés. Ha azonban egy nem végállapotban levágtuk a keresést, akkor az algoritmus szükségszerűen bizonytalan lesz annak valódi kimenetelében. Ezt a bizonytalanságot nem információs, hanem inkább számítási korlátok okozzák. Ha a kiértékelő függvény egy adott állapotban csak egy bizonyos mennyiségű számítást végezhet el, a legjobb, amit tehet, hogy kitalálja a végkimenetelt.

Próbáljuk meg ezt az ötletet pontosan megfogalmazni. A kiértékelő függvények többsége az állapot különböző **tulajdonság**ait (**feature**s) kiszámítva dolgozik – a sakkban például ilyen a mindkét félnél lévő gyalogok száma. A tulajdonságok, egybevéve, az állapotok különböző *kategóriáit,* avagy *ekvivalenciaosztályait* definiálják: az egyes kategóriákhoz tartozó állapotok az összes tulajdonság szempontjából ugyanazt az értéket képviselik. Minden egyes kategória általánosságban tartalmazni fog olyan állapotokat, melyek győzelemhez vezetnek, olyanokat, amelyek döntetlent eredményeznek, és olyanokat, amelyek vereséggel végződnek. A kiértékelő függvény nem tudhatja, hogy melyik állapot melyik, azonban visszatérhet egyetlen olyan értékkel, amely tükrözi az egyes kimenetelekhez tartozó állapotok *arányát*. Tegyük fel például, hogy a tapasztalatunk azt sugallja, hogy a kategóriához tartozó állapotok 72%-a vezet győzelemhez (+1 hasznosság), 20%-a vesztéshez (–1 hasznosság) és 8%-a döntetlenhez (0 hasznosság). Akkor a kategóriához tartozó állapotok értelmes értékelése a súlyozott átlag vagy a **várható érték** (**expected value**): (+1 × 0,72) + (–1 × 0,20) + (0 × 0,08) = 0,52. Elvben a várható értéket minden kategóriára meg lehetne határozni, ami egy tetszőleges állapot esetén is működő kiértékelő függvényhez vezetne. A végállapotokhoz hasonlóan, a kiértékelő függvénynek nem kell tényleges várható értékeket visszaadnia, amíg az állapotok *sorba rendezése* ugyanaz.

A gyakorlatban az ilyen elemzés túl sok kategóriát és így túl sok tapasztalatot kíván ahhoz, hogy a győzelem valószínűségét megállapítsuk. A kiértékelő függvények többsége ehelyett az egyes tulajdonságokból adódó numerikus értékeket külön számítja, majd a végleges értéket ezek *kombinációjaként* határozza meg. A kezdőknek szánt sakk-könyvek például egy közelítő **pontérték**et (**material value**) rendelnek minden egyes bábuhoz: minden gyalog 1 pontot, egy huszár vagy egy futó 3 pontot, egy bástya 5 pontot, míg egy vezér 9 pontot ér. Más jellemzők, mint például egy „jó gyalogstruktúra” és a „király biztonsága” érhetnek mondjuk egy fél gyalogot. Az állás kiértékeléséhez ezeket az értékeket egyszerűen összeadjuk. Egy gyaloggal ekvivalens biztonságos előny a győzelem lényeges valószínűségét adja, három gyaloggal ekvivalens biztonságos előny majdnem biztos győzelemhez vezet, ahogy ezt a 6.8. (a) ábra mutatja. Az ilyen típusú kiértékelő függvényt a matematikában **súlyozott lineáris függvény**nek (**weighted linear function**) nevezik, mert kifejezhető az alábbi alakban:

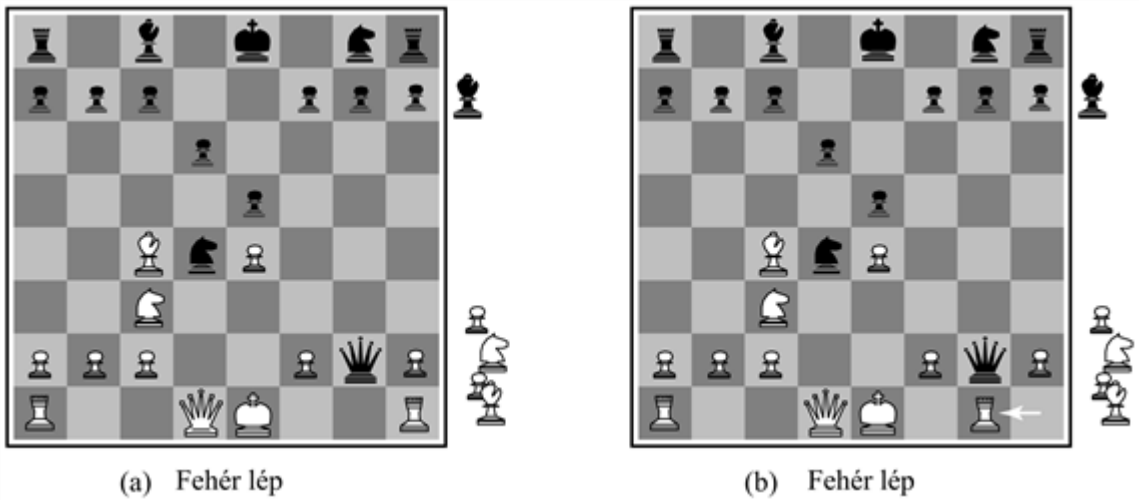
$$\text{KIÉRTÉKEL}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) = \sum_{i-1}^{n} w_i f_i(s)$$

Created by XMLmind XSL-FO Converter.

ahol a *w*-k a súlyok és az *f*-ek az adott táblaállás jellemzői. Sakk esetében az f_i-k a táblán levő egyes bábufajták számát jelölhetik és a w_i-k a bábukhoz rendelt pontértékek (gyalog 1, futó 3 stb.) lehetnek.

A tulajdonságértékek összeadása értelmes dolognak tűnik, azonban ezzel valójában egy igen erős feltételezéssel élünk: t. i. hogy az egyes tulajdonságok hozzájárulásai más tulajdonságok értékeitől *függetlenek*. A 3-as érték hozzárendelése a futóhoz figyelmen kívül hagyja például azt, hogy a végjátékban, amikor sok tér áll rendelkezésre a manőverezéshez, a futó erősebb. Ezért a jelenlegi sakk- és más játékokat játszó programok a tulajdonságok *nemlineáris* kombinációit is használják. Egy futópár egy kicsit értékesebb lehet például, mint egy futó értékének kétszerese, és egy futó a végjátékban értékesebb, mint a nyitásnál.

6.8. ábra - Két, kissé eltérő sakktáblaállás. (a) Feketének egy huszár és két gyalog előnye van, és a játékot megnyeri. (b) Fekete veszít, miután Fehér a vezérét leüti.



Az éles elméjű olvasó észreveszi, hogy a tulajdonságok és a súlyok a sakk szabályainak *nem* részei! Ezek az emberi sakkjátszás évszázados tapasztalataiból származnak. Lineáris kiértékelő képletet alkalmazva a tulajdonságok és a súlyok az állapotoknak az értékeik szerinti igazi sorrendezésük legjobb közelítését adják meg. A tapasztalat különösképpen azt sugallja, hogy az egy pontnál nagyobb biztonságos anyagi előny valószínűleg győzelemhez vezet, ha minden más tényező azonos. Hárompontos előny elegendő a majdnem biztos győzelemhez. Olyan játékokban, ahol ilyen tapasztalatra szert tenni nem lehet, a kiértékelő függvény súlyértékeit a 18. fejezetben leírt gépi tanulási technikákkal lehet becsülni. Biztató, hogy e technikák sakkra történő alkalmazása tényleg azt támasztja alá, hogy egy futó kb. három gyalogot ér.

4.2. A keresés levágása

Következő lépés az `ALFA-BÉTA-KERESÉS` olyan módosítása, hogy az algoritmus meghívja a heurisztikus `KIÉRTÉKEL` függvényt, amikor a keresést le kell vágni. Az implementációt tekintve a 6.7. ábrának a `VÉG-TESZT`-et tartalmazó két sorát az alábbi sorra cseréljük:

if `LEVÁGÁS-TESZT` (*állapot*, *mélység*) **then return** `KIÉRTÉKEL` (*állapot*)

Meg kell oldani annak az adminisztrálását is, hogy az aktuális *mélység* inkrementálisan változzék minden rekurzív hívás alkalmából. A legkézenfekvőbb megközelítés a keresés vezérlésére, ha a kereséshez rögzített mélységkorlátot rendelünk, hogy a `LEVÁGÁS-TESZT` (*állapot*, *mélység*) *igaz*zal térjen vissza egy *d*-nél nagyobb *mélységre* (*igaz*zal kell visszatérnie minden végállapot esetében is, ahogy azt a `VÉG-TESZT` tette). A *d* mélységet úgy választják meg, hogy a felhasznált idő ne haladja meg a játék szabályai által megengedett időt.

Egy kicsit robusztusabb megközelítést kapunk, ha a 3. fejezetben definiált iteratívan mélyülő algoritmust alkalmazzuk. Ha a program kifut az időből, akkor a legmélyebb, befejezett keresés által kiválasztott lépést adja vissza. Az ilyen megközelítések azonban hibákhoz vezethetnek a kiértékelő függvény közelítő jellege miatt. Tekintsük ismét a sakkban a pontelőnyön alapuló kiértékelő függvényt. Tegyük fel, hogy a program a mélységkorlátig keres, és a 6.8. (b) ábra táblaállásába jut, ahol Feketének huszár és két gyalog előnye van. A program a heurisztikus érték alapján ezt az állapotot Fekete számára valószínű győzelemnek fogja minősíteni.

Created by XMLmind XSL-FO Converter.

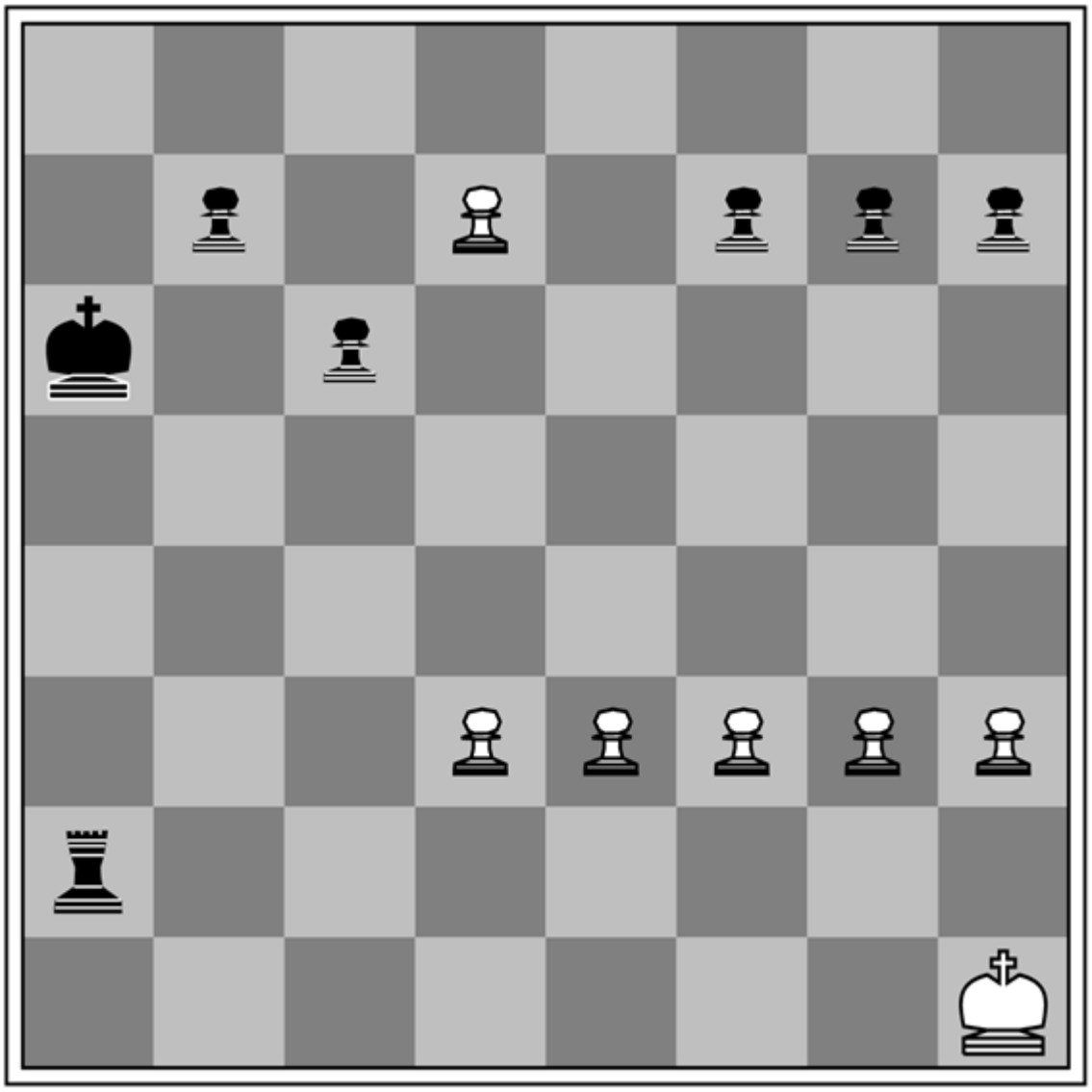
Fehér következő lépésében azonban leüti a fekete vezért, kárpótlás nélkül. Az állás így valójában Fehér számára jelent győzelmet, viszont hogy ezt belássuk, egy további lépésváltásig előre kellene nézni.

Nyilvánvalóan egy kifinomultabb levágási tesztre van szükség. A kiértékelő függvényt csak az **egyensúlyi** (**quiescent**) táblaállásokra szabad alkalmazni, vagyis olyan állásokra, amelyek értéke a közeljövőben nem változik meg radikálisan. A sakkban például az olyan táblaállások, amelyeknél tisztet lehet leütni, egy csak a pontértéket figyelembe vevő kiértékelő függvény esetén nem tekinthetők egyensúlyi állásnak. A nem egyensúlyi táblaállásokat tovább ki lehet fejteni, amíg egyensúlyi állásokat nem érünk el. Ezt a többletkeresést **egyensúlyi keresés**nek (**quiescence search**) nevezzük. Ezt a keresést néha csak bizonyos típusú lépésekre korlátozzák, mint például a táblaállás bizonytalanságait gyorsan feloldó bábuleütésekre.

A **horizontproblémá**t (**horizont problem**) nehezebb megszüntetni. A probléma akkor jelentkezik, amikor a program az ellenfél egy olyan következő lépésével kerül szembe, ami komoly károkat okoz, és egyben elkerülhetetlen. Tekintsük a 6.9. ábra sakkjátszmáját. Fekete egy kicsit erősebb pontértékű, ha azonban Fehér a hetedik sorról a nyolcadikba tudja juttatni a gyalogját, akkor vezérré változik, és Fehér könnyen megnyeri a játszmát. Fekete ezt 14 lépésváltással elodázhatja, ha bástyával sakkot ad, de a fehér gyalog elkerülhetetlenül vezérré fog változni. A rögzített mélységű keresések problémája, hogy ezek azt gondolják, hogy ilyen elodázó lépésekkel el lehet kerülni a vezérré változtató lépést. Azt mondjuk, hogy az elodázó lépések a vezér váltását „a horizonton túlra kitolják”, egy olyan helyre, ahol ezt nem lehet észrevenni.

6.9. ábra - A horizonthatás. A fekete bástyával adott sorozatos sakk a „horizonton túlra” kényszeríti az elkerülhetetlen vezérré változtató lépést, és ezt a táblaállást enyhén előnyös állásként tünteti fel Fekete számára, holott Fehér számára ez egy biztos nyerő állás.

Created by XMLmind XSL-FO Converter.



Fekete lép

Mivel a hardverek fejlődése mélyebb kereséshez vezet, azt várjuk, hogy a horizontprobléma kevésbé lesz gyakori, a nagyon hosszú késleltető lépésszekvenciák igen ritkák. A **szinguláris kiterjesztés** (**singular extension**) használata szintén hatásos a horizontprobléma elkerülésében, lényeges keresési költségtöbblet nélkül. A szinguláris kiterjesztés egy olyan lépés, amely „lényegesen jobb”, mint az állás minden más lépése. A szinguláris kiterjesztés keresése lényeges többletköltség nélkül a normális mélységkorláton túl is mehet, mert elágazási tényezője egységnyi. (Az egyensúlyi keresést a szinguláris kiterjesztés egy változatának lehet tekinteni.) A 6.9. ábrán a szinguláris kiterjesztés a lehetséges vezérlépést meg fogja találni, feltéve, hogy Fekete sakkot adó és Fehér védekező lépéseit „nyilvánvalóan jobb”-nak azonosítja a többi alternatívához képest.

Eddig a keresés egy adott szinten történő levágásáról és az eredményt igazoltan nem befolyásoló alfa-béta nyesésről beszéltünk. De **előrenyesés**t (**forward prunning**) is alkalmazhatunk, ahol egy adott csomópontban egyes lépéseket minden további elemzés nélkül azonnal lenyesünk. A sakkozók többsége egy adott állásnál nyilván csak néhány lépést vesz figyelembe (legalább tudatosan). Sajnos ez a megközelítés veszélyes lehet, mert nincs garancia arra, hogy nem fogjuk a legjobb lépést lenyesni. Ez katasztrofális lehet, ha a gyökér közelében történik, mert a program sokszor néhány „nyilvánvaló” lépést el fog nézni. Az előrenyesést speciális helyzetekben biztonságosan lehet alkalmazni – például amikor a két lépés szimmetrikus, vagy más szempontból ekvivalens, elegendő csak az egyiket figyelembe venni –, vagy olyan csomópontoknál, amelyek a keresési fában mélyen találhatók.

Created by XMLmind XSL-FO Converter.

Az itt leírt technikák kombinált alkalmazásával egy elfogadható szinten sakkozó (vagy más játékot játszó) programot kaphatunk. Tegyük fel, hogy a sakk esetére implementáltunk egy kiértékelő függvényt, egy értelmes kereséslevágást egyensúlyi kereséssel és egy nagy transzpozíciós táblát. Tegyük fel azt is, hogy hónapokig tartó munkás bitfaragással, a legújabb PC-n tudunk generálni és kiértékelni másodpercenként kb. egymillió csomópontot, ami kb. 200 millió csomópont megvizsgálását teszi lehetővé lépésenként a standard időkorlátok mellett (3 perc lépésenként). A sakk elágazási tényezője átlagosan kb. 35. Mivel 35^5 kb. 50 millióval egyenlő, a minimax keresést alkalmazva így csak öt lépésváltásig tudnánk előrenézni. Bár egy ilyen program nem hasznavehetetlen, egy átlagos emberi játékos, aki esetenként hat vagy nyolc lépésváltásig tud előrenézni, könnyűszerrel bolondot tudna belőle csinálni. Az alfa-béta nyeséssel kb. 10 lépésváltást kapunk, ami a játék mesteri színvonalának felel meg. A 6.7. alfejezet további nyesési technikákat ír le, amikkel az effektív keresési mélység durván 14 lépésváltásig kiterjeszthető. A nagymesteri minősítéshez egy lényegesen finomított kiértékelő függvény kellene továbbá az optimális megnyitások és a végjátékok nagy adatbázisa. És az sem ártana, ha lenne egy szuperszámítógépünk, amin a programot futtatnánk.

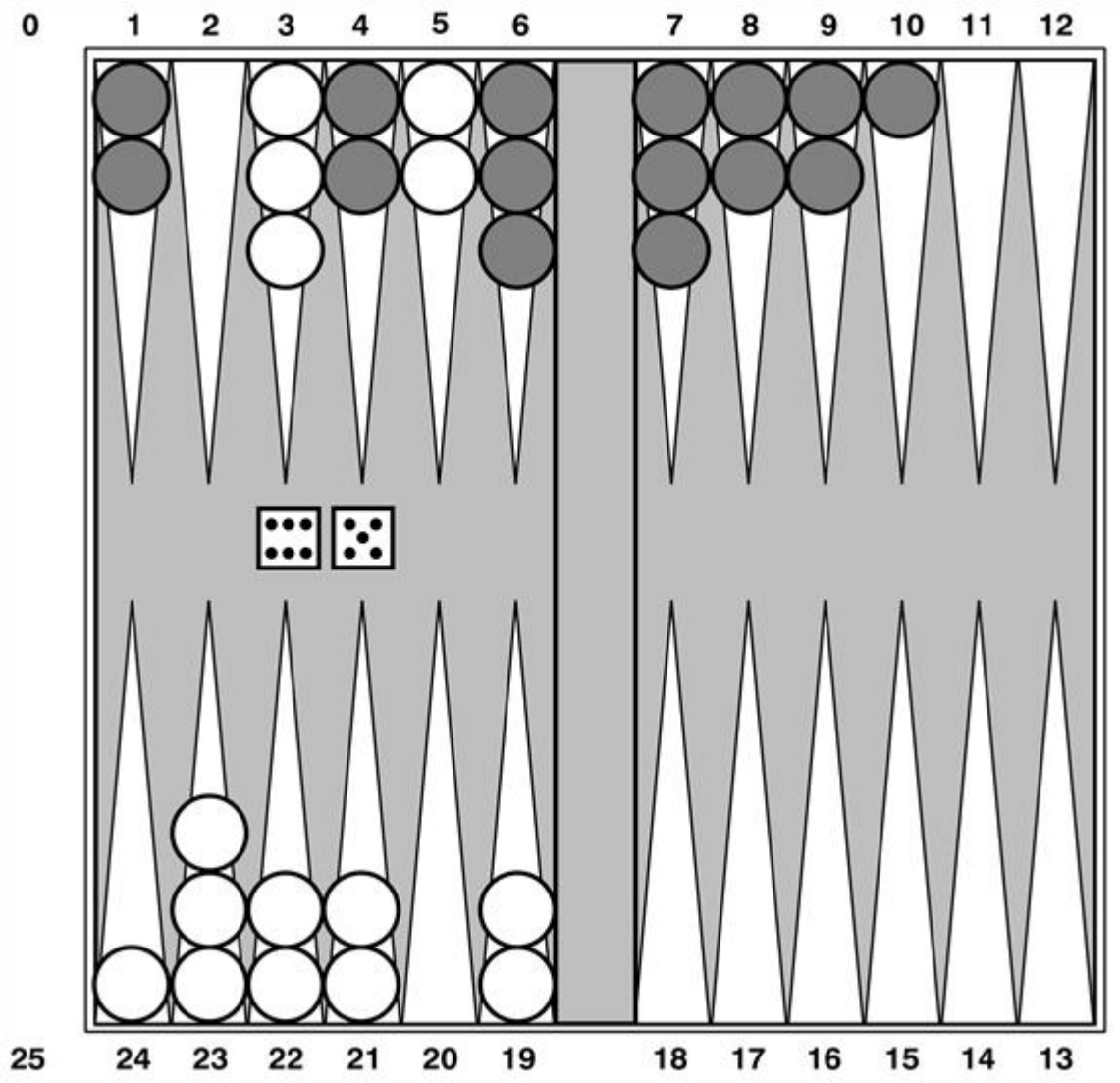
5. Véletlen elemet is tartalmazó játékok

A valós életben számos olyan megjósolhatatlan külső eseménnyel van dolgunk, ami miatt előre nem látható helyzetbe kerülünk. Sok játék ezt a megjósolhatatlanságot egy véletlen elem, mint például a kockadobás, bevezetésével veszi figyelembe. Ily módon egy lépéssel közelebb visznek bennünket a valósághoz, és megéri megvizsgálni, hogy ez milyen hatással van a döntési folyamatra.

Az ostábla tipikusan olyan játék, amely ötvözi a szerencsét és a tudást. A játékosok a lépések előtt kockát dobnak, amivel meghatározzák az adott játékos számára megengedett lépéseket. A 6.10. ábra ostáblaállásánál Fehér egy 6-ost és egy 5-öst dobott, és négy lehetséges lépés közül választhat.

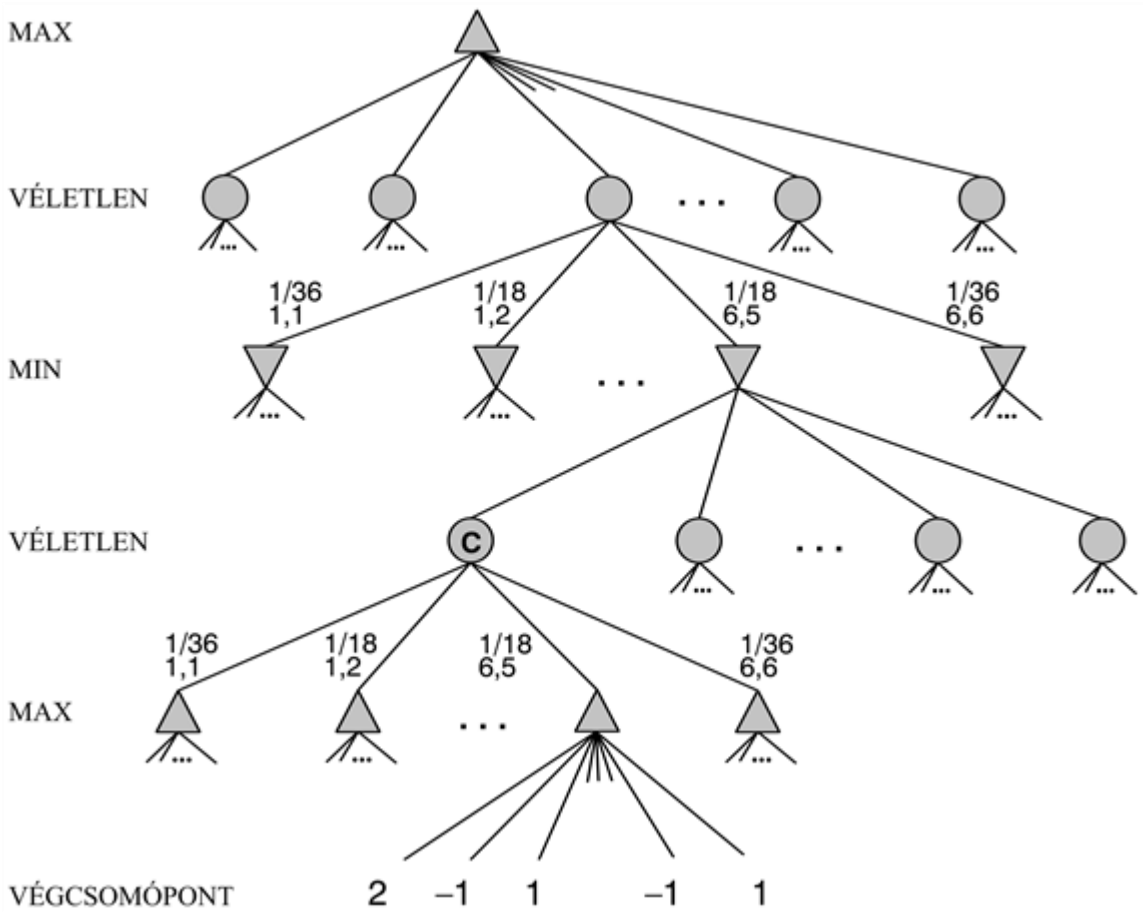
6.10. ábra - Egy tipikus ostáblaállás. A játék lényege, hogy az egyik fél összes bábuját eltávolítsuk a tábláról. Fehér az óramutató járásával egyező irányban halad a 25-ös felé, míg Fekete az óramutató járásával ellentétes irányban, a 0 felé halad. Egy bábuval bármelyik helyre lehet lépni, kivéve azokat a helyeket, ahol az ellenfélnek már legalább két bábuja található. Ha egy olyan helyre lépünk, ahol az ellenfélnek egy bábuja található, akkor azt a bábut elfogtuk, és a lépkedést az elejéről kell újrakezdenie. Ennél a táblaállásnál Fehér épp most dobott 6–5-öt, és négy megengedett lépés közül választhat: (5–10,5–11), (5–11,19–24), (5–10,10–16) és (5–11,11–16).

Created by XMLmind XSL-FO Converter.



6.11. ábra - Egy ostáblaállás sematikus játékfája

Created by XMLmind XSL-FO Converter.



Habár Fehér tudja, hogy számára melyek a megengedett lépések, azt nem tudja, hogy Fekete milyen számokat fog dobni, így nem tudhatja, hogy melyek lesznek Fekete megengedett lépései. Ez azt jelenti, hogy Fehér nem tud egy 3 × 3-as amőbánál látott teljes játékfához hasonlót elkészíteni. Az ostábla esetén a játékfának a `MAX` és `MIN` csomópontok mellett tartalmaznia kell **véletlen csomópont**okat **(chance node**s) is. A 6.11. ábrán a véletlen csomópontokat körök jelölik. A véletlen csomópontokból kiinduló ágak a lehetséges kockadobásokat jelölik. Minden ág a dobott értékkel és előfordulásának valószínűségével van felcímkézve. Két kockával történő dobásnál harminchatféle eredményt kaphatunk. Ezek a dobások mind azonos valószínűségűek, mivel azonban a 6–5 megegyezik az 5–6-tal, így csak 21 különböző dobás létezik. A hat dupla (1–1-től egészen 6–6-ig) 1/36-od valószínűséggel, míg a többi 15 különböző dobás 1/18-ad valószínűséggel fordul elő.

A következő lépés annak megértése, hogyan kell helyes döntéseket hozni. Nyilvánvalóan továbbra is azt a lépést szeretnénk kiválasztani, ami a legjobb álláshoz vezet. A lehetséges állások azonban most nem rendelkeznek egy jól meghatározott minimax értékkel. Ehelyett csak egy átlagos vagy **várható érték**et (**expected value**) tudunk kiszámítani, ahol az átlagolást az összes lehetséges kockadobásra végezzük el. Ezzel a determinisztikus játékok **minimax érték**ét (**minimax value**) a véletlen csomópontokat tartalmazó játékok **várhatóminimax érték**ére (**expectiminimax value**) általánosítottuk. A végállapotok és a `MAX` és `MIN` állapotok (ahol a kockadobás eredménye ismert) ugyanúgy viselkednek, mint eddig. A véletlen csomópontok kiértékeléséhez az összes lehetséges kockadobás figyelembevételével számított súlyozott átlagot kell venni, vagyis:

$$\text{VÁRHATÓMINIMAX}(n) = \begin{cases} \text{HASZNOSSÁG}(n) & \text{ha } n \text{ egy végállapot} \\ \max_{s \in \text{Követők}(n)} \textit{VÁRHATÓMINIMAX}\ (s) & \text{ha } n \text{ egy MAX csomópont} \\ \min_{s \in \text{Követők}(n)} \textit{VÁRHATÓMINIMAX}\ (s) & \text{ha } n \text{ egy MIN csomópont} \\ \sum_{s \in \text{Követők}(n)} \text{P}(s) \cdot \textit{VÁRHATÓMINIMAX}\ (s) & \text{ha n egy véletlen csomópont} \end{cases}$$

Created by XMLmind XSL-FO Converter.

ahol a véletlen *n* csomópont állapotátmenet-függvénye az *n* állapotot egyszerűen módosítja az összes kockadobás eredményezte *s* követő állapot értékével és a dobás *P*(*s*) valószínűségével. Ezt az eljárást rekurzívan alkalmazhatjuk a gyökérig, ugyanúgy, mint a minimax esetén. Az algoritmus részleteit feladatnak hagyjuk meg az olvasónak.

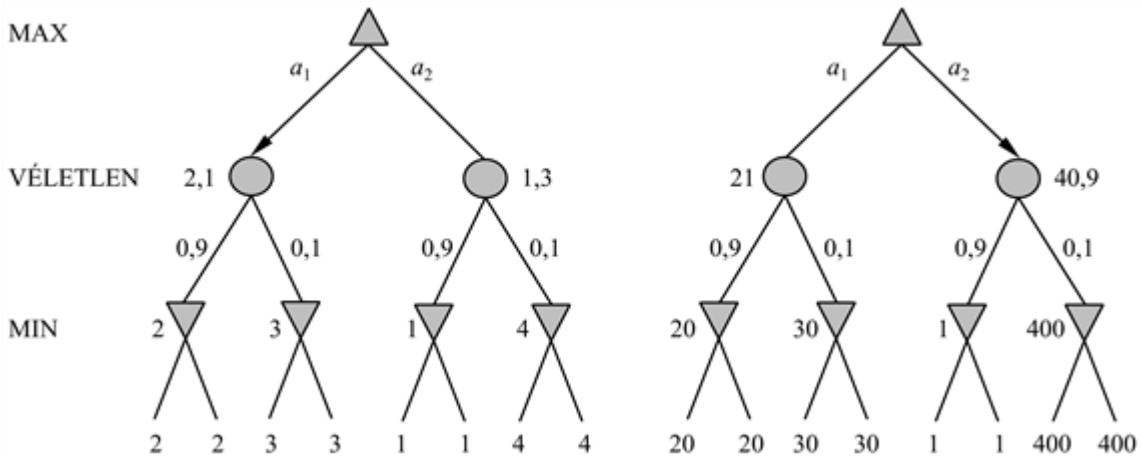
5.1. Az állás kiértékelése véletlen csomópontokat tartalmazó játékok esetén

Hasonlóan a minimaxhoz, a várhatóminimax esetén is alkalmazhatjuk a nyilvánvaló közelítést, vagyis valamilyen ponton levághatjuk a keresést, és a levelekre egy kiértékelő függvényt alkalmazhatunk. Úgy gondolhatnánk, hogy egy olyan játék esetén, mint az ostábla, a kiértékelő függvény elvben semmiben sem tér el a sakknál alkalmazott kiértékelő függvényektől, melyeknek most is magasabb pontot kell adniuk a jobb állásokra. Valójában azonban a véletlen csomópontok jelenléte azt jelenti, hogy sokkal körültekintőbbnek kell lennünk abban a tekintetben, hogy a kiértékelő függvény értékei mit jelentenek. A 6.12. ábra mutatja a problémát: ha a kiértékelő függvény a levelekhez az [1, 2, 3, 4] értékeket rendeli hozzá, az A_1 lépés lesz a legjobb, az [1, 20, 30, 400] értékekkel az A_2 lesz a legjobb. Ebből adódóan egy program teljesen másként viselkedik, ha átskálázzuk a kiértékelő függvény értékeket! Kiderül, hogy ha el akarjuk kerülni ezt az érzékenységet, akkor a kiértékelő függvény az egy adott állásból való győzelem valószínűségének (vagy általánosabban az állás várható hasznosságának) csak egy *pozitív lineáris transzformációja* lehet. Ez a bizonytalanságot is tartalmazó helyzetek fontos és általános tulajdonsága. A 16. fejezetben tovább foglalkozunk ezzel a kérdéssel.

5.2. A várhatóminimax komplexitása

Ha a program előre tudná a játék folyamán a későbbiekben előforduló összes kockadobás eredményét, akkor egy kockadobást is tartalmazó játék megoldása ugyanolyan lenne, mint egy kockadobást nem tartalmazó játéké, amit a minimax $O(b^m)$ idő alatt tesz meg. Mivel a várhatóminimax figyelembe veszi az összes lehetséges kockadobás-sorozat eredményét is, annak időigénye $O(b^m n^m)$ lesz, ahol *n* a különböző dobások számát jelöli.

6.12. ábra - A levélértékeken végrehajtott sorrendőrző transzformáció megváltoztatja a legjobb lépést



Még ha a fa mélységkorlátja egy kis értékű *d* is, a minimaxhoz képesti többletköltség miatt irreális túl messzire előrenézni a véletlen játékok többségénél. Ostáblánál *n* értéke 21, *b* pedig általában 20 körüli értéket vesz fel, de bizonyos helyzetekben akár 4000 is lehet, a dupla kockadobásoknál. Valószínűleg csak három lépésváltást tudunk kezelni.

A problémáról másféleképp is gondolkodhatunk: az alfa-béta algoritmus előnye, hogy figyelmen kívül hagyja azokat a későbbi fejleményeket, amelyek a legjobb játék során egyszerűen nem következnek be. Így a valószínű előfordulásokra koncentrál. A kockát használó játékok esetén *nem léteznek* valószínű lépéssorozatok, mivel adott lépések megtételéhez először a kockának kell a megfelelő oldalára esnie, hogy ezek a lépések egyáltalán megengedhetők legyenek. Ez egy általános probléma minden olyan esetben, amikor a bizonytalanság is képbe kerül: a lehetőségek óriási mértékben megsokszorozódnak, és értelmetlenné válik részletes cselekvési terveket kidolgozni, mert a világ valószínűleg nem ezek szerint fog viselkedni.

Created by XMLmind XSL-FO Converter.

Kétségtelen, hogy az olvasónak már eszébe jutott, hogy valami alfa-béta nyeséshez hasonló dolgot a véletlen csomópontokat tartalmazó játékfákra is alkalmazni lehetne. Kiderül, hogy ez valóban lehetséges. A MAX és a MIN csomópontok elemzése változatlan, de egy kis leleményességgel a véletlen csomópontokat is nyeshetjük. Tekintsük a 6.11. ábra *C* véletlen csomópontját, és nézzük meg, mi történik az értékével, amikor a gyermekeit vizsgáljuk és kiértékeljük. Kérdés, hogy lehetséges-e *C* értékére egy felső korlátot találni, még mielőtt az összes gyermekét megvizsgálnánk? (Idézzük vissza, hogy az alfa-béta nyesésnek pontosan erre van szüksége ahhoz, hogy egy csomópontot és az abból kiinduló részfát lenyeshesse.) Ez első pillantásra lehetetlennek tűnhet, mivel a *C* értéke a gyermekei értékének az *átlaga,* és amíg nem láttuk az összes kockadobás eredményét, addig ez az átlag akármi is lehet, mivel a meg nem vizsgált gyermekek akármilyen értéket felvehetnek. Ha azonban a hasznosságfüggvény lehetséges értékeit korlátok közé szorítjuk, akkor korlátokat kaphatunk az átlagra is. Például ha azt mondjuk, hogy a hasznosságértékek +3 és –3 között lehetnek, akkor a levélcsomópontok értékei már korlátosak, és a véletlen csomópont értékére az összes gyermekének megvizsgálása nélkül is *adhatunk* felső korlátot.

5.3. Kártyajátékok

A kártyajátékok sok szempontból érdekesek, nem csupán a szerencsejáték-jellegük miatt. A játékok óriási választékából azokra fogunk összpontosítani, ahol a kártyákat a játék elején véletlen módon osztják szét, és ahol mindegyik játékos által kapott leosztás más játékosok számára nem megfigyelhető. Ilyen játék a bridzs, a whistjáték, a hearts[5] és a póker néhány formája.

Az első pillantásra úgy tűnhet, hogy a kártyajátékok pont olyanak, mint a kockadobásos játékok. A kártyákat véletlenül osztjuk szét, és ez meghatározza a játékosok lehetséges lépéseit. Itt azonban az összes kocka a legelején gördül le! Ezt a hasonlatot próbáljuk továbbvinni. Ki fog derülni, hogy ez a gyakorlatban igen hasznos lesz. Mellesleg hibás is, igen érdekes okok miatt.

Képzeljük el, hogy két játékos, MAX és MIN, négykártyás kétkezes bridzset gyakorol, ahol minden kézleosztás látható. A leosztások az alábbiak, és MAX hív elsőnek:

MAX: ♥ 6 ♦ 6 ♣ 9 8 MIN: ♥ 4 ♠ 2 ♣ 10 5

Tegyük fel, hogy MAX ♣ 9-et játszik. MIN-nek szint kell játszania, tehát ♣ 10-et vagy ♣ 5-öt. MIN ♣ 10-et játszik és elviszi az ütést. Most MIN következik és ♠ 2-vel indul. MAX-nak nincs pikkje (és így az ütést elvinni nem tudja), következésképpen valamelyik kártyáját le kell játszania. Nyilvánvaló választás a ♦ 6, mert a két másik győztes kártya. Most függetlenül attól, hogy MIN mivel indul a következő ütésben, MAX az utolsó két ütést elviszi, és a játék két-két ütés szinten döntetlen lesz. Könnyű megmutatni a minimax megfelelő változatával (lásd 6.12. feladat), hogy MAX ♣ 9-es indulása valójában egy optimális választás.

Módosítsuk most MIN leosztását, ♥ 4-et ♦ 4-re cserélve:

MAX: ♥ 6 ♦ 6 ♣ 9 8 MIN: ♦ 4 ♠ 2 ♣ 10 5

A két eset tökéletesen szimmetrikus: a játék lefolyása azonos lesz, kivéve, hogy a második ütésnél MAX a ♥ 6-ot fogja megjátszani. A játék ugyanúgy döntetlen lesz két-két ütéssel, és a ♣ 9 optimális döntés lesz most is.

Egyelőre minden jónak tűnik. Most rejtsük el MIN kártyáinak egyikét: MAX tudja, hogy MIN-nek vagy az egyik (♥ 4-gyel), vagy a másik (♦ 4-gyel) leosztása van, de fogalma sincs melyik. MAX a következőképpen érvel:

A ♣ 9-es indulás optimális választás MIN első leosztásával szemben és hasonlóan a második leosztásával szemben is, így most is optimálisnak kell lennie, mert tudomásom szerint MIN-nél a két leosztás valamelyike van.

Amit MAX használ, azt általánosságban a „jövőbe látás szerinti átlagolásnak” nevezhetnénk. Az ötlet a cselekvés értékelése nem látott kártyák mellett oly módon, hogy először kiszámítunk minimax értékeket minden lehetséges leosztás esetére, majd várható értéket számítunk a leosztásokra nézve, azok valószínűségét felhasználva.

Ha azt gondolja, hogy ez egy értelmes stratégia (vagy ha nincs véleménye, mert nem érti a bridzset), tekintse a következő történetet:

[5] Az Angol–magyar Nagyszótár sem segít – ezeknek a játékoknak nincs a mai magyar nyelvben használatos nevük. (*A ford.*)

Created by XMLmind XSL-FO Converter.

1. nap: Az *A* út egy halom aranyhoz vezet. A *B* út egy elágazáshoz vezet. Ha az elágazásnál balra fordul, egy ékszerhegyet talál, ha az elágazásnál jobbra fordul, elüti egy busz.

2. nap: Az *A* út egy halom aranyhoz vezet. A *B* út egy elágazáshoz vezet. Ha az elágazásnál jobbra fordul, egy ékszerhegyet talál, ha az elágazásnál balra fordul, elüti egy busz.

3. nap: Az *A* út egy halom aranyhoz vezet. A *B* út egy elágazáshoz vezet. Ha jól választ, egy ékszerhalmot talál, ha rosszul, elüti egy busz.

Nyilvánvaló, nem buta döntés, ha az első két napon a *B* utat választjuk. Nincs olyan épelméjű személy azonban, aki a harmadik napon is maradna *B*-nél. Mégis pontosan ez az, amit a jövőbe látás szerinti átlagolás sugall. A *B* út az 1. nap és a 2. nap helyzetekben optimális, következésképpen optimális a 3. nap szituációban is, hiszen az első két eset egyike fog előfordulni. Térjünk vissza a kártyajátékunkhoz: miután `MAX` ♣ 9-et hív, `MIN` ♣ 10-zel győz. `MIN` ♠ 2-vel indul, mint korábban, és most `MAX` ott van az elágazásnál bármiféle eligazítás nélkül. Ha `MAX` a ♥ 6-ot játszotta meg, és `MIN`-nek van még ♥ 4-e, ♥ 4 lesz a győztes, és `MAX` a játékot elveszíti. Hasonlóan, ha `MAX` a ♦ 6-ot játszotta meg, és `MIN`-nek van még ♦ 4-e, `MAX` szintén veszteni fog. A ♣ 9 első ízben való hívása tehát olyan helyzethez vezet, ahol `MAX`-nak 50%-os esélye van a vesztésre (sokkal jobb lenne, ha a ♥ 6-ot vagy a ♦ 6-ot hívná elsőnek, döntetlent biztosítva).

A leszűrendő lecke az, hogy amikor hiányos az információ, meg kell fontolni, hogy *milyen információval fogunk rendelkezni* a játék minden pillanatában. `MAX` algoritmusával az a probléma, hogy feltételezi, hogy minden lehetséges leosztásnál a játék úgy folytatódik, mintha *minden kártya látható lenne*. Ahogy a példánk mutatja, ez olyan cselekvésre készteti `MAX`-ot, mintha minden *jövőbeli* bizonytalanság feloldódna, ha eljön az ideje. `MAX` algoritmusa sem fog soha *információgyűjtéshez* folyamodni (vagy a partner informálásához), mert egy-egy leosztáson belül erre nincs szüksége. Az olyan játékoknál, mint a bridzs, gyakran értelmes dolog olyan kártyával indulni, amely segíti kideríteni az ellenség leosztását, vagy a partnerünket a saját leosztásunkról informálja. Ilyen viselkedést automatikusan generálhatunk a nem tökéletes információjú játékokra kifejlesztett optimális algoritmussal. Az ilyen algoritmus nem a világállapotok terében (kártyaleosztások), hanem a **hiedelmi állapot**ok terében (**belief state**s) (hiedelmek, hogy kinek milyen kártyája van, milyen valószínűséggel) keres. Az algoritmust a 17. fejezetben tudjuk majd megfelelően elmagyarázni, miután felépítettük a szükséges valószínűségi apparátust. Abban a fejezetben azzal a nagyon fontos szemponttal is fogunk foglalkozni, hogy a nem tökéletes információjú játékokban az a legjobb, ha minél kevesebb információt adunk ki az ellenségnek, és ennek legjobb módszere, ha *nem megjósolható* módon cselekszünk. Ez az oka annak, hogy az étteremellenőrök véletlen módon választják meg az ellenőrzések időpontját.

6. A jelenleg legfejlettebb játékprogramok

Azt lehetne mondani, hogy a játékok valami olyat jelentenek az MI számára, mint amit a Forma–1 versenyek a gépkocsiipar számára. A legfejlettebb játékprogramok villámgyors, hihetetlenül finoman hangolt rendszerek, amelyek nagyon fejlett módszereket alkalmaznak, de amelyeknek nem sok hasznát vesszük a vásárláskor. Bár vannak kutatók, akik azt gondolják, hogy a játékok az MI fővonalát tekintve többé-kevésbé irrelevánsak, mégis élénk érdeklődést váltanak ki, és folyamatosan fenntartják az innovációt, amit a szélesebb közösség fel is karol.

6.1. Sakk

Herbert Simon 1957-ben megjósolta, hogy 10 éven belül a számítógépek legyőzik az emberi világbajnokot. Negyven év múltával hatjátszmás kirakatmérkőzésen a Deep Blue legyőzte Garri Kaszparovot. Simon tévedett, de csak egy 4-es szorzóval. Kaszparov azt írta:

> A mérkőzés döntő játszmája a 2. játszma volt, amely emlékeimben sebként maradt meg … láttunk valamit, ami a legvadabb elképzeléseinket is felülmúlta, hogy egy számítógép hogyan lesz képes döntéseinek hosszú távú pozíciós konzekvenciáit előre látni. A gép – emberi veszélyérzetet produkálva – megtagadta, hogy egy döntően rövid távú előnyt jelentő állásba kerüljön.
>
> —(Kaszparov, 1997)

A Deep Blue-t Murray Campbell, Feng-Hsiung Hsu és Joseph Hoane fejlesztették ki az IBM-nél (Campbell és társai, 2002), a Campbell és Hsu által korábban a Carnegie Mellon Egyetemen kifejlesztett Deep Thought tervére alapozva. A győzedelmes gép egy párhuzamos számítógép volt 30 IBM RS/6000 processzorral, amelyek a „szoftverkeresést” futtatták, és 480 VLSI sakk-célprocesszorral, amelyek a lépésgenerálást (beleértve a

Created by XMLmind XSL-FO Converter.

lépések sorba rendezését), a fa utolsó néhány szintjén a „hardverkeresést” és a levélcsomópontok kiértékelését valósították meg. A Deep Blue átlagosan másodpercenként 126 millió, csúcssebességnél 330 millió csomópontot vizsgált meg. Lépésenként 30 milliárd állást generált, rutinszerűen 14-es mélységet elérve. A gép szíve a standard alfa-béta keresés volt, transzpozíciós táblával, a siker kulcsa azonban úgy tűnik, az a képessége, hogy kiterjesztéseket tudott generálni az érdekes kényszerlépés-sorozatok mélységi korlátjain túl. Egyes esetekben a keresés a 40-es mélységet is elérte. A kiértékelő függvény 8000 tulajdonsággal dolgozott, ezekből egyesek a bábuk igen specifikus konfigurációit is leírták. A rendszer felhasználta a 4000 állást tartalmazó „megnyitások könyvét” és a 700 000 nagymesteri játszmaleírást tartalmazó adatbázist, ahonnan konszenzusos javaslatokat lehetett generálni. A rendszer a megoldott végjáték-pozíciók nagy adatbázisával is rendelkezett, amely minden ötbábus és számos hatfigurás állást tartalmazott. Ez az adatbázis a keresési mélység lényeges kiterjesztését jelentette, lehetővé téve Deep Blue számára egyes esetekben a tökéletes játékot annak ellenére, hogy sok lépésre volt még a matthelyzettől.

A Deep Blue sikere megerősítette azt a széles körű hiedelmet, hogy számítógépes játékokban az előrehaladás főleg az egyre hatékonyabb hardver eredménye – ezt a véleményt az IBM is bátorította. Másfelől a Deep Blue megalkotói azt nyilatkozták, hogy a keresés kiterjesztése és a kiértékelő függvény szintén kritikusak voltak (Campbell és társai, 2002). Azonban tudjuk azt is, hogy néhány közelmúltbeli algoritmikus javítás lehetővé tette, hogy standard PC-n futó programok 1992 óta minden Számítógépes Sakkvilágbajnokságot megnyerjenek, sokszor 1000-szer több csomópont keresésére is képes masszívan parallel ellenségeket is legyőzve. A nyesési heurisztikák egész választékának használatával az effektív elágazási tényezőt kevesebb mint 3-ra lehet csökkenteni (a tényleges 35-ös elágazási tényezőhöz viszonyítva). Ezek közül legfontosabb a **nulla lépés** (**null move**) heurisztika, amely sekély keresést alkalmazva – ahol az ellenség az indulásnál kétszer is léphet – az állás értékének jó alsó becslését adja. Ez az alsó korlát sokszor lehetővé teszi az alfa-béta nyesést a teljes mélységű keresés költsége nélkül. Fontos még a **hatástalanság nyesése** (**futility prunnig**), amely segíti eldönteni, hogy mely lépések vezetnek béta nyeséshez a követő csomópontok szintjén.

A Deep Blue csapata a Kaszparovval való visszavágó alól kibújt. Helyette 2002-ben a fő mérkőzés a `FRITZ` program és Vlagyimir Kramnyik világbajnok összecsapása volt, amely nyolc játszmában döntetlennel végződött. A mérkőzés feltételei az emberi játékosnak sokkal jobban kedveztek, és a hardver egy közönséges PC volt, nem egy szuperszámítógép. Kramnyik véleménye szerint mégis „most már világos, hogy a csúcsprogramok és a világbajnok közel azonos szinten vannak”.

6.1.1. Dámajáték

Az IBM-nél dolgozó Arthur Samuel szabad idejében 1952-től kezdve kifejlesztett egy dámaprogramot, mely a kiértékelő függvényét több ezerszer saját maga ellen játszva maga tanulta meg. Ezt az ötletet a 21. fejezetben tárgyaljuk részletesebben. Samuel programja egy kezdő játékos szintjén kezdett játszani, de miután néhány napig önmagával játszott, már Samuel játékszintjét is felülmúlta (bár Samuel nem számított erős játékosnak). 1962-ben a program Robert Nealyt – a „vak dámajáték” bajnokát is megverte, egy hibás lépése miatt. Sokan azt hitték, hogy ezzel bizonyítást nyert, hogy dámajátékban a gép erősebb, mint az ember, azonban ez így nem igaz. Ha viszont figyelembe vesszük, hogy Samuel számítógépe (egy IBM 704-es) 10 000 szavas memóriával, háttértárként mágnesszalagos egységgel és majdnem egy milliszekundumos ciklusidővel rendelkezett, ez a győzelem az MI nagy eredményeinek egyike marad.

Kevesen próbálták meg túlszárnyalni ezt a teljesítményt, amíg Jonathan Schaeffer és kollégái ki nem fejlesztették a Chinookot, amely közönséges PC-n futott és alfa béta keresést alkalmazott. A Chinook egy olyan előre létrehozott adatbázissal dolgozott, amely az összes nyolc vagy kevesebb bábut tartalmazó 444 milliárd állásból épült fel, hogy a végjátéka hibátlan legyen. 1990-ben az U.S. Openen a Chinook másodikként futott be és megnyerte a jogot, hogy a világbajnoki címért mérkőzzön. Ekkor a program Marion Tinsley képében egy problémába ütközött. Dr. Tinsley már 40 éve világbajnok volt és ez alatt a 40 év alatt összesen csak három játszmát vesztett. A Chinookkal vívott első mérkőzésén elszenvedte a negyedik, majd az ötödik vereségét is, de a bajnokságot 20,5:18,5 pontra megnyerte. 1994 augusztusában a Tinsley és a Chinook közötti világbajnoki mérkőzés idő előtt félbeszakadt, amikor Tinsleynek egészségügyi problémák miatt vissza kellett vonulnia. A Chinook lett a hivatalos világbajnok.

Schaeffer úgy gondolja, hogy megfelelő számítási teljesítménnyel a végjátékok adatbázisát annyira ki lehetne bővíteni, hogy a kezdeti pozíciótól induló előre keresés valamelyik már megoldott pozíciót mindig el tudná érni, vagyis a dámajáték teljesen megoldható lenne (volt, hogy a Chinook már az 5-ik lépésnél győzött). Az ilyen kimerítő elemzést kézi módszerekkel a 3 × 3-as amőbára meg lehet tenni és számítógépen a 4 × 4 × 4-es amőbára (Qubic), a Go-Mokura (öt egy sorban) és a Nine-Men’s Morrisra is elvégezték (Gasser, 1998). Ken Thompson és Lewis Stiller figyelemre méltó munkája (Thompson és Stiller, 1992) az összes öt- és hatfigurás

Created by XMLmind XSL-FO Converter.

sakkvégjátszmát is megoldotta, az eredményeket az interneten hozzáférhetővé téve. Stiller egy olyan kényszermatt esetét is felfedezte, amely 262 lépést igényelt. Ez egy kis felfordulást okozott, mert a sakk szabályai megkövetelik, hogy 50 lépés alatt valamilyen „előrehaladásnak” kell történnie.

6.1.2. Othello

Az Othello vagy más néven Reversi számítógépes játékként talán népszerűbb, mint táblajátékként. Keresési tere kisebb, mint a sakké, általában 5-15 megengedett lépés van, de a kiértékelő szaktudást a semmiből indulva kellett kifejleszteni. 1997-ben a Logistello program (Buro, 2002) az emberi világbajnokot, Takeshi Murakamit, képes volt hat nullára megverni. Általánosságban elfogadott, hogy az Othellóban az emberek nem ellenfelei a számítógépeknek.

6.1.3. Ostábla

A 6.5. alfejezetben elmagyaráztuk, hogy a kockadobásból eredő bizonytalanság a mély keresést drága luxussá teszi. Az ostáblára vonatkozó kutatások zöme a kiértékelő függvény javítására összpontosult. Gerry Tesauro (Tesauro, 1992) neurális hálós technikákkal (lásd 20. fejezet) ötvözte Samuel megerősítéses tanulási módszerét, hogy egy figyelemre méltó kiértékelő függvényt hozzon létre, amit 2-es, illetve 3-as mélységű kereséssel együtt használt. A saját magával játszott több mint egy millió tanító játszma után Tesauro programja, a `TD-GAMMON` a világranglistán stabilan az első három játékos között foglal helyet. A program eredményei alapján a játék nyitó lépéseit tekintve egyes esetekben az eddigi gyakorlat radikálisan megváltozott.

6.1.4. Gó

Ázsiában a gó a legnépszerűbb táblajáték, ami a mesterektől legalább olyan felkészültséget igényel, mint a sakk. A 19 × 19-es tábla miatt az elágazási tényező 361-gyel indul, így a hagyományos keresési algoritmusok teljesen használhatatlanok. Egészen 1997-ig kompetens programoknak híre-hamva sem volt, most azonban már programok is képesek elfogadható lépéseket tenni. A legjobb programok többsége alakzatfelismerési technikákat (amikor adott alakzat előfordul, akkor bizonyos lépés megtételét kell megfontolni) korlátos kereséssel (annak eldöntésére, hogy bizonyos köveket el lehet-e fogni, egy lokális területen maradva) kombinál. A könyv írásának idején a legerősebb programok valószínűleg Chen Zhixing Goemate és Michael Reiss Go4++ programja voltak, mindegyiket kb. 10 kyu-ra (gyenge amatőrszint) értékelték. A gó olyan területnek tűnik, ami a kifinomult következtetési módszereket alkalmazó intenzív kutatásokból valószínűleg nyerhet. Sikert hozhat például az, hogy megtaláljuk annak a módját, hogy sok, lazán csatolt „részjátszmára” (amire a gó dekomponálható) vonatkozó lokális következtetés néhány szálát hogyan kellene integrálni. Az ilyen technikák az intelligens rendszerek szempontjából hihetetlen értéket képviselnek.

6.1.5. Bridzs

A bridzs egy nem tökéletes információjú játék – az egyik játékos kártyái a többi játékos elől el vannak rejtve. A bridzs *többszemélyes* játék is egyben, négy játékossal kettő helyett, bár a játékosok két csapatot alkotnak, párban. Ahogyan ezt a 6.5. alfejezetben láttuk, egy bridzsparti optimális lejátszása az információgyűjtés, a kommunikáció, a blöffölés és a valószínűségek gondos mérlegelésének az elemeit is tartalmazhatja. E technikákból számos elemet használ a Bridge Baron™ program (Smith és társai, 1998), amely megnyerte az 1997-es Számítógépes Bridzsbajnokságot. Bár nem játszik optimálisan, a Bridge Baron azon ritka sikeres, játékot játszó programok egyike, amelyek hierarchikus tervkészítési technikákat alkalmaznak (lásd 12. fejezet), olyan magas szintű, bridzsjátékosok számára ismerős ötleteket felhasználva, mint az **impasszolás** (**finessing**) és a **dobáskényszer** (**squeezing**).

A GIB program (Ginsberg, 1999) meggyőző fölénnyel nyerte meg a 2000. évi bajnokságot. A GIB a „jövőbe látás szerinti átlagolás” módszerét használja, két lényegi módosítással. Először, ahelyett hogy megvizsgálná, hogy egy adott választás mennyire bizonyul jónak a rejtett kártyák minden lehetséges kombinációja mellett (amiből 10 millió is lehet), a program egy véletlenszerűen választott 100 kombinációból álló mintát vizsgál. Másodszor, a GIB **magyarázatalapú általánosítás**t (**explanation-based generalization**) használ arra, hogy különféle standard helyzetekre az optimális játékvezetés általános szabályait kiszámítsa és eltárolja. Ez lehetővé teszi minden leosztás *egzakt* megoldását. A GIB taktikai pontossága ellensúlyozza azt, hogy az információra következtetni nem tud. Az 1998-as emberi világbajnokságon az egyenlő rangú mérkőzésen (egy leosztás lejátszásában) 35-ből 12-ikként végzett, az emberi szakértők elvárásait messze túlszárnyalva.

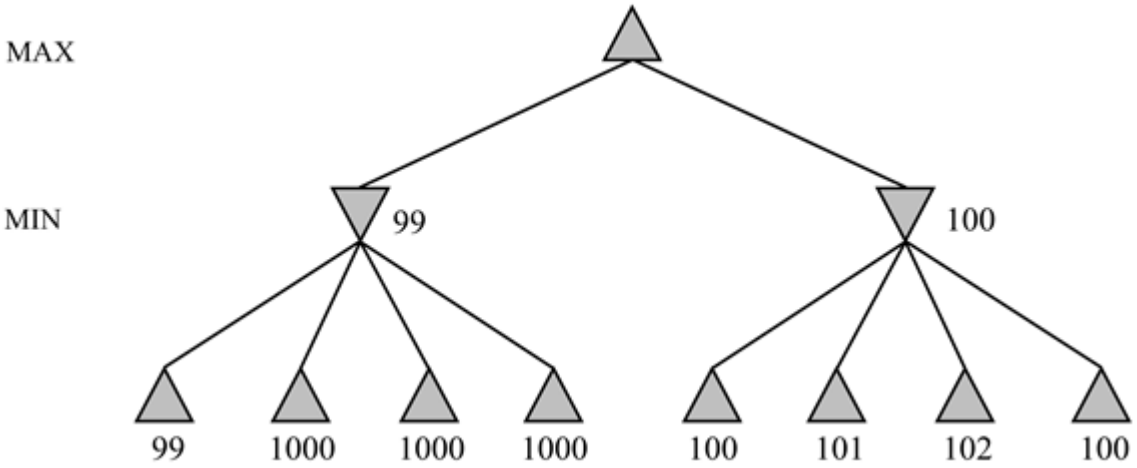
7. Értékelés

Created by XMLmind XSL-FO Converter.

Mivel a játékokban az optimális döntés kiszámítása a legtöbb esetben kezelhetetlen, ezért az összes algoritmusnak valamilyen feltételezéssel és közelítéssel kell élnie. A minimax algoritmuson, kiértékelő függvényeken és az alfa-béta algoritmuson alapuló standard megközelítés csak egy lehetséges megoldás. Valószínűleg a standard megközelítés azért emelkedik ki a versenyjátékokra alkalmazott többi módszer közül, mert korán vetették fel és intenzíven fejlesztették. A terület néhány szakértője úgy gondolja, hogy a játékok épp ezért váltak el az MI fő irányvonalától, mert a standard megközelítés már nem hagy elegendő teret ahhoz, hogy új betekintést nyerjünk a döntéshozatal általános kérdéseibe. Ebben a részben az alternatív megközelítésekre fogunk rápillantani.

Először tekintsük a minimax algoritmust. A minimax optimális módszert ad egy lépés kiválasztásához egy adott keresési fából, *feltéve, hogy a levélcsomópontok kiértékelései tökéletesen pontosak.* A valóságban azonban a kiértékelések az állás értékének durva becslői, és úgy vehetjük, hogy nagy hibával rendelkeznek. A 6.13. ábra egy olyan egylépésváltásos játékfát mutat, amire a minimax alkalmatlannak tűnik. A minimax a jobb oldali ágat javasolja, holott nagyon valószínű, hogy a bal oldali ág valódi értékei nagyobbak. A minimax azon a feltételezésen alapul, hogy a 100-as, a 101-es, a 102-es és a 100-as címkét viselő csomópontok *mindegyike valóban* jobb, mint a 99-es címkét viselő csomópont. Az a tény azonban, hogy a 99-es címkéjű csomópontnak 1000-es címkéjű testvérei vannak, azt sugallja, hogy a csomópont valódi értéke valójában magasabb. Ezen probléma kezelésének egyik lehetséges módja, ha a kiértékelés egy, a lehetséges értékek feletti *valószínűség-elosztást* ad vissza. Ekkor standard statisztikai módszerekkel ki lehet számítani a szülő értékének valószínűség-eloszlását. Sajnos a testvércsomópontok értékei általában erősen korreláltak, ezért ez költséges számítás lehet, mert olyan részletes ismeretre van szükség, amit általában nehéz beszerezni.

6.13. ábra - Egy olyan egylépésváltásos játékfa, amire a minimax alkalmatlannak tűnik



A következőkben tekintsük a fát generáló kereső algoritmust. Az algoritmustervező célja, hogy olyan algoritmust adjon meg, ami megfelelő idő alatt lefut, és jó lépést választ. Az alfa-béta algoritmus legszembetűnőbb problémája, hogy nem arra tervezték, hogy egyszerűen kiválasszon egy jó lépést, hanem arra, hogy értékkorlátokat számítson ki az összes megengedett lépés értékére. Hogy lássuk, ez a többletinformáció miért szükségtelen, tekintsünk egy állást, ahol egyetlen megengedett lépés létezik. Ettől függetlenül az alfa-béta keresés egy hatalmas és teljesen haszontalan keresési fát generál és értékel ki. Természetesen beiktathatunk egy tesztet az algoritmusba, de ezzel csak elrejtjük a problémát – az alfa-béta algoritmus által elvégzett számítások nagy része egyáltalán nem releváns. Az az eset, amikor csak egyetlen megengedett lépés létezik, nem sokban tér el attól az esettől, amikor néhány megengedett lépés létezik, ami közül egy jó, a többi pedig nyilvánvalóan katasztrofális. Egy ilyen „egyértelműen favorit helyzetben” szerencsésebb lenne egy kis keresés után gyors döntésre jutni, mintsem az időt fecsérelni, amit később egy problémásabb helyzetben esetleg hasznosabban fel lehetne használni. Ez elvezet a *csomópontkifejtés hasznosságának* gondolatához. Egy jó keresési algoritmusnak nagy hasznosságú csomópontot kell kiválasztania kifejtésre, vagyis olyat, ami valószínűleg egy lényegesen jobb lépés felfedezéséhez vezet. Ha nincs olyan csomópontkifejtés, aminek a hasznossága nagyobb, mint a kifejtésének (időre vonatkoztatott) költsége, akkor az algoritmusnak be kell fejeznie a keresést, és lépnie kell. Vegyük észre, hogy ez nemcsak az egyértelmű favorithelyzetekre működik, hanem a *szimmetrikus* lépésekre is, ahol bármennyi keresést is végeznénk, egyik lépés sem mutatkozna jobbnak.

Azt a fajta következtetést, amikor megmondjuk, hogy milyen számításokat kell végrehajtani, **metakövetkeztetés**nek (**metareasoning**) nevezik (következtetés a következtetésről). Ez nemcsak a játékokra

Created by XMLmind XSL-FO Converter.

vonatkozik, hanem mindenfajta következtetésre is. Minden számítást azért végzünk el, hogy jobb döntéseket hozhassunk. Mindegyiknek van valamilyen költsége, és mindegyik valamilyen valószínűséggel bizonyos javulást eredményezhet a döntés minőségében. Az alfa-béta algoritmus a lehető legegyszerűbb metakövetkeztetést foglalja magában: olyan tételt, miszerint a fa bizonyos ágait veszteség nélkül figyelmen kívül lehet hagyni. Ennél sokkal jobbat is lehet tenni. A 16. fejezetben megnézzük, hogy ezeket a gondolatokat hogyan lehet pontossá és megvalósíthatóvá tenni.

Végezetül ismét vizsgáljuk meg magának a keresésnek a természetét. A heurisztikus keresés és a játékok algoritmusai úgy működnek, hogy a kiinduló állapotból induló konkrét állapotsorozatokat generálnak, majd alkalmaznak egy kiértékelő függvényt.

Az emberek nem így játszanak. A sakkban gyakran van egy konkrét célunk – például le akarjuk ütni az ellenfél vezérét –, és ezt a célt használhatjuk arra, hogy a cél elérése érdekében *szelektíven* megfelelő terveket generáljunk. Ez a fajta **célvezérelt következtetés** (**goal-directed reasoning**) vagy **tervkészítés** (**planning**) néha teljesen eltünteti a kombinatorikus keresést (lásd IV. rész). David Wilkins `PARADISE` programja (Wilkins, 1980) az egyetlen olyan program, amely sikeresen alkalmazta a célvezérelt következtetést a sakkra: néhány 18 lépésből álló kombinációt igénylő sakkfeladványt is képes volt megoldani. Ez idáig azonban még nem látjuk tisztán, hogyan kellene a két algoritmustípust egyetlen robusztus és hatékony rendszerbe *ötvözni,* bár a Bridge Baron™ jelenthet egy, a helyes irányban tett lépést. Egy ilyen teljesen integrált rendszer jelentős eredménynek számítana nemcsak a játékkutatás területén, hanem általánosságban véve az MI-kutatás területén is, mert egy általános intelligens ágens számára megfelelő alapot is tudna nyújtani.

8. Összefoglalás

A játékok sokaságát néztük meg, hogy megértsük, mit is jelent optimálisan és a gyakorlatban is jól játszani. A legfontosabb gondolatokat az alábbiakban foglalhatjuk össze:

- Egy játékot a **kiinduló állapot**tal (**initial state**) (a táblaállással), mindegyik állapotban a **legális cselekvés**ekkel (**action**s), egy **végtesztt**el (**terminal test**) (ami megmondja, hogy mikor ért véget a játék) és egy **hasznossági függvénny**el (**utility function**) (ami megmondja, hogy ki és mennyivel nyert) lehet megadni.

- **Tökéletes információ**val (**perfect information**) rendelkező kétszemélyes zérusösszegű játékoknál a **minimax algoritmus** a teljes játékfa mélységi felsorolásával meg tudja határozni a játékos legjobb lépését.

- Az **alfa-béta algoritmus** ugyanazt a számítást végzi el, mint a minimax algoritmus, azonban jóval hatékonyabb, mivel lenyesi a keresési fa azon ágait, amiről be tudja bizonyítani, hogy a végső eredmény szempontjából irrelevánsak.

- A teljes játékfa általában nem kezelhető (még az alfa-béta algoritmussal sem), ezért a keresést valahol abba kell hagynunk, és egy **kiértékelő függvény**t (**evaluation function**) kell alkalmaznunk, ami az adott állapot hasznosságának becslőjét adja.

- A véletlen elemet is tartalmazó játékokat a minimax algoritmus olyan kiterjesztésével lehet kezelni, amely a **véletlen csomópont**okat (**chance node**) úgy értékeli ki, hogy az egyes hasznosságokat a gyermekcsomópontok valószínűségével súlyozva veszi az összes gyermekcsomópontjának az átlagos hasznosságát.

- A **nem tökéletes információjú játék**ok (**imperfect information**), mint például a bridzs, optimális lejátszása mindegyik játékostól az aktuális és a jövőbeli **hiedelmi állapot**aira (**belief state**s) vonatkozó következtetést igényli. Egy egyszerű közelítés kapható, ha átlagoljuk egy cselekvés értékét a hiányos információ minden lehetséges konfigurációjára.

- A programok egyenlő ellenfelek, vagy akár meg is verik a legjobb emberi játékosokat dámajátékban, Othellóban és ostáblajátékban, és igen közel állnak ehhez a bridzsben. Egy program legyőzte egy kirakatmérkőzésen az emberi sakkvilágbajnokot. A góban a programok még amatőr szinten játszanak.

8.1. Irodalmi és történeti megjegyzések

A mechanikus játékok korai történetére számos csalás rányomta a bélyegét. Ezek közül a legnevezetesebb Kempelen Farkas báró 1769-ben kiállított „Törökje”, egy feltételezett sakkautomata, amely Napóleont is

Created by XMLmind XSL-FO Converter.

megverte, mielőtt kiderült, hogy a szekrénye valójában egy törpe növésű emberi sakkmestert rejtett (Levitt, 2000). A Török 1769-től 1854-ig játszott. Úgy tűnik, Charles Babbage (aki a Török hatása alatt állt) volt az első, aki 1846-ban számítógépes sakk és dámajáték megvalósíthatóságának első komoly elemzését adta (Morrison és Morrison, 1961). Tervezett egy 3 × 3-as amőbát játszó célgépet is, amit soha sem épített meg. Az első igazi játékgépet 1890 táján egy spanyol mérnök, Leonardo Torres y Quevedo tervezte és építette meg. A „KRK" a sakkvégjátékra specializált gép volt (király és bástya a király ellen) és képes volt bármilyen kiinduló állásból mattot adni.

A minimax algoritmust sokszor Ernst Zermelo, a modern halmazelmélet atyja 1912-es cikkéhez vezetik vissza. A cikk sajnálatos módon tartalmaz hibákat, és a minimaxot nem írja le helyesen. A játékelmélet komoly alapjait a nagy hatású *Theory of Games and Economic Behavior* c. munkában Neumann és Morgenstern (Neumann és Morgenstern, 1944) fektették le, kimutatva azt is, hogy egyes játékokban *szükség* van randomizált (avagy nem megjósolható) stratégiákra. (További információért lásd 17. fejezet.)

A korai számítógépes korszak számos prominens személyét kíváncsivá tette a számítógépes sakk lehetősége. Konrad Zuse – aki elsőként tervezett programozható számítógépet – igen részletes ötleteket dolgozott ki arra, hogy ezt hogyan lehetne megvalósítani (Zuse, 1945). Norbert Wiener nagy befolyású *Cybernetics* c. könyve (Wiener, 1948) tartalmazta egy számítógépes sakkprogram működésének egy lehetséges vázlatát, a minimax keresést, a mélységi levágást és a kiértékelő függvényt is beleértve. Claude Shannon a modern számítógépes játékok elvi alapjait sokkal részletesebben fejtette ki, mint Wiener (Shannon, 1950). Shannon bevezette az egyensúlyi állás fogalmát, és néhány ötletet vázolt fel a szelektív (nem kimerítő) játékfakeresésre vonatkozólag. Slater és az ugyanabban a kötetben a cikkére reflektáló szerzők szintén megvizsgálták a számítógépes sakkozás lehetőségeit (Slater, 1950). I. J. Good Shannontól függetlenül kidolgozta az egyensúlyi állás fogalmát (Good, 1950).

1951-ben Alan Turing írta meg az első valódi számítógépes programot, ami képes volt egy teljes sakkjátszmát lejátszani (Turing és társai, 1953). Valójában azonban Turing programja sohasem futott számítógépen, kézi szimulációval tesztelték egy nagyon gyenge emberi sakkjátékos ellen, aki legyőzte a programot. Időközben D. G. Prinz megírt és valóban futtatott is egy programot (Prinz, 1952), ami sakkfeladványokat oldott meg, bár nem játszott teljes játszmát. Alex Bernstein írta az első olyan sakkprogramot, amely egy teljes standard sakkjátszmát játszott (Bernstein és Roberts, 1958; Bernstein és társai, 1958).[6]

Az alfa-béta keresés alapötletét John McCarthy dolgozta ki 1956-ban, habár nem publikálta. Az NSS-sakkprogram az alfa-béta algoritmus egy leegyszerűsített változatát használta, ez volt az első sakkprogram, ami ezt alkalmazta (Newell és társai, 1958). Nilsson szerint Arthur Samuel dámaprogramja (Samuel, 1959; 1967) szintén alfa-béta algoritmust használt, habár maga Samuel ezt nem említi a rendszerről publikált beszámolóiban (Nilsson, 1971). Az 1960-as évek elején jelentek meg az alfa-béta algoritmust ismertető cikkek (Hart és Edwards, 1961; Brudno, 1963; Slagle, 1963b). Az alfa-béta algoritmus egy teljes implementációját Slagle írta le egy cikkben (Slagle és Dixon, 1969), ami a kalah[7] játékot játszó játékprogram működését ismertette. A John McCarthy egyik diákja által írt „Kotok–McCarthy" sakkprogram (Kotok, 1962) is az alfa-béta algoritmust használta. Knuth ismerteti az alfa-béta algoritmus történetét (Knuth és Moore, 1975), megadja az algoritmus teljességének a bizonyítását és elvégzi az időigény elemzését. Knuth és Moore elemzése az alfa-béta keresésnek a követők véletlen sorba rendezésével $O((b/\log b)^d)$ aszimptotikus komplexitást mutatott ki, ami lehangoló eredmény, mert a $b/\log b$ effektív elágazási tényező magánál a b-nél nem sokkal jobb. Később jöttek rá, hogy az aszimptotikus képlet csak a $b > 1000$ igaz, és az aktuális játékokra található elágazási tényezőkre a gyakran idézett $O(b^3d^4)$ érvényes. Pearl megmutatta (Pearl, 1982b), hogy az alfa-béta algoritmus aszimptotikusan optimális az összes rögzített mélységű játékfa-keresési algoritmus között.

Az első két sakkprogram, ami egymás ellen játszott, a Kotok–McCarthy-program és a Moszkvai Elméleti és Kísérleti Fizika Intézet által megírt „ITEP" program (Adelson-Velsky és társai, 1970) volt az 1960-as évek közepén. Ezt az interkontinentális mérkőzést távírón játszották le. A küzdelem 1967-ben az ITEP-program 3:1 arányú győzelmével ért véget. A MacHack 6 volt az első olyan sakkprogram, ami sikeresen játszott emberek ellen (Greenblatt és társai, 1967). 1400-as Élő-pontszáma jóval több volt, mint a kezdők 1000-es Élő-pontja, azonban így is igen messze volt a 2800 vagy több Élő-ponttól, ami szükséges lett volna, hogy Herb Simon 1957-es jóslata teljesüljön, miszerint 10 éven belül a számítógépes sakkprogramok lesznek a sakkvilágbajnokok (Simon és Newell, 1958).

[6] Newell (Newell és társai, 1958) egy orosz BESM programot említ, amely lehet, hogy Bernstein programját megelőzte.

[7] A kalah egy afrikai eredetű kétszemélyes játék, amelyben kavicsokat (gyémántokat) kell átrakosgatni csészék között. Több számítógépes változata is létezik, l. például: *http://ceman.ecn.purdue.edu/~ee373/kalah.html.* (*A ford.*)

Created by XMLmind XSL-FO Converter.

Az 1970-es első ACM Észak-Amerikai Számítógépes Sakkbajnoksággal a sakkprogramok versengése komollyá vált. A korai 1970-es évek programjai igen bonyolultak voltak, számos trükköt vetettek be, hogy a keresés bizonyos ágait levágják, elfogadható lépéseket generáljanak stb. Az első Számítógépes Sakkvilágbajnokságot 1974-ben Stockholmban rendezték. Ezt az első világbajnokságot a Kaissa (Adelson-Velsky és társai, 1975), egy másik ITEP-program nyerte meg. Kaissa egy egyszerűbb megközelítésen alapult és kimerítő alfa-béta keresést használt egyensúlyi kereséssel vegyítve. A megközelítés felsőbbrendűségét igazolta a CHESS 4.6 győzelme az 1977-es Számítógépes Sakkvilágbajnokságon. A CHESS 4.6 lépésenként 400 000 állást elemzett és 1900 Élő-pontot ért el.

Greenblatt MacHack 6-osának egy későbbi változata volt az első olyan sakkprogram, amely már kifejezetten a sakkra tervezett célhardveren futott (Moussouris és társai, 1979), de a Belle (Condon és Thompson, 1982) volt az első olyan program, ami a célhardvernek köszönhetően jelentős sikereket ért el. A Belle lépésgeneráló és állásértékelő hardvere lehetővé tette, hogy lépésenként néhány millió állást is elemezzen. A Belle 2250 Élő-pontot ért el, és az első mesteri fokozat szintű program lett. A HITECH rendszer egy speciális rendeltetésű számítógép volt, amit Hans Berliner, a korábbi levelező sakkvilágbajnok és egy CMU-beli diákja, Carl Ebeling tervezett, hogy a kiértékelő függvények gyors számítását tegyék lehetővé (Ebeling, 1987; Berliner és Ebeling, 1989). A HITECH észak-amerikai sakkbajnok lett 1985-ben, és 1987-ben az első olyan program volt, amely egy emberi nagymestert is legyőzött. A CMU-n szintén kifejlesztett Deep Thought (Hsu és társai, 1990) a tiszta keresés sebességét tovább fokozta. 2551 Élő-pontot ért el és a Deep Blue előfutára lett. Az 1980-ban alapított Fredkin-díj 5000 dollárt ajánlott fel annak a sakkprogramnak, amelyik elsőként éri el a mesterfokozatot, és 10 000 dollárt ajánlott fel annak a sakkprogramnak, amely elsőként éri el a USCF (Amerikai Egyesült Államok Sakkszövetsége) 2500-as Élő-pontot (ez a nagymesteri szinthez közeli érték), és 100 000 dollárt ajánlott fel azon sakkprogramnak, amelyik elsőként legyőz egy emberi sakkvilágbajnokot. Az 5000 dolláros díjat 1993-ban a Belle, a 10 000 dolláros díjat 1989-ban a Deep Thought, majd a 100 000 dolláros díjat 1997-ben a Deep Blue nyerte el a Kaszparov felett aratott győzelemért. Fontos emlékezni, hogy a Deep Blue sikerét mind az algoritmikus javítások, mind a hardver biztosította (Hsu, 1999; Campbell és társai, 2002). Az olyan technikák, mint a nulla lépés heurisztika (Beal, 1990) a keresésben igen szelektív programokhoz vezettek. Az utolsó három Számítógépes Sakkvilágbajnokságot 1992-ben, 1995-ben és 1999-ben a standard PC-n futó programok nyerték meg. Egy korszerű sakkprogramnak talán a legrészletesebb leírását Ernst Heinz adja meg (Heinz, 2000), amelynek DARKTHOUGHT programja volt a legmagasabb pontszámú nem kereskedelmi program az 1999-es világbajnokságon.

Kísérletek történtek arra, hogy a „standard megközelítés” 6.7. alfejezetben leírt problémáit leküzdjék. Az első szelektív kereső algoritmus, elméleti igénnyel, valószínűleg a B* algoritmus (Berliner, 1979) volt, ami megkísérli, hogy a játékfa csomópontjainak az értékeire egyetlen becsült érték helyett intervallumkorlátokat adjon. A levélcsomópontok kifejtése annak érdekében történik, hogy a legfelső szintű korlátokat finomítsuk, amíg egy „nyilván legjobb” lépést nem találunk. Palay az alfa-béta algoritmus becsült értékei, illetve a B* algoritmus intervallumai helyett valószínűség-eloszlásokat használ (Palay, 1985). David McAllester konspirációs szám keresési algoritmusa azokat a levélcsomópontokat fejti ki, amelyek, ha megváltoznának az értékeik, előidéznék, hogy az algoritmus a gyökérnél új lépést válasszon (David McAllester, 1988). Az MGSS* (Russell és Wefald, 1989) a 16. fejezet fejlett döntéselméleti technikáit használja minden levélcsomópont kifejtésének értékbecslésére a gyökérszintű döntés minőségében tapasztalt várható javulás függvényében. Ez a program az Othellóban képes volt jobb eredményeket elérni, mint az alfa-béta algoritmus, annak ellenére, hogy egy nagyságrenddel kevesebb csomópontot vizsgált meg. Az MGSS* megközelítés elvben alkalmas a következtetés bármilyen formájának vezérlésére.

Az alfa-béta keresés több szempontból a mélységi áglenyeső megfelelője két játékos esetén, amit az egyedi ágens esetében az A* dominál. Az SSS* algoritmus (Stockman, 1979) egy kétszemélyes A*-nak tekinthető, és azonos döntés eléréséhez soha nem fejt ki több csomópontot, mint az alfa-béta algoritmus. Eredeti formájában az SSS* memóriaigénye és a sorba állítás számítási overheadje miatt nem praktikus, azonban az RLEK algoritmusból kifejlesztettek egy lineáris tárkomplexitású változatot (Korf és Chickering, 1996). Plat az SSS*-ja egy új megközelítést dolgozott ki (Plat és társai, 1996) az alfa-béta keresés és a transzpozíciós táblák együtteseként, és megmutatta, hogy az eredeti algoritmus problémáit hogyan kell elkerülni, végül MTD(*f*) néven egy új változatot fejlesztett ki, amit sok kiemelkedő teljesítményű programba építettek be.

D. F. Beal és Dana Nau a minimax közelítő kiértékelésekre történő alkalmazásoknál jelentkező gyengeségeit tanulmányozták (Beal, 1980; Dana Nau, 1980; 1983). Kimutatták, hogy a levélértékeknek a fában való eloszlására bizonyos függetlenségi feltételezésekkel élve, a minimax algoritmus *kevésbé* megbízható becsléseket ad a gyökérre, mintha a kiértékelő függvényeket közvetlenül, mindenféle keresés nélkül alkalmaznánk. Pearl *Heuristics* c. könyve (Pearl, 1984) ezt a látszólagos paradoxont részben megmagyarázza és számos játékalgoritmust elemez. Baum és Smith (Baum és Smith, 1997) a minimax egy valószínűség-alapú

Created by XMLmind XSL-FO Converter.

helyettesítését javasolja, és azt mutatja ki, hogy ez bizonyos játékokban jobb döntésekhez vezet. A keresés különféle szinteken való levágásából és a kiértékelő függvény alkalmazásából adódó hatások elmélete még mindig szegényes.

A várhatóminimax algoritmust Donald Michie vetette fel (Michie, 1966), habár az közvetlenül következik Neumann és Morgenstern játékfa-kiértékelési elméletéből. Bruce Ballard az alfa-béta nyesést kiterjesztette a véletlen csomópontokat is tartalmazó fákra (Bruce Ballard, 1983). Az első sikeres ostáblaprogram a BKG volt (Berliner, 1977; 1980b). Csak 1 mélységig keresett és bonyolult, manuálisan összeállított kiértékelő függvényt használt. Ez volt az első számítógépes program, ami képes volt legyőzni egy emberi világbajnokot az ismert klasszikus táblajátékok egyikében (Berliner, 1980a), habár Berliner elsőként ismerte el, hogy ez csak egy rövid, bemutató mérkőzés volt (nem egy világbajnoki mérkőzés), és hogy a BKG nagyon szerencsés „kézzel” dobott a kockával. Gerry Tesauro kutatásai, először a NEUROGAMMON (Tesauro, 1989), majd a TD-GAMMON (Tesauro, 1995) programokkal, azt mutatták, hogy sokkal jobb eredmény érhető el a megerősítéses tanulással (amely területtel a 21. fejezetben foglalkozunk).

A dáma, és nem a sakk volt az első olyan klasszikus játék, amire egy valóban számítógépen futó program képes volt végigjátszani egy teljes játszmát. Christopher Strachey írta az első működőképes dámajátékprogramot (Strachey, 1952). Schaeffer (Schaeffer, 1997) nagyon olvasmányosan leírja a Chinook – a világbajnok dámajátékprogram – fejlesztéstörténetét az összes műhelytitkával együtt.

Az első góprogramokat valamivel a dáma- és a sakkprogramok után fejlesztették ki (Lefkovitz, 1960, Remus, 1962), és lassabban fejlődtek a dáma- és a sakkprogramoknál. Ryder tiszta keresésalapú megközelítést használt (Ryder, 1971) a szelektív nyesési módszerek egész palettájával, hogy le tudja győzni a hatalmas elágazási tényezőből adódó problémát. Zobrist feltétel-cselekvés szabályokat használt elfogadható lépések generálására, ha a játékban ismert alakzatok jelentek meg (Zobrist, 1970). Reitman és Wilcox jó eredménnyel kombinálta a szabályokat és a keresést (Reitman és Wilcox, 1979), a legtöbb modern program követte ezt a hibrid megközelítést. A számítógépes gó jelenlegi állását Müller (Müller, 2002) foglalja össze, és tömérdek hivatkozást is közöl. Anshelevich (Anshelevich, 2000) hasonló módszereket a HEX játékra alkalmaz. A friss fejleményekről a Számítógépes Go Szövetség által kiadott *Computer Go Newsletter* számol be.

A számítógépes játékokról több fórumon jelennek meg cikkek. A felettébb félrevezető nevű konferenciakiadvány, a *Heuristic Programming in Artificial Intelligence* számol be a Számítógépes Sakkolimpiákról, amely rendezvények a játékok egész sorára terjednek ki. A játékprogramok kutatásáról fontos cikkeket publikáltak számos szerkesztett cikkgyűjteményben (Levy, 1988a; Levy, 1988a; Marsland és Schaeffer, 1990). Az 1977-ben alapított Nemzetközi Számítógépes Sakkszövetség (International Computer Chess Association, ICCA) negyedévenként adja ki az *ICGA Journal*t (korábban *ICCA Journal*). (Clarke, 1977) óta fontos cikkek jelentek meg az *Advances in Computer Chess* antológiasorozatban. Az *Artificial Intelligence* folyóirat 134. kötete (*Artificial Intelligence*, 2002) leírásokat tartalmaz a legfejlettebb sakk, Othello, Hex, shogi, gó, ostábla, póker, Scrabble™ és más játékprogramokról.

8.2. Feladatok

6.1.

Ez a feladat a játékok alapvető elveit a 3 × 3-as amőbán (körök és ikszek) keresztül gyakoroltatja be. X_n azon sorok, oszlopok vagy átlók számát jelöli, ahol pontosan n db X, míg egyetlen O sem található. Hasonlóan, O_n azon sorok, oszlopok vagy átlók számát jelöli, ahol pontosan n db O, míg egyetlen X sem található. A hasznosságfüggvény +1-et rendel minden olyan álláshoz, ahol $X_3 = 1$, és –1-et rendel minden olyan álláshoz, ahol $O_3 = 1$. Az összes többi végállás hasznossága 0. A nem végállapotok esetén az alábbi módon definiált lineáris kiértékelő függvényt fogjuk használni:

$$Kiértékel(s) = 3X_2(s) + X_1(s) - (3O_2(s) + O_1(s))$$

a. Körülbelül hány különböző 3 × 3-as amőbajátszmát lehet lejátszani?

b. Üres táblából kiindulva, a szimmetriát is figyelembe véve, 2-es mélységig (vagyis egy X és egy O a táblán) mutassa meg a teljes játékfát.

c. A fában 2-es mélységben tüntesse fel az összes álláskiértékelést.

Created by XMLmind XSL-FO Converter.

d. A fában az 1-es és a 0-s mélységben levő állásokhoz a minimax algoritmust felhasználva tüntesse fel a felfelé terjesztett értékeket, és használja fel azokat a legjobb kezdő lépés meghatározásához.

e. Karikázza be a 2-es szinten levő azon csomópontokat, amelyeket az alfa-béta nyesés alkalmazása esetén az algoritmus *nem* értékelne ki, feltételezve, hogy a csomópontokat az algoritmus *az alfa-béta nyeséshez optimális sorrendben generálja.*

6.2.

Bizonyítsa be az alábbi állítást: minden játékfa esetén a `MAX` által kapott hasznosság, amikor `MAX` a minimax döntést használva játszik egy szuboptimális `MIN`-nel szemben, sohasem lesz alacsonyabb, mint az a hasznosság, ami akkor ér el, ha egy optimális `MIN`-nel szemben játszana. Ki tud találni egy olyan játékfát, ahol `MAX` egy szuboptimális stratégiával egy *szuboptimális* `MIN`-nel szemben még jobban teljesítene?

6.3.

Tekintse a 6.14. ábrán látott kétszemélyes játékot.

a. Rajzolja fel a teljes játékfát az alábbi konvenciót alkalmazva:

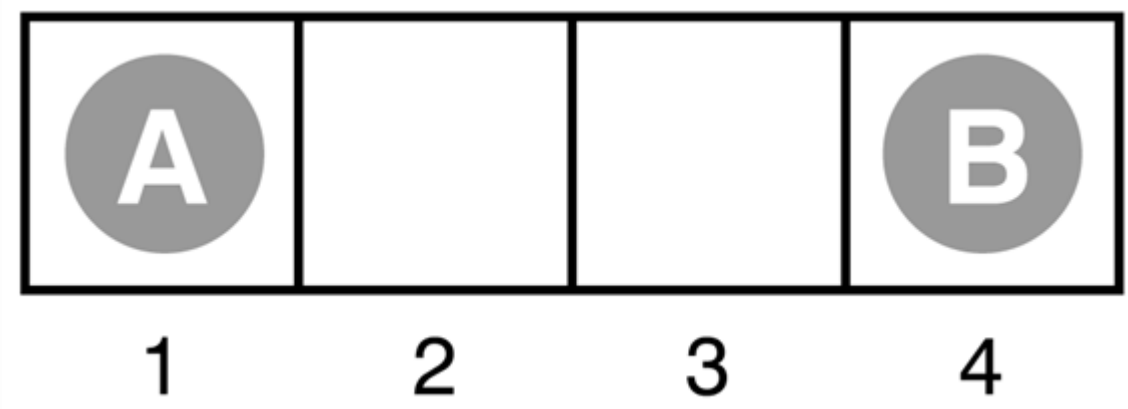
- minden állapotot (S_A, S_B) formában írja fel, ahol S_A és S_B a zsetonok helyzete.
- minden végállapotot egy négyzetbe, a játékértékét pedig egy körbe írja be.
- a *hurokállapotokat* (azon állapotok, amelyek már megjelentek a gyökérig vezető út mentén) dupla négyzetekbe írja be. Mivel nem világos, hogy az ilyen állapotok értéke mennyi, jelölje be mindegyiket egy körbe írt „?”-lel.

b. Most mindegyik csomóponthoz írja be a visszaterjesztett minimax értékét (szintén egy körbe írva). Magyarázza meg, hogy a „?” értékeket hogyan kezelte, és miért?

c. Magyarázza meg, hogy a standard minimax ebben a játékfában miért fulladna kudarcba, és röviden vázolja fel, hogy a (b) kérdésre adott válaszra támaszkodva hogyan lenne képes az algoritmust megjavítani. Képes-e a módosított algoritmus megadni az optimális döntéseket az összes hurkokat tartalmazó játékban?

d. Ez a 4-négyzetes játék *n* négyzet esetére általánosítható, minden *n* > 2-re. Bizonyítsa be, hogy *A* győz, ha *n* páros, és veszít, ha *n* páratlan.

6.14. ábra - Egy egyszerű játék kiinduló állása. Az *A* játékos indul elsőnek. Mindkét játékos felváltva lép és a zsetonját a szomszédos szabad helyre helyezheti, mindkét irányban. Ha a szomszédos mező foglalt, akkor a játékos az ellenfél felett átugorhat a következő szabad helyre, ha van ilyen. (Például ha *A* a 3-n és *B* a 2-n van, akkor *A* visszaléphet 1-re.) A játéknak vége, ha az egyik játékos eléri a tábla ellentétes végét. Ha az *A* játékos elsőnek éri el a 4-et, a játék értéke +1, ha *B* játékos éri el elsőnek az 1-et, a játék értéke az *A* számára –1.



Created by XMLmind XSL-FO Converter.

6.4.

Implementáljon lépésgenerátort és kiértékelő függvényt az alábbi játékok közül néhányat: kalah, Othello, dáma, sakk. Az implementációt használva tervezzen egy általános alfa-béta játékágenst. Hasonlítsa össze a keresési mélység növelésének, a lépéssorrendezés tökéletesítésének és a kiértékelő függvény tökéletesítésének a hatását. Milyen közel esik az Ön effektív elágazási tényezője a tökéletes lépéssorrendezés ideális esetéhez?

6.5.

Állítsa elő az alfa-béta nyesés helyességének formális bizonyítását. Ehhez tekintse a 6.15. ábrán bemutatott helyzetet. A kérdés az, hogy az algoritmus lenyesse-e az n_j csomópontot, ami egy max-csomópont és az n_1 leszármazottja. Az alapötlet, hogy akkor és csak akkor nyessük le, ha n_1 minimax értéke igazolhatóan független n_j értékétől.

a. n_1 értékét az alábbi képlet adja meg:

$$n_1 = \min(n_2, n_{21}, \ldots, n_{2b_2})$$

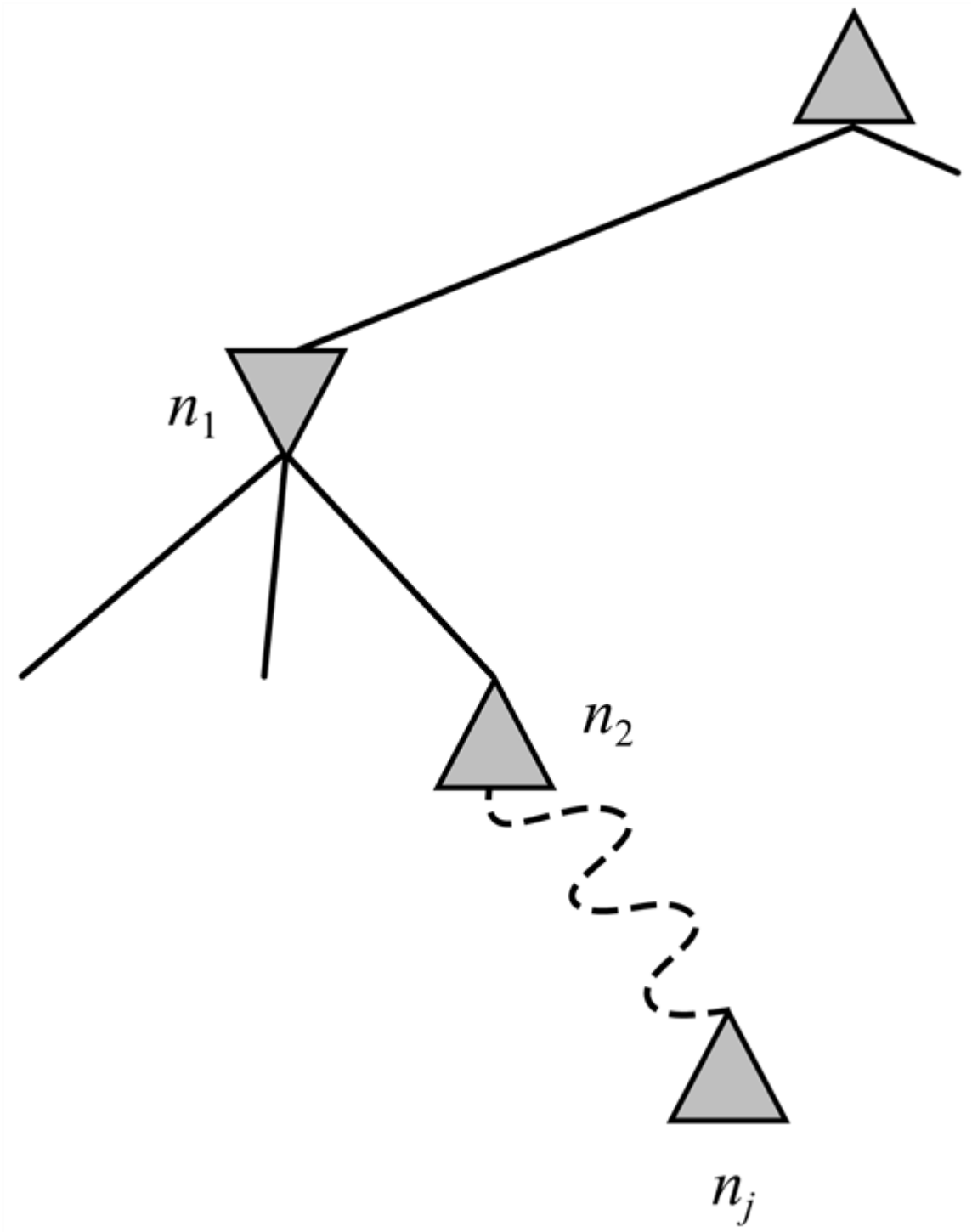
Adjon egy hasonló kifejezést n_2-re, és n_1-re, n_j-t felhasználva.

b. Legyen l_i az n_i csomóponttól balra levő, ismert minimax értékű, i mélységben található csomópontok minimális (maximális) csomópontértéke. Hasonlóan, legyen r_i az n_i csomóponttól jobbra levő, még ki nem fejtett, i mélységben található csomópontok minimális (maximális) csomópontértéke. Az n_1-re előbb meghatározott kifejezést írja át l_i és r_i értékekkel kifejezve.

c. Most fogalmazza át a kifejezést, hogy megmutassa, ahhoz, hogy n_j befolyásolja n_1-et, n_j-nek meg kell haladnia bizonyos, az l_i értékekből meghatározott korlátot.

6.15. ábra - Az a helyzet, amikor az algoritmus eldönti, hogy lenyesheti-e az n_j csomópontot

Created by XMLmind XSL-FO Converter.



d. Az eljárást ismételje meg arra az esetre, amikor az n_j egy min-csomópont.

6.6.

Implementálja a véletlen csomópontokat tartalmazó játékfák lenyesésére alkalmas várhatóminimax és *-alfa-béta algoritmust, amit Ballard (Ballard, 1983) ír le. Próbálja ki azokat az algoritmusokat olyan játékokon, mint az ostábla, és mérje meg a *-alfa-béta nyesési hatékonyságát.

6.7.

Bizonyítsa be, hogy a levélcsomóponti értékek pozitív lineáris transzformációja (azaz egy transzformáció x-től $ax + b$-ig, ahol $a > 0$), nincs befolyással a lépések választékára a játékfában, akkor sem, ha léteznek benne véletlen csomópontok.

Created by XMLmind XSL-FO Converter.

6.8.

Tekintse az alábbi eljárást a lépések megválasztására a véletlen csomópontot is tartalmazó játékokban:

- Generáljon egy megfelelő számú (mondjuk 50) kockadobásból álló sorozatot egy megfelelő (mondjuk 8) mélységig.
- Ismert kockadobások esetén a játékfa determinisztikussá válik. Minden egyes kockadobás-sorozatra oldja meg az eredményül kapott determinisztikus játékfát az alfa-béta algoritmussal.
- Az eredményeket használja fel az egyes lépések értékének megbecslésére, és válassza ki a legjobb lépést.

Helyesen fog működni ez az eljárás? Miért (nem)?

6.9.

Írjon és implementáljon egy *valós idejű, többszemélyes* játszókörnyezetet, ahol az idő a környezeti állapot része, és a játékosok rögzített időszeleteket kapnak.

6.10.

Adja meg és/vagy implementálja az alábbi játékok egyikére vagy akár többre is az állapotleírást, lépésgenerálást és a kiértékelő függvényt: Monopoly, Játék a betűkkel, bridzs (egy konkrét licitet feltételezve) és póker (válassza meg a kedvenc változatát).

6.11.

Gondosan tekintse át a 6.10. feladat minden egyes játékában a véletlen események és a részleges információk összefüggéseit.

a. Melyikre lesz jó a standard várhatóminimax modell? Implementálja az algoritmust és futtassa le a játékágensében a játékkörnyezet szükséges módosításaival.

b. Melyekre lenne jó a 6.8. feladatban leírt séma?

c. Vitassa meg, hogyan kellene kezelni azt a tényt, hogy egyes játékokban a játékosok nem rendelkeznek ugyanazzal az információval az aktuális állapotról.

6.12.

A minimax algoritmus feltételezi, hogy a játékosak felváltva lépnek, az olyan kártyajátékokban azonban, mint a whist vagy a bridzs, mindig a leütés győztese indul a következőnek.

a. Módosítsa algoritmusát, hogy e játékokra is megfelelően működjön. Feltételezheti, hogy rendelkezésére áll a `GYŐZTES` (*leütés*) függvény, amely azt adja vissza, hogy az adott leütést mely kártya nyerte meg.

b. Rajzolja fel a 5.2. szakasz - A várhatóminimax komplexitása részben látható első leosztásra a játékfát.

6.13.

A Chinook dámaprogram kiterjedten használja a végjáték adatbázisokat, amelyek pontos értéket adnak a játék utolsó hat lépésében előálló összes álláshoz. Hogyan lehet egy ilyen adatbázist hatékonyan legenerálni?

6.14.

Vitassa meg, hogyan alkalmazható a játékok standard megközelítése az olyan játékokra, mint például a tenisz, a póló és a krikett, amelyeket folytonos, fizikai állapottérben játszanak.

6.15.

Írja le, hogy a minimax és az alfa-béta hogyan változik kétszemélyes, nem zérusösszegű játékok esetén, ahol minden játékosnak külön kiértékelő függvénye van. Feltételezheti, hogy mindegyik játékos mások kiértékelő

Created by XMLmind XSL-FO Converter.

függvényeit ismeri. Ha a két véghasznosság értékére nincs korlát, lehetséges-e valamelyik csomópont számára, hogy az alfa-béta lenyesi?

6.16.

Tegyük fel, hogy egy olyan sakkprogrammal rendelkezik, amely képes egymillió csomópontot kiértékelni másodpercenként. Válassza a játékállás egy tömör reprezentációját a transzpozíciós tábla számára. Kb. mennyi állást képes eltárolni memóriában, az 500 Mbájt nagyságú táblában? Elég lesz-e ez a lépésenként allokált 3 perces kereséshez? Hány táblakiolvasást képes megvalósítani egy álláskiértékelés ideje alatt? Most tételezze fel, hogy a transzpozíciós tábla nagyobb, és nem fér el a memóriában. Hány kiértékelést lehetne megvalósítani egy diszkhozzáférés ideje alatt, standard diszkhardver mellett?

Created by XMLmind XSL-FO Converter.

7. fejezet - Logikai ágensek

Ebben a fejezetben olyan ágenseket tervezünk, amelyek képesek reprezentációkat kialakítani a világról, következtetési folyamatot alkalmaznak a világ új reprezentációjának a származtatására, és felhasználják az új reprezentációt a teendők kikövetkeztetésére.

Ez a fejezet tudásbázisú ágenseket mutat be. A fogalmak, amelyeket itt tárgyalunk – a tudás *reprezentációja* és a tudás alkalmazását lehetővé tevő *következtetési* folyamatok – központi témái a mesterséges intelligencia minden területének.

Az emberek, úgy tűnik, ismernek a világról számos dolgot, és következtetéseket végeznek. A tudás és a következtetés fontos a mesterséges ágenseknek is, mert sikeres viselkedést tesz lehetővé, amelyet nagyon nehéz volna másképpen elérni. Láthattuk, hogy a problémamegoldó ágens számára cselekvéseinek ismerete lehetővé teszi, hogy komplex környezetekben jól teljesítsen. A reflexív ágens csak a vakszerencse segítségével tudta megtalálni az utat Aradról Bukarestbe. A problémamegoldó ágens tudása azonban igen specifikus és rugalmatlan. Egy sakkprogram ki tudja számítani egy király helyes lépéseit, de nincsen semmilyen értelmezhető ismerete arról, hogy egy figura egyidejűleg nem lehet két különböző mezőn. A tudásbázisú ágens képes kihasználni a nagyon általános formában leírt tudást, újra és újra összegyűjtve ennek elemeit úgy, hogy az számos célra megfelelő legyen. Gyakran ez a folyamat nagyon messzire kerül a pillanatnyi igénytől, például amikor egy matematikus egy tételt bizonyít, vagy amikor egy csillagász a Föld várható élettartamát számítja.

A tudás és a következtetés alapvető szerepet játszanak a *részben megfigyelhető* környezetek kezelésénél is. A tudásbázisú ágens képes összekombinálni az általános tudást a pillanatnyi érzetekkel, hogy kikövetkeztesse a pillanatnyi állapot rejtett aspektusait, mielőtt cselekvést választ. Ilyen például, amikor egy orvos diagnosztizál egy beteget, azaz kikövetkeztet egy közvetlenül nem megfigyelhető betegségállapotot, mielőtt meghatározná a kezelés módját. A tudás egy része, amelyet az orvos használ, könyvekből vagy tanároktól megtanult szabályok formájában áll rendelkezésre, más része pedig asszociációs minták formájában van, amelyeket lehet, hogy az orvos nem is tud tudatosan leírni. Ha ezek is az orvos fejében vannak, akkor tudásnak számítanak.

A természetes nyelv megértése szintén igényli, hogy rejtett állapotokra következtessünk, nevezetesen, hogy a beszélő szándékát megismerjük. Amikor azt halljuk, hogy „János egy gyémántot látott az ablakon keresztül, és szeretné (azt) megkapni”, akkor tudjuk, hogy az „azt” a gyémántra vonatkozik, és nem az ablakra. Ilyenkor, talán nem is tudatosan, a relatív értékekről meglévő ismereteink segítségével következtetünk. Hasonlóan, amikor azt halljuk, hogy „János kidobott egy követ az ablakon keresztül, és betörte (azt)”, akkor tudjuk, hogy az „azt” az ablakra vonatkozik. A következtetés lehetővé teszi számunkra, hogy megbirkózzunk a kijelentések virtuálisan végtelen változatosságával, hétköznapi tudásunk egy véges halmazát felhasználva.

Az utolsó érvünk a tudásbázisú ágensek tanulmányozása mellett e rendszerek rugalmassága. Képesek explicit célok formájában megadott új feladatokat elfogadni, egy új környezetről kapott vagy megtanult új ismeretek révén kompetenssé válni, és frissítve a tudásuk releváns részét képesek alkalmazkodni a környezet változásához.

A 7.1. alfejezetben az ágens általános tervezésével kezdünk. A 7.2. alfejezet egy egyszerű új környezetet, a wumpus világot mutatja be, amelyben illusztrálni fogjuk a tudásbázisú ágens működését a technikai részletek ismertetése nélkül. Ezután a 7.3. alfejezetben elmagyarázzuk a **logika** (**logic**) általános elveit. A könyv III. részében végig a logika lesz az elsődleges eszköz a tudás reprezentálására. A logikai ágensek tudása mindig határozott, minden kijelentés vagy igaz, vagy hamis a világban, habár az ágens lehet agnosztikus néhány kijelentéssel kapcsolatban.

A logika jelentős pedagógiai előnye, hogy egy tudásbázisú ágens egyszerű reprezentációs formáját jelenti, viszont van néhány komoly korlátja is. A helyzet az, hogy az emberek vagy más ágensek által részlegesen megfigyelhető környezetekben végzett következtetések jelentős része függ *bizonytalan* tudás felhasználásától. A logika nem tudja ezt a bizonytalanságot jól reprezentálni, ezért az V. részben a valószínűséget tárgyaljuk, amely már képes erre. A VI. és VII. részben számos reprezentációt tárgyalunk, köztük néhány folytonos matematikán alapulót, mint a Gauss-görbék keverését, neurális hálózatokat és más reprezentációkat.

A 7.4. alfejezet bemutat egy egyszerű logikát, az **ítéletkalkulus**t (**propositional logic**). Miközben ez lényegesen kevésbé kifejező, mint az **elsőrendű logika** (**first-order logic**) (8. fejezet), az ítéletkalkulus lehetőséget ad arra, hogy illusztráljuk a logika minden alapvető fogalmát. Létezik egy jól megtervezett technológia az ítéletkalkuluson történő következtetésre, amelyet a 7.5. és 7.6. alfejezetben írunk le. Végül a 7.7. alfejezet összekombinálja a logikai ágensek fogalmát az ítéletkalkulus technológiájával, hogy egyszerűbb ágenseket

Created by XMLmind XSL-FO Converter.

építsünk a wumpus világ számára. Azonosítjuk majd az ítéletkalkulus bizonyos hiányosságait, amelyek még hatékonyabb logikák fejlesztését fogják motiválni a következő fejezetekben.

1. A tudásbázisú ágens

Egy tudásbázisú ágens központi eleme a **tudásbázis**a (**knowledge base**). Egyszerűen fogalmazva, a tudásbázis **mondat**ok (**sentence**s) halmaza. (Itt a „mondatot” technikai fogalomként használjuk. A fogalom kapcsolatban van az angol, a magyar vagy más természetes nyelvek mondataival, de nem azonos velük.) A mondatokat egy nyelv segítségével fejezzük ki, amelyet **tudásreprezentációs nyelv**nek (**knowledge representation language**) nevezünk, és a világról szóló állításokat fogalmazunk meg vele.

Új mondatoknak a tudásbázishoz való hozzáadására, illetve a tudás lekérdezésére valamilyen eljárásra van szükségünk. Ezeknek a feladatoknak a tipikus elnevezése a `KIJELENT`, illetve a `KÉRDEZ`. Mindkét feladat tartalmazhat **következtetés**t (**inference**), azaz új mondatok levezetését régiekből. A **logikai ágens**eknél (**logical agent**s), amelyekkel ebben a fejezetben foglalkozunk, a következtetéssel szemben az alapvető követelmény, hogy amikor valaki `KÉRDEZ` egy kérdést a tudásbázisról, akkor a válasznak következnie kell azokból a mondatokból, amit korábban a tudásbázishoz hozzáadtunk (pontosabban `KIJELENT`-ettünk). Később a fejezetben még pontosabban leírjuk az igen fontos „következik” fogalom jelentését. Most egyelőre ez jelentsen csak annyit, hogy a következtetési folyamat során nem csak úgy egyszerűen keletkeznek dolgok.

A 7.1. ábra egy tudásbázisú ágens programjának vázát mutatja. Mint ahogy az összes ágensünk, ez is észlel valamit bemenetként, és egy cselekvést ad vissza válaszként. Az ágens fenntart egy tudásbázist, *TB*-t, amely kezdetben bizonyos **háttértudás**t (**background knowledge**) tartalmazhat. Ha az ágensprogramot meghívják, az három dolgot tesz. Először `KIJELENT`-i a tudásbázisnak, hogy mit észlelt. Másodszor `KÉRDEZ`-i a tudásbázist, hogy milyen cselekvést kell végrehajtania. A lekérdezés megválaszolásának folyamatában egyre bővülő következtetést lehet végezni a világ pillanatnyi állapotáról, a lehetséges cselekvéssorozatok eredményéről és így tovább. Harmadszor az ágens rögzíti a kiválasztott cselekvést a `KIJELENT` felhasználásával, és végrehajtja a cselekvést. A második `KIJELENT` azért szükséges, hogy a tudásbázissal tudassuk, hogy a feltételezett *cselekvés* végrehajtása megtörtént.

7.1. ábra - Egy általános tudásbázisú ágens

```
function TB-ÁGENS(észlelés) returns egy cselekvés
  static: TB, egy tudásbázis
          t, egy számláló, kezdetben 0, mutatja az időt

  KIJELENT(TB, ÉSZLELÉS-MONDAT-KÉSZÍTÉS(észlelés, t))
  cselekvés ← KÉRDEZ(TB, CSELEKVÉS-KÉRDEZÉS-KÉSZÍTÉS(t))
  KIJELENT(TB, CSELEKVÉS-MONDAT-KÉSZÍTÉS)(cselekvés, t))
  t ← t + 1
  return cselekvés
```

A reprezentációs nyelv részletei három függvényben vannak elrejtve, amelyek az érzékelők és beavatkozók, az alapreprezentáció, valamint a következtető rendszer közötti kapcsolatot valósítják meg. Az `ÉSZLELÉS-MONDAT-KÉSZÍTÉS` eljárás egy olyan mondatot konstruál, amelyik megállapítja, hogy az ágens egy adott pillanatban észlelte az érzékelt dolgot. A `CSELEKVÉS-KÉRDEZÉS-KÉSZÍTÉS` az időt felhasználva bemeneti adatként, visszatér egy mondattal, amely alkalmas arra, hogy megkérdezzük, milyen cselekvés szükséges ebben a pillanatban. Végül a `CSELEKVÉS-MONDAT-KÉSZÍTÉS` egy olyan mondatot hoz létre, amely megállapítja, hogy a kiválasztott cselekvés végrehajtása megtörtént. A következtetési mechanizmus részletei a `KIJELENT` és a `KÉRDEZ` eljárások belsejében vannak elrejtve. A későbbi fejezetekben ezeket a részleteket is bemutatjuk.

A 7.1. ábra ágense egészen hasonlónak tűnik, mint a 2. fejezetben bemutatott belső állapottal rendelkező ágens. Azonban a `KIJELENT` és a `KÉRDEZ` eljárások definíciója miatt a tudásbázisú ágens nem egy tetszőleges, cselekvéseket meghatározó program. Meghatározható egy leírással a **tudásszint**en (**knowledge level**), ahol csak azt kell megfogalmaznunk, hogy mit tudjon és milyen céljai legyenek az ágensnek, hogy rögzíthessük a viselkedését. Például egy automata taxi számára a cél lehet egy utas elszállítása Marin megyébe, és a taxi ismerheti, hogy ez San Franciscónál van, és a Golden Gate híd az egyetlen, amelyen keresztül oda el lehet jutni. Ezután feltételezhetjük, hogy átmegy a Golden Gate hídon, *mivel tudja, hogy el fogja érni a célját*. Vegyük

Created by XMLmind XSL-FO Converter.

észre, hogy ez az elemzés független attól, hogy a taxi hogyan is működik az **implementációs szint**en (**implementation level**). Nem számít, hogy a földrajzi elhelyezkedéssel kapcsolatos tudása láncolt listákkal vagy pixeltérképekkel van megvalósítva, vagy hogy vajon a következtetést regiszterekben tárolt szimbólumok listáinak manipulálásával vagy neuronhálókban zajos jelek továbbküldésével végzi el.

Fontos

Említettük a fejezet bevezetőjében, hogy *egy tudásbázisú ágens megalkotható úgy, hogy* `KIJELENT`*-jük számára, hogy mit szükséges tudnia*. Az ágens inicializáló programja a tervező környezetről lévő tudását leíró mondatok egyenkénti átadásával építhető fel, még mielőtt az ágens elkezdené érzékelni a környezetét. Rendkívül leegyszerűsíti az ágens létrehozásának problémáját, ha olyan reprezentációs nyelvet tervezünk, amellyel egyszerű lesz ennek a tudásnak mondatokban való megfogalmazása. A rendszerépítésnek ezt a megközelítési módját **deklaratív** (**declarative**) módszernek nevezzük. Ezzel szemben a **procedurális** (**procedural**) megközelítés közvetlenül a program kódjában rögzíti a kívánt viselkedést, amivel minimalizálja az explicit reprezentáció és a következtetés szerepét, és így lényegesen hatékonyabb rendszer jöhet létre. Mind a kétféle ágensre látunk majd példát a 7.7. alfejezetben. Az 1970-es és 1980-as években a két megközelítés támogatói parázs vitákat folytattak. Mi mára felismertük, hogy egy sikeres ágens tervezésekor kombinálni kell a deklaratív és a procedurális elemeket.

Amellett hogy `KIJELENT`-hetjük az ágensnek, hogy mit kell tudnia, adhatunk is számára egy olyan mechanizmust, amely képessé teszi őt, hogy saját maga megtanulja ezt. Ez a mechanizmus, amit a 18. fejezetben tárgyalunk, általános, a környezetre vonatkozó tudást hoz létre érzetek sorozatának felhasználásával. Ezt a tudást bele lehet illeszteni az ágens tudásbázisába, és felhasználható a döntések meghozatalakor. Ily módon az ágens teljesen autonómmá válhat.

Ezek a képességek – a reprezentáció, a következtetés és a tanulás – mind a logika elméletének és technológiájának több évszázados fejlődésén alapulnak. Mielőtt azonban az elméletet és a technológiát elmagyaráznánk, létrehozunk egy egyszerű világot, amelyet majd felhasználunk ezek illusztrálásához.

2. A wumpus világ

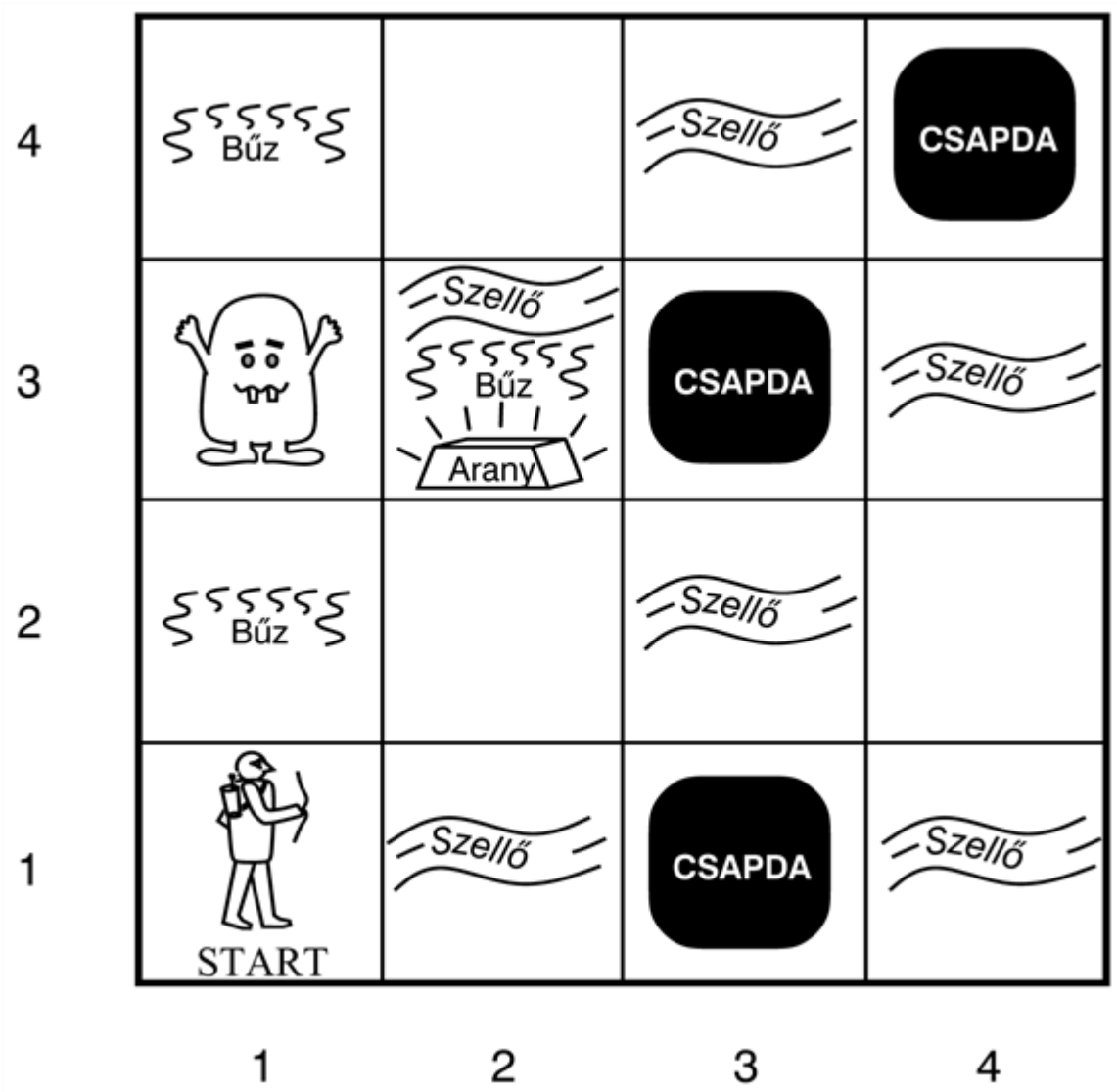
A **wumpus világ** (**wumpus world**) egy barlang, amely szobákból és az ezeket összekötő átjárókból áll. A wumpus egy szörnyeteg, aki mindenkit megesz, ha a szobájába lép, a barlangban lapul valahol. Az ágens le tudja lőni a wumpust, de csak egyetlen nyila van ehhez. Néhány szoba feneketlen csapdát tartalmaz, amely mindenkit csapdába ejt, aki belép a szobába (kivéve a wumpust, aki túl nagy ahhoz, hogy beleessen). A wumpus környezetében az egyetlen csábító lehetőség, hogy egy halom aranyat lehet találni. Habár a wumpus világ meglehetősen unalmas a modern számítógépes játékokhoz képest, azonban a játék kiváló tesztkörnyezet az intelligens ágensek számára. Michael Genesereth volt az első, aki javasolta ennek a környezetnek az alkalmazását.

Egy példa wumpus világ látható a 7.2. ábrán. A példakörnyezet pontos definícióját, ahogy a 2. fejezetben javasoltuk, a TKCSÉ-leírással adjuk meg:

- **Teljesítménymérték:** +1000 az arany felvétele, –1000 a csapdába esés vagy ha a wumpus felfal, –1 minden végrehajtott cselekvés, –10 a nyíl használata.

7.2. ábra - Egy tipikus wumpus világ. Az ágens a bal alsó sarokban van.

Created by XMLmind XSL-FO Converter.



- **Környezet:** Egy szobákból álló 4 × 4-es háló. Az ágens mindig az [1, 1]-gyel jelölt négyzetből indul, arccal jobbra nézve. Az arany és a wumpus elhelyezkedése véletlenszerűen, a kiinduló négyzeten kívüli négyzetek közül egyenletes eloszlás szerint van megválasztva. Ezen kívül még bármely, a kiinduló négyzeten kívüli négyzet 0,2 valószínűséggel lehet csapda.

- **Cselekvések:** Az ágens mozoghat előre, fordulhat balra 90°-kal, vagy fordulhat jobbra 90°-kal. Az ágens szörnyűséges halált hal, ha belép egy négyzetbe, ahol csapda van vagy egy élő wumpus található. (Biztonságos, habár meglehetősen rossz illatú egy olyan négyzetbe belépni, amelyben egy halott wumpus van.) Az előrelépésnek nincs hatása, ha egy fal van az ágens előtt. A *Megragad* cselekvést lehet arra használni, hogy az ágens felvegyen egy tárgyat, amely vele azonos szobában van. A *Lövés* cselekvést lehet használni egy nyílnak abban az irányban történő kilövésére, amerre az ágens éppen áll. A nyíl addig repül, amíg el nem találja (és egyben meg nem öli) a wumpust, vagy falnak nem ütközik. Az ágensnek csak egy nyila van, így csak egy *Lövés* cselekvésnek van hatása.

- **Érzékelők:** Az ágensnek öt érzékelője van, mindegyik egyetlen bitnyi információt ad:
  -
    - A wumpust tartalmazó négyzetben és a közvetlenül (nem átlósan) szomszédos négyzetekben az ágens bűzt érez.
  - A csapdával közvetlenül szomszédos négyzetekben az ágens szellőt érzékel.
  - A négyzetben, ahol az arany található, az ágens csillogást érzékel.
  - Ha az ágens falnak megy, akkor ütést érzékel.

Created by XMLmind XSL-FO Converter.

- Ha a wumpust megölték, akkor egy elkeseredett sikolyt hallat, amit a barlangban bárhol hallani lehet.

Az érzeteket az ágens egy öt szimbólumot tartalmazó lista formájában kapja meg; például ha bűz és szellő van egy négyzetben, de nincs ütés, csillogás vagy sikoly, akkor az ágens egy [*Bűz*, *Szellő*, *Nincs*, *Nincs*, *Nincs*] érzetet kap.

A 7.1. feladatban definiálni kell egy wumpus környezetet a 2. fejezetben megadott különböző dimenziók mentén. Az alapvető nehézség az ágens számára, hogy kezdetben semmit sem tud a környezet konfigurációjáról. Úgy tűnik, hogy logikai következtetésre van szüksége az ágensnek ahhoz, hogy felülkerekedhessen a tudatlanság okozta hátrányon. A wumpus világok legtöbb példányában az ágens számára lehetséges az arany biztonságos megszerzése. Néhány környezetben azonban az ágensnek választania kell, hogy hazamegy-e üres kézzel, vagy kockázatot vállal, ami vagy az aranyhoz vagy a halálhoz vezet. És a környezetek 21%-a teljesen tisztességtelen (mivel az arany egy csapdában van vagy csapdákkal körülvett mezőben).

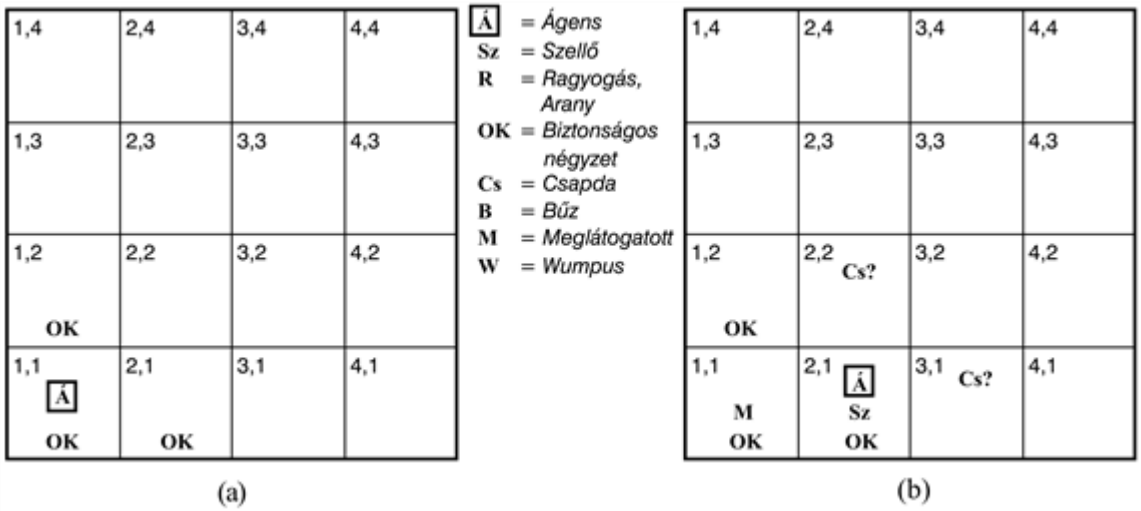
Nézzünk meg egy tudásbázisú wumpus ágenst, hogy hogyan fedezi fel a 7.2. ábrán látható környezetet. Az ágens kezdeti tudásbázisa a környezetet leíró, az előzőkben felsorolt szabályokat tartalmazza. Nevezetesen tudja, hogy az [1, 1]-ben tartózkodik és hogy az [1, 1] biztonságos hely. Látni fogjuk, hogy hogyan bővül az ágens tudása, amint új érzékelések érkeznek és cselekvések történnek.

Az első érzékelés a [*Nincs*, *Nincs*, *Nincs*, *Nincs*, *Nincs*], amiből az ágens arra tud következtetni, hogy a szomszédos négyzetek biztonságosak. A 7.3. (a) ábra mutatja az ágens tudásának állapotát ezen a ponton. Az ábrán a tudásbázis néhány mondatát soroljuk fel, betűket használva a megfelelő négyzetekben, mint az Sz (szellő) és az *OK* (biztonságos, nincs se csapda, se wumpus). A 7.2. ábra ezzel szemben magát a világot írja le.

Abból a tényből, hogy nem volt se bűz, se szellő az [1, 1]-ben, az ágens kikövetkeztetheti, hogy az [1, 2] és [2, 1] négyzetek veszélytelenek. Ennek jelzésére a megfelelő négyzetekbe *OK*-t írunk. Egy óvatos ágens csak olyan négyzetbe lép, amelyről tudja, hogy *OK*. Feltételezzük, hogy az ágens úgy dönt, hogy a [2, 1]-be megy, előállítva a 7.3. (b) ábrán látható helyzetet.

Az ágens detektálja a szellőt a [2, 1]-ben, tehát egy csapdának kell lennie valamelyik szomszédos négyzetben. A csapda nem lehet az [1, 1]-ben a játék szabályai szerint, így csapdának kell lennie a [2, 2]-ben vagy a [3, 1]-ben vagy mindkettőben. A *Cs*? jelölés egy lehetséges csapdát jelez a mezőkben a 7.3. (b) ábrán. Ezen a ponton csak egy olyan ismert négyzet van, ami *OK*, és amit még nem látogatott meg. Így a megfontolt ágens visszafordul, visszamegy az [1, 1]-be és az [1, 2]-be halad tovább.

7.3. ábra - Az ágens első lépése a wumpus világban. (a) A kezdeti helyzet a [*Nincs, Nincs, Nincs, Nincs, Nincs*] érzékelése után. (b) Az első lépés után, érzékelve a [*Nincs, Szellő, Nincs, Nincs, Nincs*]-et.



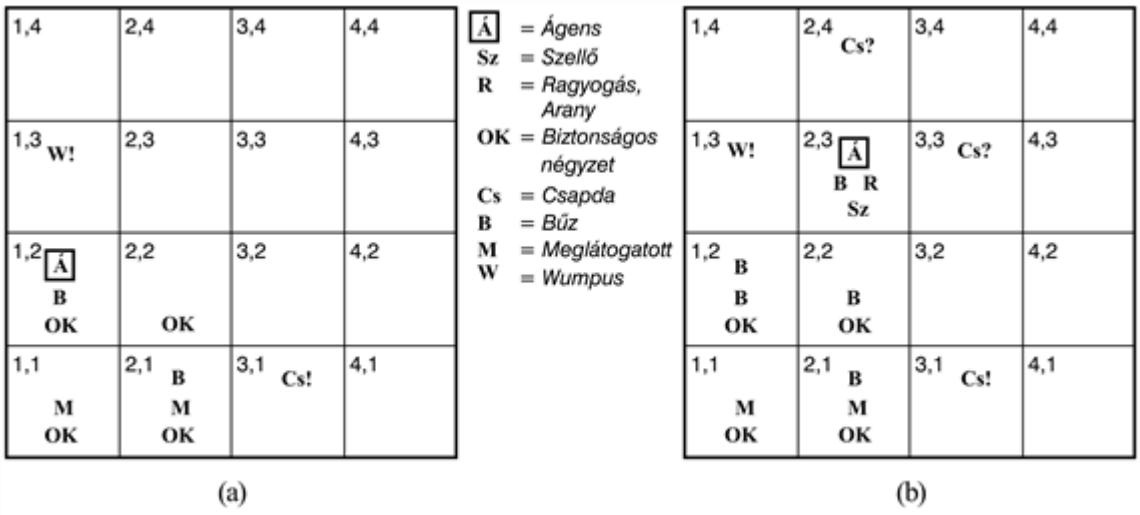
Az új érzet az [1, 2,]-ben a [*Bűz*, *Nincs*, *Nincs*, *Nincs*, *Nincs*], ami a 7.4. (a) ábrán látható helyzetet eredményezi. A bűz az [1, 2]-ben azt jelenti, hogy a wumpusnak a közelben kell lennie. De a wumpus a játék szabályai szerint nem lehet az [1, 1]-ben és nem lehet a [2, 2]-ben sem (mert akkor az ágens érezte volna a bűzt, amikor a [2, 1]-ben járt). Így az ágens kikövetkeztetheti, hogy a wumpus az [1, 3]-ban van. A *W*! jelölés ezt

Created by XMLmind XSL-FO Converter.

mutatja. Még érdekesebb, hogy a *Szellő* érzet hiánya az [1, 2]-ben azt jelenti, hogy nincs csapda a [2, 2]-ben. De mi már kikövetkeztettük, hogy vagy a [2, 2]-ben, vagy a [3,1]-ben van egy csapda, ami tehát azt jelenti, hogy a csapdának a [3, 1]-ben kell lennie. Ez egy viszonylag nehéz következtetés, mivel különböző időpontokban és különböző helyeken gyűjtött tudást használ fel, és egy érzet hiányára támaszkodva végez el egy fontos lépést. Ez a következtetés meghaladja a legtöbb állat képességeit, de tipikusan jellemzi azt a fajta következtetést, amit egy logikai ágens végez.

Az ágens így bebizonyította maga számára, hogy nincs se csapda, se wumpus a [2, 2]-ben, így a mozgás ebbe a négyzetbe *OK*. Nem mutatjuk be az ágens tudását a [2, 2]-ben, csak feltételezzük, hogy fordul és átlép a [2, 3]-ba, ami a 7.4. (b) ábrán látható. A [2, 3]-ban az ágens detektálja a csillogást, így megragadja az aranyat, és ezzel véget ér a játék.

7.4. ábra - Két későbbi helyzet az ágens előrehaladása során. (a) A harmadik lépést követően, miután [*Bűz, Nincs, Nincs, Nincs, Nincs*]-et érzékelt. (b) Az ötödik lépés és a [*Bűz, Szellő, Csillogás, Nincs, Nincs*] érzékelése után



Fontos

Bármely esetben, amikor az ágens következtetéseket von le a rendelkezésre álló információkból, a következmény garantáltan helyes lesz, ha a rendelkezésre álló információk helyesek. Ez alapvető jellegzetessége a logikai következtetéseknek. A fejezet hátralevő részében megmutatjuk, hogyan építhetünk olyan logikai ágenseket, amelyek képesek reprezentálni a szükséges információkat, és következtetéseket vonnak le, ahogy azt az eddigi fejezetekben leírtuk.

3. A logika

Ez az alfejezet áttekintést nyújt a logikai reprezentáció és következtetés alapvető fogalmairól. A logika bármely speciális formájára vonatkozó technikai részletek bemutatását a következő fejezetre halasztjuk. Ehelyett egyszerű példákat fogunk használni a wumpus világból vagy ismerős aritmetikai területekről. Azért választjuk ezt az igen rendhagyó megközelítést, mivel a logika fogalmai messze általánosabbak és szebbek, mint azt általában feltételezik.

A 7.1. fejezetben említettük, hogy a tudásbázis mondatokból áll. Ezeket a mondatokat a reprezentációs nyelv **szintaxis**a (**syntax**) szerint fejezzük ki, amely specifikálja az összes jól formált, nyelvtanilag helyes mondatot. A szintaxis fogalma elég tiszta a szokásos aritmetikai műveleteknél: „$x + y = 4$” egy jól formált mondat, míg az „$x2y + =$” nem az. A logikai nyelvek szintaxisait (és a matematikáét is egyébként) rendszerint cikkek és könyvek írása céljából tervezték. A szó szoros értelmében tucatjai léteznek a különböző szintaxisoknak, néhány tele görög betűkkel és egzotikus matematikai szimbólumokkal, néhány inkább vizuálisan követhető, nyilakat és buborékokat tartalmazó diagramokból áll. Minden esetben azonban, az ágens tudásbázisában a mondatok az ágensnek magának vagy az ágens egy részének valódi fizikai konfigurációi. A következtetés ezeknek a konfigurációknak a létrehozását vagy manipulálását fogja jelenteni.

Created by XMLmind XSL-FO Converter.

A logikának a nyelv **szemantikáját** (**semantics**) is definiálnia kell. Egyszerűen fogalmazva a szemantika a mondatok „jelentéséről” szól. A logikában a definíció pontosabb. A nyelv szemantikája definiálja minden mondat **igazságát** (**truth**) minden egyes **lehetséges világra** (**possible world**) vonatkozóan. Például egy szokásos aritmetikához választott szemantika meghatározza, hogy az „$x + y = 4$” mondat igaz abban a világban, ahol x értéke 2 és y értéke 2, de hamis abban a világban, ahol x értéke 1 és y értéke is 1.[1] A standard logikákban minden mondat vagy igaz, vagy hamis minden lehetséges világban, és nem lehet „valahol az igaz és hamis között”[2].

Amikor szükséges, hogy pontosak legyünk, a **modell** (**model**) kifejezést fogjuk használni a „lehetséges világ” helyén. (Szintén használni fogjuk az „m modellje α-nak” kifejezést, ami azt jelenti, hogy az α mondat igaz az m modellben.) Miután a lehetséges világokat úgy képzelhetjük el, mint (potenciálisan) valós környezeteket, amelyekben az ágens ott lehet vagy nem lehet ott, a modellek olyan matematikai absztrakciók, amelyek csak rögzítik az igazság vagy hamisság értékét minden releváns mondatnak. Például, tegyük fel, hogy x és y a férfiak és nők száma, akik egy kártyaasztal körül ülnek és bridzset játszanak, és az $x + y = 4$ mondat igaz, amikor négyen vannak összesen. Formálisan, a lehetséges modellek nem mások, mint minden lehetséges hozzárendelés az x és y változókhoz. Minden ilyen hozzárendelés bármely olyan aritmetikai mondat igazságát rögzíti, amely az x és y változókat tartalmazza.

Most, hogy van egy képünk az igazság fogalmáról, tudunk beszélni a logikai következtetésről. Ennek része a mondatok közötti logikai **vonzat** (**entailment**) reláció, annak kifejezése, hogy egy mondat logikusan következik egy másik mondatból. Matematikai jelöléssel ezt így írjuk:

$$\alpha \models \beta$$

aminek jelentése, hogy az α mondat maga után vonzza a β mondatot. A vonzat formális definíciója a következő: $\alpha \models \beta$ akkor és csakis akkor, ha minden modellben, amelyben α igaz, β szintén igaz. Közvetlenebbül azt mondhatjuk, hogy β igazságát „tartalmazza” α igazsága. A vonzat reláció ismerős az aritmetikából is, örömmel vehetjük észre, hogy az $x + y = 4$ mondat maga után vonzza a $4 = x + y$ mondatot. Nyilvánvaló, hogy bármely modellben, ahol $x + y = 4$, mint például az a modell, amelyben x és y is 2 értékű, a $4 = x + y$ is fenn áll. Hamarosan látni fogjuk, hogy a tudásbázist tekinthetjük egy kijelentésnek, és gyakran beszélhetünk arról, hogy egy tudásbázis maga után vonz egy mondatot.

Alkalmazhatunk hasonló elemzést az előző részben bemutatott wumpus világbeli következtetési példára is. Tekintsük a 7.3. (b) ábrán látható szituációt: az ágens nem észlelt semmit az [1, 1]-ben és szellőt észlelt a [2, 1]-ben. Ezek az érzetek, kombinálva az ágensnek a wumpus világ szabályaira vonatkozó tudásával (1. szakasz - A tudásbázisú ágens részben található TKCSÉ-leírás), alkotja a *TB*-t. Az ágenst (más dolgok mellett) az érdekli, hogy vajon a szomszédos [1, 2], [2, 2], [3, 1] négyzetek tartalmaznak-e csapdát. Bármelyik a három négyzet közül tartalmazhat vagy éppen nem tartalmaz csapdát, így a példa esetében $2^3 = 8$ lehetséges modell létezik. Ezt mutatja a 7.5. ábra.[3]

A *TB* hamis azokban a modellekben, amelyek ellentmondanak annak, amit az ágens tud. Például a *TB* hamis minden modellben, ahol az [1, 2] tartalmaz csapdát, mivel nincs szellő az [1, 1]-ben. Valójában csak három olyan modell van, amelyben a TB igaz, ezeket a 7.5. ábra a modellek egy részhalmazaként mutatja. Most tekintsünk két lehetséges következményt:

α_1 = „Nincs csapda az [1, 2]-ben”

α_2 = „Nincs csapda a [2, 2]-ben”

A 7.5. ábrán megjelöltük az α_1 és α_2 modelleket. Szemrevételezve megállapíthatjuk a következőt:

minden olyan modellben, ahol a *TB* igaz, ott α_1 is igaz.

Így $TB \not\models \alpha_1$, és nincs csapda az [1, 2]-ben. Azt is láthatjuk, hogy:

néhány modell, amelyben a *TB* igaz, az α_2 hamis.

[1] Az olvasó minden bizonnyal észrevette a hasonlóságot a mondatok igazságának fogalma és az 5. fejezetben bemutatott kényszerek kielégítésének fogalma között. Ez nem véletlen, a kényszernyelvek valójában logikák, és a kényszerproblémák megoldása a logikai következtetés egy formája.

[2] A **fuzzy logika** (**fuzzy logics**), amelyet a 14. fejezetben mutatunk be, megengedi az igazság fokának kezelését.

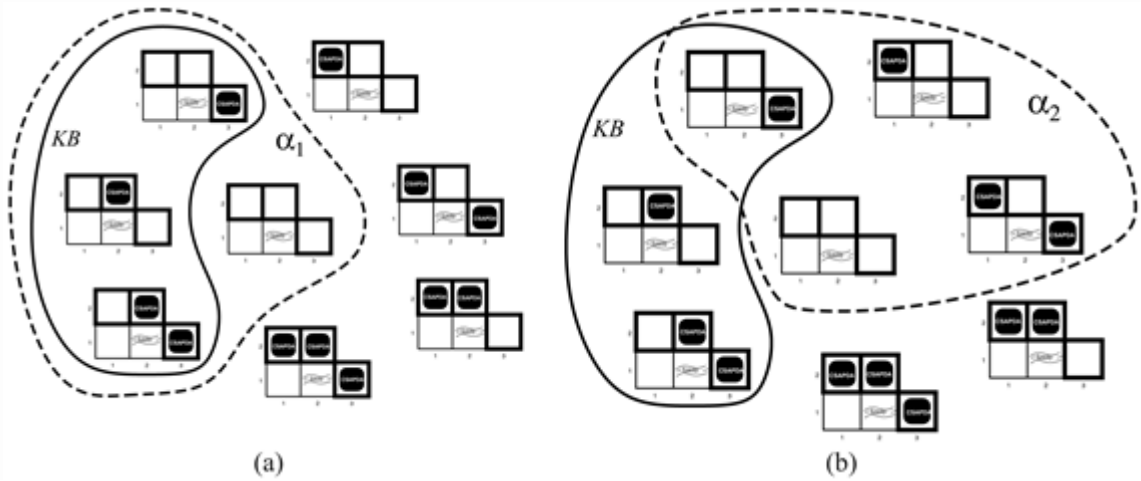
[3] Habár az ábra a modelleket, mint egy részleges wumpus világot mutatja, ezek valójában nem mások, mint az igaz és hamis értékek hozzárendelései a „csapda van az [1, 2]-ben” mondathoz. A modellek, matematikai értelemben, nem igénylik, hogy szörnyűséges illatú wumpusok legyenek benne.

Created by XMLmind XSL-FO Converter.

Így $TB \not\models \alpha_2$ és az ágens nem tudja kikövetkeztetni, hogy nincs csapda a [2, 2]-ben. (És azt sem tudja kikövetkeztetni, hogy van csapda a [2, 2]-ben.)[4]

Az előző példa nem csak illusztrálja a maga után vonzást, hanem megmutatja, hogy a vonzat definícióját fel lehet használni a következmények levezetésére, azaz, hogy **logikai következtetés**t (**logical inference**) végezzünk. A 7.5. ábrán illusztrált következtetési algoritmust **modellellenőrzés**nek (**model checking**) hívjuk, mivel számba vesz minden lehetséges modellt annak megvizsgálására, hogy α igaz-e minden modellben, amelyben a *TB* igaz.

7.5. ábra - Lehetséges modelljei a csapda jelenlétének az [1, 2], [2, 2] és [3, 1]-ben, ha adott a megfigyelés, hogy az [1, 1]-ben semmi és a [2, 1]-ben szellő érezhető. (a) A tudásbázis és α1(nincs csapda[1, 2]-ben) modelljei. (b) A tudásbázis és α2(nincs csapda[2, 2]-ben) modelljei.



A vonzat és a bizonyítás megértésében segíthet, ha a *TB* összes következményeit egy szénakazalnak, az α-t pedig egy gombostűnek képzeljük el. A vonzat olyan, mintha a gombostű benne volna a kazalban; a bizonyítás pedig nem más, mint megpróbálni megtalálni ezt a tűt. Ez a különbségtétel formálisan a következő megfogalmazásban ölt testet: ha egy *i* következtetési algoritmus képes levezetni α-t a *TB*-ből, akkor írhatjuk, hogy

$$TB \vdash_i \alpha$$

amely kimondva: „α az *i* által levezethető *TB*-ből” vagy „*i* levezeti α-t a *TB*-ből”.

Egy következtetési eljárást, amely csak vonzat mondatokat vezet le, **helyes**nek (**sound**) vagy **igazságtartó**nak (**truth-preserving**) nevezzük. A helyesség egy igencsak kívánatos tulajdonság. Egy nem helyes következtetési eljárás kitalál olyan dolgokat ahogy előrehalad, olyan tűk megtalálását jelenti be, amelyek nem is léteznek. Egyszerű belátni, hogy a modellellenőrzés, ha alkalmazható,[5] akkor helyes eljárás.

A **teljesség** (**completeness**) tulajdonság szintén kívánatos: egy következtetési eljárás teljes, ha képes levezetni minden vonzatmondatot. Valódi szénakazlak esetében, amelyek véges méretűek, nyilvánvalónak tűnik, hogy szisztematikus kutatással mindig eldönthető, hogy a tű a kazalban van-e. Sok tudásbázis esetében azonban a konzekvenciák szénakazlának mérete végtelen, és így a teljesség egy fontos kérdéssé válik.[6] Szerencsére léteznek teljes következtetési eljárások a logikához, amelyek megfelelően kifejezők ahhoz, hogy számos tudásbázist kezeljenek.

Fontos

[4] Az ágens ki tudja számolni, hogy mekkora a valószínűsége annak, hogy van csapda a [2, 2]-ben. A 13. fejezet fogja megmutatni azt, hogy hogyan.

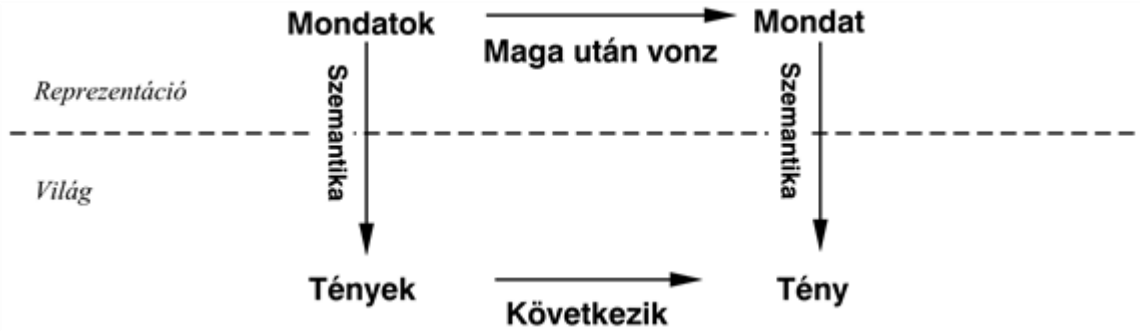
[5] A modellellenőrzés működik, ha a modellek tere véges, mint például egy rögzített méretű wumpus világ esetében. Az aritmetika esetében ezzel szemben a modellek tere végtelen, még akkor is, ha korlátozzuk magunkat az egész számokra, mivel végtelen számú *x*, *y* értékpár létezik az $x + y = 4$ egyenlethez.

[6] Hasonlítsuk össze a 3. fejezet végtelen keresési tereinek esetével, ahol látható, hogy a mélységi keresés nem teljes.

Created by XMLmind XSL-FO Converter.

Egy olyan következtetési folyamatot írtunk le, amelynek következményei bármely világban garantáltan igazak, ahol a premisszák is igazak. Nevezetesen, *ha a* TB *igaz a valódi világban, akkor bármely, a* TB*-ből helyes következtetési eljárással levezetett α mondat szintén igaz a valódi világban*. Így, míg a következtetési folyamat a „szintaxison" működik – belső fizikai konfigurációkon, mint például regiszterek bitjein vagy az agy elektromos jelzéseinek mintáin –, addig a folyamat megfelel valódi világ viszonyainak. Ennek megfelelően a valódi világ néhány aspektusa lesz az eset,[7] mivel a valódi világ bizonyos más aspektusai jelenleg képezik az esetet. Ezt a megfeleltetést a világ és a reprezentáció között mutatja a 7.6. ábra.

7.6. ábra - A mondatok az ágens fizikai konfigurációi, és a következtetés az a folyamat, amely új fizikai konfigurációkat hoz létre régiekből. A logikai következtetésnek biztosítania kell, hogy az új konfigurációk olyan aspektusait reprezentálják a világnak, amelyek ténylegesen is következnek azokból az aspektusokból, amelyeket a régi konfigurációk reprezentálnak.



Fontos

Az utolsó kérdés, amivel foglalkoznunk kell a logikai ágensek tárgyalásánál, a **megalapozottság** (**grounding**) kérdése, ami nem más, mint a kapcsolat, ha egyáltalán létezik ilyen, a logikai következtetési folyamat és a valódi környezet között, amelyben az ágens létezik. Nevezetesen *hogyan tudhatjuk meg, hogy a TB igaz-e a valódi világban?* (Ezután a *TB* már csak „szintaxis" az ágens fejében.) Ez egy filozófiai kérdés, amelyről sok-sok könyvet írtak (lásd 26. fejezet). Egy egyszerű válasz az, hogy az ágens érzékelői létesítik a kapcsolatot. Például a mi wumpus világbeli ágensünknek van egy szagló érzékelője. Az ágensprogram létrehoz egy megfelelő mondatot mindig, ha van illat. Így bármikor, ha ez a mondat a tudásbázisban van, ez igaz a valódi világban is. Ezáltal az érzetmondatok jelentését és igazságát az őket létrehozó érzékelő és mondatkonstruáló folyamatok határozzák meg. És mi a teendő az ágens tudásának egyéb részeivel, mint az a meggyőződése, hogy a wumpus rossz illatot terjeszt a szomszédos négyzetekben? Ez nem egy közvetlen reprezentációja egy egyedi érzetnek, hanem egy általános szabály, például érzékelési tapasztalatokból levezetve, de nem azonos magával a tapasztalatnak a kijelentésével. Az ilyen általános szabályokat a **tanulás**nak (**learning**) nevezett mondatkonstruáló folyamat hozza létre, ami a VI. résznek a tárgya. A tanulás nem tévedhetetlen. Lehet, hogy az eset az, hogy a wumpusok mindig rossz illatot árasztanak, *kivéve szökőévekben február 29-én,* amikor egyébként megfürödnek. Így lehet, hogy a *TB* nem igaz a valódi világban, de jó tanuló eljárásokkal van ok az optimizmusra.

7.4. Az ítéletkalkulus: egy nagyon egyszerű logika

Most egy nagyon egyszerű logikát, az **ítéletkalkulus**t (**propositional logic**) mutatjuk be.[8] Áttekintjük az ítéletkalkulus szintaxisát, majd a szemantikáját – annak a módját, ahogy a mondatok igazságát meghatározzuk. Azután megnézzük a maga után vonzást – a relációt egy adott mondat és azon mondat között, amelyik az előbbiből következik –, és megnézzük, hogy hogyan vezet ez egy egyszerű logikai következtetés algoritmushoz. Mindez természetesen a wumpus világban fog lejátszódni.

3.1. Szintaxis

[7] Mint azt Wittgenstein (1922) írta híres művében, a *Tractatus*ban: „A világ mindaz, aminek az esete fennáll."
[8] Az ítéletkalkulust **Boole-logiká**nak (**Boolean logic**) is nevezzük, a logikával foglalkozó George Boole (1815–1864) után.

Created by XMLmind XSL-FO Converter.

Az ítéletkalkulus **szintaxis**a meghatározza a lehetséges mondatokat. Az **atomi mondat**ok (**atomic sentence**s) – oszthatatlan szintaktikai elemek – egyetlen **ítéletszimbólum**ból (**proposition symbol**) állnak. Minden ilyen szimbólum egy kijelentés, ami igaz vagy hamis lehet. Nagybetűs neveket fogunk használni a szimbólumok jelölésére: *P*, *Q*, *R* és így tovább. A nevek tetszőlegesek, de gyakran úgy választjuk őket, hogy a nevek bizonyos jelentéssel is rendelkezzenek. Például használhatjuk a $W_{1,3}$-t annak az állításnak a kifejezésére, hogy a wumpus az [1, 3]-ban van. (Ne felejtsük el, hogy a $W_{1,3}$ *atomi,* így a *W*, az 1 és a 3 nem jelentéssel bíró részei a szimbólumnak.) Létezik két ítéletszimbólum, amelyeknek rögzített az értelmezése: az *Igaz* egy mindig igaz állítás, és a *Hamis* egy mindig hamis állítás.

Összetett mondatok (**complex sentence**s) létrehozhatók egyszerűbb mondatokból logikai **összekötőjel**ek (**logical connective**s) felhasználásával. Öt elterjedten használt összekötőjel van:

¬ (nem). Egy mondatot, mint a $\neg W_{1,3}$-t, **negáció**nak (**negation**) nevezünk. Egy **literál** (**literal**) vagy egy atomi mondat (egy pozitív literál), vagy egy negált atomi mondat (egy negatív literál).

∧ (és). Egy mondatot, amelynek fő kötőszava a ∧, mint például a $W_{1,3} \wedge C_{1,3}$ **konjunkció**-nak (**conjunction**) nevezünk; ennek részei a **konjunkt**ok (**conjunct**s). (Az ∧ jel hasonlít egy A-ra, az angol *And* (és) szóból.)

∨ (vagy). Egy mondat, amely használja a ∨ összekötőjelet, mint a $(W_{1,3} \wedge C_{1,3}) \vee W_{2,2}$ egy **diszjunkció** (**disjunction**), méghozzá a $(W_{1,3} \wedge C_{1,3})$ és a $W_{2,2}$ **diszjunkt**oknak (**disjunct**s) a diszjunkciója. (Történelmileg a ∨ jel a latin *vel* kifejezésből származik, ami „vagy”-ot jelent. A legtöbb ember számára könnyebb megjegyezni úgy, mint egy fejjel lefelé álló *és* jelet, vagy a magyar olvasók számára, mint a *vagy* szó első betűjét.)

⇒(implikáció). Egy mondatot, mint amilyen a $(W_{1,3} \wedge C_{1,3}) \Rightarrow \neg W_{2,2}$ **implikáció**nak (**implication**) (vagy feltételes mondatnak) nevezünk. Ennek **premisszá**ja (**premise**) vagy **előzmény**e (**antecedent**) a $(W_{1,3} \wedge C_{1,3})$, **konklúzió**ja (**conclusion**) vagy **következmény**e (**consequent**) pedig a $\neg W_{2,2}$ implikáció **szabály**ként (**rule**) vagy **ha–akkor** (**if–then**) állításként is ismert. Az implikációt más könyvekben időnként a ⊃ vagy a → szimbólumokkal jelölik.

⇔(akkor és csakis akkor). A $W_{1,3} \Leftrightarrow \neg W_{2,2}$ mondat egy **ekvivalencia** (**biconditional**).

A 7.7. ábra mutatja az ítéletkalkulus formális nyelvtanát; ha nem ismeri a BNF jelölésrendszert, akkor nézze meg az 1. szakasz - B1. Nyelvek definiálása Backus–Naur-Formában (BNF) részt.

7.7. ábra - Ítéletkalkulus-beli mondatok BNF (Backus–Naur-forma) nyelvtana

Mondat → *AtomiMondat* | *KomplexMondat*

AtomiMondat → **Igaz** | **Hamis** | *Szimbólum*
Szimbólum → **P** | **Q** | **R** | …
KomplexMondat → ¬*Mondat*
| (*Mondat* ∧ *Mondat*)
| (*Mondat* ∨ *Mondat*)
| (*Mondat* ⇒ *Mondat*)
| (*Mondat* ⇔ *Mondat*)

Vegyük észre, hogy a nyelvtan nagyon szigorú a zárójelezésnél: minden mondatot, amelyet bináris összekötőjellel hozunk létre, zárójelek közé kell tenni. Ez biztosítja, hogy a szintaxis teljesen egyértelmű. Ez azt is jelenti például, hogy $((A \wedge B) \Rightarrow C)$-t kell írjunk $A \wedge B \Rightarrow C$ helyett. Az olvashatóság javítása céljából gyakran elhagyjuk a zárójeleket, megbízva ehelyett az összekötőjeleknek egy precedencia-sorrendjében. Ez hasonló az aritmetikában alkalmazott precedenciához – például az $ab + c$-t $((ab) + c)$-nek olvassuk, és nem $a(b + c)$-nek, mert a szorzásnak magasabb a precedenciája, mint az összeadásnak. Az ítéletkalkulus precedencia-sorrendje (a legmagasabbtól a legalacsonyabb felé: ¬, ∧, ∨, ⇒ és ⇔ Így a mondat:

$\neg P \vee Q \wedge R \Rightarrow S$

ekvivalens a következő mondattal:

$((\neg P) \vee (Q \wedge R)) \Rightarrow S$

Created by XMLmind XSL-FO Converter.

A precedencia nem oldja fel a többértelműséget az olyan mondatoknál, mint az $A \wedge B \wedge C$, amelyet olvashatunk $((A \wedge B) \wedge C)$-ként vagy $(A \wedge (B \wedge C))$-nek. Mivel a mondatnak ez a két olvasása ugyanazt jelenti a következő részben definiálandó szemantika szerint, az olyan mondatok, mint az $A \wedge B \wedge C$ megengedettek. Szintén megengedjük az $A \vee B \vee C$ és az $A \Leftrightarrow B \Leftrightarrow C$ mondatokat. Olyan mondatok, mint az $A \Rightarrow B \Rightarrow C$ nem megengedettek, mert a két olvasásnak különböző jelentései vannak, ebben az esetben ragaszkodunk a zárójelhez. Végül, néha használni fogunk szögletes zárójelet az egyszerű zárójel helyett, ami a mondatot áttekinthetőbbé teszi majd.

3.2. Szemantika

Most, hogy specifikáltuk az ítéletkalkulus szintaxisát, definiáljuk a szemantikáját is. A szemantika definiálja a szabályokat, amivel meghatározható a mondat igazsága egy bizonyos modellben. Az ítéletkalkulusban a modell egyszerűen az igazságértéket – *igaz* vagy *hamis* – rögzíti minden ítéletszimbólumra. Például ha a tudásbázis mondatai a $C_{1,2}$, $C_{2,2}$ és $C_{3,1}$ ítéletszimbólumokat használják fel, akkor egy lehetséges modell:

$m_1 = \{C_{1,2} = hamis, C_{2,2} = hamis, C_{3,1} = igaz\}$

Három ítéletszimbólum esetén $2^3 = 8$ lehetséges modell van – pontosan azok, amelyek a 7.5. ábrán vannak feltüntetve. Vegyük észre azonban, hogy miután elköteleztük magunkat egy szintaxis mellett, a modellek tisztán matematikai objektumokká váltak, amelyeknek nem feltétlenül van kapcsolatuk a wumpus világgal. A $C_{1,2}$ csak egy szimbólum, jelentheti azt, hogy „csapda van az [1, 2]-ben” vagy azt, hogy „Párizsban vagyok ma és holnap”.

Az ítéletkalkulus szemantikájának meg kell határoznia, hogyan számítsuk ki bármely mondat igazságértékét egy adott modellben. Ez rekurzívan történik. Minden mondat atomi mondatokból és az ötféle összekötőjelből lett létrehozva, így meg kell határoznunk, hogy hogyan számítsuk ki az atomi mondatok igazságát, és meg kell határoznunk azt is, hogy hogyan számítsuk ki az igazságát az egyes összekötőjelek felhasználásával formált összetett mondatnak. Az atomi mondatok esete egyszerű:

- *Igaz* minden modellben igaz, és *Hamis* ha minden modellben hamis.
- Minden más ítéletszimbólumnak az igazságértékét közvetlenül a modellben kell meghatározni. Például a korábban megadott m_1 modellben a $C_{1,2}$ hamis.

Összetett mondatokra olyan szabályaink vannak, mint

- Bármely *s* mondatra és bármely *m* modellre, a ¬*s* mondat az *m*-ben akkor és csakis akkor igaz, ha *s* hamis *m*-ben.

Az ilyen szabályok visszavezetik az összetett mondatok igazságának eldöntését egyszerűbb mondatokra. Minden összekötőjelre vonatkozó szabály összefoglalható egy **igazságtábl**ában (**truth table**), amely meghatározza egy összetett mondat igazságértékét a mondat komponenseinek minden lehetséges igazságérték hozzárendeléseihez. Az öt logikai összekötőjel igazságtábláját mutatja a 7.8. ábra. Ilyen táblák felhasználásával bármely *s* mondat igazságértéke egy *m* modellre vonatkozóan kiszámolható rekurzív kiértékelések egyszerű folyamatával. Például a $\neg C_{1,2} \wedge (C_{2,2} \vee C_{3,1})$ mondatot kiértékelve m_1-ben $igaz \wedge (hamis \vee igaz) = igaz \wedge igaz = igaz$ értéket kapunk. A 7.3. feladat azt kéri, hogy írjon egy `IK-IGAZ?` (*s*,*m*) algoritmust, amely kiszámítja az *s* ítéletkalkulus mondatnak az igazságértékét egy *m* modellben.

Korábban azt mondtuk, hogy a tudásbázist mondatok halmaza alkotja. Most láthatjuk, hogy egy logikai tudásbázis ilyen mondatok konjunkciója. Tehát ha egy üres *TB*-vel kezdünk, és végrehajtjuk a `KIJELENT`(*TB*, S_1), ..., `KIJELENT` (*TB*, S_n) műveleteket, akkor a $TB = S_1 \wedge \ldots \wedge S_n$ áll elő. Ez azt jelenti, hogy tudásbázisokat és mondatokat egymással felcserélhetően használhatunk.

Az „és”, „vagy” és „nem” igazságtáblái eltérnek attól, amit a természetes nyelvi jelentésük alapján gondolnánk. A lehetséges félreértés legszembetűnőbb pontja, hogy a $P \vee Q$ kifejezés igaz akkor is, ha mind *P*, mind *Q* is igaz. Létezik másik összekötőjel is, a „kizáró vagy”-nak nevezett jel (röviden *xor*), amely hamisat ad, ha mindkét diszjunkt igaz.[9] Nincs általános egyetértés az „exkluzív vagy” szimbólumát illetően; két jelölés is ismert: $\dot{\vee}$ és a $\oplus$

[9] A latinnak létezik külön szava az „exkluzív vagy” kifejezésére, ez az *aut*.

Created by XMLmind XSL-FO Converter.

7.8. ábra - Az öt logikai összekötőjel igazságtáblája. Amikor a táblát használjuk például a *P* ∨ *Q* értékének számítására, ha *P* igaz és *Q* hamis, akkor először megkeressük azt a sort, amelyben *P igaz* és *Q hamis* (a harmadik sor). Ezután a sorban megkeressük a *P* ∨ *Q* alatti oszlopot, hogy megtaláljuk az eredményt: *igaz*. Tekinthetjük a táblázat úgy is, hogy minden sor egy modell, és az egyes oszlopbeli elemek az adott sorban azt mondják meg, hogy a megfelelő mondat igaz-e az adott modellben.

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
Hamis	*Hamis*	*Igaz*	*Hamis*	*Hamis*	*Igaz*	*Igaz*
Hamis	*Igaz*	*Igaz*	*Hamis*	*Igaz*	*Igaz*	*Hamis*
Igaz	*Hamis*	*Hamis*	*Hamis*	*Igaz*	*Hamis*	*Hamis*
Igaz	*Igaz*	*Hamis*	*Igaz*	*Igaz*	*Igaz*	*Igaz*

Az implikáció (⇒) igazságtáblája rejtélyesnek tűnhet első látásra, mivel nem teljesen illeszkedik ösztönös megértésünkhöz, hogy „*P* implikálja *Q*-t” vagy „ha *P*, akkor *Q*”. Az ítéletkalkulus nem kíván semmilyen ok-okozati relációt vagy relevanciát *P* és *Q* között. A mondat: „az a tény, hogy 5 páratlan implikálja, hogy Tokió Japán fővárosa” az ítéletkalkulusnak egy igaz mondata (a normális interpretáció szerint), még akkor is, ha ez határozottan furcsa mondatnak tűnik. Egy másik esete a félreértéseknek, hogy bármely implikáció igaz, ha az előzménye hamis. Például az „az az állítás, hogy 5 páros szám, implikálja, hogy Samu okos” mondat igaz, függetlenül attól, hogy Samu okos-e. Ez bizarrnak tűnik, de elfogadható, ha a „$P \Rightarrow Q$”-t úgy értelmezzük, hogy „ha *P* igaz, akkor azt állítom, hogy *Q* is igaz. Egyébként nem állítok semmit”. Az egyetlen eset, amikor ez a mondat *hamis*, ha *P igaz*, de *Q hamis*.

A $P \Leftrightarrow Q$ ekvivalencia igazságtáblája azt mutatja, hogy akkor igaz, ha mind $P \Rightarrow Q$ és $Q \Rightarrow P$ igaz. Ezt gyakran úgy írjuk le, hogy „*P* akkor és csakis akkor, ha *Q*” vagy matematikában szokták jelölni „*P* aa *Q*”-nak. A wumpus világ szabályait legjobban az ⇔ használatával tudjuk felírni. Például, egy négyzet szellős, ha a szomszédos, ha a szomszédos négyzetben csapda van, és egy négyzet csakis akkor szellős négyzetben csapda van. Így ekvivalenciákra van szükségünk, mint a

$$S_{1,1} \Leftrightarrow (C_{1,2} \vee C_{2,1})$$

ahol $S_{1,1}$ jelenti, hogy szellő van az [1, 1]-ben. Vegyük észre, hogy az egyirányú implikáció

$$S_{1,1} \Rightarrow (C_{1,2} \vee C_{2,1})$$

igaz a wumpus világban, de nem teljes. Nem szabályozza azokat a modelleket, amelyekben $S_{1,1}$ hamis és $C_{1,2}$ igaz, amivel megszegnénk a wumpus világ szabályait. Egy másik mód ennek érzékeltetésére, hogy az implikáció igényli a csapda jelenlétét, ha szellő van, miközben az ekvivalencia szintén megkívánja a csapda hiányát, ha nincs szellő.

3.3. Egy egyszerű tudásbázis

Most, hogy definiáltuk az ítéletkalkulus szemantikáját, létre tudunk hozni egy tudásbázist a wumpus világ számára. Az egyszerűség kedvéért csak a csapdákkal fogunk törődni, a wumpust magát feladatként az olvasóra hagyjuk. A tudás, amelyet most leírunk, elégséges ahhoz, hogy elvégezzük a 7.3. alfejezetben nem formálisan már elvégzett következtetést.

Először meg kell választanunk a szótárunkat az ítéletszimbólumaink megnevezéséhez. Minden *i*, *j*-re:

- Legyen C_{ij} igaz, ha csapda van [*i*, *j*]-ben.
- Legyen S_{ij} igaz, ha szellő van [*i*, *j*]-ben.

A tudásbázis tartalmazza a következő mondatokat (mindegyiket felcímkézzük a kényelem kedvéért):

- Nincs csapda az [1, 1]-ben:

Sz_1: $\neg C_{11}$

Created by XMLmind XSL-FO Converter.

- Egy négyzet akkor és csakis akkor szellős, ha csapda van a szomszédos négyzetben. Ezt minden négyzetre vonatkozóan ki kell jelenteni, mi most csak a releváns négyzeteket tekintjük:

Sz_2: $S_{1,1} \Leftrightarrow (C_{1,2} \vee C_{2,1})$

Sz_3: $S_{2,1} \Leftrightarrow (C_{1,1} \vee C_{2,2} \vee C_{3,1})$

- Az eddigi mondatok minden wumpus világban igazak. Most hozzáadjuk a szellő érzetet az első két meglátogatott négyzetre abban a specifikus világban, ahol az ágens jelenleg tartózkodik, ami elvezet a 7.3. (b) ábrán látott szituációhoz.

Sz_4: $\neg S_{1,1}$
Sz_5: $S_{2,1}$

A tudásbázis most Sz_1-től Sz_5-ig tartalmaz mondatokat. Tekinthetjük ezt úgy is, mint egyetlen mondatot – az $Sz_1 \wedge Sz_2 \wedge Sz_3 \wedge Sz_4 \wedge Sz_5$ konjunkciót –, mivel ez azt is kijelenti egyben, hogy minden egyes mondat is igaz.

3.4. Következtetés

Emlékezzünk, hogy a logikai következtetés célja, hogy eldöntsük, hogy $TB \models \alpha$ bizonyos α mondatokra. Például, hogy a tudásbázis maga után vonzza-e a $C_{2,2}$ állítást. Az első algoritmusunk a következtetésre a vonzat definíciójának közvetlen megvalósítása lesz: vegyük sorba a modelleket, és ellenőrizzük, hogy α igaz-e minden modellben, amelyben a *TB* igaz. Az ítéletlogikában a modellek az *igaz* és *hamis* értékek hozzárendelései minden egyes ítéletszimbólumhoz. Visszatérve a mi wumpus világbeli példánkhoz, a releváns ítéletszimbólumok az $S_{1,1}$, $S_{1,2}$, $C_{1,1}$, $C_{1,2}$, $C_{2,1}$, $C_{2,2}$, $C_{3,1}$. A 7 szimbólum $2^7 = 128$ lehetséges modellt jelent, háromban ezek közül a *TB* igaz (7.9. ábra). Ebben a három modellben $\neg C_{1,2}$ igaz, így nincsen csapda az [1, 2]-ben. Viszont a $C_{2,2}$ a három modellből kettőben igaz és egyben hamis, így még nem tudjuk megmondani, hogy van-e csapda [2, 2]-ben.

A 7.9. ábra precízebb formában megismétli a 7.5. ábrán illusztrált következtetést. A 7.10. ábra egy általános algoritmust mutat a maga után vonzás eldöntésére az ítéletkalkulusban. Mint a `VISSZALÉPÉSES-KERESÉS` algoritmus a 4.3. szakasz - Mélységi keresés részben, az `IT-VONZAT?` egy rekurzív felsorolást végez a változó hozzárendelések véges terén. Az algoritmus **helyes**, mivel közvetlenül a vonzat definícióját valósítja meg, és teljes, mivel bármely *TB*-on és α mondaton működik, és mindig sikeresen véget ér, hiszen véges számú modellt kell megvizsgálni.

7.9. ábra - A tudásbázis alapján épített igazságtábla látható az ábrán. A *TB* igaz, ha Sz_1-től és Sz_5-ig igaz, amely a 128 sorból csak 3-ban fordul elő. Mind a 3 sorban $C_{1,2}$ hamis, tehát nincsen csapda az [1, 2]-ben. Viszont lehet, hogy van csapda [2, 2]-ben (bár lehet, hogy nincs).

$S_{1,1}$	$S_{1,2}$	$C_{1,1}$	$C_{1,2}$	$C_{2,1}$	$C_{2,2}$	$C_{3,1}$	Sz_1	Sz_2	Sz_3	Sz_4	Sz_5	*TB*
hamis	*hamis*	*hamis*	*hamis*	*hamis*	*hamis*	*hamis*	*igaz*	*igaz*	*igaz*	*igaz*	*hamis*	*hamis*
hamis	*hamis*	*hamis*	*hamis*	*hamis*	*hamis*	*igaz*	*igaz*	*igaz*	*hamis*	*igaz*	*hamis*	*hamis*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
hamis	*igaz*	*hamis*	*hamis*	*hamis*	*hamis*	*hamis*	*igaz*	*igaz*	*hamis*	*igaz*	*igaz*	*hamis*
hamis	*igaz*	*hamis*	*hamis*	*igaz*	*hamis*	*igaz*	*igaz*	*igaz*	*igaz*	*igaz*	*igaz*	*igaz*
hamis	*igaz*	*hamis*	*hamis*	*hamis*	*igaz*	*hamis*	*igaz*	*igaz*	*igaz*	*igaz*	*igaz*	*igaz*
hamis	*igaz*	*hamis*	*hamis*	*hamis*	*igaz*	*igaz*	*igaz*	*igaz*	*igaz*	*igaz*	*igaz*	*igaz*
hamis	*igaz*	*hamis*	*hamis*	*igaz*	*hamis*	*hamis*	*igaz*	*hamis*	*hamis*	*igaz*	*igaz*	*hamis*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
igaz	*igaz*	*igaz*	*igaz*	*igaz*	*igaz*	*igaz*	*hamis*	*igaz*	*igaz*	*hamis*	*igaz*	*hamis*

7.10. ábra - Egy igazságtábla felsoroló algoritmus ítéletkalkulus állítás vonzatának eldöntésére. Az IT az igazságtáblát jelöli. A `IK-IGAZ?` igazat ad vissza, ha a mondatot

Created by XMLmind XSL-FO Converter.

tartalmazza a modell. A *modell* változó reprezentál egy részleges modellt, egy hozzárendelést a változók egy részéhez. A KIEGÉSZÍT (*P, igaz, modell*) függvény egy új részleges modellt ad vissza, amelyben *P* értéke igaz.

```
function IT-VONZAT?(TB, α) returns igaz vagy hamis
  input: TB, a tudásbázis, egy ítéletkalkulus mondat
         α, a lekérdezés, egy ítéletkalkulus mondat

  szimbólumok ← TB-ban és α-ban szereplő szimbólumok listája
  return IT-ELLENŐRIZ-MIND(TB, α, szimbólumok, [])

function IT-ELLENŐRIZ-MIND(TB, α, szimbólumok, modell) returns igaz vagy hamis
    if ÜRES?(szimbólumok) then
      if IK-IGAZ?(TB, modell) then return IK-IGAZ?(α, modell)
      else return igaz
    else do
      P ← ELSŐ(szimbólumok); maradék ← MARADÉK(szimbólumok)
      return IT-ELLENŐRIZ-MIND(TB, α, maradék, KIEGÉSZÍT(P, igaz, modell) and
             IT-ELLENŐRIZ-MIND(TB, α, maradék, KIEGÉSZÍT(P, igaz, modell))
```

Fontos

Természetesen a „véges számosság” nem mindig jelenti a „néhányat”. Ha a *TB* és az α mondat összesen *n* szimbólumot tartalmaz, akkor 2^n modell létezik. Így az algoritmus időigénye $O(2^n)$. (A tárigénye csak $O(n)$, mivel a felsorolás mélységi jellegű.) A fejezet későbbi részében fogunk látni olyan algoritmusokat, amelyek sokkal hatékonyabbak a gyakorlatban. Sajnos *minden ismert, az ítéletlogikára vonatkozó következtetési algoritmusnak a legrosszabb esetre vonatkozó komplexitása exponenciálisan függ a bemenetek számától*. Nem remélhetjük, hogy ennél hatékonyabbak is lehetnénk, mivel az ítéletlogikában a maga után vonzás co-NP-teljes (lásd A) függelék).

3.5. Ekvivalencia, érvényesség és kielégíthetőség

Mielőtt belemerülnénk a logikai következtetés részleteinek tárgyalásába, szükségünk van néhány további, a vonzattal kapcsolatos fogalomra. Mint a vonzat, ezek a fogalmak is a logika minden formájára alkalmazhatók, de legjobban egy konkrét logikán – mint amilyen az ítéletkalkulus – lehet illusztrálni őket.

Az első fogalom a **logikai ekvivalencia** (**logical equivalence**): két mondat, az α és a β mondatok logikailag ekvivalensek, ha ezek a mondatok a modellek ugyanazon halmazán igazak. Ezt úgy jelöljük, hogy $\alpha \Leftrightarrow \beta$. Például (igazságtáblákat használva) könnyen megmutathatjuk, hogy $P \wedge Q$ és $Q \wedge P$ logikailag ekvivalensek; további ekvivalenciák láthatók a 7.11. ábrán. Az ekvivalenciák nagyon hasonló szerepet játszanak a logikában, mint az aritmetikai identitások a közönséges matematikában. Az ekvivalencia alternatív definíciója a következő: bármely **két** α, β mondatra,

$\alpha \equiv \beta$ akkor és csakis akkor, ha $\alpha \models \beta$ és $\beta \models \alpha$

(Emlékeztetőül, a $\models$ a maga után vonzást jelöli.)

A második fogalom, amelyre szükségünk lesz az **érvényesség** (**validity**). Egy mondat érvényes, ha igaz *minden* modellben. Például a $P \vee \neg P$ mondat érvényes. Az érvényes mondatokat **tautológiá**knak (**tautologies**) is szokták nevezni, ezek szükségszerűen igazak és így feleslegesek. Mivel az *Igaz* mondat igaz minden modellben, minden érvényes mondat logikailag ekvivalens az *Igaz* mondattal.

7.11. ábra - Standard logikai ekvivalenciák. Az *α, β, γ* szimbólumok tetszőleges ítéletkalkulus mondatokat jelölnek.

Created by XMLmind XSL-FO Converter.

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$	$\wedge$ kommutativitás
$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$	$\vee$ kommutativitás
$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$	$\wedge$ asszociativitás
$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$	$\vee$ asszociativitás
$\neg(\neg \alpha) \equiv \alpha$	kettős negáció kiküszöbölés
$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$	kontrapozíció
$(\alpha \Rightarrow \beta) \equiv (\neg\beta \vee \alpha)$	implikáció kiküszöbölés
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	ekvivalencia kiküszöbölés
$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$	De Morgan
$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$	De Morgan
$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	$\wedge$ disztributivitás
$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	$\vee$ disztributivitás

Fontos

Mire jók akkor az érvényes mondatok? A vonzatokra vonatkozó definíciónk alapján levezethetjük a **dedukcióelmélet**et (**deduction theorem**), amelyet már az ókori görögök is ismertek:

Bármely α és β mondatra $\alpha \models \beta$ akkor és csakis akkor, ha az $(\alpha \Rightarrow \beta)$ mondat érvényes.

(A 7.4. feladatban ennek a bizonyítását kérjük.) A 7.10. ábrán látható következtetési algoritmust úgy tekinthetjük, mint a $(TB \Rightarrow \alpha)$ érvényességének ellenőrzését. Másik oldalról nézve viszont minden érvényes implikáció leír egy legitim következtetést.

Az utolsó fogalom, amelyre szükségünk lesz a **kielégíthetőség** (**satisfiability**). Egy mondat kielégíthető, ha igaz *néhány* modellben. Például a korábban bemutatott tudásbázis, az ($Sz_1 \wedge Sz_2 \wedge Sz_3 \wedge Sz_4 \wedge Sz_5$), kielégíthető, mert van három olyan modell, amelyben igaz, ahogy ezt a 7.9. ábrán megmutattuk. Ha egy α mondat igaz az *m* modellben, akkor azt mondjuk, hogy *m* **kielégíti** (**satisfies**) α-t, vagy *m* egy **modell**je α-nak. A kielégíthetőség ellenőrizhető úgy, hogy a lehetséges modelleket felsoroljuk mindaddig, amíg nem találunk egyet, amely kielégíti a mondatot. A mondatok kielégíthetőségének meghatározása az ítéletlogikában az első olyan probléma volt, amelyről bebizonyították, hogy NP-teljes.

Számos probléma a számítástechnikában valójában kielégíthetőségi probléma. Például a kényszerkielégítési problémák az 5. fejezetben alapvetően arra kérdeznek rá, hogy a kényszerek kielégíthetők-e néhány hozzárendeléssel. Megfelelő transzformációk után a keresési problémák szintén megoldhatók a kielégíthetőség ellenőrzésével. Az érvényesség és a kielégíthetőség természetesen kapcsolatban vannak: α érvényes akkor és csakis akkor, ha $\neg\alpha$ nem kielégíthető; és fordítva, α akkor és csakis akkor kielégíthető, ha $\neg\alpha$ nem érvényes. A következő hasznos eredményt is ismerjük:

$\alpha \models \beta$ akkor és csakis akkor, ha az $(\alpha \wedge \neg \beta)$ nem kielégíthető

Fontos

A β mondat bizonyítása α alapján, az $(\alpha \wedge \neg\beta)$ kielégíthetetlenségének ellenőrzésével, pontosan megfelel a szokásos matematikai bizonyítási technikának a *redukcio ad absurdumnak* (szó szerint „redukció egy abszurd dologra"). Szokták ezt **megcáfolás** (**refutation**) általi bizonyításnak is nevezni vagy bizonyítás **ellentmondás** (**contradiction**) által. Feltételezzük, hogy a β mondat hamis, és megmutatjuk, hogy ez a feltételezés ellentmondásra vezet az ismert α axiómákkal. Ez az ellentmondás pontosan azt jelenti, mint amikor azt mondjuk, hogy az $(\alpha \wedge \neg\beta)$ mondat kielégíthetetlen.

4. Az ítéletkalkulus következtetési mintái

Ez az alfejezet végigveszi a következtetés standard mintáit, amelyek alkalmazhatók arra, hogy következmények láncolatait vezethessük le, amelyek elvezetnek a kívánt célhoz. Ezeket a következtetési mintákat **következtetési szabály**oknak (**inference rules**) hívjuk. A legjobban ismert szabály a **Modus Ponens** és a következőképpen írható le:

Created by XMLmind XSL-FO Converter.

$$\frac{\alpha \Rightarrow \beta,\ \alpha}{\beta}$$

A jelölés azt jelenti, hogy ha bármikor adott egy $\alpha \Rightarrow \beta$ formájú mondat és adott egy α, akkor a β mondat ebből következik. Például a (*WumpusElőrefelé* ∧ *WumpusÉl*) ⇒ *Lövés* és a (*WumpusElőrefelé* ∧ *WumpusÉl*) adott, akkor a *Lövés* kikövetkeztethető.

Egy másik hasznos következtetési szabály az **És-kiküszöbölés** (**And-Elimination**), ami azt mondja ki, hogy egy konjunkcióból bármely konjunkt kikövetkeztethető:

$$\frac{\alpha \wedge \beta}{\alpha}$$

Például abból, hogy (*WumpusElőrefelé* ∧ *WumpusÉl*) a *WumpusÉl* kikövetkeztethető.

Tekintettel az α és β lehetséges igazságértékeire, könnyen megmutatható, hogy a Modus Ponens és az És-kiküszöbölés helyes egyszer és mindenkorra. Ezek a szabályok felhasználhatók bármely konkrét esetben, ahol alkalmazhatók, helyes következtetéseket eredményezve anélkül, hogy fel kellett volna sorolnunk a modelleket.

A 7.11. ábrán található összes logikai ekvivalencia használható következtetési szabályként. Például az ekvivalencia kiküszöbölés két következtetési szabályt eredményez:

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)} \text{ és } \frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}$$

Nem minden következtetési szabály működik mindkét irányban, mint ezek. Például nem futtathatjuk a Modus Ponenst ellenkező irányban, hogy megkapjuk $\alpha \Rightarrow \beta$-t és α-t a β-ból.

Nézzük meg, hogyan használhatjuk ezeket a következtetési szabályokat és ekvivalenciákat a wumpus világban. Az Sz_1, ..., Sz_5 szabályokból álló tudásbázisból indulunk ki, és megmutatjuk, hogy hogyan bizonyíthatjuk a $\neg C_{1,2}$-t azaz, hogy nincs csapda az [1, 2]-ben. Először alkalmazzuk az ekvivalencia kiküszöbölést Sz_2-re, és így kapjuk, hogy

$$Sz_6: \ (S_{1,1} \Rightarrow (C_{1,2} \vee C_{2,1})) \wedge ((C_{1,2} \vee C_{2,1}) \Rightarrow S_{1,1})$$

Ezután alkalmazzuk az És-kiküszöbölést az Sz_6-ra, és így kapjuk, hogy

$$Sz_7: \ ((C_{1,2} \vee C_{2,1}) \Rightarrow S_{1,1})$$

A kontrapozíció logikai ekvivalenciát alkalmazva:

$$Sz_8: \ (\neg S_{1,1} \Rightarrow \neg (C_{1,2} \vee C_{2,1}))$$

Most alkalmazhatjuk a Modus Ponenst az Sz_8-ra és az Sz_4 érzetre (például a $\neg S_{1,1}$) és így kapjuk:

$$Sz_9: \ \neg (C_{1,2} \vee C_{2,1})$$

Végül alkalmazzuk a De Morgan-szabályt, amely a konklúziót adja:

$$Sz_{10}: \ \neg C_{1,2} \wedge \neg C_{2,1}$$

Tehát sem az [1, 2], sem a [2, 1] négyzet nem tartalmaz csapdát.

Az előző levezetést – következtetési szabályok egy sorozatát – **bizonyítás**nak (**proof**) nevezzük. A bizonyítás megtalálása pontosan olyan, mint megoldást találni egy keresési problémára. Valójában ha a következtetési szabályok összes lehetséges alkalmazásának generálására egy új állapotátmenet-függvényt definiálnánk, akkor minden, a 3. és 4. fejezetbeli kereső algoritmust felhasználhatnánk a bizonyítás megtalálására. A bizonyítás keresése tehát egy alternatívája a modellek felsorolásának. A keresés haladhat előrefelé a kezdeti tudásbázisból kiindulva, alkalmazva a következtetési szabályokat a célmondat levezetéséhez, vagy mehet visszafelé a célmondatból, megpróbálva megtalálni a következtetési szabályoknak olyan láncolatát, amely a kiindulási tudásbázisra alkalmazható szabályokból indul. A fejezetben később bemutatunk két olyan algoritmuscsaládot, amelyek ezeket a technikákat használják.

Created by XMLmind XSL-FO Converter.

Fontos

Az a tény, hogy a következtetés az ítéletlogikában NP-teljes, azt sugallja, hogy a legrosszabb esetet tekintve a bizonyítások keresése sem hatékonyabb a modellek felsorolásánál. Számos gyakorlati esetben azonban, *α bizonyítás megtalálása sokkal hatékonyabb lehet, egyszerűen azért, mert képes figyelmen kívül hagyni az irreleváns állításokat, függetlenül attól, hogy hány van belőlük*. Például az előző bizonyítás, amely elvezetett a $\neg C_{1,2} \wedge \neg C_{2,1}$ mondathoz, nem említi az $S_{2,1}$, $C_{1,1}$, $C_{2,2}$ vagy a $C_{1,2}$ állításokat. Ezeket azért lehet figyelmen kívül hagyni, mert a $C_{1,2}$ célállítás csak az Sz_2-ben jelenik meg, az Sz_2-ben szereplő egyéb állítások pedig csak az Sz_2-ben és az Sz_4-ben, így az Sz_1, Sz_3 és Sz_5 szabályoknak nincs kihatásuk a bizonyításra. Ugyanez maradna a helyzet, ha még millió szabályt hozzáadnánk a tudásbázishoz, miközben az igazságtábla algoritmust ezzel ellentétben elárasztaná a modellek exponenciális robbanása.

A logikai rendszereknek ez a tulajdonsága valójában egy sokkal alapvetőbb jellegzetességükből, a **monotonitás**ból (**monotonicity**) következik. A monotonitás azt mondja ki, hogy a vonzatmondatok halmaza csak bővülhet, ha a tudásbázishoz információt adunk hozzá.[10]

ha $TB \models \alpha$ akkor $TB \wedge \beta \models \alpha$

Például feltételezzük azt, hogy a tudásbázis tartalmaz egy új β állítást, amely azt mondja ki, hogy pontosan 8 csapda van a világban. Ez a tudás segítheti az ágenst további konklúziók levezetésében, de nem teheti érvénytelenné egyik korábban kikövetkeztetett α konklúziót sem – így azt a konklúziót sem, hogy nincsen csapda az [1, 2]-ben. A monotonitás azt jelenti, hogy a következtetési szabályok bármikor alkalmazhatók, ha a megfelelő premisszák megtalálhatók a tudásbázisban – a szabály konklúziójának következnie kell, *függetlenül attól, hogy mi más is van még a tudásbázisban*.

4.1. Rezolúció

Megmutattuk, hogy az eddig ismertetett következtetési szabályok helyesek, de nem tárgyaltuk az ezeket használó következtetési algoritmusok *teljességének* kérdését. A keresési algoritmusok, mint az iteratívan mélyülő keresés 4.5. szakasz - Iteratívan mélyülő mélységi keresés részben teljesek abban az értelemben, hogy meg fogják találni az elérendő célt. Ha azonban a rendelkezésre álló szabályok hiányosak, akkor a cél nem érhető el – nem létezik olyan bizonyítás, amely csak ezeket a szabályokat használja. Például ha kivennénk az ekvivalencia kiküszöbölés szabályt, az előző fejezetbeli bizonyítás nem futna végig. Ez a fejezet egyetlen következtetési szabályt mutat be, a **rezolúció**t (**resolution**), amelynek alkalmazása, párosítva bármelyik teljes keresési módszerrel, egy teljes következtetési algoritmust eredményez.

Először a rezolúciós szabály egy egyszerű változatát fogjuk használni a wumpus világban. Nézzük meg a 7.4. (a) ábrához vezető lépéseket: az ágens visszafordul a [2, 1]-ből az [1, 1]-be, és innen megy az [1, 2]-be, ahol szellőt érez, de bűzt nem. A következő tényeket adjuk hozzá a tudásbázishoz:

Sz_{11}: $\neg S_{1,2}$

Sz_{12}: $S_{1,2} \Leftrightarrow (C_{1,1} \vee C_{2,2} \vee C_{1,3})$

Ugyanazzal a folyamattal, amely az Sz_{10}-hez vezetett korábban, most le tudjuk vezetni, hogy nincs csapda a [2, 2]-ben és az [1, 3]-ban (emlékezzünk, hogy már tudjuk, hogy az [1, 1] csapdamentes):

Sz_{13}: $\neg C_{2,2}$

Sz_{14}: $\neg C_{1,3}$

Az Sz_3-ra is alkalmazhatjuk az ekvivalencia kiküszöbölést, amelyet egy Modus Ponens követ az Sz_5-re, hogy megkapjuk azt a tényt, hogy csapda van az [1, 1], a [2, 2] vagy a [3, 1] négyzetekben:

Sz_{15}: $C_{1,1} \vee C_{2,2} \vee C_{3,1}$

[10] A **nemmonoton logiká**k (**nonmonotonic logic**s), amelyek megszegik a monotonitás tulajdonságot, az emberi érvelésnek azt a szokványos tulajdonságát jelenítik meg, amikor valaki meggondolja magát. Ezeket a 10.7. alfejezetben tárgyaljuk.

Created by XMLmind XSL-FO Converter.

Most következik a rezolúció szabály első alkalmazása: a $\neg C_{2,2}$ literál az Sz_{13}-ban *rezolvál* a $C_{2,2}$ literállal az Sz_{15}-ben, amely ezt adja:

$$Sz_{16}: \; C_{1,1} \vee C_{3,1}$$

Magyarul, ha van egy csapda az [1, 1], [2, 2], [3, 1] négyzetek egyikében, és ez a csapda nem a [2, 2]-ben van, akkor ez az [1, 1]-ben vagy a [3, 1]-ben van. Hasonlóan a $\neg C_{1,1}$ literál az Sz_1-ben rezolvál a $C_{1,1}$ literállal az Sz_{16}-ban, amiből adódik:

$$Sz_{17}: \; C_{3,1}$$

Tehát, ha van egy csapda az [1, 1]-ben vagy a [3, 1]-ben, és ez a csapda nem az [1, 1]-ben van, akkor ez a [3, 1]-ben van. Ez az utolsó két következtetési lépés példa az **egységrezolúció** (**unit resolution**) következtetési szabályra,

$$\frac{\ell_1 \vee \ldots \vee \ell_k, \quad m}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k}$$

ahol ℓ egy literál, ℓ_i és m pedig **kiegészítő literál**ok (**complementary literal**s) (például az egyik negáltja a másiknak). Tehát az egységrezolúció vesz egy **klóz**t (**clause**) – literálok diszjunkcióját – meg egy literált, és létrehoz egy új klózt. Vegyük észre, hogy egy egyedi literált tekinthetünk egy literál diszjunkciójának, amit szoktak **egységklóz**nak (**unit clause**) is nevezni.

Az egységrezolúció szabálya általánosítható teljes **rezolúció**s (**resolution**) szabállyá,

$$\frac{\ell_1 \vee \ldots \vee \ell_k, \qquad m_1 \vee \ldots \vee m_n}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

ahol ℓ_i és m_j kiegészítő literálok. Ha csak kettő hosszúságú klózokkal foglalkozunk, akkor ezt írhatjuk:

$$\frac{\ell_1 \vee \ell_2 \quad \neg \ell_2 \vee \ell_3}{\ell_1 \vee \ell_3}$$

Azaz a rezolúció vesz két klózt, és létrehoz egy új klózt, amely tartalmaz minden literált az eredeti két klózból, kivéve a kiegészítő literálokat. Például:

$$\frac{P_{1,1} \vee P_{3,1}, \quad \neg P_{1,1} \vee P_{2,2}}{P_{3,1} \vee \neg P_{2,2}}$$

Van még egy technikai aspektusa a rezolúciós szabálynak: az eredményklóznak minden literálnak csak egy példányát kell tartalmaznia.[11] A literálok többszörös példányainak kivonását **faktorálás** (**factoring**) hívják. Például ha rezolváljuk az $(A \vee B)$-t a $(A \vee \neg B)$-vel, akkor $(A \vee A)$-t kapunk, amelyet redukálhatunk egyszerűen A-ra.

A rezolúciós szabály *helyes*sége egyszerűen belátható, ha megvizsgáljuk az l_i literált. Ha l_i igaz, akkor m_j hamis, és így $m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n$-nek igaznak kell lennie, mert $m_1 \vee \ldots \vee m_n$ adott. Ha ℓ_i hamis, akkor $\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k$ igaz kell, hogy legyen, mivel $\ell_i \vee \ldots \vee \ell_k$ adott. Így ℓ_i akár igaz, akár hamis, az egyik vagy a másik konklúzió áll, pontosan úgy, ahogy a szabály ezt kimondja.

Fontos

Ami még meglepőbb a rezolúciós szabállyal kapcsolatban, hogy a rezolúció alapjául szolgál *teljes* következtetési algoritmusok egy családjának. *Bármely teljes keresési algoritmus, a rezolúciós szabályt alkalmazva, az ítéletlogikában képes levezetni bármilyen konklúziót, amely vonzata a tudásbázisnak.* Egy figyelmeztetés: a rezolúció csak egy speciális értelemben teljes. Ha adott, hogy A igaz, akkor nem tudjuk a rezolúciót arra használni, hogy levezessük az $A \vee B$ konzekvenciát. Azonban tudjuk arra használni a rezolúciót, hogy megválaszoljuk azt a kérdést, hogy $A \vee B$ igaz-e. Ezt **megcáfolási teljesség**nek (**refutation completeness**) nevezik, ami azt jelenti, hogy a rezolúció mindig használható

[11] Ha a klózt literálok halmazának tekintjük, akkor ezt a korlátozást automatikusan figyelembe vettük. A halmaz jelölést alkalmazva a klózokra a rezolúciós szabály sokkal tisztább lesz egy újabb jelölés bevezetésének árán.

Created by XMLmind XSL-FO Converter.

arra, hogy megerősítsünk vagy megcáfoljunk egy mondatot, de nem alkalmazható az igaz mondatok felsorolására. A következő két alfejezet azt magyarázza el, hogy a rezolúció hogyan oldja ezt meg.

4.1.1. Konjunktív normál forma

Fontos

A rezolúciós szabály csak literálok diszjunkcióira alkalmazható, így úgy tűnhet, hogy ez csak olyan tudásbázis és lekérdezés esetében érdekes, amelyek ilyen diszjunkciókat tartalmaznak. Akkor hogyan vezet ez egy teljes következtetési eljáráshoz az egész ítéletkalkulus számára? A válasz az, hogy *minden ítéletkalkulus mondat logikailag ekvivalens literálok diszjunkcióinak konjunkcióival.* Egy mondatot, amelyet literálok diszjunkcióinak konjunkcióival fejezünk ki, **konjunktív normál formá**jának (**conjunctive normal form**) vagy **CNF** formájúnak nevezünk. A későbbiekben az is hasznos lesz, ha ennek egy korlátosabb családját, a **k-CNF** mondatokat tekintjük. Egy mondat a *k*-CNF-be tartozik, ha pontosan *k* literál van a klózokban.

$$(\ell_{1,1} \vee \ldots \vee \ell_{1,k}) \wedge \ldots \wedge (\ell_{n,1} \vee \ldots \vee \ell_{n,k})$$

Ki fog derülni, hogy minden mondat transzformálható 3-CNF mondattá úgy, hogy azonos marad a modellek halmaza.

Ahelyett hogy bizonyítanánk ezeket a kijelentéseket (lásd 7.10. feladat), leírunk egy egyszerű konvertáló eljárást. Az eljárást az Sz_2 szabály, az $S_{1,1} \Leftrightarrow (C_{1,2} \vee C_{2,1})$ mondat CNF-re konvertálásával illusztráljuk. A lépések a következők:

1. Küszöböljük ki a $\Leftrightarrow$összekötőjelet, helyettesítve az $\alpha \Leftrightarrow \beta$-t $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$-vel:

$$(S_{1,1} \Rightarrow (C_{1,2} \vee C_{2,1})) \wedge ((C_{1,2} \vee C_{2,1}) \Rightarrow S_{1,1})$$

2. Kiküszöböljük az $\Rightarrow$ összekötőjelet, kicserélve $\alpha \Rightarrow \beta$t $\neg\alpha \vee \beta$-ra:

$$(\neg S_{1,1} \vee C_{1,2} \vee C_{2,1}) \wedge (\neg (C_{1,2} \vee C_{2,1}) \vee S_{1,1})$$

3. A CNF megkívánja, hogy a $\neg$ csak literálokra vonatkozzon, így most „a-t beljebb mozgatjuk” a következő 7.11. ábrán felírt ekvivalenciák ismételt alkalmazásával:

$\neg(\neg\alpha) \equiv \beta$ kettős negáció kiküszöbölés

$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$ De Morgan

$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$ De Morgan

A példában csak az utolsó szabály alkalmazására van szükségünk:

$$(\neg S_{1,1} \vee C_{1,2} \vee C_{2,1}) \wedge ((\neg C_{1,2} \wedge \neg C_{2,1}) \vee S_{1,1})$$

4. Most van egy mondatunk, amelyben egymásba ágyazott $\wedge$ és $\vee$ operátorok vannak alkalmazva a literálokra. A 7.11. ábrán bevezetett disztributivitás szabályt alkalmazzuk a $\vee$ operátorokra az $\wedge$ felett mindenhol, ahol lehetséges.

$$(\neg S_{1,1} \vee C_{1,2} \vee C_{2,1}) \wedge (\neg C_{1,2} \vee S_{1,1}) \wedge (\neg C_{2,1} \vee S_{1,1})$$

Az eredeti mondat most már CNF-ben van, három klóz konjunkciójaként. Bár sokkal nehezebb olvasni ezt a formát, de használható a rezolúciós eljárás bemeneteként.

4.1.2. A rezolúció algoritmus

A rezolúción alapuló következtetési eljárások az ellentmondásokra vezető bizonyítások elvén működnek, ahogy azt a 7.4. alfejezet végén tárgyaltuk. Tehát annak megmutatásához, hogy TB $\models \alpha$, azt mutatjuk meg, hogy a $(TB \wedge \neg\alpha)$ kielégíthetetlen. Ezt az ellentmondás bizonyításával végezzük el.

Created by XMLmind XSL-FO Converter.

A rezolúció algoritmust mutatja a 7.12. ábra. Először a ($TB \wedge \neg\alpha$)-t konvertáljuk CNF formára. Majd a rezolúciós szabályt alkalmazzuk a létrejövő klózokra. Minden egyes párt, amely kiegészítő literálokat tartalmaz, rezolválunk, hogy egy új klózt hozzunk létre, amelyet hozzáadunk a halmazhoz, ha még nem volt jelen. A folyamat addig folytatódik, amíg a következő két dolog közül valamelyik meg nem történik:

- nincs több új klóz, amit hozzá lehet adni, ilyen esetben α nem vonzza maga után β-t.
- a rezolúció alkalmazása egy *üres* klózra vezet, amely esetben α -nak vonzata β .

Az üres klóz – egy diszjunkt nélküli diszjunkció – ekvivalens a *Hamis* értékkel, mert a diszjunkció akkor igaz csak, ha legalább az egyik diszjunkt igaz. ∨ gy is beláthatjuk, hogy az üres klóz ellentmondást reprezentál, hogy megfigyeljük azt, hogy az üres klóz két kiegészítő egységklóz, mint amilyen az S és $\neg S$, rezolválásából származik.

7.12. ábra - Egy egyszerű rezolúciós algoritmus az ítéletkalkulushoz. Az IK-REZOLVÁLÁS a két bemenetként megkapott állítás rezolválásából származó összes lehetséges klóz halmazát adja vissza.

function IK-REZOLVÁLÁS(TB, α) **returns** *igaz* vagy *hamis*
  input: TB, a tudásbázis, egy ítéletkalkulus mondat
    α, a lekérdezés, egy ítéletkalkulus mondat

  klózok ← a $TB \wedge \neg\alpha$ mondatot reprezentáló CNF formájú klózok halmaza
  új ← {}
  loop do
    for each C_i, C_j **in** *klózok* **do**
      rezolvensek ← IK-REZOLVÁLÁS(C_i, C_j)
      if *rezolvensek* tartalmazzák az üres klózt **then return** *igaz*
      új ← *új* ∪ *rezolvensek*
    if *új* ⊆ *klózok* **then return** *hamis*
    klózok ← *klózok* ∪ *új*

Alkalmazzuk a rezolúciós eljárást egy nagyon egyszerű következtetésre a wumpus világban. Amikor az ágens az [1,1]-ben van, akkor nincs szellő, tehát nincs csapda a szomszédos négyzetekben. Az ennek megfelelő tudásbázis:

$$TB = Sz_2 \wedge Sz_4 = (S_{1,1} \Leftrightarrow (C_{1,2} \vee C_{2,1})) \wedge \neg S_{1,1}$$

és mi bizonyítani szeretnénk α-t, ami mondjuk $\neg C_{1,2}$. Amikor konvertáljuk a ($TB \wedge \neg\alpha$)-t CNF formára, akkor a 7.13. ábra felső részén látható klózokat kapjuk. Az ábra második sora mutatja az összes klózt, amit az első sor párjainak rezolválásából kaptunk. És akkor, amikor a $C_{1,2}$-t rezolváljuk $\neg C_{1,2}$-vel megkapjuk az üres klózt, amit egy kis négyzet jelöl. A 7.13. ábra megvizsgálása azt mutatja, hogy számos rezolúciós lépés felesleges. Például az $S_{1,1} \vee \neg S_{1,1} \vee C_{1,2}$ ekvivalens az *Igaz* ∨ $C_{1,2}$-vel, ami ekvivalens az *Igaz*zal. Levezetni, hogy az *Igaz* az igaz, nem igazán hasznos. Így bármely olyan klóz, amelyben két kiegészítő klóz szerepel, figyelmen kívül hagyható.

7.12. ábra - Az IK-REZOLÚCIÓ algoritmus részleges alkalmazása egy egyszerű wumpus világbeli következtetésre. A felső sor első négy klóza alapján származtatjuk a $\neg P_{1,2}$-t.

Created by XMLmind XSL-FO Converter.

```
function IK-REZOLVÁLÁS(TB, α) returns igaz vagy hamis
  input: TB, a tudásbázis, egy ítéletkalkulus mondat
         α, a lekérdezés, egy ítéletkalkulus mondat

  klózok ← a TB ∧ ¬α mondatot reprezentáló CNF formájú klózok halmaza
  új ← {}
  loop do
    for each Ci, Cj in klózok do
      rezolvensek ← IK-REZOLVÁLÁS(Ci, Cj)
      if rezolvensek tartalmazzák az üres klózt then return igaz
      új ← új ∪ rezolvensek
    if új ⊆ klózok then return hamis
    klózok ← klózok ∪ új
```

4.1.3. A rezolúció teljessége

Hogy befejezzük a rezolúció tárgyalását, most megmutatjuk, hogy az `IK-REZOLÚCIÓ` algoritmus teljes. Ahhoz, hogy ezt megtehessük, hasznos bevezetni az S klózok halmazának **rezolúciós lezárt**ját (**resolution closure**), $RC(S)$-t, ami az összes olyan klóz halmaza, amely levezethető a rezolúció ismételt alkalmazásaival az S halmazbeli elemekből és ezek leszármazottaiból. Az `IK-REZOLÚCIÓ` a rezolúciós lezárás halmaz elemeit sorolja fel a *klózok* változóban. Könnyű belátni, hogy $RC(S)$-nek végesnek kell lennie, mert véges sok különböző klózt lehetséges konstruálni az $P_1, \ldots, P_k$ szimbólumokból, amelyek elemei S-nek. (Vegyük észre, hogy ez nem volna igaz a faktorálási lépés nélkül, amely megszünteti a literálok többszöröződését.) Így az `IK-REZOLÚCIÓ` mindig terminálódik.

Az ítéletlogikában a rezolúció teljességi tételét **alap rezolúciós tétel**nek (**ground resolution theorem**) nevezzük.

Ha klózok egy halmaza kielégíthetetlen, akkor ezeknek a klózoknak a rezolúciós lezártja tartalmazza az üres klózt.

A tételt az ellentétjének demonstrálásával bizonyítjuk: ha az $RC(S)$ lezártja nem tartalmaz üres halmazt, akkor S kielégíthető. Valójában egy modellt konstruálunk S-nek a megfelelő $P_1, \ldots, P_k$ igazság értékekkel. A konstrukciós eljárás a következő:

Minden i-re 1-től k-ig,

- Ha létezik egy klóz $RC(S)$-ben, amely tartalmazza a $\neg P_i$ literált úgy, hogy minden más literál hamis a választott $P_1, \ldots, P_{i-1}$ hozzárendelés szerint, akkor rendeljünk *hamis* értéket P_i-hez.
- Egyébként rendeljünk *igaz* értéket P_i-hez.

Az maradt hátra, hogy megmutassuk, hogy ez a $P_1, \ldots, P_k$ hozzárendelés modellje S-nek, feltéve, hogy $RC(S)$ zárt a rezolúcióra, és nem tartalmazza az üres klózt. Ennek bizonyítását meghagyjuk feladatnak.

4.2. Előre- és hátrafelé láncolás

A rezolúciót a teljesség tulajdonsága igen fontos következtetési módszerré teszi. Számos gyakorlati esetben azonban, a rezolúció teljes erejére nincs szükség. A valósvilág-beli tudásbázisok gyakran a klózoknak csak egy, **Horn-klóz**nak (**Horn clause**) nevezett korlátosabb fajtáját tartalmazzák. A Horn-klóz literálok olyan diszjunkciója, amelyek közül legfeljebb egy pozitív. Például a $(\neg P_{1,1} \vee \neg \mathit{Szellő} \vee S_{1,1})$ klóz, ahol $P_{1,1}$ jelenti, hogy az ágens pozíciója az [1, 1], egy Horn-klóz, míg az $(S_{1,1} \vee C_{1,2} \vee C_{2,1})$ nem az.

A korlátozás, hogy csak egy pozitív literál lehet, egy kissé önkényesnek és érdektelennek tűnhet, de valójában nagyon fontos, három ok miatt is:

Created by XMLmind XSL-FO Converter.

1. Minden Horn-klóz felírható egy implikációként is, amelynek premisszája pozitív literálok konjunkciója, és konklúziója egyetlen pozitív literál (lásd 7.12. feladat). Például a ($P_{1,1} \vee$ *Szellő* $\vee S_{1,1}$) Horn-klóz átírható a ($P_{1,1} \wedge$ *Szellő*) $\Rightarrow S_{1,1}$ implikációra. Ebben az utóbbi formájában sokkal könnyebb olvasni a mondatot: azt mondja ki, hogyha az ágens az [1, 1]-ben van, és szellő érzékelhető ott, akkor az [1, 1] szellős. Számos területen az emberek számára könnyű a tudást ilyen mondatok formájában olvasni és leírni.

   A Horn-klózokat, mint amilyen az előbbi, vagyis amelyben *pontosan* egy pozitív literál van, **határozott klóz**oknak (**definite clause**s) nevezik. A pozitív literál a **fej** (**head**), míg a negatív literálok alkotják a klóz **test**ét (**body**). Egy negatív literálok nélküli határozott klóz egyszerűen kijelent egy adott állítást, amit gyakran **tény**nek (**fact**) neveznek. A határozott klózok formálják a **logikai programozás** (**logic programming**) alapját, amelyet a 9. fejezetben tárgyalunk. Egy pozitív literálok nélküli Horn-klóz felírható, mint egy olyan implikáció, amelynek a konklúziója a *Hamis* literál. Például a ($\neg W_{1,1} \vee \neg W_{1,2}$) klóz – a wumpus nem lehet mind az [1, 1], mind az [1, 2] mezőben, ekvivalens a $W_{1,1} \wedge W_{1,2} \Rightarrow$ *Hamis* mondattal. Az ilyen mondatokat **integritás kényszer**eknek (**integrity constraint**s) nevezzük az adatbázis-kezelés területén, ahol adathibák jelzésére használják. A következőkben bemutatásra kerülő algoritmusokban az egyszerűség kedvéért feltételezzük, hogy a tudásbázis csak határozott klózokat tartalmaz, és nincsenek benne integritáskényszerek. Azt mondjuk, hogy ezek a tudásbázisok Horn-formában vannak.

2. A Horn-klózokon történő következtetés történhet az **előrefelé láncolás** (**forward chaining**) vagy a **hátrafelé láncolás** (**backward chaining**) algoritmusokkal, amelyeket a következőkben elmagyarázunk. Mindkét algoritmus igen természetes, a következtetési lépések nyilvánvalóak és egyszerűen követhetőek az emberek számára.

3. A maga után vonzás kérdésének eldöntéséhez szükséges idő lineárisan függ a tudásbázis méretétől.

7.14. ábra - Előrefelé láncolás algoritmus az ítéletkalkulus számára. Az *agenda* tartalmazza azokat a szimbólumokat, amelyek ismerten igazak, de még nem „dolgozták fel” őket. A *számol* tábla követi, hogy az egyes implikációknak hány premisszája ismeretlen. Ha az agendáról egy új *p* szimbólumot feldolgozunk, minden olyan implikáció számlálója csökken eggyel, amelynek premisszájában *p* megjelenik. (Ez megoldható konstans időben, ha a *TB* megfelelően indexelt.) Ha a számláló eléri a nullát, azaz az implikáció minden premisszája ismert, akkor az implikáció konklúzióját hozzá lehet adni az agendához. Végül szükségünk van arra, hogy kövessük melyik szimbólumot dolgoztuk már fel; egy kikövetkeztetett szimbólumot nem kell hozzáadni az agendához, ha azt már korábban sikeresen feldolgoztuk. Így elkerülhető a redundáns munka, és megakadályozza a végtelen ciklusok kialakulását, amelyet olyan implikációk okozhatnak, mint a $P \Rightarrow Q$ és $Q \Rightarrow P$.

Created by XMLmind XSL-FO Converter.

```
function IK-EL-VONZAT?(TB, q) returns igaz vagy hamis
  inputs: TB, a tudásbázis, ítéletkalkulus Horn-klózok halmaza
          q, a lekérdezés, egy ítéletkalkulus szimbólum
  lokális változók: számláló, egy tábla, klózokkal indexelve, kezdetben a premisszák száma
                    következmények, egy tábla, szimbólumokkal indexelve, kezdetben minden
                      elem hamis
                    agenda, szimbólumok listája, kezdetben azok a TB-beli szimbólumok, amelyekről
                      ismerjük, hogy igazak

  while agenda nem üres do
    p ← ELEJE(agenda)
    if p = q then return igaz
    unless következmények [p] do
      következmények [p] ← igaz
      for each c Horn-klóz, amelyben a p premissza szerepel do
        csökkent számláló [c]
        if számláló [c] = 0 then
          HOZZÁAD(FEJ[c], agenda)
  return hamis
```

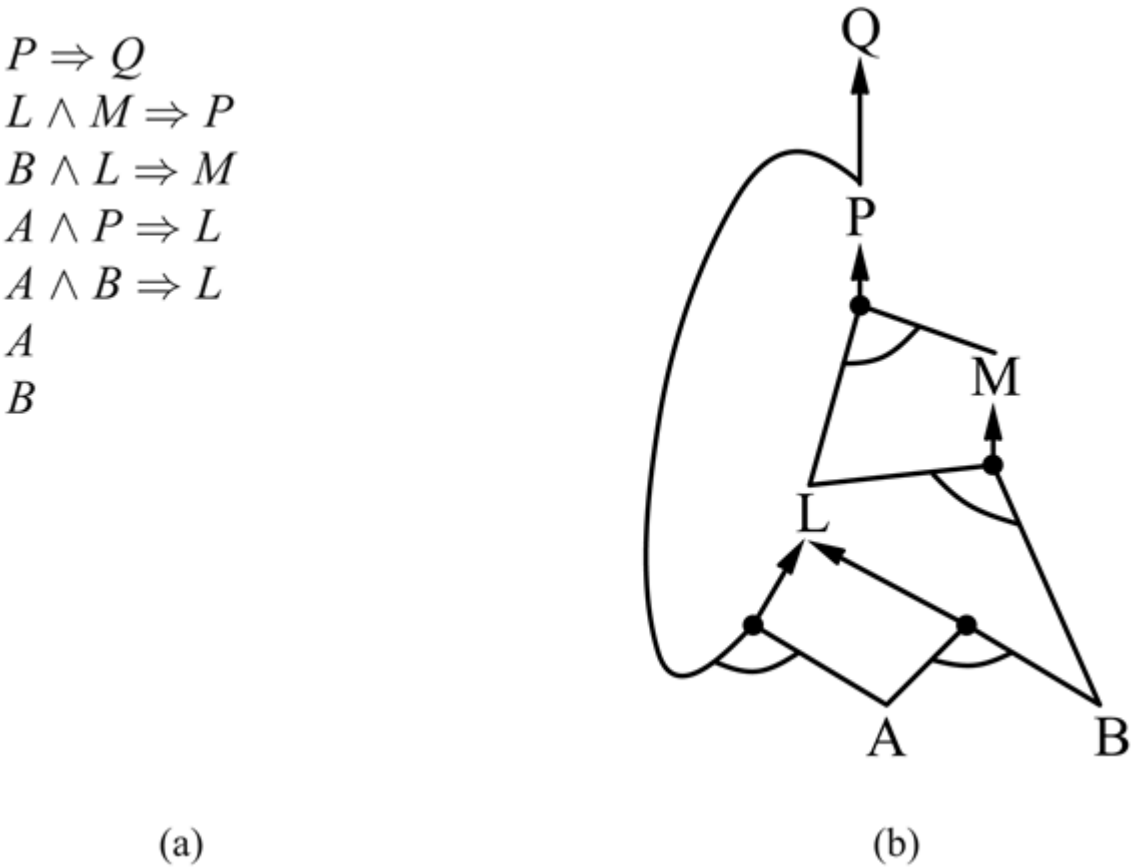
Ez az utolsó tény egy kellemes meglepetés. Azt jelenti, hogy a logikai következtetés nagyon olcsó számos, a gyakorlatban előforduló ítéletkalkulus tudásbázis esetében.

Az `IK-EL-VONZAT?` (*TB*, *q*) előrefelé láncolási algoritmus meghatározza, hogy egy *q* ítéletkalkulus szimbólum – a lekérdezés – vonzata-e egy Horn-klózokat tartalmazó tudásbázisnak. Az algoritmus a tudásbázisban található ismert tényekből (pozitív literálok) indul ki. Ha egy implikáció minden premisszája ismert, akkor a konklúzióját hozzáadjuk az ismert tények halmazához. Például, ha ismert a $P_{1,1}$, és a *Szellő* és a ($P_{1,1} \wedge$ *Szellő*) $\Rightarrow S_{1,1}$ a tudásbázisban van, akkor $S_{1,1}$-t hozzáadhatjuk ehhez. Ez a folyamat folytatódik, amíg vagy a *q* lekérdezést hozzá tudjuk adni az ismert tények halmazához, vagy már nem tudunk további következtetést végezni. A részletes algoritmust a 7.14. ábra mutatja, a legfontosabb tulajdonság, amire emlékeznünk kell, hogy ez lineáris időben fut.

Az algoritmust a legjobban úgy érthetjük meg, ha megnézünk egy példát és egy hozzá tartozó ábrát. A 7.15. (a) ábra egy egyszerű Horn-klózokból álló tudásbázist mutat be, amely az *A* és *B* ismert tényeket tartalmazza. A 7.15. (b) ábra ugyanezt a tudásbázist egy **ÉS-VAGY gráf**ként (**AND-OR graph**) ábrázolja. Az ÉS-VAGY gráfokban a konjunkciókat ívekkel összefogott kapcsolatok – minden egyes kapcsolatot bizonyítani kell – jelölik, míg a diszjunkciókat ívek nélkül összefutó kapcsolatok – elég valamelyik kapcsolatot bizonyítani – jelölik. Könnyű áttekinteni, hogyan is működik az előrefelé láncolás a gráfon. Az ismert levelek (itt *A* és *B*) adottak, és a következtetés addig halad felfelé a gráfon, amíg lehetséges. Bárhol megjelenik egy konjunkció a folyamatban, a továbbterjesztés megáll addig, amíg az összes konjunkt ismertté nem válik, majd ezután halad tovább. Javasoljuk az olvasónak, hogy vegye át részletesen a példát.

7.15. ábra - (a) Egyszerű Horn-klózokból álló tudásbázis. (b) Ugyanez a tudásbázis ÉS-VAGY gráfként.

Created by XMLmind XSL-FO Converter.



Fontos

Könnyű belátni, hogy az előrefelé láncolás **helyes**: minden következtetés valójában a Modus Ponens egy alkalmazása. Az előrefelé láncolás **teljes** is: minden vonzat atomi mondatot vezet le. A legegyszerűbben úgy érthetjük meg ezt, ha a *kikövetkeztetett* állapot tábla végső állapotát tekintjük (miután az algoritmus elért egy **fix pont**ot (**fixed point**), ahol már újabb következtetés nem lehetséges). A tábla *igaz* értéket tartalmaz minden, a folyamat során kikövetkeztetett szimbólumra, és *hamis* értéket a többi szimbólumra. A táblát úgy tekinthetjük, mint egy logikai modellt, ráadásul *minden határozott klóz az eredeti TB-ben igaz ebben a modellben*. Ennek belátására feltételezzükaz ellenkezőjét, nevezetesen, hogy valamely $a_1 \wedge \ldots \wedge a_k \Rightarrow \beta$ klóz hamis a modellben. Ez azt jelenti, hogy $a_1 \wedge \ldots \wedge a_k$-nak igaznak, míg β-nek hamisnak kell lennie a modellben. Ez viszont ellentmond annak a feltételezésünknek, hogy az algoritmus elérte a fix pontot! Így levonhatjuk azt a következtetést, hogy a kikövetkeztetett atomi mondatok halmaza a fix pontban az eredeti *TB* egy modelljét definiálja. Ezenkívül, bármely *q* atomi mondatnak, amely vonzata a *TB*-nek, igaznak kell lennie ennek minden modelljében, így ebben az említett modellben is. Ennélfogva az algoritmusnak ki kell következtetnie minden *q* vonzat mondatot.

Az előrefelé láncolás egy példája az általános **adatvezérelt**nek (**data-driven**) nevezett következtetési elvnek, amely olyan következtetéseket jelent, ahol a figyelem fókusza kezdetben az ismert adatokon van. Ez a megközelítés alkalmazható az ágenseknél a bejövő érzetekből történő következtetések levezetésére, gyakran anélkül is, hogy valamilyen kérdés lebegne a szemünk előtt. Például a wumpus ágens `KIJELENT`-heti az érzeteit a tudásbázisnak, felhasználva egy inkrementális előrefelé következtető algoritmust, amelyben új tényeket lehet felvenni egy előjegyzési listára, hogy további következtetés inicializálódjon. A mindennapi életben, jelentős számban fordul elő adatvezérelt következtetés, amikor új információk érkeznek. Például ha bent vagyok a szobában és hallom, hogy elkezd esni, akkor felmerülhet bennem, hogy a tervezett kirándulás elmarad. Ugyanakkor az valószínűleg nem fog megfordulni a fejemben, hogy a szomszéd kertjében a legnagyobb rózsa tizenhetedik szirma nedves lesz. Az emberek óvatos kontroll alatt tartják az előrefelé láncolást, nehogy elárasszák őket az irreleváns következmények.

A visszafelé láncolás algoritmus, ahogy azt a neve is sugallja, visszafelé működik a lekérdezésből indulva. Ha a *q* lekérdezésről tudjuk, hogy igaz, akkor nem szükséges további munka. Egyébként az algoritmus megtalál

Created by XMLmind XSL-FO Converter.

minden olyan implikációt a tudásbázisban, amelynek a következménye q. Ha valamelyik ilyen implikációnak az összes premisszáját be lehet bizonyítani (visszafelé láncolással), akkor q igaz. Amikor a 7.15. ábrán a Q lekérdezésre alkalmazzuk a módszert, akkor ez az ábrán követve vissza, lefelé működik, addig, amíg el nem éri az ismert tények halmazát, ami a bizonyítás alapja. A részletes algoritmust meghagyjuk feladatnak, és mint az előrefelé láncolásnál, egy hatékony implementáció itt is lineáris időben működik.

A visszafelé láncolás a **célorientált következtetés** (**goal-oriented reasoning**) egy formája. Hasznos-e megválaszolni olyan kérdéseket, mint hogy „Mit tegyek most?”, vagy hogy „Hol vannak a kulcsaim?”. Gyakran a visszafelé láncolás költsége *sokkal kisebb,* mint a tudásbázis méretétől lineárisan függő költség, mert a folyamat csak releváns tényeket érint. Általánosságban, egy ágensnek meg kell osztania a munkát az előrefelé és hátrafelé láncolás között, korlátozva az előrefelé következtetés alkalmazását azokra a tényekre, amelyek valószínűleg relevánsak azon lekérdezésekhez, amelyeket az ágens visszafelé láncolással fog megoldani.

5. Hatékony ítéletkalkulus következtetés

Ebben a szakaszban az ítéletkalkulus következtetés két hatékony családját írjuk le, amelyek a modellellenőrzésen alapulnak: az egyik megközelítés a visszalépéses keresésen alapul, a másik a hegymászó keresésen. Ezek az algoritmusok az ítéletkalkulus „technológiájának” részei. Ez a szakasz átlapozható a fejezet első olvasásakor.

Az algoritmusok, amelyeket leírunk, a kielégíthetőséget ellenőrzik. Már megfigyeltük a kapcsolatot egy logikai mondatot kielégítő modell megtalálása és egy kényszerkielégítési probléma megoldásának megtalálása között, így talán nem meglepő, hogy a két algoritmus család igen hasonlít az 5.2. alfejezetben bemutatott visszalépéses algoritmusokra és az 5.3. alfejezet lokális keresési algoritmusaira. Ezek az algoritmusok azonban rendkívül fontosak saját maguk jogán is, mert számos kombinatorikai probléma a számítástechnikában visszavezethető egy ítéletkalkulus mondat kielégíthetőségének ellenőrzésére. A kielégíthetőségi algoritmusok terén elért bármilyen haladás általánosságban is óriási hatással van a képességeinkre a komplexitás kezelésében.

5.1. Egy teljes visszalépéses algoritmus

Az első algoritmus, amit megnézünk Martin Davis és Hilary Putnam (1960) jelentős konferenciacikke alapján, az úgynevezett **Davis–Putnam-algoritmus**. Ez az algoritmus valójában egy változata annak, amelyet Davis, Logemann és Loveland (1962) publikált, így DPLL-nek fogjuk hívni szerzőik neveinek kezdőbetűi alapján. A DPLL bemenetként egy konjunktív normál formájú mondatot vesz – a klózok egy halmazát. Mint a `VISSSZALÉPÉSES-KERESÉS` vagy az `IK-VONZAT?`, ez is alapvetően rekurzív, mélységi felsorolását végzi a lehetséges modelleknek. Az algoritmus három továbbfejlesztést tartalmaz az `IK-VONZAT?` egyszerű sémájához képest:

- *Korai leállás:* Az algoritmus észreveszi, ha egy mondat már biztos igaz vagy hamis, még részben elkészült modell alapján is. Egy klóz igaz, ha *bármelyik* literál igaz, még akkor is, ha a többi literálnak még nincs igazság értéke. Ennélfogva a mondatot mint egészet igaznak ítélhetjük, még mielőtt a modell teljes volna. Például az $(A \vee B) \wedge (A \vee C)$ igaz, ha A igaz, függetlenül B és C értékétől. Hasonlóan, egy mondat hamis, ha *bármelyik* klóz hamis, ami akkor fordul elő, ha ennek a klóznak minden literálja hamis. Ismételten, ez a modell teljessé válásánál sokkal korábban is előfordulhat. A korai leállás a keresési tér egész részfáinak átvizsgálását kerüli el.

- *Tiszta szimbólum heurisztika:* Egy **tiszta szimbólum** (**pure symbol**) egy olyan szimbólum, amely mindig ugyanolyan „előjellel” szerepel minden klózban. Például a következő három mondatban, $(A \vee \neg B)$, $(\neg B \vee \neg C)$ és az A szimbólum tiszta, mivel csak a pozitív literálja jelenik meg, a B tiszta, mivel csak a negatív literálja jelenik meg, és C nem tiszta. Könnyű belátni, hogyha egy mondatnak van modellje, akkor létezik tiszta szimbólumokat tartalmazó modellje is, amelyben a tiszta szimbólumok értéke úgy van megválasztva, hogy literáljai *igazak* legyenek, hiszen így egyetlen klózt sem teszünk hamissá. Vegyük észre, hogy egy szimbólum tisztaság tulajdonságának meghatározásakor az algoritmus figyelmen kívül hagyhatja azokat a klózokat, amelyekről már tudjuk, hogy igazak a modell eddigi konstruálása alapján. Például ha a modellünk tartalmazza a $B = hamis$ hozzárendelést, akkor a $(\neg B \vee \neg C)$ klóz már igaz, és C tisztává válik, mert csak a $(C \vee A)$-ban jelenik meg.

- *Egységklóz heurisztika:* Az **egységklóz**t már definiáltuk korábban, mint egy olyan klózt, amelynek egy literálja van. A DPLL esetében, ez azokat a klózokat is jelenti, hogy egy kivételével minden literál *hamis* értéket kapott a modellben. Például ha a modell tartalmazza a $B = hamis$ hozzárendelést, akkor a $(B \vee \neg C)$

Created by XMLmind XSL-FO Converter.

egység klózzá válik, mivel ekvivalens a (*Hamis* ∨ ¬*C*)-vel, azaz a ¬*C*-vel. Természetesen ahhoz, hogy ez a klóz igaz legyen, *C*-nek *hamis* értéket kell adni. Az egységklóz heurisztika elvégzi ezeket a hozzárendeléseket, mielőtt elágazna a maradékon. Egy fontos konzekvenciája ennek a heurisztikának, hogy bármely, már a tudásbázisban található literálra vonatkozó (cáfolat általi) bizonyítási kísérlet azonnal sikeres lesz (7.16. feladat). Vegyük észre azt is, hogy értéket adva egy egységklózhoz újabb egységklózt hozhatunk létre. Például amikor *C*-nek *hamis* értéket adunk, akkor a (*C* ∨ *A*) egységklózzá válik, ami az igaz *A*-hoz való hozzárendelését eredményezi. Ezt az „egymást követő" kikényszerített hozzárendelések sorozatát **egységterjesztés**nek (**unit propagation**) nevezik. Hasonlít a folyamat a Horn-klózokkal történő előrefelé láncolásra, és valóban, abban az esetben ha a CNF formájú kifejezés csak Horn-klózokat tartalmaz, akkor a DPLL lényegében lemásolja az előrefelé láncolást (lásd 7.17. feladat).

7.16. ábra - A DPLL algoritmus ítéletkalkulus mondatok kielégíthetőségének ellenőrzése. A TISZTA-SZIMBÓLUM-KERESÉS és az EGYSÉG-KLÓZ-KERESÉS eljárásokat a szövegben elmagyaráztuk; mindkettő egy szimbólummal vagy nullával tér vissza, és a szimbólumhoz hozzárendelendő igazságértékkel. Mint az IT-VONZAT? eljárás, ez is részleges modelleken dolgozik.

```
function DPLL-KIELÉGÍTHETŐSÉG?(s) returns igaz vagy hamis
  inputs: s, egy ítéletkalkulus mondat

  klózok ← az s mondat CNF formájú klózainak halmaza
  szimbólumok ← az s mondat szimbólumainak listája
  return DPLL(klózok, szimbólumok, [])

function DPLL(klózok, szimbólumok, modell) returns igaz vagy hamis

  if minden klóz a klózok halmazban igaz a modell modellben then return igaz
  if néhány klóz a klózok halmazban hamis a modell modellben then return hamis
  P, érték ← TISZTA-SZIMBÓLUM-KERESÉS(klózok, szimbólumok, modell)
  if P nem nulla then return DPLL(klózok, szimbólumok–P, KIEGÉSZÍT(P, érték, modell))
  P, érték ← EGYSÉG-KLÓZ-KERESÉS(klózok, modell)
  if P nem nulla then return DPLL(klózok, szimbólumok–P, KIEGÉSZÍT(P, érték, modell))
  P ← ELSŐ(szimbólumok); maradék ← MARADÉK(szimbólumok);
  return  DPLL(klózok, maradék, KIEGÉSZÍT(P, igaz, modell)) or
          DPLL(klózok, maradék, KIEGÉSZÍT(P, hamis, modell))
```

A DPLL algoritmust mutatja a 7.16. ábra. Az ábrán a lényeges elemeket tartalmazó vázat adtuk meg, ami bemutatja magát a keresési folyamatot. Nem szerepel az adatstruktúra leírása, amelyet fenn kell tartani, hogy a keresési lépéseket hatékonyan lehessen elvégezni, sem azoknak a trükköknek a leírása, amelyeket a teljesítmény fokozása céljából az alapalgoritmushoz hozzá lehet adni: klóztanulás, változó választási heurisztika és a véletlenszerű újraindítások technikája. Ha ezeket is beépítjük az algoritmusba, akkor a DPLL az egyik leggyorsabb kielégíthetőségi algoritmus, antikvitása ellenére is. A CHAFF implementáció millió változós hardververifikációs problémák megoldására szolgált.

5.2. Lokális keresés algoritmus

Számos lokális keresés algoritmust láttunk már a könyvben, beleértve a HEGYMÁSZÓ algoritmust 3.1. szakasz - Hegymászó keresés részben és a SZIMULÁLT-LEHŰTÉS-t 3.2. szakasz - Szimulált lehűtés részben. Ezek az algoritmusok alkalmazhatók közvetlenül is kielégíthetőségi problémákra, feltéve, hogy megfelelő kiértékelő függvényt választunk. Mivel a cél egy olyan hozzárendelés megtalálása, amely kielégít minden klózt, megfelel számunkra egy olyan kiértékelő függvény választása, amely a kielégítetlen klózok számát számolja. Valóban, ez pontosan az a mérték, amit a MIN-KONFLIKTUS algoritmus használt kényszerkielégítési problémáknál 3. szakasz - Lokális keresés kényszerkielégítési problémáknál részben. Minden ilyen algoritmus a teljes hozzárendelések terében végez lépéseket, cserélgetve egy-egy lépésben egy-egy szimbólum igazságértékét. A tér rendszerint számos lokális minimumot tartalmaz, amelyekből a meneküléshez a véletlenszerű lépések különféle formái

Created by XMLmind XSL-FO Converter.

szükségesek. Az utóbbi években nagyon sokat foglalkoztak azzal a kérdéssel, hogy hogyan lehet egy jó egyensúlyt találni a mohóság és a véletlenszerűség között.

Az egyik legegyszerűbb és leghatékonyabb algoritmus, amely ezen munkák között felbukkant a WALKSAT (7.17. ábra). Minden iterációban az algoritmus vesz egy kielégítetlen klózt és egy szimbólumot a klózból, amelynek értékét ellenkezőre cseréli. Az értéket cserélő szimbólum kiválasztása a következő két módszer közül – véletlenszerűen választva – az egyikkel történik: (1) a „min-konfliktus" lépés, amely minimalizálja a kielégítetlen klózokat az új állapotban, és a (2) „véletlen-bejárás", amely véletlenszerűen választja a szimbólumot.

7.17. ábra - A WALKSAT algoritmus, kielégíthetőségének ellenőrzése véletlenszerűen cserélgetetett változó értékekkel. Számos változata ismert az algoritmusnak.

```
function WALKSAT(klózok, p, max_csere) returns egy kielégítő modell vagy kudarc
  inputs: klózok, ítéletkalkulus klózok halmaza
          p, a „véletlen lépés" valószínűségének értéke, általában 0,5 körüli
          max_csere, a megengedett cserék száma, mielőtt feladná az algoritmus

  modell ← igaz/hamis értékek véletlen hozzárendelései a klózok szimbólumaihoz
  for i = 1 to max_csere do
    if modell kielégíti klózok then return modell
    klózok ← véletlenszerűen választott klóz a klózok elemei közül, amely hamis a modell-ben
    with probability p érték csere egy véletlenszerűen választott klóz-ban a modell-ben
    else bármelyik szimbólum a klóz-ban, amely maximalizálja a kielégített klózok számát
  return kudarc
```

Működik-e egyáltalán a WALKSAT? Az egyértelmű, hogy ha visszaad egy modellt, akkor a bemeneti mondat valóban kielégíthető. Mi van akkor, ha *kudarccal* tér vissza? Sajnos ebben az esetben nem tudjuk megmondani, hogy a mondat kielégíthetetlen vagy az algoritmus számára több időt kellene hagyni a keresésre. Megpróbálhatjuk beállítani a *max_csere* értékét végtelenre. Erre az esetre könnyű megmutatni, hogy a WALKSAT végül vissza fog adni egy modellt (ha létezik ilyen), feltéve, hogy a p valószínűségre igaz a $p > 0$. Ez azért van így, mert mindig létezik egy olyan cserélési sorozat, amely elvezet egy kielégítő hozzárendeléshez, és előbb-utóbb a véletlen bejárás lépések elő fogják állítani ezt a sorozatot. Persze ha a *max_csere* végtelen, és a mondat kielégíthetetlen, akkor az algoritmus soha nem fog leállni!

Ezek az eredmények azt sugallják, hogy a lokális keresési algoritmusok, mint a WALKSAT, akkor a leginkább hasznosak, ha feltételezhetjük, hogy létezik megoldás – mint például a 3. és 5. fejezetben tárgyalt problémáknak általában van megoldásuk. Viszont a lokális keresés nem képes detektálni a *kielégíthetetlenséget,* amely szükséges a maga után vonzás kérdésének az eldöntéséhez. Például egy lokális keresést alkalmazó ágens nem tudja *megbízhatóan* bizonyítani, hogy egy négyzet biztonságos a wumpus világban. Ehelyett azt tudja mondani, hogy „Egy órája gondolkodom a kérdésen, és nem tudtam olyan lehetséges világot találni, amiben a négyzet nem biztonságos." Ha a lokális keresés algoritmus általában igazán gyors egy modell megtalálásában, ha létezik ilyen, akkor az ágenst tekinthetjük igazolhatónak, feltételezve, hogy a kudarc a kielégíthetetlenséget jelzi. Ez természetesen nem azonos egy bizonyítással, és az ágensnek kétszer is meg kell gondolnia, mielőtt erre bízza az életét.

5.3. Nehéz kielégíthetőségi problémák

Most megnézzük, hogyan is teljesít a DPLL és a WALKSAT a gyakorlatban. Minket elsősorban a *nehéz* problémák érdekelnek, mert a *könnyű* problémák megoldhatóak bármely régi algoritmussal is. Az 5. fejezetben láttunk néhány meglepő felfedezést bizonyosfajta problémákkal kapcsolatban. Például az n-királynő probléma – amely igen kemény feladat volt a visszalépéses keresés számára – triviálisan egyszerűnek bizonyult az olyan lokális keresési módszerek számára, mint például a min-konfliktus. Ennek az az oka, hogy a megoldások nagyon sűrűn oszlanak el a hozzárendelések terében, és bármely kezdeti hozzárendeléshez garantáltan találhatunk egy megoldást a közelben. Tehát az n-királynő könnyű probléma, mert **alulhatározott (underconstrained).**

Created by XMLmind XSL-FO Converter.

Amikor konjunktív normál formában levő kielégíthetőségi problémákat tekintünk, alulhatározott probléma az, amelyben viszonylag kevés a változókat korlátozó klóz szerepel. Például itt van egy véletlenszerűen létrehozott[12] 3-CNF mondat öt szimbólummal és öt klózzal:

$$(\neg D \vee \neg B \vee C) \wedge (B \vee \neg A \vee \neg C) \wedge (\neg C \vee \neg B \vee E) \wedge (E \vee \neg D \vee B) \wedge (B \vee E \vee \neg C)$$

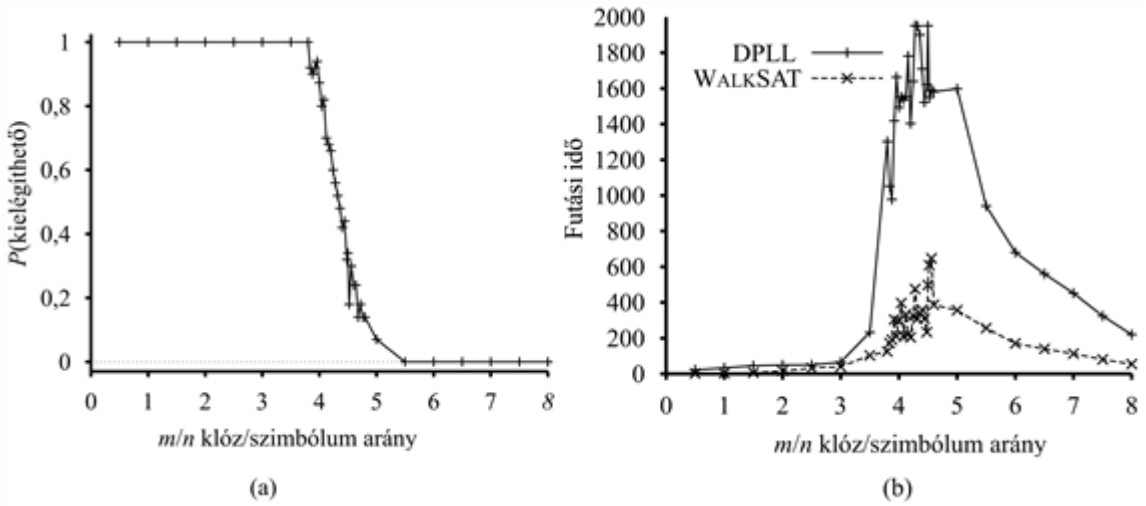
A 32 lehetséges hozzárendelésből 16 modellje a mondatnak, így átlagosan csak két véletlenszerű találgatásra van szükség egy modell megtalálásához.

Akkor melyek a nehéz problémák? Feltételezhető, hogyha *növeljük* a klózok számát, miközben a szimbólumok számát rögzítve tartjuk, a problémát erősebben határozottá tesszük, és a megoldás megtalálása egyre nehezebbé válik. Legyen m a klózok száma és n a szimbólumok száma. A 7.18. (a) ábra mutatja annak a valószínűségét, hogy egy véletlenszerűen választott 3-CNF mondat kielégíthető-e a klóz/szimbólum arány (m/n) függvényében rögzített $n = 50$ mellett. Ahogy vártuk, kis m/n aránynál a valószínűség közel van 1-hez, és nagy m/n aránynál közel van a 0-hoz. A valószínűség viszonylag élesen esik le az $m/n = 4{,}3$ értéknél. Azok a CNF mondatok, amelyek közel vannak ehhez a **kritikus pont**hoz (**critical point**), „alig kielégíthetőnek” vagy „alig kielégíthetetlennek” jellemezhetők. Ez volna az, ahol a nehéz problémák vannak?

A 7.18. (b) ábra mutatja a DPLL és a `WALKSAT` futási időit ennek a pontnak a környezetében, csak a kielégíthető problémákat véve figyelembe. Három dolog tiszta: először is, a kritikus pont körüli problémák *sokkal* nehezebbek, mint a többi probléma. Másodszor, a DPLL igen hatékony még a legnehezebb problémákra is – átlagosan néhány ezer lépést tesz, összehasonlítva az igazságtábla felsorolás $2^{50} \approx 10^{15}$ számú lépésigényével. Harmadszor, a `WALKSAT` a teljes tartományban sokkal gyorsabb, mint a DPLL.

Természetesen ezek az eredmények csak a véletlenszerűen generált problémákra vonatkoznak. A valós problémáknak nem feltétlenül hasonló a struktúrájuk – a pozitív és negatív állítások aránya, a klózok közti kapcsolatok sűrűsége, és egyéb más vonatkozások – mint a véletlen problémáknak. Mégis, a gyakorlatban a `WALKSAT` és az ehhez kapcsolódó, hasonló algoritmusok nagyon jók valós problémák megoldásában is – gyakran ugyanolyan jók, mint az adott feladatra készített legjobb speciális célú algoritmus. Több ezer szimbólumot és milliónyi klózt tartalmazó problémákat megoldókkal szokás kezelni, mint amilyen a `CHAFF`. Ezek a megfigyelések azt sugallják, hogy a min-konfliktus és a véletlen bejárás algoritmusok valamilyen kombinációja *általános célú* képességet biztosít a legtöbb szituáció elemzéséhez, ahol kombinatorikai következtetés szükséges.

7.18. ábra - (a) A grafikon a klóz/szimbólum arány függvényében annak valószínűségét mutatja, hogy $n = 50$ mondat közül véletlenszerűen választott 3-CNF mondat kielégíthető-e. (b) A grafikon a DPLL és a `WALKSAT` algoritmusok futási időinek középértékét mutatja 100 *kielégíthető* 3-CNF mondaton $n = 50$ mellett, a kritikus m/n arány melletti szűk sávban.



[12] Minden klóz három véletlenszerűen kiválasztott szimbólumot tartalmaz, amelyek 50% valószínűséggel negáltak.

Created by XMLmind XSL-FO Converter.

6. Ítéletlogikát alkalmazó ágensek

Ebben a szakaszban, összeszedünk mindent, amit eddig tanultunk, hogy olyan ágenseket konstruáljunk, amelyek az ítéletlogika felhasználásával működnek. Kétfajta ágenst fogunk megvizsgálni: olyat, amely következtetési algoritmust és egy tudásbázist használ, mint a 7.1. ábra általános tudásbázisú ágense, és olyat, amely a logikai kifejezések kiértékelését közvetlenül, áramköröket alkalmazva végzi. Mindkétfajta ágenst a wumpus világban fogjuk demonstrálni, és látni fogjuk, hogy mindkettő komoly hátrányoktól szenved.

6.1. Csapdák és wumpusok megtalálása logikai következtetés felhasználásával

Kezdjünk egy ágenssel, amely képes logikusan érvelni a csapdák és wumpusok helyével és a biztonságos négyzetekkel kapcsolatban. A feladat megoldása a tudásbázis felírásával kezdődik, amely reprezentálja a wumpus világ „fizikáját”. Az ágens tudja, hogy az [1, 1] nem tartalmaz se csapdát, se wumpust, azaz $\neg C_{1,1}$ és $\neg W_{1,1}$. Az ágens minden [*x*, *y*] négyzetre ismer egy mondatot, ami azt jelenti ki, hogy a szellő honnan ered:

$$S_{x,y} \Leftrightarrow (C_{x,y+1} \vee C_{x,y-1} \vee C_{x+1,y} \vee C_{x-1,y}) \quad (7.1)$$

Az ágens minden [*x*, *y*] négyzetre ismer egy mondatot, ami azt jelenti ki, hogy a bűz hogyan ered:

$$B_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y}) \quad (7.2)$$

Végül tudja, hogy pontosan egy wumpus van. Ezt két részben fejezzük ki. Először ki kell jelentenünk, hogy *legalább* egy wumpus van:

$$W_{1,1} \vee W_{1,2} \vee \ldots \vee W_{4,3} \vee W_{4,4}$$

Ezután azt kell mondanunk, hogy *legfeljebb* egy wumpus van. Az egyik lehetséges módja ennek, hogy minden két mezőre kijelentjük, hogy a két mező egyikének wumpusmentesnek kell lennie. Ha *n* négyzet van, akkor $n(n - 1)/2$ mondatot kapunk, olyanokat mint a $\neg W_{1,1} \vee \neg W_{1,2}$. Egy 4 × 4-es világ esetében, így összesen 64 különböző szimbólumot tartalmazó 155 mondattal kezdünk.

Az ágensprogram, amelyet a 7.19. ábra mutat, `KIJELENTI` a tudásbázisnak az összes szellő és bűz érzetet. (Ezenkívül frissíti néhány hagyományos változójának értékét is, amelyek azt őrzik, hogy az ágens merre van most és merre járt korábban – erre később még visszatérünk). Ezután a program kiválasztja, hogy melyik cellát nézze meg a peremen lévők közül – olyan négyzetek közül, amelyek szomszédosak valamelyik meglátogatottal. Egy peremen lévő [*x*, *y*] cella *bizonyíthatóan biztonságos*, ha a $(\neg C_{ij} \wedge \neg W_{ij})$ mondat vonzata a tudásbázisnak. A „második legjobb dolog”, ha egy cella *lehetséges, hogy biztonságos*, amikor az ágens nem tudja bizonyítani, hogy ott egy csapda vagy egy wumpus *van* – azaz, hogy a $(C_{ij} \vee W_{ij})$ *nem* vonzat.

A `KÉRDEZ` eljárásban levő maga után vonzás számítása implementálható a fejezetben korábban bemutatott a módszerek közül bármelyikkel. Az IT-`VONZAT`? (7.10. ábra) nyilván nem praktikus, mivel fel kellene sorolni a 2^{64} sort. A DPLL (7.16. ábra) néhány milliszekundum alatt végrehajtja a kívánt következtetést, leginkább az egység-propagáció heurisztikának köszönhetően. A `WALKSAT` szintén használható, a teljesség hiányából származó gyengeségeivel. A wumpus világokban, a modell megtalálásának kudarca, 10 000 cserét feltételezve, praktikusan megfelel a kielégíthetetlenségnek, így nem valószínű, hogy hiba fordulhatna elő a teljesség hiánya miatt.

7.19. ábra - (Egy wumpusvilág-beli ágens programja, amely ítéletkalkulust használ a csapdák, a wumpusok és a biztonságos négyzetek azonosítására. Az `ÚTVONAL-PROBLÉMA` szubrutin létrehoz egy keresési problémát, aminek megoldása egy cselekvéssorozat, amely elvezet az [*x*, *y*] négyzetből az [*i*, *j*]-be, és csak korábban már látogatott négyzeteken vezet keresztül.

Created by XMLmind XSL-FO Converter.

```
function IK-WUMPUS-ÁGENS(érzet) returns egy cselekvés
  inputs: érzet, egy lista, [bűz, szellő, ragyogás]
  static: TB, egy tudásbázis, kezdetben a wumpus világ „fizikáját” tartalmazza
          x, y, orientáció, az ágens pozíciója (kezdetben 1, 1) és orientációja (kezdetben jobbra)
          meglátogatott, egy tömb, meglátogatott négyzetek megmutatása, kezdetben hamis
          cselekvés, az ágens által végrehajtott utolsó cselekvés, kezdetben nulla
          terv, egy cselekvéssorozat, kezdetben üres

  frissít x, y, orientáció, meglátogatott a végrehajtott cselekvés alapján
  if bűz then KIJELENT(TB, B_{x,y}) else KIJELENT(TB, ¬B_{x,y})
  if szellő then KIJELENT(TB, S_{x,y}) else KIJELENT(TB, ¬S_{x,y})
  if ragyogás then cselekvés ← megragad
  else if terv nem üres then cselekvés ← ELSŐ(terv)
  else if valamely szélső [i, j] négyzetre, KÉRDEZ(TB, (¬C_{i,j} ∧ ¬W_{i,j}) igaz or
          valamely szélső [i, j] négyzetre, KÉRDEZ(TB, (C_{i,j} ∨ W_{i,j}) hamis then do
      cselekvés ← A*-GRÁF-KERESÉS(ÚTVONAL-PROBLÉMA([x, y], orientáció, ([i, j], meglátogatott))
      cselekvés ← ELSŐ(terv)
  else cselekvés ← véletlen választott lépés
  return cselekvés
```

Az `IK-WUMPUS-ÁGENS` igen jól működik kisméretű wumpus világban. Azonban mégis van valami erősen nem megnyugtató az ágens tudásbázisával kapcsolatban. A *TB* tartalmazza a fizikai leírást a (7.1) és (7.2) alakú egyenletek formájában *minden egyes négyzetre*. Minél nagyobb a környezet, annál nagyobb kezdeti tudásbázisra van szükség. Sokkal inkább azt szeretnénk, hogy csak két mondatunk lenne, amelyek kimondják, hogy hogyan is keletkezhet a szellő vagy a bűz *bármely* kockában. Ez mát túl van az ítéletkalkulus kifejezési képességén. A következő fejezetben látni fogunk egy nagyobb kifejezőerővel rendelkező logikai nyelvet, amelyben egyszerű lesz ilyen mondatot kijelenteni.

6.2. A hely és az irány nyomkövetése

A 7.19. ábra ágensprogramja „csal”, mert a helyzetének nyomkövetését a tudásbázison *kívül* oldja meg, ahelyett hogy logikai következtetést végezne.[13] Hogy „helyesen” oldjuk meg a nyomkövetést, szükségünk lesz az elhelyezkedéssel kapcsolatos állításokra. Első hallásra hajlamosak lehetünk felvenni egy $H_{1,1}$ szimbólumot, ami azt reprezentálná, hogy az ágens az [1, 1]-ben van. Így a kezdeti tudásbázis ilyen mondatokat tartalmazhatna:

$H_{1,1} \wedge ArccalJobbra \wedge Halad \Rightarrow H_{2,1}$

Azonnal láthatjuk, hogy ez nem fog működni. Ha az ágens kiindul az [1, 1]-ből arccal jobbra és halad abban az irányban, akkor a tudásbázisból következni fog a $H_{1,1}$ (az eredeti elhelyezkedés) és a $H_{2,1}$ (az új elhelyezkedés). Mégsem lehet azonban mindkét állítás igaz! A probléma az, hogy az elhelyezkedés állításoknak két különböző időpontra kell vonatkozniuk. Szükségünk van egy $H^1_{1,1}$ szimbólumra, ami azt jelentené, hogy az ágens az [1, 1]-ben van az 1 időpontban, és egy $H^2_{2,2}$ szimbólumra, ami azt, hogy az ágens a [2, 1]-ben van a 2 időpontban és így tovább. Az irány és az cselekvés állításoknak szintén függeniük kell az időtől. Így a helyes mondatok:

$$H^1_{1,1} \wedge ArccalJobbra^1 \wedge Halad^1 \Rightarrow H^2_{2,1}$$
$$ArccalJobbra^1 \wedge FordulBalra^1 \Rightarrow ArccalFelfelé^2$$

és így tovább. Az látható, hogy igen ügyesnek kell lennünk egy teljes és helyes tudásbázis felépítéséhez, hogy mindent nyomon kövessünk a wumpus világban; ennek teljes tárgyalását elhalasztjuk a 10. fejezetig. A lényeg, amire itt rá akartunk mutatni, hogy a kezdeti tudásbázis tartalmazni fog olyan mondatokat, mint az előző két példa, minden *t* időpontra és minden helyre is. Ami azt jelenti, hogy minden *t* időpontra és [*x*,*y*] helyre a tudásbázis tartalmaz egy következő formájú mondatot:

[13] A figyelmes olvasó észreveszi, hogy ez megengedte számunkra azt, hogy ügyeskedjünk a nyers érzet, mint a Szellő, és a helyspecifikus állítás, mint az $S_{1,1}$, közötti kapcsolattal.

Created by XMLmind XSL-FO Converter.

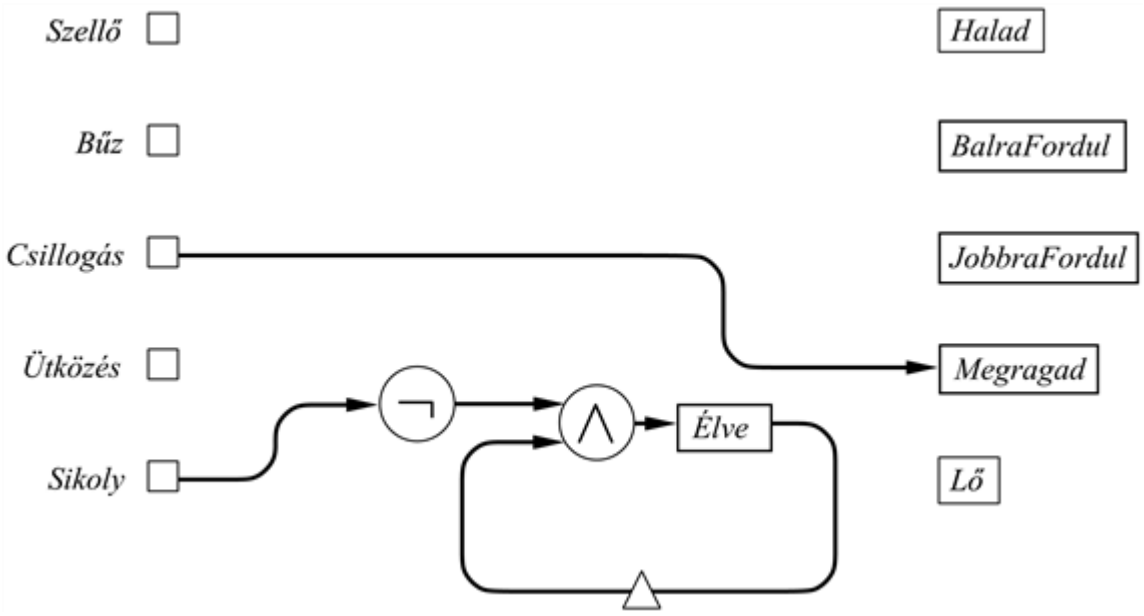
$$H^t_{x,y} \wedge ArccalJobbra^t \wedge Haladt \Rightarrow H^{t+1}_{x+1,y} \qquad (7.3)$$

Még akkor is, ha felső korlátot szabunk a megengedett lépések számának – 100-at talán – mondatok tízezreihez jutunk. Ugyanez a probléma merül fel, ha minden időlépéshez is hozzáadunk mondatokat, „ahogy szükséges”. A klózoknak ez az elburjánzása olvashatatlanná teszi számunkra a tudásbázist, de a gyors ítéletkalkulus megoldók még így is könnyen tudják kezelni a 4 × 4-es wumpus világot (korlátaikat a 100 × 100-as méret körül érik el). Az áramkörre alapuló ágens, amit a következő alfejezet mutat be, részleges megoldást kínál a klózok elburjánzásának problémájára, de a teljes megoldásra várnunk kell, míg nem alkalmazzuk az elsőrendű logikát majd a 8. fejezetben.

6.3. Az áramkörön alapuló ágens

Az **áramkörön alapuló ágens** (**circuit-based agent**) a reflexív ágens egy állapottal rendelkező speciális fajtája, mint ahogy ezt a 2. fejezetben definiáltuk. Az érzetek egy **sorrendi áramkör** (**sequential circuit**) bemenetei, amely sorrendi áramkör egy **kapu**kból (**gate**s) álló hálózatból, amelyek mindegyike egy logikai kapcsolatot valósít meg, és **regiszter**ekből (**register**s) épül fel, amelyek mindegyike az egyes állítások igazságértékeit tárolja. Az áramkör kimenetei olyan regiszterek, melyek cselekvéseknek felelnek meg – például a *Megragad* kimenet *igaz*-ra van állítva, ha az ágens meg akar ragadni valamit. Ha a *Csillogás* bemenetet közvetlenül összekötjük a *Megragad* kimenettel, akkor az ágens meg fogja ragadni a célt bármikor, ha látja (lásd 7.20. ábra).

7.20. ábra - Egy áramköralapú, wumpusvilág-beli ágens egy része, a bemeneteket és a kimeneteket mutatva, az arany megragadását megvalósító áramkör, és az áramkör, amely meghatározza, hogy a wumpus él-e. A regisztereket négyzet jelöli, és a késleltetőket kis háromszögek mutatják.



Az áramköröket az **adatfolyam**ok (**dataflow**) mintájára értékeljük ki, minden időlépésben beállítjuk a bemenetet, és a jeleket végigterjesztjük a hálózaton. Ha egy kapunak minden bemenete rendelkezésre áll, létrehoz egy kimenetet. Ez a folyamat közeli kapcsolatban van az előrefelé láncolással és az ÉS-VAGY gráfokkal, mint amilyen a 7.15. (b) ábrán látható.

Az előző szakaszban azt állítottuk, hogy az áramkörön alapuló ágens az időt kielégítőbb módon kezeli, mint az ítéletkalkulus következtetés alapú ágens. Ez azért lehet így, mert egy regiszterben tárolt változó értéke adja meg a megfelelő ítéletkalkulus szimbólumnak az igazságértékét a *pillanatnyi t időpontban,* ahelyett hogy másolatok léteznének minden egyes időlépéshez. Például lehetne egy *Élve* regiszterünk, amelynek *igaz* értéket kell tartalmaznia, ha a wumpus életben van, és *hamis*-at, ha halott. Ez a regiszter az $Élve^t$ ítéletkalkulus szimbólumnak felel meg, így minden egyes időpillanatban egy különböző ítéletállításra vonatkozik. Az ágens belső állapotai – például a memóriája – megőrizhetőek a regiszterek kimenetének egy **késleltető sor**on (**delay line**) keresztül történő viszszacsatolásával. Ez a regiszter az *előző* lépéshez tartozó értéket szolgáltatja. A 7.20.

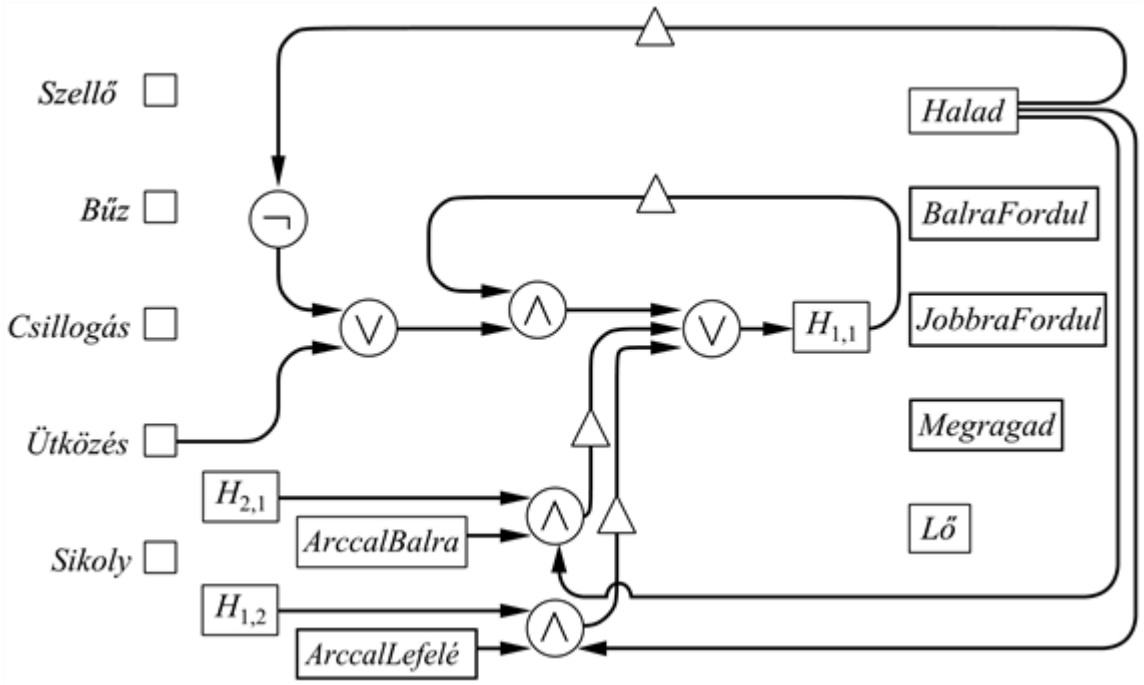
Created by XMLmind XSL-FO Converter.

ábra egy példát mutat. Az *Élve* értékét a *Sikoly* negáltjának és az *Élve* késleltetett értékének a konjunkciója adja. Állításokban megfogalmazva ez a hálózat az *Élve* fogalomra megvalósítja a következő ekvivalenciát:

$$Élve^t \Leftrightarrow \neg Sikoly^t \wedge Élve^{t-1} \quad (7.4)$$

amely mondat azt mondja ki, hogy a wumpus a t időpontban akkor és csakis akkor él, ha nem észleltünk sikolyt a t időpillanatban (egy $t - 1$ időpontbeli sikoly alapján) *és* élt a $t - 1$ időpillanatban. Feltételezzük, hogy az áramkörben az *Élve* regiszter *igazra* van inicializálva. Így az *Élve* igaz marad mindaddig, amíg nincs egy sikoly, amikor is hamissá válik és a későbbiekben hamis marad. Ez pont az, amit mi akarunk.

7.21. ábra - Egy áramkör, amely meghatározza, hogy az ágens az [1, 1]-ben van-e. Minden pozíció és orientáció regiszterhez egy hasonló áramkör csatlakozik.



Az ágens helyzete nagyon hasonló módon kezelhető, mint a wumpus halála. Szükségünk van egy $H_{x,y}$ regiszterre minden x-hez és y-hoz, és ennek az értéke *igaz* kell, hogy legyen, amikor az ágens az [x, y]-ban van. Az áramkör, amelyik beállítja Hxy értékét azonban sokkal bonyolultabb, mint az *Élve* áramköre. Például az ágens az [1, 1]-ben van a t időpontban, ha (a) az ott volt a $t - 1$-ben és vagy nem haladt semerre, vagy megpróbálta, de falnak ütközött; vagy (b) az [1, 2]-ben volt arccal lefelé és haladt előrefelé; vagy (c) a [2, 1]-ben volt arccal balra és haladt előre:

$$\begin{aligned} H_{1,1}^t \Leftrightarrow \quad & (H_{1,1}^{t-1}(\neg Halad^{t-1} \vee Fa\ln akÜtközik^t)) \\ & \vee (H_{1,2}^{t-1} \wedge (ArccalLefelé^{t-1} \wedge Halad^{t-1})) \\ & \vee (H_{2,1}^{t-1} \wedge (ArccalBalra^{t-1} \wedge Halad^{t-1})) \end{aligned} \quad (7.5)$$

A $H_{1,1}$ hálózata a 7.21. ábrán látható. Minden elhelyezkedés regiszterhez egy hasonló áramkör tartozik. A 7.13. (b) feladatban azt kérjük, hogy készítsen áramkört az irányra vonatkozó állításokhoz.

A 7.20. és 7.21. ábrákon levő áramkörök fenntartják az *Élve* és a Hxy igazságértékét minden időpillanatban. Egy ilyen áramkör által reprezentált állítások szokatlanok azonban abban az értelemben, hogy *a helyes igazságértékeiket minden pillanatban meg tudjuk határozni*. Tekintsük az előző helyett az $S_{4,4}$ állítást: a [4, 4] négyzet szellős. Habár az állításnak az igazságértéke rögzítve van valamilyen értéken, az ágens nem ismerheti az igazságértéket mindaddig, amíg meg nem látogatja a [4, 4] mezőt (vagy ki nem következteti, hogy van egy szomszédos csapda). Az ítéletlogikát és az elsőrendű logikát igaz, hamis és ismeretlen értékű állítások automatikus reprezentálására tervezték, de az áramköröket nem: az $S_{4,4}$ regiszternek tartalmaznia kell *valamely* értéket, vagy *igaz* vagy *hamis* értéket, még mielőtt az igazságot feltárják. A regiszterbeli érték lehet helytelen, és ez helytelen irányba is vezetheti az ágenst. Más

Created by XMLmind XSL-FO Converter.

szavakkal, szükségünk volna háromféle lehetséges állapot reprezentálására ($S_{4,4}$ ismerten igaz, ismerten hamis vagy nem ismert), és csak egy bitünk van arra, hogy megoldjuk ezt.

A megoldás erre a problémára az, hogy használjunk két bitet egy helyett. Az $S_{4,4}$-et reprezentálja két regiszter, amelyeket $K(S_{4,4})$-nek és $K(\neg S_{4,4})$-nek nevezünk, ahol K jelöli azt, hogy „ismert”. (Emlékezzünk, hogy ezek még mindig csak összetett nevekkel rendelkező szimbólumok, még akkor is, ha strukturált kifejezésnek tűnnek!) Ha mind $K(S_{4,4})$ és $K(\neg S_{4,4})$ hamis, akkor ez azt jelenti, hogy az $S_{4,4}$ igazságértéke nem ismert. (Ha mindkettő igaz, akkor hiba van a tudásbázisban!) Ebben a reprezentációban bármikor is használnánk az $S_{4,4}$-et a hálózat valamely részében, a $K(S_{4,4})$-et használjuk helyette; és bármikor használnánk az $\neg S_{4,4}$-et, akkor a $K(\neg S_{4,4})$-et használjuk. Általánosságban, minden potenciális határozatlan állítást két **tudásállítás**sal (**knowledge propositions**) reprezentálunk, kijelentve, hogy a vonatkozó állítás ismerten igaz vagy ismerten hamis.

Hamarosan látni fogunk egy példát arra, hogyan kell alkalmaznunk a tudásállításokat. Először is, szükségünk van annak kidolgozására, hogy hogyan határozzuk meg maguknak a tudásállításoknak az igazságértékét. Vegyük észre, hogy miközben az $S_{4,4}$-nek rögzített igazságértéke volt, a $K(S_{4,4})$ és $K(\neg S_{4,4})$ *változik,* miközben az ágens egyre többet tud meg a világról. Például a $K(S_{4,4})$ hamis értékről indul, és igazzá válik, amint az $S_{4,4}$-ről meghatározható, hogy igaz – azaz, amikor az ágens a [4, 4]-be lép, és detektálja a szellőt. Majd igaz marad ezután. Így megállapítható, hogy

$$K(S_{4,4})^t \Leftrightarrow K(S_{4,4})^{t-1} \vee (H_{4,4}^t \wedge Szellő^t) \qquad (7.6)$$

Hasonló egyenlet írható fel $K(\neg S_{4,4})^t$-re is.

Most, hogy az ágensnek van már ismerete a szellős négyzetekről, foglalkozhat a csapdákkal is. Egy csapda hiánya az adott négyzetben akkor és csakis akkor határozható meg, ha valamelyik szomszédos négyzetéről ismert, hogy nem szellős. Például ha létezik

$$K(\neg C_{4,4})^t \Leftrightarrow K(\neg S_{3,4})^t \vee K(\neg S_{4,3})^t \quad (7.7)$$

Egy csapda adott négyzetben való *jelenlétének* a meghatározása sokkal nehezebb – szellőnek kell lennie egy szomszédos négyzetben, de olyan szellőnek, amely nem egy másik csapda miatt keletkezett:

$$K(C_{4,4})^t \Leftrightarrow (K(S_{3,4})^t \wedge K(\neg C_{2,4})^t \wedge K(\neg C_{3,3})^t) \quad (7.8)$$

$$\vee \ (K(S_{4,3})^t \wedge K(\neg C_{4,2})^t \wedge K(\neg C_{3,3})^t)$$

Fontos

Miközben egy kicsit zavaros a csapdák jelenlétének és hiányának bizonyítása az áramkörök használata esetében, az áramkör *konstans számú kaput használ minden egyes négyzethez*. Ez a tulajdonság alapvetően fontos ahhoz, hogy olyan áramkörön alapuló ágenst építsünk, amelynek bonyolultsága kezelhető módon nő a probléma méretével. Ez valójában magának a wumpus világnak a jellemzője; azt mondjuk, hogy egy környezet **lokalitás** (**locality**) tulajdonságot mutat fel, ha igaz, hogy bármelyik vizsgálandó állítás igazságának meghatározásához csak egy konstans számú másik állítás megvizsgálása szükséges. A lokalitás tulajdonság nagyon érzékeny a környezet pontos „fizikájára”. Például az aknakereső tárgyterület (7.11. feladat) nemlokális, mert az akna egy adott négyzetbeli létezésének eldöntése igényelheti, hogy akármilyen távol levő négyzetet is megnézzünk. Nemlokális tárgyterületekre az áramkörre alapuló ágens alkalmazás nem mindig praktikus.

Van egy másik probléma is, amely mellett eddig óvatosan elmentünk: a **hurokmentesség** (**acyclicity**) kérdése. Az áramkört hurokmentesnek tekintjük, ha minden olyan ág, amely egy regiszter kimenetét visszacsatolja a bemenetére, tartalmaz legalább egy közbülső késleltető elemet. Megköveteljük, hogy minden áramkör hurokmentes legyen, mivel a hurkokat tartalmazó áramkörök, mint fizikai eszközök, nem működnek! Az ilyen áramkör instabil oszcillációba kerülhet határozatlan értékeket eredményezve. Példaként a hurkot tartalmazó áramkörökre, tekintsük a (7.6) egyenletnek a következő kibővítését:

$$K(S_{4,4})^t \Leftrightarrow K(S_{4,4})^{t-1} \vee (H_{4,4}^t \wedge Szellő^t) \vee K(C_{3,4})^t \vee K(C_{4,3})^t \quad (7.9)$$

A hozzáadott diszjunktok, a $K(C_{3,4})^t$ és a $K(C_{4,3})^t$ lehetővé teszik az ágens számára, hogy a szellő jelenlétét meghatározza a szomszédos négyzetekben levő csapdák jelenlétéből, ami teljesen ésszerűnek tűnik. Így viszont sajnos a szellő tulajdonság függ a szomszédos csapdáktól, és a csapdák függenek a szomszédos négyzetek szellő tulajdonságától, ahogy azt a (7.8) egyenlet leírja. Ezáltal a teljes áramkör tartalmazna hurkot.

Created by XMLmind XSL-FO Converter.

A nehézség nem az, hogy a (7.9) egyenlet *nem helyes*. A probléma inkább az, hogy az egyenletek által reprezentált függőségek összekapcsolása nem oldható fel az igazságértékek propagálásának egyszerű mechanizmusával a vizsgált logikai áramkörökben. A (7.6) egyenletet alkalmazó hurokmentes verzió, amely a szellősséget a közvetlen megfigyelésekből határozza meg, *nem teljes* abban az értelemben, hogy bizonyos pontokon az áramköralapú ágens kevesebbet tud, mint a teljes következtetési eljárást alkalmazó következtetésalapú ágens. Például ha van szellő az [1, 1]-ben, a következtetésalapú ágens képes arra a megállapításra jutni, hogy van egy szellő a [2, 2]-ben, míg a (7.6) egyenletet használó, hurokmentes áramkörre alapuló ágens erre nem képes. Lehetséges építeni egy teljes áramkört – végül is, szekvenciális áramkörök képesek emulálni bármilyen digitális számítógépet –, de ez jelentősen bonyolultabb volna.

6.4. Összehasonlítás

A következtetésalapú és az áramkörön alapuló ágensek a deklaratív és procedurális megközelítés szélsőséges esetei az ágens tervezésében. Számos szempont szerint összehasonlíthatjuk őket:

- *Tömörség.* Az áramkörön alapuló ágensnek, ellentétben a következtetésalapú ágenssel, nincs szüksége a „tudásának” önálló másolataira minden időlépésben. Ehelyett, mindig csak a pillanatnyi és a megelőző időlépésre hivatkozik. A „fizika” leírásához mindkét ágensnek szüksége van minden egyes négyzetre vonatkozó másolatokra (mondatok vagy áramkörök formájában), és így nem könnyű nagyobb környezeteket leírni. Azokban a környezetekben, ahol számos objektum összetett kapcsolatait kell leírni, a kijelentések száma eláraszt bármilyen ítéletkalkulus ágenst. Ezek a környezetek már igénylik az elsőrendű logika kifejezőerejét (lásd 8. fejezet). Mindkét típusú ítéletkalkulus ágens rosszul alkalmazható egy biztonságos négyzettel szomszédos helyre vezető útvonal megtalálásának kifejezésére vagy megoldására. (Emiatt az `IK-WUMPUS-ÁGENS` keresési algoritmusokat alkalmaz.)

- *Számítási hatékonyság. A legrosszabb esetben a következtetés időigénye exponenciálisan függ a szimbólumok számától,* míg az áramkör futtatása az áramkör méretétől lineárisan függő időt igényel (vagy a *mélységtől* lineárisan függő időt, ha fizikai eszközként megvalósítjuk). Láthattuk azonban, hogy a *gyakorlatban* a DPLL igen gyorsan elvégezte a szükséges következtetéseket.[14]

- *Teljesség.* Azt sugalltuk korábban, hogy az áramköralapú ágens lehet, hogy nem teljes a ciklusmentességgel kapcsolatos korlátozás miatt. A teljesség hiányának okai azonban még alapvetőbbek. Először is, emlékezzünk, hogy az áramkör a méretével lineárisan arányos időben oldja meg a feladatot. Ez azt jelenti, hogy bizonyos környezetek esetében egy teljes áramkörnek (egy olyannak, amely minden meghatározható állításhoz kiszámítja az igazságértéket) exponenciálisan nagyobbnak kell lennie, mint a következtetésalapú ágens tudásbázisának. Egyébként lenne módszerünk arra, hogy hogyan oldjuk meg az ítéletkalkulus vonzat problémáját kisebb mint exponenciális idő alatt, ami nagyon valószínűtlen. A második érv az ágens belső állapotainak a jellege. A következtetésalapú ágens emlékezik minden érzetre, és implicit vagy explicit módon ismer minden mondatot, ami az érzetekből és a kezdeti adatbázisból következik. Például ha adott $S_{1,1}$ és ismeri a $C_{1,2} \vee C_{2,1}$ állítást, abból $S_{2,2}$ következik. Ezzel szemben az áramköralapú ágens elfelejt minden korábbi érzetet és csak a regiszterekben tárolt egyedi állításokra emlékszik. Ekképpen az első érzet után a $C_{1,2}$ vagy a $C_{2,1}$ *önmagában* ismeretlen marad, és így semmilyen következtetés nem vonható le $S_{2,2}$-re vonatkozóan.

- *A létrehozás egyszerűsége.* Ez egy nagyon fontos kérdés, amelyet nehéz pontosan megítélni. A könyv szerzője számára a „fizika” leírása bizonyára sokkal egyszerűbb deklaratív úton, miközben kisméretű, hurokmentes, majdnem teljes áramkörök tervezése a csapdák megtalálására igen nehéznek bizonyult.

Összefoglalva, úgy tűnik, hogy létezik kompromisszum a számítási hatékonyság, a tömörség, a teljesség és a könnyű létrehozás között. Ha az érzetek és a cselekvések közötti kapcsolat egyszerű – mint a *Ragyogás* vagy a *Megragad* közötti kapcsolat esetében – az áramkör optimálisnak tűnik. Ennél bonyolultabb kapcsolatok esetében a deklaratív megközelítés jobb lehet. Egy olyan területen, mint a sakk például, a deklaratív szabályok tömörek és könnyen kódolhatók (legalábbis az elsőrendű logikában), de a tábla állapotaiból a lépéseket közvetlenül számító hálózat elképzelhetetlenül hatalmas lenne.

Eltérő dolgokat látunk ezekkel a megközelítésekkel kapcsolatban az állatok világában. Az egyszerű idegrendszerrel rendelkező kisebb állatok valószínűleg hálózatalapúak, míg a magasabb rendű állatok úgy tűnik, hogy explicit reprezentációkon következtetéseket végeznek. Ez lényegesen bonyolultabb ágensfunkciók számítását teszi lehetővé számukra. Az emberek szintén rendelkeznek áramkörökkel, amelyek a reflexeket

[14] Valójában minden olyan következtetés, amely megvalósítható egy áramkörrel, megoldható lineáris időben a DPLL-lel. Ez azért van így, mert egy áramkör kiértékelése hasonlít az előrefelé láncoláshoz, amely emulálható DPLL-lel az egység propagáció szabály alkalmazásával.

Created by XMLmind XSL-FO Converter.

valósítják meg, és talán képesek **lefordít**ani (**compile**) deklaratív ismereteket áramkörökké, amikor bizonyos következtetések rutinná válnak. Ily módon egy **hibrid ágens** (**hybrid agent**) tervezése (lásd 2. fejezet) lehet a legjobb mindkét világ számára.

7. Összefoglalás

Bevezettük a tudásbázisú ágens ötletét, és megmutattuk, hogy hogyan tudunk olyan logikát definiálni, amellyel az ágens képes következtetni a világról. A legfontosabb pontok a következők:

- Az intelligens ágensnek szüksége van tudásra a világról, hogy jó döntéseket hozhasson.
- A tudást az ágens egy **tudásbázis**ban (**knowledge base**) egy **tudásreprezentációs nyelv** (**knowledge represantion language**) **mondat**ainak (**sentence**s) formájában tárolja.
- Egy tudásbázisú ágenst a tudásbázis és a következtetési mechanizmus alkotja. Működésének lényege, hogy a tudásbázisában a világot leíró mondatokat tárol és következtetési mechanizmust alkalmaz új mondatok következtetésére, majd felhasználja ezeket cselekvések meghatározására.
- Egy reprezentációs nyelvet **szintaxis**a (**syntax**) és **szemantiká**ja (**semantics**) definiál. A szintaxis a mondatok struktúráját határozza meg, a szemantika a mondatok **igazság**át (**truth**) határozza meg minden lehetséges **világ**ban (**world**) vagy **modell**ben (**model**),
- A mondatok közötti **vonzat** (**entailment**) kapcsolatnak alapvetően fontos szerepe van a következtetés megértésében. Egy α mondat maga után vonz egy másik β mondatot, ha β igaz minden világban, ahol α igaz. Ezzel ekvivalens definíciók az $\alpha \Rightarrow \beta$ mondat **érvényesség**ét (**validity**) és a $\alpha \wedge \neg\beta$ mondat **kielégíthetetlenség**ét (**unsatisfiability**) meghatározó definíciók.
- A következtetés az a folyamat, amivel új mondatok vezethetők le régiekből. A **helyes** (**sound**) következtetési algoritmusok *csak* vonzat mondatokat vezetnek le; a **teljes** (**complete**) algoritmus az *összes* következményt levezeti.
- Az **ítéletkalkulus** (**propositional logic**) egy nagyon egyszerű nyelv, amely **ítélet-szimbólumok**ból (**proposition symbols**) és **logikai összekötőjel**ekből (**logical connective**s) áll. Képes kezelni olyan állításokat, amelyekről tudjuk, hogy igazak, tudjuk, hogy hamisak, vagy teljesen ismeretlen az igazságértéküket.
- Ha adott egy ítéletkalkulus szimbólumszótár, akkor a lehetséges modellek száma véges, így a vonzatok ellenőrzése történhet a modellek felsorolásával is. Hatékony ítéletkalkulus **modellellenőrző** (**model checking**) algoritmusok többek között viszszalépéses vagy lokális keresési eljárásokat alkalmaznak, amelyekkel gyakran nagyméretű problémákat is nagyon gyorsan meg tudnak oldani.
- A **következtetési szabály**ok (**inference rules**) helyes következtetési minták, amelyeket felhasználhatunk bizonyítások megtalálásához. A **rezolúció** (**resolution**) szabály teljes következtetési algoritmust biztosít olyan tudásbázisokhoz, amelyek **konjunktív normál formá**ban (**conjunctive normal form**) vannak kifejezve. Az **előrefelé láncolás** (**forward chaining**) és a **hátrafelé láncolás** (**backward chaining**) igen természetes érvelési algoritmusok **Horn-formá**ban (**Horn form**) adott tudásbázisokon.
- Kétfajta ágenst lehet építeni az ítéletkalkulus alkalmazására: a **következtetésalapú ágens** (**inference-based agent**) következtetési algoritmusokat használ a világ eseményeinek követésére, és képes rejtett jellemzőket is levezetni, míg az **áramkörön alapuló ágens** (**circuit-based agent**) az állításokat regiszterek bitjeiként reprezentálja, és jelek logikai áramkörökben történő terjesztésével végzi ezek frissítését.
- Az ítéletkalkulus meglehetősen hatékony bizonyos feladatokra egy ágensben alkalmazva, de kezelhetetlen nemkorlátos méretű környezetekben, mivel hiányzik a megfelelő kifejező erő az idő, a tér és az objektumok közötti kapcsolatok általános mintáinak leírására.

7.1. Irodalmi és történeti megjegyzések

John McCarthy cikke a „Programs with Common Sense” (McCarthy, 1958, 1968) tette híressé az ágens fogalmát, egy olyan programét, amely az érzetek és cselekvések összekapcsolására logikai következtetést használ. A cikk kitűzte a deklarativizmus zászlóját, megmutatva, hogy szoftverek írásának igen elegáns

Created by XMLmind XSL-FO Converter.

módszere az, ha a szükséges ismereteket közöljük az ágenssel. Allen Newell cikke (1982) a „The Knowledge Level" tárgyalja azt a megközelítést, hogy racionális ágensek leírhatók és elemezhetők egy absztrakt szinten, ami az általuk birtokolt tudást és nem azt a programot definiálja, amit futtatnak. A mesterséges intelligencia deklaratív és procedurális megközelítéseit hasonlítja össze Boden (Boden, 1977). A vitát, mások mellett Brooks (Brooks, 1991) és Nilsson (Nilsson, 1991) írásai élesztették fel újra.

Magának a logikának az eredete az ősi görög filozófiában és matematikában található. Számos logikai alapelv – a mondatok szintaktikus struktúrájának összekötése az igaz vagy hamis jellegükkel, a jelentésükkel, a bennük megjelenő argumentumok érvényességével – elszórt helyeken megtalálható Platón műveiben. Arisztotelész készítette az első ismert, rendszerezett munkát a logikáról. Munkáját diákjai gyűjtötték össze halála után, i. e. 322-ben az *Organon* c. tanulmányban. Arisztotelész **szillogizmus**ai (**syllogism**s) olyan logikai állítások voltak, amelyeket ma következtetési szabályoknak neveznénk. Habár a szillogizmus tartalmazott elemeket, mind a propozíciós, mind az elsőrendű logikából, a rendszer, mint egész, igen gyengének számít a modern kívánalmak szerint. Nem tette lehetővé tetszőleges komplexitású mondatok létrehozását a következtetési mintákban, mint a modern ítéletlogika.

A hasonló Megara és sztoikus iskolák (az i. e. 5. században indultak, és több száz éven keresztül működtek) vezették be az implikációt és más alapvető szerkezeteket, amelyeket ma is használunk a modern ítéletlogikában. Logikai összekötőjelek definiálására vezette be az igazságtábla használatát Philón és Megara. A sztoikusok öt alapvető következtetési szabályt használtak, amelyeket igazolás nélkül érvényesnek tartottak, köztük azt a szabályt, amelyet ma Modus Ponensnek nevezzünk. Számos szabályt vezettek le ebből az ötből, felhasználva többek között a dedukció elméletének alapelvét 4. szakasz - Az ítéletkalkulus következtetési mintái részben, és sokkal tisztábban használták a bizonyítás fogalmát, mint azt Arisztotelész tette. A sztoikusok azt állították, hogy az ő logikájuk teljes abban az értelemben, hogy tartalmaz minden érvényes következtetést, de ami fennmaradt munkájukból, az túlságosan töredékes ahhoz, hogy elemezhető legyen. A Megara és a sztoikus logikák történetének, már amennyire ezek ismertek, jó beszámolóját készítette el Benson Mates (Mates, 1953).

Wilhelm Leibniztől (1646–1716) származik a mesterséges formális nyelvi minták létrehozásának ötlete a logikai kapcsolatok tisztázásának és a logikai következtetés egy tisztán formális és mechanikus folyamattá való egyszerűsítése céljából. Leibniz saját matematikai logikája azonban igencsak tökéletlen volt, és rá inkább ezeknek a gondolatoknak mint célkitűzéseknek a bevezetéséért és nem e célok megvalósítására tett kísérletei miatt emlékezünk.

George Boole vezette be első igazán átfogó és működőképes formális logikai alapú rendszert a *The Mathematical Analysis of Logic* c. könyvében (Boole, 1847). Boole logikájának modellje közel állt a valós számok algebrájához, és elsődleges következtetési módszerként a logikailag ekvivalens mondatok helyettesítését használta. Bár Boole rendszere csak töredéke volt a teljes ítéletkalkulusnak, elég közel volt ahhoz, hogy a 19. század szerzői Boole-t követve hamar kitöltsék a hiányzó részeket. Schröder definiálta (Schröder, 1877) a konjunktív normál formát, míg a Horn-formát sokkal később Alfred Horn (Horn, 1951) vezette be. A modern ítéletkalkulus (és elsőrendű logika) első átfogó bemutatása Gottlob Frege *Begriffschrift* (Az írás fogalma vagy Fogalmi jelölés) c. könyvében (Frege, 1879) található.

Az első mechanikai eszközt, amely logikai következtetést végzett, Stanhope harmadik grófja (1753–1816) készítette. A Stanhope Demonstrator képes volt szillogizmusokat kezelni és bizonyos valószínűségi következtetéseket végezni. Wiliam Stanley Jevons – egyike azoknak, akik továbbfejlesztették és kiegészítették Boole munkáját – 1869-ben a Boole-logikán alapuló következtetések végrehajtására megépítette a „logical pianó"-ját. Ezeknek és más korai következtetésre készített mechanikus eszközöknek szórakoztató és tanulságos története Martin Gardner (Gardner, 1968) munkájában olvasható. Az első logikai következtetést végző, publikált számítógépes program a „Logic Theorist" volt, amelyet Newell, Shaw és Simon (Newell, Shaw és Simon, 1957) készítettek. Ezt a programot az emberi gondolkodás modellezésére szánták a szerzők. Bár Martin Davis (Davis, 1957) tervezett egy programot, ami bizonyíthatóan 1954-ben készült, de a Logic Theorist eredményeit korábban publikálták. Mind Davis 1954-es programja, mind a Logic Theorist részben *ad hoc* módszerekre épült, amelyek nem voltak jelentős hatással a későbbi automatikus dedukciós rendszerekre.

Az igazságtábláknak, mint az ítéletkalkulus nyelvében az érvényességnek vagy a mondat kielégíthetetlenségének a tesztjét egymástól függetlenül Ludwig Wittgenstein (1922) és Emil Post (1921) vezették be. A harmincas években jelentős haladást értek el az elsőrendű logika következtetési módszereinek terén. Nevezetesen, Gödel megmutatta (Gödel, 1930), hogy az elsőrendű logikában történő következtetésre egy teljes eljárást lehet kapni az ítéletlogikára történő redukcióval, felhasználva Herbrand elméletét (Herbrand, 1930). A 9. fejezetben újra áttekintjük ennek történetét, itt most a lényeges pont az, hogy a hatvanas években, a hatékony ítéletkalkulus algoritmusok fejlesztésében a matematikusok érdeklődését jelentős mértékben motiválta

Created by XMLmind XSL-FO Converter.

az a cél, hogy hatékony tételbizonyítót készítsenek az elsőrendű logika számára. A Davis–Putnam-algoritmus (Davis és Putnam, 1960) volt az első hatásos algoritmus az ítéletkalkulus rezolúció megvalósítására, de a legtöbb esetben ez is sokkal kevésbé hatékony, mint a DPLL visszalépéses algoritmus, amelyet két évvel később mutattak be (1962). A teljes rezolúciós szabály és a rezolúció teljességének bizonyítása J. A. Robinson (Robinson, 1965) nagyhatású tanulmányában jelent meg, ami azt is megmutatta, hogyan lehet elsőrendű logikai következtetést végezni ítéletkalkulus technikák igénybevétele nélkül.

Stephen Cook (Cook, 1971) mutatta meg, hogy a kielégíthetőség eldöntése az ítéletlogikában NP-teljes. Mivel a vonzat meghatározása ekvivalens a kielégíthetetlenség eldöntésével, ez is NP-teljes. Az ítéletlogika számos részhalmaza ismert, amelyről tudjuk, hogy bennük a kielégíthetőség problémája polinomiális időben megoldható. A Horn-klózok az egyik ilyen rézhalmaz. A Horn-klózokon működő, lineáris időben futó előreláncolási algoritmust Dowlingnak és Galliernek (Dowling és Gallier, 1984) köszönhetjük, akik az algoritmusokat egy adatfolyam-eljárásként írták le, hasonlóan a jelek terjedéséhez az áramkörökben. A kielégíthetőség ellenőrzése alapvető módszerré vált a problémák NP-teljességének vizsgálatában; például Kaye (Kaye, 2000), megmutatta, hogy az Aknakereső játék (lásd 7.11. feladat) NP-teljes.

Számos szerző próbálkozott lokális keresési algoritmusokat használni a kielégíthetőség eldöntésére a nyolcvanas években. Minden ilyen algoritmus a kielégíthetetlen klózok számának minimalizálásának elvén (Hansen és Jaumard, 1990) alapult. Különlegesen hatékony algoritmust fejlesztett ki Gu (Gu, 1989), és tőle függetlenül Selman és társai (Selman és társai, 1992), amelyet ez utóbbi szerzők GSAT-nak neveztek el, és megmutatták, hogy az algoritmus képes nehéz problémák széles körét igen gyorsan megoldani. A `WALKSAT` algoritmust, amelyet ebben a fejezetben bemutattunk, szintén Selma és társai publikálták (1996).

A „fázisátmenet” létezését a véletlen *k*-SAT problémák kielégíthetőségében először Simon és Dubois (Simon és Dubois, 1989) figyelték meg. Crawford és Auton tapasztalati eredményei (Crawford és Auton, 1993) azt sejtették, hogy nagyméretű, véletlenszerű 3-SAT problémák esetén ez az átmenet a 4,24-es klóz/változó arány környékén van. Cikkük szintén bemutat egy hatékony DPLL megvalósítást. Bayardo és Schrag leírt egy másik, kényszer-kielégítési technikákat alkalmazó hatékony DPLL megvalósítást (Bayardo és Schrag, 1997). Moskewicz és társai publikálták a `CHAFF` algoritmust (Moskewicz és társai, 2001), amely millió változós hardververifikációs problémákat képes megoldani, és amely megnyerte a SAT 2002 versenyt. Li és Anbulagan gyors problémamegoldókat lehetővé tevő egységpropagáción alapuló heurisztikákat tárgyalt (Li és Anbulagan, 1997). Cheeseman és társai (1991) számos hasonló problémáról adtak adatokat, és megfogalmazták azt a feltevést, hogy minden NP-teljes problémának van állapotátmenete (Cheeseman és társai, 1991). Kirkpatrick és Selman módszereket mutattak arra (Kirkpatrick és Selman, 1994), hogy a statisztikus fizika technikáit hogyan lehet alkalmazni a fázisátmenetek pontos „alakjának” jobb megértéséhez. Az átmenetek *helyének* meghatározására vonatkozó elméleti vizsgálatok igen gyengék: az egyetlen, amit sikerült bizonyítani, hogy az átmenet a [3,003 4,598] tartományba esik a véletlen 3-SAT problémáknál. Cook és Mitchell kiváló áttekintést készítettek ezekről és számos más kielégíthetőséggel kapcsolatos téma eredményeiről (Cook és Mitchell, 1997).

Korai elméleti kutatások megmutatták, hogy a DPLL átlagos komplexitása polinomiális a problémák egy bizonyos természetes eloszlású köreire. Ez a potenciálisan érdekes tény kevésbé izgalmassá vált, miután Franco és Paull megmutatták, hogy ugyanezek a problémák konstans időben megoldhatók egyszerűen találgatással, ahol véletlen hozzárendelésekkel dolgozunk (Franco és Paull, 1983). A véletlen generáló módszer, amelyet a fejezetben bemutattunk, sokkal nehezebb problémákat hoz létre. Az ilyen, a problémákon lokális kereséssel elért tapasztalati sikerek által motiválva, Koutsoupias és Papadimitriou megmutatta, hogy egy egyszerű hegymászó algoritmussal nagyon gyorsan megoldható *szinte minden* kielégíthetőségi probléma példány (Koutsoupias és Papadimitriou, 1992), ami azt sugallja, hogy a nehéz problémák ritkák. Mi több, Schöning bejelentett (Schönig, 1999) egy véletlen elemeket tartalmazó GSAT változatot, amelynek a *legrosszabb esetre* számított várható futási ideje 3-SAT problémákon $1{,}333^n$ – ami még mindig exponenciális, de lényegesen gyorsabb, mint a korábbi legrosszabb esetkorlátok. A kielégíthetőségi algoritmusok ma is igen aktív területét képezik a kutatásoknak; Du és társai cikkgyűjteménye (Du és társai, 1999) jó kiindulási pontot jelent.

Az áramköralapú ágensek gondolata visszavezethető McCulloch és Pitts nagyhatású cikkére (McCulloch és Pitts, 1943), amely a neurális hálózatok témakör kezdetét jelentette. Népszerű vélekedésekkel szemben a cikk valójában logikai áramköralapú ágensterveknek az agyban történő megvalósításával foglalkozott. Azonban az áramköralapú ágens kis figyelmet kapott a mesterséges intelligenciában. A leginkább megjegyzendő kivétel Stan Rosenschein munkája (Rosenschein, 1985; Kaelbling és Rosenschein, 1990), aki módszert dolgozott ki áramköralapú ágensek lefordítására a feladat környezetének deklaratív leírására. A regiszterekben tárolt állítások frissítésének módszere közeli kapcsolatban van a Reiter által az elsőrendű logikára kifejlesztett **követő állapot axiómá**hoz (**successor-state axiom**) (Reiter, 1991). Rod Brooks munkája (1986, 1989) demonstrálta az áramköralapú megvalósítások hatékonyságát robotok vezérlésére – a 25. fejezetben foglalkozunk ezzel a

Created by XMLmind XSL-FO Converter.

témával. Brooks amellett érvelt (Brooks, 1991), hogy az áramköralapú ágensen kívül más nem is szükséges a mesterséges intelligenciához – mivel más módszerek nehézkesek, drágák és szükségtelenek. A mi nézetünk szerint, egyik módszer sem elégséges önmagában.

A wumpus világot Gregory Yob (Yob, 1975) találta ki. Némi iróniával, Yob azért fejlesztette ki, mert unta azokat a játékokat, amelyeket egy rácson játszanak: az eredeti wumpus világ dodekaéder alakú volt, mi helyeztük vissza a régi unalmas négyzetrácsba. Michael Genesereth javasolta először, hogy a wumpus világot az ágensek tesztkörnyezeteként alkalmazzuk.

7.2. Feladatok

7.1.

Írja le a wumpus világot a 2. fejezetben felsorolt feladatkörnyezetek tulajdonságainak felhasználásával.

7.2.

Tegyük fel, hogy az ágens eljutott a 7.4. (a) ábra szerinti állapotba, úgy, hogy nem érzett semmit az [1, 1]-ben, szellőt érzett az [2, 1]-ben és bűzt az [1, 2]-ben, és most vizsgálja az [1, 3], [2, 2] és [3, 1] négyzetek tartalmát. Ezek bármelyike tartalmazhat csapdát, és legfeljebb az egyik tartalmazhatja a wumpust. A 7.5. ábra példáját követve hozza létre a lehetséges világokat (32 ilyet kell találnia). Jelölje meg azokat a világokat, amelyekben a *TB* igaz, és azokat, amelyekben a következő mondatok igazak:

α_2 = „Nincsen csapda a [2, 2]-ben”

α_3 = „Az [1,3]-ban wumpus van”

Mutassa meg ezenkívül, hogy $TB \models \alpha_2$ és $TB \models \alpha_3$.

7.3.

Tekintsük azt a problémát, hogy hogyan dönthető el egy ítéletkalkulus mondat igazsága egy adott modellben.

a. Írjon egy IK-IGAZ?(*s*, *m*) rekurzív programot, amely akkor és csakis akkor ad vissza *igaz*at, ha az *s* mondat igaz az *m* modellben (ahol *m* minden *s* szimbólumhoz egy igazságértéket rendel). Az algoritmusnak a mondat méretével lineárisan változó időben kell futnia. (Használhatja ennek a függvénynek egy változatát az online kódtárból.)

b. Adjon három példát olyan mondatokra, amelyekről meghatározható, hogy igazak vagy hamisak egy *részleges* modellben, amely nem specifikálja minden szimbólum igazságértékét.

c. Mutassa meg, hogy egy részleges modellben egy mondat igazságértéke (ha van ilyen) általánosságban nem határozható meg hatékonyan.

d. Módosítsa az IK-IGAZ? algoritmust úgy, hogy néha részleges modell alapján is meg tudjon határozni igazságértékeket és közben tartsa meg a függvény igazságértékét és lineáris futási idejét. Adjon meg három példát olyan mondatokra, amelyek igazságértékét a részleges modellben nem állapítja meg az algoritmus.

e. Vizsgálja meg, hogy a módosított algoritmus hatékonyabbá teszi-e az IT-VONZAT? eljárást.

7.4.

Bizonyítsa be a következő állításokat:

a. α akkor és csakis akkor érvényes, ha $Igaz \models \alpha$.

b. Bármilyen α-ra $Hamis \models \alpha$.

c. $\alpha \models \beta$ akkor és csakis akkor, ha az $(\alpha \Rightarrow \beta)$ mondat érvényes.

d. $\alpha \equiv \beta$ akkor és csakis akkor, ha az $(\alpha \Leftrightarrow \beta)$ mondat érvényes.

e. $\alpha \models \beta$ akkor és csakis akkor, ha az $(\alpha \wedge \neg\beta)$ mondat kielégíthetetlen.

Created by XMLmind XSL-FO Converter.

7.5.

Vegyünk egy szótárat, amelyben csak négy ítéletkalkulus állítás létezik, az *A, B, C* és *D*. Hány modellje létezik a következő mondatoknak?

a. $(A \wedge B) \vee (B \wedge C)$

b. $A \vee B$

c. $A \Leftrightarrow B \Leftrightarrow C$

7.6.

Definiáltunk 4 bináris logikai összekötőjelet.

a. Léteznek-e még továbbiak, amelyek hasznosak lehetnek?

b. Hány bináris összekötőjel lehetséges?

c. Miért nem túlságosan hasznos közülük néhány?

7.7.

Egy szabadon választott módszer felhasználásával igazolja a 7.11. ábra ekvivalenciáit.

7.8.

Vizsgálja meg a következő mondatokat, és döntse el mindegyikre, hogy érvényesek, kielégíthetetlenek, vagy egyik sem. Igazolja a döntését igazságtáblával, vagy felhasználva a 7.11. ábra ekvivalencia szabályait. Van-e olyan, amit elsőre eltévesztett?

a. $Füst \Leftrightarrow Füst$

b. $Füst \Rightarrow Tűz$

c. $(Füst \Rightarrow Tűz) \Rightarrow (\neg Füst \Rightarrow \neg Tűz)$

d. $Füst \vee Tűz \vee \neg Tűz$

e. $((Füst \wedge Hőség) \Rightarrow Tűz) \Leftrightarrow ((Füst \Rightarrow Tűz) \vee (Hőség \Rightarrow Tűz))$

f. $(Füst \Rightarrow Tűz) \Rightarrow ((Füst \wedge Hőség) \Rightarrow Tűz)$

g. $Nagy \vee Hallgatag \vee (Nagy \Rightarrow Hallgatag)$

h. $(Nagy \wedge Hallgatag) \vee \leftarrow Hallgatag$

7.9.

(Barwise és Etchemendy, 1993 alapján) Be tudja-e bizonyítani, hogy az unikornis mitikus, ha adottak az alábbiak? Mit mondhatunk a mágikusságról? Van-e szarva?

Ha az unikornis mitikus, akkor halhatatlan, de ha nem halhatatlan, akkor egy halandó emlős. Ha az unikornis vagy halhatatlan, vagy emlős, akkor van szarva. Az unikornis mágikus állat, ha van szarva.

7.10.

Minden logikai mondat logikailag ekvivalens azzal az állítással, hogy minden lehetséges világ, amelyben hamis volna, nem az eset. Felhasználva ezt a megfigyelést, bizonyítsa be, hogy minden mondat átírható konjugált normált formára.

7.11.

Created by XMLmind XSL-FO Converter.

Az ismert számítógépes játék, az Aknakereső igen hasonló a wumpus világhoz. Az aknakereső világ egy N négyzetet tartalmazó négyzetes rács, amelyek közül M elszórtan elhelyezkedő négyzet láthatatlan aknát takar. Minden négyzetet megpróbálhat az ágens, de azonnali halállal lakol, ha ott egy aknát talál. Az Aknakereső játék minden kipróbált négyzetben felfedi, hogy *hány* közvetlenül vagy átlósan szomszédos négyzetben van akna, ezzel mutatva az aknák jelenlétét. A cél az, hogy minden aknát nem tartalmazó négyzetet megpróbáljunk.

a. Legyen $X_{i,j}$ igaz akkor és csakis akkor, ha az $[i, j]$ tartalmaz egy aknát. Írja le az $X_{i,j}$ szimbólum logikai kombinációját tartalmazó mondattal azt a kijelentést, hogy pontosan két akna szomszédos az [1, 1]-gyel.

b. Általánosítsa az (a)-beli kijelentést megmutatva, hogy hogyan hozható létre konjugált normál formájú mondat annak kifejezésére, hogy az n szomszédos négyzetből k négyzet tartalmaz aknát.

c. Magyarázza meg pontosan, hogy hogyan használhat egy ágens DPLL algoritmust annak bizonyítására, hogy egy adott négyzet tartalmaz (vagy nem tartalmaz) aknát, figyelmen kívül hagyva azt a globális korlátot, hogy pontosan M akna van összesen.

d. Tegyük fel, hogy a globális korlát a (b) feladatrészben megfogalmazott módon van létrehozva. Hogyan függ a klózok száma M-től és N-től? Javasoljon egy módosítást a DPLL algoritmushoz, amellyel a globális korlátot nem kell explicit módon reprezentálni.

e. Van-e olyan a (c) feladatrészben definiált eljárás által előállított következmény, amely érvénytelenné válna, ha figyelembe vennénk a globális kényszert?

f. Mutasson példát olyan lehetséges konfigurációkra, amelyek *távoli függőségeket* idéznek elő, olyanokat, ahol egy ki nem próbált négyzet tartalma távoli négyzetek tartalmáról ad információt. [*Segítség:* vizsgáljon meg egy $N \times 1$-es táblát.]

7.12.

Ez a feladat a klózok és az implikációs mondatok közötti kapcsolatot vizsgálja.

a. Mutassa meg, hogy a $(\neg P_1 \vee \ldots \vee \neg P_m \vee Q)$ klóz logikailag ekvivalens a $(P_1 \wedge \ldots \wedge P_m \Rightarrow Q)$ implikációs mondattal.

b. Mutassa meg, minden klóz (függetlenül a pozitív literálok számától) felírható $(P_1 \wedge \ldots \wedge P_m) \Rightarrow (Q_1 \vee \ldots \vee Q_m)$ alakban, ahol P-k és Q-k ítéletkalkulus szimbólumok. Az ilyen mondatokat tartalmazó tudásbázist **implikatív normál formá**jának (**implicative normal form**) vagy **Kowalski formá**jának (**Kowalski form**) nevezzük.

c. Írja le a teljes rezolúciós szabályt implikatív normál formájú mondatokra.

7.13.

Ebben a feladatban tervezze tovább az áramkörön alapuló wumpus ágenst.

a. Írjon egy a (7.4) egyenlethez hasonló egyenletet a *Nyíl* kijelentéshez, amelynek igaznak kell lennie, amikor az ágensnek még van nyíla.

b. Ismételje meg az (a) feladatot egy *ArccalJobbra* kijelentésre, ahol használja fel a (7.5) egyenletet modellként.

c. Készítsen változatokat a (7.7) és (7.8) egyenletekre a wumpus megtalálásához, és rajzolja fel az áramkört.

7.14.

Elemezze, hogy mit is jelent az *optimális* viselkedés a wumpus világban! Mutassa meg, hogy az `IT-WUMPUS-ÁGENS` definíciója nem optimális, és tegyen javaslatot a fejlesztésére!

7.15.

Bővítse ki az `IT-WUMPUS-ÁGENS`-t úgy, hogy képes legyen folyamatosan követni a releváns tényeket a tudásbázisban!

Created by XMLmind XSL-FO Converter.

7.16.

Mennyi ideig tart a DPLL számára a bizonyítása, ahol α egy literális, amelyet *már tartalmaz* a *TB*? Adjon magyarázatot!

7.17.

Kövesse a DPLL viselkedését a 7.15. ábrán leírt tudásbázison, amikor *Q*-t próbáljuk meg bizonyítani, és hasonlítsa össze ezt a működést az előrefelé láncolási algoritmussal.

Created by XMLmind XSL-FO Converter.

8. fejezet - Elsőrendű logika

Ebben a fejezeteben a világ objektumait és ezek kapcsolatait vizsgáljuk meg, és megpróbálunk érvelni ezek felhasználásával.

A 7. fejezetben megmutattuk, hogy egy tudásbázisú ágens hogyan képes reprezentálni az őt körülvevő világot, és hogyan vezeti le a végrehajtandó cselekvéseket. Reprezentációs nyelvként az ítéletkalkulust alkalmaztuk, mert ez elégséges volt a logika és a tudásbázisú ágensek alapvető tulajdonságainak leírásához. Sajnos az ítéletkalkulus nyelvezete túlságosan korlátos ahhoz, hogy összetett környezetekről szerzett tudást megfelelő módon reprezentáljon. Ebben a fejezetben az **elsőrendű logikát**[1] (**first-order logic**) vizsgáljuk, amelynek kifejezőereje elégséges ahhoz, hogy a józan ésszel felfogható tudásunk nagy részét reprezentálja. Ezenkívül az elsőrendű logika sok más reprezentációs nyelvet magában foglal, vagy alapját képezi, és hosszú évtizedek óta vizsgálatok tárgya. A 8.1. alfejezetben a reprezentációs nyelvek általános bemutatásával kezdünk; a 8.2. alfejezetben az elsőrendű logika szintaxisát és szemantikáját tárgyaljuk; a 8.3. és 8.4. alfejezet leírja az elsőrendű logika használatát egyszerű reprezentációk esetén.

1. Még egyszer a reprezentációról

Ebben az alfejezetben a reprezentációs nyelvek természetét vizsgáljuk. Megállapításaink be fogják mutatni az elsőrendű logika fejlesztésének szükségességét, ami egy lényegesen nagyobb kifejezőerővel rendelkező nyelv, mint a 7. fejezetben bemutatott ítéletkalkulus. Megnézzük majd, hogy az ítéletkalkulus és másfajta nyelvek mire képesek, és mire nem. A kérdést itt csak felületesen tárgyaljuk, néhány bekezdésbe sűrítve évszázadok gondolatait, kísérletezéseit és kudarcait.

A programnyelvek – mint például a C++, a Java vagy a Lisp – a használatban lévő formális nyelvek messze legnagyobb csoportját alkotják. A programok önmagukban, szó szerinti értelemben, csak számítási folyamatokat reprezentálnak. A programokon belüli adatstruktúrák tényeket reprezentálhatnak; például egy program használhatja a 4 × 4-es négyzetrács-elrendezést a wumpus világ elemeinek reprezentálásához. Így tehát az a programnyelvi állítás, hogy *Világ*[2, 2] ← *Csapda* viszonylag természetes módja annak a kijelentésnek, hogy van egy csapda a [2, 2] négyzetben. (Az ilyesfajta reprezentációkat *ad hoc* jellegűnek vélhetjük; az adatbázisrendszereket pontosan azért fejlesztették ki, hogy általánosabb, tárgyterület-független módját biztosítsák a tények tárolásának és visszakeresésének.) A programnyelvek hiányossága, hogy nem rendelkeznek általános mechanizmussal arra, hogy tényeket más tényekből levezessenek; az adatstruktúra minden frissítése egy tárgyterület-specifikus eljárás segítségével történik, amelynek részleteit a programozó vezeti le a tárgyterületről meglévő ismereti alapján. Ez a **procedurális** (**procedural**) megközelítés szembeállítható az ítéletkalkulus **deklaratív** (**declarative**) természetével, amelyben a tudás és az interferencia különálló fogalmak, a következtetés pedig teljes mértékben tárgyterületfüggetlen.

A programok adatstruktúráinak (és így az adatbázisoknak is) a másik hátránya annak a képességnek a hiánya, hogy egyszerűen lehessen kijelenteni például azt, hogy „Van egy lyuk a [2, 2]-ben vagy a [3, 1]-ben”, vagy azt, hogy „Ha a wumpus az [1, 1]-ben van, akkor nincs a [2, 2]-ben”. A programok egyetlen értéket tudnak tárolni minden változóhoz, néhány rendszer lehetővé teszi azt is, hogy az érték „ismeretlen” legyen, de hiányzik belőlük a részinformációk kezeléséhez szükséges kifejezőerő.

Az ítéletkalkulus egy deklaratív nyelv, mivel szemantikája mondatok és a lehetséges világok közötti igazságrelációkon alapul. Elégséges a kifejezőereje ahhoz is, hogy részinformációkat kezeljen, diszjunkciók vagy negációk használatával. Az ítéletkalkulusnak ezeken kívül egy harmadik tulajdonsága is van, ami kívánatos a reprezentációs nyelvek esetében, nevezetesen a **kompozíciós képesség** (**compositionality**). Egy kompozíciós nyelvnél a mondat jelentése a mondatrészek jelentésének a függvénye. Például, a „$B_{1,4} \wedge B_{1,2}$” kapcsolatban van a „$B_{1,4}$” és a „$B_{1,2}$” jelentésével. Nagyon furcsa lenne, ha a „$B_{1,4}$” azt jelentené, hogy az [1, 4]-es négyzet büdös, és a „$B_{1,2}$” azt jelentené, hogy az [1, 2]-es négyzet büdös, de a „$B_{1,4} \wedge B_{1,2}$” már azt jelentené, hogy Franciaország és Lengyelország döntetlent játszott a múlt heti jégkorong-selejtező mérkőzésen. Egyértelmű tehát, hogy a kompozíciós képesség hiánya nagyban megnehezíti a következtető rendszerek működését.

[1] **Elsőrendű predikátumkalkulus**nak (**first-order predicate calculus**) is szokták nevezni, és néha ERPK-nak rövidítik.

Created by XMLmind XSL-FO Converter.

Mint azt a 7. fejezetben láttuk, az ítéletkalkulusnak nincs elégséges kifejezőereje egy sok objektumos környezet *tömör* leírásához. Például kénytelenek voltunk minden egyes négyzetnél önálló szabályt írni a csapdákra és a szellőkre:

$$S_{1,1} \Leftrightarrow (C_{1,2} \vee C_{2,1})$$

Magyarul, ezzel szemben, eléggé egyszerűnek tűnik egyszer s mindenkorra kijelenteni, hogy „A lyukas négyzetek mellett lévők szellősek.” A természetes nyelvek szintaxisa és szemantikája lehetővé teszik a környezet tömör és összefogott leírását.

Első látásra úgy tűnik, hogy a természetes nyelvek (mint az angol vagy a magyar) valóban nagyon kifejezők. Képesek voltunk csaknem a teljes könyvet természetes nyelven megírni, és csak alkalmanként kellett áttérnünk más nyelvekre (beleértve a logikát, a matematikát és a diagramok nyelvét). Nagy múltú hagyomány a nyelvészetben és a nyelvek filozófiájában, hogy a természetes nyelvet elsősorban deklaratív tudásreprezentációs nyelvnek tekintjük, és megpróbáljuk rögzíteni ennek formális szemantikáját. Ha egy ilyen kutatás sikeres lenne, nagy előrelépést jelenthetne a mesterséges intelligencia számára, mert egy természetes nyelvet (vagy annak valamilyen származékát) használhatnánk a tudásreprezentációhoz vagy a következtető rendszerekben.

A modern megközelítés szerint a természetes nyelv kissé más célt szolgál, nevezetesen a nyelv a **kommunikáció** (**communication**), és nem annyira a tiszta reprezentáció eszköze. Amikor egy beszélő rámutat valamire, és azt mondja: „Nézd!” a hallgató megtudja azt, hogy mondjuk, Superman végre megjelent a háztetők felett. Mégsem szeretnénk azt kijelenteni, hogy a „Nézd!” mondatban benne foglaltatik ez a tény is. A mondat jelentése mind magától a mondattól, mind attól a **szövegkörnyezet**től (**context**) függ, amelyben a mondat elhangzott. Nyilvánvalóan nem várható, hogy csupán a „Nézd!” mondatot az adatbázisban tárolva visszaadható ennek a jelentése a szövegkörnyezet ismerete nélkül – amiből az a kérdés következik, hogy hogyan tudjuk magát a szövegkörnyezetet reprezentálni. A természetes nyelvekből is hiányzik a kompozíciós képesség – egy olyan mondat jelentése, mint az „És akkor meglátta”, függhet az előtte álló és az utána következő mondatok által alkotott szövegkörnyezettől. Végül pedig a természetes nyelvek egyik gyenge pontja a **többértelműség** (**ambiguity**), ami nehézségeket okozhatna a következtetésben. Amint azt Pinker megállapítja (Pinker, 1995): „Mikor az emberek az *égre* gondolnak, bizonyára nem zavarja meg őket, hogy vajon az égboltról van-e szó, vagy valamiről, ami lángol – és ha egy szó két gondolatot is megjeleníthet, akkor a gondolatok nem feleltethetők meg szavaknak.”

Azt a megközelítést követjük, hogy építünk az ítéletkalkulus alapjaira felhasználva, hogy ez egy deklaratív, kompozíciós szemantika, ami független a szövegkörnyezettől, és egyértelmű. Majd ezekre az alapokra egy még kifejezőbb logikát építünk azáltal, hogy felhasználunk a természetes nyelveknél alkalmazott reprezentációs megoldásokat, miközben igyekszünk elkerülni ennek a hátrányait. A természetes nyelv szintaxisát vizsgálva a legegyszerűbben azonosítható összetevők a főnevek és a főnévi kifejezések, amelyek **objektum**okra (**object**s) utalnak (négyzetek, lyukak, wumpusok), valamint az igék és az igei kifejezés, amelyek az objektumok között fennálló **reláció**kra (**relation**s) utalnak (szellős, szomszédos, kilő). Ezeknek a relációknak egy része **függvény** (**function**) – azaz olyan relációk, amelyeknek csak egyetlen „értékük” van egy adott „bemenethez”. Könnyen találhatunk példákat objektumokra, relációkra és függvényekre:

- Objektumok: emberek, házak, számok, elméletek, Ronald McDonald, színek, baseballmeccs, háborúk, évszázadok…
- Relációk: lehetnek unáris relációk vagy **tulajdonság**ok (**properties**), úgymint piros, kerek, színlelt, elsőrendű, többemeletes…, vagy általánosabb *n*-elemű relációk, úgymint testvére, nagyobb, belsejében, része, színe, utána történt, birtokol, kettő között van…
- Függvények: apja, legjobb barátja, harmadik hivatali ideje, eggyel több mint, kezdete…

Valójában minden állítás elképzelhető úgy, mint ami objektumokra, relációkra és függvényekre vonatkozik. Íme néhány példa:

- „Egy plusz kettő egyenlő hárommal.”

Objektumok: egy, kettő, három, egy meg kettő; Reláció: egyenlő; Függvény: plusz. („Egy plusz kettő a neve annak az objektumnak, amelyet úgy kapunk, hogy a „plusz” függvényt alkalmazzuk az „egy” és „kettő” objektumokra. A három egy másik neve ugyanennek az objektumnak.)

- „A wumpus helyével szomszédos négyzetek büdösek.”

Created by XMLmind XSL-FO Converter.

Objektumok: wumpus, négyzetek; Tulajdonság: büdös; Reláció: szomszédos.

- „A gonosz János király uralkodott Angliában 1200-ban.”

Objektumok: János, Anglia, 1200; Reláció: uralkodott; Tulajdonságok: gonosz, király.

Az **elsőrendű logika** (**first order logic**) nyelvezete, amelynek a szintaxisát és szemantikáját a következő alfejezetben adjuk meg, objektumok és relációk köré épül. Az elsőrendű logika azért olyan fontos a matematika, a filozófia és a mesterséges intelligencia számára, mert ezeket a területeket – amelyek valójában az emberi lét mindennapjait írják le – tekinthetjük úgy, mint amelyek objektumokkal és a köztük lévő relációkkal dolgoznak. Az elsőrendű logika ezenkívül tényeket közölhet az univerzum *néhány* vagy *összes* objektumáról. Ez lehetővé teszi, hogy általános szabályokat vagy törvényszerűségeket tudjunk megfogalmazni, mint például azt az állítást, hogy „A wumpus helyével szomszédos négyzetek büdösek.”

Az elsődleges különbség az ítéletlogika és az elsőrendű logika között az egyes nyelvek **ontológiai meghatározottság**ában (**ontological commitment**) rejlik, abban, hogy mit feltételez a nyelv a *valóság* természetéről.

A gondolat nyelve

A filozófusok és a pszichológusok sokat töprengtek azon, hogy az emberek és más élőlények hogyan jelenítik meg a tudást. Egyértelmű, hogy a természetes nyelv fejlődése fontos szerepet játszott abban, hogy az emberekben kifejlődött ez a képesség. Másrészt, számos pszichológiai eredmény azt sugallja, hogy az emberek nem alkalmazzák közvetlenül a nyelvet a belső reprezentációkhoz. Például a két mondat közül melyikkel kezdtük a 8.1. alfejezetet?

„Amelyben a világ objektumait és ezek kapcsolatait vizsgáljuk meg…”

„Ebben az alfejezetben a reprezentációs nyelvek természetét vizsgáljuk…”

Wanner (1974) úgy találta, hogy a kísérlet alanyai véletlenszerűen – körülbelül 50%-os gyakorisággal – jól választottak, de inkább az elolvasott szöveg tartalmára emlékeztek, több mint 90%-os biztonsággal. Ez azt sugallja, hogy az emberek a szavakat úgy dolgozzák fel, hogy egy nem verbális reprezentációra alakítják át, amit *emlékezet*nek (*memory*) nevezünk.

Továbbra is nagyon érdekes kérdés, hogy mi a pontos mechanizmusa annak, ahogyan a nyelv lehetővé teszi és formálja a gondolatok reprezentációját az emberekben. A híres *Sapir-Whorf-hipotézis* szerint az általunk beszélt nyelv nagymértékben befolyásolja gondolkodásunk és döntéseink miként jét, különösen abban, ahogy kategóriarendszereket állítunk fel, amelyet felhasználva aztán felosztjuk a világot különböző fajta objektumokra. Whorf szerint (Whorf, 1956), azáltal, hogy az eszkimóknak számos szavuk van a hóra, másképpen is érzékelik a havat, mint más nyelvek beszélői. Néhány nyelvész vitatja ennek az állításnak a tényszerűségét – Pullum úgy érvel (Pullum, 1991), hogy az inuitoknak, a yupiknak és más rokon nyelvet beszélőknek ugyanannyi szavuk van a hóval rokonítható fogalmakra, mint az angoloknak – vannak azonban olyanok is, akik támogatják az állítást (Fortescue, 1984). Vitathatatlanul igaznak tűnik, hogy az olyan embercsoportok, amelyeknek több ismeretük van a világnak egy bizonyos részéről, sokkal részletesebb szókinccsel rendelkeznek erre a részre vonatkozóan – például a gyakorló rovarszakértők az általunk *bogaraknak* nevezett lényeket fajok százezreire osztják fel, és sokról ezek közül személyes ismeretekkel rendelkeznek. (Az evolúciós biológus J. B. S. Haldane meg is állapította, hogy a Teremtő „Bogarak iránti mértéktelen szeretete” különös.) Mi több, a tapasztalt síelőknek rengeteg szavuk van a hóra – porhó, latyak, tört krumpli, aludttej, kukorica, cukor, aszfalt, kordbársony, pehely, trutymó és így tovább –, és ezek olyan különbségeket mutatnak, amelyek ismeretlenek egy kívülálló számára. Az ok-okozati viszony iránya azonban ismeretlen – kérdés, hogy a síelők csak azért ismerik fel a különbségeket, mert megtanulják a szavakat, vagy a különbségek keletkeznek először a személyes tapasztalat alapján, és ezeket címkézik fel aztán a közösségben már ismert fogalmakkal. Ez a kérdés különösen a gyermekek fejlődésének tanulmányozása szempontjából fontos. Egyelőre kevés ismeretünk van arról, hogy milyen mértékben fonódik egymásba a nyelvtanulás és a gondolkodás elsajátítása. Például egy olyan fogalom nevének ismerete, mint például az *agglegény*, megkönnyíti-e bonyolultabb, erre épülő fogalmak előállítását és használatát, amelyek tartalmazzák ezt a nevet – *partiképes agglegény?*

Created by XMLmind XSL-FO Converter.

Az ítéletlogika feltételezi például, hogy a definiált tények fennállnak vagy nem állnak fenn a világban. Minden ténynek két állapota lehet: igaz vagy hamis.[2] Az elsőrendű logika bonyolultabb feltételezésekre épít, nevezetesen, hogy a világ objektumokból épül fel, amelyek között relációk léteznek, amelyek vagy fennállnak, vagy nem. A speciális célú logikák további ontológiai hozzárendeléseket tesznek; például a **temporális logika** (**temporal logic**) feltételezi, hogy a tények csak bizonyos *időben* állnak fenn, és hogy ezek az idők – amelyek lehetnek időpontok és időtartamok is – rendezhetők. Így ezek a speciális célú logikák az objektumok egy bizonyos fajtájához (és az azokhoz tartozó axiómákhoz) „első osztályú" státust rendelnek a logikán belül, ahelyett hogy egyszerűen meghatároznák őket a tudásbázison belül. A **magasabb rendű logika** (**higher-order logic**) úgy tekint az elsőrendű logikában relációknak és függvényeknek nevezettekre, mintha azok maguk is objektumok lennének. Ez lehetővé teszi, hogy *minden* relációról állításokat alkossunk – például arra, ha valaki azt szeretné meghatározni, mit jelent, hogy egy reláció tranzitív. Eltérően más speciális célú logikáktól a magasabb rendű logika egyértelműen nagyobb kifejezőerővel rendelkezik, mint az elsőrendű logika, abban az értelemben, hogy néhány magasabb rendű logikai állítás nem írható le véges számú elsőrendű logikai állítással.

Egy logika jellemezhető még az **ismeretelméleti megállapítás**aival (**epistemological commitment**s) is – az egyes tényekhez megengedett tudásállapotokkal. Mind az ítéletkalkulusban, mind az elsőrendű logikában egy állítás egy tényt reprezentál, és az ágens vagy elhiszi, hogy az állítás igaz, vagy hamisnak véli, vagy nincs róla véleménye. Ezekben a logikákban tehát a tudásnak három lehetséges állapota van, bármely állítást is tekintjük. Ezzel szemben azok a rendszerek, amelyek a **valószínűség-elmélet**et (**probability theory**) használják, tartalmazhatnak egy *hiedelemfokozatot*, ami a 0-tól (teljes hitetlenség) az 1-ig tart (teljes elfogadás).[3] Például, egy valószínűsíthető wumpusvilágbeli ágens hiheti, hogy a wumpus az [1, 3]-ban van, 0,75-ös valószínűséggel. Az öt különböző logikai rendszer ontológiai és ismeretelméleti megállapításait a 8.1. táblázatban foglaltuk össze.

8.1. ábra - Formális nyelvek, valamint azok ontológiai és ismeretelméleti hozzárendelései

Nyelv	Ontológiai meghatározottság (ami a világban létezik)	Ismeretelméleti hozzárendelések (amit az ágens elhisz a tényekről)
Ítéletlogika	Tények	Igaz/hamis/ismeretlen
Elsőrendű logika	Tények, objektumok, relációk	Igaz/hamis/ismeretlen
Temporális logika	Tények, objektumok, relációk, idők	Igaz/hamis/ismeretlen
Valószínűség-elmélet	Tények	Hiedelemfokozat $\in [0, 1]$
Fuzzy logika	Tények igazsági fokkal $\in [0, 1]$	Ismert intervallumérték

A következő alfejezetben elkezdjük az elsőrendű logika részletes tárgyalását. Ahogy egy fizikát tanuló diáknak valamilyen szinten rendelkeznie kell matematikai ismeretekkel, a mesterséges intelligenciát tanulmányozóknak is el kell sajátítaniuk bizonyos jártasságot a logikai jelölési rendszerek használatában. Másrészt az is nagyon fontos, hogy *ne* bonyolódjunk bele a logikai jelölési rendszerek sajátosságaiba – hiszen számos különböző változat létezik. A lényeges az, hogy tudatában legyünk annak, hogy a nyelv miként könnyíti meg tömör reprezentációk létrehozását, és hogy a szemantika miként vezet el helyes következtetési folyamatokhoz.

2. Az elsőrendű logika szintaxisa és szemantikája

Az alfejezetet azzal kezdjük, hogy pontosabban meghatározzuk azt a módot, amellyel az elsőrendű logikában előforduló lehetséges világok leírják az objektumokra és relációkra tett ontológiai megállapításokat. Ezután bemutatjuk a nyelv különböző elemeit, és ahogyan haladunk, megmagyarázzuk ezek szemantikáját.

2.1. Az elsőrendű logika modelljei

Emlékezzünk arra a 7. fejezetből, hogy a logikai nyelvek modelljei azok a formális struktúrák, amelyek a vizsgálat tárgyát képező lehetséges világokat alkotják. Az ítéletkalkulus modelljei csupán az ítéletszimbólumok

[2] Ezzel szemben a **fuzzy logiká**ban (**fuzzy logic**) definiált az **igazság foka** (**degree of truth**), ami egy 0 és 1 közötti érték. Például az a mondat, hogy „Bécs egy nagy város" a mi világunkban csak 0,6 fokig lehet igaz.

[3] Fontos, hogy ne keverjük össze a valószínűség-számításban alkalmazott hiedelemfokozatot a fuzzy logikában reprezentált igazsági fokozattal. Létezik néhány olyan bizonytalanságot kezelő rendszer, amely megenged határozatlanságot (a hiedelem fokát) az igazsági fokozatokkal kapcsolatban.

Created by XMLmind XSL-FO Converter.

igazságértékeinek halmazai. Az elsőrendű logika modelljei ennél érdekesebbek. Először is, ezek tartalmaznak objektumokat! A modell **tárgyterülete** (**domain**) azoknak az objektumoknak a halmaza, amelyeket a tárgyterület tartalmaz; ezeket az objektumokat szoktuk a **tárgyterület eleme**inek (**domain element**s) nevezni. A 8.2. ábra bemutat egy modellt öt objektummal: Oroszlánszívű Richárd, Anglia királya 1189-től 1199-ig; öccse, gonosz János király, aki 1199-től 1215-ig uralkodott; Richárd és János bal lába; végül egy korona.

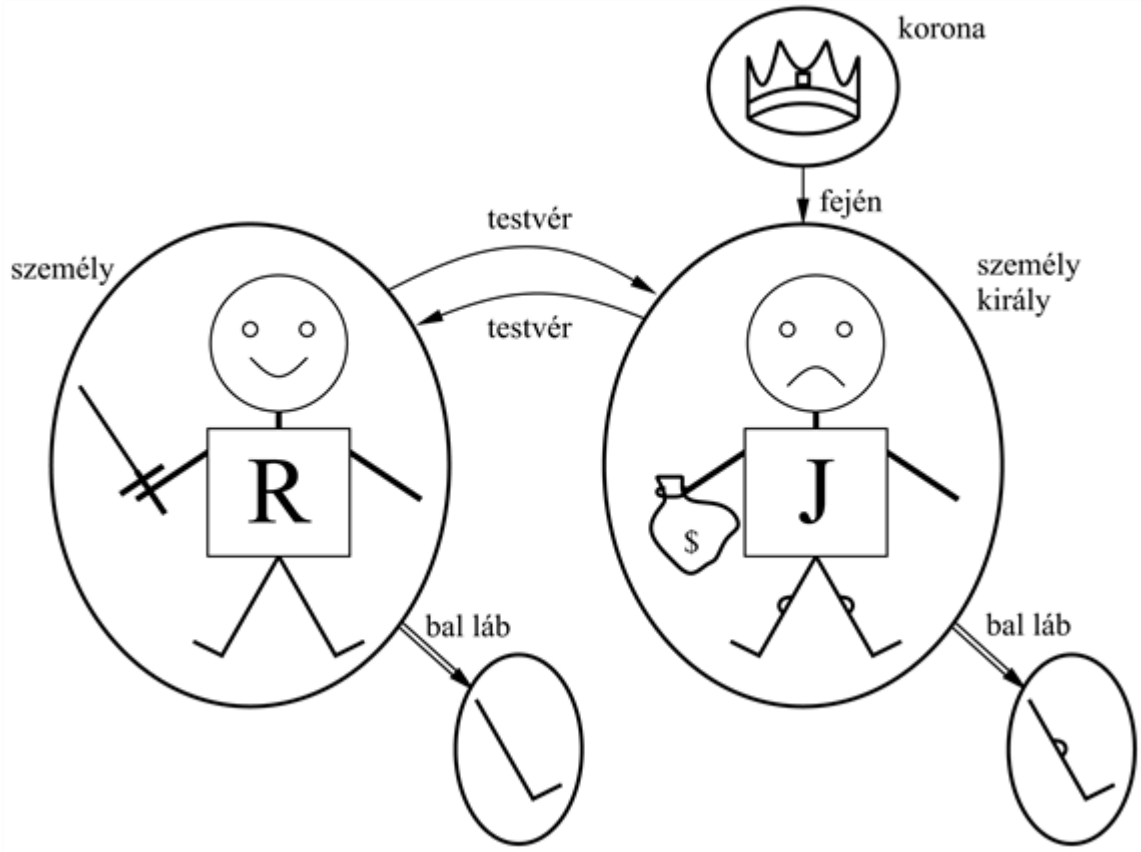
A modellben található objektumok többféle relációban lehetnek egymással. A 8.2. ábrán Richárd és János testvérek. Formálisan, egy reláció csak egymással kapcsolatban lévő objektumok **n-es**einek (**tuple**) halmaza. (Egy *n*-es az objektumok egy rögzített sorrendben felsorolt gyűjteménye, amely objektumokat a ⟨ ⟩ zárójelpár vesz körbe.) Így tehát ebben a modellben a testvérreláció a következő halmaz:

```
{⟨Oroszlánszívű Richárd, János király⟩, ⟨János király, Oroszlánszívű Richárd⟩}(8.1)
```

(Itt magyarul neveztük meg az objektumokat, de képzeletben be lehetne helyettesíteni a neveket például képekkel is.) A korona János király fején van, tehát a „fején” relációhoz csak egy *n*-es tartozik ⟨ a korona, János király⟩ .

A „testvér” és a „fején” relációk bináris relációk – ez azt jelenti, hogy két objektum között állítanak fel kapcsolatot. A modell ezenkívül egyelemű, unáris relációkat vagy tulajdonságokat is tartalmaz: A „személy” tulajdonság egyaránt igaz Richárdra és Jánosra; a „király” tulajdonság csak Jánosra igaz (feltehetően azért, mert Richárd ekkor már halott volt); míg a „korona” tulajdonság csak a koronára igaz.

8.2. ábra - Egy öt objektumot – két bináris relációt, három egyelemű relációt (címkékkel jelezve az objektumokon) és egy egyargumentumú függvényt, bal láb – tartalmazó modell



A kapcsolatok egyes típusai leginkább függvényeknek tekinthetők, ahol egy adott objektumnak pontosan csak egy másik objektummal kell ilyen kapcsolatban lennie. Például minden egyes személynek egy bal lába van, tehát a modellben található egy „bal láb” függvény, ami a következő leképezéseket teszi lehetővé:

```
⟨Oroszlánszívű Richárd⟩ ⟶Richárd bal lába
```


Created by XMLmind XSL-FO Converter.

⟨ János király⟩ →János bal lába (8.2)

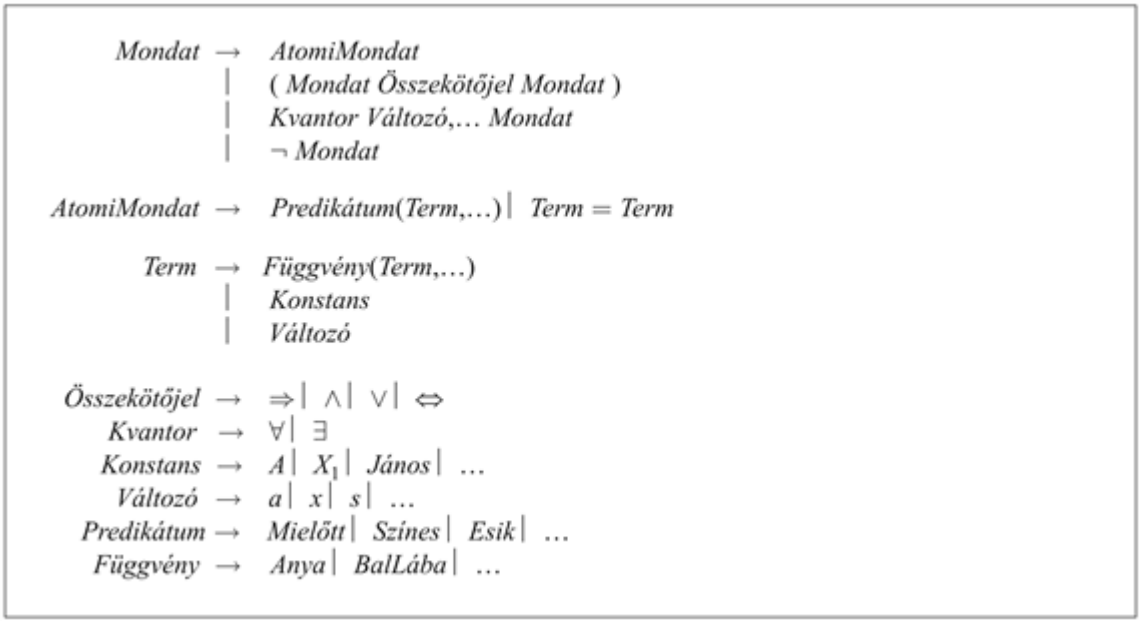
Pontosabban megfogalmazva, az elsőrendű logika modelljei **totális függvény**eket (**total function**s) kívánnak meg, ami azt jelenti, hogy minden bemeneti *n*-eshez tartoznia kell egy értéknek. Így tehát a koronához tartoznia kell egy bal lábnak, és ennek így kell lennie minden bal láb esetében. Létezik erre a problémára egy technikai megoldás, amely egy hozzáadott „láthatatlan” objektumot vezet be: egy bal lábat rendel mindenhez, amelynek nincs bal lába, beleértve önmagát is. Szerencsére, amíg valaki nem tesz kijelentéseket a bal lábakról vagy olyan dolgokról, amelyeknek nincs bal lábuk, ezeknek a technikai problémáknak nincs jelentőségük.

2.2. Szimbólumok és interpretációk

Most pedig nézzük a nyelv szintaxisát. A türelmetlen olvasó teljes leírást talál az elsőrendű logika formális nyelvtanáról a 8.3. ábrán.

Az elsőrendű logika alapvető szintaktikai elemei az objektumokat, relációkat és függvényeket megjelenítő szimbólumok. A szimbólumoknak tehát három típusa létezik: **konstansszimbólum**ok (**constant symbol**s), amelyek az objektumokat jelölik; **predikátumszimbólum**ok (**predicate symbol**s), amelyek a relációkat jelenítik meg; és a **függvényszimbólum**ok (**function symbol**s), amelyekkel függvényekre hivatkozhatunk. Azt a konvenciót fogjuk követni, hogy a szimbólumok mind nagybetűvel kezdődnek. Például használhatjuk a *Richárd* és *János* konstansszimbólumokat; *a Testvér*, *Fején*, *Személy*, *Király* és *Korona* predikátumszimbólumokat, valamint a *BalLáb* függvényszimbólumot. Úgy, mint az ítéletszimbólumoknál, a nevek megválasztása itt is teljes mértékben a felhasználóra van bízva. Minden egyes predikátum- és függvényszimbólum együtt jár egy **számosság**gal (**arity**), ami behatározja a paraméterek számát.

8.3. ábra - Az elsőrendű logika (egyenlőséggel kiegészített) szintaxisa Backus–Naur-forma jelöléseivel. (Lásd az 1. szakasz - B1. Nyelvek definiálása Backus–Naur-Formában (BNF) részben, ha ez a jelölés ismeretlen.) A szintaxis szigorú a zárójelezés tekintetében; a zárójelekről és az operátori elsőbbségről a 3.1. szakasz - Szintaxis részben szereplő megjegyzések ugyanúgy vonatkoznak az elsőrendű logikára is.



A szemantika a mondatokat a modellekhez kapcsolja, és képesnek kell lennie az igazság meghatározására. Hogy ez megtörténjen, szükségünk van egy **interpretáció**ra, amely pontosan megjelöli, hogy mely objektumok, relációk és függvények felelnek meg a konstans-, predikátum- és függvényszimbólumoknak. Egy lehetséges interpretáció – amit **szándékolt interpretáció**nak (**intended interpretation**) nevezünk – a mi esetünkben a következő:

- *Richárd* Oroszlánszívű Richárdra, míg *János* a gonosz János királyra vonatkozik.

Created by XMLmind XSL-FO Converter.

- A *Testvér* a testvéri kapcsolatra vonatkozik, ami a (8.1) egyenletben megadott *n*-es halmazban látható; a *Fején* arra a „fején” relációra vonatkozik, ami a korona és János király között áll fenn; a *Személy*, *Király* és *Korona* pedig arra az objektumhalmazra vonatkoznak, amelyek személyek, királyok és koronák.

- A *BalLáb* azt a „bal láb” függvényt jelenti, aminek a leképezését a (8.2) egyenletben adtuk meg.

Sok más lehetséges interpretáció is létezik, ami ezeknek a szimbólumoknak a modellhez való kapcsolatát írja le. Például egy lehetséges interpretációban *Richárd* jelenti a koronát és *János* a János király bal lábát. A modell öt objektumot tartalmaz, így 25 lehetséges interpretáció adható meg csak a *Richárd* és a *János* konstansszimbólumokhoz.

Figyeljük meg, hogy nem minden objektumnak van arra szüksége, hogy elnevezzék – például a szándékolt interpretáció nem nevezi meg a koronát vagy a lábakat. Az is lehetséges, hogy egy objektumnak több neve legyen; létezik olyan interpretáció, amelyben a *Richárd* és a *János* is a koronára vonatkozik. Ha ez a lehetőség zavaró, emlékezzünk arra, hogy az ítéletlogikában teljességgel elfogadható egy olyan modell, amelyben a *Felhős* és a *Napos* egyaránt igaz. A tudásbázis feladata az, hogy kizárja azokat a modelleket, amelyek ellentmondásban vannak ismereteinkkel.

Bármely mondat igazságát egy modell és a mondat szimbólumainak az interpretációja határozza meg. Így tehát a vonzat, az érvényesség és más jellemzők az *összes lehetséges modell* és az *összes lehetséges interpretáció* termjeinek felhasználásával vannak meghatározva. Fontos megjegyezni, hogy a tárgyterület elemek száma minden egyes modellben korlátlan lehet – a tárgyterület elemei például lehetnek egész vagy valós számok. Ebből következően, a lehetséges modellek száma korlátlan, mint ahogy az interpretációk száma is. A maga után vonzásnak az összes lehetséges modell felsorolásával történő vizsgálata, ami az ítéletlogikában működik, nem lehetséges az elsőrendű logikában. Még ha az objektumok száma korlátos is, a kombinációk száma nagyon nagy lehet. A példánkban szereplő objektumokkal hozzávetőlegesen 10^{25} a kombinációk száma, egy öt objektummal rendelkező tárgyterületben (lásd 8.5. feladat).

2.3. Termek

Egy **term** (**term**) egy objektumra vonatkozó logikai kifejezés. A konstansszimbólumok tehát termek, de nem mindig kényelmes önálló szimbólumot alkalmazni minden egyes objektum megnevezésénél. Magyarul inkább a „János király bal lába” kifejezést használjuk, mintsem nevet adjunk a lábának. Ezért használunk függvényszimbólumokat: a konstansszimbólumok használata helyett inkább a *BalLáb*(*János*)-t alkalmazzuk. Általános esetben egy összetett term egy függvényszimbólumból áll, amelyet a függvényszimbólum argumentumainak zárójelezett listája követ. Fontos megjegyezni, hogy egy összetett term nem más, mint egy bonyolult formájú név, és nem egy „értéket visszaadó szubrutin hívás”. Nem egy *BalLáb* szubrutinról van szó, amely bemenetként megkap egy személyt, és visszaad egy lábat. Tudunk következtetni a bal lábakkal kapcsolatban (pl. kiindulva az általános szabályból, hogy mindenkinek van egy bal lába, és ebből levezethető, hogy Jánosnak is lennie kell) anélkül, hogy valaha is megadnánk a *BalLáb* definícióját. Ez is olyasmi, amit nem tehetünk meg a programozási nyelvek szubrutinjaival.[4]

A termek formális szemantikája nyilvánvaló. Tekintsük az $f(t_1, \ldots, t_n)$ termet. Az f függvényszimbólum a modellben található valamely függvényre (nevezzük F-nek) vonatkozik; az argumentum termek a tárgyterületben fellelhető objektumokra vonatkoznak (nevezzük őket $d_1, \ldots, d_n$-nek); és a term egészében arra az objektumra vonatkozik, ami a $d_1, \ldots, d_n$-re alkalmazott F függvény értéke. Például tételezzük fel, hogy ha a *BalLáb* függvényszimbólum a (8.2) egyenletben mutatott függvényre vonatkozik, és a *János* vonatkozik János királyra, akkor a *BalLáb*(*János*) János király bal lábát jelöli. Ezen a módon az interpretáció rögzíti minden termnek a referenciáit.

2.4. Atomi mondatok

Most, hogy vannak objektumokra vonatkozó termeink és relációkra vonatkozó predikátumszimbólumaink, összerakhatjuk őket, hogy tényeket kifejező **atomi mondat**okat (**atomic sentence**s) állítsunk elő. Egy atomi

[4] A **λ-kifejezés**ek (**λ -expression**s) hasznos jelölést biztosítanak, amelyek az új függvényszimbólumokat futási időben hozzák létre. Például a függvény, amely négyzetre emeli a számokat, leírható úgy, hogy ($\lambda\ x\ \ x \times x$), és alkalmazható az argumentumokra, akár bármely más függvényszimbólum. Egy λ -kifejezést úgy is meghatározhatunk és használhatunk, mint egy predikátumszimbólumot (lásd 22. fejezet). A Lambda operátor a Lispben pontosan ugyanezt a szerepet játssza. Vegyük észre, hogy a λ használata ilyen módon nem növeli meg az elsőrendű logika formális kifejezőerejét, mert az a mondat, ami tartalmaz egy λ -kifejezést, átírható úgy, hogy hozzácsatoljuk a paramétereket, ami egy ugyanolyan mondatot eredményez.

Created by XMLmind XSL-FO Converter.

mondatot egy predikátumszimbólum és az azt követő zárójelezett listában található termek listája alkot. Például a

```
Testvér(Richárd, János)
```

azt jelenti egy korábban megadott szándékolt interpretáció szerint, hogy Oroszlánszívű Richárd testvére János királynak.[5] Az atomi mondatoknak is lehetnek összetett termek az argumentumai. Így tehát a

```
Házas(Apja(Richárd), Anyja(János))
```

azt jelenti, hogy Oroszlánszívű Richárd apja feleségül vette János király anyját (ismét csak egy alkalmas interpretációban).

Fontos

*Egy atomi mondat **igaz (true)** egy adott modellben, egy adott interpretáció mellett, ha a reláció, amire a predikátumszimbólum vonatkozik, fennáll az argumentumok által jelölt objektumok között.*

2.5. Összetett mondatok

Összetett mondatok létrehozásához használhatunk **logikai összekötőjel**eket (**logical connective**s) úgy, mint az ítéletkalkulusban. A logikai kötőszavakkal formált mondatok szemantikája azonos az ítéletkalkulusban látott szemantikával. Íme négy mondat, amelyek igazak a 8.2. ábra modelljében, a szándékolt interpretációnk szerint.

```
¬Testvér(BalLáb(Richárd), János)

Testvér(Richárd, János) ∧ Testvér(János, Richárd)

Király(Richárd) ∨ Király(János)

¬Király(Richárd) ⇒ Király(János)
```

2.6. Kvantorok

Ha egyszer létrehoztunk egy objektumokat is tartalmazó logikát, természetes, hogy objektumok egész gyűjteményeire vonatkozó tulajdonságokat is ki akarunk fejezni anélkül, hogy az objektumokat nevük felhasználásával fel kellene sorolni. Ezt a **kvantor**ok (**quantifier**s) alkalmazása teszi lehetővé. Az elsőrendű logika két standard kvantort tartalmaz, amelyeket *univerzális* és *egzisztenciális* kvantoroknak nevezünk.

2.7. Univerzális kvantor (∀)

Emlékezzünk vissza arra, hogy a 7. fejezetben milyen nehézségeink voltak általános szabályok ítéletlogikában történő megfogalmazásával. Az olyan szabályok, mint az „A wumpus mellett található négyzetek büdösek” vagy a „Minden király személy” adják meg a savát-borsát az elsőrendű logikának. Az első szabállyal a 8.3. alfejezetben fogunk foglalkozni. A második szabály, „Minden király személy”, az elsőrendű logikában így írható le:

```
∀x Király(x) ⇒ Személy(x)
```

A ∀-t általában „Minden …re”-nek olvassuk ki. A mondat tehát azt jelenti: „Minden *x*-re, ha *x* egy király, akkor *x* egy személy.” Az *x* szimbólum neve **változó** (**variable**). A konvenció alapján a változókat kisbetűvel írjuk. Egy változó önmagában egy term, és mint olyan, szerepelhet egy függvény argumentumaként – például: *BalLáb*(*x*). Egy változó nélküli termet **alapterm**nek (**ground term**) nevezünk.

Egyszerűen a ∀ *x P* mondat, ahol a *P* egy tetszőleges logikai kifejezés, azt mondja ki, hogy a *P* igaz minden *x* objektumra. Pontosabban, ∀ *x P* igaz egy adott modellre az interpretáción belül, ha *P* igaz minden lehetséges **kiterjesztett interpretáció**ban (**extended interpretation**s), amelyet az adott interpretáció felhasználásával hozunk létre, ahol minden egyes kiterjesztett interpretáció meghatároz egy tárgyterületelemet, amire az *x* vonatkozik.

[5] Rendszerint követjük az argumentumok rendezésének konvencióját, amely szerint a *P*(*x*,*y*)-t úgy értelmezzük, hogy az „*x* az *y*-nak a *P*-je”.

Created by XMLmind XSL-FO Converter.

Ez bonyolultnak hangzik, de valójában csak egy óvatos módja annak, hogy megállapítsuk az univerzális kvantorok intuitív jelentését. Tekintsük a 8.2. ábrát és a hozzá tartozó szándékolt interpretációt. Ötféle módon tudjuk kiterjeszteni az interpretációt:

x ⟶Oroszlánszívű Richárd

x ⟶János király

x ⟶Richárd bal lába

x ⟶János bal lába

x ⟶a korona

A ∀*x* *Király*(*x*) ⇒ *Személy*(*x*) univerzális kvantort alkalmazó mondat igaz az eredeti interpretációban, ha a *Király*(*x*) ⇒ *Személy*(*x*) mondat igaz mind az öt kiterjesztett interpretációra. Eszerint az univerzális kvantorral ellátott mondat egyenértékű a következő öt mondattal:

Oroszlánszívű Richárd egy király ⇒ Oroszlánszívű Richárd egy személy

János király egy király ⇒ János király egy személy

Richárd bal lába egy király ⇒ Richárd bal lába egy személy

János bal lába egy király ⇒ János bal lába egy személy

A korona egy király ⇒ a korona egy személy

Nézzük meg alaposan ezt az állításhalmazt. Mivel a mi modellünkben János király az egyetlen király, a második mondat kijelenti, hogy ő egy személy, ahogy azt reméltük is. De mi a helyzet a többi négy mondattal, amelyek azért szerepelnek, hogy állításokat fogalmazzanak meg lábakról és koronákról? Része ez annak a jelentésnek, hogy „Minden király személy”? Valójában, a többi négy állítás is igaz a modellben, de egyáltalán nem állítanak semmit a lábak, koronák vagy akár Richárd személyes tulajdonságairól. Ez azért van, mert ezen objektumok egyike sem király. Ha megnézzük a (⇒) összekötőjel igazságtábláját (7.8. ábra), láthatjuk, hogy az implikáció igaz, amikor a premisszája hamis – *tekintet nélkül* a konklúzió igazságtartalmára. Így tehát az univerzális kvantorral ellátott mondat kijelentése egyenértékű azzal, hogy egyedi implikációk teljes listáját definiáljuk. Végül is egy szabály konklúzióját adjuk meg azokra az objektumokra, amelyek premisszája igaz, de nem mondunk egyáltalán semmit azokról az egyedekről, akikre nézve a premissza hamis. Így az implikáció igazságtáblájának sorai tökéletesnek bizonyulnak univerzális kvantorokat tartalmazó általános szabályok megírásához.

Gyakori hiba, amelyet még figyelmes olvasók is elkövetnek, akik pedig már többször is elolvasták ezt a bekezdést, hogy konjunkciót használnak implikáció helyett. A

∀*x* *Király*(*x*) ∧ *Személy*(*x*)

mondat egyenértékű lenne azzal az állítással, hogy

Oroszlánszívű Richárd egy király ∧ Oroszlánszívű Richárd egy személy

János király egy király ∧ János király egy személy

Richárd bal lába egy király ∧ Richárd bal lába egy személy

és így tovább. Nyilvánvaló, hogy ez nem azt jelenti, mint amit szeretnénk elérni.

2.8. Egzisztenciális kvantor (∃)

Az univerzális kvantor állításokat fogalmaz meg minden objektumról. Hasonlóan, az egzisztenciális kvantor felhasználásához állításokat tehetünk az univerzum *néhány* objektumáról anélkül, hogy megneveznénk azokat. Azt az állítást tehát, hogy János királynak koronája van, így is leírhatjuk:

∃*x* *Korona*(*x*) ∧ *Fején*(*x*, *János*)

Az ∃ *x* kiolvasása „Létezik egy olyan *x*…” vagy „Néhány *x*-re…” .

Created by XMLmind XSL-FO Converter.

Érthetően fogalmazva, az ∃ *x P* mondat azt jelenti, hogy a *P* legalább egy *x* objektumra igaz. Pontosabban az ∃ *x P* igaz egy adott modellben, egy adott interpretáción belül, ha a *P* igaz *legalább egy* kiterjesztett interpretációban, amely az *x*-et a tartomány egy eleméhez rendeli. A mi példánkban ez azt jelenti, hogy a következő állítások közül legalább egy igaz:

Oroszlánszívű Richárd egy korona ∧ Oroszlánszívű Richárd János fején van

János király egy korona ∧ János király János fején van

Richárd bal lába egy korona ∧ Richárd bal lába János fején van

János bal lába egy korona ∧ János bal lába János fején van

A korona egy korona ∧ a korona János fején van

Az ötödik állítás igaz a modellben, így az eredeti egzisztenciális kvantorral ellátott mondat igaz a modellben. Vegyük észre, hogy definíciónk szerint a mondat igaz lenne egy olyan modellben is, ahol János király két koronát visel. Ez teljes mértékben konzisztens az eredeti mondattal, miszerint „János királynak korona van a fején.”[6]

Mint ahogy természetesnek tűnik az ⇒ használata összekötőjelnek a ∀ használatakor, az ∧ az a természetes összekötőjel, amit az ∃-vel használunk. Az ∧ használata fő összekötőjelként az ∀-val túlságosan is erős állításokhoz vezetett az előző alfejezetben tárgyalt példában; a ⇒ használata az ∃-vel igazából nagyon gyenge állításhoz vezet:

$$\exists x\ Korona(x) \Rightarrow Fején(x, János)$$

A felszínen ez a mondatunk elfogadható átírásának tűnik. A szemantikát alkalmazva látjuk, hogy a mondat azt jelenti ki, hogy a következő állítások legalább egyike igaz:

Oroszlánszívű Richárd egy korona ⇒ Oroszlánszívű Richárd János fején van

János király egy korona ⇒ János király János fején van

Richárd bal lába egy korona ⇒ Richárd bal lába János fején van

és így tovább. Mármost, egy implikáció akkor igaz, ha a premissza és a konklúzió egyaránt igaz, *vagy ha a premisszája hamis*. Tehát ha Oroszlánszívű Richárd nem egy korona, akkor az első állítás igaz, és az egzisztenciális kvantort kielégítettük. Így tehát egy egzisztenciális kvantort tartalmazó implikációs mondat igaz bármely olyan modellben, amely tartalmaz egy olyan objektumot, amelyre az implikáció premisszája hamis; következésképpen az ilyen mondatok valójában nem túl sokat mondanak.

2.9. Egymásba ágyazott kvantorok

Gyakran szükségünk van még összetettebb mondatok kifejezésére, amelyek több kvantort is használnak. A legegyszerűbb eset az, amikor a kvantorok azonos típusúak. Például „A fivérek testvérek” mondat leírható úgy, hogy:

$$\forall x\ \ \forall y\ Fivér(x, y) \Rightarrow Testvér(x, y)$$

Az egymást követő azonos típusú kvantorokat írhatjuk úgy is, hogy egy kvantort alkalmazunk az összes változóra. Például hogy azt állítsuk, a testvérség egy szimmetrikus kapcsolat, ezt írhatjuk:

$$\forall x, y\ Testvér(x, y) \Leftrightarrow Testvér(y, x)$$

Más esetekben kevert jelöléssel van dolgunk. A „mindenki szeret valakit” azt jelenti, hogy minden személyhez tartozik valaki, akit a személy szeret:

$$\forall x\ \ \exists y\ Szeret(x, y)$$

[6] Létezik az egzisztenciális kvantornak egy variánsa, amelyet ∃1-gyel vagy ∃!-lel jelölünk, és ami azt jelenti, hogy „Létezik pontosan egy…”. Ugyanez a jelentés kifejezhető egyenlőségállításokkal is, mint azt a 8.2. alfejezetben bemutattuk.

Created by XMLmind XSL-FO Converter.

Másrészt, hogy azt állítsuk: „Van valaki, akit mindenki szeret”, ezt írjuk:

$\exists y \ \forall x \ Szeret(x, y)$

A kvantorok sorrendje tehát nagyon fontos. Még világosabbá válik, ha zárójeleket is alkalmazunk. A $\forall x$ ($\exists y$ *Szeret*(x, y)) azt jelenti, hogy *mindenkinek* van egy bizonyos tulajdonsága, nevezetesen, az a tulajdonság, hogy mindenki szeret valakit. Másrészt, a $\exists x$ ($\forall y$ *Szeret*(x, y)) azt jelenti, hogy a világban *valakinek* van egy bizonyos tulajdonsága, méghozzá az, hogy mindenki szereti őt.

Zavart okozhat, ha két kvantort használunk ugyanazzal a változó névvel. Tekintsük a következő mondatot:

$\forall x \ [Korona(x) \lor (\exists x \ Fivér(Richárd, x))]$

Itt az x a *Fivér*(*Richárd*, x)-ben *egzisztenciális* kvantorral van ellátva. Az a szabály, hogy a változó ahhoz a legbelső kvantorhoz tartozik, ami a változót megemlíti, és utána a változó nem tárgya semmilyen kvantor alárendelésnek.[7]

A másik módja a megoldásnak: $\exists x$ *Fivér*(*Richárd*, x), ahol ez egy Richárdról szóló mondat (hogy neki van egy fivére), és nem pedig az x-ről. Így tehát annak nem lesz hatása, ha kívül elé teszünk egy $\forall x$-et. Éppen így írhattuk volna azt is, hogy $\exists z$ *Fivér*(*Richárd*, z). Mivel ez tévesztés forrása lehet, ezért mindig különböző változókat fogunk használni.

2.10. Az ∀ és az ∃ kapcsolata

A két kvantor valójában szoros kapcsolatban áll egymással a negáción keresztül. Ha valaki azt mondja, hogy mindenki utálja a paszternákot, akkor egyúttal azt is mondja, hogy nem létezik olyan valaki, aki szereti azt, és fordítva:

$\forall x \ \neg Szereti(x, Paszternák)$ egyenértékű azzal, hogy $\neg \exists x \ Szereti(x, Paszternák)$

Egy lépéssel tovább is mehetünk. A „Mindenki szereti a fagylaltot” mondat azt jelenti, hogy nincs olyan, aki ne szeretné a fagylaltot:

$\forall x \ Szereti(x, Fagylalt)$ egyenértékű azzal, hogy $\neg \exists x \ Szereti(x, Fagylalt)$

Mivel az ∀ igazából egy konjunkció minden univerzumbeli objektum felett, az ∃ pedig egy diszjunkció ugyanezen objektumok felett, ezért nem meglepő, hogy eleget tesznek a De Morgan-szabályoknak. A kvantorral és a kvantor nélkül álló mondatokra vonatkozó De Morgan-szabályok a következők:

$\forall x \ \neg P \equiv \neg \exists x \ P$	$\neg \Pi \land \neg Q \equiv \neg (P \lor Q)$
$\neg \forall x \ \ P \equiv \exists x \ \neg P$	$\neg (P \land Q) \equiv \neg P \lor \neg Q$
$\forall x \ P \equiv \neg \exists x \ \neg P$	$P \land Q \equiv \neg (\neg P \lor \neg Q)$
$\exists \ x \ P \equiv \neg \forall x \ \neg P$	$P \lor Q \equiv \neg (\neg P \land \neg Q)$

Így valójában nincs szükségünk egyszerre mind az ∀-ra, mind az ∃-re, mint ahogy nincs szükség egyszerre mind az ∧-ra és az ∨-ra sem. Mégis, mivel az olvashatóság fontosabb, mint a takarékosság, használni fogjuk mindkét kvantort.

2.11. Egyenlőség

[7] Az azonos változónevet használó kvantorok interferenciára való képessége teszi lehetővé a kiterjesztett interpretációk kissé barokkos mechanizmusát a kvantorral ellátott mondatok szemantikájában. Az intuitíve legegyértelműbb megközelítése az objektumok helyettesítésének az x minden előfordulásánál nem működik a példánkban, mert az x-et a *Fivér*(*Richárd*, x)-ben „foglyul ejtené” a behelyettesítés. A kiterjesztett interpretációk ezt pontosabban oldják meg, mert a belső kvantor hozzákapcsolása az x termhez kioltja a külső kvantor hatását.

Created by XMLmind XSL-FO Converter.

Az elsőrendű logika a predikátumok és termek korábban leírt használatán kívül még egy lehetőséget nyújt atomi mondatok készítésére. Alkalmazhatjuk az **egyenlőségszimbólum**ot (**equality symbol**) állítások készítésére is annak kifejezésére, hogy két term ugyanarra az objektumra vonatkozik. Például az

```
Apja(János) = Henrik
```

azt mondja ki, hogy az *Apja*(*János*) által hivatkozott objektum és a *Henrik* által hivatkozott objektum azonos. Mivel az interpretáció rögzíti bármely term referenciáit, egy egyenlőségi állítás igazságának megállapításához elegendő azt látni, hogy a két term referenciái ugyanazok az objektumok.

Az egyenlőségszimbólumot használhatjuk arra, hogy tényeket állapítsunk meg egy adott függvényről, ahogy ezt megtettük az imént az *Apja* szimbólummal. Alkalmazhatjuk az egyenlőségszimbólumot negációval is, annak a kifejezésére, hogy két term nem ugyanaz az objektum. Ha azt akarjuk állítani, hogy Richárdnak legalább két fivére van, ezt írjuk:

```
∃x,y Fivér(x, Richárd) ∧ Fivér(y, Richárd) ∧ ¬(x = y)
```

Annak a mondatnak, hogy

```
∃x,y Fivér(x, Richárd) ∧ Fivér(y, Richárd)
```

nem ugyanaz a szándékolt jelentése. Ez a mondat például igaz a 8.2. ábrán látható modellre, ahol Richárdnak csak egy fivére van. Hogy ezt megértsük, figyeljük meg azt a kiterjesztett interpretációt, ahol mind az *x*, mind az *y* János királyhoz van rendelve. A $\neg(x = y)$ viszont már nem eleme a modellnek. Az $x \neq y$-t néha rövidítésként használjuk a $\neg(x = y)$ kifejezésére.

3. Az elsőrendű logika használata

Most, hogy definiáltunk egy kifejező logikai nyelvet, itt az ideje, hogy megtanuljuk használni is. A legjobb módja ennek az, ha példákon keresztül tesszük. A logikai szintaxis többféle aspektusának illusztrálására már láttunk néhány egyszerű mondatot; ebben az alfejezetben az egyszerű **tárgyterület**eknek (**domain**) több szisztematikus reprezentációját fogjuk bemutatni. A tudásreprezentációban egy tárgyterület a világnak az a része, amelyről valamilyen tudást akarunk kifejezni.

Az elsőrendű tudásbázisokra definiálható `KIJELENT/KÉRDEZ` eljárások rövid leírásával fogunk kezdeni. Ezután megvizsgáljuk a családi kapcsolatok, a számok, a halmazok és a listák tárgyterületét, valamint a wumpus világot. A következő alfejezet egy még alapvetőbb példát (elektronikus áramkörök) tartalmaz, míg a 10. fejezet mindent áttekint az univerzumban.

3.1. Kijelentések és lekérdezések az elsőrendű logikában

A mondatokat a `KIJELENT` használatával adjuk hozzá a tudásbázishoz, pontosan úgy, mint az ítéletlogikában. Ezeket a mondatokat **kijelentés**eknek (**assertion**s) nevezzük. Például, azt jelenthetjük ki, hogy János egy király, és hogy a királyok személyek:

```
Kijelent(TB, Király(János))

Kijelent(TB, ∀x Király(x) ⇒ Személy(x))
```

Kérdéseket tehetünk fel a tudásbázisról a `KÉRDEZ` használatával. Például a:

```
Kérdez(TB, Király(János))
```

igazat ad vissza. A `KÉRDEZ`-zel feltett kérdéseket **lekérdezés**eknek (**queries**) vagy **cél**oknak (**goal**s) nevezzük (ne tévesszük össze azokkal a célokkal, amelyek az ágensek kívánt állapotát írják le). Általánosságban, bármely lekérdezésre, ami logikusan következik a tudásbázisból, igenlő választ kell adni. Például ha adott a két kijelentés az előző bekezdésben, akkor a

```
Kérdez(TB, Személy(János))
```

lekérdezésnek szintén azt kell visszaadnia, hogy *igaz*. Kvantorokat alkalmazó kérdéseket is feltehetünk, úgy mint:

Created by XMLmind XSL-FO Converter.

```
Kérdez(TB, ∃x Személy(x))
```

Erre a lekérdezésre a válasz lehet *igaz*, de ez nem segít, és nem is szórakoztató. (Ez olyasmi, mintha a „Meg tudná mondani mennyi az idő?” kérdésre azt válaszolnánk, hogy „Igen”.) Egy egzisztenciális változókkal ellátott lekérdezés valójában azt a kérdést teszi fel, hogy „Van-e olyan *x*, amely…”, és úgy oldjuk meg, hogy megadunk egy ilyen *x*-et. Egy ilyen típusú válasz szokásos formája egy **helyettesítési lista** (**substitution list**) vagy **lekötési lista** (**binding list**), amely változó/term párok halmaza. Ebben az esetben, amikor csak két kijelentés szerepel, a válasz az {*x*/*János*} lenne. Ha több lehetséges válasz van, akkor a helyettesítések listáját kapjuk vissza.

3.2. A rokonsági tárgyterület

Az első példában a családi kapcsolatokat, vagyis a rokonsági tárgyterületet vizsgáljuk meg. Ez a tárgyterület tartalmaz olyan tényeket, mint „Erzsébet Károly anyja” és „Károly Vilmos apja”, valamint olyan szabályokat, mint „Valakinek a nagyanyja az ő szülőjének az anyja”.

Egyértelmű, hogy ennek a tárgyterületnek az objektumai személyek. Két unáris predikátumunk lesz a *Férfi* és a *Nő*. A rokonsági kapcsolatokat – szülőség, testvérség, házasság és így tovább – bináris predikátumokkal fogjuk ábrázolni: *Szülője*, *Testvére*, *Fivére*, *Nővére*, *Gyereke*, *Lánya*, *Fia*, *Házastársa*, *Felesége*, *Férje*, *Nagyszülője*, *Unokatestvére*, *Nagynénje* és *Nagybátyja*. Az *Anyja* és *Apja* reprezentálására függvényeket fogunk alkalmazni, mivel minden személynek pontosan egy ilyen rokona van (legalább is a természet rendje szerint).

Leírhatjuk a predikátumokat és a függvényeket úgy, hogy más szimbólumokkal fejezzük ki, amit róluk tudunk. Például valakinek az anyja, az ő nőnemű szülője:

$$\forall m,c\ Anyja(c) = m \Leftrightarrow Nő(m) \wedge Szülője(m, c)$$

Valakinek a férje az ő férfi házastársa:

$$\forall w, h\ Férje(h, w) \Leftrightarrow Férfi(h) \wedge Házastársa(h, w)$$

A nő és a férfi diszjunkt kategóriák:

$$\forall x\ Férfi(x) \Leftrightarrow \neg Nő(x)$$

A szülője és a gyereke inverz relációk:

$$\forall p, c\ Szülője(p, c) \Leftrightarrow Gyereke(c, p)$$

Egy nagyszülő a szülője valaki szülőjének:

$$\forall g, c\ Nagyszülője(g, c) \Leftrightarrow \exists p\ Szülője(g, p) \wedge Szülője(p, c)$$

Egy testvér a másik gyereke ugyanannak a szülőnek:

$$\forall x, y\ Testvére(x, y) \Leftrightarrow x \neq y \wedge \exists p\ Szülője(p, x) \wedge Szülője(p, y)$$

Folytathatnánk így tovább, és a 8.11. feladatban kérni is fogjuk, hogy tegye ezt meg.

Minden egyes ilyen mondat tekinthető a rokonsági tárgyterület egy **axiómá**jának (**axiom**). Az axiómákat általában a tisztán matematikai tárgyterületekhez kötjük – röviden meg fogunk vizsgálni néhány axiómát számokra – de valamennyi tárgyterületben szükség van rájuk. Az axiómák ellátnak bennünket alapvető ténybeli információkkal, amelyekből hasznos következtetéseket vonhatunk le. Rokonsági axiómáink egyúttal **definíció**k (**definition**s) is; formájuk pedig $\forall x, y\ P(x, y) \Leftrightarrow \ldots$. Az axiómák meghatározzák az *Anyja* függvényt és a *Férje*, *Férfi*, *Szülője*, *Nagyszülője* és *Testvére* predikátumokat más predikátumok felhasználásával. A predikátumok alapvető halmazainál (*Gyereke*, *Házastársa* és *Nő*) a definícióinkat már nem tudjuk más predikátumokkal leírni. Ez természetes módja annak, ahogyan felépítjük egy tárgyterület reprezentációját, és analóg azzal, ahogyan a szoftvereket felépítik primitív könyvtárfüggvények szubrutinjainak az egymást követő definícióival. Vegyük észre, hogy a predikátumprimitíveknek nincs feltétlenül egyetlen kizárólagos halmaza; éppen ilyen jól használhattuk volna a *Szülője*, *Házastársa* és *Férfi* predikátumokat. Néhány tárgyterületben, mint azt látni fogjuk, nincs egyértelműen meghatározható alaphalmaz.

Created by XMLmind XSL-FO Converter.

Nem minden a tárgyterületről szóló logikus mondat axióma. Néhány közülük **tétel** (**theorem**) – amelyeket az axiómák vonnak maguk után. Például vizsgáljuk meg a kijelentést, hogy a testvérség szimmetrikus:

$\forall x, y \; Testvére(x, y) \Leftrightarrow Testvére(y, x)$

Ez vajon egy axióma vagy egy tétel? Valójában ez egy tétel, amely logikusan következik a testvérséget meghatározó axiómából. Ha ezt a mondatot KÉRDEZ-zük a tudásbázistól, akkor az eljárásnak az *igaz* eredményt kell visszaadnia.

Tisztán logikai nézőpontból egy tudásbázis csak axiómákat, és nem tételeket tartalmaz, mert a tételek nem növelik meg a konklúziók halmazát, amelyeket a tudásbázis maga után vonz. Gyakorlati szempontból a tételek elengedhetetlenek az új mondatok előállításánál a számítási költségek csökkentéséhez. Nélkülük egy bizonyítási rendszernek minden egyes alkalommal az első alapelvektől kellene indulnia, mintha egy fizikusnak mindig újra le kellene vezetnie a számítások szabályait az összes új problémánál.

Nem minden axióma definíció. Néhány axióma általánosabb információkat ír le egyes predikátumokról, anélkül hogy létrehozna egy definíciót. Valójában egyes predikátumoknak nincs teljes definíciójuk, mert nem tudunk eleget róluk ahhoz, hogy teljes mértékben meghatározzuk őket. Például nem létezik egyértelmű befejezése a következő mondatnak:

$\forall x \; Személy(x) \Leftrightarrow \ldots$

Szerencsére az elsőrendű logika lehetővé teszi számunkra, hogy használjuk a *Személy* predikátumot, anélkül hogy teljes mértékben definiálnánk. Ehelyett részleges pontosításokat írhatunk tulajdonságokról, amelyekkel minden személy rendelkezik, és olyan tulajdonságokról, amelyek valamit személlyé tesznek:

$\forall x \; Személy(x) \Rightarrow \ldots$

$\forall x \; \ldots \Rightarrow Személy(x)$

Az axiómák lehetnek „csak puszta tények”, mint amilyenek a *Férfi*(*Jim*) és a *Házastársa*(*Jim, Laura*). Az ilyen tények alkotják a speciális probléma példányok leírásait, lehetővé téve speciális kérdések megválaszolását. Ilyenkor az ezekre a kérdésekre adott válaszok tételek lesznek, amelyek az axiómákból következnek. Gyakran előfordul, hogy az elvárt válasz nem kapható meg – például a *Férfi*(*George*) és a *Házastársa*(*George, Laura*)-ból azt várnánk, hogy kikövetkeztethetőnek kellene lennie annak, hogy *Nő*(*Laura*); de ez nem következik a korábban megadott axiómákból. Ez annak a jele, hogy egy axióma hiányzik. A 8.8. feladat ennek a megadását kéri.

3.3. Számok, halmazok és listák

A számok talán a leglátványosabb példái annak, hogy hogyan építhetünk fel egy hatalmas elméletet axiómák kicsi darabjait használva. Le fogjuk írni a **természetes számok** (**natural numbers**), vagyis a nem negatív egész számok elméletét. Szükségünk van egy predikátumra *TermSzám*, amely igaz lesz minden természetes számra; szükségünk van továbbá egy konstansszimbólumra, ez lesz a 0; valamint egy függvényszimbólumra, az *S* rekurziót leíró függvényre. A **Peano-axiómák** (**Peano axioms**) meghatározzák a természetes számokat és az összeadást.[8] A természetes számoknak egy rekurzív definícióját adjuk meg:

$TermSzám(0)$

$\forall n \; TermSzám(n) \Rightarrow TermSzám(S(n))$

Ez azt jelenti, hogy a 0 egy természetes szám, és hogy minden *n* objektumra, ha az *n* egy természetes szám, akkor $S(n)$ is egy természetes szám. Tehát, a természetes számok a 0, az $S(0)$ és az $S(S(0))$ és így tovább. Szükségünk van axiómákra ahhoz is, hogy lezárjuk a rekurziót leíró függvényt:

$\forall n \; 0 \neq S(n)$

$\forall m, n \; m \neq n \Rightarrow S(m) \neq S(n)$

Most meg tudjuk határozni az összeadást az utódfüggvény szempontjából:

[8] A Peano-axiómák még tartalmazzák az indukció elvét is, amely inkább a másodrendű logikába tartozó állítás, mintsem az elsőrendű logikába. Ennek a megkülönböztetésnek a leírását lásd a 9. fejezetben.

Created by XMLmind XSL-FO Converter.

$\forall m\ TermSzám(m) \Rightarrow +(0, m) = m$

$\forall m, n\ TermSzám(m) \wedge TermSzám(n) \Rightarrow +(S(m), n) = S(+(m, n))$

Az első axióma azt mondja ki, hogy ha hozzáadunk egy 0-t bármely *m* természetes számhoz, akkor *m*-et kapunk. Figyeljük meg a „+” bináris függvényszimbólum használatát a +(*m*, 0) termben. A matematika jelölése szerint a termet a következőképpen írnánk: *m* + 0, **infix** (**infix**) jelölés használatával. (Az eddigiekben az elsőrendű logikához használt jelöléseket **prefix**nek (**prefix**) nevezzük.) Azért, hogy könnyebben olvashatóvá tegyük a számokról szóló mondatainkat, lehetővé tesszük az infix jelölések használatát. Így az *S*(*n*)-t leírhatjuk úgy is, hogy *n* + 1, ami által a második axióma a következő lesz:

$\forall m, n\ TermSzám(m) \wedge TermSzám(n) \Rightarrow (m + 1) + n = (m + n) + 1$

Ez az axióma az utódfüggvény ismételt alkalmazásává egyszerűsíti az összeadást.

Az infix jelölések használata példa a **szintaktikus édesség**re (**syntactic sugar**), ami nem más, mint az alapszintaxis kiterjesztése vagy rövidítése, a szemantika megváltoztatása nélkül. Bármely mondat, amely ezt a szintaktikai édességet használja visszaalakítható, egyenértékű mondatot előállítva a hagyományos elsőrendű logikában.

Mihelyt van egy összeadás-definíciónk, magától értetődik a szorzást ismételt hozzáadásokként definiálni, a hatványozást ismételt szorzásokként, meghatározni az egész számok osztását és maradékait, a prímszámokat és így tovább. Így tehát a teljes számelmélet (beleértve a titkosírást) felépíthető egy konstansból, egy függvényből, egy predikátumból és négy axiómából.

A **halmaz**ok (**set**s) tárgyterületének szintén alapvető szerepe van a matematikában, ugyanúgy mint a hétköznapi gondolkodásban is. (Valójában a számelmélet levezethető a halmazelméletből kiindulva.) Halmazokat szeretnénk reprezentálni, beleértve az üres halmazt is. Szükségünk van a halmazok felépítésének egy módjára, amelyben vagy hozzá tudunk adni egy elemet a halmazhoz, vagy két halmaz metszetét vagy unióját tudjuk képezni. Meg kell tudnunk állapítani, hogy egy elem része-e egy halmaznak, és szeretnénk megkülönböztetni a halmazokat az olyan objektumoktól, amelyek nem halmazok.

A halmazelmélet hagyományos szókincsét fogjuk használni szintaktikai édességként. Az üres halmaz egy konstans, amelynek a jele: { }. Definiálunk egy unáris predikátumot, a *Halmaz*-t, amely minden halmaz esetében igaz. A bináris predikátumok az $x \in s$ (az *x* az *s* halmaz része) és az $s_1 \subseteq s_2$ (az s_1 halmaz egy részhalmaza, nem feltétlenül valódi részhalmaza az s_2-nek). A bináris függvények az $s_1 \cap s_2$ (két halmaz metszete), az $s_1 \cup s_2$ (két halmaz uniója) és az $\{x|\ s\}$ (a halmaz az *x* elem *s* halmazhoz való csatlakozásának eredménye). Az axiómák egy lehetséges sorozata a következő:

1. Egy halmaz vagy az üres halmaz, vagy azok, amelyeket egy halmaz bővítésével hoztunk létre:

$\forall s\ Halmaz(s) \Leftrightarrow (s = \{\}) \vee (\exists x, s_2\ Halmaz(s_2) \wedge s = \{x|\ s_2\})$

2. Az üres halmaznak nincs hozzáadott eleme, más szóval nincs lehetőség arra, hogy az *ÜresHalmaz*-t szétválasszunk egy kisebb halmazra és egy elemre:

$\neg\exists x, s\ \{x|s\} = \{\}$

3. Egy halmazbeli elem újbóli hozzáadása a halmazhoz nem változtat a halmazon.

$\forall x, s\ x \in s \Leftrightarrow s = \{x|s\}$

4. Egy halmaznak csak az az eleme, amit hozzáadunk. Ezt rekurzívan fejezzük ki, kimondva, hogy *x* akkor és csakis akkor eleme az *s* halmaznak, ha *s* azonos valamilyen s_2 halmaz *y* elemmel való bővítésével, ahol vagy azonos *y* *x*-szel, vagy *x* eleme s_2-nek:

$\forall x, s\ x \in s \Leftrightarrow [\exists y, s_2\ (s = \{y|s_2\} \wedge (x = y \vee x \in s_2))]$

5. Egy halmaz egy másiknak a részhalmaza, ha az összes első halmazbeli elem eleme a második halmaznak:

$\forall s_1, s_2\ s_1 \subseteq s_2 \Leftrightarrow (\forall x\ x \in s_1 \Rightarrow x \in s_2)$

6. Két halmaz akkor és csakis akkor egyenlő, ha egymásnak részhalmazai:

Created by XMLmind XSL-FO Converter.

$\forall x, s_1, s_2 \ (s_1 = s_2) \Leftrightarrow (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1)$

7. Egy objektum akkor és csakis akkor eleme két halmaz metszetének, ha eleme mindkét halmaznak:

$\forall x, s_1, s_2 \ x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \wedge x \in s_2)$

8. Egy objektum akkor és csakis akkor eleme két halmaz uniójának, ha a két halmaz bármelyikének az eleme:

$\forall x, s_1, s_2 \ x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2)$

A **listá**k (**list**s) tárgyköre nagyon hasonló a halmazok tárgyköréhez. A különbség az, hogy a listák rendezettek, és ugyanaz az elem több helyen is előfordulhat egy listában. Használhatjuk a Lisp szótárát a listákra: a *Nil* az elem nélküli listát jelölő konstans, a *Cons*, az *Append*, a *First* és a *Rest* függvények. A *Find* egy predikátum, amely azt hajtja végre a listáknál, mint az *Eleme* a halmazoknál. A *List*? egy predikátum, amely csak a listákra igaz. Úgy, mint a halmazoknál, a listákról szóló logikai mondatokban is szokás használni a szintaktikai édességet. Az üres lista a: [ ]. A *Cons*(*x*, *y*) termet, ahol az *y* nem üres lista, úgy írjuk, hogy: [*x*|*y*]. A C*ons*(*x*, *Nil*) termet (egy lista, amely tartalmazza az *x* elemet) úgy írjuk, hogy: [*x*]. Több elem listája, mint az [*A*, *B*, *C*], ugyanazt jelenti, mint az egymásba ágyazott term: *Cons*(*A*, *Cons*(*B*, *Cons*(*C*, *Nil*))). A 8.14. feladat tárgya az axiómák megírása a listák tárgyköréhez.

3.4. A wumpus világ

A 7. fejezetben megadtunk néhány ítéletlogikai axiómát a wumpus világra. Az ebben az alfejezetben szereplő elsőrendű logika axiómái sokkal tömörebbek lesznek, természetes módon ábrázolva azt, amit pontosan mondani szeretnénk.

Emlékezzünk arra, hogy egy wumpus ágens egy öt elemből álló érzetvektort kap. Az ezt leíró elsőrendű logikai mondat, amelyet a tudásbázisban tárolunk, tartalmazza mind az érzetet, mind az időpontot amikor ez bekövetkezett, különben az ágens összekeverné, hogy mikor látta azt, amit látott. Az időpontokat jelölő lépésekre egész számokat használunk. Egy tipikus érzetmondat ilyen lesz:

Érzet([*Bűz*, *Szelő*, *Ragyogás*, *Nincs*, *Nincs*], 5)

Itt az *Érzet* egy bináris predikátum, míg a *Bűz* és a többi érzet listába rendezett konstansok. A wumpus világban lejátszódó cselekvéseket a következő logikai termekkel reprezentálhatjuk:

Fordul(*Jobbra*), *Fordul*(*Balra*), *Előre*, *Lő*, *Megragad*, *Elereszt*, *Mászik*

Hogy meghatározzuk, melyik cselekvés a legjobb az adott helyzetben, az ágens program létrehoz egy lekérdezést, mint például:

∃*a LegjobbCselekvés*(*a*, 5)

A KÉRDEZ-zel végrehajtjuk ezt a lekérdezést, és vissza fogunk kapni egy lekötési listát, mint például: {*a*/*Megragad*}. Az ágensprogram ezután visszaadhatja a *Megragad*-ot, mint elvégzendő cselekvést, de először a KIJELENT felhasználásával közölnie kell a saját tudásbázisával, hogy egy *Megragad*-ot hajt végre.

A nyers érzetadatok alapján következtethetünk az aktuális állapot bizonyos tényeiről. Például:

$\forall t, s, g, m, c$ *Érzet*([*s*, *Szellő*, *g*, *m*, *c*], *t*) ⇒ *Szellő*(*t*)

$\forall t, s, g, m, c$ *Érzet*([*s*, *Ragyogás*, *m*, *c*], *t*) ⇒ *Ragyogás*(*t*)

és így tovább. A szabályok a következtetési folyamat egy triviális formáját mutatják, amit **érzékelés**nek (**perception**) nevezünk, és amelyet a 24. fejezetben fogunk részletesen tanulmányozni. Vegyük észre az univerzális kvantort az idő (*t*) felett. Az ítéletlogikában minden egyes időlépéshez szükségünk volt a mondatok másolatára.

Az egyszerű „reflex” viselkedés is leírható kvantorokat alkalmazó implikációs mondatokkal. Például lehet egy ilyen mondatunk:

$\forall t$ *Ragyogás*(*t*) ⇒ *LegjobbCselekvés*(*Megragad*, *t*)

Created by XMLmind XSL-FO Converter.

Mivel adottak az érzet és az előző bekezdésekben megismert szabályok, ez a mondat a kívánt konklúziót eredményezné: *LegjobbCselekvés*(*Megragad*, 5) – ami azt jelenti, hogy a *Megragad* a legjobb cselekvés, amit végrehajthatunk. Vegyük észre az egybeesést a 7.20. ábrában az áramkörön alapuló ágensnél e szabály és a közvetlen érzet-cselekvés kapcsolat között. Az áramkör jellegéből következően *impliciten* látja el kvantorral az időt.

Eddig ebben az alfejezetben az idővel operáló mondatok mind **szinkrón**ok (**synchronic**), azaz egyidejűek voltak, ami azt jelenti, hogy a világ állapotának tulajdonságait vonatkoztatják ugyanannak a világállapotnak más tulajdonságaihoz. Azokat a mondatokat, amelyek lehetővé teszik az időbeliséget figyelembe vevő következtetést **diakrón**nak (**diachronic**) nevezzük. Például az ágensnek tudnia kell, hogy hogyan kombinálja a megelőző helyzetéről lévő információt azzal az információval, amely helyzetének meghatározásához szükséges. A diakrón mondatok tárgyalását elhalasztjuk a 10. fejezetig, most csak tételezzük fel, hogy a helyzetre és más időfüggő predikátumokra vonatkozó következtetések megtörténtek.

Bemutattuk az érzeteket és a cselekvéseket, ezután itt az ideje, hogy bemutassuk magát a környezetet. Kezdjük az objektumokkal. A négyzetek, a csapdák és a wumpus nyilvánvaló szereplők. Elnevezhetnénk minden egyes négyzetet – $Négyzet_{1,2}$ és így tovább – de akkor az a tény, hogy a $Négyzet_{1,2}$ szomszédos a $Négyzet_{1,3}$-mal, egy extra tényt igényelne, és minden egyes négyzetpárnál szükségünk lenne egy ilyen tényre. Jobb egy komplex termet használni, amelyben a sor és az oszlop egész számokként jelenik meg; például egyszerűen használhatjuk a lista termet: [1, 2]. Bármely két négyzet szomszédossága meghatározható így:

$$\forall x, y, a, b \; Szomszédos([x, y], [a, b]) \Leftrightarrow$$

$$[a, b] \in \{[x + 1, y], [x - 1, y], [x, y + 1], [x, y - 1]\}$$

Minden egyes csapdát elnevezhetnénk, de ez sem lenne megfelelő, méghozzá más okból: nincs ok arra, hogy megkülönböztetést tegyünk a csapdák között.[9] Sokkal egyszerűbb használni egy *Csapda* unáris predikátumot, amely igaz a csapdát tartalmazó négyzetekre. Végül, mivel pontosan egy wumpus van csak, ezért egy *Wumpus* konstans éppen úgy megfelelő, mintha egy unáris predikátummal jelölnénk (és a wumpus szempontjából talán tiszteletre méltóbb is). A wumpus pontosan egy négyzetben lakik, tehát jó gondolat egy függvényt használni ennek a négyzetnek a megnevezésére, ami az *Otthon*(*Wumpus*). Ez kiküszöböli azt a számos mondatot, amelyre az ítéletlogikában szükségünk van ahhoz, hogy megmondjuk, pontosan egyetlen négyzet tartalmaz wumpust. (Még nehezebb helyzetet teremtene az ítéletlogika számára, ha két wumpus volna.)

Az ágens helyzete változik az idő haladtával, amit úgy jelölünk, hogy *Akkor*(*Ágens*, *s*, *t*), ami azt jelenti, hogy az ágens az *s* négyzetnél van *t* időben. Mihelyt megadtuk az aktuális helyzetét, az ágens következtetni tud a négyzet tulajdonságaira a jelenlegi érzete tulajdonságaiból. Például, ha az ágens egy négyzetnél van, és szellőt érzékel, akkor az a négyzet szellős:

$$\forall s, t \; Akkor(Ágens, s, t) \wedge Szellő(t) \Rightarrow Szellős(s)$$

Hasznos tudni, hogy egy *négyzet* szellős, mert tudjuk, hogy a csapdák nem tudnak mozogni. Vegyük észre, hogy a *Szellős*-nek nincsen időargumentuma.

Miután felfedeztük, mely helyek szellősek (vagy büdösek) vagy – és ez nagyon fontos – *nem* szellősek (vagy *nem* büdösek), az ágens ki tudja következtetni, hogy hol vannak a csapdák (és hogy hol van a wumpus). Kétfajta szinkrón szabály van, amely lehetővé teszi az ilyen következtetéseket:

- Diagnosztikus szabályok

A diagnosztikus szabályok a megfigyelt hatásoktól a rejtett okokhoz vezetnek minket. A csapdák megtalálásához a legkézenfekvőbb diagnosztikai szabályok azt mondják ki, hogyha a négyzet szellős, akkor valamelyik szomszédos négyzet biztosan tartalmaz csapdát:

$$\forall s \; Szellős(s) \Rightarrow \exists r \; Szomszédos(r, s) \wedge Csapda(r)$$

és hogyha egy négyzet nem szellős, akkor egyetlen szomszédos négyzet sem tartalmaz csapdát:[10]

[9] Hasonlóan, legtöbbünk nem nevezi meg a madarat, amint elszáll fejünk felett, hogy melegebb vidékre költözzön télen. Egy ornitológus viszont, aki a vonulási mintákat és a túlélési arányokat szeretné tanulmányozni, megnevez minden egyes madarat azáltal, hogy meggyűrűzi őket, mivel a madarakat egyenként kell nyomon követnie.

[10] Az emberben van hajlandóság arra, hogy elfelejtse leírni az ehhez hasonló negatív információkat. Egy beszélgetés során ez a tendencia teljesen normális – furcsa lenne, ha azt mondanánk: „Van két csésze az asztalon, és nincs ott három vagy több.” Még akkor is, ha a „Van két csésze az asztalon” szigorúan szólva akkor is igaz, ha három van ott. A 10. fejezetben visszatérünk erre a problémára.

Created by XMLmind XSL-FO Converter.

$\forall s \; \text{Szellős}(s) \Rightarrow \neg\exists r \; \text{Szomszédos}(r, s) \wedge \text{Csapda}(r)$

E kettő kombinálásával, ezt a bikondicionális mondatot kapjuk:

$\forall s \; \text{Szellős}(s) \Leftrightarrow \exists r \; \text{Szomszédos}(r, s) \wedge \text{Csapda}(r) \quad (8.3)$

- Ok-okozati szabályok

Az ok-okozati szabályok a világban lévő okozatiság feltételezett irányát tükrözik: a világ néhány rejtett tulajdonsága bizonyos érzetek generálását eredményezi. Például egy csapda azt eredményezi, hogy az összes szomszédos négyzet szellős lesz:

$\forall r \; \text{Csapda}(r) \Rightarrow [\forall s \; \text{Szomszédos}(r, s) \Rightarrow \text{Szellős}(s)]$

és ha az adott négyzettel szomszédos négyzetek egyike sem tartalmaz csapdát, a négyzet nem lesz szellős:

$\forall s \; [\forall r \; \text{Szomszédos}(r, s) \Rightarrow \neg\text{Csapda}(r)] \Rightarrow \neg\text{Szellős}(s)$

Némi munkával meg lehet mutatni, hogy ezek a mondatok együtt logikusan egyenértékűek a (8.3) egyenletben leírt ekvivalenciamondatokkal. Maga az ekvivalencia is tekinthető ok-okozatinak, mert megmutatja, hogy a *Szellős* igazságértéke a világ állapota alapján határozható meg.

Azokat a rendszereket, amelyek ok-okozati szabályokat alkalmaznak **modellalapú következtető** (**modell-based reasoning**) rendszereknek nevezzük, mivel az ok-okozati szabályok a környezet működésének modelljét alkotják. A modellalapú és a diagnosztikus következtetés közötti megkülönböztetés a mesterséges intelligencia számos területén fontos. Az orvosi diagnosztika egy különösen aktív területe a kutatásnak, amelyben a tünetek és a betegségek közötti közvetlen asszociációkon alapuló (diagnosztikai megközelítések) megközelítéseket fokozatosan felváltották az olyan módszerek, amelyek a betegség kialakulásának és a betegségtünetek manifesztálódásának egy explicit modelljét használják. Ez a téma a 13. fejezetben újra előkerül.

Fontos

Bármelyik reprezentációs formát is használja az ágens, *ha az axiómák pontosan és teljesen leírják a világ működésének a módját és azt a módot, ahogyan az érzetek létrejönnek, akkor bármely teljes logikai következtetési eljárás az elérhető érzetek megadása után ki fogja következtetni a világ állapotának legvalószínűbb lehetséges leírását.* Így tehát az ágenstervező arra koncentrálhat, hogy a tudást helyesen adja meg, anélkül hogy sokat kellene töprengenie a következtetések folyamatain. Láttuk továbbá azt is, hogy az elsőrendű logika hasonlóan tömören tudja reprezentálni a wumpus világot, mint a 7. fejezetben megadott eredeti magyar nyelvű leírás.

4. Tudástervezés az elsőrendű logikában

A megelőző alfejezet bemutatta az elsőrendű logika használatát a tudás reprezentálására három egyszerű tárgyterületben. Ez az alfejezet a tudásbázis felépítésének általános folyamatát írja le – egy folyamatot, amit **tudástervezés**nek (**knowledge engineering**) nevezünk. A tudásmérnök egy olyan személy, aki egy bizonyos tárgyterületet vizsgál, megismeri, hogy mely koncepciók fontosak abban a tárgyterületben, és megalkot egy formális reprezentációt a tárgyterületben található objektumokra és relációkra. A tudástervezés folyamatát az elektronikus áramkör tárgyterületében fogjuk illusztrálni, amely már valószínűleg ismerős, így koncentrálhatunk az ezzel járó reprezentációs problémákra. Az általunk alkalmazott megközelítés *speciális célú* tudásbázisok kialakításához megfelelő, amelyek tárgyterülete alaposan körülhatárolt, és amelyek lekérdezéseinek az egész sorozatát előre ismerjük. Általános *célú* tudásbázisokat, amelyeket arra használunk, hogy lekérdezéseket tegyenek lehetővé az emberi tudás teljes területére vonatkozóan, a 10. fejezetben tárgyaljuk majd meg.

4.1. A tudástervezés folyamata

A tudástervezési projektek különbözők tárgyukat, tárgykörüket és nehézségüket tekintve, de minden ilyen projekt tartalmazza a következő lépéseket:

1. *A feladat beazonosítása.* A tudásmérnöknek fel kell vázolnia a kérdések sorát, amelyekkel a tudásbázis foglalkozni fog, és a tényeknek azokat a csoportjait, amelyek minden egyes problémaspecifikus példányban megtalálhatók lesznek. El kell döntenie például, hogy a wumpus tudásbázisnak képesnek kell-e lennie a

Created by XMLmind XSL-FO Converter.

cselekvések kiválasztására, vagy hogy csak az várható el, hogy a környezet elemeivel kapcsolatos kérdéseket válaszolja meg. Tudnia kell, hogy az érzékelőktől származó tények leírják-e a jelenlegi helyzetet. A feladat határozza meg, hogy mely tudást kell tárolni, hogy a problémapéldányokban a válaszokat megadhassuk az adott esetre vonatkozóan. Ez a lépés analóg az ágensek tervezésénél látott TKBÉ-folyamattal, amelyről a 2. fejezetben írtunk.

2. *A releváns tudás összegyűjtése.* A tudásmérnök vagy már szakértője a tárgyterületnek, vagy együtt kell működnie igazi szakértőkkel, hogy megismerje az ő tudásukat – ezt a folyamatot **tudásmegszerzés**nek (**knowledge acquisition**) nevezzük. Ezen a szinten a tudást formálisan nem reprezentáljuk. A cél az, hogy megértsük a tudásbázis tárgykörét, amit a feladat határol be, és meg kell érteni azt is, hogy a tárgyterület hogyan működik a gyakorlatban.

A wumpus világban, amelyet mesterségesen létrehozott szabályrendszer határoz meg, könnyű az idevonatkozó tudásbázist azonosítani. (Vegyük észre azonban, hogy a szomszédosság definíciója nem volt explicit megadva a wumpus világ szabályaiban.) A valódi tárgyterületekben a relevancia problémája meglehetősen bonyolult lehet – például egy VLSI tervező szimulációs rendszernek figyelembe kell vennie a szórt kapacitásokat és a felületi hatásokat.

3. *Meg kell határozni a predikátumok, függvények és konstansok szótárát.* Ez azt jelenti, hogy a fontos tárgyterület szintű koncepciókat le kell fordítani logikai szintű nevekre. Ez számos, a megközelítés *jellegét* tárgyaló kérdést érint. Hasonlóan, mint a programozási stílusnak, ennek is jelentős hatása lehet a projekt végső sikerére. Például ilyen kérdés, hogy a csapdákat vajon objektumok jelenítsék-e meg vagy egy a négyzetekre vonatkozó unáris predikátum? Az ágens iránya függvény legyen-e vagy predikátum? A wumpus helyzete az időtől függjön-e? Mihelyt a választás megtörtént, az eredmény egy szótár, amit a tárgyterület **ontológiá**jának (**ontology**) nevezzünk. Az *ontológia* fogalma egy olyan elméletet takar, ami a létezés természetét írja le. Meghatározza, hogy milyen dolgok léteznek, de nem határozza meg a rájuk jellemző tulajdonságokat, sem a köztük fennálló kapcsolatokat.

4. *A tárgyterületről szóló általános tudás kódolása.* A tudásmérnök leírja a szótár összes termjéhez tartozó axiómákat. Ez lerögzíti (amennyire lehetséges) a termek jelentését, és lehetővé teszi a szakértő számára a tartalom ellenőrzését. Ez a lépés gyakran feltárja a félreértelmezéseket vagy a hiányosságokat a szótárban, amelyeket a 3. lépéshez újra és újra visszatérve, iteratív eljárással javíthatunk.

5. *Az adott probléma példány leírásának kódolása.* Ha az ontológia jól átgondolt, ez a lépés már könnyű lesz. Egyszerű atomi mondatokat kell az ontológiában már leírt fogalmak példányaira megfogalmazni. Egy logikai ágens számára a problémapéldányokat az érzékelők biztosítják, amikor a „különálló” tudásbázist kiegészítik mondatokkal ugyanúgy, ahogy a hagyományos programoknak bemeneti adatokat adunk meg.

6. *Lekérdezéseket fogalmazunk meg a következtetési folyamat számára és válaszokat vezetünk le.* Itt kapjuk meg az eddigi munkánk jutalmát: működtethetjük a következtetési folyamatot az axiómákon és a problémaspecifikus tényeken, hogy megkapjuk a minket érdeklő tényeket.

7. *Szűrjük ki a hibákat a tudásbázisból.* Sajnos az első próbálkozásra a kérdésekre kapott válaszok nagyon ritkán lesznek helyesek. Pontosabban, a válaszok helyesek lesznek *a megadott tudásbázis szempontjából,* feltételezve, hogy a következtetési folyamat megfelelő, de a válaszok nem azok lesznek, amiket a felhasználó vár. Például ha hiányzik egy axióma, akkor bizonyos kérdések megválaszolhatatlanok lesznek a tudásbázis alapján. Ilyenkor egy hibajavítási folyamatra van szükség. A hiányzó vagy túl gyenge axiómák könnyen megtalálhatók úgy, hogy felfedezzük azokat a helyeket, ahol a következtetés lánca váratlanul megszakad. Például ha a tudásbázis tartalmazza az egyik, csapdákra vonatkozó diagnosztikus axiómát,

$$\forall s\ Szellős(s) \Rightarrow \exists r\ Szomszédos(r, s) \wedge Csapda(r)$$

de nem tartalmazza a másikat, akkor az ágens soha nem lesz képes bizonyítani a csapdák *hiányát.* A helytelen axiómák azonosíthatók, mivel ezek hamis állítások a világról. Például az a mondat, hogy:

$$\forall x\ LábakSzáma(x, 4) \Rightarrow Emlős(x)$$

hamis a hüllőkre, kétéltűekre vagy ami még fontosabb, az asztalokra nézve.

Fontos

Created by XMLmind XSL-FO Converter.

Ennek a mondatnak a hamissága a tudásbázis többi részétől függetlenül meghatározható. Ezzel szemben egy tipikus programhiba ilyen:

```
eltolás = pozíció + 1
```

Nem lehet ez alapján megmondani, hogy a mondat helyes-e anélkül, hogy megnéznénk a program többi részét. Például azt, hogy az eltolás-t a jelenlegi pozícióra vonatkozóan használjuk-e, vagy arra, amely eggyel a jelenlegi pozíció mögött van, vagy arra, hogy a pozíció értéke megváltozott egy másik állítás által, és így az eltolás-t is meg kell változtatni.

Hogy jobban megértsük ezt a hétlépéses folyamatot, alkalmazzuk most egy kiterjesztett példára – az elektronikus áramkörök tárgyterületére.

4.2. Az elektronikus áramkörök tárgyterülete[11]

Létrehozunk egy ontológiát és egy tudásbázist, amelynek felhasználásával képesek leszünk következtetéseket végezni olyan típusú digitális áramkörökről, mint amilyet a 8.4. ábra mutat. A tudástervezés hétlépéses folyamatát fogjuk követni.

4.3. A feladat meghatározása

Számos következtetési feladatot lehet a digitális áramkörökkel kapcsolatban elvégezni. A legmagasabb szinten az áramkör funkcionalitását vizsgálhatjuk. Például szabályosan ad-e össze a 8.4. ábrán látható áramkör? Ha minden bemenet magas, akkor mi az A_2 kapu kimenetének állapota? Az áramkör szerkezetéről feltett kérdések is érdekesek. Például melyek azok a kapuk, amelyek az első bemeneti ponthoz vannak kapcsolva? Tartalmaz-e az áramkör visszacsatolásokat? Ebben az alfejezetben ezeket a feladatokat vizsgáljuk meg. Léteznek az elemzésnek részletesebb szintjei is, például amelyek az időzítés késleltetésével, az áramkör területével, áramfogyasztásával, működtetési költségeivel és így tovább kapcsolatosak. Minden egyes ilyen szint vizsgálata további ismereteket igényel.

4.4. A releváns tudás összegyűjtése

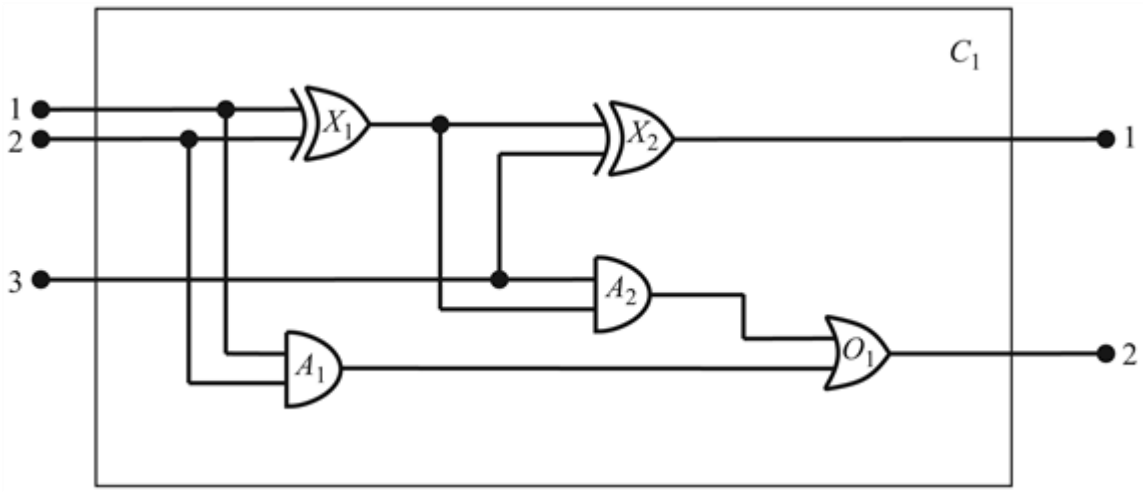
Mit tudunk a digitális áramkörökről? A céljainknak megfelelően azt, hogy vezetékekből és kapukból állnak. A jelek a vezetékeken keresztül áramlanak a kapuk bemeneti termináljához, és minden kapu egy jelet hoz létre a kimeneti terminálon, ami aztán egy másik vezetéken áramlik. Ahhoz, hogy meghatározzuk, melyek is ezek a jelek, ismernünk kell, hogy a kapuk hogyan alakítják át bemeneti jeleiket. Négyféle kaput használunk: az AND, az OR és a XOR kapuknak két bemenetük van, míg a NOT kapuknak csak egy. Minden kapunak egy kimenete van. Az áramköröknek, hasonlóan a kapukhoz, bemeneteik és kimeneteik is vannak.

Ahhoz, hogy következtetéseket végezhessünk a funkcionalitásról és az összeköttetésekről, nem szükséges magukról a vezetékekről vagy ezek útvonalairól, két vezeték találkozásánál levő kereszteződéseiről tudást megfogalmazni. Csak a be- és kimenetek közötti összeköttetések számítanak – tehát csak azt kell kimondani, hogy egy kimenet össze van-e kapcsolva egy másik bemenettel, anélkül hogy meg kellene említeni a vezetékeket, amik valójában összekötik őket. A tárgyterületnek sok más tényezője van, ami a mi vizsgálatunkban nem releváns. Ilyen például a különböző elemek mérete, formája, színe vagy ára.

8.4. ábra - Egy C_1-es digitális áramkör, amelynek az a célja, hogy egy egybites teljes összeadást végezzen. Az első két bemenet az a két bit, amit össze kell adni, míg a harmadik bemenet az átvitel. Az első kimenet az összeg, míg a második kimenet az átvitel a következő összeadó felé. Az áramkör két XOR, két AND és egy OR kaput tartalmaz.

[11] Ez az alfejezet a digitális áramkörök meglehetősen sajátos bemutatására vállalkozik. Miközben nem javasoljuk, hogy valaki ez alapján ismerkedjen meg a digitális áramkörökkel, és esetleg az itt bemutatottak alapján próbáljon digitális áramköröket tervezni, az itt leírtak mindenképpen érdekes nézőpontot tükröznek, és érdekes kísérletet jelentenek arra, hogy az elsőrendű logika fogalomkészletével mutassák be a digitális áramköröket. (*A szerk.*)

Created by XMLmind XSL-FO Converter.



Ha a célunk valami más lenne, és nem a kapuszintű tervezés helyességének ellenőrzése, akkor az ontológiánk is más lenne. Például ha az érdekelne minket, hogyan lehetne a hibákat a hibás áramkörökben megtalálni, akkor valószínűleg jó ötlet lenne a vezetékekkel is foglalkozni az ontológiában, mivel egy hibás vezeték meghamisíthatja a rajta keresztülhaladó jelet. Az időzítési hibák megtalálásához a kapuk késleltetésével kapcsolatos fogalmakat kellene leírni. Ha az érdekelne minket, hogy hogyan lehet egy nyereséges terméket tervezni, akkor az áramkörök költségének és sebességének a piacon jelen lévő egyéb termékekkel történő összehasonlíthatósága lenne fontos.

4.5. A szótár meghatározása

Tudjuk, hogy áramkörökről, be- és kimenetekről, jelekről és kapukról akarunk beszélni. A következő lépés az ezeket reprezentáló függvények, predikátumok és konstansok kiválasztása. Az egyes kaputípusoktól fogunk indulni, és végül eljutunk az áramkörökig.

Először is meg kell tudnunk különböztetni egy kaput a többi kaputól. Ezt úgy érjük el, hogy konstansokat használunk a kapuk megnevezésére: X_1, X_2 és így tovább. Habár minden kapu a maga egyedi módján kapcsolódik az áramkörhöz, a *viselkedése* – vagyis az a mód, ahogyan átalakítja a bemeneti jeleket kimeneti jelekké – csakis a *típusától* függ. Egy kapu típusának jelölésére[12] egy függvényt használhatunk. Például írhatjuk, hogy *Típus*(X_1) = *XOR*. Ez hozzárendeli az *XOR* konstanst egy bizonyos kaputípushoz. A többi konstans nevei: *OR*, *AND* és *NOT* lesznek. A *Típus* függvény nem az egyetlen lehetséges módja annak, hogy kódoljuk az ontológiai megkülönböztetést. Használhattunk volna egy bináris predikátumot is, mint a *Típus*(X_1, *XOR*), vagy több egyargumentumú predikátumot, mint például az *XOR*(X_1). Ezeknek a megoldásoknak bármelyike jól működne, de a *Típus* függvény választásával kiküszöböltük, hogy szükség legyen egy olyan axiómára, amely azt mondja ki, hogy minden egyes kapunak csak egyetlen típusa lehet.

Ezután megvizsgáljuk a végpontokat (be- és kimeneteket). Egy kapunak vagy áramkörnek egy vagy több bemenete, és egy vagy több kimenete lehet. Mindegyiket elnevezhetnénk egyszerűen egy konstanssal, mint ahogy azt a kapukkal tettük. Így az X_1 kapunak olyan végpontjai lennének, mint az X_1Be_1, az X_1Be_2 és az X_1Ki_1. A hosszú, összetett elnevezéseket azonban célszerű kerülni. Az, hogy valamit X_1Be_1-nek nevezünk, nem jelenti azt, hogy ez az X_1 első bemenete; még ekkor is hozzá kell tennünk valamit egy explicit állítást használva. Valószínűleg szerencsésebb egy függvénnyel leírni egy kaput, hasonlóan mint, ahogy János király bal lábát elneveztük *BalLáb*(*János*)-nak. Így tehát jelöljük az X_1 kapu első bemenetét úgy, hogy: *Be*(1, X_1). Egy hasonló *Ki* függvényt használunk a kimenetekre.

A kapuk közötti összeköttetést reprezentálhatjuk az *Összekapcsolt* predikátummal, ami két végpontot vesz argumentumként, például így: *Összekapcsolt*(*Ki*(1, X_1), *Be*(1, X_2)).

Végül, ismernünk kell, hogy egy jel magas vagy alacsony állapotban van-e. Erre egy lehetőség egy *On* bináris predikátum használata, és akkor igaz, ha a jel egy végponton magas értékű. Ez azonban egy kissé megnehezíti az olyan kérdések feltevését, mint például: „Mik a lehetséges értékei a C_1 áramkör kimenetein lévő jeleknek?”

[12] Vegyük észre, hogy megfelelő betűkkel kezdődő neveket használtunk – A_1, X_1 és így tovább – pusztán azért, hogy könnyebben olvashatóvá tegyük a példát. A tudásbázisnak még így is tartalmaznia kell a kapuk típusára vonatkozó információkat.

Created by XMLmind XSL-FO Converter.

Ezért be fogunk vezetni objektumokként két „jelértéket”, az 1-et és a 0-t, valamint egy *Jel* függvényt, aminek egy végpont az argumentuma, és ami kijelöli ennek a végpontnak a jelértékét.

4.6. A tárgyterülettel kapcsolatos általános tudás kódolása

Az egyik jele annak, hogy megfelelő ontológiát használunk az, hogy kevés olyan általános szabály van, amit később a példányokra specifikussá kellene tennünk. A helyes szótár jellemzője az, hogy minden egyes szabályt világosan és tömören meg tudunk fogalmazni. A mi példánkban csak hét egyszerű szabályra van szükségünk, hogy leírjunk mindent, amit tudnunk kell az áramkörökről.

1. Ha két végpont össze van kapcsolva, akkor ugyanaz lesz a jelértékük:

$$\forall t_1, t_2 \; Összekapcsolt(t_1, t_2) \Rightarrow Jel(t_1) = Jel(t_2)$$

2. A jel minden végpontnál vagy 1, vagy 0 (de soha nem mindkettő):

$$\forall t_1 \; Jel(t) = 1 \vee Jel(t) = 0$$

$$1 \neq 0$$

3. Az Összekapcsolt egy felcserélhető (kommutatív) predikátum:

$$\forall t_1, t_2 \; Összekapcsolt(t_1, t_2) \Leftrightarrow Összekapcsolt(t_1, t_2)$$

4. Egy OR kapu kimenete akkor és csak akkor 1, ha bármelyik bemenete 1:

$$\forall g \; Típus(g) = OR \Rightarrow$$

$$Jel(Ki(1, g)) = 1 \Leftrightarrow \exists n \; Jel(Be(n, g)) = 1$$

5. Egy AND kapu kimenete akkor és csakis akkor 0, ha bármelyik bemenete 0:

$$\forall g \; Típus(g) = AND \Rightarrow$$

$$Jel(Ki(1, g)) = 0 \Leftrightarrow \exists n \; Jel(Be(n, g)) = 0$$

6. Egy XOR kapu kimenete akkor és csakis akkor 1, ha a bemenetei különbözők:

$$\forall g \; Típus(g) = XOR \Rightarrow$$

$$Jel(Ki(1, g)) = 1 \Leftrightarrow Jel(Be(1, g) \neq Jel(Be(2, g))$$

7. Egy NOT kapu kimenete különbözik a bemenetétől:

$$\forall g \; (Típus(g) = NOT) \Rightarrow Jel(Ki(1, g)) \neq Jel(Be(1, g))$$

4.7. A problémaspecifikus példányok kódolása

A 8.4. ábrán bemutatott áramkör neve C_1 és az itt következő leírással adjuk meg. Először kategorizáljuk a kapukat:

$$Típus(X_1) = XOR \; Típus(X_2) = XOR$$

$$Típus(A_1) = AND \; Típus(A_2) = AND$$

$$Típus(O_1) = OR$$

Ezután leírjuk a köztük fennálló kapcsolatokat:

$$Összekapcsolt(Ki(1, X_1), Be(1, X_2)) \; Összekapcsolt(Be(1, C_1), Be(1, X_1))$$

$$Összekapcsolt(Ki(1, X_1), Be(2, A_2)) \; Összekapcsolt(Be(1, C_1), Be(1, A_1))$$

$$Összekapcsolt(Ki(1, A_2), Be(1, O_1)) \; Összekapcsolt(Be(2, C_1), Be(2, X_1))$$

$$Összekapcsolt(Ki(1, A_1), Be(2, O_1)) \; Összekapcsolt(Be(2, C_1), Be(2, A_1))$$

Created by XMLmind XSL-FO Converter.

$Összekapcsolt(Ki(1, X_2), Ki(1, C_1))$ $Összekapcsolt(Be(3, C_1), Be(2, X_2))$

$Összekapcsolt(Ki(1, O_1), Ki(2, C_1))$ $Összekapcsolt(Be(3, C_1), Be(1, A_2))$

4.8. Lekérdezések megfogalmazása a következtetési eljárás felé

Milyen bemeneti kombinációk esetében lenne a C_1 első kimenete (az összegelem) 0 és a C_1 második kimenete (a maradék elem) 1?

$$\exists i_1, i_2, i_3 \; Jel(Be(1, C_1)) = i_1 \wedge Jel(Be(2, C_1)) = i_2 \wedge Jel(Be(3, C_1)) = i_3$$

$$\wedge Jel(Ki(1, C_1)) = 0 \wedge Jel(Ki(2, C_1)) = 1$$

A válaszok az i_1, i_2 és i_3 változók behelyettesítései úgy, hogy a keletkezett mondat következzen a tudásbázisból. Három ilyen behelyettesítés létezik:

$\{i_1/1, i_2/1, i_3/0\}$ $\{i_1/1, i_2/0, i_3/1\}$ $\{i_1/0, i_2/1, i_3/1\}$

Melyek a az összeadó áramkör összes végpontjának lehetséges értékhalmazai?

$$\exists i_1, i_2, i_3, o_1, o_2 \; Jel(Be(1, C_1)) = i_1 \wedge Jel(Be(2, C_1)) = i_2$$

$$\wedge Jel(Be(3, C_1)) = i_3 \wedge Jel(Ki(1, C_1)) = o_1 \wedge Jel(Ki(2, C_1)) = o_2$$

Ez az utolsó lekérdezés egy teljes bemenet-kimenet táblázatot ad meg az eszközre, amelyet aztán ellenőrizhetünk, hogy valóban helyesen adja-e össze a bemeneteket. Ez egy egyszerű példa az **áramkör ellenőrzésére** (**circuit verification**). Az áramkör bemutatott definícióját nagyobb digitális rendszerek építésére is felhasználhatjuk, amelyekre aztán ugyanez a fajta ellenőrzési folyamat alkalmazható (lásd 8.17. feladat). Sok tárgyterület kezelhető egy ehhez hasonló strukturált tudásbázis-fejlesztéssel, amelyekben összetettebb koncepciókat határozunk meg egyszerűbb koncepciókra építve.

4.9. Hibák kiszűrése a tudásbázisból

Sokféleképpen perturbálhatjuk a tudásbázist, hogy meglássuk, milyen fajta hibás viselkedések fordulhatnak elő. Például tételezzük fel, hogy kihagyjuk az $1 \neq 0$[13] állítást. A rendszer ebben a pillanatban már nem lesz képes semmilyen kimenetet sem produkálni az áramkörben, kivéve, ha a bemenet a 000 vagy az 110. Rábukkanhatunk a problémára, ha minden egyes kapu kimenetére rákérdezünk. Például, megkérdezhetjük:

$$\exists i_1, i_2, o \; Jel(Be(1, C_1)) = i_1 \wedge Jel(Be(2, C_1)) = i_2 \wedge Jel(Ki(1, X_1))$$

ami felfedi, hogy az X_1-nél nem ismerjük a kimenetet az 10 és a 01 bemenetek esetén. Ezután megnézzük az XOR kapu axiómáit az X_1-re alkalmazva:

$$Jel(Ki(1, X_1)) = 1 \Leftrightarrow Jel(Be(1, X_1)) \neq Jel(Be(2, X_1))$$

Ha tudjuk, hogy a bemenetek, mondjuk, 1 és 0 voltak, akkor ezt lerövidíthetjük így:

$$Jel(Ki(1, X_1)) = 1 \Leftrightarrow 1 \neq 0$$

Most már látható, hogy mi a probléma oka: a rendszer nem képes kikövetkeztetni azt, hogy $Jel(Ki(1, X_1)) = 1$, így meg kell neki mondanunk, hogy $1 \neq 0$.

5. Összefoglalás

Ebben a fejezetben megmutattuk, hogy hogyan használható az **elsőrendű logika** (**first-order logic**) egy tudásalapú ágens reprezentációs nyelveként. A legfontosabb megállapítások a következők:

- A tudásreprezentációs nyelveknek deklaratívnak, kompozíciósnak, kifejezőnek, a szövegkörnyezettől függetlennek és egyértelműnek kell lenniük.

[13] Ez a fajta kihagyás eléggé gyakori, mert az emberek általában feltételezik, hogy a különböző nevek különböző dolgokat takarnak. A logikai programozási rendszerek, amelyeket bővebben a 9. fejezetben mutatunk be, szintén megteszik ezt a feltételezést.

Created by XMLmind XSL-FO Converter.

- A logikák különböznek **ontológiai** és **episztemológiai megállapítás**aikban. Amíg az ítéletlogika megállapításai csak tények létezésére vonatkoznak, addig az elsőrendű logika az objektumok és a relációk létezését is felhasználhatja, így nagyobb kifejezőerővel rendelkezik.

- Egy **lehetséges világ**ot vagy **modell**t az elsőrendű logikában objektumok egy halmaza definiál, a köztük lévő relációk és a rájuk alkalmazható függvények által.

- A **konstansszimbólum**ok (**constant symbol**s) objektumokat neveznek meg, a **predikátumszimbólum**ok (**predicate symbol**s) relációkat, míg a **függvényszimbólum**ok (**function symbol**s) függvényeket. Egy interpretáció megadja a leképezést a szimbólumok és a modell között. Az **összetett term**ek (**complex term**s) függvényszimbólumokat rendelnek hozzá a termekhez, hogy nevet adjanak egy objektumnak. Ha megadunk egy interpretációt és egy modellt, a mondat igazságtartalmát meghatároztuk.

- Egy **atomi mondat** (**atomic sentence**) egy vagy több termre alkalmazott predikátumból áll. A mondat csak akkor igaz, ha a predikátum által megnevezett reláció fennáll a termek által megnevezett objektumok között. Az **összetett mondat**ok (**complex sentence**s) – az ítéletlogikához hasonló módon – összekötőjeleket használnak, a **kvantor**ok (**quantifier**s) használata pedig lehetővé teszi általános szabályok megfogalmazását is.

- Egy tudásbázis építése az elsőrendű logikában a tárgyterület alapos elemzését igényli, valamint egy szótár megválasztását és a kívánt következtetések eléréséhez szükséges axiómák kódolását.

5.1. Irodalmi és történeti megjegyzések

Habár már Arisztotelész logikája képes volt objektumok feletti általánosítások kezelésére, az igazi elsőrendű logika keletkezését Gottlob Frege *Begriffschrift* (Fogalmak írása vagy Fogalmi jelölés; Frege, 1879) c. művének megjelenésétől, a kvantorok bevezetésétől számíthatjuk. A Frege-féle képesség a kvantorok egymásba ágyazása jelentős előrelépést jelentett, de az általa alkalmazott jelölésrendszer igen körülményes volt. Az elsőrendű logika jelenlegi jelölésrendszere alapvetően Giuseppe Peanótól származik (Peano, 1889), a szemantika azonban megegyezik a Frege által bemutatottal. Eléggé különös, hogy Peano axiómái nagyrészt Grassmann-nak (Grassmann, 1861) és Dedekindnek (Dedekind, 1888) voltak köszönhetők.

Az elsőrendű logika fejlődésének fontos akadálya volt az egyváltozós predikátumok előtérbe helyezése, illetve a többváltozós predikátumok kizárása. Az egyváltozós predikátumokhoz való ragaszkodás általánosan jellemző volt a logikai rendszerekre Arisztotelésztől Boole-ig. A logikai relációk első rendszerezett leírását Augustus De Morgan adta (De Morgan, 1864). De Morgan a következő példával mutatta be az arisztotelészi logikával nem kezelhető következtetéseket: „Minden ló állat; ezért a ló feje egy állat feje”. Ez a következtetés nem valósítható meg az arisztotelészi rendszerrel, mert bármely, a következtetésben alkalmazható szabálynak először az „*x* a feje *y*-nak” két predikátumot tartalmazó mondatot kell elemeznie. A relációk logikáját alaposan tanulmányozta Charles Sanders Peirce (Peirce, 1870), aki Frege-től függetlenül, néhány évvel később szintén kifejlesztette az elsőrendű logikát (Peirce, 1883).

Leopold Löwenheim adta meg a modellelmélet rendszerezett leírását az elsőrendű logika számára (Löwenheim, 1915). Cikke az egyenlőségszimbólumot már a logika szerves részének tekinti. Löwenheim eredményeit Thoralf Skolem fejlesztette tovább (Skolem, 1920). Alfred Tarski a halmazelméletet felhasználva megadta az igazság és a modellelméleti kielégíthetőség explicit definícióját (Tarski, 1935, 1956).

Elsősorban McCarthy érdeme az elsőrendű logika alkalmazása az MI-rendszerek eszközeként (McCarthy, 1958). A logikán alapuló MI-rendszerek fejlődésében jelentős előrelépést jelentett Robinson rezolúciós algoritmusa (Robinson, 1965), ami egy komplett elsőrendű logikai következtetési folyamat. A rezolúciót a 9. fejezetben fogjuk tárgyalni. A logikai megközelítés alapjaival a Stanfordon sok eredményt értek el. Cordell Green kifejlesztett egy elsőrendű logikai következtetési rendszert, a QA3-at (Green, 1969a, 1969b), amely ahhoz vezetett, hogy először kíséreltek meg létrehozni egy logikai robotot a Stanford Research Institute-ban (Fikes és Nilsson, 1971). Az elsőrendű logikát Zohar Manna és Richard Waldinger alkalmazta a programokban való következtetésre (Manna és Waldinger, 1971), majd később Michael Genesereth az áramkörökhöz (Genesereth, 1984). Európában a logikai programozást – az elsőrendű következtetés korlátozott változatát – fejlesztették ki a nyelvészeti elemzésekhez (Colmerauer és társai, 1973) és általános deklaratív rendszerekhez (Kowalski, 1974). A számítógépes logikával Edinburgh-ban sikeresen foglalkoztak, az LCF (Számítási Funkciók Logikája) projekten keresztül (Gordon és társai, 1979). Ezeket a fejlesztéseket a 9. és 10. fejezetben követjük nyomon.

Created by XMLmind XSL-FO Converter.

Számos színvonalas, az elsőrendű logika korszerű bevezetését adó írás ismert. Ezek közül Quine műve az egyik legjobban áttekinthető (Quine, 1982). Enderton írása inkább matematikai megközelítést követ (Enderton, 1972). Bell és Machover erősen formális leírást készített az elsőrendű logikáról, és ezzel együtt számos bonyolult, a logika tárgyterületébe tartozó témáról (Bell és Machover, 1977). Manna és Waldinger a számítástechnika oldaláról közelítő, jól olvasható bevezetést készített (Manna és Waldinger, 1985). Gallier az elsőrendű logika rendkívül precíz matematikai bemutatását adta meg, együtt a logika automatikus következtetésre történő alkalmazásának részletes tárgyalásával (Gallier, 1986). A *Logical Foundations of Artificial Intelligence* (A mesterséges intelligencia logikai alapjai) (Genesereth és Nilsson, 1987) c. könyv a logika alapos bemutatásán kívül az érzeteket és cselekvéseket kezelő logikai ágensek első rendszerezett bevezetését nyújtja.

5.2. Feladatok

8.1.

Egy logikai tudásbázis a világot mondatokkal reprezentálja határozott struktúra nélkül. Egy **analóg (analog)** reprezentáció viszont strukturált, ahol a leírás struktúrája közvetlenül megfelel a reprezentált dolog struktúrájának. Tekintsük egy ország autótérképét mint az országról ismert tények egy részének analóg reprezentációját. A térkép kétdimenziós felépítése megfelel a terület kétdimenziós felszínének.

a. Adjon öt példát a térképnyelv *szimbólumaira*.

b. *Explicit* mondatnak nevezzük az olyan mondatot, amelyet a reprezentáció létrehozója közvetlenül leír. Az *implicit* mondatok az explicit mondatokból keletkeznek az analóg reprezentáció tulajdonságai szerint. Adjon három példát a térképnyelv *implicit* és *explicit* mondataira.

c. Adjon az ország fizikai struktúráját leíró tényekre három olyan példát, amely nem reprezentálható a térképnyelvvel.

d. Adjon két példát olyan tényekre, amelyek egyszerűbben kifejezhetők a térképnyelvvel, mint az elsőrendű logikával.

e. Adjon még két példát analóg reprezentációkra. Melyek az előnyei és a hátrányai ezeknek a nyelveknek?

8.2.

Tekintsünk egy tudásbázist, amely csak két kijelentést tartalmaz: $P(a)$ és $P(b)$. Ebből a tudásbázisból következik-e az $\forall x\, P(x)$? Magyarázza meg válaszát modellek segítségével.

8.3.

Érvényes-e a következő mondat: $\exists x, y\ x = y$? Magyarázza meg.

8.4.

Írjon le egy olyan logikai mondatot, hogy minden olyan világban, ahol ez a mondat igaz, pontosan csak egy objektum legyen.

8.5.

Tekintsünk egy szimbólumszótárat, amely tartalmaz egy c konstansszimbólumot, p_k predikátumszimbólumokat minden egyes k értékre és függvényszimbólumokat minden egyes k értékhez, ahol $1 \leq k \leq A$. A tárgyterület méretét rögzítsük D-ben. Bármely adott interpretációmodell-kombinációban minden egyes predikátum vagy függvény hozzá van rendelve az ugyanazon értékhez tartozó relációhoz vagy függvényhez. Feltételezhetjük, hogy a modellben szereplő függvények lehetővé teszik, hogy néhány bemenetnek ne legyen értéke a függvény számára (vagyis, hogy az érték a láthatatlan objektum). Vezessen le egy formulát a lehetséges interpretációmodell-kombinációk számának megállapítására a D elemet tartalmazó tárgyterületben. Ne riassza meg, ha el kell távolítania a felesleges kombinációkat.

8.6.

Reprezentálja a következő mondatokat az elsőrendű logikában úgy, hogy egy következetes szótárt használ (amelyet előzőleg definiálnia kell):

Created by XMLmind XSL-FO Converter.

a. Néhány diák felvette a franciát 2001 tavaszi félévében.

b. Minden diák, aki felveszi a franciát, átmegy a vizsgán.

c. Csak egy diák vette fel a görögöt 2001 tavaszi félévében.

d. A görögben elért legmagasabb pontszám mindig magasabb a franciában elért legmagasabbnál.

e. Minden személy okos, aki biztosítást köt.

f. Senki nem köt drága biztosítást.

g. Van egy ügynök, aki csak azokkal köt biztosítást, akiknek még nincs kötvényük.

h. Van egy borbély, aki minden férfit megborotvál a városban, aki nem borotválkozik.

i. Az a személy, aki az Egyesült Királyságban születik, és akinek mindkét szülője brit állampolgár vagy ottani lakos, születésénél fogva brit állampolgár.

j. Az a személy, aki az Egyesült Királyságon kívül születik, és egyik szülője született brit állampolgár, az származása alapján brit állampolgár.

k. A politikusok bármikor bolonddá tehetnek néhány embert, és minden embert bolonddá tehetnek egy kis időre, de nem tehetnek bolonddá mindenkit örökre.

8.7.

Reprezentálja a „Minden német azonos nyelvet beszél” mondatot predikátumkalkulusban. Használja a *Beszél*(*x*, *l*) predikátumot, amely jelentse azt, hogy *x* az *l* nyelvet beszéli.

8.8.

Milyen axiómára van szükségünk, hogy ezt a tény kikövetkeztessük: *Nő*(*Laura*), ha adottak a tények: *Férfi*(*Jim*) és *Házastársak*(*Jim*, *Laura*)?

8.9.

Írjon egy általános tény- és axiómahalmazt, hogy reprezentálja a következő állítást: „Wellington hallott Napóleon haláláról”; és hogy helyesen megválaszolhassuk a kérdést: „Hallott Napóleon Wellington haláláról?”.

8.10.

Írja át a 7.5. alfejezetben bemutatott ítéletlogikai wumpus világ tényeit elsőrendű logikába. Mennyivel tömörebb ez a változat?

8.11.

Készítsen axiómákat, amelyek leírják a következő predikátumokat: *Unokája*, *Dédnagyapja*, *Fivére*, *Húga*, *Lánya*, *Fia*, *Nagynénje*, *Nagybátyja*, *Sógornője*, *Sógora*, *Unokatestvére*.

Írja le a 8.5. ábrán látható családfa tényeit. Használjon logikai következtető rendszert, és minden leírt mondat legyen a következtető rendszer számára `KIJELENT`-ve, és `KÉRDEZ`-ze meg, hogy ki Erzsébet unokája, ki Diana unokabátyja és kik Zara dédszülei.

8.12.

Írjon le egy mondatot, amely azt állítja, hogy a + egy kommutatív funkció. Következik ez a mondat Peano axiómáiból? Ha igen, magyarázza meg, hogy miért, ha nem, adjon meg egy modellt, amelyben az axiómák igazak, az Ön mondata pedig hamis.

8.13.

Magyarázza meg, mi a hibás az $\in$ halmaz tagság predikátum következő javasolt definíciójával:

Created by XMLmind XSL-FO Converter.

$\forall x,\ s\ x \in \{x|\ s\}$

$\forall x,\ s\ x \in s \Rightarrow \forall y\ \ x \in \{y|\ s\}$

8.14.

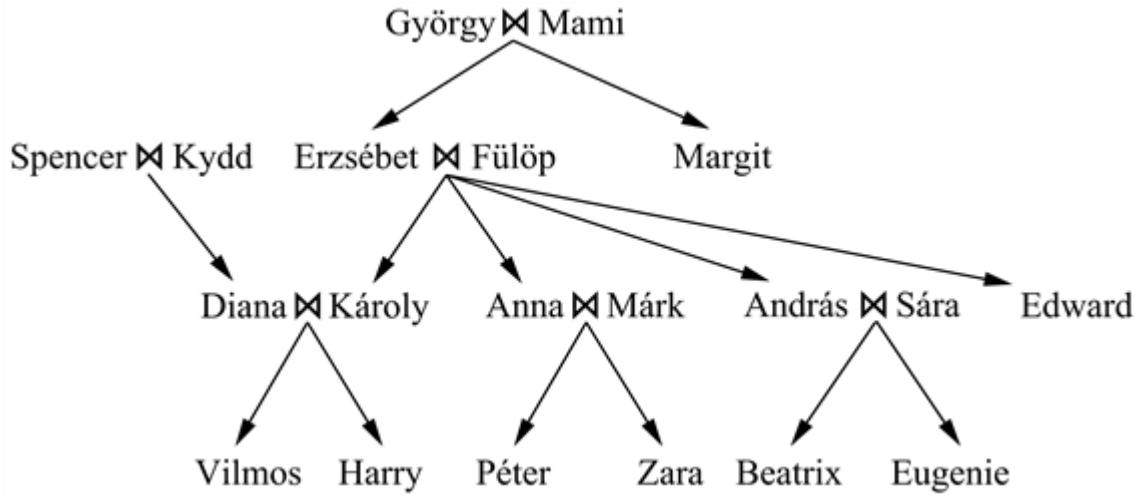
A halmazaxiómákat példaként használva, készítsen axiómákat a listák tárgyterületére, amelyek tartalmazzák a korábban a fejezetben említett összes konstanst, függvényt és predikátumot.

8.15.

Magyarázza meg, hogy mi a hiba a következő ajánlott definícióban a szomszédos négyzetekre a wumpus világban:

$\forall x,\ s\ Szomszédos([x, y], [x + 1, y]) \wedge Szomszédos([x, y], [x, y + 1])$

8.5. ábra - Egy tipikus családfa. Az „=” szimbólum a házastársakat köti össze, a nyilak a gyerekekre mutatnak.



8.16.

Írja meg a wumpus helyzetének meghatározásához szükséges axiómákat a *Wumpus* konstansszimbólum és a *Be*(*Wumpus*, *Helyzet*) bináris predikátum felhasználásával. Ne felejtse el, hogy csak egy wumpus van.

8.17.

Bővítse ki a 8.4. alfejezetben leírt szótárt úgy, hogy definiálja az összeadást az *n*-bites bináris számokra. Ezután kódolja a 8.6. ábrán látható négybites összeadó leírását, és tegye fel azokat a kérdéseket, amelyek szükségesek ahhoz, hogy igazoljuk összeadó-helyességét.

8.18.

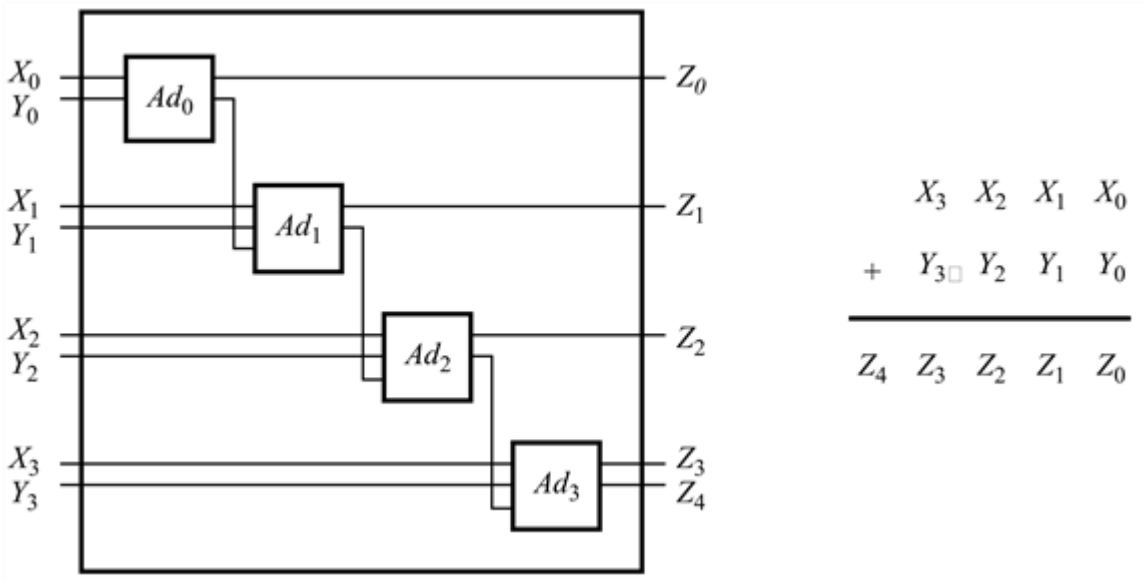
A fejezetben az áramkör reprezentálása részletesebb a szükségesnél, ha csak az áramkör működése érdekel minket. Egy egyszerűbb formula bármely *m* bemenetű, *n* kimenetű kaput vagy áramkört leír, egy *m* + *n* argumentumos predikátum használatával, úgy, hogy a predikátum pontosan akkor igaz, amikor a bemenet és a kimenet konzisztens. Például, a NOT-kapukat így írjuk le: *NOT*(*i*, *o*), amelynél a *NOT*(0, 1)-et és a *NOT*(1, 0)-át ismerjük. A kapuk összeállítása a kapupredikátumok összekötésével van definiálva, amelyben a közös változók direkt összeköttetéseket jeleznek. Például egy NAND áramkört összeállíthatunk *AND*-ekből és *NOT*-okból:

$\forall i_1,\ i_2,\ o_a,\ o\ NAND(i_1,\ i_2,\ o) \Leftarrow AND(i_1,\ i_2,\ o) \wedge NOT(o_a,\ o)$

Ezt a reprezentációt felhasználva definiálja a 8.4. ábrán látható egybites összeadót és a 8.6. ábra négybites összeadóját, és magyarázza meg, milyen lekérdezéseket használna, hogy igazolja a megoldásokat. Milyen fajta lekérdezéseket *nem* fogad el ez a reprezentáció, amelyeket pedig a 8.4. alfejezetben lévő reprezentáció elfogadott?

Created by XMLmind XSL-FO Converter.

8.6. ábra - Egy négybites összeadó



8.19.

Kérjen útlevéligénylő lapot az Ön saját országába, nevezze meg azokat a szabályokat, amelyek az igénylés jogosultságát meghatározzák, és fordítsa le az elsőrendű logika nyelvére a 8.4. alfejezetben felvázolt lépéseket követve.

Created by XMLmind XSL-FO Converter.

9. fejezet - Következtetés elsőrendű logikában

Ebben a fejezetben definiálunk egy olyan hatékony következtetési mechanizmust, amelyik képes megválaszolni az elsőrendű logikában feltett kérdéseket.

A 7. fejezetben meghatároztuk a **következtetés** (**inference**) fogalmát, és bemutattuk, hogy hogyan érhető el helyes és teljes következtetés az ítéletlogikában. Ebben a fejezetben kiterjesztjük ezeket az eredményeket, hogy olyan algoritmusokat kapjunk, amelyek bármely, az elsőrendű logikában feltehető kérdésre válaszolni tudnak. Ez jelentős eredmény, mivel az elsőrendű logikával többé-kevésbé mindent kifejezhetünk, ha elég alaposan végezzük a feladatot.

A 9.1. alfejezet következtetési szabályokat vezet be a kvantorokra, és megmutatja, hogy hogyan lehet – jóllehet nagy erőfeszítések árán – az elsőrendű logikai következtetést ítéletlogikai következtetésre redukálni. A 9.2. alfejezet leírja az **egyesítés** (**unification**) ötletét, bemutatva, hogy az egyesítés felhasználásával, hogyan alkothatunk következtetési szabályokat, amelyek közvetlenül az elsőrendű logikai mondatokra alkalmazhatók. Ezután a 9.3. alfejezetben megvizsgáljuk az elsőrendű logikai algoritmusok három nagy családját: az **előrefelé láncolás**t (**forward chaining**) és ennek alkalmazását a **deduktív adatbázis**okban (**deductive database**s) és a **produkciós rendszer**ekben (**production system**s). A **hátrafelé láncolás**t (**backward chaining**) és a **logikai programozás**t (**logic programming**) alkalmazó rendszereket a 9.4. alfejezetben tárgyaljuk; míg a rezolúció alapú **tételbizonyító** (**theorem-proving**) rendszereket a 9.5. alfejezet ismerteti. Általánosságban mindig a leghatékonyabb, az adott feladatban reprezentálandó tényeket és axiómákat kezelni képes módszert próbáljuk használni. A teljesen általános, a tetszőleges elsőrendű logikai mondatokon alkalmazható rezolúciós következtetés általában kevésbé hatékony, mint a bizonyos típusú mondatokon alkalmazható előre- vagy hátrafelé láncolást használó megoldások.

1. Ítéletlogikai következtetés kontra elsőrendű logikai következtetés

Ez és a következő alfejezet bemutatja azokat a gondolatokat, amelyeken a modern logikai következtetési rendszerek alapulnak. Néhány egyszerű következtetési szabállyal kezdjük, amelyeket kvantorral ellátott mondatokhoz alkalmazhatunk, hogy a segítségükkel kvantorok nélküli mondatokhoz juthassunk. Ezek a szabályok természetes módon elvezetnek minket ahhoz a gondolathoz, hogy az *elsőrendű* következtetés megvalósítható azáltal, hogy a tudásbázist *ítéletlogikává* alakítjuk át, és a már általunk ismert *ítéletlogikai* következtetést használjuk. A következő alfejezet rámutat egy kézenfekvő egyszerűsítésre, amellyel olyan következtetési módszerekhez juthatunk, amelyek közvetlenül képesek az elsőrendű mondatok felhasználására.

1.1. Kvantorokra vonatkozó következtetési szabályok

Kezdjük az univerzális kvantorok vizsgálatával. Tételezzük fel, hogy a tudásbázisunk tartalmazza a standard hagyományos axiómákat, amelyek szerint minden mohó király gonosz:

$$\forall x \; Király(x) \land Mohó(x) \Rightarrow Gonosz(x)$$

Ebből lehetségesnek tűnik kikövetkeztetni a következő mondatokat:

$$Király(János) \land Mohó(János) \Rightarrow Gonosz(János)$$

$$Király(Richárd) \land Mohó(Richárd) \Rightarrow Gonosz(Richárd)$$

$$Király(Apja(János)) \land Mohó(Apja(János)) \Rightarrow Gonosz(Apja(János))$$

...

Az **univerzális példányosítás** (**Universal Instantiation**, **UI**) szabálya kimondja, hogy az adott változó bármely **alapterm**mel (**ground term**) (változók nélküli termmel) való helyettesítésével elérhető mondatokat

Created by XMLmind XSL-FO Converter.

kikövetkeztethetjük.[1] Ahhoz, hogy formálisan le tudjuk írni a következtetési szabályokat, a 8.3. alfejezetben bevezetett **helyettesítés** (**substitution**) fogalmát használjuk. A HELYETTESÍT (θ, α) fogja jelölni annak az eredményét, hogy egy θ behelyettesítést alkalmazunk az α mondathoz. Ekkor a szabály így írható le:

$$\frac{\forall v\ \alpha}{HELYETTESÍT(\{v/g\}, \alpha)}$$

bármely v változóra és g alaptermre. Például a korábban megadott három mondatot megkaphatjuk a következő helyettesítésekkel $\{x/János\}$, $\{x/Richárd\}$ és $\{x/Apja(János)\}$.

Az ennek megfelelő **egzisztenciális példányosítás**i (**Existential Instantiation**) szabály az egzisztenciális kvantorra egy kicsivel komplikáltabb. Minden α mondatra, v változóra és olyan k konstans szimbólumra, amely sehol máshol nem jelenik meg a tudásbázisban:

$$\frac{\exists v\ \alpha}{HELYETTESÍT(\{v/k\}, \alpha)}$$

Például abból a mondatból, hogy:

$\exists x\ Korona(x) \wedge Fején(x, János)$

kikövetkeztethetjük ezt a mondatot:

$Korona(C_1) \wedge Fején(C_1, János)$

feltéve, ha a C_1 nem jelenik meg sehol máshol a tudásbázisban. Alapjában véve az egzisztenciális mondat azt mondja ki, hogy van valamely objektum, amely eleget tesz egy feltételnek, és a példányosítás folyamata csak névvel látja el ezt az objektumot. Ez a név viszont már nem tartozhat más objektumhoz. Egy szép példát találunk a matematikában: tételezzük fel, hogy tudjuk, létezik olyan szám, amely egy kicsit kisebb, mint 2,71828, és eleget tesz annak az egyenletnek, hogy: $d(x^y)/dy = x^y$ az x-re. Adhatunk nevet ennek a számnak, mint például e, de hiba lenne olyan nevet adni, amely már egy létező objektumhoz tartozik, mint például a Π. A logikában ezt az új nevet **Skolem-konstans**nak (**Skolem constant**) nevezzük. Az egzisztenciális példányosítás egy speciális esete egy általánosabb folyamatnak, amelyet **skolemizáció**nak (**skolemization**) nevezünk, és amelyet a 9.5. alfejezetben mutatunk be.

Amellett hogy az egzisztenciális példányosítás bonyolultabb, mint az univerzális példányosítás, kissé különböző szerepet is játszik a következtetésben. Míg az univerzális példányosítást többször is alkalmazhatjuk, hogy sok különböző eredményhez jussunk, az egzisztenciális példányosítást csak egyszer végezhetjük el, majd az egzisztenciális kvantorral ellátott mondattól megszabadulhatunk. Például amint hozzáadtuk a tudásbázishoz a *Meggyilkol*(*Gyilkos*, *Áldozat*) mondatot, már nincs szükségünk a $\exists x$ *Meggyilkol*(*x*, *Áldozat*) mondatra. Pontosabban megfogalmazva, az új tudásbázis logikailag nem ekvivalens a régivel, de tekinthetjük úgy, hogy a **következtetés szempontjából ekvivalens** (**inferentially equivalent**), azaz pontosan akkor kielégíthető, amikor az eredeti tudásbázis is kielégíthető.

1.2. Redukálás ítéletlogikára

Ha egyszer vannak szabályaink arra, hogy kvantorral ellátott mondatokból hogyan következtethetünk kvantor nélküli mondatokra, akkor vissza tudjuk vezetni az elsőrendű következtetést az ítéletlogikai következtetésre. Ebben az alfejezetben áttekintjük ennek a módszernek a legfontosabb elemeit; a részletekről a 9.5. alfejezetben fogunk szólni.

Az első gondolat az, hogy éppen úgy, ahogyan egy egzisztenciális kvantorral ellátott mondatot felcserélhetünk a mondat egy példányával, egy univerzális kvantorral ellátott mondatot is felcserélhetünk a mondat *összes lehetséges* példányosításainak halmazával. Például tételezzük fel, hogy a tudásbázisunk mindössze a következő mondatokat tartalmazza:

$\forall x\ Király(x) \wedge Mohó(x) \Rightarrow Gonosz(x)$

[1] Ne keverjük össze ezeket a helyettesítéseket a kiterjesztett interpretációkkal, amelyeket a kvantorok szemantikájának a meghatározására használtunk. A helyettesítés egy változót cserél le egy termre (egy szintaktikai elemre), hogy új mondatot hozzon létre, ezzel szemben egy interpretáció hozzárendel egy változót egy objektumhoz a tárgyterületben.

Created by XMLmind XSL-FO Converter.

Király(*János*) (9.1)

Mohó(*János*)

Fivér(*Richárd*, *János*)

Ekkor alkalmazzuk az univerzális példányosítást az első mondatra úgy, hogy az összes lehetséges alaptermhelyettesítést felhasználjuk a tudásbázisszótárból, ami ebben az esetben az {*x/János*} és az {*x/Richárd*}. Így ezt kapjuk:

Király(*János*) ∧ *Mohó*(*János*) ⇒ *Gonosz*(*János*)

Király(*Richárd*) ∧ *Mohó*(*Richárd*) ⇒ *Gonosz*(*Richárd*)

és megválhatunk az univerzális kvantorral ellátott mondattól. Mármost, a tudásbázist alapvetően ítéletkalkulusbelinek tekinthetjük, ha az alap atomi mondatokat, mint a *Király*(*János*), *Mohó*(*János*) stb., ítéletlogikai szimbólumoknak tekintjük. Így tehát a 7. fejezetben bemutatott bármely teljes ítéletlogikai algoritmust alkalmazhatjuk, hogy olyan következtetéshez juthassunk, mint a *Gonosz*(*János*).

Ez a technika az **ítéletlogikai állításokra való visszavezetés** (**propositionalization**), amelyet teljesen általánossá tehetünk, mint azt a 9.5. alfejezetben be is fogjuk mutatni. Ez azt jelenti, hogy minden elsőrendű tudásbázis és lekérdezés átalakítható ítéletlogikai mondatokra úgy, hogy a tudásbázis vonzatai nem változnak. Így tehát van egy teljes következtetési folyamatunk vonzatok levezetésére. Vagy talán mégsem? Van egy probléma: amikor a tudásbázis tartalmaz függvényszimbólumot, a lehetséges alaptermek helyettesítéseinek halmaza végtelen! Például ha a tudásbázis megemlíti az *Apja* szimbólumot, akkor végtelen számú egymásba ágyazott termeket hozhatunk létre, mint például az *Apja*(*Apja*(*Apja*(*János*))). Az ítéletlogikai algoritmusainknak tehát nehézségeik lesznek a végtelen nagyságú mondathalmazokkal.

Szerencsére ismert egy Jacques Herbrandnak köszönhető híres tétel az előbbi problémára (Herbrand, 1930), ami azt mondja ki, hogy ha egy mondat következik az eredeti, elsőrendű tudásbázisból, akkor létezik olyan bizonyítás, amely csak egy véges méretű részhalmazt használ fel az ítéletkalkulus-belire átalakított tudásbázisból. Mivel bármely ilyen részhalmazban az alaptermeknek van egy maximális beágyazási mélysége, meg lehet találni a részhalmazt úgy, hogy először generáljuk az összes példányt a konstans szimbólumokhoz (*Richárd* és *János*), azután az összes 1-es mélységű termet (*Apja*(*Richárd*) és *Apja*(*János*)), majd az összes 2-es mélységű termet, és így tovább mindaddig, amíg képesek leszünk megalkotni a vonzatmondat ítéletlogikai bizonyítását.

Fontos

Felvázoltuk az elsőrendű következtetés egy megközelítését az ítéletlogikai állításokra való visszavezetés révén, amely **teljes** (**complete**) – tehát bármely következményként kapott mondatot bizonyítani tudunk. Ez jelentős eredmény, ha figyelembe vesszük, hogy a lehetséges modellek száma végtelen. Másrészt viszont nem tudhatjuk, hogy a mondat *vonzat-e*, amíg a bizonyítás nincs kész. Mi történik, ha *α mondat nem vonzata a TB-nek*? Meg tudjuk-e ezt állapítani? Nos, ahogy ez majd kiderül, az elsőrendű logika esetében erre a kérdésre a válasz nemleges. A bizonyítási eljárás tovább folytatódik, egyre mélyebben beágyazott termeket generálva, és nem tudhatjuk, hogy a folyamat egy végtelen ciklusba kerül, vagy hamarosan megleljük a bizonyítást. Ez nagyon hasonló a Turing gépek leállási problémájához. Alan Turing és Alonzo Church egymástól függetlenül és eltérő módon bizonyították be ennek a kérdésnek az eldönthetetlenségére vonatkozó tételt (Turing, 1936; Church, 1936). *Az elsőrendű logikában a maga után vonzás kérdése* ***félig eldönthető*** (***semidecidable***), *ami azt jelenti, hogy létezik olyan algoritmus, amely igent mond minden vonzatmondatra, de nem létezik olyan algoritmus, amely emellett képes nemet mondani a nem levezethető mondatokra.*

2. Egyesítés és kiemelés

Az előző alfejezet bemutatta az elsőrendű következtetésnek azt az értelmezését, amely az 1960-as évekig létezett. Az éles szemű olvasók (és bizonyára a hatvanas évek elejének logikával foglalkozó kutatói) észrevehették, hogy az ítéletlogikai megközelítés nem nagyon hatékony. Például ha megnézzük a *Gonosz*(*x*) lekérdezést és az ehhez tartozó tudásbázist a (9.1) egyenletben, feleslegesnek tűnik olyan mondatok generálása, mint a *Király*(*Richárd*) ∧ *Mohó*(*Richárd*) ⇒ *Gonosz*(*Richárd*). Valójában a *Gonosz*(*János*) mondatra vonatkozó következtetés bárki számára teljesen nyilvánvaló a következő mondatok alapján:

Created by XMLmind XSL-FO Converter.

$\forall x\ Király(x) \wedge Mohó(x) \Rightarrow Gonosz(x)$

$Király(János)$

$Mohó(János)$

Most megmutatjuk, hogyan tehetjük ezt a számítógép számára is egyértelművé.

2.1. Egy elsőrendű következtetési szabály

Az a következtetés, hogy János gonosz a következőképpen megy végbe: találjon egy *x*-et, amelyre igaz, hogy *x* egy király és *x* mohó, majd következtesse ki, hogy *x* gonosz. Általánosabban, ha van valamely θ helyettesítés, amely az implikáció premisszáját megegyezővé teszi a tudásbázisban már létező mondatokkal, akkor a θ helyettesítés elvégzése után az implikációkövetkezmény részét hozzáadhatjuk a tudásbázishoz. Ebben az esetben az {*x*/*János*} helyettesítés eléri ezt a célt.

Tulajdonképpen több feladatot is elvégeztethetünk a következtetés ezen lépésével. Tételezzük fel, hogy a *Mohó*(*János*) mondat ismerete helyett csak azt tudjuk, hogy *mindenki* mohó:

$\forall y\ Mohó(y)$ (9.2)

Ebben az esetben is szeretnénk azt a következtetést levonni, hogy *Gonosz*(*János*), mivel tudjuk, hogy János egy király (ez adott), és hogy János mohó (mert mindenki mohó). Egy megfelelő helyettesítés megtalálására van szükségünk ennek elvégzéséhez, mind az implikációs mondatban, mind az ehhez illesztendő mondatokban található változókra. Ebben az esetben az {*x*/*János*, *y*/*János*} helyettesítés alkalmazása az implikáció *Király*(*x*) és *Mohó*(*x*) premisszáihoz és a *Király*(*János*) és *Mohó*(*y*) tudásbázis mondatokhoz létrehozza az azonosságot. Így tehát ki tudjuk következtetni az implikáció konklúzióját.

Ez a következtetés elvégezhető egyetlen következtetési szabály alkalmazásával, amelyet **általánosított Modus Ponens**nek (**Generalized Modus Ponens**) nevezünk: a p_i, p'_i és q atomi mondatokra, amelyekre létezik olyan θ helyettesítés, hogy HELYETTESÍT (θ, p'_i) = HELYETTESÍT (θ, p_i) minden *i*-re, akkor:

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\text{HELYETTESÍT}(\theta, q)}$$

A szabály $n + 1$ premisszát tartalmaz: n számú p'_i atomi mondatot és egy implikációt. A konklúzió a q konzekvenciára történő helyettesítés alkalmazásának az eredménye. A mi példánkra ezt így alkalmazhatjuk:

$p'_1 = Király(János)$ $p_1 = Király(x)$

$p'_2 = Mohó(y)$ $p_2 = Mohó(x)$

$\theta = \{x/János, y/János\}$ $q = Gonosz(x)$

$Helyettesít(\theta, q) = Gonosz(János)$

Könnyű megmutatni, hogy az általánosított Modus Ponens helyes következtetési szabály. Először is, megfigyelhetjük, hogy bármely p mondatra (amelyeknek változóiról feltételezzük, hogy univerzális kvantorokkal vannak ellátva) és bármely θ helyettesítésre:

$p \models Helyettesít(\theta, p)$

Ez hasonlóan igazolható, mint az univerzális példányosítás szabály, és alkalmazható olyan θ-ra, amely kielégíti az általánosított Modus Ponens szabály feltételeit. Így tehát, a p_1',…, p_n' -ből következtethetjük, hogy:

$Helyettesít(\theta, p'_1) \wedge \ldots \wedge Helyettesít(\theta, p'_n)$

és a $p_1 \wedge \ldots \wedge p_n \Rightarrow q$ implikációból ezt következtethetjük, hogy:

$Helyettesít(\theta, p_1) \wedge \ldots \wedge Helyettesít(\theta, p_n) \Rightarrow Helyettesít(\theta, q)$

Mivel a θ az általánosított Modus Ponensben úgy van definiálva, hogy: HELYETTESÍT (θ, p_i') = HELYETTESÍT (θ, p_i) minden *i*-re; ebből kifolyólag a két mondatból az első pontosan illeszkedik a másodiknak a premisszájához. Így tehát a HELYETTESÍT (θ, q) következik a Modus Ponensből.

Created by XMLmind XSL-FO Converter.

Az általánosított Modus Ponens egy **kiemelt** (**lifted**) változata a Modus Ponensnek – átemeli a Modus Ponenst az ítéletlogikából az elsőrendű logikába. Látni fogjuk a fejezet későbbi részében, hogy kifejleszthetjük az előrefelé láncolás, a hátrafelé láncolás és a 7. fejezetben bemutatott rezolúciós algoritmusok kiemelt változatait is. A kiemelt következtetési szabályok legfontosabb előnye az ítéletlogikára történő átalakítással szemben az, hogy csak azokat a helyettesítéseket hajtják végre, amelyek bizonyos következtetések végrehajtását teszik lehetővé. Egy potenciális korlát az, hogy egy bizonyos szempontból az Általánosított Modus Ponens kevésbé általános, mint a Modus Ponens (lásd 4. szakasz - Az ítéletkalkulus következtetési mintái részben): a Modus Ponens bármely α mondatot megenged az implikáció bal oldalán, ezzel szemben az Általánosított Modus Ponens speciális formátumot kíván meg erre a mondatra. Csak abban az értelemben nevezhető általánosítottnak, hogy tetszőleges számú p_i'-re alkalmazható.

2.2. Egyesítés

A kiemelt következtetési szabályok olyan helyettesítések megtalálását igénylik, amelyek a különböző logikai kifejezéseket látszólag azonossá teszik. Ezt a folyamatot **egyesítési lépés**nek (**unification**) nevezzük, ami kulcsfontosságú eleme minden elsőrendű következtetési algoritmusnak. Az EGYESÍT algoritmus vesz két mondatot, és visszaad egy rájuk vonatkozó **egyesítés**t (**unifier**), ha létezik ilyen:

`Egyesít(`*p*`, `*q*`) = `*θ*`, ahol Helyettesít(`*θ*`, `*p*`) = Helyettesít(`*θ*`, `*q*`)`

Nézzünk meg néhány példát arra, hogy az EGYESÍT-nek hogyan kell viselkednie. Tételezzük fel, hogy van egy *Ismer*(*János*, *x*) lekérdezésünk: kit ismer János? Erre a kérdésre néhány választ úgy találhatunk, hogy megkeressük az összes mondatot a tudásbázisban, amely egyesíthető az *Ismer*(*János*, *x*)-szel. Itt vannak az egyesítési eredmények négy különböző, a tudásbázisban előforduló mondatra.

`Egyesít(`*Ismer*`(`*János*`, `*x*`), `*Ismer*`(`*János*`, `*Júlia*`)) = {`*x*`/`*Júlia*`}`

`Egyesít(`*Ismer*`(`*János*`, `*x*`), `*Ismer*`(`*y*`, `*Lajos*`)) = {`*x*`/`*Lajos*`, `*y*`/`*János*`}`

`Egyesít(`*Ismer*`(`*János*`, `*x*`), `*Ismer*`(`*y*`, `*Anyja*`(`*y*`))) = {`*y*`/`*János*`, `*x*`/`*Anyja*`(`*János*`)}`

`Egyesít(`*Ismer*`(`*János*`, `*x*`), `*Ismer*`(`*x*`, `*Erzsébet*`)) = `*sikertelen*

Az utolsó egyesítés sikertelen, mert az *x* nem tudja a *János* és az *Erzsébet* értékeit egyszerre felvenni. Emlékezzünk arra, hogy az *Ismer*(*x*, *Erzsébet*) azt jelenti, hogy: „Mindenki ismeri Erzsébetet”, így képesnek *kellene* lennünk azt kikövetkeztetni, hogy János ismeri Erzsébetet. A probléma abból származik, hogy a két mondat történetesen ugyanazt a változónevet, az *x*-et használja. A problémát úgy kerülhetjük el, hogy az egyesítendő mondatokban **átnevezzük a változókat** (**standardizing apart**) úgy, hogy elkerüljük a nevek egybeeséseit. Például átnevezhetjük az *x*-et az *Ismer*(*x*, *Erzsébet*)-ben például z_{17}-re (egy tetszőleges új változónévre) anélkül, hogy a jelentését megváltoztatnánk. Most már működik az egyesítésünk:

`Egyesít(`*Ismer*`(`*János*`, `*x*`), `*Ismer*`(`z_{17}`, `*Erzsébet*`)) = {`*x*`/`*Erzsébet*`, `z_{17}`/`*János*`}`

A 9.7. feladat tovább vizsgálja az átnevezés szükségességének problémáját.

9.1. ábra - Az egyesítési algoritmus. Az algoritmus összehasonlítja a bemenetek felépítését elemről elemre. A *θ* helyettesítés, amely az EGYESÍTÉS argumentuma, útközben épül fel, és arra használjuk, hogy meggyőződjünk arról, hogy a későbbi összehasonlítások konzisztensek lesznek az általunk előzőleg létrehozott lekötésekkel. Egy összetett kifejezésben, mint például az *F*(*A*, *B*), a VÁL függvény kiveszi az *F* függvényszimbólumot, és az ARGOK függvény kiveszi az (*A*, *B*) argumentumlistát.

Created by XMLmind XSL-FO Converter.

```
function EGYESÍT(x, y, θ) returns egy helyettesítést, hogy az x és az y megegyező legyen
  inputs: x, egy változó, lista vagy egy összetétel
          y, egy változó, lista vagy egy összetétel
          θ, az eddig felépített helyettesítés (választható, az üres kizárva)

  if θ = kudarc then return kudarc
  else if x = y then return θ
  else if VÁLTOZÓ?(x) then return EGYESÍT-VÁLT(x, y, θ)
  else if VÁLTOZÓ?(y) then return EGYESÍT-VÁLT(y, x, θ)
  else if ÖSSZETÉTEL?(x) and ÖSSZETÉTEL?(y) then
  return EGYESÍT(ARGOK[x], ARGOK[y], EGYESÍT(VÁL[x], VÁL[y], θ))
  else if LISTA?(x) and LISTA?(y) then
  return EGYESÍT(MARADÉK[x], MARADÉK[y], EGYESÍT(ELSŐ[x], ELSŐ[y], θ))
  else return kudarc

  function EGYESÍT-VÁLT(vált, x, θ) returns egy helyettesítés
  inputs: vált, egy változó
          x, bármely kifejezés
          θ, az eddig felépített helyettesítések

  if {vált/ért} ∈ θ then return EGYESÍT(ért, x, θ)
  else if {x/ért} ∈ θ then return EGYESÍT(ért, vál, θ)
  else if ELŐFORDUL-PRÓBA?{vált, x}θ-re then return kudarc
  else return hozzáad {vált/x} θ-hoz
```

Még egy további nehézségen kell túljutni. Azt mondtuk, hogy az EGYESÍT olyan helyettesítéseket ad vissza, amelyek a két argumentumot látszólag azonossá teszi. Azonban több különböző ilyen egyesítés létezhet. Például az EGYESÍT (*Ismer*(*János*, *x*), *Ismer*(*y*, *z*)) visszaadhatja azt is, hogy: {*y*/*János*, *x*/*z*}, vagy azt is, hogy: {*y*/*János*, *x*/*János*, *z*/*János*}. Az első egyesítési lépés azt eredményezi, hogy: *Ismer*(*János*, *z*), ezzel szemben a második azt adja, hogy: *Ismer*(*János*, *János*). A második eredményt megkaphatjuk az elsőből is egy további helyettesítés hozzáadásával: {*z*/*János*}. Láthatjuk, hogy az első egyesítés *általánosabb,* mint a második, mert kevesebb korlátozást ad meg a változók értékeire. Azt állapíthatjuk meg, hogy minden egyesítendő kifejezéspárra létezik egy **legáltalánosabb egyesítés** (**most general unifier**), amely egyedi a változók átnevezésében. Ebben az esetben ez: {*y*/*János*, *x*/*z*}.

A legáltalánosabb egyesítések kiszámításához a 9.1. ábrán láthatunk egy algoritmust. Az eljárás igen egyszerű: egy rekurzív algoritmussal egymással párhuzamosan tárjuk fel a két kifejezést, felépítve az egyesítést, kivéve akkor, ha a struktúrákban a két megfelelő elem nem illeszkedik. Van egy drága lépés: amikor egy változót kell egy komplex termhez illesztünk, meg kell vizsgálni, hogy a változó előfordul-e a termben; ha igen, akkor az illesztés sikertelen, mert nem tudunk konzisztens egyesítést megalkotni. Ez a lépés az úgynevezett **előfordulási próba** (**occur check**), amely az algoritmus komplexitását az egyesítendő kifejezés méretével négyzetesen növekvővé teszi. Néhány rendszer, beleértve az összes logikai programozási rendszert, egyszerűen kihagyja az előfordulási próbát, és ennek eredményeként a következtetésük nem helyes. Más rendszerek időben lineáris komplexitású, ennél összetettebb algoritmusokat használnak.

2.3. Tárolás és visszakeresés

A KIJELENT és a KÉRDEZ függvények, amelyek szerepe, hogy informálják és lekérdezzék a tudásbázist, két egyszerűbb függvényt, a TÁROL és a BETÖLT függvényeket használják fel. A TÁROL (*s*) eltárol egy *s* mondatot a tudásbázisban, míg a BETÖLT (*q*) visszaadja az összes egyesítést, amelyet a *q* lekérdezés felhasználásával lehet létesíteni a tudásbázis illeszthető mondataival. A probléma, amivel illusztráltuk az egyesítést – az összes olyan tény megtalálása, amely egyesíthető az *Ismer*(*János*, *x*)-szel – a BETÖLT egyik példája.

A TÁROL és a BETÖLT megvalósításának legegyszerűbb módja, hogy a tudásbázisban az összes tényt egyetlen hosszú listán tároljuk, és egy *q* lekérdezés megválaszolásakor meghívjuk az EGYESÍT (*q*, *s*)-t minden *s* mondatra a listán. Ez az eljárás nem hatékony, de működik, és egyelőre elég a fejezet további részének megértéséhez. Az

Created by XMLmind XSL-FO Converter.

alfejezet hátralévő része áttekinti azokat az eljárásokat, amelyekkel a visszakeresést hatékonyabbá tehetjük, így ezt első olvasásra átugorhatjuk.

A BETÖLT függvényt azzal tehetjük hatékonyabbá, hogy biztosítjuk, hogy az egyesítési lépést csak olyan mondatokkal kíséreljük meg, amelyeknél van egyáltalán esély az egyesítésre. Például nincs értelme, hogy megpróbáljuk az egyesítést az *Ismer*(*János, x*)-re és a *Fivér*(*Richárd, János*)-re. Elkerülhetjük az ilyen egyesítési lépéseket, ha **indexel**jük (**indexing**) a tényeket a tudásbázisban. Egy egyszerű séma, amit **predikátum indexelés**nek (**predicate indexing**) nevezünk, beteszi az összes *Ismer* tényt egy verembe és az összes *Fivér* tényt egy másikba. A vermek egy hash-táblában[2] tárolhatók, hogy hatékonyan elérhessük őket.

A predikátumok indexelése hasznos, ha sok predikátumszimbólum van, de csak kevés állítás az egyes szimbólumokhoz. Néhány alkalmazásban azonban sok mondat tartozik egy adott predikátumszimbólumhoz. Például tételezzük fel, hogy az adóhatóságok nyomon akarják követni, hogy ki alkalmaz kit, és ezeket a kapcsolatokat egy *Alkalmaz*(*x, y*) predikátum használatával írjuk le. Ez egy igen nagy verem lenne, akár munkaadók millióival és alkalmazottak tízmillióival. Egy olyan kérdés megválaszolása, mint az *Alkalmaz*(*x, Richárd*), a predikátumok indexelésével az egész verem átböngészését igényelné.

Egy ilyen lekérdezés esetében segítene, ha a tényeket mind a predikátummal, mind a második argumentummal indexelnénk, például úgy, hogy egy kombinált hash-tábla kulcsot használunk. A kulcsot egyszerűen előállíthatjuk a lekérdezésből, és pontosan visszakereshetjük közvetlenül azokat a tényeket, amelyek egyesíthetők a lekérdezéssel. Más lekérdezésekhez, mint amilyen az *Alkalmaz*(*AIMA.org, y*), szükségünk van arra, hogy a tényeket a predikátum és az első argumentum kombinálásával indexeljük. Így aztán a tényeket tárolhatjuk többszörös indexkulcsok alatt, azonnal hozzáférhetővé téve minden, a ténnyel egyesíthető lekérdezés számára.

Ha adott egy tárolandó mondat, akkor létre lehet hozni az indexeket az *összes lehetséges* vele egyesíthető lekérdezéshez. Arra a tényre, hogy *Alkalmaz*(*AIMA.org, Richárd*), ezek a lekérdezések:

```
Alkalmaz(AIMA.org, Richárd) Az AIMA.org alkalmazza Richárdot?

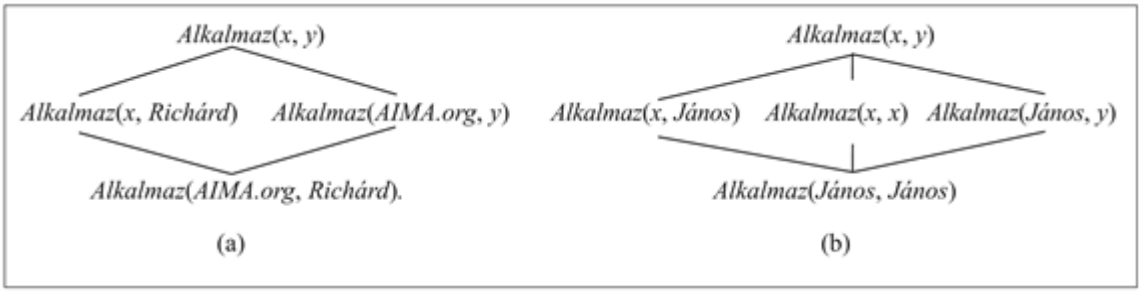
Alkalmaz(x, Richárd) Ki alkalmazza Richárdot?

Alkalmaz(AIMA.org, y) Kit alkalmaz az AIMA.org?

Alkalmaz(x, y) Ki alkalmaz kit?
```

Ezek a lekérdezések egy **bennfoglalási hálót** (**subsumption lattice**) alkotnak, amint az a 9.2. (a) ábrán látható. A hálónak van néhány érdekes tulajdonsága. Például a háló bármely csomópontjának a gyereke a szülőjéből egy egyszerű behelyettesítéssel megkapható, és a „legmagasabb” közös leszármazottja bármely két csomópontnak a legáltalánosabb egyesítőjük használatával érhető el. A háló részeiből bármely alaptény szisztematikusan létrehozható (9.5. feladat). Egy ismétlődő konstansokkal rendelkező mondatnak egy kissé más hálója van, ezt a 9.2. (b) ábrán láthatjuk. A függvényszimbólumok és a változók az eltárolandó mondatokban még érdekesebb hálófelépítéseket tesznek szükségessé.

9.2. ábra - (a) A bennfoglalási háló, amelynek a legalacsonyabb csomópontja ez a mondat: *Alkalmaz(AIMA.org, Richárd)*. (b) A bennfoglalási háló arra a mondatra, hogy: *Alkalmaz(János, János)*.



[2] A hash-tábla információk tárolására és visszakeresésére szolgáló olyan adatstruktúra, amelyeket rögzített *kulcsokkal* indexelünk. Gyakorlati okokból egy hash-táblát tekinthetünk olyannak, mint amelynek a tárolási és a visszakérdezési ideje állandó, akkor is, ha a tábla nagyon nagy számú elemet tartalmaz.

Created by XMLmind XSL-FO Converter.

A séma, amelyet leírtunk, nagyon jól működik, amikor a háló csak kisszámú csomópontot tartalmaz. Egy *n* argumentumú predikátumra a háló $O(2^n)$ csomópontot tartalmaz. Ha a függvényszimbólumok használata megengedett, akkor a csomópontok száma szintén exponenciális a tárolandó mondat termjeinek függvényében. Ez rendkívül nagy számú indexet eredményezhet. Egy ponton az indexelés elveszti előnyeit az összes index tárolásának és fenntartásának problémája miatt. Ez ellen egy rögzített stratégia bevezetésével próbálhatunk védekezni úgy, hogy csak azokon a kulcsokon tartunk fenn indexeket, amelyek predikátumokból és az egyes argumentumokból állnak. Egy másik megoldás, hogy egy adaptív tervet használunk, amely mindig olyan indexeket hoz létre, amelyek megfelelnek az aktuális lekérdezéstípus igényeinek. A legtöbb MI-rendszerben a tárolni kívánt tények száma elég kicsi ahhoz, hogy a hatékony indexelés problémáját megoldottnak tekintsük. Az ipari és kereskedelmi adatbázisok esetében jelentős technikai fejlesztések történtek a feladat megoldására.

3. Előrefelé láncolás

Az ítéletlogikai határozott klózokra már megadtunk egy előrefelé láncolási algoritmust a 7.5. alfejezetben. A gondolat egyszerű: kezdjük a tudásbázisban szereplő atomi mondatokkal, és alkalmazzuk a Modus Ponenst előrefelé haladva, új atomi mondatokat hozzáadva, egészen addig, amíg további következtetések már nem végezhetők. Most megmutatjuk, hogyan alkalmazzuk az algoritmust az elsőrendű határozott klózokra, és hogyan valósíthatjuk ezt meg hatékonyan. A határozott klózok, mint például a *Helyzet* ⇒ *Válasz* különösen hasznosak az olyan rendszerek számára, amelyek újonnan érkezett információk alapján végeznek következtetéseket válaszként. Számos rendszert tervezhetünk ilyen módon, és ezekben az esetekben az előrefelé láncolással történő következtetés sokkal hatékonyabb lehet, mint a rezolúciós tételbizonyítás. Ebből az következik, hogy gyakran érdemes megpróbálni olyan tudásbázist építeni, amely csak határozott klózokat használ, és így elkerülhetjük a rezolúcióval járó nehézségeket.

3.1. Elsőrendű határozott klózok

Az elsőrendű határozott klózok nagyon emlékeztetnek az ítéletlogikai határozott klózokra (4.1.3. szakasz - A rezolúció teljességerészben). Az ilyen klózok olyan literálok diszjunkciói, amelyek közül *pontosan egy diszjunkt pozitív*. Egy határozott klóz vagy egy atomi mondat, vagy egy implikáció, amelynek a feltétel része pozitív literálok konjunkciója, és amelynek következménye egyetlen pozitív literál. A következő mondatok elsőrendű határozott klózok:

$$Király(x) \land Mohó(x) \Rightarrow Gonosz(x)$$

$$Király(János)$$

$$Mohó(y)$$

Eltérően az ítéletlogikai literáloktól, az elsőrendű literálok tartalmazhatnak változókat, amely esetben a változókat univerzális kvantorral ellátottnak tételezzük fel. (A határozott klózok írásánál általában elhagyjuk az univerzális kvantorokat.) A határozott klóz egy megfelelő normál forma, hogy az általánosított Modus Ponensszel alkalmazhassuk. Nem minden tudásbázist lehet átalakítani határozott klózok halmazává, az egyetlen pozitív literális korlátja miatt, de sokat igen. Gondoljuk át a következő problémát:

A törvény kimondja, hogy bűntény az, ha egy amerikai polgár fegyvert ad el egy Amerikával ellenséges nemzetnek. A Nono ország, amely ellensége Amerikának, fel van szerelve rakétákkal, és ezeket a rakétákat mind West ezredes adta el, aki amerikai.

Be fogjuk bizonyítani, hogy West bűnöző. Először leírjuk a tényeinket elsőrendű határozott klózokként. A következő alfejezet fogja bemutatni, hogy az előrefelé láncolás algoritmus hogyan oldja meg a problémát.

„…bűntény az, ha egy amerikai polgár fegyvert ad el egy Amerikával ellenséges nemzetnek”:

$$Amerikai(x) \land Fegyver(y) \land Ellenséges(z) \land Elad(x, z, y) \Rightarrow Bűnöző(x) \quad (9.3)$$

„Nono… fel van szerelve rakétákkal.” Azt a mondatot, hogy $\exists x\ Birtokol(Nono, x) \land Rakéta(x)$ átalakítjuk két határozott klózzá Egzisztenciális Eliminációval, egy új konstans, az M_1 bevezetésével:

$$Birtokol(Nono, M_1) \quad (9.4)$$

$$Rakéta(M_1) \quad (9.5)$$

Created by XMLmind XSL-FO Converter.

„És ezeket a rakétákat mind West ezredes adta el”:

$$Rakéta(x) \land Birtokol(Nono, x) \Rightarrow Elad(West, x, Nono) \quad (9.6)$$

Tudnunk kell még, hogy a rakéták fegyverek:

$$Rakéta(x) \Rightarrow Fegyver(x) \quad (9.7)$$

és hogy Amerika ellensége „ellenségesnek” számít:

$$Ellensége(x, Amerika) \Rightarrow Ellenséges(x) \quad (9.8)$$

„West ezredes…, aki amerikai”:

$$Amerikai(West) \quad (9.9)$$

„Nono ország, amely ellensége Amerikának…”

$$Ellensége(Nono, Amerika) \quad (9.10)$$

A tudásbázis nem tartalmaz függvényszimbólumokat, és így egy példája a **Datalog** (**Datalog**) tudásbázisok osztályának – ami függvényszimbólumok nélküli elsőrendű határozott klózok halmaza. Látni fogjuk, hogy a függvényszimbólumok hiánya sokkal könnyebbé teszi a következtetést.

3.2. Egy egyszerű előrefelé láncolási algoritmus

Az első előrefelé láncolási algoritmus, amit megvizsgálunk, nagyon egyszerű lesz, amint azt láthatjuk a 9.3. ábrán. Az algoritmus az ismert tényekből kiindulva végrehajtja az összes olyan szabályt, amelynek premisszái ki vannak elégítve, és ezek következményeit hozzáadja az ismert tényekhez. Ez az eljárás addig ismétlődik, amíg a kérdést meg nem válaszoltuk (feltételezve, hogy csak egy válaszra van szükségünk), vagy amikor már nem tudunk új tényeket a tudásbázishoz hozzáadni. Vegyük észre, hogy egy tény nem számít „újnak”, ha csak **átnevezése** (**rename**) egy már ismert ténynek. Egy mondat átnevezése egy másiknak, ha az alkalmazott változók neveitől eltekintve megegyeznek. Például a *Szereti*(*x*, *Fagylalt*) és a *Szereti*(*y*, *Fagylalt*) mondatok egymás átnevezésének számítanak, mert csak az *x* és az *y* választásában különböznek. A két mondat jelentése megegyezik: mindenki szereti a fagylaltot.

Az előző bűntényproblémánkat fogjuk felhasználni az ERL-EL-KÉRDEZ algoritmus bemutatásához. Az implikációs mondatok a (9.3), (9.6), (9.7) és a (9.8). Két iterációs lépés szükséges:

- Az első iterációban a (9.3) szabály nem elégítette ki a premisszákat.

A (9.6) szabályt kielégíti az {x/M_1} és az *Elad*(*West*, M_1, *Nono*) hozzáadása.

A (9.7) szabályt kielégíti az {*x*/M_1} és a *Fegyver*(M_1) hozzáadása.

A (9.8) szabályt kielégíti az {*x*/*Nono*} és az *Ellenséges*(*Nono*) hozzáadása.

- A második iterációban a (9.3) szabályt kielégíti az {*x*/*West*, *y*/M_1, z/*Nono*), és hozzáadjuk azt, hogy: *Bűnöző*(*West*).

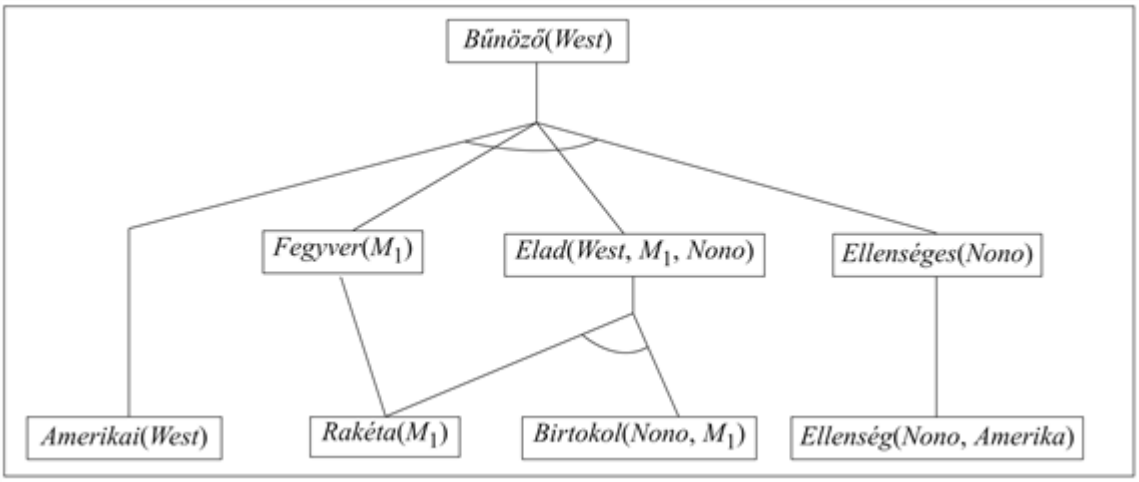
9.3. ábra - Egy koncepcionálisan egyszerű, de kevéssé hatékony előrefelé láncolási algoritmus. Minden egyes iterációban hozzáadja a *TB*-hez az összes atomi mondatot, amelyet egy lépésben az implikációs mondatokból és a *TB*-ben már meglévő atomi mondatokból kikövetkeztethetünk.

Created by XMLmind XSL-FO Converter.

```
function ERL-EL-KÉRDEZ(TB, α) returns egy helyettesítés, vagy hamis
  inputs: TB, a tudásbázis, elsőrendű határozott klózok halmaza
          α, a lekérdezés, egy atomi mondat
  local variables: új, a minden egyes iterációk során kikövetkeztetett új mondatok

  repeat until az új üres
    new ← { }
    for each r mondatra in TB do
      (p1 ∧ ... ∧ pn ⇒ q) ← VÁLTOZÓ-ÁTI EVEZ(r)
      for each θ olyan mint HELYETTESÍT(θ, p1 ∧ ... ∧ pn) = HELYETTESÍT(θ, p1' ∧ ... ∧ pn')
            néhány p1', ..., pn'-re a TB-ben
        q' ← HELYETTESÍT(θ, q)
        if q' nem egy már a TB-ben meglévő mondat átnevezése vagy új then do
          hozzáad q' az új-hoz
          φ ← EGYESÍT(q', α)
          if φ nem sikertelen then return φ
    hozzáad új-at a TB-hez
  return hamis
```

9.4. ábra - Az előrefelé láncolás algoritmus által generált bizonyítási fa a bűntény példára. A kiinduló tények az alsó sorban jelennek meg, az első iterációban kikövetkeztetett tények a középső sorban, míg a második iterációban kikövetkeztetett tények a legfelső sorban.



A 9.4. ábra bemutatja a generált bizonyítási fát. Vegyük észre, hogy új következtetések már nem lehetségesek ezen a ponton, mivel minden mondatot, amelyet kikövetkeztethetnénk az előrefelé láncolással, már explicit módon tartalmaz a *TB*. Egy ilyen tudásbázist a következtetési folyamat **fix pont**jának (**fixed point**) nevezzük. Az elsőrendű határozott klózok előrefelé láncolásával létrehozott fix pontok hasonlítanak az ítéletlogikai előrefelé láncolással (4.2. szakasz - Előre- és hátrafelé láncolásrészben) generáltakhoz. A leglényegesebb különbség az, hogy egy elsőrendű fix pont tartalmazhat univerzális kvantorral ellátott atomi mondatokat.

A ERL-EL-KÉRDEZ-t könnyű kielemezni. Először is megállapítható, hogy az eljárás **helyes** (**sound**), mivel minden következtetés csak az általánosított Modus Ponens alkalmazása, amelyről már igazoltuk, hogy helyes. Másodszor, **teljes** (**complete**) a határozott klózokat tartalmazó tudásbázisokra, ami azt jelenti, hogy képes minden olyan lekérdezést megválaszolni, amely következik bármely határozott kózokból álló tudásbázisból. A Datalog tudásbázisok esetére, amelyek nem tartalmaznak függvényszimbólumokat, a teljesség bizonyítása meglehetősen egyszerű. Először megszámoljuk a *TB*-hez hozzáadható tényeket, amely szám meghatározza az iterációk maximális számát. Legyen a k ez a maximális érték (argumentumok száma) az adott predikátumokra, p a predikátumok száma és n a konstansszimbólumok száma. Egyértelmű, hogy nem lehet pn^k különböző alapténynél több, tehát ennyi iteráció után az algoritmus el fog érni egy fix pontot. Ezután már az ítéletlogikai

Created by XMLmind XSL-FO Converter.

előrefelé láncolás bizonyításánál leírtakhoz nagyon hasonlóan érvelhetünk (4.2. szakasz - Előre- és hátrafelé láncolásrészben.) Az ítéletlogikai teljes eljárás átalakítását egy elsőrendű teljes eljárássá a 9.5. alfejezetben mutatjuk majd meg.

Függvényszimbólumokat is tartalmazó általános határozott klózok esetében az ERL-EL-KÉRDEZ végtelen számú új tényt generálhat, így ilyenkor óvatosabbnak kell lennünk. Ha egy lekérdezésre adható válaszmondat levezethető a *TB*-ből, akkor Herbrand tételét kell alkalmaznunk, hogy biztosítsuk, hogy az algoritmus megtalál egy bizonyítást (lásd a 9.5. alfejezetet a rezolúciós esetre). Ha a lekérdezésre nincs válasz, akkor az algoritmus néhány esetben nem tud leállni. Például ha a tudásbázis a Peano-axiómákat tartalmazza,

$$TermSzám(0)$$

$$\forall n \; TermSzám(n) \Rightarrow TermSzám(S(n))$$

akkor az előrefelé láncolás hozzáadja a $TermSzám(S(0))$, $TermSzám(S(S(0)))$, $TermSzám(S(S(S(0))))$ mondatokat és így tovább. Általánosságban ezt a problémát nem lehet kikerülni. Hasonlóan, mint az általános elsőrendű logikában, a határozott klózokkal való következtetés félig eldönthető.

3.3. Hatékony előrefelé láncolás

A 9.3. ábrán látható előrefelé láncolási algoritmust inkább a megértés megkönnyítése céljából mutattuk be, és nem mint egy hatékonyan végrehajtható algoritmust. A komplexitásnak három lehetséges forrása van. Először is, az algoritmus „belső hurka” elvégzi az összes lehetséges egyesítés megtalálását, ahol egy szabály premisszája egyesíthető a *TB* egy alkalmas tényhalmazával. Ezt gyakran **mintaillesztés**nek (**pattern matching**) nevezzük, és igen költséges lépés. Másodszor, az algoritmus újra ellenőriz minden szabályt minden iterációban, hogy megvizsgálja, hogy a premisszák ki vannak-e elégítve, még olyankor is, amikor nagyon kevés változtatást végzünk a tudásbázisban az egyes ciklusokban. Végül, az algoritmus számos olyan tényt is generálhat, melyek irrelevánsak a cél szempontjából. Mindhárom problémaforrást meg fogjuk vizsgálni.

3.3.1. Szabályok illesztése az ismert tényekhez

A szabályok premisszáinak a *TB* tényeihez történő illesztése egyszerű problémának tűnhet. Például tételezzük fel, hogy a következő szabályt akarjuk alkalmazni:

$$Rakéta(x) \Rightarrow Fegyver(x)$$

Ezután meg kell találnunk az összes tényt, amely egyesíthető a $Rakéta(x)$-szel. Egy megfelelően indexelt tudásbázisban ez elvégezhető a tények számával lineáris időben. Vizsgáljunk meg egy szabályt, mint amilyen például a:

$$Rakéta(x) \wedge Birtokol(Nono, x) \Rightarrow Elad(West, x, Nono)$$

Itt is megkereshetjük a Nono által birtokolt összes objektumot objektumonként konstans idő alatt. Ezután minden egyes objektumra meg tudjuk vizsgálni, hogy az rakéta-e.

Amennyiben a tudásbázis sok Nono által birtokolt objektumot tartalmaz és nagyon kevés rakétát, akkor azonban célszerűbb lenne először az összes rakétát megkeresni, és aztán ellenőrizni, hogy azokat Nono birtokolja-e. Ez a **konjunkt sorrendezés** (**conjunct ordering**) probléma: találjunk egy olyan sorrendet, amely a szabály premissza részének konjunktjait megoldja úgy, hogy ezzel a teljes költséget minimalizáljuk. Megmutatható, hogy az optimális sorrend megtalálása önmagában is NP-nehéz, de egy jó heurisztika elérhető. Például ilyen a **legkorlátozottabb változó** (**most constrained variable**) heurisztikája, amelyet az 5. fejezetben a kényszerkielégítési problémákra használtunk. Ez azt a logikát követi, hogy rendezzük sorba a konjunktokat úgy, hogy először a rakétákat keressük, ha kevesebb rakéta van, mint Nono által birtokolt objektum.

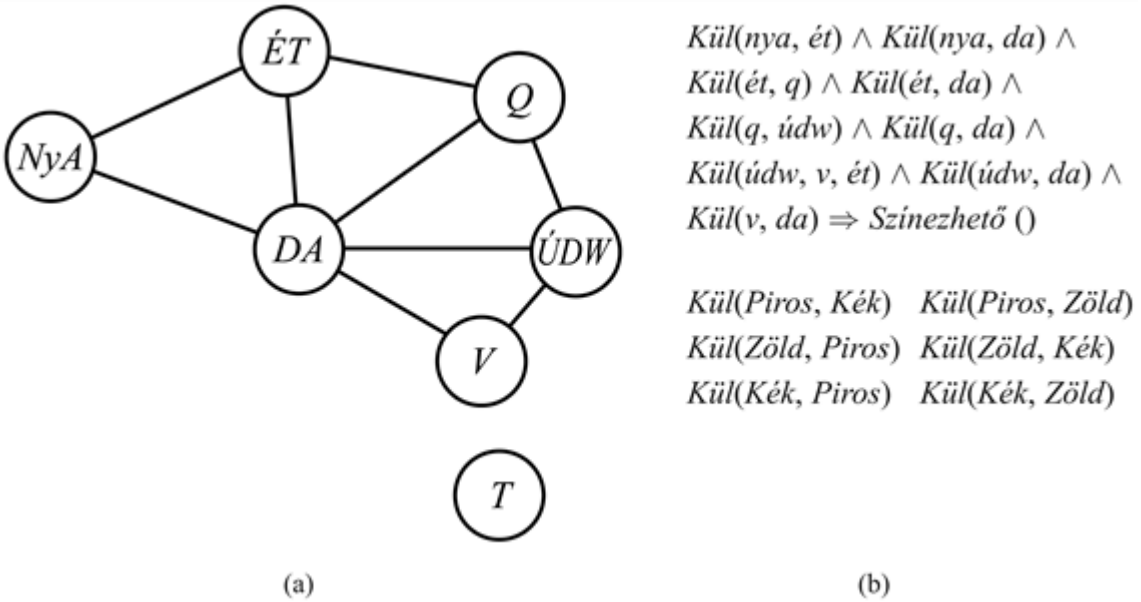
Fontos

A kapcsolat a mintaillesztés és a kényszer kielégítése között valójában nagyon szoros. Bármelyik konjunktot úgy tekinthetünk, mint az általa tartalmazott változók korlátozását – például, a $Rakéta(x)$ egy unáris korlátozás az x-en. Ezt a gondolatot kiterjesztve, *minden véges tárgyterületű kényszerkielégítési problémát kifejezhetünk egyszeri határozott klózként, néhány társított alapténnyel kiegészítve.* Vizsgáljuk meg az 5.1. ábrán látható térképszínezési problémát, amelyet a 9.5. (a) ábrán újra bemutatunk. Egy ezzel megegyező formulát adtunk meg egy határozott klóz formájában a 9.5. (b)

Created by XMLmind XSL-FO Converter.

ábrán. Világos, hogy a *Színezhető*() konklúzió csak akkor kikövetkeztethető, ha a kényszerkielégítési problémának van egy megoldása. Mivel a kényszerkielégítési problémák általában magukba foglalják a 3SAT problémákat különleges esetekként, levonhatjuk azt a következtetést, hogy *egy határozott klóz illesztése egy tényhalmazhoz NP-nehéz.*

9.5. ábra - (a) Ausztrália térképének kiszínezését bemutató kényszergráf (5.1. ábra). (b) A térképszínező kényszerkielégítési probléma egy határozott klózzal reprezentálva. Vegyük észre, hogy a változók tárgyterülete a KÜL-re megadott alaptények konstansai által implicit módon definiáltak.



Elég elkeserítőnek tűnhet, hogy az előrefelé láncolás tartalmaz egy NP-nehéz illesztési problémát a belső hurokban. Három módja van annak, hogy felvidítsuk magunkat:

- Emlékezhetünk arra, hogy a legtöbb szabály a valódi tudásbázisokban kisméretű és egyszerű (mint a bűntény példa szabályai), és nem nagy és komplex (mint a 9.5. ábrán látható kényszerproblémánál). Az adatbázisok területén fel szokták tételezni, hogy mind a szabályok mérete, mind a predikátumok argumantumszáma egy konstanssal megadható korlát alatt marad, és így csak az **adatkomplexitás** (**data complexity**) miatt kell aggódni – vagyis a következtetés komplexitása miatt, ami az adatbázisban lévő alaptények számának függvénye. Könnyű megmutatni, hogy az előrefelé láncolás adatkomplexitása polinomiális.

- Tekinthetjük a szabályok azon csoportját, amelyekre az illesztés hatékony tud lenni. Alapjában véve minden Datalog klózt tekinthetünk úgy, mint ami egy kényszerkielégítési problémát határoz meg, így az illesztés kivitelezhető akkor, ha a megfelelő kényszerkielégítési probléma is nyomon követhető. Az 5. fejezet leírja a kényszerkielégítési problémák néhány praktikusan megoldható családját. Például ha a kényszergráf (egy olyan gráf, amelynek a csomópontjai változók és az élei kényszerek) fát formáz, akkor a kényszerkielégítési probléma lineáris időben megoldható. Pontosan ugyanez a szabály áll fenn a szabályillesztésre. Például ha eltávolítjuk Dél-Ausztráliát a 9.5. ábráról, akkor az új klóz a következő lesz:

```
Kül(nya, ét) ∧ Kül(ét, q) ∧ Kül(q, údw) ∧ Kül(údw, v) ⇒ Színezhető()
```

Ez megfelel az 5.11. ábrán bemutatott redukált kényszerkielégítési problémának. A faszerkezetű kényszerkielégítési problémák megoldására használt algoritmusokat közvetlenül alkalmazhatjuk a szabályillesztés problémájára.

- És végül dolgozhatunk azon, hogy megszüntessük a felesleges szabályillesztési kísérleteket az előrefelé láncolási algoritmusban, amely a következő alfejezet témája lesz.

3.3.2. Inkrementális előrefelé láncolás

Fontos

Created by XMLmind XSL-FO Converter.

Amikor a bűntény példán bemutattuk az előrefelé láncolás működését, akkor csaltunk, nevezetesen abban, hogy kihagytunk néhány szabályillesztést, amelyet a 9.3. ábrán bemutatott algoritmus elvégzett. Például a második iterációban a:

```
Rakéta(x) ⇒ Fegyver(x)
```

szabály (ismét) illeszthető a *Rakéta*(M_1)-hez, és természetesen a *Fegyver*(M_1) konklúziót már ismerjük, így semmi sem történik. Az ilyen felesleges szabályillesztést elkerülhetjük, ha figyelembe vesszük a következő megfigyelést: *Minden, α t-edik iterációban kikövetkeztethető új tény levezetéséhez szükséges legalább egy, α t – 1 ciklusban kikövetkeztetett új tény felhasználása.* Ez azért igaz, mert bármely olyan következtetés, amely nem igényel egy új tényt a $t - 1$ ciklusból, már elvégezhető lett volna a $t - 1$ ciklusban.

Ez a megfigyelés természetes módon elvezet minket egy inkrementális előrefelé láncolási algoritmushoz, ahol a t ciklusban csak akkor ellenőrzünk egy szabályt, ha annak premisszája tartalmaz egy p_i konjunktot, amely egyesíthető egy p'_i ténnyel, és amelyre újonnan következtettünk a $t - 1$ ciklusban. A szabályillesztő lépés aztán rögzíti a p_i-t, hogy illeszkedjen a p'_i-hez, de lehetővé teszi, hogy a szabály többi konjunktja illeszkedjen bármely megelőző ciklus tényeihez. Ez az algoritmus pontosan ugyanazokat a tényeket generálja minden egyes ciklusban, mint amelyeket a 9.3. ábrán látható algoritmus, de annál sokkal hatékonyabb.

Megfelelő indexeléssel könnyű megtalálni azokat a szabályokat, amelyeket egy adott tény kielégíthetővé tehet. Valójában számos rendszer egy ilyen frissítési módban működik, ahol az előrefelé láncolás minden egyes olyan tényre aktualizálódik, amelyet KIJELENT-ettünk a rendszernek. A következtetések sorban végigveszik a szabályok halmazát, amíg el nem érik a fix pontot, és ez a folyamat a következő új ténynél újra kezdődik.

A tudásbázisban lévő szabályoknak tipikusan csak egy kis töredékét eredményezi egy adott tény hozzáadása. Ez azt jelenti, hogy jelentős mennyiségű felesleges munkát végzünk néhány ki nem elégített premisszát is tartalmazó részleges illesztések ismételt létrehozásával. A bűntény példánk túl kicsi ahhoz, hogy ezt megfelelően bemutassuk, de vegyük észre, hogy egy részleges illesztést az első ciklusban már létrehoztunk az:

```
Amerikai(x) ∧ Fegyver(y) ∧ Elad(x, y, z) ∧ Ellenséges(z) ⇒ Bűnöző(x)
```

szabály és az *Amerikai*(*West*) tény között. Ezt a részleges illesztést aztán kiselejtezzük, de újraépítjük a második ciklusban is (amikor a szabály sikeres). Hatékonyabb lenne megőrizni, és fokozatosan kiegészíteni újabb részleges illesztésekkel, amikor az új tények beérkeznek, ahelyett hogy kiselejteznénk őket.

A **rete** algoritmus[3] volt az első, amely alaposan foglalkozott ezzel a problémával. Az algoritmus feldolgozza a tudásbázis szabályait, hogy létrehozzon egy adatfolyam-hálózatot, amelyben minden csomópont egy literál egy szabály premissza részéből. Változó lekötések áramlanak a hálózaton, megszűrve azokat, amelyek nem illeszkednek egy literálhoz. Ha egy szabályban két literálnak ugyanaz a változója – például *Elad*(*x*, *y*, *z*) ∧ *Ellenséges*(*z*) a bűnöző példában –, akkor a literálokhoz tartozó lekötések egy egyenlőségi csomóponton mennek keresztül. Egy változó lekötésnek, amely elér egy csomópontot egy *n* argumentumú literálnál úgy, mint az *Elad*(*x*, *y*, *z*), várakoznia kell, mielőtt a folyamat újra kezdődne, amíg a többi változóhoz tartozó lekötéseket is létrehozzuk. Egy adott pontban a rete háló állapota megadja a szabályok összes, addig elvégzett részleges illesztéseit, és így elkerülhető a jelentős újraszámolás.

A rete hálók, és más, hatékonyságot növelő fejlesztések mindig is jelentős szerepet játszottak az úgynevezett **produkciós rendszer**ekben (**production system**s), amelyek az első széles körben használt előrefelé láncolási rendszerek voltak.[4] Az XCON szakértői rendszer (eredetileg R1-nek hívták, McDermott, 1982) egy produkciós rendszer felépítését felhasználva készült. Az XCON néhány ezer szabályt tartalmazott számítógép-tartozékok konfigurációinak megtervezésére a DEC cég vásárlóinak számára. Ez volt az egyik első igazi kereskedelmi siker a szakértő rendszerek feltörekvő piacán. Sok más hasonló rendszer épült ugyanezt a technológiát felhasználva, amelyet be is építettek egy általános célú programozási nyelvbe, az OPS-5-be.

A produkciós rendszerek a **kognitív architektúrá**kban (**cognitive architecture**s), mint például az ACT (Anderson, 1983) vagy a SOAR (Laird és társai, 1987), is népszerűek. A kognitív architektúrák az emberi gondolkodás modelljei. Az ilyen rendszerekben a rendszer „munkamemóriája” az ember rövid távú memóriáját modellezi, és a produkált következmények a hosszú távú memória részei. A működés minden egyes ciklusában a produkciókat a tények munkamemóriájához illesztik. Egy olyan következmény, amelynek feltételeit kielégítik,

[3] A rete hálót jelent latinul, angol kiejtésben (riti) a „treaty”-vel rímel.
[4] A **produkció** kifejezés a **produkciós rendszer**ekben egy feltétel–cselekvés szabályt jelent.

Created by XMLmind XSL-FO Converter.

hozzátehet és kitörölhet tényeket a működő memóriából. Az adatbázisokkal ellentétben a produkciós rendszerekben gyakran sok szabály és viszonylag kevés tény van. Megfelelően optimalizált illesztési technológiával néhány modern rendszer képes valós időben működni több mint egymillió szabállyal is.

3.3.3. Irreleváns tények

Úgy tűnik, hogy a rossz hatékonyság problémájának az utolsó forrása az előrefelé láncolásban a megközelítésből fakad, ez már az ítéletlogikai kontextusban is felmerült. (lásd 7.5. alfejezet). Az előrefelé láncolás az ismert tényeken alapuló összes lehetséges következtetést elvégzi, *akkor is, ha azok irrelevánsak az elérendő célhoz.* A bűntény példánkban nem voltak olyan szabályok, amelyek irreleváns konklúziókat vontak volna maguk után, így az irányítottság hiánya nem jelentett problémát. Más esetekben (például ha több szabályunk van, amelyek leírják az amerikaiak étkezési szokásait és a rakéták árát is), a ERL-EL-KÉRDEZ számos irreleváns konklúziót fog generálni.

Az irreleváns konklúziók elkerülésének egy lehetséges módja a hátrafelé láncolás használata, amint az a 9.4. alfejezetben látható lesz. Másik megoldás, hogy az előrefelé láncolást a kiválogatott szabályok egy részhalmazára korlátozzuk. Ezt a megközelítést az ítéletlogikai kontextusban már tárgyaltuk. Egy harmadik megközelítést használnak a deduktív adatbázisok területén, ahol az előrefelé láncolás elterjedt eszköznek számít. Az alapötlet az, hogy írjuk át a szabályhalmazt felhasználva a célállítást, hogy aztán csak releváns változó kapcsolatokat – amelyek az úgynevezett **mágikus halmaz**hoz (**magic set**) tartoznak – vegyük figyelembe az előrefelé következtetésben. Például ha a cél állítás a *Bűnöző*(*West*), akkor a szabályt, amely a *Bűnöző*(*x*)-re következtet, át kell írni úgy, hogy tartalmazzon egy további konjunktot, amely korlátozza az *x* értékét:

$$Mágikus(x) \wedge Amerikai(x) \wedge Fegyver(y) \wedge Elad(x, y, z) \wedge Ellenséges(z) \Rightarrow Bűnöző(x)$$

A *Mágikus*(*West*) tény is hozzáadódik a *TB*-hez. Ily módon, még akkor is, ha a tudásbázis amerikaiak millióiról tartalmaz is adatokat, csak West ezredest fogjuk figyelembe venni az előrefelé láncolási folyamatban. A mágikus halmazok definiálásának és a tudásbázis átírásának teljes folyamata túl összetett ahhoz, hogy most itt részletezzük, de az alapötlet egyfajta „generikus” hátrafelé való következtetés elvégzése a célból kiindulva azért, hogy megtaláljuk, mely változókapcsolatokat kell korlátozni. A mágikus halmaz megközelítést egy hibrid algoritmusnak tekinthetjük az előrefelé következtetés és a hátrafelé haladó előfeldolgozó folyamat között.

4. Hátrafelé láncolás

A logikai következtetési algoritmusoknak a második nagy családja a 7.5. alfejezetben bemutatott **hátrafelé láncolás** (**backward chaining**) megközelítést használja. Ezek az algoritmusok a célból kiindulva hátrafelé dolgoznak, láncszerűen haladva a szabályokon keresztül, hogy megtalálják a bizonyítást alátámasztó már ismert tényeket. Először bemutatjuk az alapvető algoritmust, majd leírjuk, hogyan használják ezt a **logikai programozás**ban (**logic programming**), amely a legelterjedtebb formája az automatizált következtetéseknek. Azt is látni fogjuk, hogy a hátrafelé láncolásnak vannak hátrányai is az előrefelé láncolással szemben, és meg fogjuk vizsgálni, milyen módon kerekedhetünk felül ezeken. Végül megmutatjuk a logikai programozás és a kényszerek kielégítésének problémája közötti szoros kapcsolatot.

4.1. Egy hátrafelé láncolási algoritmus

A 9.6. ábra egy egyszerű hátrafelé láncolási algoritmust mutat be, az ERL-HL-KÉRDEZ-t. Az algoritmust célok listájával hívjuk meg, amely kezdetben egy elemet tartalmaz, az eredeti lekérdezést, és az eljárás visszaadja az összes olyan helyettesítéshalmazt, amelyek kielégítik a lekérdezést. A célok listáját úgy tekinthetjük, mint egy feldolgozásra váró „vermet”. A bizonyítás aktuális ága akkor lesz sikeres, ha a verem *összes elemét* ki tudjuk elégíteni. Az algoritmus először veszi a lista első célját, és megtalálja az összes olyan klózt a tudásbázisban, amelynek a pozitív literálja, vagyis a **feje** (**head**) egyesíthető a céllal. Minden ilyen klóz létrehoz egy új rekurzív hívást, amelyben a premissza, vagyis a klóz **törzse** (**body**) hozzá lesz adva a célveremhez. Emlékezzünk, hogy a tények olyan klózok, amelyeknek fejük van, de törzsük nincs, így amikor egy cél egyesül egy ismert ténnyel, nem adunk új alcélokat a veremhez, és így a célt megoldjuk. A 9.7. ábra a *Bűnöző*(*West*) származtatásának a bizonyítási fáját mutatja be, a (9.3) mondataiból kiindulva a (9.10)-en keresztül.

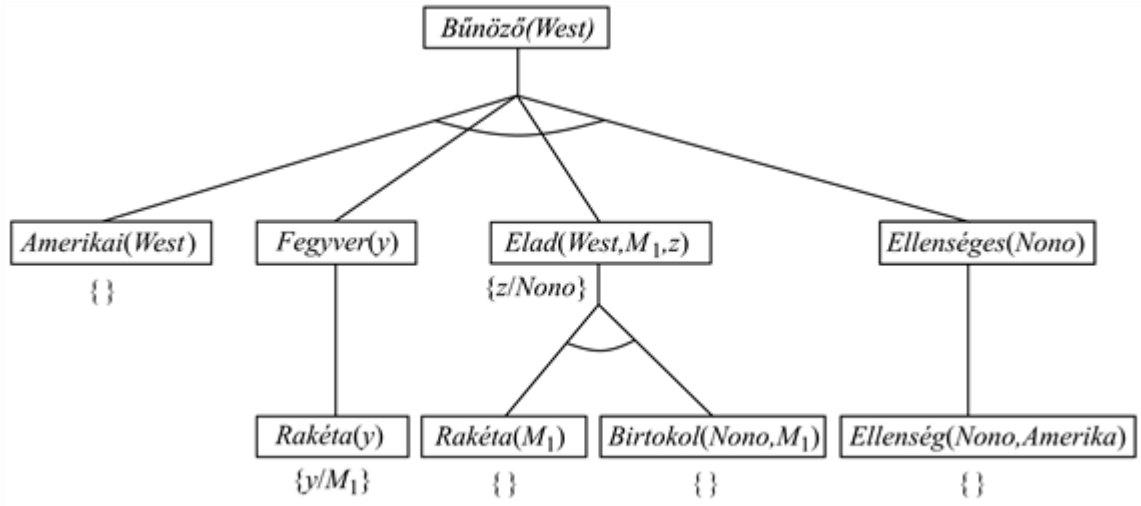
9.6. ábra - Egy egyszerű hátrafelé láncolási algoritmus

Created by XMLmind XSL-FO Converter.

```
function ERL-HL-KÉRDEZ(TB, célok, θ) returns helyettesítések egy halmaza
  inputs: TB, egy tudásbázis
          célok, egy lekérdezést alkotó konjunkciók egy listája, ahol θ már alkalmazva van
          θ, az aktuális helyettesítés, kezdetben az üres helyettesítés {}
  local variables: válaszok, helyettesítések egy halmaza, kezdetben üres

  if célok üres then return {θ}
  q' ← HELYETTESÍTÉS(θ, ELSŐ(célok))
  for each r mondatra in TB, ahol VÁLTOZÓ-ÁTI EVEZ(r) = (p1 ∧ ... ∧ pn ⇒ q)
          és θ' ← EGYESÍT(q, q') sikeres
      új_célok ← [ p1, ..., pn |MARADÉK(célok)]
      válaszok ← ERL-HL-KÉRDEZ(TB, új_célok, KOMPOZÍCIÓ(θ', θ)) ∪ válaszok
  return válaszok
```

9.7. ábra - Egy hátrafelé láncolással létrehozott bizonyítási fa, annak bizonyítására, hogy West egy bűnöző. A fát mélységi kereséssel kell kiértékelni, balról jobbra. Hogy bebizonyítsuk a *Bűnöző*(*West*) klózt, be kell bizonyítanunk az alatta elhelyezkedő négy konjunktot is. Néhányan közülük megtalálhatók a tudásbázisban, míg mások további hátrafelé láncolást igényelnek. Mindegy egyes sikeres egyesítés lekötéseit láthatjuk a megfelelő részcél mellett. Jegyezzük meg, hogy mihelyt egy konjunkcióban egy részcél sikeres, helyettesítéseit a következő részcélokhoz alkalmazzuk. Így tehát, mire az ERL-HL-KÉRDEZ eljut az utolsó konjunkcióig, az eredetileg *Ellenséges*(*z*)-ig, az már a *Nono*-hoz lesz lekötve.



Az algoritmus a helyettesítések egy **kompozíció**ját (**composition**) használja. A KOMPOZÍCIÓ (θ_1, θ_2) olyan helyettesítés, amelynek hatása ugyanaz, mintha sorban minden helyettesítést alkalmaznánk. Tehát:

`Helyettesít(Kompozíció(θ1, θ2), p) = Helyettesít(θ2, Helyettesít(θ1, p))`

Az algoritmusban a θ-ban tárolt aktuális változólekötéseket összevonjuk azokkal a lekötésekkel, amelyeket a célnak a klózfejjel való egyesítése eredményez, megkapva így egy új aktuális lekötéshalmazt a visszafelé történő híváshoz.

A hátrafelé láncolás, ahogyan bemutattuk, egy tiszta mélységi keresési algoritmus. Ez azt is jelenti, hogy a helyigénye a bizonyítás méretének lineáris függvénye (most figyelmen kívül hagyva a megoldások összegyűjtéséhez szükséges helyet). Ez azt is jelenti, hogy a hátrafelé láncolás alkalmazása (eltérően az előrefelé láncolástól) olyan ismert problémákkal jár, mint amilyen az ismétlődő állapotok és a nem teljesség. Később meg fogjuk vizsgálni ezeket a problémákat és néhány lehetséges megoldást, de először nézzük meg, hogyan használják a hátrafelé láncolást a logikai programozási rendszerekben.

Created by XMLmind XSL-FO Converter.

4.2. Logikai programozás

A logikai programozás közel áll egy olyan technológiához, amely megtestesíti azt a deklaratív ideált, amelyet a 7. fejezetben írtunk le, miszerint a rendszereket úgy kell létrehozni, hogy a tudást formális nyelven fejezzük ki, és a problémákat egy következtetési folyamat végigfuttatásával oldjuk meg. Ezt az ideális megközelítést Robert Kowalski egyenlete foglalja össze:

```
Algoritmus = Logika + Vezérlés
```

A **Prolog** messze a legszélesebb körben használt logikai programozási nyelv. Több százezer használója van. Elsősorban gyors prototípusnyelvként használták, valamint szimbólummanipulációs feladatokhoz, mint például fordítóprogramok írására (Van Roy, 1990) vagy természetes nyelvek elemzésére (Pereira és Warren, 1980). Számos szakértőrendszert írtak Prologban jogi, orvosi, pénzügyi és más tárgyterületeken.

A Prolog programok határozott klózok halmazai, jelölési rendszere kissé eltér a standard elsőrendű logikáétól. A Prolog nagybetűket használ a változókra és kisbetűket a konstansokra. A klózok leírásában a fej megelőzi a törzset; a „:-” -t használja a bal oldali implikációra, vesszők választják el a literálokat a törzsben, és pont jelzi a mondat végét:

```
bűnöző(X):- amerikai(X), fegyver(Y), elad(X,Y,Z), ellenséges(Z)
```

A Prolog a listák jelölésére és az aritmetikára tartalmazza a „szintaktikai nyalánkságot”. Példaként íme egy Prolog program a csatol(X,Y,Z)-re, amely akkor sikeres, ha a Z- lista az X- és az Y -listák csatolásának az eredménye:

```
csatol([],Y,Y)

csatol([A1|2 X, Y, A½Z]):- csatol(X,Y,Z)
```

Természetes nyelven úgy olvashatóak ezek a klózok, hogy (1) egy üres listát hozzácsatolva az Y listához, egy ugyanolyan Y listát kapunk eredményként, és a (2) szerint az [A|Z] az [A|X] Y-hoz csatolásának az eredménye. A csatol ezen definíciója eléggé hasonlónak tűnik a Lispben megtalálható megfelelő definíciókhoz, de valójában ez sokkal kifejezőbb. Például feltehetjük a következő kérdést: csatol(A,B, [1, 2]): melyik két listát kell csatolni ahhoz, hogy összeállítsuk az [1, 2]-t? A programot futtatva ezeket a megoldásokat kapjuk vissza:

```
A = [] B = [1, 2]

A = [1] B = [2]

A = [1, 2] B = []
```

A Prolog-programok végrehajtása mélységi keresést alkalmazó hátrafelé láncoláson keresztül történik, ahol a klózokat olyan sorrendben alkalmazzák, ahogyan a tudásbázisban szerepelnek. A Prolog néhány aspektusa a standard logikai következtetésen kívül esik.

- Beépített aritmetikai függvényeket tartalmaz. Az ilyen függvényszimbólumokat használó literálokat a kód végrehajtásával „bizonyítja” a program, és nem végez további következtetéseket. Például az „X egyenlő 4+3” akkor sikeres, ha az X értékét a 7-hez kötjük. Ugyanakkor, az a célállítás, hogy: „5 egyenlő X+Y” sikertelen lesz, mert a beépített függvények nem képesek önállóan egy tetszőleges egyenletmegoldást elvégezni.[5]

- Vannak olyan beépített predikátumok, amelyeknek mellékhatásai vannak, ha végrehajtjuk őket. Ilyenek a bemeneti-kimeneti predikátumok és a tényeket a *TB*-hez hozzáadó, illetve onnan törlő kijelent/visszavon predikátumok. Ezeknek a predikátumoknak nincs megfelelőjük a logikában, és hatásukat sokszor nem egyszerű végigkövetni – például ha a tényeket egy olyan bizonyítási fa egyik ágán jelentjük ki, amely végül sikertelennek bizonyul.

- A Prolog lehetővé tesz olyan egyedi formájú tagadást, amelyet **sikertelenségi tagadás**nak (**negation as failure**) nevezünk. Egy negált cél, mint a a nem P, akkor tekintető bizonyítottnak, ha a rendszernek nem sikerül bebizonyítania a P-t. Így a következő mondat:

[5] Jegyezzük meg, hogy ha a Peano-axiómák meg vannak adva, akkor az ilyen célok megoldhatók egy Prolog-programon belüli következtetéssel.

Created by XMLmind XSL-FO Converter.

```
élő(X) :- nem halott(X)
```

úgy olvasható, hogy: „Mindenki élő, ha nem bizonyítható, hogy halott.”

- A Prolognak van egy egyenlőségi operátora (=), de ez nem rendelkezik a logikai egyenlőség teljes hatékonyságával. Egy egyenlőséget tartalmazó célállítás akkor sikeres, ha a két term *egyesíthető,* máskülönben sikertelen. Tehát az X + Y = 2 + 3 akkor sikeres, ha az X lekötött értéke a 2, és az Y lekötött értéke a 3, de a hajnalcsillag=esthajnalcsillag sikertelen. (A klasszikus logikában az utóbbi egyenlet lehet igaz, vagy lehet nem igaz.) Az egyenlőségről sem tényt, sem szabályt nem tudunk hozzáadni a *TB*-hez.

- A Prolog egyesítési algoritmusa nem tartalmazza az **előfordulási próbá**t. Ennek az a következménye, hogy lehetséges nem helyes következtetést végezni. Ez azonban ritkán okoz problémát, leginkább olyankor, amikor a Prologot matematikai tételek bizonyítására használjuk.

A Prolog tervezésénél meghozott döntések kompromisszumot jelentenek a deklarativitás és a végrehajtási hatékonyság között – mármint amit hatékonyságon értettünk a Prolog tervezésének idején. Visszatérünk majd ehhez a témához, miután megvizsgáltuk, hogyan valósítják meg a Prologot.

4.3. A logikai programok hatékony megvalósítása

Egy Prolog program végrehajtása kétféle módon történhet: lehet interpretált és lehet lefordított. Az interpretáció általában az ERL-HL-KÉRDEZ algoritmus (lásd 9.6. ábra) futtatását jelenti a programmal mint tudásbázissal. Azt mondjuk, hogy „általában”, mivel a Prolog különböző interpretációi különböző megoldásokat tartalmaznak a sebesség maximalizálására. Itt mi csak két implementációt nézünk meg.

Először is ahelyett, hogy minden lépésben összeállítanánk az összes lehetséges válasz listáját minden egyes részcélhoz, a Prolog interpreter egy választ hoz létre, és egy „ígéretet” ad meg arra, hogy a többi lehetséges választ is generálni fogja, miután az aktuális választ teljes mértékben feltárta. Ezt az ígéretet **választási pont**nak (**choice point**) nevezzük. Mikor a mélységi keresés befejezte az aktuális válaszból fakadó lehetséges megoldások feltárását, és visszalép a választási ponthoz, a választási pontot kiterjesztjük úgy, hogy megadjon egy új választ a részcélhoz, és egy új választási pontot. Ez a megközelítés időt és tárolóhelyet takarít meg. Ezenkívül nagyon egyszerű határfelületet biztosít a hibakereséshez, mert egy adott pillanatban csak egyetlen megoldási útvonal vizsgálata történik meg.

Másodszor, az ERL-HL-KÉRDEZ egyszerű alkalmazása a helyettesítések generálásakor és kompozíciójakor jól gazdálkodik az idővel. A Prolog a helyettesítéseket úgy alkalmazza, hogy olyan logikai változókat használ, amelyek emlékezni tudnak az aktuális lekötéseikre. A program futásának bármely pontján a programban lévő minden egyes változó vagy szabad, vagy valamely értékhez lekötött. Ezek a változók és értékek együtt definiálják implicit módon a helyettesítést a bizonyítás aktuális ágához. A vizsgált ágak kiterjesztése csak további változólekötéseket hoz létre, mivel egy új lekötés hozzáadása egy már lekötött változóhoz az egyesítés sikertelenségét eredményezi. Mikor a keresésben sikertelennek bizonyul egy ág, a Prolog visszalép egy megelőző választási ponthoz, és ekkor esetleg fel kell oldania néhány változót. Ezt úgy hajtja végre, hogy nyomon követi a veremben – amelynek **útvonal** (**trail**) a neve – lekötött összes változót. Mivel minden egyes új változó az EGYESÍT-VÁLT-tal kapcsolódik, a változót beteszi az útvonalba. Amikor egy cél sikertelen, és ideje visszatérni egy megelőző választási ponthoz, minden egyes változó szabaddá válik, minthogy kivesszük az útvonalból.

Még a leghatékonyabb Prolog interpretereknek is néhány ezer gépi utasításra van szükségük egy következtetési lépés végrehajtásához; az index kikeresésének, az egyesítésnek, a rekurzív hívási verem felépítésének a költsége miatt. Valójában az interpreter úgy viselkedik, mint aki még soha sem látta a programot. Például újra meg kell *találnia* a célhoz illeszkedő klózokat. Egy lefordított Prolog program ezzel szemben egy olyan következtetési eljárás, amelyet egy specifikus klózhalmazra készítettek, és így *tudja,* mely klózok illeszthetők az adott célhoz. A Prolog fordító alapjában véve minden egyes különböző predikátumra egy miniatűr tételbizonyítót generál, kiküszöbölve így az interpretációs lépések nagy részét. Lehetséges az is, hogy az egyesítési eljárást **nyitott kód**dal (**open-code**) lássuk el az egyes eljárás hívásokra, elkerülve így a termek struktúrájának explicit elemzését. A nyitott kódú egyesítés részleteit megtalálhatók a (Warren és társai, 1977)-ben.

A mai számítógépek utasításkészletei szegényesnek tűnnek összehasonlítva a Prolog szemantikájával, így tehát a Prolog fordítóit egy átmeneti nyelvre fordítják le ahelyett, hogy közvetlenül gépi nyelvre fordítanák. A legnépszerűbb átmeneti nyelv a Warren Absztrakt Gép (Warren Abstract Machine) vagy WAM, amelyet David H. D. Warrenről neveztek el, aki az első Prolog fordító egyik megvalósítója volt. A WAM a Prolog számára

Created by XMLmind XSL-FO Converter.

alkalmas absztrakt utasításkészlet, és interpretálható vagy lefordítható gépi nyelvre. Más fordítók, mint például a Lisp vagy a C, egy magas szintű nyelvre fordítják le a Prologot, és ezután használják a nyelvnek a fordítóját, hogy azt a gépi nyelvre fordítsa. Például, a Csatol definícióját a 9.8. ábrán bemutatott kódra lehet fordítani. Van még néhány kérdés, amit fontos megemlíteni:

- Ahelyett hogy keresgélni kellene a tudásbázisban a Csatol klózokat, a klózokat eljárássá lehet alakítani, és a következtetéseket egyszerűen úgy hajtjuk végre, hogy meghívjuk az eljárást.

- Mint azt korábban leírtuk, az aktuális változók lekötéseit egy útvonalhoz rendeljük. Az eljárás első lépése elmenti az útvonal pillanatnyi állapotát úgy, hogy azok visszaállíthatóak lesznek a `VISSZAÁLLÍT-ÚTVONAL`-lal, ha az első klóz sikertelen. Ez fel fog oldani minden lekötést, amelyet az első `EGYESÍT` hívás generált.

- A legtrükkösebb rész a **folytatás**ok (**continuation**s) listájának használata a választási pontok implementálására. Gondolhatunk úgy egy folytatásra, mint egy eljárás és egy argumentumlista becsomagolására, melyek együtt határozzák meg, hogy mit kell legközelebb csinálni, amikor az aktuális cél sikeres lesz. Nem elég csak úgy egyszerűen visszatérni egy olyan eljárásból, mint a `CSATOL`, amikor a cél sikeres, mivel itt sokféle módon lehet sikeres az eljárás, és minden egyes esetet fel kell deríteni. A folytatás argumentum megoldja ezt a problémát, mert minden egyes esetben lehívhatjuk, ha egy cél sikeres. A `CSATOL` kódban, ha az első argumentum üres, akkor a `CSATOL` predikátum sikeres. Ezután `HÍVJUK` a folytatást, az útvonalhoz rendelt lekötésekkel, hogy végrehajtsa, amit a folytatásban legközelebb meg kell tenni. Például ha a `CSATOL` hívása a legfelső szinten történik, a folytatás kinyomtatja a változók lekötéseit.

9.8. ábra - A Csatol predikátum fordításának pszeudokódos reprezentációja. Az `ÚJ-VÁLTOZÓ` függvény egy új változót ad vissza, amely különbözik a korábban visszaadott változóktól. A `HÍV` (*folytatás*) eljárás folytatja a végrehajtást a megadott folytatással.

```
procedure CSATOL(ax, y, az, folytatás)

  útvonal ← GLOBÁLIS-ÚTVONAL-MUTATÓ()
  if ax = [] és EGYESÍT(y, az) then HÍV(folytatás)
  VISSZAÁLLÍT-ÚTVONAL(útvonal)
  a ← ÚJ-VÁLTOZÓ(); x ← ÚJ-VÁLTOZÓ(); z ← ÚJ-VÁLTOZÓ()
  if EGYESÍT(ax, [a | x]) és EGYESÍT(az, [a | z]) then CSATOL(x, y, z, folytatás)
```

Mielőtt Warren a Prolog fordítót elkészítette volna, a logikai programozás túl lassú volt általános célú felhasználáshoz. A Warren és mások által létrehozott fordítók lehetővé tették, hogy a Prolog kód olyan sebességet érjen el, amelyek – különböző standard tesztek szerint is – versenyképessé tették a C nyelvvel (Van Roy, 1990). Természetesen az a tény, hogy egy tervkészítőt vagy egy elemzőt egy természetes nyelvhez néhány tucat Prolog sorban meg lehet írni, lényegesen vonzóbbá teszi a Prologot a C-nél a legtöbb kisebb MI-kutatási projekt prototípusának az elkészítésénél.

A párhuzamosítás szintén jelentős gyorsítást eredményezhet. Két elsődleges tere van a párhuzamosításnak. Az első, amelynek neve **VAGY-párhuzamosság** (**OR-paralellism**), abból a lehetőségből származik, hogy egy cél számos különböző klózzal lehet egyesíthető a tudásbázisban. Mindegyik egyesítés egy potenciális megoldáshoz vezető független ág megjelenését eredményezi a keresési helyen, és minden ilyen ágat párhuzamosan is megoldhatunk. A második, amelynek neve **ÉS-párhuzamosság** (**AND-paralellism**), abból származik, hogy az implikáció törzsében minden egyes konjunktot egy párhuzamos implikációval is meg lehet oldani. Az ÉS-párhuzamosságot nehezebb megvalósítani, mert a megoldás az egész konjunkcióra konzisztens lekötéseket igényel az összes változóra. Így minden egyes konjunktív ágnak kommunikálnia kell a többi ággal, hogy biztosítsák a globális megoldást.

4.4. Redundáns következtetés és végtelen hurkok

Most nézzük meg a Prolog nyelv Achilles-sarkát, a mélységi keresés és az ismételt állapotokat és végtelen útvonalakat tartalmazó fák illeszkedésének hiányát. Vizsgáljuk meg a következő logikai programot, amely eldönti, hogy létezik-e egy útvonal két pont között egy irányított gráfban:

```
útvonal(X, Z):- kapcsolat(X, Z)
```


Created by XMLmind XSL-FO Converter.

```
útvonal(X, Z):- útvonal(X, Y), kapcsolat(Y, Z)
```

Egy egyszerű, három csomópontos gráfot, amelyet a kapcsolat(a,b) és a kapcsolat(b,c) definiál, mutat be a 9.9. (a) ábra. Az útvonal(a,c) lekérdezésére a program a 9.10. (a) ábrán látható bizonyítási fát generálja. Másrészt viszont, ha a két klózt ebbe a sorrendbe tesszük:

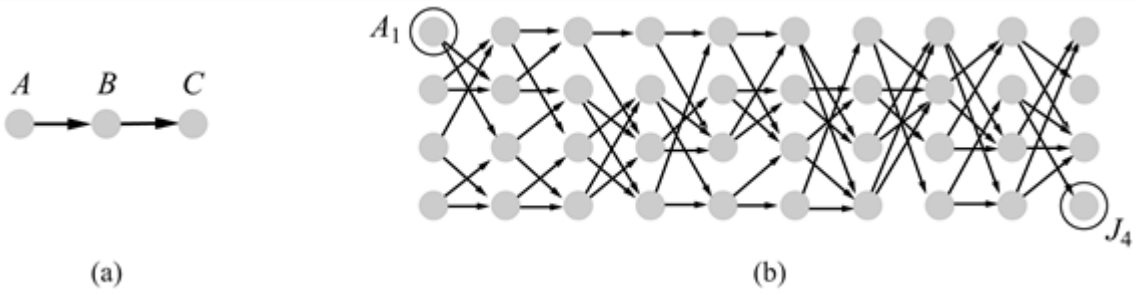
```
útvonal(X, Z):- útvonal(X, Y), kapcsolat(Y, Z)

útvonal(X, Z):- kapcsolat(X, Z)
```

akkor a Prolog a 9.10. (a) ábrán bemutatott végtelen keresést végzi. A Prolog tehát, mint egy tételbizonyító, határozott klózokból álló *TB*-re **nem teljes** (**incomplete**) (még a Datalog programok esetében sem teljes, mint azt a példánk mutatja), mivel léteznek olyan tudásbázisok, amelyekben nem tud bizonyítani vonzat mondatokat. Vegyük észre, hogy az előrefelé láncolással nincs ilyen probléma: ha egyszer az útvonal(a, b), az útvonal(b, c) és az útvonal(a, c) útvonalak ki lettek következtetve, az előrefelé láncolás leáll.

A mélységi keresésen alapuló hátrafelé láncolásnak is akad problémája a felesleges számításokkal. Például amikor meg akarunk találni egy útvonalat az A_1-ből a J_4-be a 9.9. (b) ábrán, a Prolog 877 következtetést hajt végre, amelyek legnagyobb részét a csomópontokhoz tartozó összes lehetséges útvonal megtalálása képezi, amelyekből pedig nem lehet elérni a célt. Ez hasonló a 3. fejezetben megtárgyalt ismételt állapot problémához. A következtetési lépések teljes száma a generált alaptények számának exponenciális függvénye lehet. Ha az előrefelé láncolást alkalmazzuk ehelyett, akkor legfeljebb n^2 útvonal(X, Y) tény generálható az n csomópont összekötésével. A 9.9. (b) ábrán látható problémához csak 62 következtetési lépés szükséges.

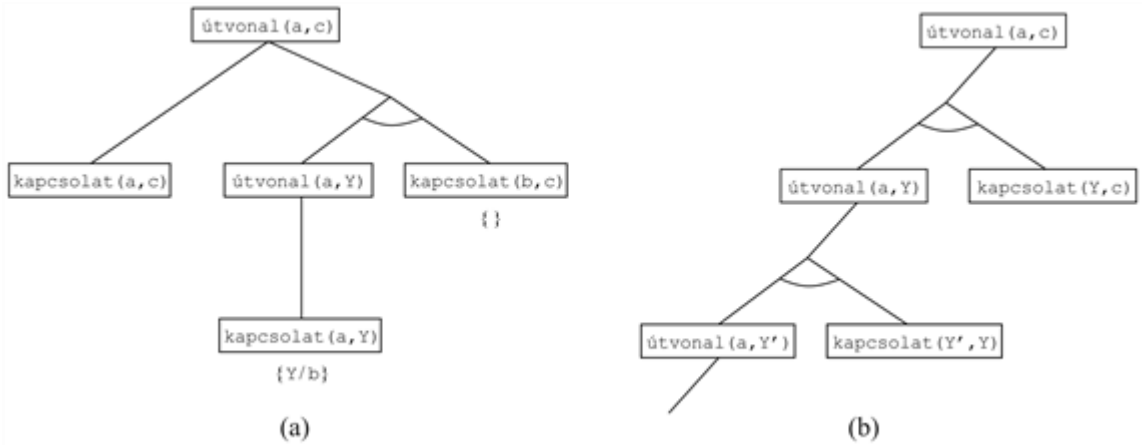
9.9. ábra - (a) Az *A*-ból a *C*-be vezető útvonal megtalálása a Prologot egy végtelen hurokba vezetheti. (b) Egy gráf, amelyben minden csomópontot két véletlenszerű következő rétegbeli csomóponthoz kapcsolunk a következő rétegben. Egy útvonal megtalálása az A_1-ből a J_4-be 877 következtetést igényel.



Az előrefelé láncolás a gráfkeresési problémáknál a **dinamikus programozás** (**dynamic programming**) egyik példája, amelyben a részproblémákra vonatkozó megoldások előállítása inkrementálisan történik a kisebb részproblémáktól indulva, végig tárolva a részmegoldásokat, hogy elkerüljük az újraszámolást. Ugyanezt a hatást érhetjük el egy hátrafelé láncoló rendszerben a **memók gyűjtésé**nek (**memoization**) használatával, ami azt jelenti, hogy a részcélok megoldásait tároljuk, amint megtaláltuk őket, hogy azután újra felhasználhassuk ezeket a megoldásokat, amikor a részcél visszatér, megtakarítva így az előző számítás megismétlését. Ezt a megközelítést használják a **táblázatos logikai programozás**i (**tabled logic programming**) rendszerek, amelyeknek hatékony tárolási és visszakeresési mechanizmusuk van a memók gyűjtésének megvalósítására. A táblázatos logikai programozás egyesíti a hátrafelé láncolás célirányosságát az előrefelé láncolás dinamikus programozási hatékonyságával. Ez az eljárás teljes a Datalog programokra is, ami azt eredményezi, hogy a programozónak kevesebbet kell aggódnia a végtelen hurkok problémája miatt.

9.10. ábra - (a) Az *A*-ból a *C*-be vezető útvonal bizonyítása. (b) Végtelen mélységű bizonyítási fa, amely akkor generálódik, ha a klózok „rossz” sorrendben vannak.

Created by XMLmind XSL-FO Converter.



4.5. Korlátozott logikai programozás

Az előrefelé láncolás tárgyalásánál (lásd 9.3. alfejezet) bemutattuk, hogy a kényszerkielégítési problémákat hogyan lehet határozott klózokként kódolni. A standard Prolog képes megoldani az ilyen problémákat, pontosan úgy, mint az 5.3. ábrán bemutatott visszalépéses algoritmus.

Mivel a visszalépéses módszer sorba veszi a változók tárgyterületeit, ezért csak **véges tárgyterületű** (**finite domain**) kényszerkielégítési problémákra működik. A Prolog termjeiben véges számú megoldásnak szabad léteznie bármely szabad változókkal rendelkező célra. (Például, a kül(q, sa) célnak, amely azt jelenti ki, hogy Queenslandnek és Dél-Ausztráliának különböző színűnek kell lenni, hat megoldása van, ha három szín megengedett.) A végtelen méretű tárgyterületekre alkalmazott módszerek – például az egész számokat felvevő vagy valós értékű változókat tartalmazó problémáknál alkalmazható módszerek – teljesen különböző algoritmust igényelnek, mint például a kényszerpropagálást vagy a lineáris programozást.

A következő klóz akkor tehető sikeressé, ha a három szám kielégíti a háromszög egyenlőtlenséget:

```
háromszög(X, Y, Z):-

X>=0, Y>=0, Z>=0, X+Y>=Z, Y+Z>=X, X+Z>=Y
```

Ha a Prolognak a háromszög (3, 4, 5) lekérdezést adjuk, akkor ez jól működik. Ha viszont azt kérdezzük: háromszög (3, 4, Z), akkor erre nem talál megoldást, mert a Z>=0 részcélt a Prolog nem tudja kezelni. A nehézséget az okozza, hogy a Prologban a változóknak két állapot valamelyikében kell lenniük: egy változó szabad vagy egy bizonyos termhez lekötött.

Egy változó lekötése egy termhez a korlátozás egyik speciális fajtájának tekinthető, nevezetesen egy egyenlőségkényszernek. A **korlátozott logikai programozás** (**constraint logic programming**, **CLP**) lehetővé teszi, hogy a változók *korlátozva* legyenek, ahelyett hogy *lekötöttek* lennének. Egy korlátozott logikai program megoldása a korlátoknak a tudásbázisból származtatható legspecifikusabb halmaza a lekérdezés változóin. Például a háromszög (3, 4, Z) lekérdezésre a megoldás a 7 >= Z >= 1 kényszer. A standard logikai programok csak a CLP speciális esetei, amelyekben a megoldás kényszereinek egyenlőségi kényszereknek – azaz lekötéseknek – kell lenniük.

A CLP-rendszerek különféle kényszermegoldó algoritmusokat is tartalmaznak a nyelvben megengedett kényszerek kezelésére. Például egy valós értékű változókon értelmezett, lineáris egyenlőtlenségeket megengedő rendszer tartalmazhat egy lineáris programozási algoritmust ezen kényszerek feloldására. A CLP-rendszerek egy sokkal rugalmasabb megközelítést is adoptálnak a standard logikai programozási lekérdezések megoldására. Például a mélységi, balról jobbra történő visszalépéses keresés helyett használhatják bármelyik ennél hatékonyabb, az 5. fejezetben tárgyalt algoritmust, beleértve a heurisztikus konjunktok sorba rendezését, a visszaugrást, a vágási halmaz kondicionálást és így tovább. A CLP-rendszerek tehát vegyesen alkalmazzák a kényszerekkielégítési algoritmusok, a logikai programozás és a deduktív adatbázisok elemeit.

A CLP-rendszerek szintén hasznosítják az 5. fejezetben bemutatott különböző kényszerprogramozási keresési optimalizációs módszereket, mint a változók és értékek sorba rendezését, az előrefelé ellenőrzést vagy az intelligens visszalépés módszerét. Számos rendszert úgy terveztek meg, hogy a programozónak nagyobb kontrollja legyen a következtetések keresésének a sorrendje felett. Például az MRS nyelv (Genesereth és Smith,

Created by XMLmind XSL-FO Converter.

1981; Russell, 1985) lehetővé teszi a programozó számára, hogy **metaszabály**okat (**metarule**s) írjon a konjunktok megvizsgálási sorrendjének meghatározására. A felhasználó például írhat egy olyan szabályt, amely azt mondja ki, hogy a legkevesebb változóval rendelkező cél legyen először kipróbálva, vagy írhat tárgyterület-specifikus szabályokat bizonyos predikátumokhoz.

5. Rezolúció

A logikai rendszerek általunk bemutatandó három családjának az utolsó tagja a **rezolúció**n (**resolution**) alapul. Láttuk a 7. fejezetben, hogy a propozíciós rezolúció egy cáfolásteljes következtetési folyamat az ítéletlogikában. Ebben az alfejezetben meg fogjuk vizsgálni, hogyan lehet kiterjeszteni a rezolúciót az elsőrendű logikára.

A teljes bizonyítási eljárás létezésének kérdése a matematikusokat érdekli közvetlenül. Ha egy teljes bizonyítási eljárás megtalálható matematikai állításokra, ebből két dolog következik. Először is, az összes tétel előfeltételeit mechanikusan elő tudjuk állítani; másodszor, a teljes matematika felépíthető alapaxiómák halmazának logikai következményeként. A teljesség kérdésének vizsgálata így a 20. századi matematika néhány legfontosabb eredményének a megszületéséhez vezetett. 1930-ban Kurt Gödel német matematikus bebizonyította az első **teljességi tétel**t (**completeness theorem**) az elsőrendű logikára, megmutatva, hogy minden kikövetkeztetett mondatnak létezik véges bizonyítása. (Gyakorlatban is felhasználható bizonyítási módszert viszont nem találtak egészen addig, amíg J. A. Robinson nem publikálta a rezolúciós algoritmust 1965-ben.) 1931-ben Gödel bebizonyította a még híresebb **nemteljességi tétel**t (**incompleteness theorem**). A tétel kimondja, hogy egy logikai rendszer, amely tartalmazza az indukció elvét – amely nélkül a diszkrét matematika igen kis része építhető fel – szükségszerűen nem teljes. Ebből az következik, hogy léteznek olyan kikövetkeztetett mondatok, amelyeknek a rendszeren belül nincs véges bizonyítása. Lehet, hogy a tű ott van a metaforikus szénakazalban, de nincs olyan eljárás, amely garantálná, hogy megtaláljuk.

Gödel tétele ellenére, a rezolúcióalapú tételbizonyításokat széles körben alkalmazták matematikai tételek levezetésére, beleértve néhány olyan tételt is, amelyre előzőleg nem volt ismert bizonyítás. A tételbizonyításokat – más alkalmazások mellett – használták például hardvertervezés verifikálására vagy logikailag helyes programok generálására.

5.1. Az elsőrendű logika konjunktív normál formája

Mint az ítéletlogika esetében, az elsőrendű rezolúció is megköveteli, hogy a mondatok **konjunktív normál formá**ban (**conjunctive normal form**) (**CNF**) legyenek, tehát minden mondat klózok konjunkciója, ahol minden egyes klózt literálok diszjunkciója alkot.[6] A literálok tartalmazhatnak változókat, amelyeket univerzális kvantorral ellátottnak feltételezünk. Például a

$$\forall x\ Amerikai(x) \land Fegyver(y) \land Elad(x, y, z) \land Ellenséges(z) \Rightarrow Bűnöző(x)$$

mondat CNF-formára átalakítva ilyen lesz:

$$\neg Amerikai(x) \lor \neg Fegyver(y) \lor \neg Elad(x, y, z) \lor \neg Ellenséges(z) \lor Bűnöző(x)$$

Fontos

Minden elsőrendű logikai mondat átalakítható a következtetés szempontjából egyenértékű CNF-mondattá. Ami azt jelenti, hogy egy CNF-mondat akkor kielégíthetetlen, ha az eredeti mondat is kielégíthetetlen, ezáltal a CNF-mondatok között ellentmondásokat keresve létre tudunk hozni bizonyításokat.

A CNF-mondatok átalakításának eljárása nagyon hasonló az ítéletlogikai eljáráshoz, amelyet a 4.1.1. szakasz - Konjunktív normál formarészben láthatunk. A legfontosabb különbség abból fakad, hogy ki kell vonnunk a mondatokból az egzisztenciális kvantorokat. Ezt az eljárást a következő mondat lefordításával illusztráljuk: „Mindenkit, aki az összes állatot szereti, valaki szeret”, vagyis:

$$\forall x\ [\forall y\ Állat(y) \Rightarrow Szereti(x, y)] \Rightarrow [\exists y\ Szereti(y, x)]$$

[6] Egy klózt implikációként is reprezentálhatunk a bal oldali atomok konjunkciójával és a jobb oldali atomok diszjunkciójával, mint azt a 7.12. feladatban láthatjuk. Ezt a formát, amelyet **Kowalski-formá**nak (**Kowalski form**) szoktak nevezni, gyakran sokkal könnyebb kiolvasni, ha az implikációk írása jobbról balra történik (Kowalski, 1979b).

Created by XMLmind XSL-FO Converter.

A lépések a következők:

- Az implikációk eliminálása.

$\forall x\ [\neg\forall y\ \neg Állat(y) \vee Szereti(x, y)] \vee [\exists y\ Szereti(y, x)]$

- **A ¬befelé mozgatása.** A negált összekötőjelekre vonatkozó már ismert szabályok mellett szükségünk van a negált kvantorokra vonatkozó szabályokra is. Így ezt kapjuk:

$\neg\forall x\ p$-ből lesz $\exists x\ \neg p$

$\neg\exists x\ p$-ből lesz $\forall x\ \neg p$

A mondatunk a következő átalakításokon megy keresztül:

$\forall x\ [\exists y\ \neg(\neg Állat(y) \vee Szereti(x, y))] \vee [\exists y\ Szereti(y, x)]$

$\forall x\ [\exists y\ \neg\ \neg Állat(y) \wedge \neg Szereti(x, y)] \vee [\exists y\ Szereti(y, x)]$

$\forall x\ [\exists y\ \ Állat(y) \wedge \neg Szereti(x, y)] \vee [\exists y\ Szereti(y, x)]$

Figyeljük meg, hogyan alakítottuk át az univerzális kvantort ($\forall y$) az implikáció premisszájában egzisztenciális kvantorrá. A mondatot most így lehet kiolvasni „Vagy van olyan állat, amelyet x nem szeret, vagy (ha ez nem áll fenn) valaki szereti x-et.” Világos, hogy az eredeti mondat jelentését megőriztük.

- **A változók átnevezése.** Az olyan mondatokban, mint a $(\forall x\ P(x)) \vee (\exists x\ Q(x))$, amelyek ugyanazt a változó nevet kétszer is használják, változtassuk meg az egyik változó nevét. Ezzel elkerüljük, hogy később keveredés lehessen, amikor elhagyjuk a kvantorokat. Így ezt kapjuk:

$\forall x\ [\exists y\ Állat(y) \wedge \neg Szereti(x, y)] \vee [\exists z\ Szereti(z, x)]$

- **Skolemizáció.** A **skolemizáció** (**skolemization**) az az eljárás, amelynek során eliminációval eltávolítjuk az egzisztenciális kvantorokat. Egyszerű esetben ez megegyezik a 9.1. alfejezetben található egzisztenciális példányosítási szabállyal: fordítsuk le a $\exists x\ P(x)$-et $P(A)$-ra, ahol az A egy új konstans. Ha ezt a szabályt alkalmazzuk a mintamondatunkra, akkor azonban ezt kapjuk:

$\forall x\ [Állat(A) \wedge \neg Szereti(x, A)] \vee Szereti(B, x)$

amelynek teljesen téves a jelentése: azt mondja ki, hogy mindenki vagy nem tud szeretni egy bizonyos A állatot, vagy egy bizonyos B entitás szereti őt. Valójában, az eredeti mondatunk lehetővé teszi minden egyes személy számára, hogy ne szeressen egy másik állatot, vagy egy másik személy szeresse őt. Így tehát, azt szeretnénk, hogy a Skolem entitások az x-től függjenek:

$\forall x\ [Állat(F(x)) \wedge \neg Szereti(x, F(x))] \vee Szereti(G(x), x)$

Itt az F és a G **Skolem-függvény**ek**.** Az általános szabály az, hogy a Skolem-függvények argumentumai mind univerzális kvantorral ellátott változók, amelyeknek a hatókörében az egzisztenciális kvantor értelmezett. Hasonlóan az Egzisztenciális Példányosításhoz, a skolemizált mondat is pontosan akkor kielégíthető, ha az eredeti mondat is kielégíthető.

- **Az univerzális kvantorok elhagyása.** Ezen a ponton minden megmaradt változó univerzális kvantorral van ellátva. Mi több, a mondat egyenértékű azzal a mondattal, amelyben az összes univerzális kvantort balra rendezték. Így tehát most már elhagyhatjuk az univerzális kvantorokat:

$[Állat(F(x)) \wedge \neg Szereti(x, F(x))] \vee Szereti(G(x), x)$

- **Az ∧ elosztása a ∨ felett.**

$[Állat(F(x)) \vee Szereti(G(x), x)] \wedge [\neg Szereti(x, F(x)) \vee Szereti(G(x), x)]$

Ennek a lépésnek az elvégzéséhez szükség lehet az egymásba ágyazott konjunkciók és diszjunkciók kibontására is.

A mondat most CNF formájú, és két klózból áll. Eléggé olvashatatlan lett. (Némiképpen segíthet, ha megmagyarázzuk, hogy az $F(x)$ függvény egy, az x személy által potenciálisan nem szeretett állatra vonatkozik,

Created by XMLmind XSL-FO Converter.

míg a $G(x)$ olyan valakire utal, aki szeretheti x-et.) Szerencsére, nekünk ritkán kell CNF-mondatokat vizsgálnunk – a fordítási folyamat könnyen automatizálható.

5.2. A rezolúciós következtetési szabály

Az elsőrendű klózokra vonatkozó következtetési szabály egyszerűen a 4.1. szakasz - Rezolúció részben megadott propozíciós rezolúciós szabály kiterjesztett változata. Két klóz, amelyek standardizálva vannak, tehát nem tartalmaznak azonos változókat, akkor rezolválható, ha tartalmaznak komplememens literálokat. Az ítéletlogikai literálok akkor komplemensek, ha az egyik a negációja a másiknak; míg az elsőrendű literálok akkor komplemensek, ha az egyik *egyesül* a másik negációjával. Így ezt kapjuk:

$$\frac{\ell_1 \vee \ldots \vee \ell_k, m_1 \vee \ldots \vee m_n}{\text{HELYETTESÍT}(\theta, \ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}$$

ahol az EGYESÍT $(l_i, \neg m_j) = \theta$. Például rezolválhatjuk ezt a két klózt:

$[Állat(F(x)) \vee Szereti(G(x), (x)]$ és $[\neg Szereti(u, v) \vee \neg Megöli(u, v)]$

a $\theta = \{u/G(x), v/x\}$ egyesítő felhasználásával a komplemens literálokat, a $Szereti(G(x), x)$-t és a $\neg Szereti(u, v)$-t eliminálva kapjuk meg a **rezolvens** (**resolvent**) klózt:

$[Állat(F(x)) \vee \neg Megöli(G(x), x)]$

A bemutatott szabály neve **bináris rezolúció**s (**binary resolution**) szabály, mert pontosan két literált old fel. A bináris rezolúciós szabály önmagában nem eredményez egy teljes következtetési folyamatot. A teljes rezolúciós szabály literálok részhalmazait rezolválja minden egyes egyesíthető klózban. Egy alternatív megközelítés a **faktorálás**nak (**factoring**), a felesleges literálok eltávolításának a kiterjesztése az elsőrendű logikára. A propozíciós faktorálás két literált eggyé redukál, ha azok *azonosak.* Az elsőrendű faktorálás két literált eggyé redukál, ha azok *egyesíthetők.* Az egyesítőt a teljes klózra kell alkalmazni. A bináris rezolúció és a faktorálás kombinációja már teljes eljárást eredményez.

5.3. Példabizonyítások

A rezolúció a $TB \models \alpha$ állítást úgy igazolja, hogy bebizonyítja, hogy a $TB \wedge \neg\alpha$ nem kielégíthető, vagyis a bizonyítás az üres klóz származtatásával történik. Az algoritmikus megközelítés megegyezik az ítéletlogikában lévővel, amelyet a 7.12. ábrán már bemutattunk, így ezt itt most nem ismételjük meg. Ehelyett inkább két példabizonyítást adunk meg. Az első a bűntény példa a 9.3. alfejezetből. A mondatok CNF-ben a következők:

$\neg Amerikai(x) \vee \neg Fegyver(y) \vee \neg Elad(x, z, y) \vee \neg Ellenséges(z) \vee Bűnöző(x)$

$\neg Rakéta(x) \vee \neg Birtokol(Nono, x) \vee Elad(West, x, Nono)$

$\neg Ellenség(x, Amerika) \vee Ellenséges(x)$

$\neg Rakéta(x) \vee Fegyver(x)$

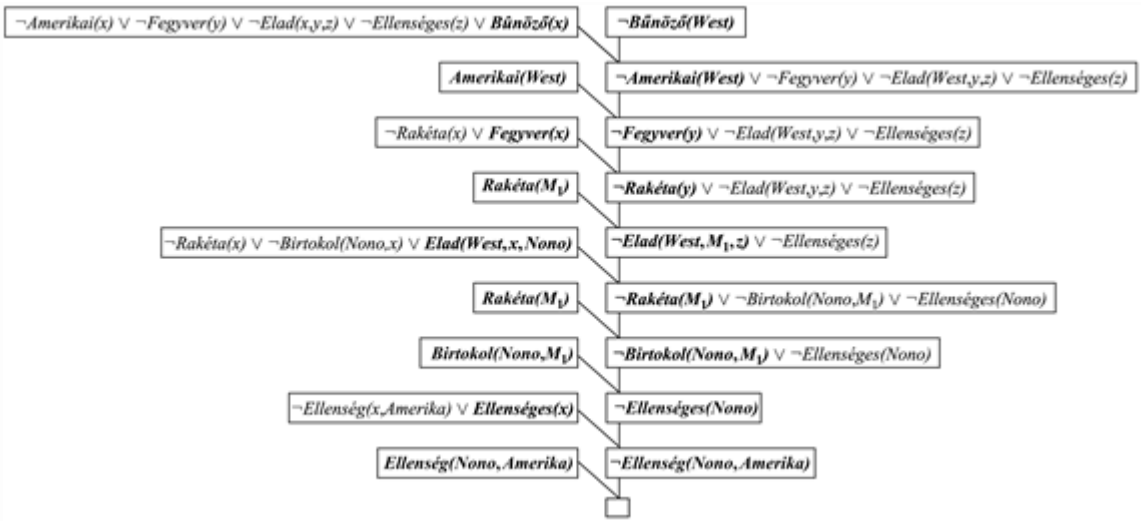
$Birtokol(Nono, M_1)$ $Rakéta(M_1)$

$Amerikai(West)$ $Ellenség(Nono, Amerika)$

Hozzáadjuk a mondatok halmazához a negált célt is $\neg Bűnöző(West)$. A rezolúciós bizonyítást a 9.11. ábra mutatja be. Figyeljük meg a szerkezetét: a bizonyításnak egy egyszerű „gerince” van, amely a célklózzal kezdődik, és folyamatosan rezolvál a tudásbázisból származó klózokkal, mindaddig, amíg az üres klózt nem generálja. Ez jellemző a Horn-klózokat tartalmazó tudásbázisokon végzett rezolúcióra. Valójában a fő gerinc mentén található klózok *pontosan* megfelelnek a 9.6. ábrán látható hátrafelé láncolási algoritmusban *a célváltozók* egymást követő értékeinek. Ez azért van így, mert mindig azt a klózt választjuk ki rezolválásra, amelynek a pozitív literálja egyesíthető a gerincen lévő „aktuális” klóz bal szélen elhelyezkedő literáljával, és pontosan ez történik a hátrafelé láncolásban is. Így tehát a hátrafelé láncolás valójában a rezolúció egy speciális esete, egy különleges vezérlési stratégiával, amely meghatározza, hogy melyik rezolúciót hajtsuk végre legközelebb.

9.11. ábra - Egy rezolúciós bizonyítás arra, hogy West bűnöző

Created by XMLmind XSL-FO Converter.



A második példánk kihasználja a skolemizációt, és nem határozott klózokat is tartalmaz. Ez valamivel bonyolultabb bizonyítási struktúrát eredményez. Természetes nyelven megfogalmazva a probléma a következő:

Mindenkit, aki az összes állatot szereti, szeret valaki.

Bárkit, aki megöl egy állatot, senki sem szeret.

Jankó szereti az összes állatot.

Vagy Jankó, vagy a Kíváncsiság ölte meg a macskát, akinek a neve Tuna.

A Kíváncsiság ölte meg a macskát?

Először is leírjuk az eredeti mondatokat, némi háttértudást, és a G negált célt az elsőrendű logikában:

A. $\forall x\ [\forall y\ Állat(y) \Rightarrow Szereti(x, y)] \Rightarrow [\exists y\ Szereti(y, x)]$

B. $\forall x\ [\exists y\ Állat(y) \wedge Megöli(x, y)] \Rightarrow [\forall z\ \neg Szereti(z, x)]$

C. $\forall x\ Állat(x) \Rightarrow Szereti(Jankó, x)$

D. $Megöli(Jankó, Tuna) \vee Megöli(Kíváncsiság, Tuna)$

E. $Macska(Tuna)$

F. $\forall x\ Macska(x) \Rightarrow Állat(x)$

G. $\neg Megöli(Kíváncsiság, Tuna)$

Most pedig alkalmazzuk az átalakítási procedúrát, hogy minden egyes mondatot CNF-be konvertáljunk:

A1. $Állat(F(x)) \vee Szereti(G(x), x)$

A2. $\neg Szereti(x, F(x)) \vee Szereti(G(x), x)$

B. $\neg Állat(y) \vee \neg Megöli(x, y) \vee \neg Szereti(z, x)$

C. $\neg Állat(x) \vee Szereti(Jankó, x)$

D. $Megöli(Jankó, Tuna) \vee Megöli(Kíváncsiság, Tuna)$

E. $Macska(Tuna)$

F. $\neg Macska(x) \vee Állat(x)$

¬G. $\neg Megöli(Kíváncsiság, Tuna)$

Created by XMLmind XSL-FO Converter.

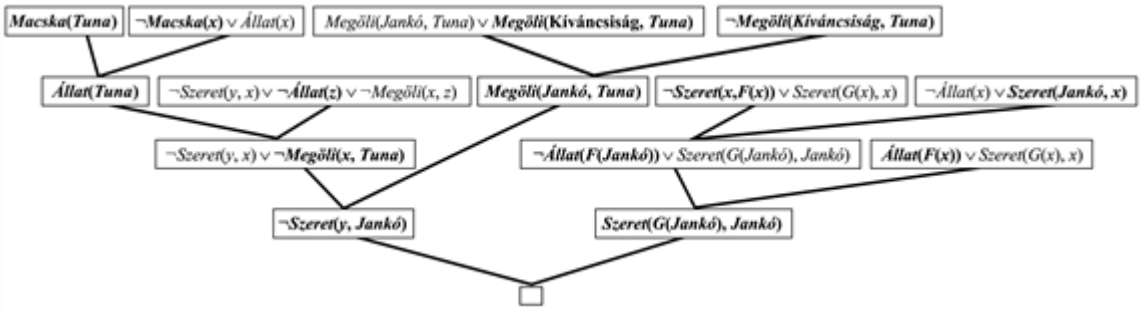
A 9.12. ábrán látható a rezolúciós bizonyítása annak, hogy a Kíváncsiság ölte meg a macskát. Magyarul a bizonyítást így írhatjuk át:

Tegyük fel, hogy a Kíváncsiság nem ölte meg Tunát. Tudjuk, hogy vagy Jankó, vagy a Kíváncsiság tette; tehát biztosan Jankó tehette. Mármost, Tuna egy macska, és a macskák állatok, tehát Tuna egy állat. Mivel bárkit, aki megöl egy állatot, senki sem szeret, tudjuk, hogy Jankót nem szereti senki. Másrészt, Jankó minden állatot szeret, tehát valaki szereti őt, tehát ez az ellentmondás áll fenn. Így tehát a Kíváncsiság ölte meg a macskát.

A bizonyítás képes megválaszolni a „A Kíváncsiság ölte meg a macskát?” kérdést, de gyakran ennél általánosabb kérdéseket akarunk feltenni, mint például: „Ki ölte meg a macskát?” A rezolúció meg tudja válaszolni ezt is, de ennek a válasznak a levezetése egy kicsit több munkát igényel. A célállítás a $\exists w$ *Megöli*(*w*, *Tuna*), amely negáláskor CNF-ben ¬*Megöli*(*w*, *Tuna*) lesz. Megismételve a bizonyítást a 9.12. ábrán az új negált céllal, hasonló bizonyítási fát kapunk, de a következő helyettesítéssel: {*w*/*Kíváncsiság*} az egyik lépésben. Így ebben az esetben, annak a kiderítése, hogy ki ölte meg a macskát, nem áll másból, mint a lekérdezés változóiban lévő lekötések nyomon követése a bizonyításban.

Sajnos, a rezolúció csak **nem konstruktív bizonyítás**t (**nonconstructive proof**s) tud előállítani az egzisztenciális célmondatokra. Például a ¬*Megöli*(*w*, *Tuna*) rezolválható a *Megöli*(*Jankó*, *Tuna*) ∨ *Megöli*(*Kíváncsiság*, *Tuna*)-val, hogy a *Megöli*(*Jankó, Tuna*) mondatot megkapjuk, amely megint rezolválható ezzel: ¬*Megöli*(*w*, *Tuna*), és így kapjuk meg az üres klózt. Figyeljük meg, hogy ebben a bizonyításban a *w*-nek két különböző lekötése van. A rezolúció megmutatja azt, hogy igen, valaki megölte Tunát – vagy Jankó, vagy a Kíváncsiság. Ez nem túl meglepő! Egy lehetséges megoldás korlátozni a rezolúciós lépéseket úgy, hogy a kérdés változóinak csak egy lekötésük lehessen egy adott bizonyításban, majd képesnek kell lennünk arra, hogy visszakövessük a lehetséges lekötéseket. Egy másik megoldás egy speciális **válasz literál** (**answer literal**) hozzáadása a negált célhoz, amelyből ez lesz: ¬*Megöli*(*w, Tuna*) ∨ *Válasz*(*w*). Ilyenkor a rezolúciós folyamat mindig generál egy pontosan egy literált tartalmazó választ, amikor egy ilyan klóz generálódik. A 9.12. ábrán látható bizonyításra ez a *Válasz*(*Kíváncsiság*).A nem konstruktív bizonyítás ezt a klózt generálná: *Válasz*(*Kíváncsiság*) ∨ *Válasz*(*Jankó*), amely nem ad számunkra választ.

9.12. ábra - Egy rezolúciós bizonyítása annak, hogy a Kíváncsiság ölte meg a macskát. Figyeljük meg a faktorálás használatát a *Szereti*(*G*(*Jankó*), *Jankó*) klóz származtatásánál.



5.4. A rezolúció teljessége

Ebben a részben bebizonyítjuk a rezolúció teljességét. A fejezet elolvasását nyugodtan kihagyhatja bárki, aki megelégszik azzal, hogy elhiszi ezt az állítást.

Be fogjuk mutatni, hogy a rezolúció **cáfolásteljes** (**refutation-complete**), ami azt jelenti, hogy *ha* egy mondathalmaz kielégíthetetlen, akkor a rezolúció mindig képes levezetni egy ellentmondást. A rezolúció nem alkalmazható arra, hogy mondatok egy halmazának összes logikai következményét generálja, de használható arra, hogy megmondjuk, hogy egy adott mondat következménye-e a mondathalmaznak. Így alkalmazható egy adott kérdésre adható összes válasz megtalálására, felhasználva a korábban a fejezetben már bemutatott célnegálás módszert.

Fontos

Created by XMLmind XSL-FO Converter.

Tényként tekintjük, hogy minden (egyenlőséget nem tartalmazó) elsőrendű logikai mondat átírható CNF-klózok halmazára. Ez a mondat alakján végzett indukciós bizonyítással igazolható, az atomi mondatból, mint alapesetből kiindulva (Davis és Putnam, 1960). Célunk tehát bizonyítani a következőt: *ha S egy kielégíthetetlen klózhalmaz, akkor az S-en elvégzett véges számú rezolúciós lépés alkalmazása ellentmondáshoz vezet.*

Bizonyításunk váza követi Robinson eredeti bizonyítását, néhány Geneserethtől és Nilssontól származó egyszerűsítés hozzáadásával (Genesereth és Nilsson, 1987). A 9.13. ábra mutatja be a bizonyítás alapvető struktúráját, ami a következő:

1. Először megvizsgáljuk, hogy ha *S* kielégíthetetlen, akkor létezik *S* klózai között bizonyos *alappéldányoknak* egy olyan halmaza, ami szintén kielégíthetetlen. (Herbrand-tétel).

2. Ekkor az **alap rezolúciós tétel**hez (**ground resolution theorem**) fordulunk, amelyet a 7. fejezetben adtunk meg, és amely kijelenti, hogy a propozíciós rezolúció teljes az alapmondatokra.

3. Ezután alkalmazzuk a **kiterjesztéslemmá**t (**lifting lemma**) annak megmutatására, hogy bármely propozíciós rezolúciós bizonyításhoz, amely alapmondatok halmazát használja, található olyan elsőrendű rezolúciós bizonyítás, amely azokat az elsőrendű logikai mondatokat használja, amelyekből az alapmondatokat megkaptuk.

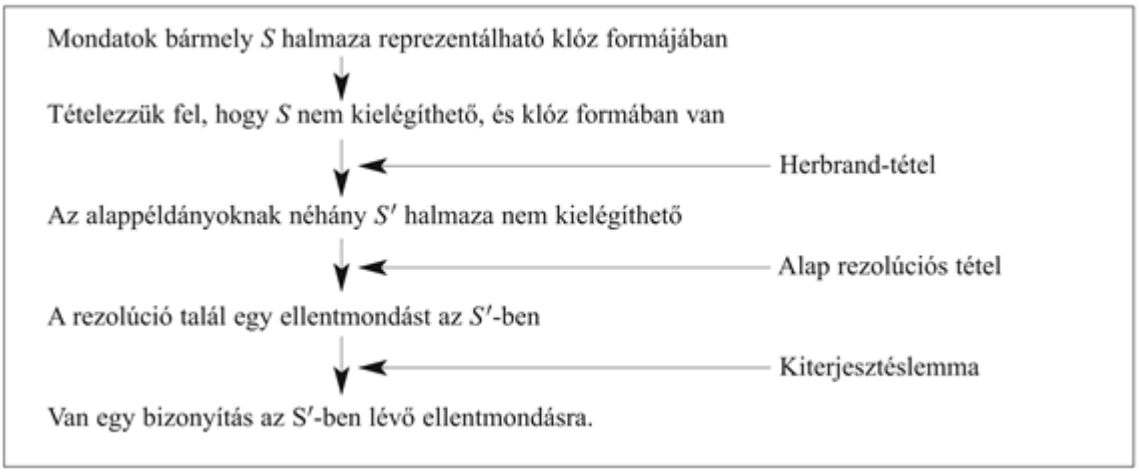
Az első lépés elvégzéséhez szükségünk lesz három új fogalomra:

- **Herbrand-univerzum**: Ha *S* a klózok egy halmaza, akkor a H_S a Herbrand-univerzuma *S*-nek, vagyis az összes alaptermből álló halmaz, amelyet létrehozhatunk a következőkből:

  a. Az *S* függvény szimbólumai, ha vannak ilyenek.

  b. Az *S* konstansszimbólumai, ha vannak ilyenek. Ha nincs ilyen, akkor az *A* konstans szimbólum.

9.13. ábra - A rezolúcióteljesség bizonyításának a szerkezete



Például, ha az *S* csak a $\neg P(x, F(x, A)) \wedge \neg Q(x, A) \vee R(x, B)$ klózt tartalmazza, akkor a H_S az alapmondatok következő végtelen halmaza:

```
{A, B, F(A, A), F(A, B), F(B, A), F(B, B), F(A, F(A, A)), …}
```

- **Telítődés**: Ha *S* a klózok egy halmaza, és *P* az alaptermek egy halmaza, és *P*(*S*) az *S* telített halmaza *P*-re vonatkoztatva, akkor *P*(*S*) az összes alapklóz halmaza, amelyet úgy kapunk meg, hogy alkalmazzuk az összes lehetséges konzisztens helyettesítését a *P*-beli alaptermeknek az *S*-beli változókkal.

- **Herbrand-bázis**: Egy *S* klózhalmaznak a hozzá tartozó Herbrand-univerzumra vonatkozó telítődését *S* Herbrand-bázisának nevezzük, és $H_S(S)$-sel jelöljük. Például, ha *S* csak a korábban példaként használt klózt tartalmazza, akkor a $H_S(S)$ a következő végtelen klózhalmaz :

```
{¬P(A, F(A, A)) ∨ ¬Q(A, A) ∨ R(A, B),
```


Created by XMLmind XSL-FO Converter.

$\neg P(B, F(B, A)) \vee \neg Q(B, A) \vee R(B, B)$,

$\neg P(F(A, A), F(F(A, A), A)) \vee \neg Q(F(A, A), A) \vee R(F(A, A), B)$

$\neg P(F(A, B), F(F(A, B), A)) \vee \neg Q(F(A, B), A) \vee R(F(A, B), B), \ldots\}$

Ezek a definíciók lehetővé teszik, hogy a **Herbrand-tétel** (**Herbrand's theorem**) egyik formáját állítsuk:

> Ha a klózok egy *S* halmaza nem kielégíthető, akkor létezik egy véges $H_S(S)$ részhalmaz, amely szintén nem kielégíthető.
>
> —(Herbrand, 1930)

Legyen *S′* ez a véges részhalmaza az alapmondatoknak. Mármost, alkalmazhatjuk az alap rezolúciós tételt (4.1.3. szakasz - A rezolúció teljessége részben), hogy kimutassuk, hogy a **rezolúciós! lezárt** (**resolution! closure**) *RC*(*S′*) tartalmazza az üres klózt. Ez azt jelenti, hogy a propozíciós rezolúció futtatása a teljességig az *S′*-en egy ellentmondást fog eredményezni.

Fontos

Gödel nemteljesség tétele

A **matematikai indukciós sémá**kkal (**mathematical induction schema**) kibővítve az elsőrendű logikát, Gödel **nemteljesség tétel**e (**incompleteness theorem**) megmutatta, hogy léteznek olyan igaz aritmetikai mondatok, amelyek nem bizonyíthatók.

A nemteljesség elmélet igazolása mintegy 30 oldalt kívánna, így túlmutat ennek a könyvnek a keretein, de a bizonyítás lényegét felvázoljuk a következőkben. Először a számok logikai elméletét vezetjük be. Ebben az elméletben egyetlen konstans létezik, a 0, és egyetlen függvény, az *S* (az utód-függvény). Az így létrehozott modellben *S*(0) fejezi ki az 1-et, *S*(*S*(0)) jelöli a 2-t és így tovább. A nyelv tehát alkalmas az összes természetes szám kifejezésére. A nyelv szótára tartalmazza ezenkívül a +, a × és az *exp* (exponenciális) függvényszimbólumokat és a logikai összekötőjelek és a kvantorok szokásos halmazát. Először figyeljük meg, hogy a nyelv mondatai megszámozhatók. (Definiáljunk alfabetikus sorrendet a szimbólumok között, és rendezzük alfabetikus sorrendbe a nyelv 1, 2, …, *n*, … hosszúságú mondatait.) Így minden α mondathoz hozzárendelhetünk egy egyedi $\#\alpha$ természetes számot, a **Gödel-szám**ot (**Gödel number**). Fontos tehát, hogy a számelmélet minden saját mondatához tartalmaz egy nevet. Hasonlóan megszámozhatunk minden egyes *P* bizonyítást egy *G*(*P*) Gödel-számmal, mivel egy bizonyítás nem más, mint mondatok véges sorozata.

Most tegyük fel, hogy adott egy *A* halmaz, amely a természetes számokról igaz állításokat megfogalmazó mondatokat tartalmaz. Mivel az *A* halmaz megadható egész számoknak egy adott halmazával (az *A* halmazban található mondatok Gödel-számaival), ezért létre tudunk hozni a definiált nyelvben egy $\alpha(j, A)$ mondatot, amely a következő állítás:

$\forall i$ *i* nem a Gödel-száma a *j* Gödel-számú mondat olyan bizonyításának, amely bizonyítás csak *A*-beli premisszákat használ.

Legyen a σ az $\alpha(\#\sigma, A)$ mondat, vagyis egy olyan mondat, amely kifejezi önmagának a bizonyíthatatlanságát *A*-ból. (Igazolható, hogy ilyen mondat mindig létezik, de nem egyszerű ezt belátni.)

Kövessük végig a következő ügyes érvelést. Tegyük fel, hogy σ bizonyítható *A*-ból, de akkor σ hamis, mivel σ épp azt mondja ki, hogy nem igazolható. Ebben az esetben viszont létezik egy hamis mondat, ami bizonyítható *A*-ból, így *A* nem tartalmazhatna csak igaz mondatokat – ez az előzetes feltételezésünkkel ellentétes. Tehát σ nem bizonyítható *A*-ból. Ez viszont éppen az, amit σ állít saját magáról, tehát σ egy igaz mondat.

Tehát megmutattuk (29 és fél oldalnyi levezetést átugorva), hogy bármely, a számelméletben definiálható igaz mondatokat tartalmazó halmaz esetében, bizonyos alapaxiómák feltételezése mellett, léteznek olyan igaz mondatok, amelyek *nem* bizonyíthatók az axiómákból. Ez az eredmény, más következmények mellett azt jelenti, hogy a matematika *bármely axiómarendszerében* megfogalmazható olyan tétel, amely nem bizonyítható az adott rendszer axiómáiból. Ez a

Created by XMLmind XSL-FO Converter.

matematikának egy igen fontos eredménye. A Gödel-tétel hatását a mesterséges intelligencia területére sokan vitatták, köztük maga Gödel is. A 26. fejezetben visszatérünk erre a kérdésre.

Most, hogy megállapítottuk, hogy mindig van egy rezolúciós bizonyítás, amely magában foglal néhány véges részhalmazt a Herbrand S bázisból, a következő lépés annak megmutatása, hogy létezik egy rezolúciós bizonyítás, amely magának az S-nek a klózait tartalmazza, amelyek nem szükségszerűen alapklózok. Azzal kezdjük, hogy megvizsgáljuk a rezolúciós szabály egy egyszerű alkalmazását. Robinson alaplemmája magába foglalja a következő tényt:

Legyen C_1 és C_2 két klóz, amelyeknek nincsenek közös változói. Legyenek C_1' és C_2' az alappéldányai C_1-nek és C_2-nek. Ha C' rezolvense C_1'-nek és C_2'-nek, akkor létezik egy olyan C klóz, amelyik (1) rezolvense C_1-nek és C_2-nek és (2) C' alappéldánya C-nek.

Ezt **kiterjesztéslemmá**nak (**lifting lemma**) nevezik, mivel kiterjeszti a bizonyítási lépést az alap klózokról az általános elsőrendű klózokra. A kiterjesztési lemma bizonyításához Robinsonnak szüksége volt az egyesítés módszerének megalkotására és a legáltalánosabb egyesítő tulajdonságainak levezetésére is. A bizonyítás áttekintése helyett a következőkben illusztráljuk a lemmát:

$C_1 = \neg P(x, F(x, A)) \lor \neg Q(x, A) \lor R(x, B)$

$C_2 = \neg N(G(y), z)) \lor P(H(y), z)$

$C'_1 = \neg P(H(B), F(H(B), A)) \lor \neg Q(H(B), A) \lor R(H(B), B)$

$C'_2 = \neg N(G(B), F(H(B), A)) \lor P(H(B), F(H(B), A))$

$C' = \neg N(G(B), F(H(B), A)) \lor \neg Q(H(B), A) \lor R(H(B), B)$

$C = \neg N(G(y), F(H(y), A)) \lor \neg Q(H(y), A) \lor R(H(y), B)$

Látható, hogy C' valóban C alaptermje. Általánosságban ahhoz, hogy C_1'-nek és C_2'-nek legyen rezolvense, úgy kell létrehozni őket, hogy először a C_1-beli és a C_2-beli komplemens literálok legáltalánosabb egyesítőjét alkalmazzuk C_1-re és C_2-re. A kiterjesztési lemma felhasználásával könnyű hasonló állításokat levezetni a rezolúciós szabály alkalmazásának bármely sorozatára:

Bármely az S' lezárásához tartozó C' klózhoz létezik egy S lezárásában levő C klóz, amelyre a C' klóz a C klóz alappéldánya, és a C levezetése azonos hosszúságú a C' levezetésével.

Ebből a tényből következik, hogy ha S' lezárásában megjelenik az üres klóz, akkor ez szintén megtalálható az S rezolúciós lezárásában. Ez azért van, mivel az üres klóz semmilyen más klóznak sem alappéldánya. Összefoglalva: megmutattuk, hogyha S nem kielégíthető, akkor létezik az üres klóznak egy véges méretű rezolúciós szabályt alkalmazó levezetése.

Az elmélet bizonyításában az alapklózokról az elsőrendű klózokra történő kiterjesztés az állítás erejének igen jelentős növelése. Ez abból következik, hogy az elsőrendű bizonyításban már csak annyiszor kell a változókat helyettesíteni, amennyiszer ez a bizonyításhoz szükséges, míg az alapklózmódszereknél szükséges volt nagyszámú önkényes példányosítást megvizsgálni.

5.5. Az egyenlőség kezelése

Az ebben a fejezetben az eddig leírt következtetési módszerek közül egyik sem foglalkozott az egyenlőséggel. Három jól elkülöníthető megközelítést alkalmazhatunk. Az első az, hogy axiómákkal látjuk el az egyenlőséget – vagyis leírunk mondatokat az egyenlőségi relációról a tudásbázisban. El kell mondanunk, hogy az egyenlőség reflexív, szimmetrikus, tranzitív, és hogy az egyenlők helyettesíthetők egyenlőkkel bármely predikátumban vagy függvényben. Ezért van szükségünk három alapaxiómára, majd ezután egy továbbira minden egyes predikátumhoz vagy függvényhez:

$\forall x \; x = x$

$\forall x, \; y \; x = y \Rightarrow y = x$

$\forall x, \; y, \; z \; x = y \land y = z \Rightarrow x = z$

$\forall x, \; y \; x = y \Rightarrow (P_1(x) \Leftrightarrow P_1(y))$

Created by XMLmind XSL-FO Converter.

$\forall x,\ y\ x = y \Rightarrow (P_2(x) \Leftrightarrow P_2(y))$

.

.

.

$\forall w,\ x,\ y,\ z\ w = y \wedge x = z \Rightarrow (F_1(w,\ x) = F_1(y,\ z))$

$\forall w,\ x,\ y,\ z\ w = y \wedge x = z \Rightarrow (F_2(w,\ x) = F_2(y,\ z))$

.

.

.

Mikor ezek a mondatok adottak, egy standard következtetési eljárás, mint amilyen a rezolúció elvégezhet olyan feladatokat, amelyek egyenlőségi következtetést igényelnek, mint például a matematikai egyenletek megoldása.

Az egyenlőség kezelésének másik módja egy további következtetési szabály alkalmazása. A legegyszerűbb szabály, a **demoduláció** (**demodulation**) vesz egy egységklózt, $x = y$ és helyettesíti az y-t bármely termmel, ami x-szel egyesíthető egy másik klózban. Formálisabban kifejezve ezt kapjuk:

- **Demoduláció**. Bármely x, y és z termekre, ahol EGYESÍT (x, z) = θ, és az $m_n[z]$ egy literál, ami tartalmazza a z-t.

$$\frac{x = y,\quad m_1 \vee \ldots \vee m_n[z]}{m_1 \vee \ldots \vee m_n[\text{HELYETTESÍT}(\theta, y]}$$

A demodulációt jellemzően arra használják, hogy leegyszerűsítsen állítások kollekcióit használó kifejezéseket, mint például $x + 0 = x$, $x^1 = x$ és így tovább. A szabályt ki lehet terjeszteni, hogy olyan nem egység klózokkal is tudjon foglalkozni, amelyekben egy egyenlőségi literál megjelenik.

- **Paramoduláció** (**paramodulation**). Bármely x, y és z termre, ahol az EGYESÍT (x, z) = θ

$$\frac{\ell_1 \vee \ldots \vee \ell_k \vee x = y,\quad m_1 \vee \ldots \vee m_n[z]}{\text{HELYETTESÍT}(\theta, \ell_1 \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_n[y])}$$

A demodulációval ellentétben a paramoduláció egy teljes következtetési eljárást eredményez az egyenlőséggel rendelkező elsőrendű logikában.

Egy harmadik megközelítés az egyenlőségi következtetést teljes mértékben egy kiterjesztett egyesítési algoritmuson belül kezeli. Ez azt jelenti, hogy a termek egyesíthetők, ha *bizonyíthatóan* egyenlők egy bizonyos helyettesítés alatt, ahol a „bizonyíthatóan” lehetővé tesz egy bizonyos mennyiségű egyenlőségi következtetést. Például, azok a termek, hogy 1 + 2 és 2 + 1 normális esetben nem egyesíthetők, de egy egyesítési algoritmus, amely ismeri, hogy $x + y = y + x$, tudná őket egyesíteni az üres helyettesítéssel. Az ilyen fajta **egyenleti egyesítés** (**equational unification**) elvégezhető hatékony algoritmusokkal, amelyeket arra terveztek, hogy a felhasznált bizonyos axiómákat használja (kommutativitás, asszociativitás és így tovább), ahelyett hogy explicit következtetéseket tenne ugyanazokkal az axiómákkal. A tételbizonyítások, amelyek ezt a technikát használják, szoros kapcsolatban állnak a korlátozott logikai programozási rendszerekkel, amelyeket a 9.4. alfejezetben írtunk le.

5.6. Rezolúciós stratégiák

Tudjuk, hogy a rezolúció ismételt alkalmazása megtalálja a bizonyítást, ha létezik. Ebben a részben olyan stratégiákat tekintünk át, amelyek *hatékonyan* segítenek megtalálni a bizonyításokat.

5.6.1. Egységpreferencia

Created by XMLmind XSL-FO Converter.

Ez a stratégia előnybe helyezi azoknak a mondatoknak az alkalmazását, amelyek egyetlen literált tartalmaznak (szokták ezeket **egységklóz**nak (**unit clause**) is nevezni). A stratégia alapötlete az, hogy próbáljunk meg létrehozni egy üres klózt, így tehát jó ötlet lehet előnyben részesíteni az olyan következtetéseket, amelyek rövidebb klózokat hoznak létre. Egy egységmondat feloldása (mint amilyen a P) egy bármely más mondattal (mint amilyen a $\neg P \vee \neg Q \vee R$), mindig egy olyan klózt eredményez (ebben az esetben: $\neg Q \vee R$), amely rövidebb lesz, mint a másik klóz. Amikor az egységpreferencia-stratégiát először alkalmazták az ítéletkalkulusban, akkor ez drámai gyorsulást eredményezett, megvalósíthatóvá téve olyan bizonyítások elvégzését, amelyek a preferenciák meghatározása nélkül nem voltak kivitelezhetők. Az egységpreferencia azonban önmagában nem csökkenti le eléggé az elágazások számát a közepes méretű problémákban ahhoz, hogy ezek kezelhetők legyenek rezolúcióval. Mindenesetre a módszer egy hasznos heurisztika, amit jól lehet kombinálni más stratégiákkal.

Az **egységrezolúció** (**unit resolution**) a rezolúció egy korlátozott formája, amelyben minden rezolúciós lépésnek tartalmaznia kell egy egységklózt. Az egységrezolúció általában nem teljes, de a Horn-tudásbázisokban teljes. Az egységrezolúciós bizonyítások a Horn-tudásbázisokon emlékeztetnek az előrefelé láncolásra.

5.6.2. Támogató halmaz

Hasznosak azok a módszerek, amelyek megpróbálják meghatározni, hogy melyik rezolúciót érdemes először elvégezni, de még hatékonyabbak, ha megpróbáljuk kizárni a lehetséges rezolúciók egy csoportját is. A **támogató halmaz** (**set of support**) stratégia pontosan ezt teszi. Az eljárás mondatok egy halmazának kiválasztásával kezdődik, amelyet támogató halmaznak nevezünk. Minden rezolúció egy támogató halmazbeli elemet és egy másik, nem halmazbeli elemet kombinál össze, és hozzáadja a rezolvenst a támogató halmazhoz. Ha a támogató halmaz viszonylag kicsi a tudásbázishoz képest, akkor a módszer alkalmazása jelentős mértékben lecsökkenti a keresési teret.

Óvatosan kell alkalmaznunk a megközelítést, mivel az algoritmus nem lesz teljes, ha a támogató halmazt rosszul választjuk meg. Ha az S támogató halmazt úgy választjuk meg, hogy a maradék mondatok együttesen kielégíthetők, akkor a támogató halmaz stratégiáját alkalmazó rezolúció teljes. Elterjedt módszer, hogy a negált lekérdezés mondatot használjuk támogató halmazként, azt feltételezve, hogy az eredeti tudásbázis konzisztens. (Végül is, ha a tudásbázis nem konzisztens, akkor a lekérdezésből következtethető tény is jelentés nélküli.) A támogató halmaz stratégiának további előnye, hogy a létrehozott bizonyítási fák célorientáltak, így az olvasók számára könnyen érthetők.

5.6.3. Bemeneti rezolúció

A **bemeneti rezolúció** (**input resolution**) stratégiában minden rezolúció egy (tudásbázisbeli vagy lekérdező-) mondatot kombinál más mondatokkal. A 9.11. ábrán látható bizonyítás csak bemeneti rezolúciót használ. Ezt az ábrákon sok oldalkapcsolattal rendelkező vonal jellemzi, ahol a kapcsolódásokon egyedi mondatok találhatók. Egyértelmű, hogy az ilyen alakú bizonyítási fák mérete kisebb, mint bármely más bizonyítási fáé. Horn formájú tudásbázisok esetében a Modus Ponens egyfajta bemeneti rezolúciós stratégia, mivel mindig az eredeti *TB* egy mondatát kombinálja egy másik mondattal. Így nem meglepő, hogy a bemeneti rezolúció teljes a Horn formájú tudásbázisok esetében, de nem teljes általános esetben. A **lineáris rezolúció** (**linear resolution**) stratégia egy olyan általánosítás, amely megengedi, hogy P és Q együtt szerepeljenek a rezolúcióban, ha P eredeti eleme a *TB*-nek, vagy ha P leszármazottja Q-nak a bizonyítási fában. A lineáris rezolúció teljes eljárás.

5.6.4. Bennfoglalás

A **bennfoglalás** (**subsumption**) módszere kizárja a keresésből azokat a mondatokat, amelyek benne foglaltatnak (például mert specifikusabbak) más tudásbázisbeli mondatokban. Például ha $P(x)$ megtalálható a tudásbázisban, akkor nincs értelme hozzáadni $P(A)$-t, még kevesebb értelme van hozzáadni $P(A) \vee Q(B)$-t. A bennfoglalás segít kis méreten tartani a *TB*-t, amely a keresési tér méretének csökkentését eredményezi.

5.7. Tételbizonyítók

A tételbizonyítók (amelyeket automatizált következtetőknek is szoktunk nevezni) két szempontból különböznek a logikai programozási nyelvektől. Először is a legtöbb logikai programozási nyelv csak Horn klózokkal dolgozik, ezzel szemben a tételbizonyítások elfogadják a teljes elsőrendű logikát. Másodszor, a Prolog programok egymásba fűzik a logikát és a kontrollt. Ha a programozó választása az A:-B, C-re esik az A:-C, B

Created by XMLmind XSL-FO Converter.

helyett, ez a program végrehajtására van hatással. A legtöbb tételbizonyításban a mondatok választott szintaktikai formája nincs hatással a kapott eredményekre. A tételbizonyításoknak is szükségük van az információ kontrolljára, hogy hatékonyan működhessenek, de ezt az információt általában külön tárolják a tudásbázistól, ahelyett, hogy magának a tudásreprezentációnak a részeként jelenne meg. A legtöbb kutatás a tételbizonyítások tárgyában magában foglalja az általában hasznos kontrollstratégiákat, amelyek a sebességet is növelhetik.

5.7.1. Egy tételbizonyítás szerkesztése

Ebben a részben leírjuk az OTTER (Organized Techniques for Theoremproving and Effective Research) nevű tételbizonyítót (McCune, 1992), különös figyelmet fordítva a vezérlési stratégiájára. Ahhoz, hogy egy problémát készítsen elő az OTTER számára, a felhasználónak négy részre kell osztania a tudásbázist:

- Egy klózhalmazra, melyet **támogató halmaz**nak (**set of support**) (vagy *sos*-nek) nevezünk, és amely definiálja a problémáról szóló legfontosabb tényeket. Minden rezolúciós lépés a támogató halmaz egy tagját rezolválja egy másik axiómához képest, tehát a keresést a támogató halmazra koncentráljuk.

- A **használható axiómá**kra (**usable axioms**), amely egy halmaz a támogató halmazon kívül. Ez háttértudást biztosít a problématerületről. A határ megválasztása a probléma részei (tehát ami az *sos*-en belül van) és a háttér (tehát a használható axiómák) között a felhasználó megítélésén múlik.

- Az egyenletek egy halmazára, amelyeket **átírás**oknak (**rewrite**s) vagy **demodulátor**oknak (**demodulator**s) nevezünk. Habár a demodulátorok egyenletek, mindig balról jobbra irányban alkalmazzák őket. Így tehát egy kanonikus formát határoznak meg, amelyben az összes term leegyszerűsödik. Például az $x + 0 = x$ demodulátor kimondja, hogy minden termet az $x + 0$ formában az x termmel kell helyettesíteni.

- Paraméterek és klózok egy halmazára, amelyek meghatározzák a vezérlési stratégiát. A felhasználónak egy heurisztikus funkciót szükséges specifikálni, hogy kontrollálja a keresést, és egy szűrő funkciót, hogy kitöröljön néhány lényegtelen részcélt.

Az OTTER úgy működik, hogy folyamatosan rezolválja a támogatóhalmaz egy elemét az egyik használható axiómával szemben. A Prologgal ellentétben, ez egy „a legjobbat-először” keresést használ. A heurisztikus funkciója megméri minden egyes klóz „súlyát”, ahol a könnyebb klózokat részesíti előnyben. A heurisztika egzakt kiválasztása a felhasználón múlik, de általában a klóz súlyának korrelálnia kell a méretével vagy a nehézségével.

9.14. ábra - Az OTTER tételbizonyítás vázlata. A heurisztikus kontrollt alkalmazzák a „legkönnyebb” klóz kiválasztására és a FILTER funkcióban, amely kitörli a jelentéktelen klózokat a további vizsgálatból.

Created by XMLmind XSL-FO Converter.

```
procedure OTTER (sos, használható)
  inputs: sos, egy támogató halmaz – a problémát meghatározó klózok (egy globális változó)
          használható, háttértudás, amely potenciálisan releváns a problémával

  repeat
    klóz ← az sos legkönnyebb tagja
    átteszi a klóz-t az sos-ből a használható-ba
    FELDOLGOZ(KÖVETKEZTET(klóz, használható), sos)
  until sos = [] or egy cáfolatot talál

function KÖVETKEZTET(klóz, használható) returns klózokat

  rezolvál klózok-at a használható minden egyes tagjával
  return az eredményezett klózokat a FILTER alkalmazása után

procedure FELDOLGOZ(klózok, sos)

  for each klóz in klózok do
    klóz ← EGYSZERŰSÍT(klóz)
    egybevon megegyező literálokat
    selejtez klózt, ha az tautológia
    sos ← [klóz|sos]
    if a klóz-nak nincs literálja then egy cáfolatot talált
    if a klóz-nak egy literálja van then keres egység cáfolatot
```

Az egységklózokat könnyűként kezeljük; így tehát a keresést az egységpreferencia-stratégia általánosításának tekinthetjük. Minden egyes lépésnél az OTTER átteszi a „legkönnyebb” klózt a támogatóhalmazból a használható listára, és hozzáadja a használható listához a legkönnyebb klóz és a használható lista elemeinek rezolválásakor keletkezett néhány közvetlen következményt. Az OTTER leáll, amikor talál egy cáfolatot, vagy amikor nincs több klóz a támogatóhalmazban. Az algoritmus a 9.14. ábrán látható részletesebben.

5.7.2. A Prolog kiterjesztése

Egy tételbizonyító létrehozásának alternatív módja, ha a Prolog fordítóból indulunk ki, és ezt kiterjesztjük úgy, hogy egy helyes és teljes következtetőgépet kapjunk a teljes elsőrendű logikához. Ezt a megközelítést alkalmazták a Prolog Technológia Tételbizonyítóban (Prolog Technology Theorem Prover, PTTP) (Stickel, 1988). A PTTP négy jelentős változtatást tartalmaz a Prologhoz képest, hogy helyreállítsa a teljességet és a kifejezőképességet:

- Az előfordulási próba visszakerül az egyesítési folyamatba, hogy azt biztosabbá tegye.
- A mélységi keresést felváltja egy iteratívan mélyülő keresés. Ez a keresési stratégiát teljessé teszi, és csak egy konstans értékkel vesz több időt igénybe.
- A negált literálok (mint a $\neg P(x)$) engedélyezettek. A megvalósításban két különálló folyamat van, az egyik a P-t akarja bebizonyítani, a másik a $\neg P$-t.
- Egy n atommal rendelkező klóz n különböző szabályként kerül tárolásra. Például az $A \Leftarrow B \wedge C$-t úgy is tárolnánk, mint $\neg B \Leftarrow C \wedge A$ és úgy is, mint $\neg C \Leftarrow B \wedge \neg A$. Ez a technika, amelyet **zárás**nak (**locking**) nevezünk, azt eredményezi, hogy az aktuális célt csak minden egyes klóz fejével kell egyesíteni, de még így is lehetővé teszi a negálás helyes kezelését.
- A következtetést teljessé teszik (még a nem Horn-klózokra is) a lineáris bemeneti szabály hozzáadásával: ha a z aktuális cél egyesül a veremben lévő egyik cél negáltjával, akkor azt a célt rezolváltnak kell tekinteni. Ez egy módja az ellentmondásokkal való következtetésnek. Tételezzük fel, hogy az eredeti cél a P, és ezt redukáltuk következtetések sorozatával a $\neg P$ célra. Ez a $\neg P \Rightarrow P$ mondatra vezet, amely logikailag egyenértékű a P-vel.

Created by XMLmind XSL-FO Converter.

A felsorolt változtatások ellenére a PTTP megőrzi azokat a tulajdonságokat, amelyek a Prologot gyorssá teszik. Az egyesítések még így is a változók közvetlen változtatásával történnek úgy, hogy a lekötések feloldása a visszalépés során az útvonal lezárásával együtt történik. A keresési stratégia itt is a bemeneti rezolúción alapszik, ami azt jeleneti, hogy minden rezolúció a probléma egyik eredeti kijelentésében megadott klózzal szemben történik (és nem egy származtatott klózzal). Ez lehetővé teszi a probléma eredeti kijelentésében adott összes klóz lefordítását.

A PTTP fő hátránya az, hogy a felhasználó elveszít minden vezérlési lehetőséget a megoldások keresése során. Minden következtetési szabályt a rendszer mind az eredeti, mind a kontrapozitív formájában felhasznál. Ez nehezen értelmezhető kereséseket eredményez. Például figyeljük meg ezt a szabályt:

$$(f(x, y) = f(a, b)) \Leftarrow (x = a) \wedge (y = b)$$

Prolog szabályként tekintve, ez is egy következetes módja annak a bizonyításának, hogy két *f* term megegyezik. De a PTTP még a kontrapozitív állítást is generálhatja:

$$(x \neq a) \Leftarrow (f(x, y) \neq f(a, b)) \wedge (y = b)$$

Úgy tűnik, hogy ez igen fáradságos módja annak, hogy bizonyítsuk két term, *x* és *a* különbözőségét.

5.7.3. Tételbizonyítások mint segédeszközök

Eddig úgy tekintettünk a következtető rendszerekre, mint független ágensre, amelynek önállóan kell döntéseket hoznia és cselekednie. A tételbizonyítások másik felhasználása, amikor segítőként használjuk, és tanácsokkal lát el, mondjuk egy matematikust. Ebben a használati módban a matematikus felügyelőként viselkedik, feltérképezi a következő lépés meghatározásának stratégiáját, hogy mi legyen, és megkéri a tételbizonyítót, hogy töltse ki a részleteket. Egy bizonyos fokig mellékessé teszi a félig eldönthetőség problémáját, mivel a felügyelő megszüntethet egy lekérdezést, és próbálhat egy másik megközelítést, ha a lekérdezés túl sok időt vesz igénybe. A tételbizonyítás **bizonyítás-ellenőrző**ként (**proof checker**) is működhet, ahol a bizonyítást mi adjuk meg vázlatosan, nagyobb lépések sorozataként, és a rendszer tölti ki az egyedi következtetések részleteit, amelyek igazolják lépéseink helyességét.

A **Socratic következtető** (**Socratic reasoner**) egy tételbizonyító, amelynek a KÉRDEZ függvénye nem teljes, de amely mindig elérkezhet egy megoldáshoz, ha a megfelelő kérdéssorozatot tesszük fel. Így tehát a Socratic következtetők jó segítők, feltéve, ha van egy felügyelő, aki a megfelelő KÉRDEZ hívásokat összeállítja. Az ONTIC (McAllester, 1989) egy Socratic következtető rendszer matematikai tételbizonyításhoz.

5.7.4. A tételbizonyítások gyakorlati felhasználása

A tételbizonyítók újszerű matematikai eredményeket produkáltak. A SAM (Semi-Automated Mathematics) program volt az első, amely bebizonyított egy lemmát a rácselméletben (Guard és társai, 1969). Az AURA program szintén nyitott kérdéseket válaszolt meg a matematika különféle területein (Wos és Winker, 1983). A Boyer–Moore-tételbizonyítót (Boyer és Moore, 1979) sok éven keresztül használták és bővítették. Natarajan Shankar ezt használta fel, hogy megadja az első teljes, precíz formális bizonyítását Gödel nemteljesség tételének (Shankar, 1986). Az OTTER program az egyik legerősebb tételbizonyító. Használták a kombinatorikus logika néhány nyitott kérdésének megoldására. A legismertebb ezek közül a **Robbins-algebra**. 1933-ban Herbert Robbins egy egyszerű axiómasorozatot javasolt, amely a Boole-algebra definiálására látszott alkalmasnak, de nem találtak hozzá bizonyítást (több matematikus, köztük Alfred Tarski, jelentős munkája ellenére). 1996. október 10-én, nyolc nap számítás után az EQP (az OTTER egyik változata) megtalálta a bizonyítást (McCune, 1997).

A tételbizonyításokat alkalmazhatjuk a hardver- és szoftvertervezésben a **verifikáció** (**verification**) és a **szintézis** (**synthesis**) során felmerülő problémákra, mivel mindkét tárgyterülethez lehetséges megfelelő axiómarendszert definiálni. Így tehát a tételbizonyítási kutatást megtalálhatjuk a hardvertervezés, a programozási nyelvek és a szoftverfejlesztés területein is – nem csupán az MI-ben. A szoftver esetében az axiómák meghatározzák a programozási nyelv minden egyes szintaktikus elemének tulajdonságait. (A programokról történő következtetés eléggé hasonló az akciókról történő következtetéshez a szituációkalkulusban.) Egy algoritmus akkor tekinthető igazoltnak, ha a kimenetei megfelelnek a specifikációnak minden bemenet mellett. Az RSA nyilvános kulcskódolási algoritmust és a Boyer–Moore-féle húrillesztési algoritmust is ilyen módon igazolták (Boyer és Moore, 1984). A hardver esetében az axiómák leírják a jelek és az áramkör elemei közötti interakciókat (lásd a 8. fejezetbeli példát). Egy 16 bites összeadó

Created by XMLmind XSL-FO Converter.

tervét az AURA igazolta (Wojcik, 1983). A logikai következtetők, amelyeket speciálisan igazolásokra terveztek, egész CPU-kat voltak képesek verifikálni, beleértve ezek időzítési tulajdonságait is (Srivas és Bickford, 1990).

Az algoritmusok formális szintézise volt a tételbizonyítások egyik első felhasználása, amint azt Cordell Green (Green, 1969a) felvázolta, aki Simon korábbi ötleteire támaszkodott (Simon, 1963). Az alapötlet az volt, hogy egyféle módon bizonyítsák azt a tételt, hogy „létezik egy *p* program, amely eleget tesz egy bizonyos specifikációnak”. Ha a bizonyítást úgy korlátozzuk, hogy konstruktív legyen, a program maga is kinyerhető lesz a bizonyításból. Habár ez egy teljesen automatizált **deduktív szintézis** (**deductive synthesis**), legalábbis így nevezzük, még nemigen használható általános célú programok készítésére, ám a kézzel irányított deduktív szintézis már sikeresen tervezett néhány újszerű és bonyolult algoritmust. A speciális célú programok szintézise szintén aktív kutatási terület. Az AURA tételbizonyítót sikerrel alkalmazták a hardverszintézis területén olyan áramkörök tervezésére, amelyek kompaktabbak, mint minden azt megelőző terv (Wojciechowski és Wojcik, 1983). Sok áramkör tervezésénél az ítéletlogika elegendő, mivel a legfontosabb ítéletállítások halmaza rögzített, ugyanis ezek az áramkör elemeit írják le. Az ítéletlogikai következtetés alkalmazása a hardverszintézisben ma már egy standard technika, amelynek sok nagyméretű hálózati alkalmazása létezik (lásd például Nowick és társai, 1993).

Ugyanezeket a technikákat mostanában kezdik el alkalmazni a szoftverek verifikációjához is, például a SPIN modell ellenőrző programmal (Holzmann, 1997). A Remote Agent űrhajó vezérlési programot például sikerült verifikálni a repülés előtt és után (Havelund és társai, 2000).

6. Összefoglalás

Az elsőrendű logikai következtetés egy elemzését mutattuk be, továbbá számos algoritmust, melyek ilyen következtetéseket képesek végrehajtani.

- Egy első megközelítés következtetési szabályokat használ kvantorok példányosítására azért, hogy átalakítsa a következtetési problémát az ítéletkalkulusra. Ez a megközelítés jellemzően nagyon lassú megoldásokat eredményez.

- A **egyesítés** használata, azaz a megfelelő helyettesítések megkeresése változókhoz, megszünteti a példányosítás lépését az elsőrendű bizonyításokban, és így sokkal hatékonyabbá teszi a folyamatot.

- A **Modus Ponens** kiterjesztett változata az egyesítést használja, létrehozva egy természetes és nagy modellező erejű következtetési szabályt, az **általánosított Modus Ponens**t (**generalized Modus Ponens**). Az **előrefelé láncolás** (**forward chaining**) és a **hátrafelé láncolás** (**backward chaining**) algoritmusok határozott klózok halmazára alkalmazzák ezt a szabályt.

- Az általánosított Modus Ponens teljes a határozott klózokra, habár a következtetési probléma **félig eldönthető** (**semidecidable**). A **Datalog** programok esetében, amely programok csak függvénymentes határozott klózokat tartalmaznak, a következtetés eldönthető.

- Az előrefelé láncolást a **deduktív adatbázis**okban (**deductive database**s) használják, ahol sikeresen kombinálják ezt a módszert a relációs adatbázis más műveleteivel. Ezenkívül az előrefelé láncolást használják a **produkciós rendszer**ekben (**production system**s) is, amelyek hatékonyan képesek frissíteni a tudásbázisukat nagyméretű szabályhalmazokkal is.

- Az előrefelé láncolás teljes a Datalog programokra, és polinomiális időben fut.

- A hátrafelé láncolást a **logikai programozási rendszer**ekben (**logic programming system**s) használják, mint amilyen például a **Prolog**, amely kifinomult fordítási technikákat alkalmaz, hogy biztosítsa a gyors következtetést.

- A hátrafelé láncolásban sok a felesleges következtetés, és előfordulhatnak végtelen hurkok. Ezeket a **memók gyűjtésé**vel (**memoization**) lehet kezelni.

- Az általánosított **rezolúció**s (**resolution**) következtetés teljes bizonyítási rendszer az elsőrendű logikában. Az eljárás konjunktív normál formájú tudásbázisokat használ fel.

Created by XMLmind XSL-FO Converter.

- Számos stratégia ismert a rezolúciós rendszerek keresési terének a csökkentésére, anélkül hogy a teljességet veszélybe sodornánk. A hatékony rezolúcióalapú tételbizonyításokat felhasználták, hogy érdekes matematikai tételeket bebizonyítsanak, és hogy verifikáljanak, valamint szintetizáljanak szoftvereket és hardvereket.

6.1. Irodalmi és történeti megjegyzések

A logikai következtetéseket széles körben vizsgálták már az ókori görög matematikusok is. Az Arisztotelész által legtöbbet vizsgált következtetési módszer a **szillogizmus** (**syllogism**) volt. A szillogizmus „szituációkra” és „hangulatokra” bontható a termek mondatbeli sorrendjétől függően (amelyeket predikátumoknak nevezünk ma) és az egyes termek általánosságának mértékétől függően (amit ma a kvantorok segítségével értelmezünk), illetve attól függően, hogy a termek negálva voltak-e. A legalapvetőbb szillogizmus, amely az első szituáció első hangulata, a következő:

Minden *S* az *M* is.

Minden *M* a *P* is.

Ezért minden *S* a *P* is.

Arisztotelész más szillogizmusok érvényességét is megpróbálta igazolni, visszavezetve ezeket az első szituációra. Pontosan leírta az egyes szillogizmusokhoz tartozó hangulatokat és szituációkat, de a szillogizmusok igazolása már kevésbé volt precíz.

Gottlob Frege, aki teljes elsőrendű logikát fejlesztett ki 1879-ben, következtetési rendszerét logikailag érvényes sémák nagyméretű gyűjteményére és egyetlen következtetési szabályra alapozta, a Modus Ponensre. Frege kihasználta azt a tényt, hogy ennek a formának „A *P*-ből következtesd a *Q*-t” következtetési szabályának a hatását szimulálhatja a Modus Ponens alkalmazásával a *P*-re, egy logikailag érvényes sémával, a $P \Rightarrow Q$-val. A kifejezésnek ezt az „axiomatikus” stílusát, amely a Modus Ponenst, plusz számos logikailag érvényes sémát használ, Frege után még számos matematikus használta; a leginkább említésre méltó, hogy a *Principia Mathematicá*ban (Whitehead és Russell, 1910) is felhasználták.

A következtetési szabályok, mint az axiomatikus sémáktól különálló dolgok álltak a **természetes dedukció** (**natural deduction**) központjában, amelyet Gerhard Gentzen és Stanislaw Jáskowski vezetett be (Gentzen, 1934; Jáskowski, 1934). A természetes dedukciót azért nevezik „természetesnek”, mert nem kívánja meg a mondatok átalakítását az (olvashatatlan) normál formára, és következtetési szabályaikat úgy tervezték, hogy az emberek számára természetesnek tűnjenek. Prawitz egy egész könyv hosszúságú elemzést adott a természetes dedukcióról (Prawitz, 1965). Gallier az automatikus dedukció elméleti alátámasztásának kifejtéséhez alkalmazta Gentzen szekvenciáit (Gallier, 1986).

A klózforma felfedezése nagy jelentőségű lépés volt az elsőrendű logika matematikai elemzésének kifejlesztésében. Whitehead és Russell (Whitehead és Russell, 1910) kiterjesztették az úgynevezett *átviteli szabályt* [maga a kifejezés Herbrandtól származik (Herbrand, 1930)], amit a kvantoroknak a formula elé történő kiemelésére alkalmaztak. A Skolem-konstansokat és a Skolem-függvényeket, viszonylag precíz leírást adva, Thoralf Skolem vezette be (Skolem, 1920). A skolemizáció teljes eljárása, a Herbrand-univerzum fontos fogalmának bevezetésével együtt egy későbbi írásban található (Skolem, 1928).

Herbrand elmélete, amit Jacques Herbrand francia matematikusról neveztek el, fontos szerepet játszott az automatikus következtető rendszerek módszereinek fejlesztésében, mind a Robinsonhoz fűződő rezolúció bevezetése előtt és után (Herbrand, 1930). Ezt tükrözi az, hogy habár maga a fogalom valójában Skolem eredménye, erre „Skolem-univerzum” helyett „Herbrand-univerzumként” hivatkozunk. Herbrandot említhetjük még az egyesítés felfedezőjeként is, mivel az egyesítés algoritmus egy változata megtalálható a (Herbrand, 1930)-ban. Gödel (Gödel, 1930) mutatta meg, Skolem és Herbrand elméleteire építve, hogy az elsőrendű logikának létezik teljes bizonyítási eljárása. Alan Turing és Alonzo Church egymástól függetlenül, igen különböző bizonyításokat alkalmazva megmutatták, hogy az érvényesség kérdése az elsőrendű logikában nem eldönthető (Turing, 1936; Church 1936). Enderton kiválóan megfogalmazott írása egy precíz, de viszonylag érthető stílusban magyarázza el ezeket az eredményeket (Enderton, 1972).

Habár McCarthy (McCarthy, 1958) javasolta először az elsőrendű logika használatát reprezentációs feladatokra és következetésre az MI-ben, az első ilyen rendszert a matematikai tételbizonyítás iránt érdeklődő matematikai logika kutatói fejlesztették ki. Abraham Robinson javasolta először, hogy használják az ítéletlogikára történő átalakítást és Herbrand-elméletet együtt. Gilmore (Gilmore, 1960) készített először egy olyan programot, amely

Created by XMLmind XSL-FO Converter.

ezen a megközelítésen alapult. Davis és Putnam (Davis és Putnam, 1960) vezették be a klózformát, és hoztak létre egy olyan programot, amely megkísérelt cáfolatokat találni, oly módon, hogy a Herbrand-univerzum elemeit a változók helyére alapklózokat helyettesítve és ítéletinkonzisztenciákat keresve az alapklózok között. Prawitz (Prawitz, 1960) vezette be azt az alapötletet, hogy az ítéletinkonzisztencia lekérdezése vezesse a keresési folyamatot, és azt is, hogy csak akkor hozzunk létre termeket a Herbrand-univerzumban, hogyha ez szükséges az ítélet-inkonzisztencia megállapítása céljából. Más kutatók egyéb fejlesztései után, ez a gondolat vezette el J. A. Robinsont (nem rokona az előzőnek) a rezolúció módszerének kifejlesztéséhez (Robinson, 1965). Az úgynevezett „inverz módszer", amit egyidejűleg fejlesztett ki S. Maslov szovjet kutató, kissé eltérő elveken alapult, mint Robinson rezolúciós módszere, de hasonló számítási előnyöket nyújt a propozicionalizációhoz képest (Maslov, 1964, 1967). Wolfgang Bibel **kapcsolati módszer**e (**connection method**) e megközelítés kiterjesztésének tekinthető.

A rezolúció kifejlesztése után az elsőrendű következtetéssel foglalkozó munkák különböző irányokban ágaztak el. Az MI-ben a rezolúciót a kérdés-válasz rendszerekre adoptálta Cordell Green és Bertram Raphael (Green és Raphael, 1968). Egy kevésbé formális megközelítést alkalmazott Carl Hewitt (Hewitt, 1969). Az ő PLANNER nyelve, habár sohasem valósították meg teljességében, előfutára volt a logikai programozásnak, és útmutatásokat tartalmazott az előrefelé és hátrafelé láncoláshoz, valamint a negáltak sikertelenségének vizsgálati módszeréhez. Az eredeti nyelv egy részét, a MICRO-PLANNER-t (Sussman és Winograd, 1970) megvalósították és felhasználták az SHRDLU természetes nyelvek megértését segítő rendszerben (Winograd, 1972). A korai MI-alkalmazások jelentős erőfeszítést tettek olyan adatstruktúrák előállítására, amelyek lehetővé teszik tények hatékony előhívását. Ezeket a munkákat is bemutatják az MI-programozással kapcsolatos cikkek (Charniak és társai, 1987; Norvig, 1992; Forbus és de Kleer, 1993).

A hetvenes évek elejére az **előrefelé láncolás** (**forward chaining**) jól megalapozottnak tekinthető az MI-ben, egy könnyen érthető alternatívája lett a rezolúciónak. Rendszerek széles körében került felhasználásra, Nevins geometriai tételbizonyítójától (Nevins, 1975) a VAX konfigurációjához használt R1 szakértői rendszerig (McDermott, 1982). Az MI-alkalmazások jellemzően nagyszámú szabályt tartalmaztak, tehát fontos volt kifejleszteni hatékony szabályillesztési technológiát, különösen az inkrementális frissítések problémájára. A **produkciós rendszer**ek (**production system**s) technológiáját azért fejlesztették ki, hogy segítse az ilyen típusú az alkalmazások megvalósítását. Az OPS-5 produkciós rendszer nyelvet (Forgy, 1981; Brownston és társai, 1985) használták az R1-hez és a SOAR kognitív szerkezetéhez (Laird és társai, 1987). Az OPS-5 tartalmazta a rete illesztési folyamatot is (Forgy, 1982). A SOAR, amely új szabályokat generál, hogy megőrizze a megelőző számítások eredményeit, igen nagy szabályhalmazokat állíthat elő – több mint 1 000 000 szabályt a TACAIR-SOAR rendszer esetében, amely katonai repülőgépek szimulációját kontrollálja (Jones és társai, 1998). A CLIPS (Wygant, 1989) egy C-alapú produkciós rendszer nyelv, amelyet a NASA-nál fejlesztettek ki, és hatékony integrációt tett lehetővé más programokkal, eszközökkel és érzékelő rendszerekkel, valamint felhasználták űrjárművek automatizálására és egyéb harcászati alkalmazásokra.

A **deduktív adatbázis**okként (**deductive database**s) ismert kutatási terület szintén hozzájárult az előrefelé következtetés megértéséhez. Egy Toulouse-i szellemi műhelyben kezdődött a munka 1977-ben, amelyet Jack Minker szervezett, összehozva a logikai következtetési és az adatbázisrendszerekkel foglalkozó szakembereket (Gallaire és Minker, 1978). Egy történeti áttekintés (Ramakrishnan és Ullman, 1995) szerint: a „deduktív [adatbázis-] rendszerek megkísérlik a Prolog adaptálását, a »kis mennyiségű adat« megközelítésről a »nagymennyiségű adat« világára." Így tehát, ennek a munkának a célja az, hogy egybeolvassza a relációs adatbázis technológiát, amelyet nagyméretű tényhalmazok előhívására terveztek, a Prolog-alapú következtetési technológiával, amely jellemzően egyszerre csak egy tényt hív elő. A deduktív adatbázisokról szóló szövegek közé tartoznak az (Ullman, 1989) és a (Ceri és társai, 1990) munkák.

Chandra és Harel (Chandra és Harel, 1980), valamint Ullman (Ullman, 1985) nagy hatású munkái a Datalognak mint egy, a deduktív adatbázisokhoz kidolgozott standard nyelvnek az elfogadásához vezettek. Az „alulról felfelé" irányú következtetés, az előrefelé láncolás szintén standard lett – részben azért, mert elkerüli a leállási problémákat és a felesleges számítások problémáját, amely a hátrafelé láncolásnál előfordul, és részben azért, mert természetesebben felhasználható az alapvető relációs adatbázis-operációkhoz. A **mágikus halmaz**ok (**magic set**s) technikája a szabályok átírására, amelyet Bancilhon (Bancilhon és társai, 1986) fejlesztettek ki, lehetővé tette az előrefelé láncolásnak, hogy a hátrafelé láncolástól kölcsönözze a célorientáltság előnyös tulajdonságát. A „fegyverkezési verseny" kiegyenlítése céljából, a táblázatos logikai programozási módszerek kölcsönveszik a dinamikus programozás előnyét az előrefelé láncolástól.

A logikai következtetések komplexitásáról szerzett tudásunk legnagyobb része a deduktív adatbázisokkal foglalkozó tudományos közösségből származik. Chandra és Merlin (Chandra és Merlin, 1977) mutatták ki először, hogy egy egyszerű nem rekurzív szabály (vagyis egy **konjunktív lekérdezés** (**conjunctive query**) az

Created by XMLmind XSL-FO Converter.

adatbázisok terminológiájában) illesztése NP-nehéz lehet. Kuper és Vardi (Kuper és Vardi, 1993) javasolták az **adatkomplexitás** (**data complexity**) használatát – annak a mértéknek a használatát, ami a komplexitást, mint az adatbázis méretének egy függvényét méri, miközben a szabály méretét konstansnak tekinti – a lekérdezések megválaszolásának mértékére. Gottlob és társai (Gottlob és társai, 1999b) a konjuktív lekérdezések és a kényszerkielégítés közötti kapcsolatot vizsgálták, megmutatva, hogy a hiper-fák lebontása hogyan optimizálja az illesztési folyamatot.

Mint azt már előzőleg említettük, a **hátrafelé láncolás** (**backward chaining**) Hewitt `PLANNER` nyelvében jelent meg a logikai következtetéshez (Hewitt, 1969). A logikai következtetés magától értetődően ettől a munkától függetlenül fejlődött tovább. A lineáris rezolúció egy korlátozott változatát, amelyet **SL-rezolúció**nak (**SL-resolution**) nevezünk, Kowalski és Kuehner (Kowalski és Kuehner, 1971) fejlesztette ki Loveland **modelleliminációs** (**model elimination**) technikájára építve (Loveland, 1968). Ennek a határozott klózokra alkalmazott változata az **SLD-rezolúció** (**SLD-resolution**), amely önmagában alkalmas határozott klózok, mint programok interpretálására (Kowalski, 1974; 1979a; 1979b). Ezalatt, 1972-ben, a francia kutató, Alain Colmerauer kifejlesztette és alkalmazni kezdte a **Prolog**ot, a természetes nyelvek elemzésének céljából. A Prolog klózait először kontextusmentes nyelvtani szabályoknak szánták (Roussel, 1975; Colmerauer és társai, 1973). Az elméleti háttér túlnyomó részét a logikai programozás részére Colmerauerrel együttműködve Kowalski fejlesztette ki. A szemantikai definíció, amely a rögzített pontokat használja, Van Emdennek és Kowalskinak köszönhető (Van Emden és Kowalski, 1976). Kowalski (Kowalski, 1988) és Cohen (Cohen, 1988) jó történeti áttekintést nyújtanak a Prolog eredetéről. A *Logikai programozás alapjai (Foundations of Logic Programming)* (Lloyd, 1987) egy elméleti elemzése a Prolog és egyéb logikai programozónyelvek megalapozásának.

A hatékony Prolog fordítókat általában a Warren Abstract Machine (WAM) számítási modelljére alapozzák, amelyet David H. D. Warren (Warren, 1983) fejlesztett ki. Van Roy (Van Roy, 1990) megmutatta, hogy a további fordítási technikák alkalmazása, mint amilyen a típusinterferencia, versenyképessé teszi a Prolog programokat a C programokkal a sebesség tekintetében. A Japán Ötödik Generációs projekt, amely egy 1982-ben kezdődő 10 éves kutatási erőfeszítés volt, teljes egészében a Prologon, mint az intelligens rendszerek kifejlesztésének alapeszközén alapult.

A rekurzív logikai programok felesleges ciklusainak elkerüléséhez egymástól függetlenül Smith (Smith és társai, 1986), valamint Tamaki és Sato (Tamaki és Sato, 1986) fejlesztettek ki módszereket. Az utóbbi tanulmány tartalmazta a memók gyűjtését is a logikai programokra, egy olyan módszert, amelyet teljes mértékben David S. Warren fejlesztett ki a **táblázatos logikai programozás** (**tabled logic programming**) módszerében. Swift és Warren (Swift és Warren, 1994) bemutatták, hogy hogyan terjesszük ki a WAM-ot táblázatok kezelésére, amely képessé tette a Datalog programokat, hogy egy nagyságrenddel gyorsabban fussanak, mint az előrefelé láncolást alkalmazó deduktív adatbázisrendszerek.

A korai elméleti munkákat a korlátozott logikai programozás területén Jaffar és Lassez végezték (Jaffar és Lassez, 1987). Jaffar és társai (Jaffar és társai, 1992a) kifejlesztették a CLP(R) rendszert a valós értékű kényszerek kezelésére. Jaffar (Jaffar és társai, 1992b) általánosította a WAM-ot, létrehozva ezzel a CLAM-ot (Korlátozott Logikai Absztrakt Gép, Constraint Logic Abstract Machine) a CLP megvalósításainak specifikálásához. Ait-Kaci és Podelski (Ait-Kaci és Podelski, 1993) bemutattak egy kifinomult LIFE-nak nevezett nyelvet, amely egyesíti a CLP-t a funkcionális programozással és az öröklődés következtetéssel. Kohn (Kohn, 1991) egy ambiciózus projektet mutat be a korlátozott logikai programozás használva, a valós idejű vezérlési architektúrákat megalapozva, teljesen automata pilóták alkalmazására.

A logikai programozásról és a Prologról számos könyvet írtak. A *Logika a problémamegoldáshoz (Logic for Problem Solving)* (Kowalski, 1979b) egy korai általános tanulmány a logikai programozásról. A Prologgal foglalkozó könyvek közé tartoznak Clocksin és Mellish (Clocksin és Mellish, 1994), Shoham (Shoham, 1994) és Bratko (Bratko, 2001) írásai. Marriott és Stuckey (Marriott és Stuckey, 1998) kitűnő leírását adják a CLP-nek. A 2000-ben történt megszűnéséig a *Journal of Logic Programming* volt a téma legfontosabb folyóirata; helyét mára átvette a *Theory and Practice of Logic Programming.* A logikai programozások konferenciái között a legjelentősebbek az International Conference on Logic Programming (ICLP) és az International Logic Programming Symposium (ILPS).

A kutatások a **matematikai tételbizonyítás**ok (**mathematical theorem proving**) területén már az első teljes elsőrendű logikai rendszerek kifejlesztése előtt megkezdődtek. Herbert Gelernter Geometriai Tételbizonyítója (Gelernter, 1959) heurisztikai keresési módszereket használt diagramokkal kombinálva, hogy kiselejtezze a hamis részcélokat, és képes volt bebizonyítani néhány nagyon bonyolult eredményt az euklideszi geometriában. Ettől kezdve azonban nem volt jelentős együttműködés a tételbizonyítás és az MI között.

Created by XMLmind XSL-FO Converter.

A korai munkák a teljességre koncentráltak. Robinson korszakalkotó tanulmányát követően a demodulációs és a paramodulációs szabályokat az egyenlőségi következtetésekre Wos (Wos és társai, 1967), valamint Wos és Robinson (Wos és Robinson, 1968) vezették be, ebben a sorrendben. Ezeket a szabályokat a term átírási rendszerek kontextusában is kifejlesztették (Knuth és Bendix, 1970). Az egyenlőségi következtetés beépítése az egyesítési algoritmusba Gordon Plotkinnak köszönhető (Plotkin, 1972); ez fontos vonása volt a QLISP-nek is (Sacerdoti és társai, 1976). Jouannaud és Kirchner (Jouannaud és Kirchner, 1991) az egyenlőségi egyesítést a termek átírásának szempontjából vizsgálják. Hatékony algoritmusokat a standard egyesítésre Martelli és Montanari (Martelli és Montanari, 1976), valamint Paterson és Wegman (Paterson és Wegman, 1978) fejlesztettek ki.

Az egyenlőségi következtetések mellett a tételbizonyítások tartalmaztak különféle speciális célú döntési folyamatokat. Nelson és Oppen (Nelson és Oppen, 1979) javasoltak egy nagy hatású sémát az ilyen eljárásoknak egy általános következtetési rendszerbe integrálására. Más módszerek is hasonló problémákkal foglalkoztak, beleértve Stickel (Stickel, 1985) „elméleti rezolúcióját", valamint Manna és Waldinger (Manna és Waldinger, 1986) „speciális relációit".

Számos vezérlési stratégiát javasoltak a rezolúcióra, kezdve az egységpreferencia stratégiával (Wos és társai, 1964). A támogató halmaz stratégiát Wos (Wos és társai, 1964) javasolta, hogy biztosítson egy bizonyos fokú célorientáltságot a rezolúcióban. A lineáris rezolúció először Lovelandnél (Loveland, 1970) jelent meg. Genesereth és Nilsson (Genesereth és Nilsson, 1987, 5. fejezet) rövid, de átfogó elemzést nyújtanak a vezérlési stratégiák széles skálájáról.

Guard és társai (Guard és társai, 1969) egy korai SAM tételbizonyítót írnak le, amely segített megoldani egy megoldatlan problémát a rácselméletben. Wos és Winkler (Wos és Winkler, 1983) áttekintést nyújtanak az AURA tételbizonyító eredményeiről, amelyeket a matematika és a logika különböző területein fellelhető problémák megoldása terén ért el. McCune (McCune, 1992) ezt folytatja, amikor felhasználja ezeket az eredményeket az AURA utódjának, az OTTER-nek alkalmazásában megoldatlan problémák megoldására. Weidenbach (Weidenbach, 2001) bemutatja a SPASS-t, az egyik legerőteljesebb jelenlegi tételbizonyítót. *A Computational Logic* (Boyer és Moore, 1979) című könyv az alapreferencia a Boyer–Moore-tételbizonyítókhoz. Stickel (Stickel, 1988) leírja a Prolog Technológiai Tételbizonyítót (PTTP), amely egyesíti a Prolog fordítás előnyeit a modell elimináció teljességével (Loveland, 1968). A SETHEO (Letz és társai, 1992) egy másik széles körben használt tételbizonyító, amely hasonló megközelítésen alapul; másodpercenként több millió következtetést tud végrehajtani egy 2000-es munkaállomáson. A LEANTAP (Beckert és Posegga, 1995) hatékony tételbizonyító, amelyet mindössze 25 Prolog sorral valósítottak meg.

Az automata programszintézisről szóló korai munkákat Simon (Simon, 1963), Green (Green, 1969a), valamint Manna és Waldinger (Manna és Waldinger, 1971) készítették. Burstall és Darlington transzformációs rendszere (Burstall és Darlington, 1977) egyenlőség következtetést használt a rekurzív programszintézishez. A KIDS (Smith, 1990, 1996) az egyik legerősebb modern rendszer; amely egy szakértő segédeszközként működik. Manna és Waldinger (Manna és Waldinger, 1992) egy áttekintő bevezetést adtak a témakör aktuális helyzetéről fókuszba állítva a saját deduktív megközelítésüket. Az *Automating Software Design* (Lowry és McCartney, 1991) számos, a témáról szóló tanulmányt gyűjtött össze. A logikának a hardvertervezésben való felhasználásáról Kern és Greenstreet (Kern és Greenstreet, 1999) adott egy áttekintést; Clarke (Clarke és társai, 1999) műve pedig a modellellenőrzéssel foglalkozik a hardververifikálásban.

A *Computability and Logic* (Boolos és Jeffrey, 1989) egy jó referencia a teljesség és a nem eldönthetőség témaköréhez. Számos korai tanulmány a matematikai logikáról megtalálható a *From Frege to Gödel: A Source Book in Mathematical Logic* (van Heijenoort, 1967) című könyvben. A tiszta matematikai logika témakörének folyóirata a *Journal of Symbolic Logic*. Az automatizált dedukció irányába haladó könyvek közé tartozik a klasszikus *Symbolic Logic and Mechanical Theorem Proving* (Chang és Lee, 1973), és számos más írás, köztük Wos, Bibel és Kaufmann később íródott munkái (Wos és társai, 1992; Bibel, 1993; Kaufmann és társai, 2000). Az *Automation of Reasoning* c. antológia (Siekmann és Wrightson, 1983) sok fontos korai tanulmányt tartalmaz az automatizált dedukcióról. További áttekintő műveket írtak Loveland (Loveland, 1984) és Bundy (Bundy, 1999). A legjelentősebb folyóirat a tételbizonyítások területén a *Journal of Automated Reasoning;* a legfontosabb konferencia az évente megtartott Conference on Automated Deduction (CADE). A tételbizonyítás területén folyó kutatás szintén szoros kapcsolatban áll a logika használatával a programok és programozási nyelvek elemzésében, amely tárgyban a legfőbb konferencia a Logic in Computer Science.

6.2. Feladatok

Created by XMLmind XSL-FO Converter.

9.1.

Vezesse le az alaptételekből, hogy az univerzális példányosítás helyes, és azt, hogy az egzisztenciális példányosítás következtetési szempontból egyenértékű tudásbázist hoz létre.

9.2.

A *Szereti*(*János*, *Fagylalt*) mondatból kiindulva logikusnak tűnik arra következtetni, hogy $\exists x$ *Szereti*(*x*, *Fagylalt*). Adjon meg egy általános következtetési szabályt, az **egzisztenciális bevezetést** (**Existential Introduction**), amely megerősíti ezt a következtetést. Gondosan vizsgálja meg a feltételeket, amelyeket ki kell elégíteni a felhasznált változókkal és termekkel.

9.3.

Tételezzük fel, hogy egy tudásbázis csak egy mondatot tartalmaz: $\exists x$ *OlyanMagasMint*(*x*, *Everest*). A következők közül melyek a törvényszerű következményei az Egzisztenciális Példányosítás alkalmazásának?

a. *OlyanMagasMint*(*Everest*, *Everest*)

b. *OlyanMagasMint*(*Kilimandzsáró*, *Everest*)

c. *OlyanMagasMint*(*Kilimandzsáró*, *Everest*) $\wedge$ *OlyanMagasMint*(*BenNevis*, *Everest*)

(a szabály kétszeri alkalmazása után).

9.4.

Adja meg az alábbi atomi mondatpárok legáltalánosabb egyesítőjét, ha egyáltalán létezik ilyen:

a. *P*(*A*, *B*, *B*), *P*(*x*, *y*, *z*)

b. *Q*(*y*, *G*(*A*, *B*)), *Q*(*G*(*x*, *x*), *y*)

c. *Idősebb*(*Apja*(*y*), *y*), *Idősebb*(*Apja*(*x*), *János*)

d. *Ismeri*(*Apja*(*y*), *y*), *Ismeri*(*x*, *x*)

9.5.

Figyelje meg a 9.2. ábrán bemutatott bennfoglalási rácsokat.

a. Hozza létre a rácsot a következő mondatra: *Alkalmaz*(*Anyja*(*János*), *Apja*(*Richárd*)).

b. Hozza létre a rácsot a következő mondatra: *Alkalmaz*(*IBM*, *y*) („Mindenki az IBM-nél dolgozik”). Figyeljen arra, hogy minden olyan lekérdezést felsoroljon, amely egyesíthető a mondattal.

c. Tételezzük fel, hogy a `TÁROL` indexel minden egyes mondatot, minden egyes csomópontra a bennfoglalási rácsban. Magyarázza meg, hogy a `BETÖLT` eljárásnak hogyan kell működnie, amikor néhány mondat ezek közül változókat is tartalmaz; példaként használja a mondatokat az (a) és (b) részfeladatokból, és a következő lekérdezést: *Alkalmaz*(*x*, *Apja*(*x*)).

9.6.

Tételezzük fel, hogy betöltjük egy logikai adatbázisba az amerikai választási címjegyzéket, amely felsorolja a korát, a lakóhelyét, a születési dátumát és az anyja nevét minden személynek, társadalombiztosítási számokat használva azonosító adatként. Így tehát György kora így van megadva: *Kora*(443-65-1282, 56). A következő S1–S5 indexelő sémák melyike tesz lehetővé hatékony megoldást a Q1–Q4 lekérdezésekre (tekintsünk egy normál hátrafelé láncolást)?

- **S1:** egy index minden egyes atomra minden egyes pozícióban.
- **S2:** egy index minden egyes első argumentumra.

Created by XMLmind XSL-FO Converter.

- **S3:** egy index minden egyes predikátumatomra.
- **S4:** egy index a predikátum és az első argumentum minden egyes *kombinációjára*.
- **S5:** egy index a predikátum és a második argumentum minden egyes *kombinációjára*, és egy index minden egyes első argumentumra (nem standard).
- **Q1: Kora**(443-44-4321, *x*)
- **Q2: Lakik**(*x*, *Houston*)
- **Q3: Anyja**(*x*, *y*)
- **Q4: Kora**(*x*, 34) ∧ *Lakik*(*x*, *PiciVárosUSA*)

9.7.

Feltételezhetnénk, hogy az egyesítés során a változó konfliktus problémáját elkerülhetjük úgy, ha minden mondatnál egyszerre átnevezzük az összes változót a tudásbázisban. Mutassa meg, hogy léteznek olyan mondatok, amelyekre ez a megközelítés nem alkalmazható. (*Segítség:* tekintsünk egy olyan mondatot, amelynek egyik része egyesíthető a többivel.)

9.8.

Mutassa meg, hogy hogyan írhatunk meg egy tetszőleges méretű adott 3-SAT problémát, egy elsőrendű határozott klózt és nem több, mint 30 alapténytet felhasználva.

9.9.

Adja meg az alábbi mondatok olyan logikai reprezentációját, amely alkalmas arra, hogy az Általánosított Modus Ponens szabályt alkalmazzuk rájuk:

a. A lovak, a tehenek és a malacok emlősök.

b. Egy ló leszármazottja is ló.

c. Kékszakáll egy ló.

d. Kékszakáll Charlie szülője.

e. A leszármazott és a szülő inverz relációk.

f. Minden emlősnek van szülője.

9.10.

Ebben a problémában a 9.4. feladatban bemutatott mondatokat fogjuk használni. Válaszolja meg a következő kérdéseket hátrafelé láncolást alkalmazva!

a. Rajzolja fel egy kimerítő hátrafelé láncolás algoritmus bizonyítási fáját a következő mondat igazolásához: ∃ *h Ló*(*h*)

b. Mi figyelhető meg erről a tárgyterületről?

c. Hány megoldás származtatható le *h*-ra a mondatokból?

d. Tudna-e olyan módszert mondani, amellyel mindet megkaphatjuk? (*Segítség:* érdemes megnézni (Smith és társai, 1986).)

9.11.

Egy népszerű találós kérdés gyerekeknek a következő: „Nincs se bátyám, se nővérem, mégis annak az embernek az apja az én apám fia.” Használja fel a család tárgykör szabályait (lásd 7. fejezet), és mutassa meg, hogy ki is az említett ember. Használhatja bármelyik, a fejezetben bemutatott következtetési módszert.

Created by XMLmind XSL-FO Converter.

9.12.

Kövesse nyomon a 9.6. ábrán látható hátrafelé láncolási algoritmus végrehajtását, amikor azt a bűntény probléma megoldására alkalmazzuk. Mutassa be azt az érték-szekvenciát, amelyet a *célok* változó vesz fel, és rendezze egy fa formába.

9.13.

A következő Prolog kód egy P predikátumot határoz meg:

```
P(X,[XC|Y])

P(X,[Y|Z]):-P(X,Z)
```

a. Mutasson be bizonyítási fákat és megoldásokat ezekre a lekérdezésekre: P(A[1,2,3])és P(2,[1,A,3])

b. Milyen standard operációs listát reprezentál a P?

9.14.

Ebben a feladatban megvizsgáljuk egy sorbarendezés megvalósítását a Prologban.

a. Írjon Prolog-klózokat, amelyek definiálják a rendezés(L) predikátumot, amely akkor és csakis akkor igaz, ha az L lista emelkedő sorrendben van rendezve.

b. Írjon egy Prolog-definíciót erre a predikátumra: perm(L,M), amely akkor és csakis akkor igaz, ha az L a permutációja az M-nek.

c. Definiálja a rendez(L, M)-et (az M egy válogatott verziója az L-nek) a perm és a rendezés használatával.

d. Futtassa a rendez predikátumot minél hosszabb listákon, amíg el nem veszíti a türelmét. Mekkora a programjának az időkomplexitása?

e. Írjon egy gyorsabb rendező algoritmust Prologban, mint amilyen például a beszúrásos válogatás vagy a gyorsválogatás (quicksort).

9.15.

Ebben a feladatban megvizsgáljuk az újraíró szabályok rekurzív alkalmazását a logikai programozás felhasználásával. Egy újraíró szabály (vagy **demodulátor** az OTTER terminológiában) egy egyenlet egy megadott iránnyal. Például az $x + 0 \rightarrow x$ újraíró szabály azt sugallja, hogy minden, az $x + 0$-hoz illeszkedő kifejezést fel kell cserélni x-re. Az újraíró szabályok alkalmazása központi része az egyenletkövetkeztető rendszereknek. Ezt a predikátumot fogjuk használni: újraír(X, Y) az újraíró szabályok reprezentálására. Például a korábbi újraíró szabályt így írtuk: újraír(X+0, X). Néhány term *primitív,* és nem lehet tovább egyszerűsíteni; így tehát ezt fogjuk írni: primitív(0) , ami azt jelenti, hogy a 0 egy primitív term.

a. Írjon egy definíciót az egyszerűsít(X,Y) predikátumra, amely akkor igaz, amikor az Y az egyszerűsített változata X-nek – tehát amikor már több újraíró szabály nem alkalmazható az Y egyik részkifejezésére sem.

b. Írjon egy szabálygyűjteményt az aritmetikai operátorokat tartalmazó kifejezések egyszerűsítésére, és alkalmazza az egyszerűsítési algoritmusát néhány minta kifejezésre.

c. Írjon egy újraíró szabálygyűjteményt a szimbolikus differenciálásra, és használja az egyszerűsítési szabályaival együtt, hogy differenciáljon, és egyszerűsítsen aritmetikai kifejezéseket tartalmazó kifejezéseket, beleértve a hatványozást.

9.16.

Ebben a feladatban megvizsgáljuk a keresési algoritmusok alkalmazását a Prologban. Tételezzük fel, hogy a következő(X, Y) akkor igaz, ha az Y állapot az X állapotot követi; és ha a cél(X) akkor igaz, amikor az X a célállapot. Írjon egy definíciót a megold(X,P)-re, ami azt jelenti, hogy a P az útvonal (az állapotok listája), amely az X-szel kezdődik, a célállapotban végződik, és szabályos lépésekből áll, amelyeket a következő határoz

Created by XMLmind XSL-FO Converter.

meg. Valószínűleg a mélységi keresés a legkönnyebb módja annak, hogy ezt végrehajtsa. Mennyire lenne könnyű egy heurisztikus keresési vezérlés hozzáadása?

9.17.

Hogyan alkalmazható a rezolúció annak megmutatásra, hogy egy mondat

a. érvényes-e?

b. kielégíthetetlen-e?

9.18.

Abból a mondatból, hogy „a lovak állatok” következik-e, hogy „egy ló feje egy állat feje”. Demonstrálja, hogy ez a következtetés érvényes, ha végrehajtjuk a következő lépéseket:

a. Fordítsa le a premisszát és a következményt az elsőrendű logika nyelvére. Használjon három predikátumot: *Feje*(*h*, *x*), *Ló*(*x*) és *Állat*(*x*)!

b. Negálja a következményt, és konvertálja a premisszát és a negált következményt konjunktív normál formára!

c. Mutassa meg a rezolúció felhasználásával, hogy a következmény valóban következik a premisszából!

9.19.

Tekintsük a következő két elsőrendű logikai mondatot:

(A): $\forall x\ \exists y\ (x \geq y)$

(B): $\exists y\ \forall x\ (x \geq y)$

a. Legyenek a változók lehetséges értékei a természetes számok 0, 1, 2, …, ∞, és jelentse a „≥” predikátum azt, hogy „nagyobb vagy egyenlő”. Fordítsa le ezeket a mondatokat magyarra ebben az interpretációban!

b. Igaz-e (A) ebben az interpretációban?

c. Igaz-e (B) ebben az interpretációban?

d. Maga után vonja-e az (A) mondat a (B) mondatot?

e. Maga után vonja-e a (B) mondat az (A) mondatot?

f. Próbálja meg a rezolúció felhasználásával bizonyítani, hogy (A) következik (B)-ből! Tegye meg ezt akkor is, ha azt gondolja, hogy (B) nem vonja maga után (A)-t; folytassa az eljárást addig, amíg a bizonyítás már nem folytatható. Tüntesse fel az egyesítő helyettesítéseket minden rezolúciós lépésnél! Ha a bizonyítás sikertelen, magyarázza meg, hogy hol, hogyan és miért akad meg!

g. Most próbálja meg igazolni azt, hogy (B) következik (A)-ból!

9.20.

A rezolúció változókkal való lekérdezésekre létrehozhat nem konstruktív bizonyításokat, ezért egy speciális eljárást kell bevezetnünk a definit válaszok kinyerésére. Magyarázza meg, hogy ez a probléma miért nem merül fel csak definit klózokat tartalmazó tudásbázisokkal!

9.21.

Ebben a fejezetben azt mondtuk, hogy a rezolúció nem használható egy mondathalmaz összes logikai következményének a generálására. Van olyan algoritmus, ami képes erre?

Created by XMLmind XSL-FO Converter.

10. fejezet - Tudásbázis reprezentáció

Ebben a fejezetben megmutatjuk, hogy az elsőrendű logikát hogyan kell használni a valódi világ olyan legfontosabb aspektusainak ábrázolására, mint a cselekvés, a tér, az idő, a mentális események és a bevásárlás.

Az utolsó három fejezet a tudásalapú ágens technológiáját mutatta be, avagy az ítélet- és az elsőrendű logika szintaxisát, szemantikáját, bizonyításelméletét és az ilyen logikákat használó ágens implementációját. Ebben a fejezetben azzal a kérdéssel foglalkozunk, hogy milyen *tartalmat* tegyünk az ágens tudásbázisába, azaz, hogy a világról szóló tényeket hogyan reprezentáljuk.

A 10.1. alfejezet bevezeti az általános ontológia gondolatát, amely világban mindent a kategóriák hierarchiájába szervez. A 10.2. alfejezet az objektumok és a szubsztanciák alapvető kategóriáival foglalkozik. A 10.3. alfejezet a cselekvések reprezentációját elemzi, amelyek a tudásalapú ágens megtervezése szempontjából központi fontosságúak, továbbá bemutatja a tér-idő darabkák, az **esemény**ek (**event**s) általánosabb fogalmát. A 10.4. alfejezet a hiedelmekkel foglalkozik, a 10.5. alfejezet pedig az egészet egybefogja az internetes bevásárlás kontextusában. A 10.6. és a 10.7. alfejezet a bizonytalan és a változó ismeretekkel dolgozó specializált következtető rendszerekkel foglalkozik.

1. Ontológiaszervezés

„Játék” tárgyterületeken a reprezentáció megválasztása nem annyira lényeges. Könnyű egy konzisztens fogalomkészletet kialakítani. Az olyan komplex területeken azonban, mint például az interneten történő bevásárlás vagy egy robot vezérlése változó fizikai környezetben, általánosabb és rugalmasabb reprezentációra van szükség. Ebben a fejezetben megmutatjuk, hogy ilyen reprezentációkat hogyan lehet létesíteni, olyan általános fogalmakra fókuszálva, mint a *Cselekvés*, az *Idő*, a *FizikaiObjektum* és a *Hiedelem*, amelyek számos tárgyterületen fordulnak elő. Ezen absztrakt fogalmak reprezentálását néha **ontológiaszervezés**nek (**ontological engineering**) nevezik. Az ontológiaszervezés kapcsolatban áll a 8.4. alfejezetben leírt tudásszervezés folyamatával, de a hatása szélesebb körű.

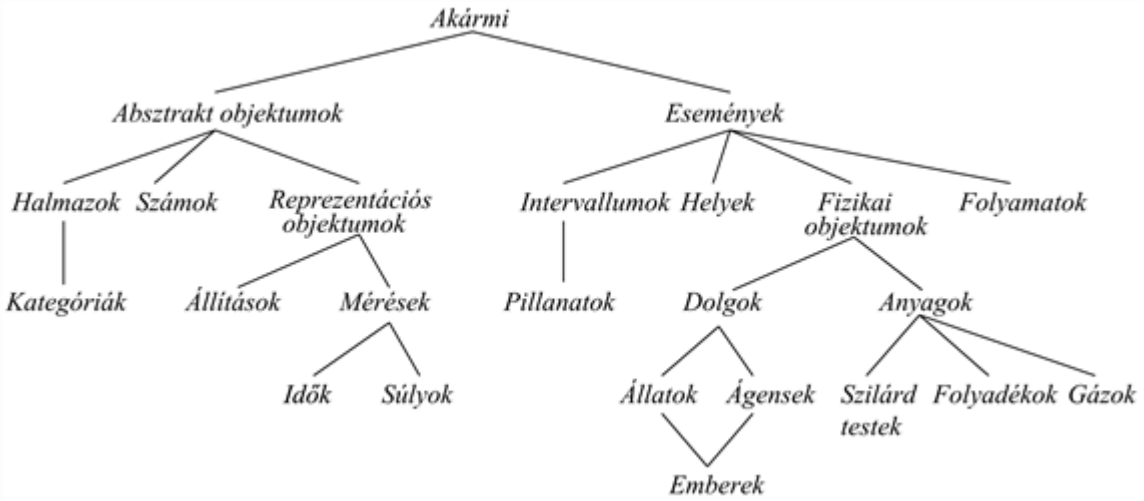
A világról *mindent* reprezentálni ijesztő perspektíva. Valójában természetesen nem fogjuk mindennek a komplex leírását megadni – ez még egy 1000 oldalas könyvnek is sok lenne –, de megjelöljük világosan annak a helyét, ahova egy tetszőleges területről szóló új tudás beilleszthető. Így például definiálni fogjuk, hogy mit is jelent fizikai objektumnak lenni, és a különböző objektumok részletei – robotok, tv-készülékek, könyvek vagy bármi más – később illeszthetők ebbe a keretbe. A fogalmak egy általános keretét **felső ontológiá**nak (**upper ontology**) nevezzük, mert az az általános konvenció, hogy a gráfszerű ábrázolásnál az általánosabb fogalmak felül helyezkednek el, a konkrétabb fogalmak pedig alul (lásd 10.1. ábra).

Mielőtt az ontológiát tovább tekintenénk, egy fontos figyelmeztetést kell tennünk. A tudás tartalmának és szervezésének a megvitatására elsőrendű logikát választottunk. A valódi világ bizonyos aspektusait ezen a nyelven nehéz lesz kifejezni. Egy alapvető tulajdonságot kénytelenek leszünk kihagyni: azt, hogy minden általánosítás alól léteznek kivételek, vagy hogy a részletesebb információ hiányában csupán az alapértelmezést tekinthetjük, vagy pedig hogy minden általánosítás csak egy bizonyos mértékig érvényes. Bár a „paradicsom piros” hasznos szabály, van zöld, sárga és narancssárga paradicsom is. Hasonló kivételeket ebben a részfejezetben majdnem mindegyik állítás esetén lehet találni. A kivételek és a bizonytalanság kezelése nagyon fontos képességek, azonban egy általános ontológia megértéséhez viszonyítva ortogonálisak. Ezért a kivételek és az alapértelmezés megvitatását a 10.6. alfejezetre, a bizonytalan információ sokkal általánosabb témáját pedig a 13. fejezetre hagyjuk.

Milyen haszna van egy felső ontológiának? Gondoljunk vissza a 8.4. alfejezet áramköri ontológiájára. Megkonstruálásánál igen sok egyszerűsítő feltételezéssel éltünk. Így például az időt teljes egészében ki is hagytuk. A jelek rögzítettek voltak, és a jeltovábbterjedésről nem is esett szó. Az áramkörök struktúrája változatlan maradt. Az általánosság felé úgy tudnánk lépni, hogy a jeleket konkrét időpillanatokban definiálnánk és foglalkoznánk a vezetékek hosszával, valamint a vezetékekben és a berendezésekben fellépő jelterjedési késleltetésekkel. Ezzel szimulálni tudnánk az áramkör időzítési tulajdonságait; és valóban az áramkörtervezők gyakran folyamodnak is az ilyen elemzéshez. A kapuk érdekesebb osztályait is be lehetne vezetni, például leírva a technológiát (TTL, MOS, CMOS stb.) és a bemeneti/kimeneti specifikációkat is. Ha a megbízhatósággal vagy diagnózissal szeretnénk foglalkozni, engedélyezni kellene, hogy az áramkör struktúrája spontán módon megváltozhasson. A szórt kapacitások figyelembevételéhez a tisztán topológiai reprezentációval fel kellene hagyni, és át kellene térni a geometriai tulajdonságok valósághűbb leírására.

Created by XMLmind XSL-FO Converter.

10.1. ábra - A világ ontológiájának felsőbb szintjei, amelyek a fejezetben később tárgyalt témákat mutatják. Minden él azt jelzi, hogy az alsó fogalom a felső fogalom egy specializálódása.



Hasonló megfontolások érvényesek, ha a wumpus világát nézzük. Bár foglalkozunk az idővel, ennek struktúrája igen egyszerű. A dolgok csak az ágens cselekvésével egy időben történnek, és minden változás azonnali jellegű. Egy általánosabb, a valódi világhoz jobban illeszkedő ontológia engedélyezné például, hogy az egyidejű változásoknak legyen időbeli kiterjedésük. Arra, hogy egy konkrét négyzetben csapda van, a *Csapda* konstanst használtuk, mert minden csapda azonos volt. Engedélyezni lehetne a csapdák több fajtáját is, a csapdák osztályához tartozó, eltérő tulajdonságú példányokat használva. Hasonló módon több állatfajtát is be tudnánk vezetni a wumpuson kívül. Elképzelhető, hogy a rendelkezésre álló érzékelésekből az állat konkrét fajtáját nehéz lenne megállapítani. Az ágens megsegítésére a wumpus világ biológiai taxonómiáját fel lehetne állítani, hogy gyenge nyomravezető jelekből képes legyen megjósolni az állat viselkedését.

Az ilyen módosítások révén minden speciális rendeltetésű ontológiában lehetséges egy nagyobb általánosság felé elmozdulni. A nyilvánvaló kérdés ilyenkor az, hogy konvergálnak-e ezek a módosítások egy általános célú ontológiához? Évszázadokon át folyó filozófiai és számítási kutatások után a válasz az, hogy „lehetséges”. Ebben a fejezetben egy lehetséges változatot mutatunk be, amely az évszázadok során született gondolatok szintetizálását képviseli. Az általános célú ontológiának két olyan fő jellemzője van, ami azt a speciális rendeltetésű ontológiák sokaságától megkülönbözteti:

- Az általános ontológiát (a tárgytartományra vonatkozó axiómák hozzáadásával) többé-kevésbé minden speciális rendeltetésű tárgytartományban kell tudnunk alkalmazni. Lehetőség szerint tehát egyetlen reprezentációs problémával sem lehet ravaszkodni vagy azt a szőnyeg alá söpörni.

- Minden kellően igényes tárgytartományban a tudás egyes részeit *egyesíteni* kell, hiszen a következtetés és a problémamegoldás egyszerre több területet is igényelhet. Egy robot áramkörjavító rendszer esetén például az áramkörökről a villamos összeköttetések és a fizikai elrendezés szempontjából kell tudnunk következtetni, de az idő az áramkör-időzítési analízis és a munkaköltségek szempontjából egyaránt lényeges. Az időt leíró állításokat tehát össze kell tudnunk kombinálni a fizikai elrendezést leíró állításokkal, és ezeknek az állításoknak egyformán jól kell működniük nanomásodpercekre és percekre is, valamint nanométerekre és méterekre is.

Miután az általános ontológiát megalkottuk, felhasználjuk az internetes bevásárlás tárgytartományra. Ez a tárgytartomány több mint alkalmas arra, hogy az ontológiánkkal kísérletezzünk. Sok helyet hagy az olvasó részére is, hogy a reprezentációba a saját kreativitását is bevihesse. Gondoljunk például arra, hogy egy interneten bevásárló ágensnek rengeteg témát és szerzőt kell ismernie, hogy az Amazon.com-tól könyveket vásároljon, élelmiszerek egész választékáról kell tudnia, hogy a Peapod.com-nál élelmiszert vásároljon, és mindenről, amit egy bolhapiacon találni lehet ismeretekkel kell rendelkeznie, hogy az Ebay.com[1]-nál az alkalmi jó üzletekre vadásszon.

[1] Az olvasó elnézését kérjük, ha rajtunk kívül eső okok miatt ezen online boltok valamelyike a könyv olvasásának pillanatában már nem működik.

Created by XMLmind XSL-FO Converter.

2. Kategóriák és objektumok

Fontos

Az objektumok **kategóriá**kba (**categories**) való szervezése a tudásreprezentáció szempontjából létfontosságú. Bár a világgal való kölcsönhatás az egyedi objektumok szintjén történik, *a következtetések zöme a kategóriák szintjén valósul meg*. A bevásárló célja egy kosárlabda megvásárlása és nem egy konkrét kosárlabdapéldánynak, mondjuk KL_9-nek a megvétele. Ha az objektumok osztályozását elvégeztük, kategóriák segítségével megjósolhatjuk az objektumok tulajdonságait. Érzékelő jelek alapján következtetünk bizonyos objektumok jelenlétére, az érzékelt tulajdonságokból következtetünk, hogy az objektumok milyen kategóriákhoz tartoznak, majd a kategóriákra vonatkozó ismereteket felhasználhatjuk, hogy az objektumokra vonatkozóan előrejelzéseket hozzunk. Így például a zöld, foltos felület, a tekintélyes nagyság és az ovális vagy gömbszerű alak alapján görögdinnyére következtethetünk; a görögdinnyére vonatkozó ismereteink alapján pedig arra, hogy az felhasználható lenne gyümölcssaláta készítéséhez.

Az elsőrendű logikában a kategóriák reprezentálására két alapvető választásunk lehet: a predikátumok és az objektumok. Használhatunk egy *KosárLabda*(*l*) predikátumszimbólumot, vagy a kategóriát *KosárLabda* objektumként **reifikál**hatjuk (**reification**). Mondhatjuk akkor, hogy *Eleme*(*l*, *KosárLabda*) (rövidítve $l \in KosárLabda$) annak a kifejezésére, hogy *l* a kosárlabda-kategória eleme. Azt mondjuk, hogy *Részhalmaza*(*KosárLabda*, *Labda*) (rövidítve $KosárLabda \subset Labda$) annak a kifejezésére, hogy a *KosárLabda* a *Labda* egy alkategóriája vagy részhalmaza. Egy kategóriát tekinthetünk az elemeiből álló halmaznak, de egy sokkal bonyolultabb objektumnak is képzelhetjük, olyannak, amire az *Eleme* és a *Részhalmaza* relációk definiáltak.

A kategóriák **öröklődés** (**inheritance**) révén szolgálják a tudásbázis szervezését és egyszerűsítését. Ha kijelentjük, hogy az *Élelem* kategória minden egyes példánya ehető, és feltételezzük, hogy a *Gyümölcs Élelem*, az *Alma* viszont a *Gyümölcs* kategória egy alosztálya, akkor tudjuk, hogy minden alma ehető. Azt mondjuk, hogy az egyes almák az ehetőségi tulajdonságukat **örökölt**ék (**inherit**), jelen esetben az *Élelem* kategóriához való tartozás révén.

Az alosztály-relációk a kategóriákat **taxonómiá**ba vagy **taxonomikus hierarchiá**ba (**taxonomy**, **taxonomic hierarchy**) szervezik. A taxonómiákat tudományos területeken évszázadok óta alkalmazták. Így például a rendszerező biológia az összes élő és kihalt faj taxonómiáját igyekszik megadni, a könyvtártudomány az összes tudományos területet átfogó taxonómiáját – a Dewey Decimális rendszert – alakította ki, az adóhatóságok és más kormányzati szervek a foglalkozások és termékek kiterjedt taxonómiáját hozták létre. A taxonómiák – ahogy a további elemzéseinkben látni fogjuk – fontos aspektusai a józan ész tudásnak is.

A kategóriákról az elsőrendű logikában könnyű állításokat megfogalmazni az objektumok és a kategóriák egymáshoz való rendelésével vagy a kategóriához tartozó egyedek szerinti kvantifikálás révén:

- Egy objektum egy kategória egyede, például:

$KL_9 \in KosárLabda$

- Egy kategória egy másik kategória alosztálya, például:

$KosárLabda \subset Labda$

- Egy kategória összes egyede egy bizonyos tulajdonsággal rendelkezik, például:

$\forall x \in KosárLabda \Rightarrow Gömbölyű(x)$

- Egy kategória egyedeit bizonyos tulajdonságaik alapján fel lehet ismerni, például:

$Narancsszínű(x) \land Gömbölyű(x) \land Átmérő(x) = 24 \land x \in Labda \Rightarrow x \in KosárLabda$

- Egy kategóriának önmagában bizonyos tulajdonságai vannak, például:

$Kutya \in HonosítottFajta$

Created by XMLmind XSL-FO Converter.

Jegyezzük meg, hogy mivel a *Kutya* egy kategória és a *HonosítottFajta* kategóriának egy egyede, így a *HonosítottFajta* a kategóriák kategóriája. A kategóriák kategóriáinak a kategóriáiról is lehetne beszélni, de ennek kevés a haszna.

Bár az alosztály- és az egyedrelációk a kategóriák szempontjából a legfontosabbak, olyan kategóriák közötti relációkat is ki szeretnénk fejezni, amelyek egymásnak nem alosztályai. Ha például azt mondjuk, hogy a *Hím* és a *Nőstény* az *Állat* alosztályai, ezzel nem mondtuk azt, hogy egy hím nem lehet nőstény. Azt mondjuk, hogy két vagy több kategória **diszjunkt** (**disjoint**), ha közös egyedei nincsenek. Ha tudjuk azt is, hogy a hímek és a nőstények kölcsönösen kizárják egymást, attól még nem tudjuk, hogy egy állatnak, amely nem hím, nősténynek kell lennie, hacsak nem mondjuk ki, hogy a hímek és a nőstények az állatok **kimerítő felosztás**át (**exhaustive decomposition**) képezik. A diszjunkt kimerítő felosztás a **partíció** (**partition**). Ezt a három fogalmat az alábbi példák illusztrálják:

$Diszjunkt(\{Állat, Zöldség\})$

$KimerítőFelosztás(\{Amerikai, Kanadai, Mexikói\}, ÉszakAmerikai)$

$Partíció(\{Hím, Nőstény\}, Állat)$

(Jegyezzük meg, hogy az *ÉszakAmerikai KimerítőFelosztás*-a nem *Partíció*, mert vannak kettős állampolgárságú személyek is.) E három predikátum definíciója az alábbi:

$$Diszjunkt(s) \Leftrightarrow (\forall c_1, c_2\ c_1 \in s \wedge c_2 \in s \wedge c_1 \neq c_2 \Rightarrow Metszet(c_1, c_2) = \{\})$$

$$KimerítőFelosztás(s, c) \Leftrightarrow (\forall i\ i \in c \Leftrightarrow \exists c_2\ c_2 \in s \wedge i \in c_2)$$

$$Partíció(s, c) \Leftrightarrow Diszjunkt(s) \wedge KimerítőFelosztás(s, c)$$

Kategóriákat úgy is *definiálhatunk,* hogy megadjuk a tagságuk elégséges és szükséges feltételeit. Például egy agglegény egy felnőtt, nem házas férfi:

$$x \in Agglegény \Leftrightarrow NemHázas(x) \wedge x \in Felnőtt \wedge x \in Férfi$$

Ahogy erről a természetes fajtákról szóló rövid kitérőben szó lesz, a kategóriák szigorú logikai definíciója nem mindig lehetséges, és nem is mindig szükséges.

2.1. Fizikai összetétel

Az a gondolat, hogy egy objektum része lehet egy másik objektumnak, nem újkeletű. Valakinek az orra része a fejének, Románia Európa része, ez a fejezet a könyvünk egy része. Hogy megmondhassuk, hogy egy dolog része egy másiknak, egy általános *Része* relációt fogunk használni. Az objektumokat tehát *Része* hierarchiába lehet szervezni, ami hasonlít a *Részhalmaza* hierarchiára:

$Része(Bukarest, Románia)$

$Része(Románia, KeletEurópa)$

$Része(KeletEurópa, Európa)$

$Része(Európa, Föld)$

A *Része* reláció tranzitív és reflexív, azaz:

$$Része(x, y) \wedge Része(y, z) \Rightarrow Része(x, z)$$

$$Része(x, x)$$

Ebből adódóan kikövetkeztethető, hogy *Része*(*Bukarest, Föld*)

Az **összetett objektum**ok (**composite object**) kategóriáit gyakran ezen objektumok részei között értelmezett strukturális relációkkal jellemezzük. Például egy kétlábúnak pontosan két lába van, ami egy testhez van rögzítve:

$$KétLábú(a) \Rightarrow \exists l_1, l_2, b\ Láb(l_1) \wedge Láb(l_2) \wedge Test(b) \wedge Része(l_1, a) \wedge Része(l_2, a)$$

Created by XMLmind XSL-FO Converter.

$\wedge$ *Része*(b, a) $\wedge$ *Rögzített*(l_1, b) $\wedge$ *Rögzített*(l_2, b) $\wedge$ $l_1 \neq l_2$

$\wedge$ [$\forall l_3$ *Láb*(l_3) $\wedge$ *Része*(l_3, a) $\Rightarrow$ ($l_3 = l_1 \vee l_3 = l_2$)

A „pontosan kettő” jelölés egy kicsit fura. Kénytelenek vagyunk kijelenteni, hogy két láb van, és ezek nem azonosak, és ha valaki egy harmadik lábbal jön elő, annak azonosnak kell lennie a kettő valamelyikével. A 10.6. alfejezetben látni fogjuk, hogy a leíró logikának nevezett formalizmus a „pontosan kettő” típusú korlátozásokat egyszerűbben fejezi ki.

A kategóriákra vonatkozó *Partíció* reláció mintájára egy *RészPartíció* relációt definiálhatunk (lásd 10.6. feladat). Egy objektum a *RészPartíció*-jában felsorolt részeiből áll, és bizonyos tulajdonságaira ezekből a részekből lehet következtetni. Így például egy összetett objektum tömege a részeihez tartozó tömegek összege. Jegyezzük meg, hogy ez a kategóriákra nem vonatkozik: egy kategóriának nincs tömege annak ellenére, hogy az elemeinek lehet tömege.

Hasznos olyan összetett objektumokat is definiálni, amelyeknek meghatározott részei vannak, de konkrét struktúrájuk nincsen. Előfordulhat, hogy azt szeretnénk mondani, hogy: „A zacskóban 3 kg alma van.” Kísértést érezhetnénk, hogy a zacskóban-alma *halmazhoz* súlyt rendeljük, ez azonban hiba lenne, mert egy halmaz absztrakt matematikai fogalom, aminek vannak elemei, de súlya nincs. Ahelyett egy új fogalomra van szükségünk, amit **köteg**nek (**bunch**) fogunk nevezni. Ha az almák például az $Alma_1$, az $Alma_2$ és az $Alma_3$, akkor a:

Köteg({$Alma_1$, $Alma_2$, $Alma_3$})

a három almából (mint részből, de nem mint elemből) álló összetett objektumot jelöli. A köteget egy közönséges, bár nem strukturált objektumként használhatjuk. Jegyezzük meg, hogy *Köteg*({x}) = x.Továbbá, hogy a *Köteg*(*Alma*) az összes almából álló összetett objektum, amit az *Alma* kategóriával összetéveszteni nem szabad.

A *Köteg*-et a *Része* relációval tudjuk definiálni. Az s minden eleme, eleme a *Köteg*({s})-nek is:

$\forall x \quad x \in s \Rightarrow$ *Része*(x, *Köteg*(s))

Továbbá, a *Köteg*(s) a legkisebb objektum, amely ezt a feltételt teljesíti. Más szóval a *Köteg*(s) részének kell lennie minden olyan objektumnak, amely az s összes elemét részeként tartalmazza:

$\forall y$ [$\forall x$ $x \in s \Rightarrow$ *Része*(x, y)] $\Rightarrow$ *Része*(*Köteg*(s), y)

Ezek az axiómák a **logikai minimalizálás**nak (**logical minimization**) nevezett általános módszer egy példáját jelentik. A logikai minimalizálás azt jelenti, hogy egy objektumot bizonyos feltételeket kielégítő legkisebb objektumnak definiálunk.

2.2. Mértékek

A világnak mind a tudományos, mind a józan ész elméleteiben az objektumoknak magassága, tömege, ára van és így tovább. Az ezekhez a tulajdonságokhoz előírt értékek a **mérték**ek (**measure**s). Közönséges, kvantitatív mértékeket könnyű reprezentálni. Képzeljük el, hogy az univerzum absztrakt „mértékobjektumokat” tartalmaz, mint például a *hossz*, ami az itt látható vonalszegmens hossza:



Nevezhetjük ezt a hosszat 1,5 hüvelyknek vagy 3,81 cm-nek. Ugyanannak a hossznak tehát a nyelvünkben több, különböző neve is lehet. Logikailag ez úgy lehetséges, hogy egy **egységfüggvény**t (**unit function**) egy számmal kombinálunk. (Egy alternatív sémával a 10.8. feladatban foglalkozunk.) Ha L_1 a vonalszegmens neve, akkor azt írhatjuk, hogy:

Hossz(L_1) = *Hüvelyk*(1,5) = *Centiméter*(3,81)

Az egységek közötti konverziót olyan állításokkal lehet megoldani, amelyek az egyik egység többszörösét a másik egységgel teszik azonossá:

Centiméter(2,54 $\times$ l) = *Hüvelyk*(l)

Created by XMLmind XSL-FO Converter.

Hasonló axiómákat a fontokra és kilogrammokra, a másodpercekre és a napokra, a dollárokra és a centekre is felírhatunk. A mértékekkel az objektumokat az alábbi módon tudjuk leírni:

$$Átmérő(KosárLabda_{12}) = Hüvelyk(9,5)$$

$$Ár(KosárLabda_{12}) = \$(19)$$

$$d \in Napok \Rightarrow Tartam(d) = Óra(24)$$

Jegyezzük meg, hogy a $(1) nem egy egydolláros bankó! Az egydolláros bankóból lehet kettő, de a $(1) nevű objektumból csak egy van. Jegyezzük meg azt is, hogy míg a *Hüvelyk*(0) és *Centiméter*(0) ugyanarra a zérushosszra hivatkozik, más zérusmértékekkel, mint például a *Másodperc*(0) nem azonosak.

Egyszerű, kvantitatív mértékeket reprezentálni könnyű. Más mértékek több problémát okoznak, mert nincs hozzájuk elfogadott értékskála. A gyakorlat nehéz, a desszert finom és a vers szép, azonban e kvalitásokhoz nehéz számokat rendelni. Valaki megkísérelhetné az ilyen tulajdonságokat teljesen elutasítani, mint olyanokat, melyeknek a logikai következtetés szempontjából nincs hasznuk, vagy – ami még rosszabb – megkísérelhetne a szépségre egy numerikus skálát rákényszeríteni. Ez súlyos hiba lenne, mert ilyen lépésre nincs is szükség. A mértékek legfontosabb tulajdonsága nem az, hogy valamilyen konkrét numerikus értékkel rendelkeznek, hanem, hogy *rendezettek*.

Annak ellenére, hogy a mértékek nem számok, összehasonlításukra a > rendező szimbólumot fogjuk használni. Így például hihetjük azt, hogy a Norvig írta feladatok nehezebbek, mint azok, amiket Russell írt, és hogy egy nehezebb feladatnál kevesebb pontot lehet elérni:

$$e_1 \in Feladatok \land e_2 \in Feladatok \land Írta(Norvig, e_1) \land Írta(Russell, e_1) \Rightarrow$$

$$Nehézség(e_1) > Nehézség(e_2)$$

$$e_1 \in Feladatok \land e_2 \in Feladatok \land Nehézség(e_1) \land Nehézség(e_2) \Rightarrow$$

$$VárhatóEredmény(e_1) < VárhatóEredmény(e_2)$$

Ez elég is ahhoz, hogy valaki eldönthesse, melyik feladattal érdemes foglalkoznia, bármennyire nem használtunk semmiféle numerikus értéket a nehézség kifejezésére. (Azért azt meg kell tudnunk állapítani, hogy melyik feladatot ki írta.) A mértékek közötti efféle monoton reláció az alapja a **kvalitatív fizika** (**qualitative physics**) területének. Ez az MI egy részterülete, amely azt vizsgálja, hogy hogyan lehetne a fizikai rendszerekre következtetni anélkül, hogy a részletes egyenletekbe és a numerikus szimulációkba belebonyolódnánk. A kvalitatív fizikával a történeti megjegyzésekben foglalkozunk.

Fontos

Természetes fajták

Egyes kategóriáknak szigorú definíciói vannak. Egy objektum akkor és csak akkor háromszög, ha egy háromoldalú sokszög. A valódi világban viszont a kategóriák többsége **természetes fajtájú** (**natural kind**) anélkül, hogy bármilyen letisztult definíciója lenne. Tudjuk például, hogy a paradicsomok mélyvörös színűek és nagyjából gömbszerűek szoktak lenni, a tetejükön, ahol a száruk csatlakozott, egy kis bemélyedéssel rendelkeznek, nagyjából 5–8 cm átmérőjűek, és vékony, de erős bőrrel, belül hússal, magvakkal és lével rendelkeznek. Azt is tudjuk azonban, hogy eltérések is vannak. Egyes paradicsomok narancsszínűek, az éretlen paradicsom zöld, egyesek kisebbek, illetve nagyobbak, mint az átlag, a miniparadicsomok egységesen kisméretűek. Ahelyett hogy a paradicsomok tökéletes definíciójára törekednénk, inkább egy tulajdonsághalmazzal rendelkezünk, amely arra szolgál, hogy azonosítsunk bizonyos objektumokat, amelyek nyilvánvalóan tipikus paradicsomok, más objektumok esetén azonban lehet, hogy a felismerés csődöt mond. (Lehet egy paradicsom szőrös, mint egy őszibarack?)

Egy logikai ágens részére ez problémát jelent. Az ágens nem lehet biztos abban, hogy az általa érzékelt objektum valóban egy paradicsom, és ha mégis biztos lenne benne, nem tudná eldönteni, hogy a tipikus paradicsom tulajdonságai közül az érzékelt objektum melyekkel rendelkezik. Ez a probléma egyenes következménye annak, hogy az ágens egy hozzáférhetetlen környezetben működik.

Created by XMLmind XSL-FO Converter.

Egy hasznos megközelítés szétválasztani mindazt, ami a kategória minden egyedére igaz, attól, ami csupán a kategória tipikus egyedeire igaz. A *Paradicsom* kategória mellett szükség van még a *Tipikus*(*Paradicsom*) kategóriára. A *Tipikus* függvény egy kategóriát egy olyan alosztályra képez le, amely csak tipikus egyedekből áll:

$$Tipikus(c) \subseteq c$$

A természetes fajtákra vonatkozó tudás többsége tulajdonképpen tipikus egyedekről szól:

$$x \in Tipikus(Paradicsom) \Rightarrow Piros(x) \wedge Gömbölyű(x)$$

Ily módon képesek vagyunk a kategóriákra vonatkozó fontos tényeket feljegyezni anélkül, hogy pontos definíciókat kellene megadnunk.

Azzal, hogy a természetes kategóriák többsége esetén az egzakt definíció megalkotása nehézségekbe ütközik, részletesen Wittgenstein foglalkozott a *Philosophical Investigations* c. művében (Wittgenstein, 1953). A *játékok* példáját használta fel, hogy kimutassa, a kategória egyedei inkább egyfajta „családi hasonlóságban” és nem valami szükséges és elégséges jellegzetességekben osztoznak.

Quine szintén támadta a szigorú definíció hasznosságát (Quine, 1953). Azt mutatta ki, hogy az „agglegény” definíciója, mint egy nem házas felnőtt férfi is gyanús. Valaki megkérdőjelezhetné például, hogy „A Pápa agglegény” értelmes állítás-e. Bár nem kifejezetten *hamis*, ez a fogalomhasználat mégis *szerencsétlen*, mert az olvasónak nem szándékolt következtetések levonását teszi lehetővé. A feszültséget fel lehetne oldani megkülönböztetést téve a tudásreprezentációban való belső használatra alkalmas logikai definíció és a helyes nyelvészeti használat jobban árnyalt kritériumai között. A másik megkapható az elsőből a levont következtetések szűrésével. Az is lehetséges, hogy a nyelvészeti használat kudarcai visszacsatolásként hatnak a belső definíció módosítása érdekében, fölöslegessé téve a szűrést.

2.3. Szubsztanciák és objektumok

A világot lehetne úgy szemlélni, hogy primitív objektumokból (részecskékből) és az azokból felépülő összetett objektumokból áll. Az olyan nagy objektumok, mint például az almák vagy az autók szintjén végzett következtetéssel meg tudunk birkózni a primitív objektumok óriási számából eredő bonyolultsággal. A valóság tekintélyes része azonban az **egyedesítés** (**individuation**) – az elkülönülő objektumokra való felbontásnak – látszólag ellenáll. A valóságnak ezt a részét az **anyag** (**stuff**) általános névvel fogjuk illetni. Példaképpen tételezzük fel, hogy van előttem egy malac és egy kevés vaj.[2] Azt mondhatom, hogy malacból egy van, de a „vaj objektum” nem megszámlálható, hiszen akármilyen része a vaj objektumnak szintén vaj objektum, legalábbis amíg az igazán kicsi részekhez el nem jutunk. Ez az anyag és a dolgok közötti fő különbség. Sajnos nem lesz két malacunk, ha a malacot kettészeljük.

Vegyük észre, hogy a magyar nyelv az anyag és a dolgok között különbséget tesz.[3] Azt mondjuk „egy malac”, de bizonyos éttermi vagy bolti gyakorlattól eltekintve, nem mondhatjuk azt, hogy „egy vaj”. A nyelvészek különbséget tesznek a **megszámlálható főnevek** (**count nouns**), mint például malacok, gödrök vagy tételek, és a **nem megszámlálható főnevek** (**mass nouns**), mint például vaj, víz és energia között. Néhány versengő ontológiáról azt állítják, hogy e különbségeket képes kezelni. Az egyiket itt írjuk le, a többiről a történeti feljegyzésekben lehet olvasni.

Hogy az anyagot jól reprezentálhassuk, egy nyilvánvaló dologgal kezdünk. Ontológiánkban az anyag nagyobb kötegeit kell tudnunk objektumként kezelni. Például a vajban felismerjük azt a vajat, amit tegnap este az asztalon hagytunk, esetleg felemelhetjük, megmérhetjük, eladhatjuk, vagy akármi mást csinálhatunk vele. Ilyen értelemben ez egy objektum, pontosan olyan, mint egy malac. Nevezzük Vaj_3-nak. Definiálni fogjuk a *Vaj* kategóriát is. Ennek elemei, informálisan, mindazok az objektumok, amikre azt lehet mondani: „Ez vaj”, a Vaj_3-at is beleértve. A nagyon kicsi, de most elhanyagolt alkotórészek szerepére vonatkozó intésről nem megfeledkezve, a vaj objektum minden része szintén vaj objektum:

$$x \in Vaj \wedge Része(y, x) \Rightarrow y \in Vaj$$

Most már mondhatjuk, hogy vaj kb. 30 foknál olvad meg:

[2] A szövegben eredetileg „aardvark” (földimalac) szerepel, ami egy malacszerű dél-afrikai állatfajta. (*A ford.*)
[3] A szövegben eredetileg természetesen angol példa szerepel – „an aardvark” és „a butter/butter”. (*A ford.*)

Created by XMLmind XSL-FO Converter.

```
Vaj(x) ⇒ OlvadásiPont(x, Celsius(30))
```

A vaj sárga, kevésbé sűrű, mint a víz, szobahőmérsékleten puha, magas zsírtartalmú stb. Másrészt a vajnak nincs konkrét nagysága, alakja vagy súlya. Specializáltabb kategóriákat is definiálhatunk, mint például *NemSózottVaj*, ami szintén egy anyag, mert a nem sózott vaj objektum akármilyen része szintén nem sózott vaj objektum. Ha azonban az *EgyKilóVaj* kategóriát definiáljuk, amely az összes, egy kiló súlyú vaj objektumot tartalmazza, nincs többé anyagunk! Ha egy kiló vajat kettévágunk, nem kapunk kétszer egy kiló vajat – ami a valódi világ azon bosszantó dolgainak egyike, amivel sajnos együtt kell élni.

Valójában arról van szó, hogy vannak ún. **belső** (**intrinsic**) **tulajdonság**ok: ezek inkább magához az objektum szubsztanciájához tartoznak, mint az objektum egészéhez. Ha valamit kettévágunk, a részei a belső tulajdonságukat megtartják – legyen ez sűrűség, forráspont, íz, szín, a tulajdonos stb. A **külső** (**extrinsic**) **tulajdonság**ok éppen az ellenkezőt jelentik: olyan tulajdonságokat, mint a súlyt, hosszat, alakot, funkciót stb., amelyeket a részekre bontásnál megtartani nem lehet.

Az objektumok olyan osztálya, amelyek definíciójában csakis *belső* tulajdonságok szerepelnek, a szubsztanciák, illetve a nem megszámlálható főnevek osztálya. Az az osztály, amelynek definíciójában *bármilyen* külső tulajdonság is szerepel, a megszámlálható főnevek osztálya. Az *anyag* kategória a legáltalánosabb szubsztanciakategória, egyetlen belső tulajdonsága sincs. A *dolog* a diszkrét objektumok legáltalánosabb kategóriája, egyetlen külső tulajdonsága sincs. Minden fizikai objektum mindkét kategóriába tartozik, így a kategóriák együtt léteznek – ugyanazokra az entitásokra utalnak.

3. Cselekvések, szituációk és események

A cselekvések eredményeiről következtetni kulcsfontosságú egy tudásalapú ágens számára. A 7. fejezetben példákat láttuk olyan ítéletállításokra, amelyek leírják, hogy a cselekvések hogyan hatnak a wumpus világra. A 6.2. szakasz - A hely és az irány nyomkövetése részben lévő (7.3) egyenlet azt állítja például, hogy az ágens lokációja hogyan változik az ágens mozgásával. Az ítéletkalkulus egyik hátránya, hogy cselekvés leírásáról külön másolatokkal kell rendelkeznünk azokra az időpontokra, amikor a cselekvést végre fogják hajtani. Ebben az alfejezetben leírt, az elsőrendű logikára alapozó reprezentációs módszer ezt a problémát elkerüli.

3.1. A szituációkalkulus ontológiája

Az axiómák többszörös másolatainak az elkerülésére nyilvánvaló módszer az idő szerinti kvantifikálás – azaz azt mondani, hogy „$\forall$ *t*-re a *t*-beli cselekvésnek ez a $t + 1$-beli eredménye”. A $t + 1$ típusú explicit időpillanatok helyett ebben az alfejezetben szituációkkal dolgozunk, amelyek a cselekvések végrehajtásából adódó állapotok jelölései. E megközelítés neve **szituációkalkulus** (**situation calculus**) és az alábbi ontológiára támaszkodik:

- A 8. fejezethez hasonlóan a cselekvések logikai termek, például az *Előre*, a *Fordulj*(*Jobbra*). Egyelőre feltételezzük, hogy a környezetben csakis egy ágens tartózkodik. (Ha több lenne, egy további argumentummal megjelölhetjük, hogy melyik ágens hajtja végre a cselekvést).

- A **szituáció**k (**situation**s) logikai termek, amelyekhez tartozik egy (általában S_0-nak jelölt) kezdeti szituáció és minden más szituáció, amely a szituációra vonatkozó cselekvés végrehajtásából adódott. Ha az *a* cselekvést az *s* szituációban hajtották végre, az eredményezett szituációt *Eredmény*(*a*, *s*) (néha *Csináld*-nak nevezett) függvény nevezi meg. A gondolatot a 10.2. ábra illusztrálja.

- A **folyó esemény**ek (**fluent**s) olyan függvények és predikátumok, amelyek szituációról szituációra változnak. Ilyen például az ágens lokációja vagy a wumpus jóléte. A szótár szerint a folyó valamilyen folyadékszerű viselkedésre utal. Használatunkban a szituációk menti lefolyását vagy változását jelenti. $\neg$*Tart*(G_1, S_0) például azt mondja, hogy az ágens a kezdeti S_0 szituációban a G_1 aranyat nem tartja a kezében. *Kor*(Wumpus, S_0) a wumpus S_0-beli életkorára vonatkozik.

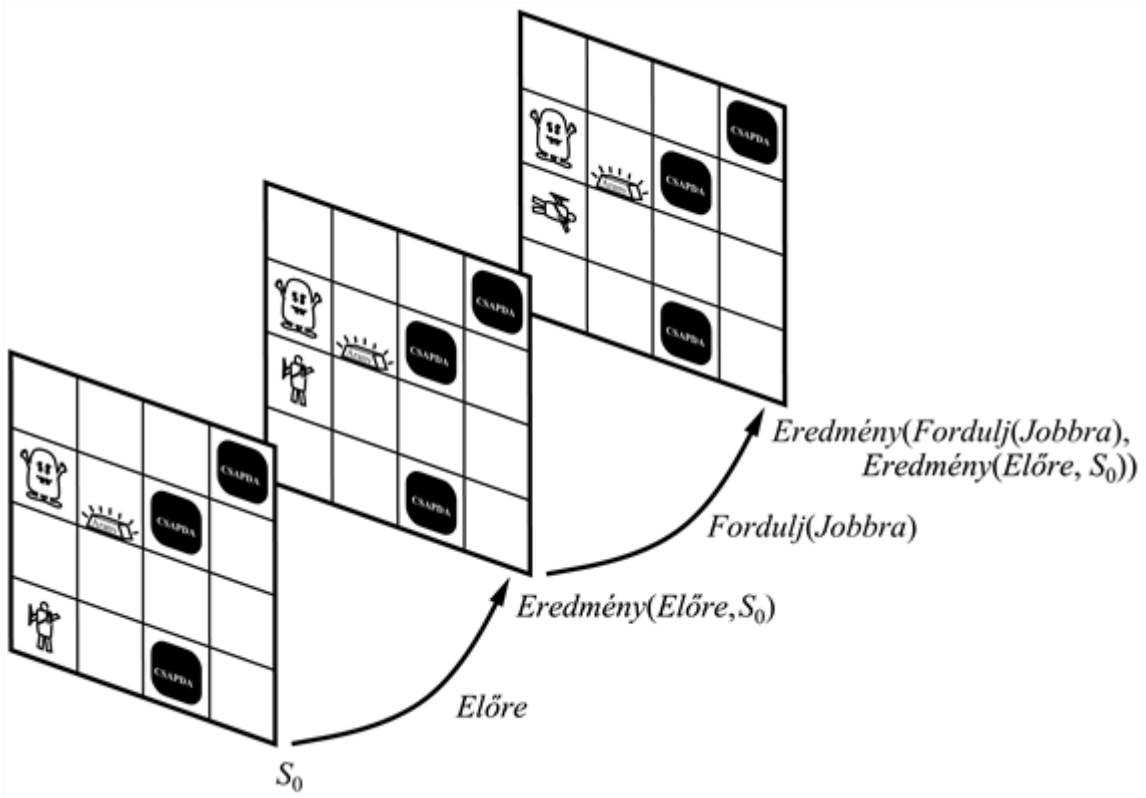
- Engedélyezzük az **időtlen**, **örök** (**atemporal**, **eternal**) predikátumokat és függvényeket is. Példaként tekinthetjük az *Arany*(G_1) vagy a *BalLába*(*Wumpus*) predikátumokat.

Az egyedi cselekvéseken túlmenően hasznos, ha a cselekvések sorozatairól is tudunk következtetni. A sorozat eredményét az egyedi cselekvések eredménye alapján definiálhatjuk. Először azt fogjuk mondani, hogy egy üres sorozat végrehajtása a szituációt változatlanul meghagyja:

```
Eredmény([], s) = s
```


Created by XMLmind XSL-FO Converter.

10.2. ábra - Szituációkalkulusban minden szituáció (az S_0-t kivéve) valamilyen cselekvés eredménye



Egy nem üres sorozat végrehajtása nem más, mint az első cselekvés végrehajtása, majd az eredményül adódó szituációban a maradó sorozatnak a végrehajtása:

```
Eredmény([a]sorozat, s) = Eredmény(sorozat, Eredmény(a, s))
```

Egy szituációkalkulus ágensnek tudnia kellene egy adott cselekvéssorozat eredményét kikövetkeztetni, ez az ún. **előrevetítő feladat** (**projection task**). A megfelelő konstruktív következtetési mechanizmus birtokában képesnek kellene lennie arra is, hogy egy kívánatos eredményt biztosító sorozatot *megtaláljon,* ez az ún. **tervkészítési feladat** (**planning task**).

Egy módosított wumpus világról fogunk példát venni, ahol az ágens orientációjával nem törődünk, és ahol az ágens egy helyről egy szomszédos helyre *Megy*. Tegyük fel, hogy ágens az [1, 1]-nél és arany az [1, 2]-nél van. A cél az aranyat az [1,1] helyen birtokoljuk. A folyó esemény predikátumok a *Nála*(*o, x, s*) és *Tart*(*o, s*). A kezdeti tudásbázis az alábbi leírást tartalmazhatná:

```
Nála(Ágens, [1, 1], S0) ∧ Nála(G1, [1, 2], S0)
```

Ez azonban még nem elég, mert nem mondtuk, hogy az S_0-ban mi *nem* igaz (a probléma további elemzését lásd a 7. szakasz - Következtetés alapértelmezett információval részben). A teljes leírás az alábbi:

```
Nála(o, x, S0) ⇔∧ [(o = Ágens ∧ x = [1, 1]) ∨ (o = G1 ∧ x = [1, 2])]

¬Tart(o, S0)
```

Arra is szükség van, hogy kijelentsük, G_1 egy arany, és [1, 1] és [1, 2] szomszédosak:

```
Arany(G1) ∧ Szomszédos([1, 1], [1, 2]) ∧ Szomszédos([1, 2], [1, 1])
```

Valaki bizonyára azt szeretné bebizonyítani, hogy ágens a célját eléri, ha az [1, 2]-re átmegy, ott megfogja az aranyat és az [1, 1]-re visszatér, azaz:

```
Nála(G1, [1, 1], Eredmény([Megy([1, 1], [1, 2]), Megfog(G1), Megy([1, 2], [1, 1])], S0)
```


Created by XMLmind XSL-FO Converter.

Annál érdekesebb lehetőség az arany birtoklását biztosító tervet készíteni, amit a „milyen cselekvéssorozat eredménye az arany az [1, 1]-en?” kérdés megválaszolásával lehet megtenni.

$\exists sorozat\ Nála(G_1, [1, 1], Eredmény(sorozat, S_0))$

Nézzük, hogy e kérdések megválaszolásához mivel kellene a tudásbázist kiegészíteni.

3.2. Cselekvések leírása a szituációkalkulusban

A szituációkalkulus legegyszerűbb változatában minden cselekvést két axiómával lehet leírni. A **lehetőségi axióma** (**possibility axiom**) megmondja, hogy a cselekvést mikor lehet elvégezni, a **hatásaxióma** (**effect axiom**) pedig azt, hogy a cselekvés végrehajtásával mi is fog történni. Annak jelölésére, hogy az *s* szituációban az *a* cselekvés végrehajtása lehetséges, a *Lehet*(*a, s*) predikátumot fogjuk használni. Az axiómák formája az alábbi:

Lehetőségi Axióma: *Előfeltételek* ⇒ *Lehet*(*a, s*)

Hatásaxióma: *Lehet*(*a, s*) ⇒ *A cselekvés végrehajtásából adódó változások*

Ezeket az axiómákat a módosított wumpus világra fogjuk felírni. A mondatok rövidsége érdekében el fogjuk hagyni az egész mondatra vonatkozó univerzális kvantorokat. Feltételezzük, hogy az *s* változó a szituációkra, az *a* változó a cselekvésekre, az *o* változó az objektumokra (az ágenseket beleértve), a *g* változó az aranyra, az *x* és az *y* változók pedig a helyre vonatkoznak.

Az erre a világra vonatkozó lehetőségi axióma azt mondja, hogy egy ágens a szomszédos helyek között mozoghat, az aktuális helyen megfoghatja az aranyat és elengedheti az aranyat, amelyet tartott:

$Nála(Ágens, x, s) \wedge Szomszédos(x, y) \Rightarrow Lehet(Megy(x, y), s)$

$Arany(g) \wedge Nála(Ágens, x, s) \wedge Nála(g, x, s) \Rightarrow Lehet(Megfog(g), s)$

$Tart(g, s) \Rightarrow Lehet(Elenged(g), s)$

A hatásaxióma azt állítja, hogy ha egy cselekvés lehetséges, akkor a cselekvés végrehajtásából adódó szituációban bizonyos tulajdonságok (folyó események) érvényesek lesznek. Az *x*-ről az *y*-ra menni cselekvés eredménye az *y*-ban tartózkodni, az aranyat megfogni eredménye az aranyat tartani, végül az aranyat elengedni eredménye az aranyat nem tartani:

$Lehet(Megy(x, y), s) \Rightarrow Nála(Ágens, y, Eredmény(Megy(x, y), s))$

$Lehet(Megfog(g), s) \Rightarrow Tart(g, Eredmény(Megfog(g), s))$

$Lehet(Elenged(g), s) \Rightarrow \neg Tart(g, Eredmény(Elenged(g), s))$

Az axiómák megállapításával tudjuk-e bizonyítani, hogy a tervünk biztosítja a cél elérését? Sajnos még nem! Először minden jól működik: a *Megy*([1, 1], [1, 2]) valójában lehetséges az S_0-ban, és a *Megy* hatásaxiómája meggyőzhet minket, hogy az ágens tényleg eléri az [1, 2]-t:

$Nála(Ágens, [1, 2], Eredmény(Megy([1, 1], [1, 1\ ]), S_0))$

Most tekintsük a *Megfog*(G_1) cselekvést. Ki kell mutatni, hogy ez lehetséges az új szituációban, azaz:

$Nála(G_1, [1, 2], Eredmény(Megy([1, 1], [1, 2]), S_0))$

Fontos

Sajnos a tudásbázisunkban az ilyen konklúziót semmi sem támasztja alá. Intuitíve persze megértjük, hogy az ágens *Megy* cselekvése az arany helyzetére nincs hatással, így az még mindig az [1, 2]-ben van, ahogy ott volt az S_0 szituációban is. *A probléma az, hogy a hatásaxióma kijelenti, hogy mi változik, de nem mond semmit arról, hogy mi nem változik*.

A változatlanul megmaradó dolgok reprezentálása az ún. **keretprobléma** (**frame problem**). A keretproblémára hatékony megoldást kell találnunk, mert a valódi világban az idő többségében majdnem minden változatlan marad. Minden egyes cselekvés a folyó eseményeknek csak kis töredékét befolyásolja.

Created by XMLmind XSL-FO Converter.

Az egyik megközelítés az expblicit **keretaxiómá**k (**frame axiom**s) felírása, amelyek *azt mondják,* hogy mi marad változatlan. Az ágens mozgásai például más objektumok helyzetét változatlanul hagyják, hacsak azokat az ágens nem tartja magánál:

$$Nála(o, x, s) \wedge (o \neq Ágens) \wedge \neg Tart(o, s) \Rightarrow Nála(o, x, Eredmény(Megy(y, z), s))$$

Ha összesen F folyó esemény és A cselekvés lenne, akkor $O(AF)$ keretaxiómára lenne szükségünk. Másfelől, ha minden cselekvésnek legfeljebb E hatása lenne, ahol az E tipikusan jóval kisebb, mint F, akkor a történteket sokkal kisebb, $O(AE)$ nagyságú tudásbázissal is ki tudnánk fejezni. Ez a **reprezentációs keretprobléma** (**representational frame problem**). A vele szorosan kapcsolódó **következtetési keretprobléma** (**inferential frame problem**) egy t-lépéses cselekvéssorozat eredményeinek $O(Et)$ időben, és nem $O(Ft)$ vagy $O(AEt)$ időben való kivetítése. Sorban egymás után az összes problémával fogunk foglalkozni. Azonban egy probléma mégis marad – biztosítani, hogy egy cselekvés sikeres elvégzéséhez minden feltételt sikerült megfogalmazni. Így például a *Megy* kudarccal fenyeget, ha az ágens *útközben* meghal. Ez az ún. **kvalifikációs probléma** (**qualification problem**), melynek teljes megoldása nincs.

3.3. A reprezentációs probléma megoldása

A reprezentációs probléma megoldása csak egy kis változást igényel abban, hogy az axiómákat hogyan írjuk fel. Ahelyett hogy felírnánk minden cselekvés hatását, azzal foglalkozunk inkább, hogy egy folyó esemény időben hogyan evolvál.[4] Az általunk használt axiómákat **követő állapot axiómá**knak (**successor-state axiom**s) hívják. Alakjuk a következő:

Követő-Állapot Axióma:

$$Cselekvés\ lehetséges \Rightarrow$$

$$(Folyó\ esemény\ igaz\ az\ eredmény\ állapotban \Leftrightarrow A\ cselekvés\ hatása\ igazzá\ tette$$

$$\vee\ Igaz\ volt\ a\ cselekvés\ előtt\ és\ a\ cselekvés\ nem\ változtatta)$$

Feltételezve, hogy lehetetlen cselekvésekkel nem foglalkozunk, vegyük észre, hogy a definícióban ⇔szerepel ⇒ helyett. Ez azt jelenti, hogy a folyó esemény *akkor és csak akkor* lesz igaz, ha a jobb oldali rész igaz. Más szóval minden folyó eseménynek a következő állapotban érvényes igazságértékét a cselekvésnek és az aktuális állapotban érvényes értékének a függvényeként határozzuk meg. Ez azt jelenti, hogy a következő állapot teljesen meghatározott az aktuális állapottól kiindulva, és így pótlagos keretaxiómákra nincs szükség.

Az ágens helyére érvényes követő állapot axióma azt mondja, hogy ágens egy cselekvés után az y-ban lesz, ha a cselekvés lehetséges, és az az y-ba való mozgásból áll, vagy ha az ágens már az y-ban tartózkodott, és a cselekvésnek a mozgáshoz nem volt köze:

$$Lehet(a, s) \Rightarrow$$

$$(Nála(Ágens, y, Eredmény(a, s)) \Leftrightarrow (a = Megy(x, y))$$

$$\vee\ (Nála(Ágens, y, s) \wedge (a \neq Megy(y, z)))$$

A *Tart* axiómája azt fejezi ki, hogy ágens tarja a g-t, ha a cselekvés a g megfogását jelentette, és a cselekvés lehetséges volt, vagy ha az ágens már tartotta a g-t és a cselekvés nem annak elengedése volt:

$$Lehet(a, s) \Rightarrow$$

$$(Tart(g, Eredmény(a, s)) \Leftrightarrow (a = Megfog(g)) \vee (Tart(g, s) \wedge (a \neq Elenged(g)))$$

Fontos

A követő állapot axiómák a reprezentációs keretproblémát megoldják, mert az axiómák össz-száma $O(AE)$ literál. Az E hatások és az A cselekvések mindegyike pontosan egyszer kerül említésre. A literálok F különböző axióma között vannak szétosztva, így egy axióma átlagos nagysága AE/F.

[4] Ez lényegében az a megközelítés, amit a 7. fejezetben a logikai áramkör-ágens tervezésénél alkalmaztunk. Az olyan axiómákat, mint a (7.4) és a (7.5), valóban követő állapot axiómáknak lehet nézni.

Created by XMLmind XSL-FO Converter.

Az éles szemű olvasó észreveszi, hogy ezek az axiómák a *Nála* folyó eseményt kezelik az ágens számára, de nem így az arany esetében. Még mindig nem tudjuk bebizonyítani, hogy ez a háromlépéses terv eljut a célhoz, az arany eléréséhez az [1, 1]-ben. Amire szükségünk lenne még, az annak a kijelentése, hogy az ágens *x*-től *y*-ig való mozgásának **implicit hatás**a (**implicit effect**), hogy minden, általa megfogott arany is vele fog mozogni (ahogy az aranyon a hangya, a hangyán a bacilus stb.). Az implicit hatások kezelését **ramifikációs problém**ának (**ramification problem**) nevezzük. A problémát általánosságban később vitatjuk meg. A jelenlegi speciális tárgyterületen e probléma a *Nála* számára kissé általánosabb követő állapot axiómák felírásával megoldható. Az új axiómák, amelyek a régieket magukban foglalják, azt fejezik ki, hogy minden *o* objektum *y*-ban tartózkodik, ha az ágens az *y*-ba ment, és az *o* maga az ágens, vagy bármi, amit az ágens tart; vagy ha az *o* már az *y*-ban volt, és az ágens nem ment sehová, miközben az *o* maga az ágens, vagy bármi, amit az ágens tart.

$$Lehet(a, s) \Rightarrow$$

$$Nála(o, y, Eredmény(a, s)) \Leftrightarrow (a = Megy(x, y) \land (o = Ágens \lor Tart(o, s)))$$

$$\lor (Nála(o, y, s) \land \neg(\exists z\ y \neq z \land a = Megy(y, z) \land$$

$$(o = Ágens \lor Tart(o, s))))$$

Egy további technikai nehézséggel számolnunk kell. Az ilyen axiómákat felhasználó következtetési eljárásnak képesnek kell lennie az azonosság hiányát kimutatni.A legegyszerűbb eset a két konstans esete, például $Ágens \neq G_1$. Az elsőrendű logika általános szemantikája lehetővé teszi, hogy a különböző konstansok ugyanazt az objektumot jelentsék, a tudásbázisnak tartalmaznia kell tehát az ilyen eseteket kitiltó axiómát. Az **egyedi elnevezések axiómá**ja (**unique names axiom**) a konstansok azonossághiányát fejezi ki, a tudásbázisban létező minden konstanspárra. Ha a tételbizonyító rendszer az azonossághiányt feltételezi, ahelyett hogy explicit módon a tudásbázisban ez le lenne írva, az **egyedi elnevezések feltételezés**ről (**unique names assumption**) beszélünk. Az azonossághiányát ki kell jelentenünk a cselekvésekre is: a *Megy*([1, 1], [1, 2]) cselekvés és a *Megy*([1, 2], [1, 1]), vagy a *Megfog*(G_1) cselekvés nem ugyanaz. Először azt fogjuk mondani, hogy a cselekvések minden típusa különböző – semmilyen *Megy* cselekvés nem lehet egy *Megfog* cselekvés. A cselekvésnevek minden párjára:

$$A(x_1, \ldots, x_m) \neq B(y_1, \ldots, y_n)$$

Most kijelentjük, hogy két, ugyanahhoz a cselekvéshez tartozó cselekvésterm ugyanazt a cselekvést jelenti, ha a benne szereplő objektumok mind azonosak:

$$A(x_1, \ldots, x_m) = B(y_1, \ldots, y_m) \Leftrightarrow x_1 = y_1 \land \ldots \land x_m = y_m$$

Ezeket az állításokat együttesen **egyedi cselekvés axiómá**knak (**unique action axiom**s) nevezzük. A kezdeti állapotleírás, a követő állapot axiómák, az egyedi elnevezések axiómák és az egyedi cselekvésaxiómák együttese elegendő annak bizonyításához, hogy a javasolt terv megvalósítja a célt.

3.4. A következtetési keretprobléma megoldása

Követő állapot axiómák a reprezentációs keretproblémát ugyan megoldják, de a következtetési keretproblémát nem. Tekintsünk egy olyan *t*-lépésű *p* tervet, hogy $S_t = Eredmény(p, S_0)$. Hogy eldönthessük, mely folyó esemény igaz S_t-ben, szükséges minden egyes *F* keretaxiómát minden *t* időlépésben kiértékelni. Mivel az axiómáknak átlagosan *AE* / *F* a nagyságuk, *O*(*AEt*) következtetéssel kell számolnunk. A munka zöme rá fog menni arra, hogy a folyó eseményeket az egyik szituációról a másikra változatlanul másoljuk át.

A következtetési keretprobléma megoldására két lehetőségünk van. Először is, eldobhatjuk a szituációkalkulust, és kitalálhatjuk az axiómák egészen új felírási módját. Ez történt meg az olyan formalizmusokban, mint a **folyó esemény kalkulus** (**fluent calculus**). Másodszor, megváltoztathatjuk magát a következtetési eljárást, hogy a keretaxiómákat hatékonyabban dolgozza fel. Erre utalást ad magának az egyszerű megközelítésnek az *O*(*AEt*) jellege. De miért függ ez a cselekvések *A* számától, amikor pontosan tudjuk, hogy mindegyik időlépésben melyik cselekvés kerül végrehajtásra? Hogy belássuk, hogyan tudnánk a dolgokon javítani, nézzük meg először a keretaxiómák formátumát:

$$Lehet(a, s) \Rightarrow$$

$$F_1(Eredmény(a, s)) \Leftrightarrow (a = A_1 \lor a = A_2 \ldots)$$

$$\lor F_2(s) \land (a \neq A_3) \land (a \neq A_4) \ldots$$

Created by XMLmind XSL-FO Converter.

Vagyis mindegyik axióma említ tehát néhány cselekvést, amelyek a folyó eseményt igazzá, és néhányat, amelyek azt hamissá teszik. Ezt a körülményt formalizálhatjuk a *PozHatás*(a, F_i) és a *NegHatás*(a, F_i) predikátumok bevezetésével. A *PozHatás*(a, F_i) azt jelenti, hogy az a cselekvés az F_i-t igazzá, a *NegHatás*(a, F_i) pedig azt, hogy hamissá teszi. A fenti axióma ezzel a következőképpen írható át:

$$Lehet(a, s) \Rightarrow$$

$$F_1(Eredmény(a, s)) \Leftrightarrow PozHatás(a, F_1) \vee [F_1(s) \wedge \neg NegHatás(a, F_1)]$$

$$PozHatás(A_1, F_1)$$

$$PozHatás(A_2, F_1)$$

$$NegHatás(A_3, F_1)$$

$$NegHatás(A_4, F_1)$$

Az, hogy ez mennyire automatizálható, a keretaxiómák pontos formátumán múlik. Hogy az ilyen axiómákra alapozva hatékony következtetési eljárást dolgozhassunk ki, három dolgot kell még megtenni:

1. Indexeljük a *PozHatás* és a *NegHatás* predikátumokat az első argumentumaik szerint, hogy amikor adott egy t időpontban bekövetkező cselekvés, a hatásait $O(1)$ időben meg lehessen keresni.

2. Indexeljük az axiómákat úgy, hogy amikor világos már, hogy egy cselekvés hatása F_i, $O(1)$ idő alatt ki lehessen keresni az F_i-re vonatkozó axiómákat. Ekkor a cselekvés hatásai között nem szereplő folyó esemény axiómáival nem is kell törődni.

3. Minden szituációt reprezentáljunk egy megelőző szituáció plusz egy növekményként. Így, amikor lépésről lépésre nem változik semmi, nem kell semmiféle munkát végeznünk. A régebbi megközelítésben $O(F)$ munka ment volna rá, hogy minden folyó esemény esetén a megelőző $F_i(s)$ feltételezésből $F_i(Eredmény(a, s))$ feltételezést generáljunk.

Minden időlépésnél tehát megnézzük az aktuális cselekvést, megkeressük a hatásait és az igaz folyó események halmazát felfrissítjük. Minden időlépésben átlagosan E ilyen frissítéssel kell számolnunk, ami az eredő komplexitásra $O(Et)$-t ad. Ez megadja a következtetési keretprobléma megoldását.

3.5. Idő- és eseménykalkulus

A szituációkalkulus akkor működik jól, ha egy ágens diszkrét, azonnal lezajló cselekvéseket hajt végre. Ha cselekvéseknek időtartamuk is van, és egymással át is lapolódhatnak, a szituációkalkulus nehézkessé kezd válni. Emiatt e kérdésekkel egy más megközelítésen belül próbálkozunk, amit **eseménykalkulus**nak (**event calculus**) fogunk nevezni és amely inkább az időpontokon, mint a szituációkon alapul. (Az „esemény” és a „cselekvés” fogalmakat felváltva is használhatjuk. Egy „esemény” informálisan a cselekvések tágabb osztályát jelenti, az explicit ágenst nélkülöző cselekvéseket beleértve. Az eseményeket az eseménykalkulusban könnyebben kezelhetjük, mint a szituációkalkulusban.)

Az eseménykalkulusban a folyó események időpillanatokra és nem szituációkra vonatkoznak, és a kalkulust úgy tervezték, hogy az időintervallumokról is lehessen következtetni. Az eseménykalkulus axióma azt mondja ki, hogy egy folyó esemény igaz egy időpontban, ha a folyó eseményt valamilyen múltbeli esemény kezdeményezte, és időközben a folyó eseményt semmilyen közbenső esemény nem állította le. Az *Inicializál* és a *Leállít* relációk a szituációkalkulus-beli *Eredmény* relációhoz hasonló szerepet töltenek be. Az *Inicializál*(e, f, t) azt jelenti, hogy az e esemény t időpontban történő bekövetkezése az f folyó eseményt igazzá teszi, míg a *Leállít*(w, f, t) jelentése, hogy f igaz értéke megszűnt. A *Történik*(e, t) azt fogja jelenteni, hogy az e esemény a t időpontban történik, a *Levág*(f, t, t_2)-t pedig arra fogjuk használni, hogy leírhassuk, hogy az f-et valamilyen esemény a t és a t_2 időpontok között leállította. Formálisan az axióma a következő:

Eseménykalkulus Axióma:

$$T(f, t_2) \Leftrightarrow \exists e, t\ Történik(e, t) \wedge Inicializál(e, f, t) \wedge (t < t_2) \wedge \neg Levág(f, t, t_2)$$

$$Levág(f, t, t_2) \Leftrightarrow \exists e, t_1\ Történik(e, t_1) \wedge Leállít(e, f, t_1) \wedge (t < t_1) \wedge (t_1 < t_2)$$

Created by XMLmind XSL-FO Converter.

A kapott funkcionalitás hasonlít a szituációkalkulusra, azonban az időpontokról és az időintervallumokról is nyilatkozhatunk. Így képesek vagyunk a *Történik*(*Kikapcs*(*LámpaKapcs*$_1$), 1:00)-val azt mondani, hogy a lámpakapcsolót pontosan 1:00-kor kapcsolták ki.

Az eseménykalkulus számos továbbfejlesztését dolgozták ki, hogy a közvetett hatásokkal, a nem zérus időtartalmú eseményekkel, a folyamatosan változó eseményekkel, a nemdeterminisztikus hatásokkal, a kauzális korlátozásokkal és más bonyodalmakkal is tudjanak dolgozni. Egyes kérdésekkel a következő alfejezetben fogunk találkozni. A tisztesség kedvéért megjegyezzük, hogy teljesen elfogadható megoldások máig sem születtek, azonban legyőzhetetlen akadályok sem merültek fel.

3.6. Általánosított események

Egyelőre két fő fogalommal – a cselekvésekkel és az objektumokkal – foglalkoztunk. Ideje most utánanézni annak, hogyan illeszkednek ezek a fogalmak egy olyan befogadó ontológiába, ahol mind a cselekvések, mind az objektumok a fizikai univerzum aspektusának foghatók fel. Egy konkrét univerzumról feltesszük, hogy térbeli és időbeli dimenziója is van. A wumpus világ a kétdimenziós rács által definiált térbeli dimenzióval és diszkrét idővel rendelkezett. A világ térben háromdimenziós és időben egydimenziós,[5] mely dimenziók mindegyike folytonos. Egy **általánosított esemény** (**generalized event**) a többdimenziós univerzum egy részének – a „tér-idő darabkának” – aspektusaiból áll össze. Ez az absztrakció az eddig látott fogalmak többségét általánosítja, a cselekvéseket, a lokációkat, az időket, a folyó eseményeket és a fizikai objektumokat beleértve. Az általános ötletet a 10.3. ábra szemlélteti. Mostantól kezdve az „esemény” egyszerű fogalommal az általánosított eseményeket fogjuk nevezni.

A második világháború például egy olyan esemény, amely a tér-idő különböző pontjaiban történt meg, ezt a szabálytalan szürke folt jelzi. Az eseményt a **részesemény**ekre (**subevent**s)[6] bonthatjuk szét:

```
RészEsemény(AngliaiCsata, MásodikVilágHáború)
```

Hasonlóképpen a második világháború a 20. század részeseménye:

```
RészEsemény(MásodikVilágHáború, HuszadikSzázad)
```

A 20. század az időnek egy *intervalluma*. Intervallumok a tér-idő olyan darabkái, amelyek két időpont között az egész teret tartalmazzák. A *Periodus*(*e*) függvény az *e* eseményt bezáró legkisebb intervallumot jelöli. A *Tartam*(*i*) egy intervallum által foglalt idő hossza, mondhatjuk tehát, hogy *Tartam*(*Periodus*(*MásodikVilágHáború*)) > *Év*(5).

Ausztrália egy *hely*, egy darabka rögzített térbeli határokkal. A határok időben változnak, geológiai vagy politikai okoknál fogva. A *Benne* predikátumot fogjuk használni az olyan részesemény reláció megjelölésére, amely akkor áll fenn, ha egy esemény térbeli vetülete *Része* egy másik esemény vetületének:

```
Benne(Sydney, Ausztrália)
```

A *Hely*(*e*) függvény az *e* eseményt befogadó legkisebb helyet jelenti.

Mint minden más objektumot, az eseményeket szintén lehet kategóriákba szervezni. A *MásodikVilágHáború* például a *Háború* kategóriába tartozik. Azt, hogy Angliában 1640-ben polgárháború zajlott, a következőképpen mondhatjuk:

```
∃w w ∈ PolgárHáború ∧ RészEsemény(w, 1640) ∧ Benne(Lokáció(w), Anglia)
```

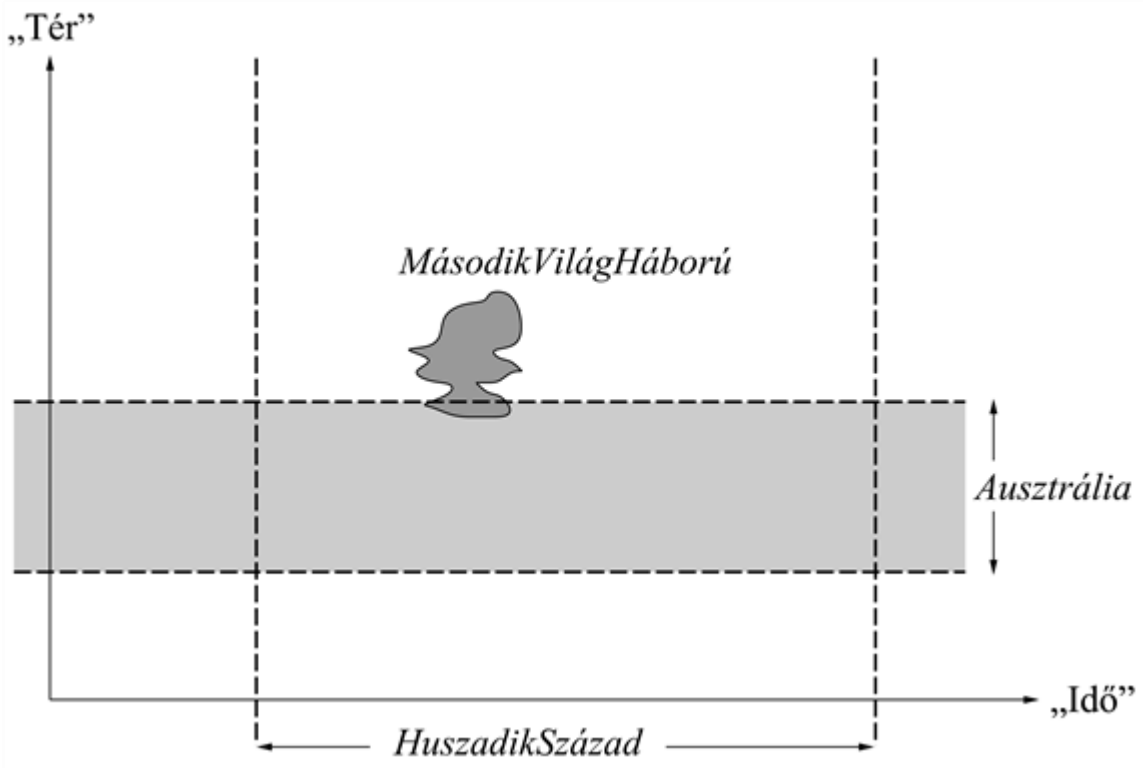
Az eseménykategória fogalma segít megválaszolni azt a kérdést, amit igyekeztünk elkerülni, amikor a 10.3. alfejezetben a cselekvések hatásairól beszéltünk. Mire hivatkoznak pontosan az olyan logikai termek, mint a *Megy*([1, 1], [1, 2])? Események ezek?

10.3. ábra - Általánosított események. Az univerzumnak térbeli és időbeli dimenziója van. Ezen az ábrán csak egy térbeli dimenziót mutatunk. Minden esemény az univerzum *Része*. Az olyan esemény, mint a *MásodikVilágHáború* a tér-idő egy

[5] A füzérelméletet tanulmányozó néhány fizikus a 10 vagy több dimenzió mellett foglal állás, mások diszkrét világról beszélnek, a 4-D folytonos tér-idő azonban a józan ész következtetési céloknak megfelelő reprezentáció.
[6] Jegyezzük meg, hogy a *RészEsemény* a *Része* reláció speciális esete, és ugyanúgy tranzitív és reflexív.

Created by XMLmind XSL-FO Converter.

tartományában történik, melynek határai kissé szabadon választhatók, és időben változók. Egy *Intervallum*, mint amilyen a *HuszadikSzázad*, rögzített és korlátos időbeli, valamint maximális térbeli kiterjedéssel rendelkezik. A *Hely*, amilyen például Ausztrália, nagyjából rögzített térbeli és maximális időbeli kiterjedéssel rendelkezik.



A válasz, talán meglepő, de az, hogy *nem*. Ennek megértéséhez nézzünk meg két „azonos” cselekvést tartalmazó tervet, mint például:

```
[Megy([1, 1], [1, 2]), Megy([1, 2], [1, 1]), Megy([1, 1], [1, 2])]
```

Ebben a tervben a *Megy*([1, 1], [1, 2]) nem lehet egy esemény neve, mert *két különböző esemény* van, amelyek különböző időpontban történnek meg. Helyette a *Megy*([1, 1], [1, 2]) egy *eseménykategória* neve – azoké az eseményeké, amikor az ágens az [1, 1]-ről az [1, 2]-re lép át. A háromlépéses terv azt mondja ki, hogy e három eseménykategória példányai fognak előfordulni.

Jegyezzük meg, hogy ez az első alkalom, amikor a kategóriák megnevezésére komplex termeket vetettünk be, egyszerű konstansszimbólumok helyett. Ez nem jelent új nehézséget. Az argumentumstruktúrát valójában a hasznunkra is fordíthatjuk. Az argumentumok eliminálása az általánosabb kategóriák bevezetését teszi lehetővé:

```
Megy(x, y) ⊆ MegyHová(y) Megy(x, y) ⊆ MegyHonnan(x)
```

Hasonlóan, argumentumok hozzáadásával még specifikusabb kategóriákat hozhatunk létre. Más ágensek cselekvéseinek leírásához például hozzáadhatjuk az ágenseket jelölő argumentumot. Így ahhoz, hogy azt mondhassuk, hogy „tegnap Shankar New Yorkból Újdelhibe utazott”, azt írhatnánk, hogy:

```
∃e e ∈ Repül(Shankar, NewYork, Újdelhi) ∧ RészEsemény(e, Tegnap)
```

Az állítás e formája annyira gyakori, hogy külön rövidítést fogunk rá alkalmazni: $E(c, i)$, aminek az a jelentése, hogy a c eseménykategória egy eleme egy i intervallum, vagy az esemény részeseménye:

```
E(c, i) ⇔∃e e ∈ c ∧ RészEseménye(e, i)
```

Így:

```
E(Repül(Shankar, NewYork, Újdelhi), Tegnap)
```


Created by XMLmind XSL-FO Converter.

3.7. Folyamatok

Az eddig látott események meghatározott struktúrával rendelkező **diszkrét esemény**ek (**discrete event**s) voltak. Shankar utazásának van kezdete, közepe és vége. Ha félbeszakítjuk, az esemény megváltozik – nem lesz többé egy New York és Újdelhi közötti utazás, hanem helyette egy New York és egy Európában valahol fekvő pont közötti utazás. A *Repül*(*Shankar*) által jelölt eseménykategóriának ezzel szemben más a jellege. Ha Shankar repülésének egy rövid intervallumát vesszük, mondjuk a harmadik 20 perces időszakaszát (amikor is türelmetlenül várja már a második csomag mézben pörkölt földimogyorót), ez az esemény még mindig a *Repül*(*Shankar*) egy része. Valójában ez akármilyen részintervallum esetében is igaz.

Az ilyen tulajdonságú események kategóriáit **folyamat**- (**process**) kategóriáknak, illetve **folytonos esemény** (**liquid event**) kategóriáknak nevezzük. Egy folyamat akármilyen részintervalluma ugyanannak a folyamatkategóriának a tagja. A diszkrét eseményekre kialakított jelölést alkalmazva mondhatjuk például, hogy Shankar valamikor tegnap repült:

```
E(Repül(Shankar), Tegnap)
```

Gyakran ki szeretnénk jelenteni, hogy valamilyen folyamat egy teljes intervallumon *keresztül* tartott, és nem csak annak valamilyen részintervallumában. Erre *T* predikátumot fogjuk használni:

```
T(Dolgozik(István), MaiEbédIdő)
```

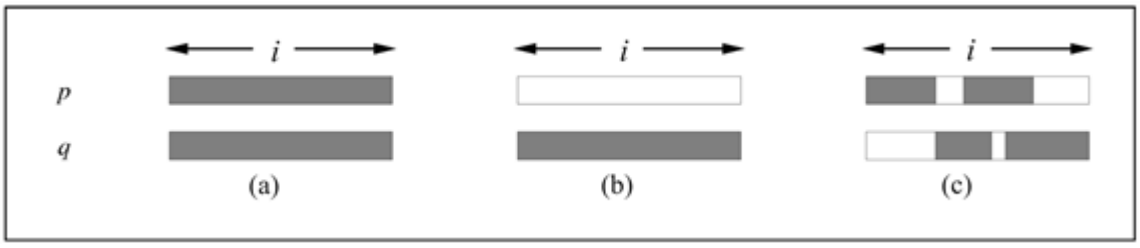
T(*c*, *i*)-nek az a jelentése, hogy egy *c* típusú esemény az egész *i* intervallum ideje alatt tartott – azaz az esemény pontosan akkor kezdődik és fejeződik be, amikor az intervallum.

Folyékony és nem folyékony események közötti megkülönböztetés pontosan analóg a szubsztanciák, avagy az *anyag* és az egyedi objektumok közötti különbséggel. Tény, hogy egyes kutatók folyékony eseményeket **temporális szubsztanciá**knak (**temporal substance**s) nevezték el, míg a vajszerű dolgok **térbeli szubsztanciá**k (**spatial substance**s).

A folytonos események a folyamatosan változó folyamatokon túlmenően képesek a folyamatosan nem változó folyamatok leírására is. Ezeket gyakran **állapot**oknak (**state**s) nevezzük. A „Shankar New Yorkban tartózkodik” például egy olyan állapotkategória, amit *Benne*(*Shankar*, *NewYork*)-kal lehetne jelölni. Azt, hogy Sankar az egész mai napot New Yorkban töltötte, úgy írhatnánk, hogy:

```
T(Benne(Shankar, NewYork), Ma)
```

10.4. ábra - Összetett események ábrázolása. (a) *T*(*Mindkettő*(*p*, *q*), *i*), másképpen *T*(*p* o *q*, *i*), (*b*) *T*(*Egyik*(*p*, *q*), *i*), (*c*) *T*(*VagyVagy*(*p*, *q*), *i*).



Primitív állapotok vagy események kombinálásával bonyolultabb képződményeket is kialakíthatunk. Ez a megközelítés a **folyó esemény kalkulus** (**fluent calculus**). A folyó esemény kalkulus nem egyedi folyó eseményeket, hanem folyó események kombinációit reifikálja. Láttuk már annak a módját, hogy az egyszerre történő két dolog eseményét hogyan reprezentáljuk, azaz láttuk a *Mindkettő*(e_1, e_2) függvényt. A folyó kalkulusban ezt általában az e_1 o e_2 infix jelöléssel rövidítik. Hogy megmondhassuk, hogy például „valaki sétált és közben rágógumit rágott”, azt írhatjuk, hogy:

```
∃p, i (p ∈ Ember) ∧ T(Sétál(p) o RágótRág(p), i)
```

A „o” függvény kommutatív és asszociatív, pontosan úgy, mint a logikai konjunkció. Definiálhatjuk a diszjunkció és a negálás analógiáját is, azzal viszont óvatosabban kell eljárnunk, mivel a diszjunkciót kétféle módon lehet értelmesen interpretálni. Amikor azt mondjuk, hogy „az ágens az utóbbi két percben vagy sétált, vagy rágógumit rágott”, gondolhatunk arra, hogy az ágens az egész idő alatt az egyik cselekvést végezte, vagy

Created by XMLmind XSL-FO Converter.

pedig, hogy a két cselekvést felváltva tette. E két lehetőség megjelölésére az *Egyik* és a *VagyVagy* függvényeket fogjuk használni. A komplex eseményeket a 10.4. ábra szemlélteti.

3.8. Intervallumok

Az idő fontos minden cselekvő ágens számára, és az időintervallumok reprezentálására számos erőfeszítés történt. Az időintervallumok két fajtájával foglalkozunk: az időpillanatokkal és a kiterjesztett intervallumokkal. A különbség köztük az, hogy csak az időpillanatnak lehet zérus időtartama:

```
Partíció({IdőPillanatok, KiterjesztettIntervallumok}, IdőIntervallumok)

i ∈ IdőIntervallumok ⇔ (Tartam(i) = Másodperc(0)
```

Most egy időskálát konstruálunk, amelynek pontjait időpillanatokhoz rendeljük hozzá; ezzel abszolút időpontokat kapunk. Az időskála tetszőleges lehet. Az időt másodpercekben fogjuk mérni, és kimondjuk, hogy a 0. idő az 1900. január 1-jei (GMT) éjfél. A *Kezdet* és a *Vég* függvények az időintervallum legkorábbi és a legkésőbbi időpillanatát adják vissza, az *Idő* függvény egy időpillanathoz tartozó pontot keres ki az időskálán. Az *IdőTartam* függvény megadja a kezdeti idő és a végidő közötti különbséget.

```
IdőIntervallum(i) → IdőTartam(i) = (Idő(Vége(i)) − Idő(Kezdete(i)))

Idő(Kezdete(AD1900)) = Másodperc(0)

Idő(Kezdete(AD2001) = Másodperc(3187324800)

Idő(Vége(AD2001)) = Másodperc(3218860800)

IdőTartam(AD2001) = Másodperc(31536000)
```

Hogy a számokat könnyebb legyen olvasni, vezessünk be egy határgumentumú *Dátum* függvényt (óra, percek, másodpercek, hónap, nap és év), melynek visszaadott értéke egy pont az időskálán:

```
Idő(Kezdete(AD2001)) = Dátum(0, 0, 0, 1, Január, 2001)

Dátum(0, 20, 21, 24, 1, 1995) = Másodperc(3000000000)
```

Két intervallum *Találkozik*, ha az egyiknek a végideje megegyezik a másiknak a kezdeti idejével. Már egyedül a *Találkozik* segítségével is lehetséges olyan további predikátumokat definiálni, mint az *Előtte*, az *Utána*, a *Közben* és az *Átlapolódik*. Azonban sokkal intuitívebb, ha a definiálásuk az időskála pontjainak segítségével történik (lásd 10.5. ábrán a grafikus reprezentációt).

```
Találkozik(i, j) ⇔ Idő(Vége(i)) = Idő(Kezdete(j))

Előtte(i, j) ⇔ Idő(Vége(i)) < Idő(Kezdete(j))

Utána(j, i) ⇔ Előtte(i, j)

Közben(i, j) ⇔ Idő(Kezdete(j)) ≤ Idő(Kezdete(i)) ∧ Idő(Vége(i)) ≤ Idő(Vége(j))

Átlapolódik(i, j) ⇔ ∃k Közben(k, i) ∧ Közben(k, j)
```

Annak megadására, hogy II. Erzsébet uralkodása VI. György uralkodását követte, Elvis uralkodása viszont az 1950-es évekkel átlapolódott, a következőt írhatjuk:

```
Utána(Uralkodása(II.Erzsébet), Uralkodása(VI.György))

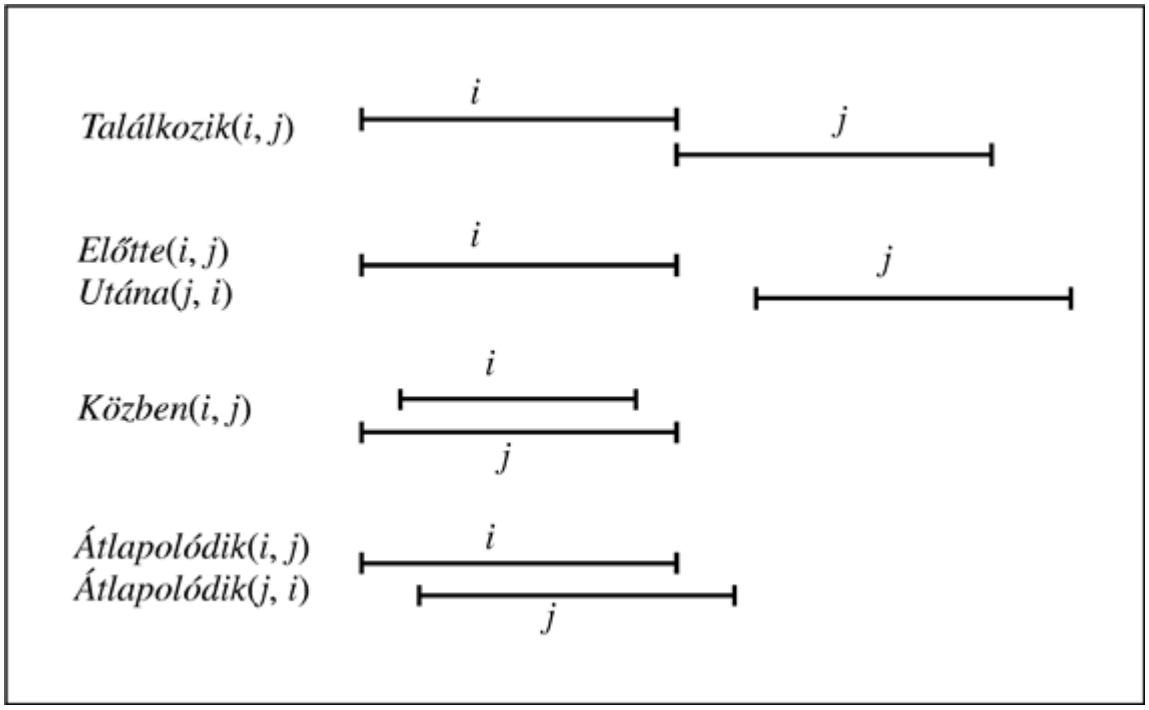
Átlapolódik(ÖtvenesÉvek, Uralkodása(Elvis))

Kezdete(ÖtvenesÉvek) = Kezdete(AD1950)

Vége(ÖtvenesÉvek) = Vége(AD1950)
```

10.5. ábra - Az időintervallumok predikátumai

Created by XMLmind XSL-FO Converter.



3.9. Folyó események és objektumok

Említettük, hogy fizikai objektumokat általánosított objektumoknak lehet tekinteni, abban az értelemben, hogy egy objektum a tér-idő egy darabkája. Az *USA* például egy olyan eseménynek képzelhető el, amely mondjuk 1776-ban, 13 állam uniójaként kezdődött és ma, 50 állam uniójaként, még folyamatban van. Az *USA* változó aspektusait állapot folyó eseményekkel írhatjuk le. Mondhatjuk például, hogy valamikor, az 1999. évben a lakossága 271 millió fő volt:

```
E(Lakosság(USA, (271000000), AD1999)
```

Egy másik ilyen aspektus, amely a szerencsétlenségektől eltekintve, négy- vagy nyolcévenként változik, az ország elnöke. Felvehető, hogy az *Elnök*(*USA*) egy olyan logikai term, amely különböző időkben különböző objektumokat jelent. Ez sajnos nem lehetséges, mert egy term egy adott modellstruktúrában pontosan egy objektumot jelöl meg. (Az *Elnök*(*USA*, *t*) jelölhetne különböző objektumokat *t* értékének megfelelően, az ontológiánk azonban az időpontokat és a folyó eseményeket szétválasztja.) Egyetlen lehetőség, hogy az *Elnök*(*USA*) egy olyan egyedi objektumot jelöl, amely különböző időkben különböző emberekből áll. Az *Elnök*(*USA*), mint objektum, George Washington volt 1789-tól 1796-ig, John Adams volt 1796-tól 1800-ig stb. (lásd 10.6. ábra). Azt, hogy George Washington 1790-ben elnök volt, a következőképpen írhatjuk le:

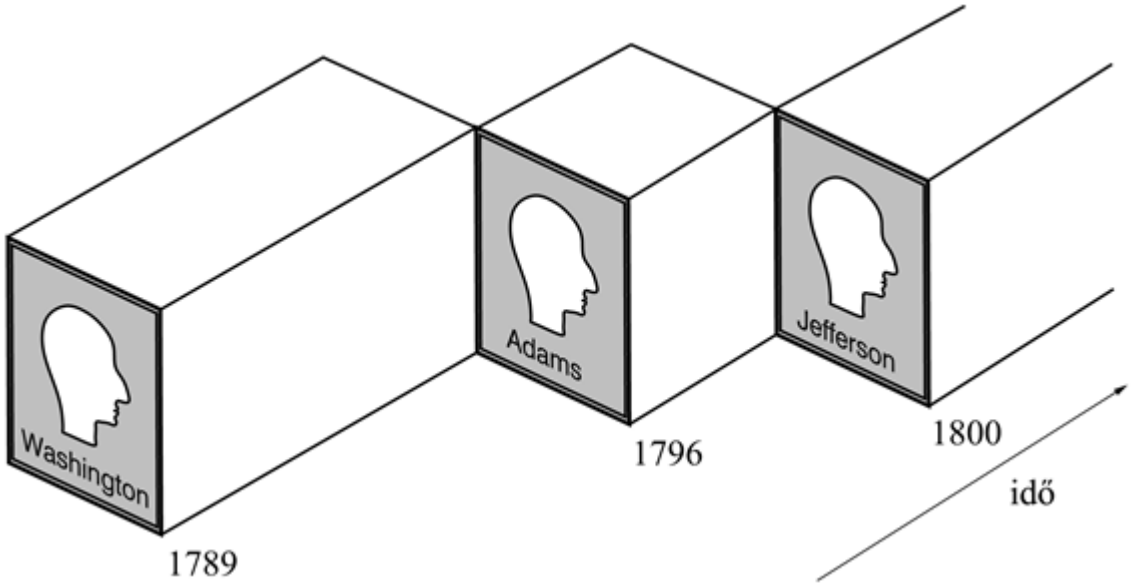
```
T(Elnök(USA) = GeorgeWashington, AD1790)
```

Azonban óvatosnak kell lenni. Ebben a mondatban az „=” inkább egy függvény, és nem egy standard logikai operátor. Az interpretáció *nem* az, hogy *GeorgeWashington* és az *Elnök*(*USA*) 1790-ben logikailag azonosak. A logikai azonosság nem olyan dolog, ami időben változhat. A logikai azonosság az egyes objektumok azon részeseményei között áll fenn, amelyeket az 1790-es elnöki periódus definiált.

Ne tévessze össze a *GeorgeWashington* fizikai objektumot az atomok egy gyűjteményével. George Washington logikailag nem azonos semmilyen konkrét atomgyűjteménnyel, mert az az atomhalmaz, amiből áll, időben lényegesen változik. Neki rövid élettartama van, minden atomnak saját hosszú élettartama van. Bizonyos periódusban metszik egymást, amikor is az atom temporális szelete a *RészEsemény*e George-nak, majd külön útjukat járják.

10.6. ábra - Az *Elnök*(*USA*) objektum sémaszerű ábrázolása létezésének első 15 évében

Created by XMLmind XSL-FO Converter.



4. Mentális események és mentális objektumok

Az eddig megkonstruált ágenseknek hiedelmeik vannak, továbbá az ágensek új hiedelmek kikövetkeztetésére is képesek. Ennek ellenére *hiedelmekre és következtetésre vonatkozó* tudással egy ágens sem rendelkezik. Az egyágenses problématerületeken egy ágensnek a saját tudásáról és következtető eljárásáról meglévő tudása hasznos a következtetés irányításában. Ha például valaki tudatában van annak, hogy Románia földrajzáról semmit sem tud, nem kell óriási erőfeszítéseket tennie, hogy az Arad és Bukarest közötti legrövidebb utat kiszámítsa. Következtethetünk a saját tudás állapotunkról annak érdekében, hogy terveket készítsük annak megváltoztatására – például azáltal, hogy Románia térképét megvásároljuk. Többágenses problématerületeken fontos, hogy egy ágens más ágensek mentális folyamataira tudjon következtetni. Egy román rendőr nyilván jól tudja, hogy melyik a legjobb út Bukarest felé, így az ágens kérhet tőle segítséget.

Végeredményben tehát olyan mentális objektumok és az azokat manipuláló mentális folyamatok modelljére van szükségünk, amelyek másvalaki fejében (vagy másvalami tudásbázisában) találhatók meg. A modellnek valósághűnek kell lennie, azonban nem kell a részletekre kitérnie. Nem kell, hogy képesek legyünk megjósolni, hogy egy konkrét ágens a következtetéseit hány milliszekundum alatt állítja elő, vagy hogy egy konkrét vizuális ingerrel szembesítve, mely neuronok tüzelnek az állat agyában. Boldogok leszünk, ha kikövetkezhetjük, hogy román rendőr eligazít minket Bukarest irányába, ha tudja az utat, és azt hiszi, eltévedtünk.

4.1. A hiedelmek formális elmélete

Az ágensek és a „mentális objektumok” közötti olyan kapcsolatokkal kezdünk, mint a *Hiszi*, a *Tudja* és az *Akarja*. Az ilyen relációkat **ítéletlogikai attitűd**öknek (**propositional attitude**s) nevezzük, mert egy ágens attitűdjét írják le egy ítéletlogikai állítással szemben. Tegyük fel, hogy Lujza hisz valamit, azaz *Hiszi*(*Lujza*, *x*). Vajon milyenfajta dolog az *x*? Először is világos, hogy *x* logikai állítás nem lehet. Ha *Repül*(*Superman*) egy logikai állítás, akkor nem mondhatjuk azt, hogy *Hiszi*(*Lujza*, *Repül*(*Superman*)), mert predikátumok argumentumaként csak termeket (és nem állításokat) fogadunk el. Ha azonban a *Repül* egy függvény, akkor a *Repül*(*Superman*) már jó jelölt egy mentális objektumra, és a *Hiszi* lehet az a reláció, amely egy ágenst összekapcsol ezzel az ítélet jellegű folyó eseménnyel. Egy állítás átalakítása objektummá nem más, mint **reifikálás**[7] (**reification**).

Úgy tűnik meg is kaptuk, amit akartunk: egy ágensnek azt a képességét, hogy következtetni tudjon más ágensek hiedelmeiről. Sajnos e megközelítésnek vannak nehézségei. Ha Clark és Superman egy és ugyanaz a személy (azaz *Clark* = *Superman*), akkor Clark repül és Superman repül egy és ugyanaz az eseménykategória, azaz *Repül*(*Clark*) = *Repül*(*Superman*). Azaz el kell fogadnunk, hogy ha Lujza azt hiszi, hogy Superman tud repülni, el kell hinnie, hogy Clark tud repülni, *akkor is, ha nem hiszi, hogy Clark Superman*. Azaz:

[7] A „reifikálás” szó a latin *res*, avagy dolog szóból származik. John McCarthy a „dolgosítás” (thingification) elnevezést javasolta, de ez nem terjedt el.

Created by XMLmind XSL-FO Converter.

$(Superman = Clark) \models (Hiszi(Lujza, Repül(Superman)) \Leftrightarrow Hiszi(Lujza, Repül(Clark)))$

Bizonyos értelemben ez igaz is. Egy bizonyos személyről, akit néha Clarknak hívnak, Lujza azt hiszi, hogy tud repülni. Más értelemben azonban ez nincs rendjén. Ha megkérdezzük Lujzát, hogy „Clark tud-e repülni?”, minden bizonnyal nemmel válaszolna. A reifikált objektumok és események jól működnek a *Hiszi* első értelmezése esetén, a másodikhoz viszont az szükséges, hogy reifikáljuk ezen objektumok és események *leírását*, így *Clark* és *Superman* különböző leírások legyenek, annak ellenére, hogy ugyanarra az objektumra vonatkoznak.

Azt a tulajdonságot, hogy egy termet egy vele ekvivalens termmel szabadon helyettesíthetünk, formálisan **referenciális átláthatóság**nak (**referential transparency**) nevezzük. Elsőrendű logikában minden reláció referenciálisan átlátható. Számunkra az lenne a jó, ha a *Hiszi* (és más ítéletlogikai attitűdök is) olyan relációkként lennének definiálhatók, melynek második argumentuma **elmosódott** (**opaque**) – azaz, ahol ekvivalens termeket egymással nem lehet helyettesíteni az értelmezés megváltoztatása nélkül.

Ennek két módja van. Az első a logika egy új változatának – az ún. **modális logiká**nak (**modal logic**) a használata, amelyben az olyan ítéletlogikai attitűdök, mint a *Hiszi* és a *Tudja* referenciálisan elmosódott **modális operátorok** (**modal operators**) lesznek. Ezzel a megközelítéssel az irodalmi és történeti megjegyzések keretében foglalkozunk. A második, a továbbiakban kifejtett megközelítés a hatékony elmosódottság elérése a különben referenciálisan átlátható nyelvben a mentális objektumok ún. **szintaktikai elmélet**e (**syntactic theory**) révén. Ez azt jelenti, hogy mentális objektumokat **füzér**ekkel (**string**s) reprezentálunk. Az eredmény az ágens tudásbázisának egy olyan durva modellje, ahol a tudásbázis az ágens által elhitt állításokat reprezentáló füzérekből áll. Egy füzér nem más, mint egy szimbólumlistát megjelölő komplex term, így a *Repül*(*Clark*) eseményt a [*R*, *e*, *p*, *ü*, *l*, (, *C*, *l*, *a*, *r*, *k*,)] karakterfüzérrel reprezentáljuk, amit „*Repül*(*Clark*)”-ként fogunk rövidíteni. A szintaktikai elmélet eleme az **egyedi füzér axióma** (**unique string axiom**), amely azt mondja ki, hogy két füzér akkor és csak akkor azonos, ha azonos karakterekből áll. Ebben a megfogalmazásban a *Clark* = *Superman* ellenére „*Clark*” ≠ „*Superman*”.

Most meg kell adnunk a füzér reprezentációs nyelv részére a szintaxist, a szemantikát és a bizonyítási elméletet, pontosan úgy, ahogyan ezt a 7. fejezetben tettük. A különbség az, hogy ezeket most mind elsőrendű logikában kell definiálnunk. A *Jelent* függvénnyel kezdünk, amely a füzért arra az objektumra képezi le, amit jelöl. A *Neve* egy olyan függvénnyel definiálható, amely egy objektumot az azt megnevező füzérre képezi le. A „*Clark*” és a „*Superman*” jelölés jelentése egy objektum, amire a *VasEmber* konstansszimbólummal fogunk hivatkozni, és aminek a neve a tudásbázisban lehet „*Clark*”, „*Superman*” vagy valamilyen más konstans, mondjuk „X_{11}”.

$Jelent(„Clark”) = VasEmber \wedge Jelent(„Superman”) = VasEmber$

$Neve(VasEmber) = „X_{11}”$

A következő lépés a következtetési szabályok definiálása egy logikai ágens számára. Kívánhatnánk például, hogy egy logikai ágens képes legyen Modus Ponenst elvégezni: ha elhiszi a *p*-t, és elhiszi, hogy $p \Rightarrow q$, akkor elhiszi a *q*-t is. Az axióma első verziója valahogy így nézhetne ki:

$LogikaiÁgens(a) \wedge Hiszi(a, p) \wedge Hiszi(a, „p \Rightarrow q”) \Rightarrow Hiszi(a, q)$

Ez azonban helytelen, mert a „$p \Rightarrow q$” füzér tartalmazza ugyan a „*p*” és „*q*” betűket, de semmi köze az olyan füzérekhez, amelyekben a *p* és *q* változók értékei szerepelnek. A helyes megfogalmazás az alábbi:

$LogikaiÁgens(a) \wedge Hiszi(a, p) -\wedge Hiszi(a, Concat(p, „\Rightarrow”, q)) \Rightarrow Hiszi(a, q)$

ahol a *Concat* egy füzérfüggvény, amely az elemeit konkátenálja. A *Concat*(*p*, „⇒”, *q*)-t le fogjuk rövidíteni „p ⇒ q”-ra. Azaz x előfordulása egy füzéren belül azt jelenti, hogy az *x* változó értékét be kell helyettesíteni. A Lisp-programozók ebben felismerik a backquote operátort, a Pearl-programozók pedig a $-változó interpolációt.

Ha Modus Ponensen kívül más következtetési szabályokat is hozzáveszünk, olyan kérdések megválaszolására leszünk képesek, mint a „Feltéve, hogy a logikai ágens ezeket a premisszákat ismeri, képes-e azt a konklúziót kikövetkeztetni?”. A közönséges következtetési szabályokon túlmenően szükségünk van hiedelemspecifikus szabályokra is. Így például az alábbi szabály azt mondja ki, hogy ha egy logikai ágens elhisz valamit, akkor azt is elhiszi, hogy elhiszi azt.

$LogikaiÁgens(a) \wedge Hiszi(a, p) \Rightarrow Hiszi(a, „Hiszi(Neve(a) , p)”)$

Created by XMLmind XSL-FO Converter.

Az ilyen ágens, axiómáinkból adódóan, minden érvényes konklúziót képes azonnal kikövetkeztetni. Ezt a jelenséget **logikai mindentudás**nak (**logical omniscience**) hívják. Számos kísérlet történt arra, hogy korlátos racionalitású ágenseket definiáljunk, amelyek véges idő alatt korlátozott számú következtetésre képesek. Ezek egyike sem teljesen kielégítő, azonban e megfogalmazások lehetővé teszik, hogy korlátos ágensekről igen korlátozott terjedelmű jóslásokat végezzünk.

4.2. Tudás és hiedelem

A filozófusok évszázadok óta tanulmányozták a hinni és a tudni közötti kapcsolatot. Általánosan elfogadott, hogy tudás a bizonyított, igaz hiedelem. Azaz, ha „megtámadhatatlanul jó oknál fogva” hiszünk valamiben, és ez a valami igaz is, akkor ezt a valamit tudjuk is. A „megtámadhatatlanul jó oknál fogva” lényeges, hogy ne mondhassunk olyat, hogy: „Tudom, hogy ez a feldobott érme fejjel felfelé fog leesni”, és ne legyen igazunk az esetek felében.

Legyen a *Tudja*(*a*, *p*) jelentése az, hogy egy *a* ágens *tudja,* hogy a *p* állítás igaz. A tudás más fajtáját is lehet definiálni. Nézzük például a „tudja, hogy vagy…, vagy” definícióját:

$$TudjaVagyVagy(a, p) \Leftrightarrow Tudja(a, p) \vee Tudja(a, „\neg p”)$$

Példánkat folytatva, Lujza tudja, hogy Clark vagy tud repülni, vagy nem, ha tudja, hogy Clark tud repülni, vagy ha tudja, hogy Clark nem tud repülni.

A „tudja, mi” koncepció valamivel bonyolultabb. Kísértést érezhetünk arra, hogy azt állítsuk, egy ágens tudja, mi Béla telefonszáma, ha létezik egy *x*, amiről az ágens azt tudja, hogy *x* = *TelefonSzáma*(*Béla*). De ez így nem jó, mert például az ágens tudhatja, hogy Aliz és Béla telefonszáma ugyanaz (azaz *TelefonSzáma*(*Aliz*) = *TelefonSzáma* (*Béla*)), de ez sokat nem segít, ha Aliz telefonszámát sem tudja. A „tudja, mi” jobb definíciója azt mondja ki, hogy az ágensnek valami olyan *x*-et kell ismernie, amely egy számjegyekből álló füzér és amely Béla telefonszáma:

$$TudjaMi(a, „TelefonSzáma(b)”) \Leftrightarrow$$

$$\exists x\ Tudja(a, „x = TelefonSzáma(b)”) \wedge x \in SzámjegyFüzér$$

Természetesen más kérdések esetén az elfogadható válasznak mások lesznek a kritériumai. Arra a kérdésre, hogy „Mi New York állam fővárosa?”, a jó válasz „Albany”, és nem az, hogy „az a város, ahol a városháza áll”. Hogy ez lehetséges legyen, a *TudjaMi*-t háromargumentumos relációvá alakítjuk át: lesz benne egy ágens, egy term és egy predikátum, amelynek igaznak kell lennie a válaszra alkalmazva. Például:

$$TudjaMi(Ágens, „Főváros(NewYork”), SajátNév)$$

$$TudjaMi(Ágens, „TelefonSzáma(Béla”), SzámjegyFüzér)$$

4.3. Tudás, idő és cselekvés

A valódi esetek többségében az ágensnek az időben változó – saját vagy más ágensek – hiedelmeivel kell foglalkoznia. Az ágensnek terveket kell szőnie, amelyek a saját hiedelmeinek változásaival járnak együtt – mint amilyen például egy térkép vásárlása, hogy Bukarestbe találjon. Más predikátumokhoz hasonlóan lehetséges a *Hiszi* reifikálása, és egy időperiódus alatt történő hiedelmek megtárgyalása. Így mondhatjuk például, hogy Lujza ma hiszi, hogy Superman tud repülni:

$$T(Hiszi(Lujza, „Repül(Superman)”, Ma)$$

Ha a hiedelem objektuma egy olyan állítás, amely időben változhat, ezt a *T* operátorral a füzéren belül írhatjuk le. Ekkor mondhatnánk azt, hogy Lujza ma hiszi, hogy Superman tegnap tudott repülni:

$$T(Hiszi(Lujza, „T(Repül(Superman), Tegnap)”, Ma)$$

Ha már az időben evolváló hiedelmeket le tudjuk írni, felhasználhatjuk az eseménykalkulus mechanizmusát, hogy hiedelmeket tartalmazó terveket készítsünk. A cselekvéseknek lehetnek **tudás-előfeltételei** (**knowledge preconditions**) és **tudáshatás**ai (**knowledge effect**s). Annak a cselekvésnek például, hogy valakinek a telefonszámát tárcsázzuk, az az előfeltétele, hogy tudjuk a számot, és annak a cselekvésnek, hogy a telefonszámot a könyvből kikeressük, az a hatása, hogy a számot tudni fogjuk. Ezt az utolsó cselekvést az eseménykalkulus mechanizmusával az alábbi módon írhatjuk le:

Created by XMLmind XSL-FO Converter.

```
Iniciálizál(Kikeres(a, „TelefonSzáma(b)”),

TudjaMi(a, „TelefonSzáma(b)”, SzámjegyFüzér), t)
```

Az információgyűjtő és az azt felhasználó terveket sokszor röviden **futási idejű változó**knak (**runtime variable**s) nevezzük, ami a korábban leírt értékbehelyettesítő változó konvenciójához szorosan kapcsolódik. Béla telefonszámának a kikeresésére, majd a tárcsázására vonatkozó tervet az alábbi alakban fel lehet írni:

```
[Kikeres(Ágens, „TelefonSzáma(Béla)”, n), Tárcsáz(n)]
```

Itt n egy futási idejű változó, aminek értékét a *Kikeres* cselekvés fogja rögzíteni, és amit majd a *Tárcsáz* cselekvés felhasználhat. Az ilyen tervek sűrűn előfordulnak a részben megfigyelhető tárgyterületeken. Látni fogunk erre példákat a következő alfejezetben és a 12. fejezetben.

5. Az internetes bevásárlás világa

Ebben az alfejezetben az internetes bevásárlással kapcsolatos tudás egy részét fogjuk kódolni. Egy olyan vásárló-kutató ágenst fogunk tervezni, amely segíti a vásárlónak az interneten megtalálni a termékek ajánlatait. A vásárló ágens a vásárlótól a termék leírását kapja és feladata a termék eladását hirdető weblapok listáját előállítani. Egyes esetekben a vásárló leírása pontos lesz, mint amilyen például a *Coolpix 995 digitális fényképezőgép* – a feladat ekkor a legjobb ajánlatot tevő boltot megtalálni. Más esetekben a leírás csak részben lesz specifikált, mint például a *300 dollárnál olcsóbb digitális fényképezőgép,* és az ágens kénytelen lesz különböző termékeket összehasonlítani.

A vásárló ágens környezete az egész világháló – ami semmiképpen sem egy játékkörnyezet. Ugyanaz a komplex, folyamatosan evolváló környezet, amit az emberek milliói mindennap használnak. Az ágens érzetei weblapok, de amíg egy emberi webfelhasználó a lapokat a képernyőre kivetített pixelek formájában látja, a vásárló ágens a lapokat karakterfüzérek alakjában fogja érzékelni, ahol közönséges szavak keverednek a HTML jelölőnyelv formattáló utasításaival. A 10.7. ábra egy weblapot mutat és a hozzá tartozó HTML-karakterfüzért. A vásárló ágens érzékelési problémájának lényege a hasznos információnak az ilyen fajtájú érzetből való kiemelése.

Világos, hogy weblapokat érzékelni egyszerűbb, mint mondjuk, Kairóban taxi vezetése közben érzékeléseket szerezni. Az internetes érzékelés mégsem mentes a bonyodalmaktól. A 10.7. ábrán látható weblap igen egyszerű a valódi vásárlási lapokhoz képest, ahol találkozni lehet olyan elemekkel, mint sütik, Java, Javascript, Flash, robot kizáró protokollok, elgépelt HTML, hangfájlok, filmfelvételek és egy JPEG kép részeként megjelenő szövegek. Egy ágens, amely képes az internet *egészével* elbánni, majdnem olyan bonyolult, mint egy robot, amely a valódi világban képes mozogni. Mi egy egyszerű ágensre összpontosítunk, amely e bonyodalmak zömét nem fogja figyelembe venni.

Az ágens első feladata, hogy releváns termékajánlatokat találjon (majd később meglátjuk, hogy a releváns ajánlatok közül hogyan kell kiválasztani a legjobbat). Legyen a *lekérdezés* a terméknek a felhasználó által begépelt leírása (például „laptop”), akkor egy weblap a *lekérdezés* szempontjából releváns ajánlat, ha a lap releváns és tényleg egy ajánlat. A lappal kapcsolatos URL-t is nyomon fogjuk követni:

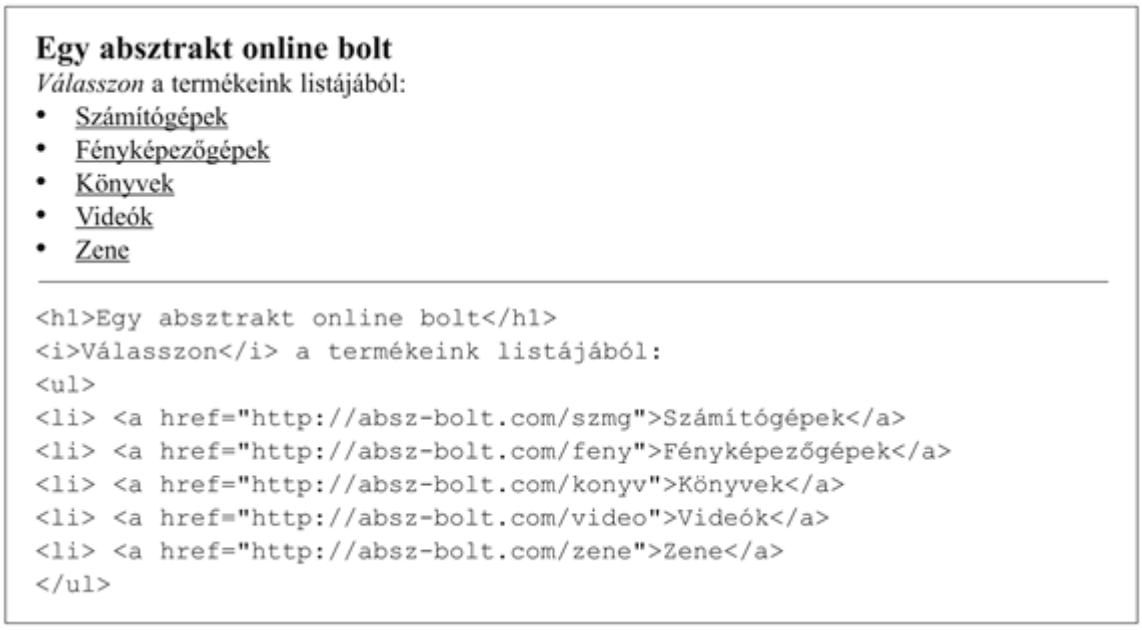
```
RelevánsAjánlat(lap, url, lekérdezés) ⇔ Releváns(lap, url, lekérdezés) ∧ Ajánlat(lap)
```

A legmodernebb laptopot bemutató lap releváns lenne, ha azonban a vásárlásra nem ad lehetőséget, nem ajánlat. Egyelőre azt fogjuk mondani, hogy egy lap ajánlat, ha a lapon egy HTML-hivatkozáson, vagy űrlapon belül a „vásárol” vagy az „ára” szavakat tartalmazza. Más szóval, ha a lap tartalmaz egy „<a … vásárol … ” alakú füzért, akkor ez egy ajánlat. A „vásárol” helyett lehetne az „ára”, illetve az „a” helyett a „form”. Felírhatjuk ennek axiomatikus alakját:

10.7. ábra - Egy absztrakt online bolt weblapja, ahogy egy böngészőt használó ember látja (felül), és a hozzá tartozó HTML-füzér, ahogy azt a böngésző vagy a vásárló ágens látja (alul). A HTML-ben a < és a > közötti karakterek a jelölő direktívák, amelyek meghatározzák, hogy a lapot hogyan kell kijelezni. Az Válasszon füzér például jelzi, hogy dőlt betűre kell átkapcsolni, a *Válasszon* szót kijelezni és a dőlt betű használatát befejezni. Egy lapazonosító, mint amilyen a http://absz-bolt.com/zene az ún. egységes erőforrás azonosító (uniform resource locator, URL).Az <a

Created by XMLmind XSL-FO Converter.

href="*url*">*hivatkozás* bejelölés jelzi, hogy az *url* felé létesíteni kell egy hipertextkapcsolatot a hivatkozó szöveg (anchor text) *hivatkozással*.



$Ajánlat(lap) \Leftrightarrow (Címkében(„a”, füzér, lap) \vee$

$Címkében(„form”, füzér, lap))$

$\wedge (Benne(„vásárol”, füzér)) \vee Benne(„ára”, füzér))$

$Címkében(„címke”, füzér, lap) \Leftrightarrow Benne(„<” + címke + füzér + „</” + címke, lap)$

$Benne(alfüzér, füzér) \Leftrightarrow \exists i \;\; füzér[i : i + Hossza(alfüzér)] = alfüzér$

Most szükséges releváns lapokat találni. A stratégia az online bolt honlapjától kezdeni, és azokkal a lapokkal foglalkozni, amelyet a releváns linkek mentén el lehet érni.[8] Ágensünk számos boltról fog tudni, például:

$Amazon \in OnlineBolt \wedge HonLap(Amazon, „amazon.com”)$

$Ebay \in OnlineBolt \wedge HonLap(Ebay, „ebay.com”)$

$ÁltBolt \in OnlineBolt \wedge HonLap(ÁltBolt, „ált-bolt.com”)$

E boltok termékeit termékkategóriákba sorolják, és a fő kategóriákhoz linkeket biztosítanak a honlaptól kiindulva. Kisebb kategóriákat a releváns linkek láncának követésével lehet elérni, és végül az ajánlatokhoz is eljutunk. Más szóval, egy lap a lekérdezés szempontjából releváns, ha a bolt honlapjától a releváns kategóriakapcsolat láncán át elérhető, és egy további link követésével a termékajánlathoz is eljutunk:

$Releváns(lap, url, lekérdezés) \Leftrightarrow$

$\exists bolt, hon \; bolt \in OnLineBolt \wedge Honlap(bolt, hon)$

$\wedge \exists url_2 \; RelevánsLánc(hon, url_2, lekérdezés) \wedge Link(url_2, url)$

$\wedge lap = VeddElőLapot(url)$

A *Link*(*tól*, *ig*) predikátum jelentése, hogy a *tól* URL-től az *ig* URL-ig létezik egy hiperhivatkozás (lásd 10.13. feladat). Hogy a *RelevánsLánc*-ot definiálhassuk, nem valamilyen régi hiperhivatkozást kell követnünk, hanem azokat, amelyeknél a csatolt hivatkozó szöveg jelzi, hogy a hivatkozás a lekérdezés szempontjából releváns. Erre a *LinkSzöveg*(*tól*, *ig*, *szöveg*)-et fogjuk használni, melynek jelentése, hogy létezik egy hivatkozás *tól*-tól *ig*-ig, *szöveg*-gel mint hivatkozó szöveggel. A *start* és a *vége* URL-ek közötti hivatkozáslánc a *d* leírás

[8] A hivatkozáskövető stratégia alternatívája egy internetkereső gépnek a használata. Az internetkeresésről, az információkinyerés mögött húzódó technológiákról a 23.2. alfejezetben lesz szó.

Created by XMLmind XSL-FO Converter.

szempontjából releváns, ha minden hivatkozás hivatkozó szövege a *d* szempontjából releváns kategórianév. A lánc létezését egy rekurzív definíció biztosítja, az üres lánccal (*start* = *vége*) mint kiinduló állapottal:

$$RelevánsLánc(start, vége, lekérdezés) \Leftrightarrow (start = vége)$$

$$\vee\ (\exists u, szöveg\ LinkSzöveg(start, u, szöveg)$$

$$\wedge\ RelevánsKategóriaNév(lekérdezés, szöveg)$$

$$\wedge\ RelevánsLánc(u, vége, lekérdezés))$$

Most definiálnunk kell, hogy egy *lekérdezés* szempontjából egy *szöveg* mikor lesz egy *RelevánsKategóriaNév*. Először tudnunk kell a füzéreket és az általuk megnevezett kategóriákat kapcsolatba hozni. Jó lesz erre a *Név*(*s*, *c*), amely azt állítja, hogy az *s* füzér a *c* kategória neve – mondhatjuk például hogy *Név*(„*laptop*”, *LaptopSzámítógép*). A *Név* predikátumra további példák a 10.8. ábrán találhatók. A következő a relevancia definiálása. Tegyük fel, hogy a lekérdezés a „laptop”. *RelevánsKategóriaNév*(*lekérdezés*, *szöveg*) igaz lesz, feltéve, hogy az alábbiak közül egy teljesül:

- A *szöveg* és a *lekérdezés* ugyanazt a kategóriát nevezi meg – például „laptop számítógép” és „laptop”.
- A *szöveg* egy szuperkategóriát nevez meg, mint például „számítógép”.
- A *szöveg* egy alkategóriát nevez meg, mint például „ultrakönnyű laptop”.

A *RelevánsKategóriaNév* logikai definíciója az alábbi:

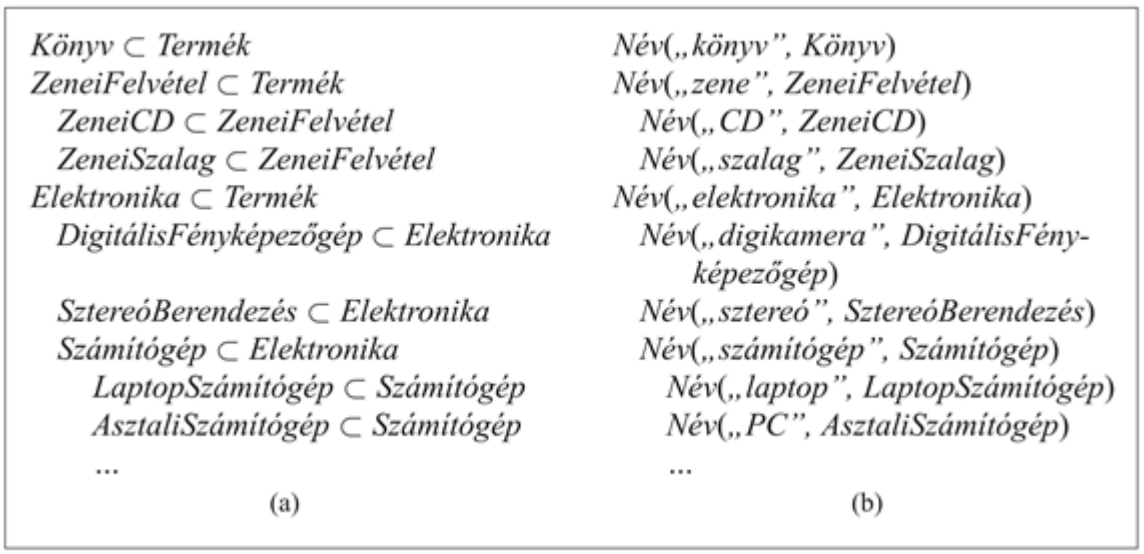
$$RelevánsKategóriaNév(lekérdezés, szöveg) \Leftrightarrow$$

$$\exists c_1, c_2\ Név(lekérdezés, c_1) \wedge Név(szöveg, c_2) \wedge (c_1 \subseteq c_2) \vee (c_2 \subseteq c_1) \quad (10.1)$$

Máskülönben a hivatkozó szöveg irreleváns, mert egy, ezen a vonulaton kívüli kategóriát nevez meg, mint például a „mainframe számítógép” vagy „pázsit és kert”.

Hogy képesek legyünk releváns kapcsolatokat követni, lényeges, hogy rendelkeznünk kell a termékek gazdag kategóriahierarchiájával. E hierarchia legfelső része hasonlíthat például a 10.8. ábrán látható hierarchiára. Nem lenne jó ötlet *az összes* lehetséges bevásárlási kategóriát listázni, mert a bevásárlónak mindig lehetnek új kívánalmai, és a termékgyártók mindig új termékekkel fognak előrukkolni, hogy a vásárlókat kielégítsék (elektromos térdmelegítő?). Egy, mondjuk ezer kategóriából álló ontológia azonban a vásárlók többsége számára hasznos eszköznek fog bizonyulni.

10.8. ábra - (a) A termékkategóriák taxonómiája. (b) Ezen kategóriák számára a hivatkozó szövegek.



Created by XMLmind XSL-FO Converter.

A termékhierarchián túlmenően rendelkeznünk kell a kategórianevek gazdag szótárával. Az élet sokkal egyszerűbb lenne, ha a kategóriák és az azokat megnevező füzérek között egy-egyértelmű kapcsolat állna fenn. A **szinonima** (**synonymy**) problémáját – két név, például „laptop számítógép” és „laptop” ugyanazon kategória számára – már láttuk. Van még továbbá az **egyértelműsítés** (**ambiguity**) problémája – egy név két vagy több különböző kategória számára. Ha például a 10.8. (b) ábrán látható tudásbázishoz a:

Név(*„CD”*, *DiplomataJármű*)

mondatot is hozzáadjuk, akkor a „CD” két különböző kategóriát fog megnevezni.

A szinonimák és az egyértelműsítés lényegesen megnövelheti az ágens által követendő utak számát, és néha megnehezítheti annak eldöntését, hogy egy adott lap tényleg releváns-e. Sokkal komolyabb probléma, hogy a felhasználó által begépelhető leírások vagy a bolt által használt kategórianevek választéka igen széles lehet. A link például a „laptop”-ról szólhat, holott a tudásbázis csak „laptopok”-ról tud, vagy pedig a felhasználó „egy számítógépet” keres, „amely elférne a Boeing 737 turistaosztályán az étkezőasztalkán”. Lehetetlen előre végig felsorolni egy kategória megnevezéseinek minden változatát, így az ágensnek bizonyos esetekben tovább kell következtetnie, hogy kiderítse, érvényes-e a *Név* reláció. A legrosszabb esetben ez a természetes nyelv teljes megértését igényelné, mely téma tárgyalását a 22. fejezetig késleltetjük. A gyakorlatban néhány egyszerű szabály, mint például engedélyezni, hogy a „laptop” illeszkedjen a „laptopok” kategóriára, igen jól szuperál. A 10.5. feladatban megkérjük majd az olvasót, hogy egy ilyen szabálykészletet fejlesszen ki, azt követően, hogy az online boltokban egy kicsit körbenézett.

Az előbbi bekezdésben megadott logikai definíció, valamint a termékkategóriákat és a megnevezési konvenciókat tartalmazó megfelelő tudásbázisok birtokában készek vagyunk-e már, hogy a lekérdezésünkre vonatkozó releváns ajánlatok halmazát kikövetkeztető algoritmust alkalmazzuk? Még nem! A hiányzó elem a *VeddElőLapot*(*url*) függvény, amely egy adott *url* címen lévő HTML lapra hivatkozik. Az ágens nem rendelkezik a tudásbázisában minden URL lap tartalmával, és az ilyen tartalom kitalálására vonatkozó explicit szabályokkal sem rendelkezik. Ehelyett elintézhetjük, hogy amikor egy részcél a *VeddElőLapot* függvényre hivatkozik, a megfelelő HTTP eljárás fog meghívódni. A következtető gép számára ily módon mintha az egész világháló jelenne meg a tudásbázisában. Ez az ún. **procedurális kibővítés**ként (**procedural attachment**) ismert általános módszer egy példája, ahol az egyes predikátumokat és függvényeket speciális rendeltetésű eljárásokkal kezeljük.

5.1. Ajánlatok összehasonlítása

Tegyük fel, hogy az előbbi részfejezet következtetési eljárása a „laptop” lekérdezésünkre több ajánlatlapot is adott vissza. Az ajánlatok összehasonlításához az ágensnek a lapokból releváns információt – ár, sebesség, diszkkapacitás, súly stb. – ki kell nyerni. Az előbb említett okoknál fogva, valódi weblapok esetén, ez igen nehéz feladatnak bizonyulhat. E probléma kezelésének megszokott módja a lapról információt kinyerő ún. **csomagolóprogram**ok (**wrapper**s) használata. Az információkinyerés technológiájával a 23.3. alfejezetben foglalkozunk. Egyelőre feltételezzük, hogy csomagolóprogram létezik, és egy lap és a tudásbázis birtokában a tudásbázishoz tényeket ad hozzá. A lapra tipikusan csomagolóprogramok egész hierarchiáját lehetne alkalmazni. Egy általánosat a dátumok és az árak kinyeréséhez, egy specifikusabbat, hogy a számítógépes termékek attribútumait találja meg, végül, ha szükséges, egy weboldal-specifikust, amely az adott bolt weboldalformátumát is tudja. Ha az ált-bolt.com egy weboldalt tartalmaz:

```
YVM ThinkBook 970. Ára: 300.000.- Ft
```

szöveggel, amit mindenféle technikai specifikáció követ, a csomagolótól elvárnánk, hogy az alábbi információt nyerje ki:

$$\exists lc, ajánlat\ lc \in LaptopSzámítógép \wedge ajánlat \in TermékAjánlat \wedge$$

$$KépernyőNagyság(lc, Cm(32)) \wedge KépernyőTípus(lc, SzínLCD) \wedge$$

$$MemóriaNagyság(lc, Mbyte(512)) \wedge CPUSebesség(lc, GHz(2,4)) \wedge$$

$$AjánlottTermék(ajánlat, lc) \wedge Bolt(ajánlat, AbsztBolt) \wedge$$

$$URL(ajánlat, \text{„abszt-bolt.com/szg/34356.html”}) \wedge$$

$$Ára(ajánlat, Ft(300000)) \wedge Dátum(ajánlat, Ma)$$

Created by XMLmind XSL-FO Converter.

Ez a példa néhány olyan problémát hivatott megmutatni, amelyek akkor jelentkeznek, ha a kereskedelmi tranzakciókhoz szükséges tudásszervezést komolyan vesszük. Vegyük észre például, hogy az ár az *ajánlatnak,* és nem magának a terméknek az attribútuma. Ez fontos, mert egy adott bolt ajánlata napról napra változhat, akár ugyanarra az egyedi laptopra nézve. Egyes kategóriák esetén – mint például a házak és a festmények esetén – ugyanazt az egyedi objektumot különböző közvetítők egy időben különböző áron is ajánlhatják. Több bonyodalom is elképzelhető, amelyek kezelésével nem foglalkoztunk. Ilyen például annak a lehetősége, hogy az ár függhet a fizetés módjától, vagy hogy a vásárlóra bizonyos kedvezmények vonatkozhatnak. Összegezve, sok érdekes munka még hátramaradt.

Az utolsó feladat a kiemelt ajánlatok összehasonlítása. Tekintsük például az alábbi három ajánlatot:

A: 2,4 GHz CPU, 512 Mbyte RAM, 80 Gbyte diszk, DVD, CDRW, 350000 Ft

B: 2,0 GHz CPU, 1 Gbyte RAM, 120 Gbyte diszk, DVD, CDRW, 400000 Ft

C: 2,2 GHz CPU, 512 Mbyte RAM, 80 Gbyte diszk, DVD, CDRW, 400000 Ft

A *C*-t az *A* **dominálj**a. Az *A* olcsóbb és gyorsabb, különben ugyanolyan. Általánosságban azt mondjuk, hogy *X* dominálja *Y*-t, ha *X* legalább egy attribútum esetében kedvezőbb értékű, és semmilyen más attribútum esetében sem rosszabb. Az *A* és a *B* közül egyik sem dominálja a másikat. Hogy megállapítsuk, melyik a jobb, tudnunk kell, hogy a vásárló hogyan mérlegeli a CPU sebességét és árát a memóriához és a diszkkapacitáshoz képest. Többszörös attribútum esetében a preferenciák általános tárgyalását a 16.4. alfejezet tartalmazza. Most az ágensünk egyszerűen ki fogja jelezni a vásárló leírásával megegyező, nem dominált ajánlatok listáját. Ebben a példában sem az *A*, sem a *B* nem dominált. Jegyezzük meg, hogy ez az eredmény azon a feltevésen alapul, hogy mindenki jobban kedveli az olcsóbb árakat, a gyorsabb processzorokat és a nagyobb tárat. Bizonyos attribútumok, mint például a notebook képernyőmérete, a vásárló partikuláris preferenciáitól függenek (a hordozhatóság a láthatósággal szemben). Az ilyenekről a vásárló ágensnek a felhasználót kell kérdeznie.

A leírt vásárló ágens egy egyszerű rendszer, és számos finomítása lehetséges. Mégis elegendő a képessége arra, hogy a megfelelő területspecifikus tudással a vásárlót ténylegesen tudja segíteni. Deklaratív konstrukciója miatt könnyen felskálázható bonyolultabb alkalmazásokhoz. Ennek az alfejezetnek a fő célja az volt, hogy kimutassuk, hogy bizonyos tudásreprezentáció – különösképpen a termék hierarchiája – szükséges az ilyen típusú ágensek számára, és ha már az ilyen formájú tudás rendelkezésre áll, a többi már nem is olyan nehéz egy tudásalapú ágens számára.

6. Következtető rendszerek kategóriák számára

Láttuk, hogy a kategóriák alapvető építőkockái bármely nagyobb méretű reprezentációs sémának. Ebben az alfejezetben a kategóriák szervezéséhez és a kategóriákkal való következtetéshez célzottan megtervezett rendszereket mutatjuk be. Két szoros kapcsolatban lévő rendszercsaládról beszélhetünk. A **szemantikus hálók** (**semantic network**) grafikus segítséget nyújtanak a tudásbázis vizualizálásában, és hatékony algoritmusokat biztosítanak, hogy egy objektum tulajdonságait a kategóriához való tartozásából kikövetkeztessük. A **leíró logikák** (**description logics**) formális nyelvet adnak a kategóriadefiníciók konstruálásához és kombinálásához, valamint hatékony algoritmusokat annak eldöntéséhez, hogy a kategóriák között fennáll-e a részhalmaz- és a szuperhalmaz-reláció.

6.1. Szemantikus hálók

1909-ben Charles Peirce egy **egzisztenciális gráf**oknak (**existential graph**s) elnevezett grafikus jelölésrendszert javasolt, ami az ő elnevezése szerint a „jövő logikája”. Ezzel kezdetét vette a „logika” és a „szemantikus hálók” szószólóinak hosszú vitája. Sajnos a vita elfedte azt a tényt, hogy a szemantikus hálók – legalább a jól definiált szemantikával ellátottak – a logika *egy formáját* jelentik. Bizonyosfajta állítások számára a szemantikus hálók nyújtotta jelölés gyakran kényelmesebb, azonban ha az „emberi interfész” problémáktól eltekintünk, a mögötte felsorakozó fogalmak – az objektumok, a relációk, a kvantifikálás stb. – mind ugyanazok.

A szemantikus hálóknak több változata van, mindegyik képes azonban az egyedi objektumokat, az objektumok kategóriáit és az objektumok közötti relációkat reprezentálni. A tipikus grafikus jelölés objektumok, illetve kategóriák neveit ovális keretekben vagy dobozokban mutatja, címkézett élekkel összekapcsolva. A 10.9. ábrán például a *Mária* és a *NőneműSzemély* között *Tagja* kapcsolat van, ami annak a logikai állításnak felel meg, hogy *Mária* ∈ *NőneműSzemély*. Hasonlóan a *Mária* és a *János* közötti *Húga* kapcsolat annak az állításnak felel meg,

Created by XMLmind XSL-FO Converter.

hogy *Húga*(*Mária*, *János*). A kategóriákat a *Részhalmaza* éllel kapcsolhatjuk össze. A krumplik és a nyilak rajzolása annyira szórakoztató, hogy könnyű kísértésbe esni. Tudjuk például, hogy minden személy anyja egy nőnemű személy, szabad-e akkor a *Személy*-től a *NőneműSzemély*-ig egy *Anyja* élt húzni? A válasz nem, merthogy az *Anyja* egy reláció egy személy és az anyja között, a kategóriáknak viszont nincsen anyjuk.[9] Emiatt a kapcsolatra a 10.9. ábrán egy külön jelölést használtunk – egy kettős vonallal történő bekeretezést. Ez a kapcsolat azt állítja, hogy:

$$\forall x \; x \in Személy \Rightarrow [\forall y \; Anyja(x, y) \Rightarrow y \in NőneműSzemély]$$

Esetleg azt is állíthatnánk, hogy egy személynek két lába van, azaz:

$$\forall x \;\; x \in Személy \Rightarrow Lába(x, 2)$$

Mint korábban, most is óvatosnak kell lennünk, nehogy azt állítsuk, hogy kategóriának van két lába. A 10.9. ábra egyszeres vonallal végzett keretezése jelzi az egy kategória minden tagjának tulajdonságaira vonatkozó állítást.

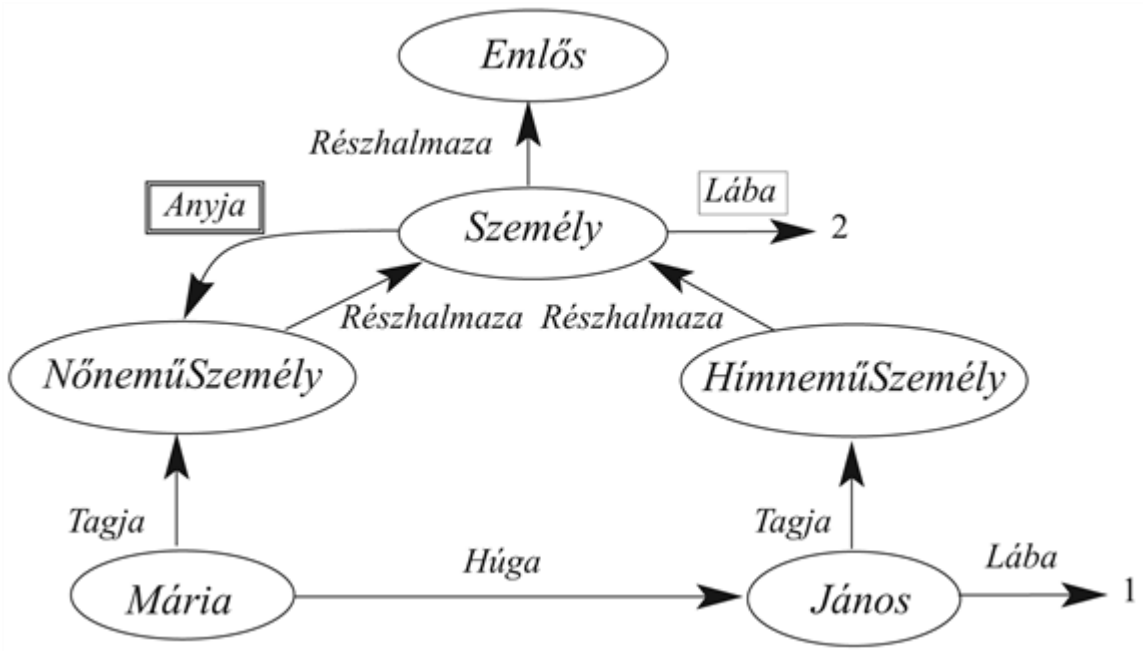
A szemantikus háló jelölésrendszere igen alkalmas a 10.2. alfejezetben bevezetett típusú **öröklődés**es (**inheritance**) következtetés végrehajtására. Így személy révén, Mária örökli azt a tulajdonságot, hogy két lába van. Ahhoz, hogy kitaláljuk, hány lába van Máriának, az öröklődéses algoritmus követi a *Tagja* élt a *Máriá*-tól az őt tartalmazó kategóriáig, majd a *Részhalmaza* élt a hierarchiába felfelé, amíg egy olyan kategóriát nem talál meg, amelyhez létezik egy bekeretezett *Lába* él – ebben ez esetben ez *Személy* kategória lesz. Ennek az öröklődéses algoritmusnak az egyszerűsége és hatékonysága a logikai tételbizonyítóhoz képest, a szemantikus hálók egyik fő vonzereje volt.

Az öröklődés komplikálódik, amikor egy objektum egyszerre több kategóriához tartozhat, vagy pedig amikor egy kategória több kategória részhalmaza. Ezt az esetet **többszörös öröklődés**nek (**multiple inheritance**) nevezzük. Az ilyen helyzetben a öröklődéses algoritmus esetleg két vagy több, egymással ellentétes választ talál a felkérésre. Ennél fogva a többszörös öröklődés bizonyos **objektumorientált programozás**i (**object-oriented programming**, **OOP**) nyelvekből, mint amilyen például a Java, ami az osztályhierarchiában öröklődést használ, ki van zárva. Szemantikus hálókban a többszörös öröklődést általában megengedjük, de ennek részletes tárgyalását a 10.7. alfejezetre hagyjuk.

10.9. ábra - Egy szemantikus háló négy objektummal (János, Mária, 1 és 2) és négy kategóriával. A relációkat a címkézett élek jelölik

[9] Bizonyos rendszerek nem tudtak különbséget tenni egy kategória tagjának tulajdonságai és kategória mint egész tulajdonságai között. Ez egyenes úton vezethet inkonzisztenciákhoz, ahogy ezt Drew McDermott kiemelte az „Artificial Intelligence Meets Natural Stupidity” című cikkében (McDermott, 1976). Más gyakori probléma volt a részhalmaz és a tagság kapcsolatoknál alkalmazott *IsA* választással, az angol használatnak megfelelően: „a cat is a mammal” (a macska egy emlős) és „Fifi is a cat” (Fifi egy macska). E témákról többet lásd 10.25. feladat.

Created by XMLmind XSL-FO Converter.



A következtetés másik közönséges formája az **inverz kapcsolat**ok (**inverse link**s) használata. Például a *VanHúga* a *Húga* inverze, azaz:

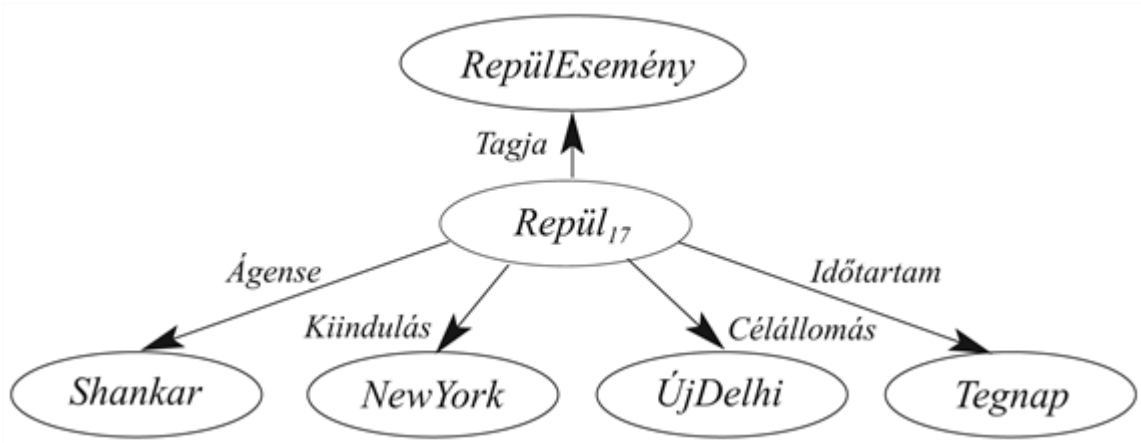
$\forall p, s \; VanHúga(p, s) \Leftrightarrow Húga(s, p)$

Ezt az állítást megfogalmazhatjuk szemantikus hálóval, feltéve hogy a kapcsolatok **reifikált**ak (**reified**), azaz önálló objektumnak tekinthetők. Így például *VanHúga* objektumot az *Inverze* éllel lehetne összekapcsolni a *Húga* objektummal. Ha egy lekérdezés azt firtatná, hogy ki Jánosnak a *Húga*, az öröklődéses algoritmus felfedezheti, hogy a *VanHúga* a *Húga* inverze, és a lekérdezést úgy megválaszolhatja, hogy a *VanHúga* kapcsolatot *János*-tól *Máriá*-ig követi végig. Az inverz információ nélkül esetleg szükséges lehetne az összes nőnemű személyt megvizsgálni, van-e netán *Húga* kapcsolata Jánossal. Ez amiatt van így, mert a szemantikus háló közvetlen indexelést csak az objektumok, a kategóriák és a belőlük induló kapcsolatok esetén biztosít. Az elsőrendű logika nyelvében ez annak felel meg, mintha a tudásbázist a predikátumok csakis első argumentumai szerint indexelnénk.

Az olvasó talán felfedezte már a szemantikus háló jelölésnek egy nyilvánvaló hátrányát az elsőrendű logikához képest. A krumplik közötti élek csak *bináris* relációkat reprezentálnak. A *Repül*(*Shankar*, *NewYork*, *Újdelhi*, *Tegnap*) állítást a hálóban közvetlenül megfogalmazni nem tudjuk. Az *n* értékű állítások hatását megkaphatjuk azonban, ha az állítást magát egy, az események megfelelő kategóriájához tartozó eseményként reifikáljuk (lásd 10.3. alfejezet). A 10.10. ábra ehhez a konkrét eseményhez tartozó szemantikus hálóstruktúrát mutat. Jegyezzük meg, hogy a bináris relációkra vonatkozó korlátozás a reifikált fogalmak gazdag ontológiáját teszi szükségessé. Az ebben a fejezetben kifejlesztett ontológia zöme ténylegesen a szemantikus hálós rendszerekben gyökerezik.

10.10. ábra - A szemantikus háló egy részlete, amely a *Repül*(*Shankar, NewYork, Újdelhi, Tegnap*) logikai állítás reprezentációját valósítja meg

Created by XMLmind XSL-FO Converter.



Állítások reifikálása lehetővé teszi, hogy az elsőrendű logika minden rögzített, függvénymentes atomi állítása reprezentálható szemantikus hálós jelöléssel. Univerzálisan kvantifikált állítások bizonyos típusai reprezentálhatók a kategóriákra alkalmazott inverz kapcsolatok, valamint egyszeresen és kétszeresen bekeretezett élek alkalmazásával, ettől még persze az elsőrendű logikától messze vagyunk. A negálás, a diszjunkció, a beágyazott függvényszimbólumok és az egzisztenciális kvantifikálás mind hiányzik. Manapság lehetséges a jelölés kiterjesztése, hogy az az elsőrendű logikával ekvivalens legyen – ilyenek például a Pierce-féle egzisztenciális gráfok vagy Hendrix-féle particionált szemantikus hálók (Hendrix, 1975) – ez viszont a szemantikus hálók fő előnyét – az öröklődéses folyamat egyszerűségét és átláthatóságát – semmisíti meg. A tervezők nagy hálókat építhetnek, és még mindig lehetnek jó ötleteik, hogy mely lekérdezések lesznek hatékonyak, mert (a) az öröklődéses eljárás lépéseit könnyű vizualizálni, (b) egyes esetekben a lekérdezés nyelve olyan egyszerű, hogy bonyolult lekérdezéseket feltenni nem is lehet. Azokban az esetekben, amikor a kifejezőerő túlságosan korlátozott, számos szemantikus hálórendszer a **procedurális kiegészítés**hez (**procedural attachment**) folyamodik, hogy a hiányokat kitöltse. A procedurális kiegészítés egy olyan módszer, amikor egy bizonyos relációra vonatkozó lekérdezés (néha állítás) egy, a relációhoz megtervezett speciális eljárás, és nem egy általános öröklődéses algoritmus meghívását eredményezi.

A szemantikus hálók egyik legfontosabb aspektusa, hogy képesek a kategóriák számára az **alapértelmezett érték**eket (**default value**s) reprezentálni. A 10.9. ábra gondos végignézésével észrevehetjük, hogy Jánosnak egy lába van annak ellenére, hogy ő egy személy, és minden személynek két lába van. Egy tisztán logikai tudásbázisban ez egy ellentmondás lenne, de a szemantikus hálóban az az állítás, hogy minden személynek két lába van, egy alapeseti állítás, azaz egy személyről feltételezzük, hogy két lába van, ha nincs ezzel ellentmondó specifikusabb információ. Az alapeseti szemantikát természetesen az öröklődéses következtetés kényszeríti ki, mert a kapcsolatokat magától az objektumtól (itt Jánostól) felfelé követi, és megáll, hacsak egy értéket meg nem talál. Azt mondjuk, hogy az alapeseti értéket egy specifikusabb érték **felülír**hatja (**overridden**). Vegyük észre, hogy lábak számát felül tudnánk írni egy *EgyLábúSzemély* kategória létesítésével, ami a *Személy Részhalmaza* lenne, és amelynek *János* lenne a tagja.

A háló szigorúan logikai szemantikáját megtarthatjuk, ha azt mondjuk, hogy a *Lába* állítás a *Jánosra* vonatkozó kivételt tartalmazza:

$$\forall x \; x \in Személy \wedge x \neq János \Rightarrow Lába(x, 2)$$

Egy rögzített háló esetén ez szemantikusan megfelelő, azonban ha sok kivételünk lesz, kevésbé tömör lesz a felírásunk, mint a hálós jelölés maga. Több állítással frissített háló esetén az ilyen megközelítés sajnos kudarcra van ítélve – amit valójában mondani szeretnénk az az, hogy minden, egyelőre még ismeretlen egylábú személy szintén kivételnek számít. Ezzel a témával és az alapeseti következtetéssel általában bővebben a 10.7. alfejezet foglalkozik.

6.2. Leíró logikák

Az elsőrendű logika szintaxisát úgy tervezték, hogy egyszerű legyen objektumokról kijelentéseket tenni. A **leíró logiká**k (**description logic**s) olyan jelölések, amelyeket a kategóriák definícióinak és tulajdonságainak könnyebb leírására terveztek. A leíró logikák a szemantikus hálókból alakultak ki, válaszul arra a nyomásra, hogy a hálók jelentését formalizáljuk, hangsúlyozva emellett a taxonómia szervezési elvként való használatát.

Created by XMLmind XSL-FO Converter.

A leíró logika alapvető következtetési feladata a **részvizsgálat** (**subsumption**) – annak ellenőrzése, hogy egy kategória része-e egy másiknak a definícióik alapján – és az **osztályozás** (**classification**) – annak ellenőrzése, hogy egy objektum egy kategóriába tartozik-e. Bizonyos rendszerek egy kategóriadefiníció **konzisztenciavizsgálat**át (**consistency**) is tartalmazzák arra, hogy a tagsági kritérium logikailag kielégíthető-e.

10.11. ábra - A leírások szintaxisa a CLASSIC nyelv egy részhalmazában

Fogalom	→	**Dolog** | *Fogalomnév*
	|	**És**(*Fogalom*, …)
	|	**Mind**(*SzerepNév*, *Fogalom*)
	|	**Legalább**(*Egész*, *SzerepNév*)
	|	**Legfeljebb**(*Egész*, *SzerepNév*)
	|	**Betölt**(*SzerepNév*, *EgyediNév*, …)
	|	**UgyanazMint**(*Út*, *Út*)
	|	**EgyBelőle**(*EgyediNév*, …)
Út	→	[*SzerepNév*, …]

A CLASSIC nyelv (Borgida és társai, 1989) egy tipikus leíró logika. A 10.11. ábra mutatja a CLASSIC leírások szintaxisát (az érthetőség kedvéért a nyelv kulcsszavait magyar fordításban adjuk meg – *a ford.*).[10] Azt az állítást például, hogy az agglegények nőtlen, felnőtt férfiak, következőképpen írhatnánk le:

```
Agglegény = És(Nőtlen, Felnőtt, Férfi)
```

Ennek elsőrendű logikai ekvivalense a következő lenne:

```
Agglegény(x) ⇔ Nőtlen(x) ∧ Felnőtt(x) ∧ Férfi(x)
```

Vegyük észre, hogy a leíró logika hatékonyan teszi lehetővé azt, hogy a predikátumokon direkt logikai műveleteket hajtsunk végre anélkül, hogy először mondatokat alkotnánk, amiket azután összekapcsolunk kötőszavakkal. Bármilyen leírás a CLASSIC-ban kifejezhető elsőrendű logikával, de bizonyos leírások a CLASSIC-ban sokkal világosabbak. Például az emberek egy olyan halmazának leírásához, akiknek legalább három fiuk van, és a fiúk mind munkanélküliek, és a feleségeik orvosok, valamint maximum két lányuk van, akik vagy fizika, vagy matematika tanszéken professzorok, azt használnánk, hogy:

```
És(Férfi, Legalább(3, Fiú), Legfeljebb(2, Lány),

Mind(Fiú, És(Munkanélküli, Házas, Mind(Feleség, Orvos))),

Mind(Lány, És(Professzor, Betölt(Tanszék, Fizika, Matematika))))
```

Gyakorlatként meghagyjuk ennek lefordítását elsőrendű logikára.

A leíró logikák talán egyik legfontosabb aspektusa a következtetés nyomon követhetőségére tett hangsúly. Egy problémaeset megoldása a leírásával kezdődik, majd annak megkérdezésével folytatódik, hogy része-e a lehetséges megoldás a kategóriák egyikének. Standard elsőrendű logikában a válasz idejének megjóslása gyakran lehetetlen. Gyakran a felhasználóra vár a reprezentáció olyan átalakítása, hogy kikerülje azokat a mondathalmazokat, amelyek miatt úgy tűnik, hogy a rendszernek több hétbe telik a probléma megoldása. A leíró logikák viszont azt biztosítják, hogy a részvizsgálat megoldható legyen polinomiális idő alatt a probléma leírásától függően.[11]

[10] Vegyük észre, hogy a nyelv *nem* engedi meg annak állítását, hogy az egyik fogalom vagy kategória egy másiknak a része. Ez egy átgondolt irányvonal: a kategóriák közötti részviszonyt a kategóriák bizonyos aspektusaiból kell tudni levezetni. Ha ez nem sikerül, akkor valami hiányzik a leírásból.

[11] A CLASSIC nyelv a gyakorlatban hatékony részvizsgálatot biztosít, a legrosszabb eset azonban exponenciális futásidejű.

Created by XMLmind XSL-FO Converter.

Ez a gyakorlatban csodálatosan hangzik, míg az ember rá nem jön, hogy a következő két konzekvencia valamelyike lehetséges csak: a nehéz problémákat vagy nem lehet kifejezni, vagy exponenciálisan nagy leírást igényelnek! Mindenesetre a kezelhetőség eredményei vetnek némi fényt arra, hogy milyen konstrukciók okoznak bajt, és így segítik a felhasználót annak megértésében, hogyan viselkednek különböző reprezentációk. Például a leíró logikákból általában hiányzik a *negálás* és a *diszjunkció*. Ezek mindketten arra kényszerítik az elsőrendű logikai rendszert, hogy lényegében egy exponenciális eset analízisen menjen keresztül a teljesség biztosítása érdekében. A negálás és a diszjunkció ugyanezen ok miatt van kizárva a Prolog nyelvből is. A CLASSIC csak egy korlátozott formájú diszjunkciót engedélyez a *Kitölt* és az *EgyBelőle* szerkezetekben, ami lehetővé tesz diszjunkciót explicit módon megszámolt egyedek felett, de nem a leírások felett. Diszjunktív leírással az egymásba ágyazott definíciók könnyen exponenciális számú alternatív úthoz vezethetnek, amelyekben az egyik kategória könnyen része lehet egy másiknak.

7. Következtetés alapértelmezett információval

Az előbbi alfejezetben láttunk egy egyszerű példát egy alapértelmezett státussal rendelkező állításra: „az embereknek két lába van”. Ezt az alapértelmezett értéket specifikusabb információval, mint például „a Kékszakállúnak egy lába van”, felülírhatjuk. Láttuk, hogy a szemantikus háló öröklődési mechanizmusa az alapértelmezett értékek felülírását egyszerű és természetes módon oldja meg. Ebben a részben az alapértelmezett értékeket mélyebben tanulmányozzuk annak érdekében, hogy az alapértelmezett értékek *szemantikáját* alaposabban megértsük, és ne szorítkozzunk csupán a procedurális mechanizmus megadására.

7.1. Nyitott és zárt világok

Tegyük fel, hogy egyetemen a számítástudományi tanszék hirdetőtábláját nézzük, és azt az üzenetet látjuk, hogy: „Az alábbi tantárgyakat fel lehet venni: SZT 101, SZT 102, SZT 106 és VIM 101.” Próbáljuk most megválaszolni, vajon hány tárgyat lehet felvenni? Ha a válasza az, hogy „négy”, ez megegyezik egy tipikus adatbázis válaszával. Ha adott egy relációs adatbázisa:

$$Tantárgy(SZT, 101),\ Tantárgy(SZT, 102),\ Tantárgy(SZT, 106),$$

$$Tantárgy(VIM, 101) \quad (10.2)$$

négy állítás megfelelőjével, a count * from Tantárgy SQL-felkérés 4-es válasszal tér vissza. Másfelől egy elsőrendű logikai rendszer válasza az lenne, hogy „egy és végtelen között valahány”, és nem „négy”. Ennek magyarázata, hogy a *Tantárgy* állítás nem zárja ki, hogy más, nem említett tantárgyakat is fel lehessen venni, és azt sem, hogy az említett tantárgyak mind különbözők.

A példa mutatja, hogy az adatbázisok és az emberi kommunikáció konvenciója az elsőrendű logikától legalább két aspektusban különbözik. Először, az adatbázisok (és az emberek) feltételezik, hogy a megadott információ *teljes*, vagyis hogy az igazként ki nem jelentett rögzített atomi formulákról feltételezhetjük, hogy hamisak. Ez az ún. **zárt világ feltételezés** (**closed-world assumption**, **CWA**). Másodszor, elfogadjuk általában, hogy különböző nevek különböző objektumokat jelentenek. Ez az ún. **egyedi elnevezések feltételezés** (**unique names assumption**, **UNA**), amit a cselekvés-nevek kontextusában először a 10.3. alfejezetben vezettünk be.

Az elsőrendű logika e konvenciókhoz nem folyamodik, így precízebbnek kell lennie. Hogy kijelentsük, *csak* négy különböző tantárgyat lehet választani, azt kellene írni, hogy:

$$Tantárgy(d, n) \Leftrightarrow [d, n] = [SZT, 101] \vee [d, n] = [SZT, 102]$$

$$\vee [d, n] = [SZT, 106] \vee [d, n] = [VIM, 101] \quad (10.3)$$

A (10.3) egyenletet a (10.2) **lezárás**ának[12] (**completion**) nevezzük. A lezárás általában minden predikátumhoz egy definíciót, egy „akkor és csak akkor” állítást rendel hozzá. Minden definícióban mindegyik, a predikátumot a fejében tartalmazó definit klózhoz egy diszjunkció fog tartozni.[13] A lezárást általánosságban az alábbi módon szerkesztik meg:

1. Gyűjtsük ki az ugyanolyan (P) predikátum névvel és (n) aritással rendelkező klózokat.

[12] A felfedezőjéről, Keith Clarkról néha Clark-lezárásnak is nevezik.
[13] Jegyezzük meg, hogy ez egyben a 10.3. alfejezetben megadott követő állapot axiómák alakja is.

Created by XMLmind XSL-FO Converter.

2. Minden klózt az ún. **Clark Normál Formára** (**Clark Normal Form**) transzformáljuk: cseréljük a:

$P(t_1, \ldots, t_n) \leftarrow Törzs$

kifejezést, ahol a t_i-k termek, a:

$P(t_1, \ldots, t_n) \leftarrow \exists w_1 \ldots w_m\ [v_1, \ldots, v_n] = [t_1, \ldots, t_n] \wedge Törzs$

kifejezésre, ahol a v_i-k az újonnan bevezetett változók és a w_i-k az eredeti klóz változói. Használjuk minden klóz esetén a v_i változók ugyanazon halmazát. Eredményként a:

$P(v_1, \ldots, v_n) \leftarrow B_1$

.

.

.

$P(v_1, \ldots, v_n) \leftarrow B_k$

klózok halmazát kapjuk.

3. Kombináljuk ezeket össze egy nagy diszjunktív klózzá:

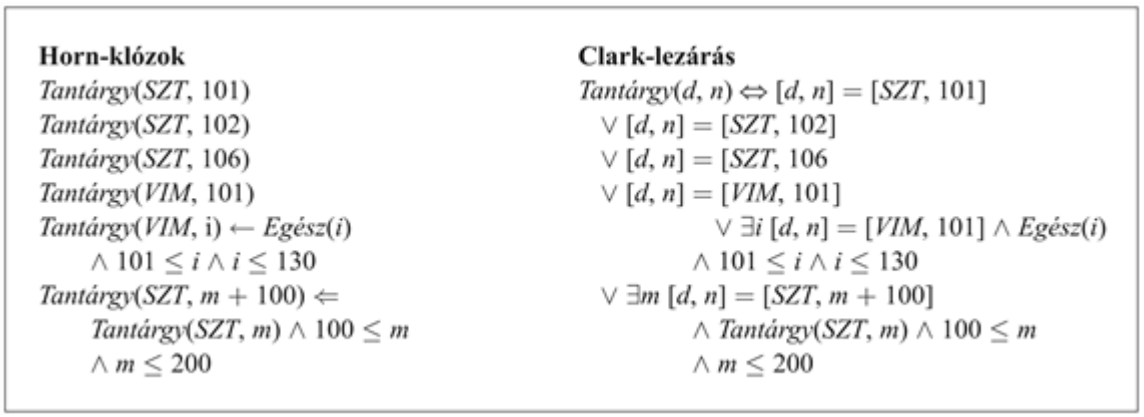
$P(v_1, \ldots, v_n) \leftarrow B_1 \vee \ldots \vee Bk$

4. Zárjuk le azáltal, hogy a ←-at ekvivalenciára cseréljük:

$P(v_1, \ldots, v_n) \Leftrightarrow B_1 \vee \ldots \vee B_k$

A Clark-lezárás egy példáját – szabályokat és rögzített tényeket tartalmazó tudásbázis esetén – a 10.12. ábrán láthatjuk. Hogy hozzáadhassuk az egyedi elnevezések feltételezést is, egyszerűen adjuk meg az azonossági reláció Clark-lezárását, ahol csak azok az ismert tények, hogy *SZT* = *SZT* és 101 = 101 stb. Ennek megvalósítását gyakorlásképpen meghagyjuk az olvasónak.

10.12. ábra - Horn-klózok egy halmazának Clark-lezárása. Az eredeti Horn-program (balra) négy tantárgyat említ explicit módon, és azt is állítja, hogy minden egészre a 101 és a 130 között létezik egy matematikai tárgy, meg azt is, hogy minden SZT tárgyhoz a 100-as (BSc) sorozatban létezik egy megfelelő tantárgy a 200-as (MSc) sorozatban. A Clark-lezárás (jobbra) azt mondja, hogy más tantárgy nincs is. A lezárással és az egyedi elnevezések feltételezéssel (valamint az *Egész* predikátum nyilvánvaló definíciójával) együtt eljutunk a kívánt konklúzióig, miszerint pontosan 36 tantárgy van: 30 matematikai és 6 SZT tárgy.



A zárt világ feltételezés lehetővé teszi, hogy megtaláljuk egy reláció **minimálmodell**jét (**minimal model**). Ez azt jelenti, hogy a *Tantárgy* relációhoz a legkevesebb elemet tartalmazó modellt találhatjuk meg. A (10.2) egyenletben a *Tantárgy* minimálmodellje négyelemű, kevesebb már ellentmondáshoz vezet. Horn-klóz

Created by XMLmind XSL-FO Converter.

tudásbázisok esetén mindig létezik egy *egyértelmű* minimálmodell. Jegyezzük meg, hogy az egyedi elnevezések feltételezés mellett ez az azonossági relációra is vonatkozik: minden term csakis saját magával azonos. Paradox módon ez azt jelenti, hogy a minimál modellek egyben maximálisak abban az értelemben, hogy annyi objektumot tartalmaznak, amennyi csak lehetséges.

Lehetséges egy Horn-program Clark-lezárását képezni, majd a következtetések levonása végett egy tételbizonyítónak átadni. Hatékonyabb azonban általában egy olyan speciális rendeltetésű következtető gépet használni, mint amilyen a Prolog, amelynél a zárt világ és az egyedi elnevezések feltételezések a következtetési mechanizmusba be vannak építve.

Akik a zárt világ feltételezéshez folyamodnak, óvatosan kell megválasztaniuk az általuk használt következtetést. Egy népszámlálási adatbázisban például, ésszerű a zárt világ feltételezést használni, ha a következtetés a városok populációjáról történik, de nyilván helytelen konklúzió lenne elfogadni, hogy a jövőben nem születnek gyerekek, csakis annak alapján, hogy az adatbázis a jövőbeli születési bejegyzéseket nem tartalmazza. A zárt világ feltételezés az adatbázist **teljes**sé (**complete**) teszi abban az értelemben, hogy minden atomi lekérdezésre vagy pozitív, vagy negatív választ kapunk. Ha valamely tényállásról (mint például a jövőbeli születések) tényleg tudatlanok vagyunk, a zárt világ feltételezést használni nem szabad. Egy komplikáltabb tudásreprezentációs rendszerben a felhasználónak esetleg szabad lenne specifikálni a zárt világ feltételezés használatát leíró szabályokat.

7.2. Negálás mint kudarc és stabil modell szemantika

A 7. és a 9. fejezetekben láttuk, hogy Horn-klóz formájú tudásbázisnak kedvező számítástechnikai tulajdonságai vannak. Sok alkalmazásban azonban nem kényelmes azt biztosítani, hogy klózok törzseiben csak pozitív literálok legyenek. Szeretnénk például azt mondani, hogy „Kimehetsz, ha nem esik”, anélkül hogy olyan predikátumokhoz kellene folyamodni, mint a *NemEsik*. Ebben a részben annak a lehetőségét kutatjuk, hogy a Horn-klózokhoz az explicit negálás egy formáját adjuk hozzá a **negálás mint kudarc** (**negation as failure**) ötletét felhasználva. Az ötlet az, hogy egy negatív „*not P*” literált igaznak „bizonyíthatunk”, hasonlóan, mint ahogy *P* bizonyítása kudarcba fulladhat. Ez az alapeseti következtetés egy formája, ami a zárt világ feltételezéshez szorosan kapcsolódik: tételezzük fel, hogy valami hamis, ha nem bizonyítható, hogy igaz. Hogy a negálás mint kudarc-ot a logikai „¬” operátortól megkülönböztethessük, megjelölésére a „*not*”-ot fogjuk használni.

A Prolog megengedi a *not* operátort egy klóz törzsében. Tekintsük például az alábbi Prolog programot:

```
IDEmeghajtó ← Meghajtó ∧ not SCSImeghajtó

SCSImeghajtó ← Meghajtó ∧ not IDEmeghajtó

SCSIvezérlő ← SCSImeghajtó

Meghajtó
```

Az első szabály azt mondja, hogy ha számítógépben merevlemez-meghajtónk van, és ez nem SCSI, akkor IDE-nek kell lennie. A másik szabály azt mondja, hogy ha ez nem IDE, akkor SCSI-nek kell lennie. A harmadik azt mondja, hogy az SCSI-meghajtó létezése egy SCSI-vezérlőt tételez fel, végül a negyedik kijelenti, hogy tényleg van egy meghajtó. Ennek a programnak *két* minimálmodellje van:

M_1 = {*Meghajtó, IDEmeghajtó*}

M_2 = {*Meghajtó, SCSImeghajtó, SCSIvezérlő*}

A minimálmodellek nem képesek a negálás mint kudarc-ot használó programok szándékolt szemantikáját kifejezni. Tekintsük a következő programot:

```
P ← not Q
```

Ennek két minimálmodellje van: {*P*} és {*Q*}. Az elsőrendű logika szemszögéből van ennek értelme, mivel a $P \Leftarrow \neg Q$ a $P \vee Q$-val ekvivalens. A Prolog szemszögéből azonban aggódni kellene: *Q* soha nem jelenik meg a nyíl bal oldalán, hogyan lehet hát egy következmény?

Egy alternatíva a **stabil modell** (**stable model**), ami egy olyan minimálmodell, ahol minden atomnak **igazolása** (**justification**) van, azaz létezik hozzá olyan szabály, amelyben a fej az atom, és ahol a törzs minden literálja

Created by XMLmind XSL-FO Converter.

kielégített. Formálisan M egy H program egy stabil modellje, ha M a H az M-re vonatkozó **redukált**jának (**reduct**) egy egyértelmű minimálmodellje. Egy H program redukáltját úgy definiáljuk, hogy H-ból először minden olyan szabályt törlünk, amely törzsében a *not A* literál szerepel, ahol A a modell része, majd a maradó szabályokban a negatív literálokat töröljük. Mivel H redukáltja most már egy Horn-klóz lista, egy egyértelmű minimálmodellel kell rendelkeznie.

A $P \leftarrow not\ Q$ redukáltja a $\{P\}$-re nézve maga a $\{P\}$, aminek minimálmodellje $\{P\}$. Így $\{P\}$ egy stabil modell. A $\{Q\}$-ra vonatkozó redukció egy üres program, aminek minimálmodellje $\{\}$. $\{Q\}$ tehát nem stabil modell, mert Q-nak a (10.5) egyenletben nincs igazolás. Egy másik példa a (10.4) redukáltjára az M_1-re nézve:

```
IDEmeghajtó ← Meghajtó

SCSIvezérlő ← SCSImeghajtó

Meghajtó
```

Ennek minimálmodellje M_1, így M_1 egy stabil modell. A **válaszhalmaz programozás** (**answer set programming**) a logikai programozásnak a negálás mint kudarc-cal bővített fajtája, amely úgy működik, hogy a logikai programot rögzített formába transzformálja, majd stabil modelleket (amelyeket **válaszhalmaz**oknak – **answer set**s – is neveznek) keres, ítéletkalkulus modell-ellenőrzési technikákat felhasználva. A válaszhalmaz-programozás leszármazottja tehát mind a Prolognak, mind az olyan gyors ítéletkalkulus-beli kielégíthetőségi tételbizonyítóknak, mint a `WALKSAT`. A válaszhalmaz programozást sikerrel alkalmazták tervkészítési problémákra, hasonlóan az ítéletkalkulus-beli kielégíthetőségi tételbizonyítókhoz. A válaszhalmaz-programozás előnye más tervkészítőkkel szemben a rugalmassága: a tervkészítő operátorokat és a kényszereket logikai programokkal fejezi ki, és így azok nem kötődnek a konkrét tervkészítési formalizmus korlátozott formátumához. A válaszhalmaz-programozás hátránya ugyanaz, mint minden ítéletkalkulus szintű technikáé: ha az univerzumban túl sok objektum létezik, egy exponenciális lassulással kell számolnunk.

7.3. Körülírás és alapeseti logika

Két példát láttunk, ahol látszólag természetes következtetési folyamatok megsértik a logika a 7. fejezetben bebizonyított **monotonitás** (**monotonicity**) tulajdonságát.[14] Az első példában egy szemantikus hálóban egy kategória összes tagja által megörökölt tulajdonságot egy alkategóriára vonatkozó specifikusabb információ felülírhatja. A másik példában a zárt világ feltételezésből származtatott negált literálokat pozitív literálok hozzáadása szintén felülbírálja.

Ha egy kicsit magunkba nézünk, rájövünk, hogy a monotonitás ilyen sérülései a józan ész következtetésben igen elterjedtek. Úgy tűnik, az emberek gyakran „elhamarkodott következtetéseket" vonnak le. Ha az utcán parkoló kocsit látunk, mindenki elhiszi, hogy a kocsinak négy kereke van, holott csak három kerék látszik (ha valaki a negyedik kerék létezésében kételkedik, akkor vegye fontolóra azt a kérdést, vajon a három látható kerék valódi-e, vagy csupán a valódinak egy papírmása.) A valószínűség-elmélet nyilván igazolhatja, hogy a negyedik kerék nagy valószínűséggel létezik, az emberek többségében az a lehetőség, hogy a gépkocsinak esetleg nincs négy kereke, egyáltalán fel sem merül, *hacsak valami új tényállás erre nem derít fényt*. A négy kerék konklúzióját tehát *alapesetként* kezeljük, a kételkedésre okot adó tények hiányában. Ha új tények érkeznek – észrevesszük például, hogy a tulajdonos a kereket elviszi, és a gépkocsi egy emelőn áll – akkor a konklúziót vissza kell vonni. Az ilyen következtetésről azt mondjuk, hogy **nemmonoton** (**nonmonoton**), mert a hiedelemek halmaza idővel, ahogy az új evidenciák megjelennek, nem növekszik monoton módon. Az ilyen viselkedés leírására **nemmonoton logiká**kat (**nonmonoton logic**s) fejlesztettek ki, módosított igazságdefinícióval és vonzatrelációval. Két ilyen intenzíven tanulmányozott logikát fogunk megnézni: a körülírást és az alapeseti logikát.

A **körülírás**t (**circumscription**) a zárt világ feltételezés erősebb és precízebb változatának lehetne tekinteni. Az ötlet, hogy konkrét predikátumokat specifikálunk, amelyekről feltételezzük, hogy „amennyire csak lehetséges hamisak" – azaz minden objektumra hamisak, kivéve azokat, amelyekre tudjuk, hogy igazak. Tegyük fel például, hogy ki akarjuk jelenteni azt az alapértelmezett szabályt, hogy a madarak tudnak repülni. Vezessük be az $Abnormális_1(x)$ predikátumot, és írjuk fel:

$$Madár(x) \land \neg Abnormális_1(x) \Rightarrow Repül(x)$$

[14] Emlékezzünk vissza, hogy a monotonitás megköveteli, hogy minden vonzatállítás igaz maradjon, amikor a tudásbázishoz új állításokat adunk hozzá. Azaz ha $TB \models \alpha$, akkor $TB \land \beta \models \alpha$.

Created by XMLmind XSL-FO Converter.

Ha azt mondjuk, hogy az *Abnormális*$_1$-et **körül kell írni** (**circumscribed**), a körülíró következtető feltételezheti, hogy ¬*Abnormális*$_1$(*x*), hacsak az *Abnormális*$_1$(*x*)-et nem fogja igaznak találni. Ez lehetővé teszi, hogy a *Repül*(*Tweety*) konklúziót levonjuk a *Madár*(*Tweety*) premissza alapján, azonban a konklúzió csak addig igaz, amíg az *Abnormál*(*Tweety*)-t ki nem jelentjük.

A körülírást a **modellpreferencia-logika** (**model preference logic**) egy példájának is tekinthetjük. Az ilyen logikákban egy állítás vonzatrelációban van (alapeseti státuson), ha igaz a tudásbázis minden *preferált* modelljében, ellentétben a klasszikus logika igazságkövetelményével, hogy minden modellben legyen igaz. Körülírás esetében egy modell egy másiknál preferáltabb, ha kevesebb abnormális objektuma van.[15] Nézzük meg, hogy ez az ötlet hogyan működik szemantikus hálóban fellépő többszörös öröklődés esetén. A többszörös öröklődés standard példája az ún. „Nixon-gyémánt”.[16] Az eredete az a megfigyelés, hogy Richard Nixon egy kvéker vallási szektához tartozott (és így alapesetben pacifista) és republikánus is volt (és így alapesetben éppen nem pacifista). Ezt az alábbi módon írhatjuk fel:

$$Republikánus(Nixon) \wedge Kvéker(Nixon)$$

$$Republikánus(x) \wedge \neg Abnormális_2(x) \Rightarrow \neg Pacifista(x)$$

$$Kvéker(x) \wedge \neg Abnormális_3(x) \Rightarrow Pacifista(x)$$

Ha az *Abnormális*$_2$-t és az *Abnormális*$_3$-at körülírjuk, két preferált modellt kapunk: az egyikben igaz az *Abnormális*$_2$(*Nixon*) és a *Pacifista*(*Nixon*), a másikban igaz az *Abnormális*$_3$(*Nixon*) és a ¬*Pacifista*(*Nixon*). A körülíró következtető rendszer tehát megfelelően tudatlannak mutatkozik a pacifista Nixon ügyében. Ha ezentúl szeretnénk kijelenteni, hogy a vallásos meggyőződések elsőbbséget élveznek a politikai nézetekkel szemben, a **prioritásos körülírás** (**prioritized circumscription**) formalizmusát használhatjuk, hogy azoknak a modelleknek adjunk elsőbbséget, ahol az *Abnormális*$_3$ minimálizálva van.

Az **alapértelemzett logika** (**default logic**) egy olyan formalizmus, ahol **alapértelmezett szabály**okat (**default rule**s) lehet írni feltételes, nemmonoton következtetések generálásához. Egy alapértelmezett szabály a következőképpen néz ki:

$$Madár(x) : Repül(x) / Repül(x)$$

A szabály jelentése, hogy ha a *Madár*(*x*) igaz, és ha a *Repül*(*x*) a tudásbázissal konzisztens, akkor a *Repül*(*x*)-et alapértelmezésben lekövetkeztethetjük. Általánosságban egy alapértelmezett szabály formája a:

$$P : J_1, \ldots, J_n / C$$

ahol *P*-t előfeltételnek, *C*-t konkluziónak és a J_i-ket igazolásoknak nevezik – ha ezek közül bármelyik igazolhatóan hamis, a konklúziót levonni nem szabad. A *C*-ben és a J_i-ben előforduló minden változónak *P*-ben is meg kell jelennie. A Nixon-gyémánt példáját alapértelmezett logikában egy ténnyel és két alapértelmezett szabállyal fejezhetjük ki:

$$Republikánus(Nixon) \wedge Kvéker(Nixon)$$

$$Republikánus(x) : \neg Pacifista(x) / \neg Pacifista(x)$$

$$Kvéker(x) : Pacifista(x) / Pacifista(x)$$

Egy alapértelmezett szabály jelentésének interpretálásához az alapértelmezett elmélet **kiterjesztés**ét (**extension**) kell definiálnunk, az elmélet konklúzióinak maximális halmazaként. Egy *S* kiterjesztés tehát az eredetileg ismert tényekből és az alapértelmezett szabályokból levont konklúzióhalmazból áll úgy, hogy *S*-ből semmilyen más konklúziót levonni már nem lehet, és *S*-beli alapértelmezett konklúzió minden igazolása *S*-sel konzisztens. A körülírásban tárgyalt preferált modellekhez hasonlóan a Nixon-gyémántra két lehetséges kiterjesztésünk van: egy, amelyben pacifista és egy, amelyben nem az. Léteznek prioritásos sémák, ahol egyes alapértelmezett szabályoknak precedenciát lehet biztosítani más szabályokkal szemben, és ezzel a nem egyértelmű helyzeteket valamelyest fel lehet oldani.

[15] A zárt világ feltételezésnél egy modell egy másiknál preferáltabb, ha kevesebb igaz atomja van – azaz a preferált modellek **minimálmodell**ek. A **CWA** és a definit klóz tudásbázis között létezik természetes kapcsolat, mert az ilyen tudásbázisban az előrecsatolt következtetés révén elért fix pont az egyértelmű minimálmodell (4.2. szakasz - Előre- és hátrafelé láncolásrészben).

[16] A „gyémánt” elnevezés a két úton futó öröklődési lánc gráfszerű alakjától származik. (*A ford.*)

Created by XMLmind XSL-FO Converter.

1980 óta, amikor is a nemmonoton logikákat első ízben javasolták, igen nagy előrehaladás történt a matematikai tulajdonságuk megértésében. A késői 1990-es évektől kezdve a logikai programozáson alapuló gyakorlati rendszerek ígéretesnek bizonyultak a tudásreprezentációs eszközök szerepében. Vannak azonban még meg nem válaszolt kérdések. Így például ha „a gépkocsiknak négy kerekük van” hamis, mit is jelent, ha egy ilyen állítás a tudásbázis része? Milyen az alapértelmezett szabályok egy jó halmaza? Ha minden szabályról külön-külön nem tudjuk eldönteni, hogy a tudásbázisunkhoz tartozik-e, akkor igen komoly problémánk van a modularitás hiányával. Végül, hogyan lehet alapértelmezett státussal rendelkező hiedelmeket a döntéshozatalban felhasználni? Ez talán az alapértelmezett következtetés legnehezebb problémája. A döntések sokszor kompromiszszumokkal járnak együtt, és így szükség van különböző cselekvések eredményeként megjelenő hiedelmek *erősségét* összehasonlítani. Azokban az esetekben, amikor ugyanilyen jellegű döntéseket sokszor hozunk meg, lehetőség adódik, hogy az alapértelmezett szabályokat „küszöb-valószínűségi” állításokként értelmezzük. Például „a fékjeim rendben vannak” alapértelmezett szabály valójában azt jelenti, hogy „annak a valószínűsége, hogy a fékjeim rendben vannak, feltéve, hogy más információm nincs, elegendően magas ahhoz, hogy az optimális döntés számomra az, hogy induljak útnak a fékek ellenőrzése nélkül”. Amikor a döntés kontextusa megváltozik – például amikor egy súlyosan megrakott teherkocsit vezetünk egy lejtős hegyi úton –, az alapértelmezett szabály hirtelen alkalmatlanná válik, akkor is, ha semmilyen új evidenciánk nincs, ami azt sugallná, hogy a fékek hibásak. Az ilyen mérlegelések néhány kutatót arra vezettek, hogy mérlegeljék, hogyan ágyazzák be az alapértelmezett következtetést a valószínűség-elméletbe.

8. Igazság-karbantartó rendszerek

Az előbbi részben arról érveltünk, hogy egy tudásreprezentációs rendszer által levont sok következtetésnek csupán alapértelmezett és nem tökéletesen biztos státusa lehet. Az ilyen kikövetkeztetett konklúziókból néhány szükségszerűen hibásnak fog bizonyulni, és új információ fényében vissza kell vonnunk. Ez a **hiedelemrevízió** (**belief revision**) folyamata.[17] Tegyük fel, hogy a tudásbázis tartalmazza P állítást – az előrecsatolt algoritmussal levont alapértelmezett konklúziót vagy egyszerűen egy hibás feltételezést –, és szándékunkban áll végrehajtani az ÁLLÍT (TB, $\neg P$)-t. Hogy az ellentmondást elkerüljük, először végre kell hajtani a VISSZAVON (TB, P)-t. Ez elég egyszerűnek tűnik. Probléma akkor jelentkezik, ha P-ből *további* állításokat következtettünk le és jegyeztünk fel a tudásbázisba. A $P \Rightarrow Q$ implikációt például felhasználtuk a Q tudásbázishoz való hozzáadásához. A nyilvánvaló „megoldás” – a P-ből kikövetkeztetett összes állítás visszavonása – kudarcba fut, mert ilyen állításoknak P-n túlmenően más igazolásuk is lehet. Ha R és $R \Rightarrow Q$ is bekerült a tudásbázisba, akkor Q-t mégsem kell eltávolítanunk. Az **igazság-karbantartó rendszer**eket (**truth maintenance system**s) ilyen bonyodalmakat kezelésére tervezték.

Az igazság-karbantartás egyszerű módja, hogy nyomon követjük, ahogyan sorban hozzáadjuk a mondatokat a tudásbázishoz, a mondatok P_1-től P_n-ig történő megszámozásával. Amikor a VISSZAVON (TB, P_i) hívás megtörténik, a rendszert a P_i hozzáadása előtti állapotába állítjuk vissza, a P_i-t és a P_i-ből levont összes következtetést visszavonva. Ha szükséges, akkor a Pi_{+1} és a P_n közötti mondatok ismét hozzáadhatók. Ez egyszerű megoldás, és garantálja, hogy a tudásbázis konzisztens lesz, de azt jelenti, hogy a P_i visszavonása $n - i$ állítás visszavonását és újbóli kijelentését, valamint ezen állításokból történő következtetések visszavonását és újbóli lefuttatását igényli. Olyan rendszerekben, ahol sok tény kerül hozzáadásra, az ilyen megoldás nem praktikus.

Hatékonyabb megközelítés az igazolásalapú igazság-karbantartó rendszer vagy **JTMS** (**justification-based truth maintenance system**). Egy JTMS-ben a tudásbázis minden mondatát olyan **igazolás**sal (**justification**) látják el, amely meghatározza azokat a mondatokat, amelyekből lekövetkeztették azokat. Például ha a tudásbázis már tartalmazza a $P \Rightarrow Q$-t, akkor az ÁLLÍT (P) hatására a Q-*t* hozzáadjuk, az $\{P, P \Rightarrow Q\}$ igazolással. Általánosságban bizonyos mondatoknak egynél több igazolása lehet. Az igazolások felhasználhatók hatékony visszavonásokra. A VISSZAVON (P) hívást követően, a JTMS pontosan azokat a mondatokat fogja a tudásbázisból eltávolítani, amelyeknél P minden igazolás része. Így, ha egy Q mondatnak a $\{P, P \Rightarrow Q\}$ lenne az egyetlen igazolása, akkor törlődne; ha az $\{P, P \vee R \Rightarrow Q\}$ igazolása is létezne, még mindig visszavonásra kerülne, de ha az $\{R, P \vee R \Rightarrow Q\}$ igazolása is létezne, akkor megmaradna. Ily módon a P visszavonásához szükséges idő csakis azon múlik, hogy P-ből hány állítást vezettünk le, és nem azon, hogy P hozzáadása után hány állítás került még be a tudásbázisba.

[17] A hiedelemrevíziót sokszor a **hiedelemfrissítés**sel (**belief update**) állítják szembe, amelyről akkor van szó, amikor a tudásbázis inkább a világbeli változások hatására, és nem a rögzített világról befutó új információ hatására kerül felülvizsgálásra. A hiedelemfrissítés összekapcsolja a hiedelemrevíziót az időre és a változásra vonatkozó következtetéssel. Kapcsolatban áll a 15. fejezetben leírt **szűrés** (**filtering**) folyamatával.

Created by XMLmind XSL-FO Converter.

A JTMS feltételezi, hogy az egyszer figyelembe vett mondatokat máskor is használni fogjuk, így ahelyett, hogy törölnénk a tudásbázisból, amikor elvesztik az összes igazolását, csak megjelöljük őket, mint *kintlevőket*. Ha egy későbbi állítás visszaállítja az igazolások egyikét, akkor a mondat ismét *bentlevő* jelölést kap. Ily módon a JTMS megtartja a használt összes következtetési láncot, és nem kell újra lekövetkeztetnie azokat a mondatokat, amelyek igazolása ismét érvényessé válik.

Azon túl, hogy a TMSek a hibás információt visszavonják, felhasználhatók a többszörös hipotetikus helyzet elemzésének a felgyorsításához is. Tegyük fel, hogy a Román Olimpiai Bizottság helyszíneket keres a 2048-ban Romániában megrendezendő olimpiai játékokhoz az úszó, a könnyűatlétikai és a lovas versenyszámok számára. Legyen az első hipotézis a következő: *Helyszín*(*Úszás*, *Piteşti*), *Helyszín*(*Könnyűatlétika*, *Bukarest*) és *Helyszín*(*Lovasverseny*, *Arad*). A választás logisztikai következményeinek kiderítéséhez, és így a helyszínek kívánatosságának meghatározásához igen sok következtetés szükséges. Ha figyelembe akarjuk venni a *Helyszín*(*Könnyűatlétika*, *Nagyszeben*)-t, a TMS elkerüli annak a szükségességét, hogy újra nullából induljunk ki. Egyszerűen viszavonjuk a *Helyszín*(*Könnyűatlétika*, *Bukarest*)-t, és hozzáadjuk a *Helyszín*(*Könnyűatlétika*, *Nagyszeben*)-t, és a TMS a szükséges revíziókat magára vállalja. A Bukarestre vonatkozó következtetési láncok Nagyszeben esetén újrafelhasználhatók, feltéve, hogy a konklúziók ugyanazok lesznek.

Egy **feltételezésalapú igazság-karbantartó rendszer**t (**assumption-based truth maintenance system**, **ATMS**) arra terveztek, hogy ezt a fajta hipotetikus világok közötti kontextusátkapcsolást kifejezetten hatékonnyá tegye. Egy JTMS-ben az igazolások karbantartása lehetővé teszi, hogy az egyik állapotból gyorsan egy másikba lépjünk át néhány visszavonással és kijelentéssel, de minden időpillanatban csak egyetlen állapotot reprezentálunk. Egy ATMS az *összes* állapotot egyszerre reprezentálja, amivel valaha is foglalkoztunk. Míg egy JTMS egyszerűen *bent*- vagy *kintlevőnek* jelöli a mondatokat, egy ATMS nyomon követi, minden egyes mondatra, hogy a mondatot mely feltételezések tennék igazzá. Más szavakkal minden egyes mondatnak van egy címkéje, amely a feltételezés halmazokat tartalmazó halmazból áll. A mondat csak azon esetben áll fenn, amikor egy feltételezés halmaz összes feltételezése fennáll.

Az igazság-karbantartó rendszerek mechanizmust biztosítanak **magyarázat**ok (**explanation**s) generálásához is. Technikailag, *P* mondat magyarázatát olyan *E* mondatok halmazával definiáljuk, melyekre *E* maga után vonja *P*-t. Ha *E* mondatai igazak, akkor *E* egyszerűen P igazolásához egy szükségszerű alap. Azonban a magyarázat részei lehetnek **feltételezés**ek (**assumption**s) is – olyan állítások, amelyek nem igazak, de ha igazak lennének, elegendők lennének *P* igazolásához. Lehetséges például, hogy nincs elég tény bebizonyítani, hogy az autó nem fog indulni, de egy jó magyarázat része lehet az elromlott akkumulátor. Ez, összekapcsolva a gépkocsik működésére vonatkozó tudással, elegendő, hogy a megfigyelt jelenséget megmagyarázza. Az esetek többségében egy minimális *E* magyarázatot fogunk preferálni, azaz olyan magyarázatot, amelynek nincs olyan megfelelő részhalmaza, amelyik szintén egy magyarázat lenne. Egy ATMS a „gépkocsi nem indul” problémához magyarázatokat generál azáltal, hogy tetszőleges sorrendben feltételezéseket tesz (mint például „benzin a tankban” vagy „hibás az akkumulátor”), akkor is, ha azok egy része ellentmondásos. Elég majd a „gépkocsi nem indul” állítás címkéjét megnézni, hogy az állítást igazoló feltételezések halmazát megismerjük.

Az igazság-karbantartó rendszereket megvalósító algoritmusok egy kicsit komplikáltak, és itt nem tárgyaljuk őket. Az igazság-karbantartási probléma komplexitása legalább olyan nagy, mint az ítéletlogikai következtetésé – azaz NP-teljes. Ezért nem várható el az, hogy az igazság-karbantartás csodaszer legyen. Ugyanakkor, ha körültekintően használják, a TMS-rendszer lényegesen javíthat egy logikai rendszer komplex környezeteket és hipotéziseket kezelő képességén.

9. Összefoglalás

Ez a fejezet eddig a könyv legrészletesebb fejezete. Azzal, hogy sokféle tudás reprezentálásának részleteivel foglalkozunk, remélhetően érzékeltetni tudtuk az olvasóval, hogy hogyan hozhatók létre valódi tudásbázisok. A lényegi témák az alábbiak:

- Nagyméretű tudásreprezentáció egy általános rendeltetésű ontológiát igényel, a specifikus problématerületek szervezéséhez és az egymással való összekapcsolásához.
- Egy általános ontológiának a tudás széles választékát kell lefednie, és elvben képesnek kell lennie bármilyen problématerület kezelésére.
- Bemutattunk egy **felső ontológiá**t (**upper ontology**), amely a kategóriákon és az eseménykalkuluson alapul. Foglalkoztunk strukturált objektumokkal, az idővel és a térrel, a változással, folyamatokkal, szubsztanciákkal és hiedelmekkel.

Created by XMLmind XSL-FO Converter.

- A cselekvéseket, az eseményeket és az időt vagy szituációkalkulusban, vagy az olyan nagyobb kifejezőerejű reprezentációkban, mint az eseménykalkulus vagy a folyó esemény kalkulus fejezhetjük ki. Az ilyen reprezentációk révén egy ágens cselekvési terveket konstruálhat logikai következtetés segítségével.

- Egy ágens mentális állapotait a hiedelmeket jelentő füzérekkel lehet reprezentálni.

- Az internetes bevásárlási problématerület részletes elemzését mutattuk be általános ontológia bevetésével, megmutatva, hogy egy bevásárló ágens hogyan használhatja a problématerületi tudását.

- A kategóriahierarchia szervezéséhez speciális rendeltetésű rendszereket terveztek, mint például a **szemantikus hálók** (**semantic nets**) és a **leíró logikák** (**description logics**). A következtetés fontos esete az **öröklődés** (**inheritance**), mely révén az objektumok tulajdonságait a kategóriatagságukból ki lehet következtetni.

- A logikai programokban a **zárt világ feltételezést** (**closed-world assumption**) implementáljuk, hogy rengeteg negatív információ megadását elkerülhessük. A legjobb, ha ezt alapeseti helyzetnek tekintjük, amit további információ felülírhat.

- A **nemmonoton logikák** (**nonmonotonic logics**), amilyen például a **körülírás** (**cirumscription**) vagy az **alapeseti logika** (**default logic**), az általános alapeseti következtetés leírására készültek. Nemmonoton következtetést a **válaszhalmaz-programozás** (**answer set programming**) lényegesen gyorsítja, hasonlóan ahhoz, ahogy a `WALKSAT` meggyorsítja az ítéletlogikai következtetést.

- Az **igazság-karbantartó rendszerek** (**truth maintenance systems**) hatékonyan kezelik a tudásfelfrissítést és a tudásrevíziót.

9.1. Irodalmi és történeti megjegyzések

Hihetőnek hangzik (Briggs, 1985), hogy a formális tudásreprezentáció kezdete a sásztrai szanszkrit nyelvtanáról alkotott klasszikus indiai elméletekkel kezdődött az i. e. első évezredben.[18] A nyugati világban az ókori görög matematikusok által használt definíciók tekinthetők a tudásreprezentáció legkorábbi megjelenésének. Tény, hogy a műszaki szóhasználat vagy egy mesterséges nyelv kialakítása, bármilyen területről van is szó, tekinthető a tudásreprezentáció egyfajta formájának.

Az MI-ben a reprezentációkról folytatott korai viták inkább a *„problémareprezentációra”,* mint a *„tudásreprezentációra”* irányultak [lásd például a misszionáriusok és kannibálok problémájáról szóló Amareltől származó elemzést (Amarel, 1968)]. Az 1970-es években az MI-ben a hangsúly a „szakértő rendszerek” („tudásalapú rendszereknek” is nevezett) fejlesztésén volt, amelyek a megfelelő problématerületi tudás birtokában, szűkebb definiált feladatok esetén, az emberi szakértők hatékonyságát megközelítették, vagy akár túl is szárnyalták. Az első szakértő rendszer, a `DENDRAL`, például a tömegspektrométer (a szerves vegyi anyagok struktúraelemzését segítő berendezés) kimeneti adatait képes volt szakértő vegyészek szintjén interpretálni (Feigenbaum és társai, 1971; Lindsay és társai, 1980). Bár a `DENDRAL` sikere meghatározó volt abban, hogy az MI kutatói ráébredjenek a tudásreprezentáció fontosságára, a benne alkalmazott reprezentációs formalizmusok igen specifikusak voltak, és kifejezetten a vegyészeti problématerülethez alakították ki azokat. Idővel a kutatók a szabványosított tudásreprezentációs formalizmusok és az ontológiák felé fordultak, hogy így a korábban még fel nem tárt területeken alkalmazandó új szakértő rendszerek előállítási nehézségeit csökkentsék. Eme kutatás révén olyan területre merészkedtek ki, amit korábban a tudományfilozófusok és a nyelvészettel foglalkozó filozófusok műveltek. Az elméletek „munkába” állításával az MI területén kialakult új diszciplína sokkal mélyebb és gyorsabb előrehaladáshoz vezetett a korábbi időkhöz képest, amikor ezek a problémák kizárólag a filozófia tárgyát képezték (bár időnként előfordult a spanyolviasz ismételt felfedezése is).

Átfogó taxonómiák, illetve osztályozások megalkotása az ókori időkig nyúlik vissza. Az osztályozási és kategorizálási sémák fontosságát nyomatékosan hangsúlyozta Arisztotelész. Halála után hallgatói révén megszerkesztett *Organon,* amely logikai munkák gyűjteménye volt, tartalmazta a *Kategóriák* c. tanulmányt is, amelyben Arisztotelész megkísérelt kimerítő osztályozást adni arra, amit ma felső ontológiának nevezünk. Alacsonyabb szintű osztályozás céljára bevezette a **faj** (**genus**) és a **fajta** (**species**) fogalmakat, bár e fogalmak akkor nélkülözték a ma velük asszociált precíz és specifikusan biológiai jelentést. A biológiai osztályozás mai rendszerét, beleértve a „binomiális nómenklatúrát” (technikai értelemben a faj-, fajtaalapú osztályozást) is, Carolus Linnaeus vagy másképpen Carl von Linné (1707–1778), svéd biológus találta fel. A természetes

[18] A szanszkrit nyelv több változata közül a sásztrai szanszkrit a vallás és a tudomány nyelve. (*A ford.*)

Created by XMLmind XSL-FO Converter.

fajtákkal és a pontatlan kategóriahatárokkal kapcsolatos problémákkal többek közt Wittgenstein, Quine, Lakoff és Schwartz foglalkozott (Wittgenstein, 1953; Quine, 1953; Lakoff, 1987; Schwartz, 1977).

Az érdeklődés a nagyméretű ontológiák iránt növekszik. A CYC projekt (Lenat, 1995; Lenat és Guha, 1990) kapcsán egy kb. 60 000 tényt és 6000 fogalmat tartalmazó felső ontológiát hoztak nyilvánosságra, és egy sokkal nagyobb globális ontológiát szabadalmaztattak. Az IEEE létrehozta a P1600.1 albizottságot – a Standard Upper Ontology Working Groupot, az Open Mind Initiative viszont több mint 7000 internetfelhasználót kért fel, hogy több mint 400 000 tényt vigyenek be közhasználatú fogalmakról. A weben alakulóban vannak az olyan standardok, mint az RDF, az XML és a szemantikus háló (Berners-Lee és társai, 2001), ámbár azokat még nem használják széles körben. A *Formal Ontology in Information Systems* (FOIS) konferenciákon sok érdekes cikket publikálnak mind területspecifikus, mind általános ontológiákról.

A jelen fejezetben kialakított taxonómia a szerzőktől származik, és részben azokon a tapasztalatokon alapul, melyeket a CYC projektből nyertek, részben Hwang, Schubert és Davis munkájára támaszkodva (Hwang és Schubert, 1993; Davis, 1990). Józan ész reprezentációs projekt inspiráló vitái Hayes *The Naive Physics Manifestó*jában jelentek meg (Hayes, 1978; 1985b).

Az időt, a változást, a cselekvéseket és az eseményeket mind az MI-ben, mind a filozófiában és az elméleti számítástudományban intenzíven tanulmányozták. A legrégebbi megközelítés a **temporális logika** (**temporal logic**), ami egy speciálizált logika, ahol minden modell egy teljes trajektóriát ír le az (általában lineáris vagy elágazó) időben, a statikus relációs struktúrák helyett. A logika **modális operátor**okat (**modal operator**s) tartalmaz, amelyeket állításokra alkalmaznak. A $\Box p$ azt jelenti, hogy „*p* a jövőben minden pillanatban igaz lesz”, a $\Diamond p$ pedig azt, hogy „*p* a jövőben valamikor igaz lesz”. A temporális logika tanulmányozását az ókori Görögországban Arisztotelész, valamint a Megara és a sztoikus iskola kezdeményezte. A modern időkben elsőnek Findlay vetette fel az időről való következtetés „formális kalkulusának” a gondolatát (Findlay, 1941), azonban a legnagyobb hatásúnak Arthur Prior munkássága számít (Prior, 1967). Jó tankönyvek Rescher és Urquhart, valamint van Benthem munkái (Rescher és Urquhart, 1971; van Benthem, 1983).

Az elméleti számítástudomány kutatói régóta érdeklődtek az iránt, hogy számítási cselekvések szekvenciájaként értelmezett programok tulajdonságait hogyan lehetne formalizálni. A modális logikát számítógépes programokról való következtetésre Burstall vezette be (Burstall, 1974). Rövidesen utána Vaughan Pratt a **dinamikus logiká**t (**dynamic logic**) dolgozta ki (Pratt, 1976), amelyben a modális operátorok programok vagy más cselekvések hatását jelzik (lásd még Harel, 1984). Így például ha dinamikus logikában α egy program neve, akkor „$[\alpha]\ p$” azt jelenti, hogy „*p* igaz lesz a világ minden olyan állapotában, amely az α programnak a jelenlegi állapotból való indításából származik”. Az „$\langle\ \alpha\rangle\ p$” pedig azt jelenti, hogy „*p* igaz lesz a világ legalább egy olyan állapotában, amely az α program mostani állapotából való indításából származik”. Programok konkrét elemzésére a dinamikus logikát Fischer és Ladner használta (Fischer és Ladner, 1977). Pnueli programokról való következtetésre klasszikus temporális logikát javasolt (Pnueli, 1977).

A temporális logikában az időt általában a nyelv modellelméletébe ágyazzák be. Az MI-ben a tendencia az volt, hogy az időpontokra és az eseményekre vonatkozó axiómákat explicit módon a tudásbázisban írjuk fel, anélkül hogy az időnek a logikában valamilyen speciális státust adnánk. Az ilyen megközelítés egyes esetekben nagyobb rugalmasságot és áttekinthetőséget biztosít. Ráadásul az elsőrendű logikában kifejezett temporális logika nagyobb eséllyel integrálható az ebben a formalizmusban felhalmozott más tudásanyaggal.

Az MI-ben az idő és a cselekvés legkorábbi kezelő apparátusa John McCarthy szituációkalkulusa volt (McCarthy, 1963). A QA3 volt az első MI-rendszer, amely nagyban felhasználta a cselekvésre vonatkozó általános rendeltetésű elsőrendű logikai következtetést (Green, 1969b). Kowalski fejlesztette ki az állítások reifikálását a szituációkalkuluson belül (Kowalski, 1979b).

A keretproblémát, mint olyant, elsőként McCarthy és Hayes ismerték fel (McCarthy és Hayes, 1969). Számos kutató a problémát megoldhatatlannak tartotta az elsőrendű logikán belül, és ez intenzív kutatáshoz vezetett a nemmonoton logikák irányába. A filozófusok Dreyfustól (Dreyfus, 1972) Crockettig (Crockett, 1994) a keretproblémát az egész MI-vállalkozás szükségszerű kudarca egyik tünetének tartották. A reprezentációs keretprobléma részleges megoldása követő-állapot axiómák segítségével Ray Reitertől származik (Reiter, 1991). A következtetési keretprobléma megoldása Holldobler és Schneeberger munkájára vezethető vissza (Holldobler és Schneeberger, 1990), amit most folyó esemény kalkulusnak nevezünk (Thielscher, 1999). Jelen fejezetben bemutatott elemzés Lin és Reiter, valamint Thielscher elemzésein alapul (Lin és Reiter, 1997; Thielscher, 1999). A szituációkalkulusban a cselekvésről való következtetés teljes, korszerű tárgyalását Shanahan és Reiter könyvei adják (Shanahan, 1997; Reiter, 2001b).

Created by XMLmind XSL-FO Converter.

A keretprobléma részleges megoldásával újra fellángolt az érdeklődés a cselekvésekről való következtetés deklaratív megközelítései iránt, amelyeket az 1970-es évek elejétől a speciális tervkészítő rendszerek háttérbe szorítottak (lásd 11. fejezet). A **kognitív robotika** (**cognitve robotics**) zászlaja alatt lényeges előrehaladás történt az idő és a cselekvés logikai reprezentációja terén. A GOLOG programozási nyelv a cselekvések és a tervek kifejezésére a logikai programozás teljes kifejező erejét felhasználja (Levesque és társai, 1997a). A nyelvet a parallel cselekvések (Giacomo és társai, 2000), a sztochasztikus környezetek (Boutilier és társai, 2000) és az érzékelés (Reiter, 2001a) kezelésére is kiterjesztették.

Az eseménykalkulust Kowalski és Sergot vezették be folytonos idő kezelésésre (Kowalski és Sergot, 1986). Ennek számos változata is született (Sadri és Kowalski, 1995). Shanahan ad róla egy jó és rövid áttekintést (Shanahan, 1999). Ugyanerre a célra James Allen vezette be az időintervellumokat (Allen 1983, 1984) azzal érvelve, hogy azok a szituációknál természetesebb eszközt jelentenek a kiterjedő és a konkurrens események leírására. Peter Ladkin „konkáv" időintervallumokat (intervallumok szakadásokkal, lényegében a közönséges „konvex" időintervallumok uniói) vezetett be, és az időreprezentációhoz alkalmazta a matematikai absztrakt algebra módszertanát (Ladkin, 1986a; Ladkin, 1986b). Allen szisztematikusan vizsgálta az idő reprezentálásához ma hozzáférhető technikák széles választékát (Allen, 1991). Shoham leírta az események reifikációját, és e célból egy új sémát vezetett be (Shoham, 1987). Lényeges egybeesés lelhető fel az e fejezetben említett eseményalapú ontológia és Donald Davidson filozófus eseményelemzése között (Davidson, 1980). Hasonló jellegű a folyadékok Pat Hayestől származó ontológiája is (Hayes, 1985a).

A szubsztanciák ontológiai státusának kérdése hosszú történet. Platón szerint a szubsztanciák a fizikai objektumoktól teljesen eltérő absztrakt entitások. A $Vaj_3 \in Vaj$ helyett ő inkább $BólVan(Vaj_3, Vaj)$-at mondana. Ez elvezet a szubsztanciák egy olyan hierarchiájához, ahol például *SózatlanVaj* a *Vaj*-nál specifikusabb szubsztancia. A jelen fejezetben elfogadott álláspontot, miszerint a szubsztanciák objektumok kategóriái, Richard Montague vette védelmébe (Montague, 1973). Ezt az álláspontot a CYC projektben is elfogadták. Ezzel szemben egy komoly, ám nem kivédhetetlen támadást intézett Copeland (Copeland, 1993). A fejezetben említett alternatív megközelítést, ahol a vaj egy egyedi objektum, ami az univerzum összes vajszerű objektumából áll, eredetileg Les´niewski lengyel logikus javasolta (Les´niewski, 1916). **Mereológiá**ja (**mereology**) (a név a „rész" görög nevéből származik) a rész-egész relációt matematikai halmazelmélet helyettesítésére használja a célból, hogy az olyan absztrakt entitásokat, mint a halmazok, eltüntesse. A mereológia olvashatóbb változatát Goodman és Leonard adták (Leonard és Goodman, 1940). Goodman *The Structure of Appearance* c. művében ezeket az ötleteket különböző problémák megoldására alkalmazza a tudásreprezentációk területén (Goodman, 1977). Bár a mereologikus megközelítés néhány aspektusa nehézkes – például a rész-egész reláción alapuló külön öröklődési mechanizmusra van szükség –, a megközelítést Quine is támogatja (Quine, 1960). A tudásreprezentációban való használatának kimerítő elemzését Harry Bunt adta meg (Bunt, 1985).

A mentális objektumokat és állapotokat a filozófiában és az MI-ben is intenzíven tanulmányozták. A filozófiában a **modális logika** (**modal logic**) a tudásról való következtetés klasszikus eszköze. Modális logika az elsőrendű logikát bizonyos modális operátorokkal, mint a *B* (hisz, *believes*) és a *K* (tud, *knows*) kiterjeszti, ezek az operátorok nem a termekre, hanem az *állításokra* vonatkoznak. A modális logika bizonyításelmélete a behelyettesítést a modális kontextusra korlátozza, és így referenciális átláthatatlansághoz vezet. A tudás modális logikáját Jaakko Hintikka találta fel (Hintikka, 1962). A modális logika szemantikáját Saul Kripke definiálta **lehetséges világ**ok (**possible world**s) segítségével (Kripke, 1963). Durván fogalmazva egy világ lehetséges egy ágens részére, ha konzisztens mindazzal, amiről az ágensnek tudomása van. Ebből már meg lehet fogalmazni a *K* operátorra vonatkozó következtetési szabályokat. Robert C. Moore (Moore, 1980; 1985a) kapcsolatba hozza tudás modális logikáját azzal a következtetési stílussal, amely az elsőrendű logikában közvetlen módon a lehetséges világokra hivatkozik. A modális logika ijesztően titokzatos terület lehet, azonban elosztott számítógépes rendszereknél az információról való következtetést illetően jelentős alkalmazásra talált. A modális megközelítéshez részletes bevezetőt Fagin *Reasoning about Knowledge* c. könyve ad (Fagin, 1995). A kétévente megrendezett *Theoretical Aspects of Reasoning About Knowledge* (TARK) konferenciák mutatják be a tudáselmélet alkalmazását az MI-ben, a gazdasági tudományokban és az elosztott rendszerekben.

A mentális objektumok szintaktikai elméletét részletesen először Kaplan és Montague tanulmányozták, akik megmutatták, hogy az elmélet, megfelelő elővigyázatosság hiányában, paradoxonokhoz vezet (Kaplan és Montague, 1960). Mivel ez a hiedelmeket egy fizikai rendszer konfigurációival kapcsolja össze, és ehhez rendelkezik egy természetes modellel egy emberi agy vagy egy számítógép formájában, ez az elmélet az MI területén az utóbbi években igen népszerű volt. Az elméletet korlátos hatékonyságú következtető gépek leírására Konolige és Haas használta, Morgenstern viszont megmutatta, hogyan használható előfeltételek ábrázolására tervkészítés esetén (Konolige, 1982; Haas, 1986; Morgenstern, 1987). A megfigyelési cselekvések tervezésének a 13. fejezetben található módszerei szintaktikai elméleten alapulnak. A tudás szintaktikai és modális elméleteinek kiváló összehasonlítása található a (Davis, 1990)-ben.

Created by XMLmind XSL-FO Converter.

Arisztotelész *Kategóriák* c. művéhez kommentárt fűzve Porfiriusz görög filozófus (kb. i. e. 234–305) rajzolt fel valamit, amit első szemantikus hálónak lehetne minősíteni. Charles S. Peirce korszerű logikára alapozva egzisztenciális gráfokat fejlesztett ki, mintegy kifejlesztve az első modern logikát használó szemantikus háló formalizmusát (Peirce, 1909). Az MI-n belül a szemantikus hálók kutatását Ross Quillian kezdeményezte (Quillian, 1961), a nyelvfeldolgozás és az emberi memória iránti érdeklődéséből kiindulva. Marvin Minsky nagy hatású cikkében a szemantikus hálók **keret**eknek (**frame**s) nevezett változatát mutatta be (Minsky, 1975). A keret egy objektum vagy egy kategória reprezentációja más objektumokkal vagy kategóriákkal képzett relációival. Bár a cikk érdeklődést keltett *magának* a tudásreprezentációnak a területe iránt, mégis kritizálták, hogy a Minsky cikkét megelőző objektumorientált programozás eszközeit, illetve az öröklődés és az alapértelmezett értékek használatát újból felkínálja (Dahl és társai, 1970; Birtwistle és társai, 1973). Nem világos, hogy az objektumorientált programozásról szóló későbbi cikkeket – viszonzásképpen – mennyire befolyásolta a szemantikus hálókkal kapcsolatos korai MI-munka.

A szemantika kérdése különösen élesen merül fel Quillan szemantikus hálójával kapcsolatban (és mindazokkal kapcsolatban, akik Quillan megközelítését követték), a szemantikus hálóban mindenütt jelen levő és nagyon bizonytalan „ISA-kapcsolatok", valamint egyéb korai tudásreprezentációs formalizmusok – mint amilyenek MERLIN (Moore és Newell, 1973) misztikus „lapos" és „fedi" operációi – miatt. Az MI-kutatók figyelmét Woods híres *What's In a Link?* c. cikke irányította a precíz szemantika kialakításának szükségességére a tudásreprezentációs formalizmusban (Woods, 1975). Brachman dolgozta fel ezt a témát és a problémára megoldásokat is javasolt (Brachman, 1979). Patrick Hayes *The Logic of Frames* c. cikke még mélyebbre vágott azt állítva, hogy „a keret formalizmusának többsége nem más, mint egy új szintaktika az elsőrendű logika egy részére" (Hayes, 1979). Drew McDermott *Tarskian Semantics, or, No Notation Without Denotation!* c. cikkében amellett érvelt, hogy az elsőrendű logika modellalapú szemantikáját kell alkalmazni mindenféle tudásreprezentációs formalizmus esetében (McDermott, 1978b). Ez vitatott kérdés maradt. Megemlítendő, hogy McDermott maga is felülvizsgálta álláspontját *A Critique of Pure Reason* (McDermott, 1987) c. cikkében. A NETL (Fahlman, 1979) egy kifinomult szemantikus háló rendszer volt, melynek ISA-kapcsolatai (amit virtuális másolásnak – virtual copy –, azaz VC-kapcsolatnak hívtak) sokkal inkább a keretrendszerek vagy az objektumorientált programozási nyelvek „öröklődés" jellemvonásának jelölésén alapult, mint a részhalmazkapcsolaton, és sokkal pontosabban volt definiálva, mint Quillian kapcsolatai a Woods előtti időkből. A NETL különösen érdekes, mivel párhuzamos hardveren szándékoztak megvalósítani, hogy így leküzdhessék a nagy szemantikus hálókból történő információ-visszakeresés problémáját. David Touretzky az öröklődést rigorózus matematikai analízisnek vetette alá (Touretzky, 1986). Selman és Levesque a kivételekkel kiegészített öröklődés komplexitását taglalja, megmutatva, hogy ez a legtöbb formalizmusban NP-teljes (Selman és Levesque, 1993).

A leíró logikák fejlődése csupán egy hosszú kutatási folyamat jelenlegi állása, mely az elsőrendű logika olyan használható részhalmazainak megtalálását célozza, melyekre a következtetés számítástechnikai alapon kezelhető. Hector Levesque és Ron Brachman megmutatták, hogy bizonyos logikai konstrukciók, például a diszjunkció és a negálás bizonyos használata felelős elsődlegesen a logikai következtetés kezelhetetlenségéért (Levesque és Brachman, 1987). A KL-ONE rendszerre (Schmolze és Lipkis, 1983) alapozva számos olyan rendszert fejlesztettek ki, melyek tervezése magában foglalta az elméleti komplexitásanalízis eredményeit. Leginkább említésre méltó ilyen rendszerek a KRYPTON (Brachman és társai, 1983) és a CLASSIC (Borgida és társai, 1989). Az eredmény a következtetés sebességének lényeges növekedése lett, és a következtető rendszerek komplexitása és kifejezőkészsége közötti hatások sokkal jobb megismerése. A kutatások állását Calvanese foglalta össze (Calvanese és társai, 1999). Más részről, ahogy Doyle és Patil érvelnek, egy nyelv kifejezőkészségének korlátozása vagy lehetetlenné teszi bizonyos problémák megoldását, vagy a nyelv nem logikai eszközökkel történő kitágítására ösztönöz (Doyle és Patil, 1991).

A nemmonoton következtetés mind a három fő formalizmusát – a körülírást (McCarthy, 1980), az alapértelmezett logikát (Reiter, 1980) és a modális nemmonoton logikát (McDermott és Doyle, 1980) – az *AI Journal* egyetlen különszámában publikálták. A válaszhalmaz-programozást a negálás mint kudarc kiterjesztésének vagy a körülírás finomításának lehet nézni. A mögötte lévő stabil modell szemantikát Gelfond és Lifschitz vezették be (Gelfond és Lifschitz, 1988). A vezető válaszhalmaz-programozási rendszerek a DVL (Eiter és társai, 1998) és SMODELS (Niemelä és társai, 2000). A diszkmeghajtó példa a SMODELS felhasználói kézikönyvéből származik (Syrjänen, 2000). A válaszhalmazprogramozás használatát a tervkészítés számára Lifschitz tárgyalja (Lifschitz, 2001). A nemmonoton logika különböző megközelítéseiről jó áttekintést ad Brewka (Brewka és társai, 1997). A logikai programozás negálás mint kudarc megközelítésével és a Clark-lezárással Clark foglalkozik (Clark, 1978). Van Emden és Kowalski azt mutatták ki, hogy minden negálás nélküli Prolog programnak létezik egyértelmű minimálmodellje. Az utóbbi években tanúi lehetünk, ahogy növekszik az érdeklődés a nemmonoton logika nagyméretű tudásreprezentációs rendszerekben való alkalmazása

Created by XMLmind XSL-FO Converter.

iránt. Egy nemmonoton öröklődési rendszer első sikeres kereskedelmi alkalmazása talán a BENINQ rendszereké volt a biztosítási feltételek megtudakolásához (Morgenstern, 1998). A *Logic Programming and Nonmonoton Reasoning* (LPNMR) konferenciák kiadványai sok különböző, logikai programozáson alapuló nemmonoton következtető rendszerről számolnak be.

Az igazság-karbantartó rendszerek tanulmányozása TMS (Doyle, 1979) és RUP rendszerekkel (McAllester, 1980) kezdődött, mindkettő alapvetően JTMS-rendszer volt. Az ATMS megközelítést Johan de Kleer cikkeinek sorozata írta le (De Kleer, 1986c; 1986a; 1986b). A *Building Problem Solvers* (Forbus és De Kleer, 1993) részletesen elmagyarázza, hogyan használhatók a TMS-ek MI-alkalmazásokban. Nayak és Williams megmutatják, hogy egy hatékony TMS hogyan teszi lehetővé, hogy a NASA-űrhajók műveleteit valós időben megtervezhessük (Nayak és Williams, 1997).

Nyilvánvaló okoknál fogva a jelen fejezet a tudásreprezentáció minden területével elmélyülten nem foglalkozik. A három fő kihagyott témakör az alábbi:

- **Kvalitatív fizika (qualitative physics)**: A tudásreprezentáció egy olyan részterülete, amely speciálisan a fizikai objektumok és folyamatok logikai nemnumerikus elméletének megkonstruálásával foglalkozik. A kvalitatív fizika elnevezést Johan de Kleer alkotta meg (de Kleer, 1975), bár állítható, hogy a kutatás Fahlman BUILD rendszerével indult meg (Fahlman, 1974). A BUILD egy komplex tervkészítő rendszer volt, bonyolult kockatornyok építéséhez. Fejlesztése során Fahlman észrevette, hogy az erőfeszítések zöme (becslése szerint 80%-a) nem magára a tervkészítésre fordítódott, hanem a különböző kocka részstruktúrák stabilitásának eldöntésére. Fahlman egy hipotetikus naiv fizikai jellegű folyamatot vázol, hogy megmagyarázza, miért képesek a kisgyerekek BUILD-szerű problémákat megoldani a BUILD-del végzett fizikai modellezésnél használt, nagy sebességű lebegőpontos aritmetika nélkül. Hayes a „történetek” – a tér-idő 4-dimenziójú, Davidson eseményeire hasonlító szeletei – felhasználásával megalkotja a folyadékok igen bonyolult naiv fizikáját (Hayes, 1985a). Hayes volt az első, aki bebizonyította, hogy ha egy csapból folyamatosan folyik a víz, miközben a lefolyót bedugaszoljuk, a kád végül túlcsordul, és azt is, hogy aki beleesik a tóba, az teljesen elázik. De Kleer és Brown, valamint Ken Forbus megkísérelték a világ egyfajta, a fizikai egyenletek kvalitatív absztrakcióján alapuló általános rendeltetésű elméletének a felállítását (De Kleer és Brown, 1985; Forbus, 1985). Az utóbbi időben a kvalitatív fizika eljutott addig a pontig, ahol már lehetséges a komplex fizikai rendszerek széles választékának az elemzése (Sacks es Joskowicz, 1993; Yip, 1991). Kvalitatív technikákat használtak teljesen új tervezésű órák, szélvédőtörlők és hatlábú sétálórobotok megkonstruálásához is (Subramanian, 1993; Subramanian és Wang, 1994). A területhez jó bevezető olvasmány a *Readings in Qualitative Reasoning about Physical Systems* c. gyűjtemény (Weld és De Kleer, 1990).

- **Térbeli következtetés (spatial reasoning)**: A wumpus és a bevásárló világban a navigáláshoz szükséges következtetés triviális a valódi világ gazdag térbeli struktúrájához képest. A térről történő józan ész következtetés legteljesebb kísérlete Ernest Davis (Davis, 1986; 1990) munkájában található. Cohn régiókapcsolódási kalkulusa (Cohn és társai, 1997) az alapja az egyfajta kvalitatív térbeli következtetésnek, ami elvezetett a földrajzi információs rendszerek új fajtáihoz. A kvalitatív fizikához hasonlóan, úgy tűnik, hogy egy ágens sokáig elboldogul egy teljes metrikus reprezentáció használata nélkül. Ha egy ilyen reprezentációra van szükség, használhatók a robotikában kifejlesztett megoldások (lásd 25. fejezet).

- **Pszichológiai következtetés (psychological reasoning)**: A pszichológiai következtetés magában foglalja egy működőképes *pszichológia* kifejlesztését a mesterséges ágensek számára, hogy az ágensek képesek legyenek saját magukra és más ágensekre vonatkozó következtetéseket meghozni. Ennek gyakran az ún. „népi pszichológia” („folk psychology”) az alapja, amely egy olyan elmélet, amiről azt gondoljuk, hogy az emberek saját maguk és mások megítélésében használják. Amikor az MI-kutatók mesterséges ágenseiket más ágensekre vonatkozó következtetést támogató pszichológiai elméletekkel látják el, ezen elméletek alapja gyakran a logikai ágens rendszertervének a kutató által megfogalmazott leírása. A pszichológiai következtetés pillanatnyilag a természetes nyelv megértésének kontextusán belül a leghasznosabb, ahol elsődleges fontosságú a beszélő intencióját megjósolni.

Az ezen a területen folyó munkák legfrissebb forrását a *Principles of Knowledge Representation and Reasoning* nemzetközi konferenciák kiadványai jelentik. A *Readings in Knowledge Representation* (Brachman és Levesque, 1985) és a *Formal Theories of the Commonsense World* (Hobbs és Moore, 1985) a tudásreprezentáció kiváló antológiái. Az előbbi a reprezentációs nyelvek és formalizmusok történelmileg fontos publikációira összpontosít, az utóbbi pedig magának a tudásnak az akkumulálására helyezi a hangsúlyt. Davis, Stefik és Sowa könyvei jó bevezető munkák a tudásreprezentációkhoz (Davis, 1990; Stefik, 1995; Sowa, 1999).

Created by XMLmind XSL-FO Converter.

9.2. Feladatok

10.1.

Írja fel a wumpusvilág-beli *Lőj* cselekvés hatását leíró állításokat. Írja le a wumpusra kifejtett hatását, és ne felejtse, hogy a lövéshez az ágensnek nyílra van szüksége.

10.2.

A szituációkalkuluson belül fogalmazzon meg egy olyan axiómát, amelyben a 0. időpillanatot az S_0 szituációval kapcsolja össze, valamint egy olyan másik axiómát, ahol a t időpillanatot az S_0-ból a t számú cselekvés szekvenciájával előállított szituációval kapcsolja össze.

10.3.

Ebben a feladatban egy út megtervezésével foglalkozunk egy robot számára, két város között. A robot alapcselekvése a *Megy*(*x*, *y*), mely révén az *x* városból az *y* városba kerül át, ha a két város között létezik közvetlen út. A *KözvetlenÚt*(*x*, *y*) akkor és csak akkor igaz, ha az *x* és az *y* között létezik egy közvetlen útszakasz. Feltételezhetjük, hogy az ilyen tények a rendszer tudásbázisába már bekerültek (1.2. szakasz - A problémák megfogalmazása részben lévő térképet). A robot Aradnál kezd, és Bukarestet kell elérnie.

a. Adja meg robot kezdő állapotának egy alkalmas logikai leírását.

b. Írjon fel egy alkalmas logikai kérdést, amelynek megoldásai megadják a célhoz vezető lehetséges utakat.

c. Írja fel a *Megy* cselekvést leíró logikai állítást.

d. Tegyük most fel, hogy két város között a közvetlen út követése a közvetlen út hosszával arányos üzemanyag-fogyasztást jelent. A robot teli tankkal indul. Bővítse reprezentációját, hogy ezek a szempontok helyet kapjanak benne. A cselekvés leírásának olyannak kell lennie, hogy az előbb megfogalmazott kérdés megválaszolásának továbbra is a lehetséges terveket kell megadnia.

e. Írja le a kezdeti szituációt, és adja meg a *Megy* cselekvést leíró új szabályt, illetve szabályokat.

f. Tegyük most fel, hogy a csomópontok némelyike benzinkút is egyben, ahol a robot az üzemanyagtankját teletöltheti. Bővítse ennek megfelelően a reprezentációját, és írja fel a benzinkutak leírásához szükséges új szabályokat, a *TeleTölt* cselekvést is beleértve.

10.4.

Olvasnivaló. Vizsgálja meg annak a lehetőségét, hogy az eseménykalkulust az *egyidejű* események kezelésével bővíthesse. Elkerülhető-e az axiómák kombinatorikus robbanása?

10.5.

A fejezetben kifejlesztett reprezentációt felhasználva és kiterjesztve adja meg az alábbi hat állítás reprezentációját.

a. 0 °C és 100 °C között cseppfolyós a víz.

b. A víz 100 °C-nál forr fel.

c. János palackjában megfagyott a víz.

d. Az ásványvíz egyfajta víz.

e. János palackjában ásványvíz van.

f. Minden folyadéknak van fagyási pontja.

g. Egy liter víz súlya több, mint egy liter alkohol súlya.

Created by XMLmind XSL-FO Converter.

Ismételje meg a feladatot mereológiai megközelítésen alapuló reprezentációt felhasználva, amelyben például a *Víz* egy objektum, amely a világ minden vizét, mint az objektum részét tartalmazza.

10.6.

Írja fel a:

a. *KimerítőRészFelosztás*

b. *RészPartíció*

c. *PáronkéntKölcsönösenKizáró*

definícióját a *KimerítőFelosztás*, a *Partíció* és a *KölcsönösenKizáró* definíciójával analóg módon. Igaz-e, hogy *RészPartíció*(s, *Kötege*(s))? Ha igen, bizonyítsa be, ha nem, adjon meg egy ellenpéldát, és definiáljon elégséges feltételeket, amelyek mellett igaz lesz.

10.7.

Írjon fel egy olyan állításhalmazt, amely lehetővé teszi a különálló paradicsomok (vagy más objektumok) ármegállapítását, ha adott a kilónkénti ár. Az elmélet kiterjesztésével tegye lehetővé egy zacskó paradicsom árának a kiszámítását.

10.8.

Mértékeket reprezentáló alternatív sémában az egységfüggvényeket egy absztrakt hosszúság objektumra alkalmazzuk. Az ilyen sémában azt lehetne írni, hogy: *Centiméterek*(*Hossz*(L_1)) = 15. Hogyan viszonyul ez a séma a fejezetbelihez? A szempontok többek között az átváltási axiómák, az absztrakt mennyiségek elnevezései (mint például „ezer forint”) és az eltérő egységekben kifejezett absztrakt mérések összehasonlítása (50 hüvelyk több, mint 50 cm).

10.9.

Konstruáljon a valuták átváltási árfolyamára egy olyan reprezentációt, amely lehetővé teszi a napi fluktuációk figyelembevételét.

10.10.

Ebben a feladatban az eseménykategóriák és azon időintervallumok közötti relációkkal foglalkozunk, amelyekben az események megtörténnek.

a. Definiálja a *T*(*c*, *i*) predikátumot a *Közben* és a $\in$ segítségével.

b. Adja meg a precíz magyarázatát annak, hogy a konjunktív eseménykategória leírásánál miért nincs szükség a kétféle jelölésmódra.

c. Adja meg a *T*(*Egyik*(*p*, *q*), *i*) és a *T*(*VagyVagy*(*p*, *q*), *i*) formális definícióját.

d. Magyarázza meg, miért értelmes a kétfajta diszjunkcióval analóg módon az események kétfajta negálásával rendelkezni. *Nem*-nek és *Sohasem*-nek nevezze el őket, és adja meg a formális definíciójukat.

10.11.

Definiálja a *Rögzített* predikátumot, ahol a *Rögzített*(*Helye*(*x*)) azt jelenti, hogy az *x* objektum helye az időben rögzített.

10.12.

Definiálja az *Előtte*, az *Utána*, a *Közben* és az *Átlapolódik* predikátumokat a *Találkozik* predikátum és a *Kezdet* és a *Vég* függvények segítségével, de az *Idő* függvény és a < predikátum nélkül.

10.13.

Created by XMLmind XSL-FO Converter.

A 10.5. alfejezetben a *Link* és a *LinkSzöveg* predikátumokat használjuk weblapok közötti kapcsolat leírására. Többek közt a *Címkében* és a *VeddElőLapot* predikátumok felhasználásával írja fel a *Link* és a *LinkSzöveg* definícióit.

10.14.

A vásárlási folyamat egyik része, amivel e fejezetben nem foglalkoztunk, az egyes tételek közötti kompatibilitás ellenőrzése. Ha a kliens például egy számítógépet rendel, illeszkedik-e az a megfelelő perifériákhoz? Ha digitális kamerát rendel, megvan-e a hozzá való memóriakártya és szárazelem? Írjon fel egy olyan tudásbázist, amely eldönti, hogy a tételek adott halmaza kompatibilis-e. És ezt a tudásbázist fel lehessen használni cserékre, illetve további tételek vásárlására vonatkozó javaslatok generálására, ha mégsem lennének kompatibilisek. Bizonyosodjon meg arról, hogy a tudásbázis legalább a termékek egy körével jól működik, és könnyen terjeszthető ki másokra is.

10.15.

Adjon szabályokat a *Név*(*s, c*) predikátum definíciójának olyan esetekre történő kiterjesztéséhez, hogy az olyan füzérek, mint például a „laptop számítógép” több bolthoz tartozó idevágó kategórianévre illeszkedjen. Kíséreljen meg általános érvényű definíciót alkotni. A definíciót tesztelje tíz online boltra és az általuk három kategóriára használt elnevezésekre. A laptop kategória esetén például a „Notebooks”, „Laptops”, Notebook számítógépek”, „Notebook”, „Notebooks és Laptops” és „Notebook PC” neveket találtuk. Az egyes elnevezéseket explicit *Név* tényekkel, mások a többes számot, a konjunkciót stb. kezelő szabályokkal lefedhetők.

10.16.

A vásárló vásárlásleírására vonatkozó pontatlan illeszkedés teljes megoldása nagyon nehéz, és szükségessé teszi a természetes nyelvfeldolgozási és információ-kinyerési technikák egész sorát (lásd 22. és 23. fejezet). Egy kis lépés megengedni a felhasználónak, hogy a különböző attribútumok minimális és maximális értékeit határozhassa meg. Megköveteljük, hogy a vásárló a termékek leírásához az alábbi nyelvtant vegye igénybe:

```
Leírás → Kategória [Kapcsolat Módosító]*

Kapcsolat → „val”|„és”|„,”

Módosító → Attribútum|Attrribútum Op Érték

Op → „=”|>”|„<”
```

Itt a *Kategória* a termék kategóriája, az *Attribútum* a termék valamilyen tulajdonsága, mint például „CPU” vagy „ár”, és az *Érték* egy konkrét termék attribútumának az értéke. A „számítógép, min 2,5 GHz CPU, 250 000 Ft alatt” lekérdezést tehát úgy kell kifejezni, hogy „számítógép, CPU > 2,5 GHz és ár < 250 000 Ft”. Implementálja az ilyen nyelven megfogalmazott termékleírásokat elfogadó bevásárló ágenst.

10.17.

Az internetes bevásárlásról szóló leírásunk nélkülözte az alapvető fontosságú lépést – a termék tényleges *megvásárlását*. Adja meg a vásárlás formális logikai leírását eseménykalkulusra támaszkodva. Azaz definiálja az események olyan sorozatát, amelyek megtörténnek, ha a vásárló hitelkártyával fizet, és végül sor kerül a számlázásra és az áru leszállítására.

10.18.

Írja le a valamit valamiért kereskedés eseményét. A vásárlást egyfajta kereskedésként írja le, ahol a kereskedésben az egyik részt vevő termék a pénz.

10.19.

Olvasnivaló. A két megelőző feladatban a tulajdonviszony igen egyszerű fogalmát használtuk. A vásárló például úgy indul, hogy a forint bankjegyek már a *tulajdonában vannak*. A modell kezd szétesni, ha például a pénze bankban van, hiszen ilyenkor semmilyen konkrét bankjegymennyiség nincs, amiről ki lehetne jelenteni, hogy a tulajdonában van. A helyzetet a kölcsön, a bérlés, a haszonbérlés és a letétbe helyezés még tovább

Created by XMLmind XSL-FO Converter.

bonyolítja. Vizsgálja meg a tulajdonviszony józan ész és jogi koncepcióit, és javasoljon egy olyan sémát, amelyben ezek formálisan kifejezhetők.

10.20.

A feladata egy tanácsadó rendszer kifejlesztése, amely az informatika szak hallgatóinak ajánlaná, hogy milyen tárgyakat válasszanak a képzés feltételeinek teljesítéséhez (az intézményére jellemző feltételekkel dolgozzon). Először döntse el, hogy az összes információ reprezentálásához milyen szókészletet fog használni, majd végezze el a reprezentáció építését. Megfelelő kérdésekkel érje el, hogy a rendszer egy legális képzési programot adjon vissza. A rendszernek az egyes hallgatók igényeihez kell illeszkednie, például a fakultatív tárgyak javaslatánál.

Javasoljon módszereket a rendszer tökéletesítésére, felhasználva például a hallgatói preferenciákra, a terhelésre, a jó/rossz előadókra vonatkozó információkat. Minden egyes tudásfajta esetén magyarázza meg, hogy hogyan lehetne azt logikailag kifejezni. Könnyen be tudná-e a rendszere fogadni ezt az információt úgy, hogy egy hallgató részére *a legjobb* képzési programot ajánlja fel?

10.21.

A 10.1. ábra a dolgok hierarchiájának felső szintjeit mutatja. Terjessze ki az ábrát úgy, hogy a lehető legtöbb valódi kategóriát tartalmazzon. Erre jó módszer, ha a mindennapi életével kapcsolatos minden dolgot leír. Ez objektumokat és eseményeket fog magában foglalni. Kezdje a reggeli felkeléssel, rendszeresen haladjon tovább, megfigyelve, hogy mit érint, mit csinál, mire gondol. Egy véletlen mintavétel eredménye lehetne például a következő: zene, hírek, tej, séta, kocsivezetés, benzin, Skála, szőnyeg, beszélgetés, Kovács tanár úr, csirkepörkölt, nyelv, 150 Ft, napi újságok stb.

El kellene készíteni egy egységes hierarchiafát (egy nagy papírlapon) és egy listát az objektumokról és a kategóriákról, az egyes kategóriaegyedek által kielégített egy vagy több relációval együtt. Mindegyik objektumnak egy kategóriához kell tartoznia, és minden kategóriának a hierarchia részének kell lennie.

10.22.

(Doug Lenat egy példájának adaptációja.) A feladat az, hogy elegendő, logikai formában kifejezett kapcsolódó tudást gyűjtsön össze, hogy az alábbi egyszerű mondatra vonatkozó kérdéssort meg tudja válaszolni:

Tegnap János a Skálába ment, és vett két kiló paradicsomot és fél kiló darált húst.

Kezdje azzal, hogy a mondat tartalmát egy sor kijelentéssel reprezentálja. Áttekinthető logikai struktúrájú állításokat kell írnia (például állítások arról, hogy az objektumoknak bizonyos tulajdonságai vannak, hogy az objektumok bizonyos relációban vannak egymással, és hogy egy adott tulajdonságot teljesítő minden objektum egy másik tulajdonsággal is rendelkezik). Az alábbiak feltehetően segítségére lesznek:

- Milyen osztályokra, egyedekre, relációkra stb. lesz szüksége? Mik ezeknek a szuper- és alosztályai? (Többek közt eseményekre és időrendi rendezésre is szüksége lesz.)
- Milyen módon lehetnének ezek a részei egy általánosabb hierarchiának?
- Milyenek a közöttük lévő kényszerek és relációk?
- Mennyire részletesen kell leírni az egyes fogalmakat?

A megkonstruálandó tudásbázisnak képesnek kell lennie a továbbiakban közölt kérdéslistát megválaszolni. A kérdések közül néhány közvetlenül magával a történettel foglalkozik, de a többségük a háttértudást – a sorok közötti olvasást – igényli. Foglalkozni kell azzal, hogy milyen dolgokat lehet egy bevásárlóközpontban vásárolni, hogy a kiválasztott tételek vásárlásánál mi történik, hogy mi a vásárolt tételek rendeltetése stb. Kísérelje meg a reprezentációját a lehető a legáltalánosabbra kialakítani. Egy triviális példával szemlélve, ne azt állítsa, hogy „Az emberek a Skálában ennivalót vásárolnak”, mert ez nem segít azok kezelésében, akik máshol szoktak vásárolni. Ne azt állítsa, hogy „János darált hússal és paradicsommal spagettit készített”, mert ez semmilyen más dolog kezelésében nem segít. Ne rejtsen válaszokat a kérdésekbe, a (c) kérdés például azt kérdezi, hogy „Vásárolt János húst?”, és nem azt, hogy „Vásárolt-e János fél kiló darált húst?”.

Adja meg a kérdéseket megválaszoló következtetés sémáját. Ennek során minden bizonnyal új fogalmakat is kell kreálnia és új állításokat kell megfogalmaznia stb. Ha lehetséges, használjon egy logikai következtető

Created by XMLmind XSL-FO Converter.

rendszert, hogy kimutassa a tudásbázis elégségességét. Sok dolog, amit leír, a valóságban csak közelítőleg lesz igaz. Ne törődjön ezzel túl sokat, a cél azt a józan ész tudást kinyerni, ami a kérdések megválaszolását egyáltalán lehetségessé teszi. E kérdéskör igazán teljes megválaszolása *rendkívül* nehéz, valószínűleg meg is haladja a mai tudásreprezentációk lehetőségeit.

a. János gyerek vagy felnőtt? [Felnőtt]

b. Van Jánosnak most legalább 2 db paradicsoma? [Igen]

c. Vásárolt János húst? [Igen]

d. Látta-e János Marit, ha Mari ugyanabban az időben vásárolt paradicsomot? [Igen]

e. A paradicsomot a bevásárlóközpontban készítik-e? [Nem]

f. Mit fog csinálni János a paradicsommal? [Megeszi]

g. Kapható-e a Skálában dezodor? [Igen]

h. Hozott-e János magával elég pénzt a bevásárlóközpontba? [Igen]

i. A vásárlást követően kevesebb pénze van-e Jánosnak? [Igen]

10.23.

Az előbbi példa tudásbázisához készítsen szükséges bővítéseket/módosításokat úgy, hogy megválaszolhatók legyenek az alábbi kérdések. Mutassa ki, hogy a tudásbázis tényleg képes ezeket megválaszolni, és a jelentésében számoljon be a javításokról, miért volt a javításokra szükség, és arról is, hogy a javítások lényegtelenek, illetve lényegesek voltak.

a. Vannak-e a Skálában más emberek, amikor János ott vásárol? [Igen – a személyzet!]

b. Vegetáriánus-e János? [Nem]

c. A Skálában ki a dezodor tulajdonosa? [Skála Rt.]

d. Kapott-e János 1 kg darált húst? [Igen]

e. A szomszédos MOL benzinkútnál van-e benzin? [Igen]

f. Belefér-e a paradicsom János kocsijába? [Igen]

10.24.

Emlékezzen arra, hogy a szemantikus háló öröklődési információját logikailag alkalmas implikációs állításokkal ki lehet fejezni. Ebben a feladatban megfontoljuk az ilyen állításokra épülő öröklődési következtetés hatékonyságát.

a. Vegyük egy használtautó-katalógus információtartalmát – hogy az 1999-es Mitsubishi Carisma értéke 3 millió Ft. Tegyük fel, hogy az összes információ (a több tízezer modell részére) a fejezetben ajánlott logikai szabályok formájában van megfogalmazva. Írjon fel három ilyen szabályt, az egyik az 1999-es Carismáról szóljon. Hogyan használná a szabályokat, hogy egy *konkrét* gépkocsi értékét megtalálhassa (például XY, ami egy 1999-es Carisma), ha adott egy hátrafelé láncolt tételbizonyító, mint amilyen például a Prolog?

b. Hasonlítsa össze az ennek a problémának a megoldására alkalmazott hátrafelé láncolt következtetési módszer és a szemantikus hálóban használt öröklődési séma hatékonyságát.

c. Magyarázza meg, hogy az előrefelé láncolt következtetés hogyan teszi lehetővé egy logikai rendszer számára ugyanannak a problémának egy hatékony megoldását, feltéve, hogy a tudásbázis csak 10 000 szabályt tartalmaz az árakról.

d. Írjon le egy olyan helyzetet, ahol sem az előrefelé, sem a hátrafelé láncolt következtetés nem teszi lehetővé a konkrét gépkocsira vonatkozó árlekérdezés hatékony megvalósítását.

Created by XMLmind XSL-FO Converter.

e. Tud olyan megoldást ajánlani, ami lehetővé teszi, hogy ezt a fajta lekérdezést a logikai rendszerek mindegyikében hatékonyan tudjuk kezelni? (*Segítség:* emlékezzen arra, hogy az ugyanahhoz a kategóriához tartozó két gépkocsinak ugyanaz az ára.)

10.25.

Feltételezhető, hogy egy szemantikus hálóban a sima és az egy vonallal bekeretezett kapcsolat közötti szintaktikai különbségtétel felesleges, mert az utóbbi kapcsolatok csak a kategóriákhoz tartoznak. Az öröklődési algoritmusnak elég lenne feltételeznie, hogy a kategóriához csatolt sima kapcsolat szándékoltan a kategória minden tagjára vonatkozik. Mutassa meg a lehetséges hibák megadásával, hogy az érv hamis.

Created by XMLmind XSL-FO Converter.

11. fejezet - Tervkészítés

Ebben a fejezetben láthatjuk, hogy hogyan használhatja ki egy ágens a probléma szerkezetét egy összetett cselekvési terv kialakításához.

Egy cél elérésére irányuló cselekvéssorozat kialakítását **tervkészítés**nek (**planning**) nevezzük. Tervkészítő ágensekre eddig két példát láttunk: a 3. fejezet keresésalapú problémamegoldó ágensét és a 10. fejezetben bemutatott logikai tervkészítő ágenst. Ez a fejezet elsősorban az olyan nagyobb bonyolultságú tervezési feladatokra való felskálázással foglalkozik, melyek meghaladják az eddig ismertetett megközelítések képességeit.

A 11.1. fejezet egy kifejező, de mégis megfelelően korlátozott nyelvet mutat be az állapotokat és cselekvéseket tartalmazó tervkészítési feladatok leírására. Ez a nyelv szoros kapcsolatban áll a 7. és 10. fejezetben szereplő ítéletlogikai, illetve elsőrendű leírásokkal. A 11.2. fejezet megmutatja, hogy az előre- és a hátrafelé kereső algoritmusok hogyan tudják előnyösen kihasználni ezt a reprezentációt, elsősorban a leírás szerkezetéből automatikusan levezethető heurisztikák segítségével. (Ez analóg az 5. fejezetben bemutatott, a kényszerkielégítési problémához kialakított heurisztika elkészítésével.) A 11.3–11.5. alfejezetben olyan tervkészítő algoritmusok kerülnek bemutatásra, amelyek felhasználva a probléma reprezentációját, képességeikben túlmutatnak az előre-, illetve hátrafelé keresésen. Nevezetesen olyan eljárásokat vizsgálunk meg, amelyek nem csak teljesen rendezett cselekvések sorozatát képesek figyelembe venni.

Ebben a fejezetben csak olyan környezetekre szorítkozunk, melyek teljesen megfigyelhetők, determinisztikusak, végesek, statikusak (azaz ahol változások csak akkor történnek, amikor az ágens cselekszik), diszkrétek (időben, cselekvésekben, objektumokban és hatásokban). Ezeket **klasszikus tervkészítés**i (**classical planning**) környezeteknek nevezzük. Ezzel ellentétben a nem klasszikus tervkészítés a részben megfigyelhető vagy a sztochasztikus környezetek kezelésére szolgál, és itt megjelennek eltérő, a 12. és 17. fejezetben bemutatott, algoritmusok és ágenstervek.

1. A tervkészítési probléma

Fontoljuk meg, hogy mi történne, ha egy közönséges, a standard keresési algoritmusokat – mélységi keresés, A* stb.– használó problémamegoldó ágens valós nagyméretű problémákkal kerülne szembe. Ez segíthet abban, hogy jobb tervkészítő ágenseket tervezzünk.

A legkézenfekvőbb nehézség, hogy a problémamegoldó ágenst lebéníthatnák a szükségtelen cselekvések. Vegyük például a *Mesterséges intelligencia modern megközelítésben* c. könyv internetes megvásárlásának a feladatát. Tegyük fel, hogy minden egyes 10 jegyű ISBN szám megvásárlása egy-egy cselekvés, ez összesen 10 milliárd cselekvést jelent. A keresési algoritmusnak mind a 10 milliárd cselekvés végállapotát meg kellene vizsgálnia, hogy megtalálja a célnak megfelelőt, nevezetesen, hogy birtokoljuk a 9635454112 ISBN számú könyvet. Másrészről egy értelmes tervkészítő ágensnek képesnek kell lenni arra, hogy a pontos cél *Birtokol*(*ISBN9635454112*) leírásából visszafelé dolgozva közvetlenül eljusson a *Vásárol*(*ISBN9635454112*) cselekvéshez. Hogy ezt megtehesse, az ágensnek arra az általános tudásra van szüksége, hogy a *Vásárol*(*x*) következménye a *Birtokol*(*x*). Ha adott ez a tudás, a tervező egy egyszerűsítési lépésben el tudja dönteni, hogy a *Vásárol*(*ISBN9635454112*) a helyes cselekvés.

A következő nehézség, egy jó **heurisztika** (**heuristic function**) meghatározása. Tegyük fel, hogy az ágens feladata négy különböző könyv megvásárlása. Ebből 10^{40} négylépéses terv adódik, azaz nem kérdéses, hogy egy megfelelő heurisztika nélküli keresés értelmetlen. Az ember számára egy állapot költségének becslésére kézenfekvő heurisztika a továbbiakban még megvásárolandó könyvek száma. Sajnos ez nem nyilvánvaló egy problémamegoldó ágens számára, hisz az a céltesztet egy fekete dobozként látja, mely minden állapotra mindössze egy igaz-hamis értéket ad vissza. Ennek következményeképp a problémamegoldó ágens nem autonóm; azaz emberi beavatkozást igényel minden új problémánál az alkalmazható heurisztika megadására. Másrészről, ha az ágensnek rendelkezésére áll a cél leírása, mint részcélok konjunkciója, akkor használhat egy egyszerű feladatfüggetlen heurisztikát: a még nem teljesített részcélok számát. A könyvvásárlási feladat esetén a cél $Birtokol(A) \wedge Birtokol(B) \wedge Birtokol(C) \wedge Birtokol(D)$ lenne, és a $Birtokol(A) \wedge Birtokol(C)$ állapot költsége 2. Így az ágens számára automatikusan elérhető a helyes heurisztika erre és más problémákra is. A fejezet későbbi részében látni fogjuk, hogy hogyan lehet létrehozni olyan kifinomultabb heurisztikákat, ami a célstruktúráján túl számba veszi a végrehajtható cselekvéseket is.

Created by XMLmind XSL-FO Converter.

Végül a problémamegoldó ágens nem hatékony, mert nem tudja kihasználni a **problémadekompozíció** (**problem decomposition**) lehetőségét. Vegyük például a következő feladatot: több csomagot kell kiszállítanunk a megfelelő címekre, melyek Ausztrália különböző pontjain találhatók. Jó megközelítés, ha megkeressük a célpontokhoz legközelebb eső reptereket, és felosztjuk a teljes problémát több részfeladatra; repterenként egyre. Az egy reptéren keresztül irányított csomagok esetén a további dekompozíció lehetősége a célvárostól függ. Az 5. fejezetben láttuk, hogy egy ilyen felbontás képessége hozzájárul a kényszerkielégítési feladatmegoldók hatékonyságához. A tervkészítőkre ugyanez igaz: a legrosszabb esetben n csomag legjobb kiszállítási tervének elkészítése $O(n!)$, míg ha a feladat k egyenlő részre bontható ez mindössze $O((n/k)! \times k)$ komplexitású feladat.

Ahogy az 5. fejezetben megjegyeztük, a teljesen dekomponálható problémák jók, de ritkák.[1] A legtöbb tervkészítő rendszer felépítése – különösen a 11.3. fejezetben bemutatásra kerülő részben rendezett tervkészítő felépítése – azon a feltételezésen alapul, mely szerint a legtöbb valós, életszerű probléma **majdnem dekomponálható** (**nearly decomposable**). Ez annyit jelent, hogy a tervkészítő dolgozhat független részcélokon, de a résztervek összekombinálása további feladatokat eredményezhet. Néhány feladat esetén ez a feltételezés nem helytálló, mert az egyik részcél kidolgozása gyakran megbontja a másik részcélt. Ezek a részcélok közötti kölcsönhatások azok, amelyek a fejtörőket (mint a nyolcas kirakó) valójában fejtörővé teszik.

1.1. A tervkészítési problémák nyelve

A fent leírtak alapján a tervkészítési problémák reprezentációjának – ami az állapotokat, cselekvéseket és célokat jelenti – lehetőséget kellene biztosítania a tervkészítő algoritmus számára, hogy a feladat logikai struktúráját kihasználhassa. Ennek kulcsa, hogy olyan nyelvet találjunk, ami kellően kifejező ahhoz, hogy a problémák egy széles körét leírja, ugyanakkor kellően szigorú ahhoz, hogy a leírásokon hatékony algoritmusok működhessenek. Ebben a fejezetben először körvonalazzuk a klasszikus tervkészítők által használt alapnyelvet, ami `STRIPS` néven ismert.[2] Később rámutatunk a számos módosítási lehetőségből néhányra a `STRIPS`-szerű nyelvekben.

Az állapotok leírása. A tervkészítők a világot logikai feltételekre dekomponálják, és az állapotokat a pozitív literálok konjunkciójaként írják le. Tekintsük az ítéletlogikai literálokat; például a *Szegény* ∧ *Ismeretlen* reprezentálhatja egy szerencsétlen ágens állapotát. Elsőrendű literálokat is felhasználunk; például $Ott(Repülő_1, Melbourne) \wedge Ott(Repülő_2, Sydney)$ a csomagszállítási feladat egy állapotát írhatja le. Az elsőrendű logikai állapotleírások literáljainak **alap**- és **függvénymentes literál**nak (**ground and function-free**) kell lenniük. Az olyan literálok, mint az $Ott(x, y)$ vagy az $Ott(Apja(Ferenc), Sydney)$ nem megengedettek. A **zárt világ feltételezés**t (**closed-world assumption**) használjuk, ami annyit tesz, hogy a nem felsorolt állításokat hamisnak vesszük.

A célok leírása. A cél egy részlegesen definiált állapot, melyet pozitív alapliterálok konjunkciója reprezentál, mint *Gazdag* ∧ *Híres* vagy $Ott(P_2, Tahiti)$. Az s ítéletlogikai állapot **kielégíti a** c **célt** (**goal satisfiction**), ha s tartalmazza a c-ben szereplő összes atomot (és esetleg még továbbiakat). Például a *Gazdag* ∧ *Híres* ∧ *Szomorú* állapot kielégíti a *Gazdag* ∧ *Híres* célt.

A cselekvések leírása. A cselekvést a következő két állapot határozza meg: az előfeltétel, aminek teljesülni kell az akció végrehajtásához, és a következmény, ami a végrehajtás eredményeként lép fel. Például két állomás közötti repülés leírása az alábbi:

```
Cselekvés(Repül(p, honnan, hova),

Előfeltétel: Ott(p, honnan) ∧ Repülő(p) ∧ Repülőtér(honnan) ∧ Repülőtér(hova)

Következmény: ¬Ott(p, honnan) ∧Ott(p, hova))
```

Ezt pontosabban **cselekvési séma**nak (**action schema**) nevezzük, ami azt takarja, hogy ez számos különböző cselekvést reprezentál, ami a *p*, *honnan* és a *hova* változók különböző behelyettesítéseivel származtatható. Általánosságban a cselekvési séma három fő részből áll:

- A cselekvés megnevezése és paraméterlistája – például a $Repül(p, honnan, hova)$ – a cselekvés azonosítására szolgál.

[1] Vegyük észre, hogy még a csomagkiszállítási feladat sem teljesen dekomponálható. Vannak esetek, amikor jobb a csomagokat mégis egy távolabbi reptérre irányítani, ha ezzel megspórolhatunk egy külön repülőjáratot a közelire. Mindemellett a legtöbb szállítócég inkább a már bejáratott dekomponált megoldásokhoz ragaszkodik, hogy csökkentse a számítási és szervezési nehézségeket.

[2] A `STRIPS` a STandford Research Institute Problem Solver rövidítése.

Created by XMLmind XSL-FO Converter.

- Az **előfeltétel** (**precondition**), függvényektől mentes pozitív literálok konjunkciója, azt mutatva, hogy milyen feltételeknek kell előzetesen teljesülni a cselekvés végrehajtásához.

- A **következmény** (**effect**), függvényektől mentes literálok konjunkciója, ami leírja, hogy az állapot hogyan változik, amikor a cselekvés végrehajtásra kerül. A cselekmény eredményeképp adódó következményrészben szereplő *P* pozitív literál igaz értéket kap, míg a ¬*P* hamis értéket vesz fel. A következményrészben szereplő változóknak a cselekvés előfeltételei között is szerepelni kell.

Az olvashatóság javítása érdekében néhány tervkészítő következményrészt szétválasztja egy **hozzáadás listá**ra (**add list**) a pozitív és egy **törlés listá**ra (**delete list**) a negatív literáloknak.

Most hogy a tervkészítők reprezentációjának szintaxisát definiáltuk, adjuk meg a szemantikát is. Ennek legegyszerűbb módja, ha leírjuk, hogy a cselekvések hogyan módosítják az állapotot. (Egy másik lehetséges módszer, hogy egy direkt fordítást specifikálunk a következő állapot axiómákra, melyek szemantikája az elsőrendű logikából származik. Lásd 11.3. feladat.) Először is azt mondjuk, hogy egy cselekvés **alkalmazható** (**applicable**) minden állapotban, ami kielégíti az előfeltételeket; egyébként a cselekvés hatástalan. Egy elsőrendű séma esetében az alkalmazhatóság elérése az előfeltételek egy θ behelyettesítését vonja maga után. Tegyük fel például, hogy a jelen állapot leírása:

$$Ott(P_1, JFK) \land Ott(P_2, SFO) \land Repülő(P_1) \land Repülő(P_2)$$

$$\land Repülőtér(JFK) \land Repülőtér(SFO)$$

Ez teljesíti az

$$Ott(p, honnan) \land Repülő(p) \land Repülőtér(honnan) \land Repülőtér(hova)$$

előfeltételt a $\{p/P_1, honnan/JFK, hova/SFO\}$ behelyettesítésekkel (és másokkal is – lásd 11.2. feladat). Így a konkrét $Repül(P_1, JFK, SFO)$ cselekvés alkalmazható.

Az *s* állapotból kiindulva az alkalmazható *a* cselekvés végrehajtásának *eredménye* az *s*' állapot, ami azonos *s*-sel, kivéve, hogy az *a* cselekvés következményrészében szereplő pozitív *P* literálokat az *s*'-höz adjuk, míg bármilyen ¬*P* negatív literált eltávolítjuk *s*'-ből. Így a $Repül(P_1, JFK, SFO)$ cselekvés után az állapot a következő:

$$Ott(P_1, SFO) \land Ott(P_2, SFO) \land Repülő(P_1) \land Repülő(P_2)$$

$$\land Repülőtér(JFK) \land Repülőtér(SFO)$$

Vegyük észre, hogy a már szereplő pozitív következményeket nem szúrjuk be még egyszer, illetve ha egy negatív literál nem szerepel az állapotleírásban, akkor a következmény ezen része figyelmen kívül hagyható. Ez a definíció testesíti meg az úgynevezett `STRIPS` **feltételezést** (**assumption**): minden a következményben nem szereplő literál változatlan marad. Így a `STRIPS` elkerüli a 10. fejezetben bemutatott **reprezentációs keret problémá**t (**representional frame problem**).

Végezetül definiálhatjuk a tervkészítési probléma **megoldás**át (**solution**). Legegyszerűbb formájában ez csak egy cselekvéssorozat, melyet a kiindulási állapotból végrehajtva a célállapotot eredményezi. A fejezet további részeiben a megoldások cselekvések részben rendezett sorozatai is lehetnek, amennyiben minden cselekvéssorozat, ami megfelel ennek a részben rendezésnek, megoldás.

1.2. Kifejezőképesség és kiterjesztések

A sokféle megkötés, korlátozás amit a `STRIPS` nyelv tartalmaz, abban a reményben került beépítésre, hogy a tervkészítő algoritmusok egyszerűbbek és hatékonyabbak lehessenek, anélkül hogy a valós problémák leírását megnehezítenék. Egyike a legfontosabb megkötéseknek, hogy a literáloknak *függvénymenteseknek* kell lenniük. Ezzel a megkötéssel biztosíthatjuk, hogy egy adott problémához tartozó bármely akció séma ítéletkalkulus formára, azaz változómentes ítéletlogikai cselekvés reprezentációk véges halmazára hozható. (A téma bővebb leírását lásd a 9. fejezetben.) Például a légi szállítási problémakörben 10 repülő és 5 repülőtér esetén a *Repül*(*p, honnan, hova*) séma $10 \times 5 \times 5 = 250$ ítéletlogikai cselekvésre fordítható. A 11.4. és 11.5. alfejezet tervkészítői közvetlenül az ítéletkalkulusra hozott leírással dolgoznak. Ha függvényszimbólumokat is megengedünk, akkor végtelen sok állapot és cselekvés határozható meg.

Created by XMLmind XSL-FO Converter.

11.1. ábra - A *Strips* és az ADL nyelv összehasonlítása a tervkészítési feladatok reprezentációjának szempontjából. Mindkét esetben a célok úgy viselkednek, mint egy paraméterek nélküli cselekvés előfeltételei.

STRIPS **nyelv**	ADL **nyelv**
Csak pozitív literálok az állapotokban: *Szegény* ∧ *Ismeretlen*	Pozitív és negatív literálok az állapotokban: ¬ *Gazdag* ∧ ¬ *Híres*
Zárt világ feltételezés: A nem szereplő literálok hamisak.	Nyílt világ feltételezés: A nem szereplő literálok meghatározatlanok.
A $P \wedge \neg Q$ jelentése: hozzáadjuk P-t, és töröljük Q-t.	A $P \wedge \neg Q$ jelentése: hozzáadjuk P-t és $\neg Q$-t, valamint töröljük $\neg P$-t és Q-t.
A célokban csak alapliterálok szerepelnek: *Gazdag* ∧ *Híres*	Kvantifikált változók a célokban: $\exists x\ Ott(P_1, x) \wedge Ott(P_2, x)$ cél azt adja meg, hogy P_1 és P_2 azonos helyen van.
A célok konjunkciók: *Gazdag* ∧ *Híres*	A célok konjunkciókat és diszjunkciókat is tartalmazhatnak: ¬ *Szegény* ∧ (*Híres* ∨ *Okos*)
A következmények konjunkciók.	Feltételes következmények is megengedettek: **amikor** P: E jelentése, hogy E akkor érvényes, ha P teljesül.
Nincs támogatás az egyenlőségvizsgálathoz.	Beépített egyenlőség ($x = y$) predikátum.
Nincs támogatás típusokhoz.	A változóknak lehet típusa, mint (p : *Repülő*).

Napjainkra, nyilvánvalóvá vált, hogy a STRIPS nem eléggé kifejező néhány valós problémakörhöz. Ennek eredményeképpen számos nyelvváltozatot dolgoztak ki. A 11.1. ábra röviden összefoglalja az egyik legfontosabbat, a **cselekvésleíró nyelv**et (**Action Description Language** – **ADL**) úgy, hogy összehasonlítja azt a STRIPS alapverziójával. ADL nyelven a *Repülés* leírása az alábbi:

```
Cselekvés(Repül(p : Repülő, honnan : Repülőtér, hova : Repülőtér),

Előfeltétel: Ott(p, honnan) ∧ (honnan ≠ hova)

Következmény: ¬Ott(p, honnan) ∧ Ott(p, hova))
```

A p : *Repülő* írásmód az előfeltételek a paraméterlistájában a *Repülő*(p) egy rövidítése, ami nem növeli a kifejezőképességet, de javítja az olvashatóságot. (Mindemellett redukálja a létrehozható ítéletlogikai cselekvések számát.) A (*honnan* ≠ *hova*) előfeltétel azt a tényt fejezi ki, hogy egy repülőút kiindulási és célállomása nem lehet azonos. Ezt a STRIPS nyelvben nem lehetne tömören kifejezni.

A mesterséges intelligenciában használt változatos tervkészítő formalizmusokat egy szabványos szintaxisba rendszerezik, amit **tervkészítési terület definíciós nyelv**nek (**Planning Domain Definition Language** – **PDDL**) neveznek. Ez a nyelv lehetővé teszi a kutatók számára, hogy benchmark problémákat cseréljenek ki egymás között, és összevessék az eredményeket. A PDDL résznyelveket tartalmaz az ADL és a 12. fejezetben bemutatásra kerülő hierarchikus feladathálózatok számára.

A STRIPS és az ADL jelölésrendszere számos valós problémakörre megfelelő. A következő alfejezetek néhány egyszerű példát mutatnak be. Néhány számottevő megkötés azért még megmaradt. A legnyilvánvalóbb, hogy közvetlenül nem tartalmazzák a cselekvések **véghatás**ait (**ramifications**). Például ha vannak emberek, csomagok vagy porcicák egy repülőn, akkor mindannyian helyet változtatnak egy repülés során. Ezeket a változásokat leírhatjuk, mint a repülés egyenes következményeit, így természetesebbnek tűnik a repülőgép tartalmának helyét, mint a gép helyének logikai következményét ábrázolni. Az ilyen **állapotmegkötés**ekre (**state constraint**s) a 11.5. alfejezet tartalmaz példákat. A hagyományos tervkészítő rendszerek meg sem próbálják megoldani a **kvalifikációs problémá**t (**qualification problem**), azaz a nem reprezentált körülmények problémáját, melyek a cselekvés meghiúsulását okozhatják. A 12. fejezetben látni fogjuk, hogy a kvalifikációs probléma hogyan közelíthető meg.

Created by XMLmind XSL-FO Converter.

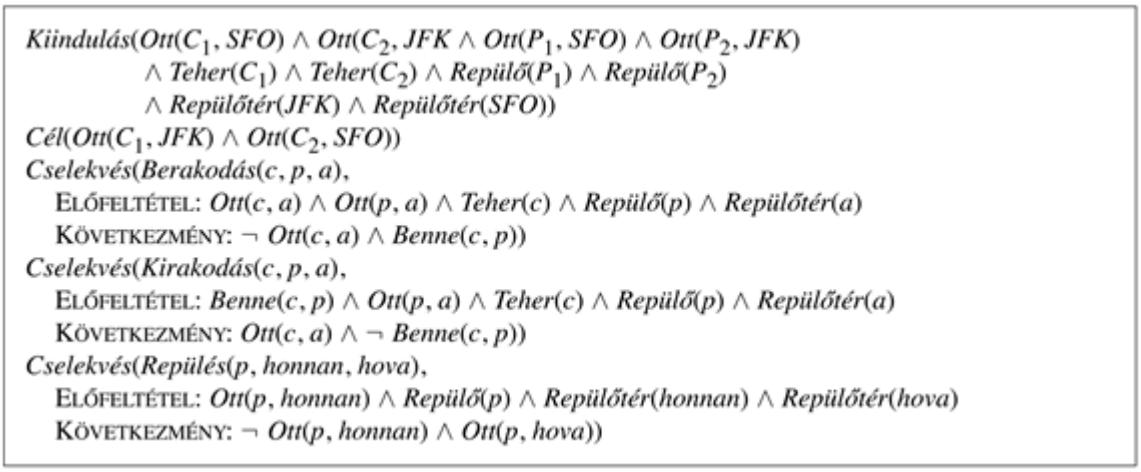
1.3. Példa: Légi teherszállítás

A 11.2. ábra egy teherszállítási problémát mutat be, ami a teher be-, illetve kirakodását és állomások közötti légi szállítását tartalmazza. A probléma három cselekvéssel írható le: *Berakodás*, *Kirakodás* és *Repülés*. A cselekvések két predikátumot érintenek: a *Benne*(*c*, *p*) jelentése, hogy a *c* teher a *p* repülőgépben van, és az *Ott*(*x*, *a*) jelentése, hogy az *x* objektum (teher vagy repülőgép) az *a* repülőtéren található. Vegyük észre, hogy a teher nincs *Ott* sehol, ha *Benne* van egy repülőgépben, azaz az *Ott* valójában azt jelenti, hogy az objektum „elérhető a megadott helyen”. Tapasztalattal kell rendelkezni a cselekvésdefiníciók területén, ahhoz hogy az ilyen részleteket konzisztensen ábrázoljuk. A következő terv megoldása a feladatnak:

$$[Berakodás(C_1, P_1, SFO), Repülés(P_1, SFO, JFK, Kirakodás(C_1, P_1, JFK)$$

$$Berakodás(C_2, P_2, JFK), Repülés(P_2, JFK, SFO, Kirakodás(C_2, P_2, SFO)]$$

11.2. ábra - A STRIPS-probléma repülőterek közötti légi teherszállítási feladathoz



A mi reprezentációnk tisztán STRIPS nyelvű. Nevezetesen ez engedélyezi, hogy egy repülő kiinduló és célállomása azonos repülőtér legyen. Az ADL egyenlőtlenség operátorai ezt kizárhatnák.

1.4. Példa: A pótkerék probléma

Vegyük a kerékcsere problémáját. Pontosabban a célunk az, hogy egy jó pótkerék legyen felszerelve az autó tengelyére, ahol a kiinduló állapotban egy lapos kerék van felszerelve, míg a jó pótkerék a csomagtartóban található. Az egyszerűség kedvéért a mi feladatunk elég absztrakt, azaz nincsenek beragadt csavarok vagy egyéb más nehézségek. Csak négy cselekvés van: a pótkerék kivétele a csomagtartóból, a lapos kerék eltávolítása a tengelyről, a pótkerék felszerelése és az autó magára hagyása reggelig. Feltételezzük, hogy az autó egy igen rossz környéken áll, ahol az autó magára hagyása azt eredményezi, hogy reggelre eltűnnek a kerekek.

A 11.3. ábra a feladat ADL leírását tartalmazza. Vegyük észre, hogy ez a megadás pusztán ítéletlogikai. A STRIPS nyelven túlmutat, hogy a *TeddFel*(*Pótkerék*, *Tengely*) cselekvéshez a ¬*Ott*(*Pótkerék*, *Tengely*) negált előfeltétel tartozik. Ezt elkerülhetnénk egy *SzabaddáTesz*(*Tengely*) használatával, amint azt a következő példában látni fogjuk.

11.3. ábra - Az egyszerű pótkerék probléma

Created by XMLmind XSL-FO Converter.

Kiindulás(*Ott*(*LaposKerék*, *Tengely*) ∧ *Ott*(*Pótkerék*, *Csomagtartó*)
Cél(*Ott*(*Pótkerék*, *Tengely*))
Cselekvés(*Eltávolít*(*Pótkerék*, *Csomagtartó*),
ELŐFELTÉTEL: *Ott*(*Pótkerék*, *Csomagtartó*)
KÖVETKEZMÉNY: ¬ *Ott*(*Pótkerék*, *Csomagtartó*) ∧ *Ott*(*Pótkerék*, *Föld*))
Cselekvés(*Eltávolít*(*LaposKerék*, *Tengely*),
ELŐFELTÉTEL: *Ott*(*LaposKerék*, *Tengely*)
KÖVETKEZMÉNY: ¬ *Ott*(*LaposKerék*, *Tengely*) ∧ *Ott*(*LaposKerék*, *Föld*))
Cselekvés(*Felszerel*(*Pótkerék*, *Tengely*),
ELŐFELTÉTEL: *Ott*(*Pótkerék*, *Föld*) ∧ ¬ *Ott*(*LaposKerék*, *Tengely*)
KÖVETKEZMÉNY: ¬ *Ott*(*Pótkerék*, *Föld*) ∧ *Ott*(*Pótkerék*, *Tengely*))
Cselekvés(*OtthagyÉjszakára*,
ELŐFELTÉTEL:
KÖVETKEZMÉNY: ¬ *Ott*(*Pótkerék*, *Föld*) ∧ ¬ *Ott*(*Pótkerék*, *Tengely*) ∧ ¬ *Ott*(*Pótkerék*, *Csomagtaró*)
∧ ¬ *Ott*(*LaposKerék*, *Föld*) ∧ ¬ *Ott*(*Laposkerék*, *Tengely*))

1.5. Példa: A kockavilág

Az egyik leghíresebb tervkészítési terület a **kockavilág** (**blocks world**) probléma. A terület asztallapon elhelyezett kockákból áll.[3] A kockákat egymásra rakhatjuk, de egy kockán közvetlenül mindig csak egyetlen másik helyezhető el. A kockákat egy robotkarral mozgathatjuk, amely fel tud venni egy kockát, majd azt vagy az asztalra, vagy egy másik kocka tetejére le tudja tenni. A robotkar egyszerre csak egy kockát tud felemelni, vagyis olyat nem, amelynek a tetején egy másik kocka van. A cél mindig egy vagy több kockaoszlop építése, amelyekben a kockák egymáshoz képesti elhelyezkedése meghatározott. A cél lehet például két oszlop építése, amelyek közül az egyikben az *A* kocka a *B* tetején van, a másikban pedig a *C* kocka van a *D* tetején.

A *Rajta*(*b*, *x*) jelölést használjuk annak leírására, hogy a *b* kocka az *x*-en van, ahol az *x* egy másik kockát vagy az asztallapot jelenti. A *Mozgat*(*b*, *x*, *y*) cselekvés a *b* kockát a *x* tetejéről az *y* tetejére mozgatja. A *b* kocka mozgatásának előfeltétele, hogy rajta semmi ne legyen. Ennek leírása az elsőrendű logikában a ¬∃ *x Rajta*(*x*, *b*) vagy ∀*x* ¬*Rajta*(*x*, *b*). Az ADL nyelvben ezek előfeltételek lehetnének. A STRIPS nyelv keretei között maradhatunk az *Üres*(*x*) predikátum bevezetésével, ami akkor igaz, ha semmi nincs *x*-en.

A *Mozgat* cselekvés a *b* kockát az *x*-ről az *y*-ra mozgatja, ha mind a *b*, mind pedig az *y* üres. A mozgatás után az *x* üres, de az *y* már nem. A *Mozgat* formális leírása STRIPS-ben a következő:

```
Cselekvés(Mozgat(b, x, y),

Előfeltétel: Rajta(b, x) ∧ Üres(b) ∧ Üres(y)

Következmény: Rajta(b, y) ∧ Üres(x) ∧ ¬Rajta(b, x) ∧ ¬Üres(y))
```

Sajnos ez a cselekvés nem kezeli jól az *Üres* predikátumot, ha az *x* vagy az *y* az asztalon van.

Ha *x* = *Asztal*, a cselekvés következményei között szerepel az *Üres*(*Asztal*) is, de az asztalnak nem kell kiürülnie a mozgatás után, ha pedig *y* = *Asztal*, megjelenik az *Üres*(*Asztal*) előfeltétel, holott az asztalnak nem kell üresnek lenni ahhoz, hogy bármit is tehessünk rá. Ennek kiküszöbölésére két dolgot tehetünk. Először is bevezetünk egy új cselekvést, amellyel egy *b* kockát az asztalra tehetünk:

```
Cselekvés(AsztalraTesz(b, x, y),

Előfeltétel: Rajta(b, x) ∧ Üres(b)

Következmény: Rajta(b, Asztal) ∧ Üres(x) ∧ ¬Rajta(b, x))
```

Másodszor az *Üres*(*b*) predikátumot úgy értelmezhetjük, hogy „*b* tetején van elég szabad hely, ahová a kockát letehetjük”. Ezek szerint az *Üres*(*Asztal*) mindig igaz. Az egyetlen probléma, hogy semmi nem gátolja a tervkészítőt abban, hogy a *Mozgat*(*b*, *x*, *Asztal*) cselekvést alkalmazza az *AsztalraTesz*(*b*, *x*) helyett. Vagy együtt

[3] A tervkészítés kutatásában használt kockavilág sokkal egyszerűbb, mint az SHRDLU verzió, amit az 3.3. szakasz - Korai lelkesedés, nagy elvárások (1952–1969) részben mutattunk be.

Created by XMLmind XSL-FO Converter.

élünk ezzel a problémával (amely egyébként a szükségesnél nagyobb keresési teret eredményez, de nem vezet hibás válaszokhoz), vagy bevezethetjük a *Kocka* predikátumot, és a *Mozgat* cselekvés előfeltételeihez hozzátehetjük a $Kocka(x) \wedge Kocka(b)$ részt.

11.4. ábra - A kockavilág tervkészítési problémája: egy három kockából álló torony építése. A [*Mozgat*(*B, Asztal, C*), *Mozgat*(*A, Asztal, B*)] cselekvéssor egy lehetséges megoldás.

$$
\begin{aligned}
&\mathit{Kiindulás}(\mathit{Rajta}(A, \mathit{Asztal}) \wedge \mathit{Rajta}(B, \mathit{Asztal}) \wedge \mathit{Rajta}(C, \mathit{Asztal}) \\
&\qquad \wedge \mathit{Kocka}(A) \wedge \mathit{Kocka}(B) \wedge \mathit{Kocka}(C) \\
&\qquad \wedge \mathit{Üres}(A) \wedge \mathit{Üres}(B) \wedge \mathit{Üres}(C)) \\
&\mathit{Cél}(\mathit{Rajta}(A, B) \wedge \mathit{Rajta}(B, C)) \\
&\mathit{Cselekvés}(\mathit{Mozgat}(b, x, y), \\
&\quad \text{Előfeltétel: } \mathit{Rajta}(b, x) \wedge \mathit{Üres}(b) \wedge \mathit{Üres}(y) \wedge \mathit{Kocka}(b) \wedge \\
&\qquad (b \neq x) \wedge (b \neq y) \wedge (x \neq y) \\
&\quad \text{Következmény: } \mathit{Rajta}(b, y) \wedge \mathit{Üres}(x) \wedge \neg\, \mathit{Rajta}(b, x) \wedge \neg\, \mathit{Üres}(y)) \\
&\mathit{Cselekvés}(\mathit{AsztalraTesz}(b, x), \\
&\quad \text{Előfeltétel: } \mathit{Rajta}(b, x) \wedge \mathit{Üres}(b) \wedge \mathit{Kocka}(b) \wedge (b \neq x) \\
&\quad \text{Következmény: } \mathit{Rajta}(b, \mathit{Asztal}) \wedge \mathit{Üres}(x) \wedge \neg\, \mathit{Rajta}(b, x))
\end{aligned}
$$

Végül itt van még az olyan hibás műveletek esete, mint a *Mozgat*(*B, C, C*), amely hatástalan kellene legyen, ehelyett azonban ellentmondó következményekhez vezet. Általában nem foglalkozunk az ilyen jellegű problémákkal, mert nemigen vannak hatással az előállított tervekre. A helyes megközelítés, hogy egyenlőtlenségi előfeltételeket vezetünk be, amint azt a 11.4. ábra mutatja.

2. Tervkészítés állapottér-kereséssel

Most fordítsuk figyelmünket a tervkészítő algoritmusok felé. A legkézenfekvőbb megközelítés az állapottér-keresés. Mivel a tervkészítési problémák cselekvéseinek leírása tartalmazza az előfeltételeket és a következményeket, a keresés mindkét irányban végrehajtható: vagy előrefelé egy kiindulási állapotból, vagy visszafelé a céltól, amint azt a 11.5. ábra is mutatja. Mindezeken túl felhasználhatjuk az explicit cselekvés- és célreprezentációkat, hogy automatikusan hatékony heurisztikákat származtassunk.

2.1. Előrefelé keresés az állapottérben

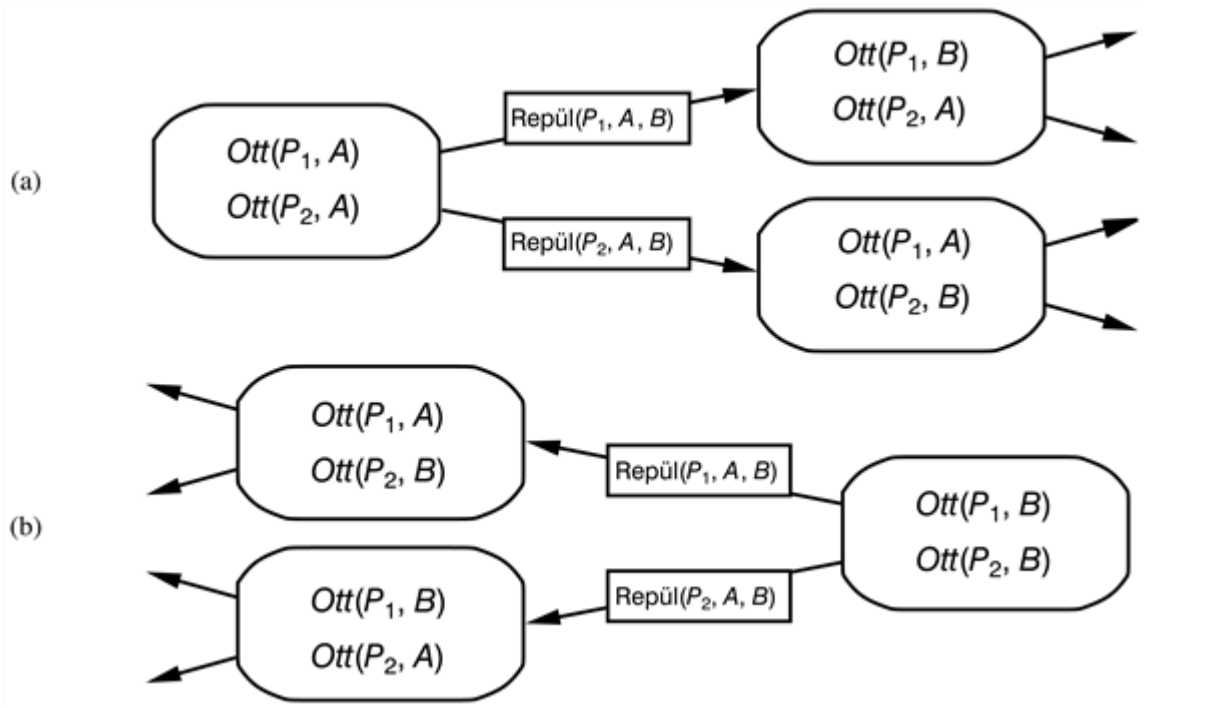
Ez előrefelé kereséssel történő tervkészítés hasonlít a 3. fejezetben bemutatott problémamegoldó megközelítéshez. Ezt progresszív tervkészítésnek nevezzük, mert előrefelé haladva dolgozik.

A feladat kiinduló állapotából indulva végignézzük a lehetséges cselekvéssorozatokat, amíg olyat nem találunk, ami a célállapothoz vezet. A tervkészítési probléma állapottér-keresési problémaként formalizálva a következő:

- A keresés **kiindulási állapot**a (**initial state**) megegyezik a tervkészítési probléma kiindulási állapotával. Általánosságban minden állapot pozitív alapliterálok egy halmaza; az itt nem szereplő literálok hamisak.
- Egy állapotban minden olyan **cselekvés** (**action**) alkalmazható, aminek az előfeltételei teljesülnek. A cselekvés végrehajtása után következő állapotot úgy állítjuk elő, hogy a következményrész pozitív literáljait hozzáadjuk, negatív literáljait pedig töröljük. (Elsőrendű logika esetén az előfeltételekből az egyesítőt alkalmazni kell a következményliterálokra.) Megjegyezzük, hogy annak következményeként, hogy egy explicit cselekvés reprezentációt használunk, egyetlen állapotátmenet-függvény működik az összes tervkészítő problémára.
- A **célteszt** (**goal test**) ellenőrzi, hogy az adott állapot kielégíti-e a tervkészítési probléma célját.
- A **lépésköltség** (**step cost**) minden cselekvésre tipikusan 1. Habár könnyű lenne a különböző cselekvésekhez különböző költségeket hozzárendelni, ezt a `STRIPS` tervkészítőkben nagyon ritkán alkalmazzák.

Created by XMLmind XSL-FO Converter.

11.5. ábra - A tervkészítés két megközelítése. (a) Előrefelé (progresszív) állapottér-keresés, a kiinduló állapotból indulva és a probléma cselekvéseit használva halad a cél felé. (b) Visszafelé (regressziós) állapottér-keresés: valószínűségi állapot keresés (lásd 6.1. szakasz - Szenzor nélküli problémák részben) a célállapot(ok)ból indulva a cselekvések inverzét alkalmazva keresi visszafelé a kezdeti állapotot.



Emlékezzünk vissza, hogy a függvényszimbólumok hiányában a tervkészítési probléma állapottere véges, ezért teljes gráf keresési algoritmusok, mint például az A* keresés, egyben teljes tervkészítő algoritmusok is.

A tervkészítők kutatásának korai napjaitól (kb. 1961-től) egészen mostanáig (kb. 1998-ig) az a feltételezés élt, hogy az előrefelé kereső algoritmusok nem eléggé hatékonyak a gyakorlati alkalmazásra. Ma már, tekintsünk csak vissza a 11.1. alfejezetre, ezt nem nehéz megindokolni. Először is az előrefelé keresés nem foglalkozik a lényegtelen cselekvések problémájával, mivel minden állapotban minden alkalmazható cselekvést figyelembe vesz. Másodszor pedig ez a megközelítés gyorsan elakad egy jó heurisztika nélkül. Vegyük például a légi csomagszállítási problémát 10 repülőtérrel, egyenként 5-5 repülővel és 20 darab csomaggal. A cél, hogy az *A* repülőtér összes terhét a *B* repülőtérre szállítsuk. Van egy egyszerű megoldása a feladatnak: tegyük be mind a 20 csomagot az egyik *A*-beli repülőgépbe, ezzel repüljünk *B*-be, és ott rakodjunk ki. A megoldás megkeresése ellenben nagyon nehéz lehet, hiszen az átlagos elágazási faktor óriási: mind az 50 repülő 9 másik reptérre tud repülni, míg mind a 200 csomag kirakható (ha be volt rakva), vagy berakható a reptér bármelyik repülőjébe. Átlagosan mondjuk 1000 lehetséges cselekvés hajtható végre, így a keresési fában a kézenfekvő megoldás mélységéig közel 1000[41] csomópont van. Ebből nyilvánvaló, hogy egy megfelelő heurisztikára lesz szükség, hogy ezt a keresést hatékonnyá tegyük. A hátrafelé keresés áttekintése után néhány lehetséges heurisztikát mutatunk be.

2.2. Hátrafelé keresés az állapottérben

Az állapottérben történő hátrafelé keresést a kétirányú keresés egy részeként a 3. fejezetben röviden már bemutattuk. Ott megjegyeztük, hogy a hátrafelé keresés megvalósítása nagyon nehéz lehet, ha a célállapotokat az explicit megadás helyett megkötésekkel határozzuk meg. Nevezetesen nem mindig nyilvánvaló, hogy hogyan generáljuk le a célállapotok halmazának a lehetséges **elődállapot**ait (**predecessor**s). Látjuk majd, hogy a `STRIPS` reprezentáció esetén ez meglehetősen egyszerű, mert az állapothalmazokat azokkal a literálokkal írhatjuk le, amelyeknek igaz értékűeknek kell lenni az adott állapotokban.

A hátrafelé keresés fő előnye az, hogy lehetővé teszi számunkra, hogy csak a **releváns** (**relevant**) cselekvéseket vegyük figyelembe. Egy cselekvés releváns egy konjunktív cél szempontjából, ha eléri a cél egy konjunktját.

Created by XMLmind XSL-FO Converter.

Például a 10-repülőteres légi csomagszállítási problémánk esetében a cél az, hogy a 20 csomag a *B* repülőtéren legyen, vagy pontosabban

$$Ott(C_1, B) \wedge Ott(C_2, B) \wedge \ldots \wedge Ott(C_{20}, B)$$

Vegyük az $Ott(C_1, B)$ literált. Hátrafelé dolgozva, olyan cselekvéseket kereshetünk, melyeknek ez a következménye. Ilyen csak egy van: $Kirakodás(C_1, p, B)$, ahol a *p* repülőgép meghatározatlan.

Vegyük észre, hogy számos *irreleváns* cselekvés is van, ami szintén a célállapotra vezet. Például elrepíthetünk egy üres gépet a *JFK* repülőtérről (John Fitzgerald Kennedy repülőtér New Yorkban) az *SFO*-ra (San Francisco repülőtere), ami a célállapothoz vezet egy olyan elődállapotból, melyben a repülőgép a *JFK*-n van, és minden célfeltétel teljesül. Egy hátrafelé keresés, ami megengedi a hatástalan cselekvéseket még teljes, de sokkal kevésbé hatékony. Ha létezik megoldás, akkor azt egy olyan hátrafelé keresés is megtalálja, amely csak releváns cselekvéseket enged meg. A hátrafelé keresés szűkítése a releváns cselekvésekre, gyakran jóval alacsonyabb elágazási tényezőt eredményez, mint az előrefelé keresés. Például a légi szállítási problémánk esetén a kiindulási állapotból előrefelé közel 1000 cselekvés hajtható végre, míg a célból visszafelé indulva mindössze 20.

A visszafelé keresést **regresszió**s (**regression**) tervkészítésnek is nevezik. A regressziós keresés meghatározó kérdése a következő: melyek azok az állapotok, amelyekből egy adott cselekvés a célhoz vezet? Ezen állapotok leírásának számítását a cél adott állapoton keresztüli **regresszálás**ának (**regressing**) nevezzük. Hogy lássuk ennek menetét, vegyük a légi szállítási feladatot. Adott a cél az

$$Ott(C_1, B) \wedge Ott(C_2, B) \wedge \ldots \wedge Ott(C_{20}, B)$$

és a releváns cselekvés *a* $Kirakodás(C_1, p, B)$, ami az első literált adja. A cselekvés csak akkor működik, ha az előfeltételei teljesülnek. Ebből kifolyólag az elődállapotnak tartalmaznia kell a $Benne(C_1, p) \wedge Ott(p, B)$ előfeltételeket. Mindezeken túl az $Ott(C_1, B)$ részcél nem lehet igaz a megelőző állapotban.[4] Így a megelőző állapot leírása az alábbi:

$$Benne(C_1, p) \wedge Ott(p, B) \wedge Ott(C_2, B) \wedge \ldots \wedge Ott(C_{20}, B)$$

Ennek tetejébe, ha elvárjuk, hogy egy cselekvés kielégítsen egy literált, azt is biztosítanunk kell, hogy nem *ront el* egy másikat. Az olyan cselekvést, ami megfelel ennek az elvárásnak **konzisztens**nek (**consistent**) nevezzük. Például a $Berakodás(C_2, p)$ cselekvés nem lenne konzisztens az aktuális céllal, mert negálja az $Ott(C_2, B)$ literált.

Most, hogy a relevanciát és a konzisztenciát definiáltuk, megadhatunk egy általános módszert a hátrafelé keresés elődállapotának kereséséhez. Adott egy *G* cél leírása. Legyen az *A* cselekvés releváns és konzisztens. A megfelelő elődállapot a következő:

- Az *A* minden pozitív következményét, ami szerepel *G*-ben, töröljük.
- Az *A* minden, még nem szereplő előfeltételét hozzáadjuk.

A standard keresési algoritmusok bármelyike felhasználható a keresésre. A leállási feltétel, hogy a generált elődállapot leírását a keresés kiindulási állapota kielégítse. Az elsőrendű esetben, ehhez néhány változó behelyettesítésére is szükség lehet az elődállapotának leírásában. Például az előző bekezdés elődállapotának leírását a

$$Benne(C_1, P_{12}) \wedge Ott(P_{12}, B) \wedge Ott(C_2, B) \wedge \ldots \wedge Ott(C_{20}, B)$$

kiindulási állapot a $\{p/P_{12}\}$ behelyettesítéssel teljesíti. A behelyettesítéseket arra a cselekvésre kell végrehajtani, ami az állapottól a célhoz vezet, így eredményezve a $[Kirakodás(C_1, P_{12}, B)]$ megoldást.

2.3. Állapottér-keresési heurisztikák

Sem az előre-, sem pedig a hátrafelé keresés nem hatékony egy jó heurisztika nélkül. Emlékezzünk vissza a 4. fejezetből, hogy egy heurisztikus függvény egy állapot célállapottól való távolságát becsli; a `STRIPS` esetén minden cselekvés költsége 1, így a távolság az alkalmazandó cselekvések száma. Az alapötlet, hogy nézzük a cselekvés következményeit és a célokat, és becsüljük meg, hogy hány lépésre van szükség a célok elérésére. A

[4] Ha a részcél igaz lenne az elődállapotban, akkor a cselekvés még mindig a célállapothoz vezetne. Ellenben az ilyen cselekvések hatástalanok (irrelevánsak), mivel nem *teszik* igazzá a célt.

Created by XMLmind XSL-FO Converter.

pontos szám meghatározása NP-teljes probléma, de a legtöbb esetben találhatók kellően pontos becslők, melyek nem nagyon számításigényesek. Szintén képesek lehetünk egy **elfogadható** (**admissable**) heurisztika származtatására, azaz olyanra, ami nem becsül túl. Ez az A* kereséssel együtt használható optimális megoldások megkeresésére.

Két megközelítést lehet kipróbálni. Az egyik, hogy a megadott feladatleírásból egy **relaxált problémá**t (**relaxed problem**) származtatunk, ahogy azt a 4. fejezetben bemutattuk. A relaxált problémára (ami várhatóan könnyen megoldható) adott optimális megoldás költsége elfogadható heurisztikát ad az eredeti problémára. A második megközelítés során feltételezzük, hogy a tiszta „oszd meg és uralkodj” algoritmus működni fog. Ezt nevezzük a **részcélfüggetlenség** (**subgoal independence**) feltételezésnek: a részcélok konjunkciójának megoldási költségét, az egymástól *függetlenül* megoldott részcélok költségeinek összegével becsüljük. A részcélfüggetlenség feltételezés lehet optimista vagy pesszimista. Akkor optimista, hogyha vannak negatív kölcsönhatások a részcélokhoz készített résztervek között (például amikor az egyik részterv egy cselekvése töröl egy célt, melyet egy másik részterv ért el). Akkor pesszimista és egyben elfogadhatatlan is, ha a résztervek redundáns cselekvéseket tartalmaznak (például két cselekvés eggyel helyettesíthető az összefésült tervben).

Vizsgáljuk meg, hogyan származtathatunk relaxált tervkészítési problémákat. Mivel az előfeltételek és a következmények explicit megadásai rendelkezésre állnak, az eljárás ezeket a reprezentációkat módosítja. (Vessük össze ezt a megközelítést a keresési feladatokkal, ahol az állapotátmenet-függvény egy fekete doboz.) A legegyszerűbb ötlet, hogy lazítsuk a problémát úgy, hogy *eltávolítjuk az összes előfeltételt* a cselekvésekből. Ekkor minden cselekvés mindig alkalmazhatóvá válik, és minden literál elérhető egy lépésben (ha van alkalmazható cselekvés, egyébként a cél elérése lehetetlen). Úgy tűnhet, ez azt jelenti, hogy a célok konjunkciójához vezető lépésszám azonos a kielégítetlen célliterálok számával, de nem egészen; (1) lehet két cselekvés, melyek kölcsönösen törlik a másik által teljesített célliterált, (2) néhány cselekvés több célt is elérhet. Ha kombináljuk a relaxált problémát és a részcél függetlenségi feltételezést, mindkét problémát kiküszöböltnek vehetjük, és az előálló heurisztika pontosan a még kielégítetlen célok száma.

Sokszor pontosabb heurisztika nyerhető, ha legalább a több célt kielégítő cselekvések pozitív kölcsönhatásait figyelembe vesszük. Először tovább relaxáljuk a problémát azáltal, hogy *eltávolítjuk a negatív következményeket* (lásd 11.6. feladat). Ezután kiszámítjuk, hogy mennyi az a minimális cselekvésszám, melyek pozitív következményeinek uniója kielégíti a célt. Vegyük például a

```
Cél(A ∧ B ∧ C)

Cselekvés(X, Következmény: A ∧ P)

Cselekvés(Y, Következmény: B ∧ C ∧ Q)

Cselekvés(Z, Következmény: B ∧ P ∧ Q)
```

A minimális halmaz az {*X*, *Y*} cselekvés, ami az {*A*, *B*, *C*} célt kielégíti, így a fedőhalmaz-heurisztika szerinti költség 2. Ez javít a részcél függetlenség feltételezésen, ami 3-as költséget adna. Egyetlen apró probléma van: a fedőhalmaz probléma NP-teljes. Az egyszerű fedőhalmaz-keresési algoritmus garantáltan visszaad egy értéket, ami egy log *n*-es faktor pontossággal az igazi minimumérték, ahol *n* a cél literáljainak száma, és a gyakorlatban rendszerint ennél sokkal pontosabb. Sajnos a mohó algoritmus elveszíti a heurisztika elfogadhatóságának garanciáját.

Az is lehetséges, hogy a relaxált problémát a negatív következmények eltávolításával állítjuk elő anélkül, hogy az előfeltételeket elhagynánk. Ez annyit tesz, hogy ha egy cselekvés következménye $A \wedge \neg B$ az eredeti problémában, akkor *A* a relaxált problémában. Ez annyit tesz, hogy nem kell foglalkoznunk a résztervek közötti negatív kölcsönhatásokkal, mert egyetlen cselekvés sem törölhet egy másik által előállított literált. Az így előálló relaxált feladat megoldásának költsége az **üres-törlési-lista** (**empty-delete-list**) heurisztikát adja. A heurisztika meglehetősen pontos, de kiszámítása egy (egyszerű) tervkészítő algoritmus futtatását igényli. A gyakorlatban a relaxált probléma megoldása elég gyors ahhoz, hogy megérje alkalmazni.

Az itt bemutatott heurisztika mind a progresszív (előrefelé), mind pedig a regresszív (hátrafelé) irányba használható. A könyv írásának pillanatában az üres-törlési-lista heurisztikát használó előrefelé keresők a listavezetők. Ez valószínűleg változik, ahogy újabb heurisztikák és keresési technikák jelennek meg. Mivel a tervkészítés exponenciális komplexitású,[5] nincs olyan algoritmus, amely minden problémára hatékony lesz, de

[5] Technikailag a STRIPS-alapú tervkészítés PSPACE-teljes, kivéve, ha a cselekvéseknek csak pozitív előfeltételei vannak egy következmény literállal (Bylander, 1994).

Created by XMLmind XSL-FO Converter.

számos gyakorlati feladat oldható meg ezen fejezet heurisztikáival – sokkal több, mint ami néhány éve megoldható volt.

3. Részben rendezett tervkészítés

Az előre- és hátrafelé keresés a *teljesen rendezett* terv keresés speciális fajtái. Ezek a cselekvéseknek csak szigorúan lineáris sorozatait tárják fel, melyek közvetlenül kapcsolódnak a kiinduláshoz vagy a célhoz. Ez annyit tesz, hogy nem használhatják ki a problémadekompozíció lehetőségeit. Ahelyett hogy a részproblémákon külön dolgoznának, mindig arról kell dönteniük, hogy hogyan sorrendezzék a részproblémák cselekvéseinek összességét. Előnyösebb lenne egy olyan megközelítés, ami több részcélon dolgozik függetlenül, ezeket több résztervvel megoldja, majd összerakja a részterveket.

Egy ilyen megközelítésnek megvan az az előnye is, hogy rugalmas a terv *összeállításának* sorrendjében. A tervkészítő így nem kényszerül arra, hogy a lépéseken időrendi sorrendben dolgozzon, hanem először inkább a „nyilvánvaló” vagy „fontos” döntéseken dolgozhat. Például egy Berkeleyben lévő tervkészítő ágens, amely Monte Carlóba igyekszik, először egy San Francisco–Párizs járatot vizsgál meg, majd amikor ismeri a pontos indulási és érkezési dátumokat, foglalkozhat a reptérre kijutás, illetve az onnan továbbjutás problémájával.

Azt az általános stratégiát, mely szerint egy választást késleltetünk egy keresés során, **legkisebb megkötés** (**least commitment**) stratégiának nevezzük. A legkisebb megkötésnek nincs formális definíciója, és nyilvánvaló, hogy valamennyi megkötés szükséges, különben a keresés nem haladna. Az informális definíció ellenére a legkisebb megkötés nagyon hasznos megközelítés annak meghatározására, hogy a döntéseket mikor kell meghozni egy keresési problémában. Az első példánk jóval egyszerűbb, mint egy nyaralás megtervezése. Vegyük a cipőfelvétel egyszerű esetét. Ezt formálisan a következő tervkészítési problémával írhatjuk le:

```
Cél(JobbCipőFenn ∧ BalCipőFenn)

Kiindulás()

Cselekvés(JobbCipő, Előfeltétel: JobbZokniFenn, Következmény: JobbCipőFenn)

Cselekvés(JobbZokni, Következmény: JobbZokniFenn)

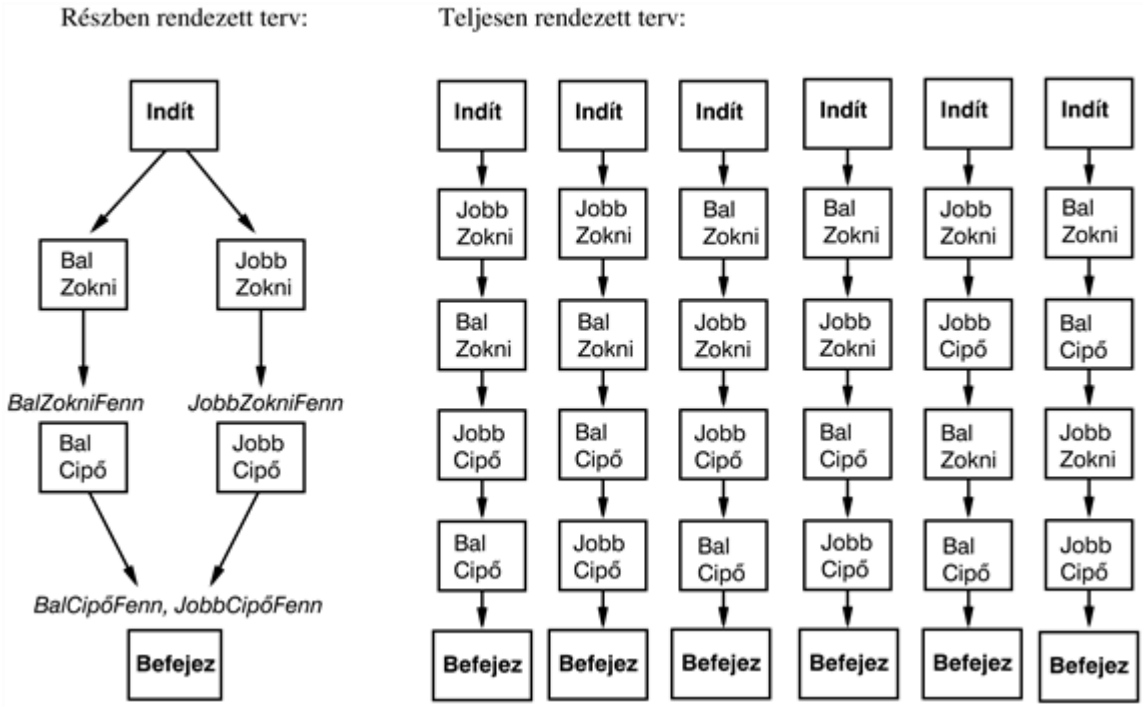
Cselekvés(BalCipő, Előfeltétel: BalZokniFenn, Következmény: BalCipőFenn)

Cselekvés(BalZokni, Következmény: BalZokniFenn)
```

A tervkészítőnek két cselekvéssorozattal kell előállnia, a *JobbZokni*-t a *JobbCipő* követi, hogy a cél első konjunktját elérjük és a *BalZokni*-t a *BalCipő* követi a második konjunkthoz. A két cselekvéssorozat kombinálható, hogy teljes tervet nyerjünk. Így a tervkészítő a két részsorozatot egymástól függetlenül kezelheti anélkül, hogy felállítana bármi megkötést arra, hogy az egyik sorozat egy cselekvése a másikéhoz képest előbb vagy később következik. Bármely olyan tervkészítő algoritmust, ami két cselekvést be tud illeszteni egy tervbe anélkül, hogy meghatározná azok sorrendjét, **részben rendezett tervkészítő**nek nevezünk (**partial-order planner**). A 11.6. ábra a cipő és zokni felvételére készített részben rendezett tervet szemlélteti. Vegyük észre, hogy a megoldást a cselekvések gráfja ábrázolja, nem egy szekvencia. Érdemes megfigyelni az *Indít* és a *Befejez* „technikai” cselekvéseket, melyek a terv kezdetét, illetve végét jelölik. Ezeket cselekvésnek nevezni egyszerűsíti a dolgokat, mert így a terv minden lépése egy cselekvés. A részben rendezett megoldás hat teljesen rendezett megoldásnak felel meg, melyek mindegyike a részben rendezett megoldásnak egy **sorba rendezése** (**linearization**).

11.6. ábra - Egy részben rendezett terv a zokni és a cipő felvételéhez, és a hat lehetséges sorba rendezés a teljesen rendezett tervhez

Created by XMLmind XSL-FO Converter.



A részben rendezett tervkészítés megvalósítható, mint egy keresés a részben rendezett tervek terében. (Ezentúl ezeket egyszerűen csak „terv”-nek nevezzük.) Egy üres tervvel indulunk. Ezután a terv finomításának módjait keressük egészen addig, míg elő nem állunk a teljes tervvel, ami megoldása a problémának. A cselekvések ebben a keresésben nem a való világ cselekvései, hanem terveken végrehajtott műveletek: egy lépés hozzáadása a tervhez, a cselekvésekhez egy sorrend rendelése, ami egy cselekvést egy másik elé helyez és így tovább.

A továbbiakban definiálni fogjuk az RRT algoritmust a részben rendezett tervkészítéshez. Hagyományosan az RRT-t mint egy önálló programot szokás bemutatni, ehelyett mi a részben rendezett tervkészítést, mint a keresési probléma egy példányát fogjuk megfogalmazni. Ez lehetővé teszi, hogy az alkalmazható tervfinomító lépésekre koncentráljunk ahelyett, hogy azt vizsgálnánk, hogyan térképezi fel a keresési teret az algoritmus. Valójában, ha a keresési feladatot megfogalmaztuk a nem informált vagy heurisztikus keresési módszerek széles választéka áll rendelkezésre.

Emlékezzünk vissza, hogy a keresésünk állapotai (többnyire még befejezetlen) tervek lesznek. Hogy elkerüljük a valós cselekvésekkel való keveredést, itt tervekről és nem állapotokról fogunk beszélni. Minden terv a következő négy komponensből áll, ahol az első kettő a terv lépéseit definiálja, míg az utolsó kettő naplózásra szolgál, hogy abból meghatározzuk, hogyan bővíthetők a tervek:

- **Cselekvés**ek (**action**s) halmaza, amelyek a terv lépéseit adják. Ezeket a tervkészítési probléma cselekvéshalmazából vesszük. Egy „üres” terv csak az *Indít* és a *Befejez* cselekvéseket tartalmazza. Az *Indít*-nak nincsenek előfeltételei, míg a következményrésze a tervkészítési feladat kiindulási állapotának literáljait tartalmazza. A *Befejez* cselekvésnek nincsen következmény része, míg előfeltételei megegyeznek a célállapot literáljaival.

- **Rendezési kényszer**ek (**ordering constraint**s). Minden rendezési kényszer $A \prec B$ alakú, amit „*B* előtt *A*”-nak olvasunk, és azt jelenti, hogy az *A*-t valamikor a *B* előtt kell végrehajtani, de nem feltétlenül közvetlen előtte. A rendezési kényszerek egy megfelelő részben rendezést kell leírjanak. Bármilyen ciklus (mint az $A \prec B$ és a $B \prec A$) ellentmondást jelent, ezért olyan rendezési kényszer nem adható a tervhez, amely ciklust hozna létre.

- **Okozati kapcsolatok** (**causal link**s). Az *A* és *B* cselekvések közötti okozati kapcsolatot $A \xrightarrow{p} B$ -vel jelöljük és úgy olvassuk, hogy „*A* teljesíti *p*-t *B*-hez”. Például a $JobbZokni \xrightarrow{JobbZokniFenn} JobbCipő$ okozati kapcsolat jelentése, hogy a *JobbZokniFenn* egy következménye a *JobbZokni* cselekvésnek és előfeltétele a *JobbCipő*-nek. Tartalmazza azt a megkötést is, hogy a *JobbZokniFenn*-nek igaznak kell maradnia a *JobbZokni* cselekvéstől a *JobbCipő* cselekvésig. Más szavakkal, a terv nem mindig bővíthető egy új *C* cselekvéssel, ami **ütközik** egy okozati kapcsolattal. Egy *C* cselekvés ütközik az $A \xrightarrow{p} B$ -vel, ha *C*

Created by XMLmind XSL-FO Converter.

következménye a $\neg p$, és ha C az A után és B előtt jöhet (a rendezési kényszereknek megfelelően). Néhány szerző az okozati kapcsolatokat **védett tartomány**oknak (**protection interval**s) nevezi, mert az $A \xrightarrow{p} B$ megvédi p-t a negálástól az A–B intervallumban.

- **Nyitott előfeltétel**ek (**open precondition**s). Egy előfeltétel nyitott, ha nem teljesül a terv egy akciójának hatására. A tervkészítők feladata, hogy a nyitott előfeltételek halmazát üres halmazra csökkentse, ellentmondások bevezetése nélkül.

Például a 11.6. ábrán bemutatott teljes terv a következő komponenseket tartalmazza (nem tüntettük fel a rendezési kényszereket, amelyek minden cselekvést az *Indít* után és a *Befejez* elé helyeznek):

```
Cselekvések:{JobbZokni, JobbCipő, BalZokni, BalCipő, Indít, Befejez}

Rendezések: {JobbZokni ≺ JobbCipő, BalZokni ≺ BalCipő}
```

$$\text{Kapcsolatok:}\{JobbZokni \xrightarrow{JobbZokniFenn} JobbCipő, BalZokni \xrightarrow{BalZokniFenn} BalCipő,$$
$$JobbCipő \xrightarrow{JobbCipőFenn} Befejez, BalCipő \xrightarrow{BalCipőFenn} Befejez\}$$

```
Nyitott előfeltételek: {}
```

Fontos

Konzisztens tervnek (**consistent plan**) nevezzük azt a tervet, amelyben nincsenek ciklusok a rendezési megkötésekben és nincsenek ellentmondások az okozati kapcsolatokkal. Egy konzisztens terv, mely nyitott előfeltételektől mentes, a **megoldás** (**solution**). Rövid gondolkodással belátható: *egy részben rendezett megoldás minden sorba rendezése egy teljesen rendezett megoldás*, *melynek végrehajtása az induló állapotból egy célállapotra vezet.* Ez annyit tesz, hogy a „terv végrehajtását” a teljesen rendezett tervekről kiterjeszthetjük a részben rendezett tervekre. Egy részben rendezett terv végrehajtása során ismétlődően mindig választunk egy következő cselekvést, a lehetséges következő cselekvések *bármelyikét*. A 12. fejezetben látni fogjuk, hogy az ágens számára a terv végrehajtásának rugalmassága nagyon hasznos lehet, ha a környezet nem az elvártaknak megfelelően működik. A rugalmas sorrendezés egyszerűsíti a kisebb tervek összekombinálását nagyobbakká, mert minden részterv átrendezheti a cselekvéseinek sorrendjét, hogy elkerülje a más tervekkel való ütközést.

Most már készen állunk arra, hogy formalizáljuk azt a keresési problémát, amit az RRT megold. Egy olyan formalizálással kezdünk, ami az ítéletlogikai tervkészítési problémákra alkalmas, az elsőrendű logikai megoldással járó komplikációkat későbbre hagyjuk. Mint rendszerint, a definíció tartalmazza a kiinduló állapotot, a cselekvéseket és a céltesztet.

- A kiindulási terv egy *Indít* és egy *Befejez* cselekvést, egy *Indít* $\prec$ *Befejez* rendezést tartalmaz, és nincsenek okozati kapcsolatok. A *Befejez* minden előfeltétele nyitott.

- Az állapotátmenet-függvény önkényesen kiválasztja B egy p nyitott előfeltételét és legenerál egy következő tervet egy A cselekvés minden olyan lehetséges konzisztens megválasztásához, mely teljesíti p-t. A konzisztenciát a következőképp biztosítjuk:

  1. Az $A \xrightarrow{p} B$ okozati kapcsolatot és az $A \prec B$ rendezési megkötést hozzáadjuk a tervhez. Az A cselekvés lehet a terv egy már létező vagy egy új cselekvése. Amennyiben új, hozzáadjuk a tervhez az *Indít* $\prec A$ és az $A \prec$ *Befejez* megkötésekkel.

  2. Feloldunk minden konfliktust, az új okozati kapcsolat és az összes létező cselekvés, valamint az A cselekvés (ha új) és minden létező okozati kapcsolat között. Egy az $A \xrightarrow{p} B$ és C között fennálló konfliktust azáltal oldunk fel, hogy C-t valamikor kívül helyezzük a védett intervallumon a $B \prec C$ vagy a $C \prec A$ hozzáadásával. Amennyiben konzisztens tervet adnak, beszúrjuk a követő állapotokat, valamelyikhez vagy akár mindkettőhöz.

- A célteszt ellenőrzi, hogy a terv megoldása-e az eredeti tervkészítési problémának. Mivel csak konzisztens terveket állítunk elő, a céltesztnek csak azt kell ellenőrizni, hogy vannak-e nem nyitott előfeltételek.

Created by XMLmind XSL-FO Converter.

Emlékezzünk, hogy keresési algoritmus által vizsgált cselekvések ebben a felírásban sokkal inkább tervfinomító lépések, mint valós cselekvések a feladattérből. Az út költsége ezért szigorú értelembe véve irreleváns, hiszen az egyetlen ami fontos, az azon terv valós cselekvéseinek költsége, amire ez az út vezetett. Mindazonáltal *lehetséges* egy útvonalköltség-függvény specifikálása, ami a valós terv költségeit tükrözi: minden valós cselekvésnek a tervhez való hozzáadását egységnyi költséggel, míg a finomító lépéseket 0 költséggel vesszük figyelembe. Így $g(n)$, ahol n a terv, egyenlő lesz a tervben található valós cselekvések számával. Egy heurisztikus $h(n)$ becslés szintén alkalmazható.

Első ránézésre azt gondolhatjuk, hogy az állapotátmenet-függvénynek *minden* nyitott *p*-hez tartalmaznia kellene egy következő állapotot, nem csak egyhez. Ez azonban, redundáns lenne és nem lenne hatékony, ugyanazon okból, amiért a kényszerkielégítési algoritmusok nem szúrnak be minden lehetséges változóhoz utódot: a nyitott előfeltételek kezelési sorrendje (csakúgy, mint a kényszerkielégítési probléma változóinak kezelési sorrendje) kommutatív. (Lásd 2. szakasz - A visszalépéses keresés alkalmazása kényszerkielégítési problémákra részben.) Így tetszőleges sorrendet választhatunk, és az algoritmusaink még mindig teljesek. A helyes rendezés megválasztása gyorsabb kereséshez vezethet, de minden sorrendezés ugyanazokra a megoldásjelöltekre vezet.

3.1. Példa a részben rendezett tervkészítésre

Most nézzük meg, hogy az RRT hogyan oldja meg a 11.1. alfejezetben szereplő pótkerék problémát. A probléma a leírását a 11.7. ábrán megismételjük.

11.7. ábra - Az egyszerű kerékcsere probléma

Kiindulás(*Ott*(*LaposKerék*, *Tengely*) ∧ *Ott*(*Pótkerék*, *Csomagtartó*)
Cél(*Ott*(*Pótkerék*, *Tengely*))
Cselekvés(*Eltávolít*(*Pótkerék*, *Csomagtartó*),
ELŐFELTÉTEL: *Ott*(*Pótkerék*, *Csomagtartó*)
KÖVETKEZMÉNY: ¬ *Ott*(*Pótkerék*, *Csomagtartó*) ∧ *Ott*(*Pótkerék*, *Föld*))
Cselekvés(*Eltávolít*(*LaposKerék*, *Tengely*),
ELŐFELTÉTEL: *Ott*(*LaposKerék*, *Tengely*)
KÖVETKEZMÉNY: ¬ *Ott*(*LaposKerék*, *Tengely*) ∧ *Ott*(*LaposKerék*, *Föld*))
Cselekvés(*Felszerel*(*Pótkerék*, *Tengely*),
ELŐFELTÉTEL: *Ott*(*Pótkerék*, *Föld*) ∧ ¬ *Ott*(*LaposKerék*, *Tengely*)
KÖVETKEZMÉNY: ¬ *Ott*(*Pótkerék*, *Föld*) ∧ *Ott*(*Pótkerék*, *Tengely*))
Cselekvés(*OtthagyÉjszakára*,
ELŐFELTÉTEL:
KÖVETKEZMÉNY: ¬ *Ott*(*Pótkerék*, *Föld*) ∧ ¬ *Ott*(*Pótkerék*, *Tengely*) ∧ ¬ *Ott*(*Pótkerék*, *Csomagtartó*)
∧ ¬ *Ott*(*LaposKerék*, *Föld*) ∧ ¬ *Ott*(*Laposkerék*, *Tengely*))

A megoldás keresése egy kezdeti tervből indul, ami egy *Indít* cselekvést tartalmaz az *Ott*(*Pótkerék*, *Csomagtartó*) ∧ *Ott*(*LaposKerék*, *Tengely*) következménnyel és egy *Befejez* cselekvést az *Ott*(*Pótkerék*, *Tengely*) előfeltétellel. Ezután követő állapotokat állítunk elő úgy, hogy egy nyitott előfeltételt választunk a munkához (visszavonhatatlanul), majd választunk az ehhez vezető lehetséges cselekvések közül. Most nem foglalkozunk a heurisztikákkal, amik segíthetik ezt a döntést; látszólag véletlenszerű döntést hozunk. Az események sorrendje a következő:

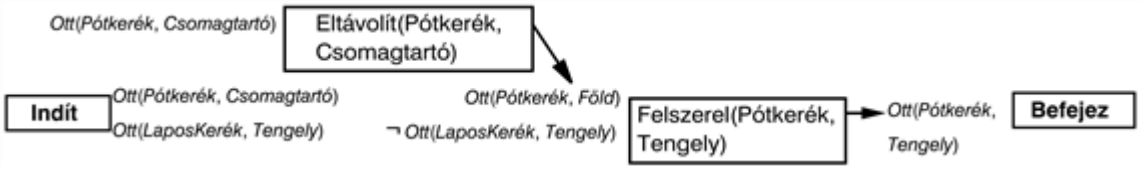
1. Vegyük az egyetlen nyitott előfeltételt, a *Befejez* cselekvés *Ott*(*Pótkerék*, *Tengely*) feltételét. Válasszuk ki az egyetlen alkalmazható cselekvést, a *Felszerel*(*Pótkerék*, *Tengely*)-t.

2. Vegyük a *Felszerel*(*Pótkerék*, *Tengely*) cselekvés *Ott*(*Pótkerék*, *Föld*) előfeltételét. Válasszuk az *Eltávolít*(*Pótkerék*, *Csomagtartó*) cselekvést, ami az egyetlen alkalmazható cselekvés az előfeltétel biztosításához. A végső terv a 11.8. ábrán látható.

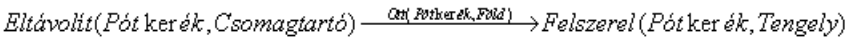
11.8. ábra - A nem teljes részben rendezett terv a kerékcsere problémához, miután megválasztottuk a cselekvéseket az első két nyitott előfeltételhez. A dobozok cselekvéseket testesítenek meg, a bal oldalon az előfeltételekkel, a jobb oldalon pedig a

Created by XMLmind XSL-FO Converter.

következményekkel. (Az *Indít* cselekvést kivéve a következményeket elhagytuk.) A fekete nyilak az okozati kapcsolatokat mutatják, melyek a nyíl fejénél szereplő állítást védik.



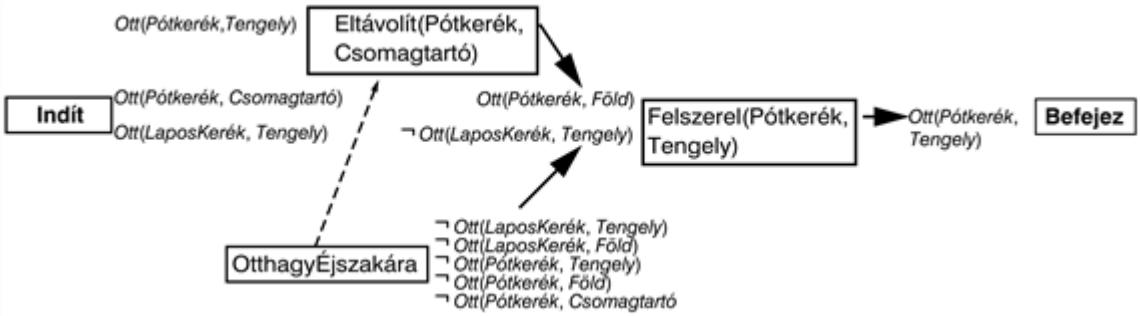
3. Vegyük a *Felszerel*(*Pótkerék*, *Tengely*) cselekvés ¬*Ott*(*LaposKerék*, *Tengely*) előfeltételét. Hogy ellenkezzünk egy kicsit, válasszuk az *OtthagyÉjszakára* cselekvést, a kézenfekvő *Eltávolít*(*LaposKerék*, *Tengely*) helyett. Vegyük észre, hogy az *OtthagyÉjszakára* cselekvésnek szintén következménye a ¬*Ott*(*Pótkerék*, *Föld*), ami ütközik az



okozati kapcsolattal. Hogy feloldjuk ezt az ütközést, egy rendezési megkötést illesztünk be, ami az *OtthagyÉjszakára* cselekvést az *Eltávolít*(*Pótkerék*, *Csomagtartó*) elé rendeli. Az így előálló terv a 11.9. ábrán látható. (Miért oldja ez fel az ütközést és miért nincs erre más lehetőség?)

4. Most az egyetlen fennmaradó nyitott előfeltétel az *Ott*(*Pótkerék*, *Csomagtartó*), ami az *Eltávolít*(*Pótkerék*, *Csomagtartó*) cselekvés előfeltétele. Az egyetlen cselekvés, ami ezt elérheti, a meglévő *Indít* cselekvés, de az *Indít* és az *Eltávolít*(*Pótkerék*, *Csomagtartó*) közötti okozati kapcsolat ütközik az *OtthagyÉjszakára* cselekvés ¬*Ott*(*Pótkerék*, *Csomagtartó*) következményével. Most nincs mód arra, hogy feloldjuk az ütközést az *OtthagyÉjszakára* cselekvéssel: nem sorolhatjuk az *Indít* elé (mert semmi nem kerülhet az *Indít* elé), és nem rendelhető az *Eltávolít*(*Pótkerék*, *Csomagtartó*) után (mert már van egy rendezés, ami elé helyezi). Ezért muszáj visszalépnünk, törölnünk az *Eltávolít*(*Pótkerék*, *Csomagtartó*) cselekvést és a két utolsó okozati kapcsolatot, és visszalépnünk a 11.8. ábrán bemutatott állapothoz.

11.9. ábra - A terv alakulása, az *OtthagyÉjszakára* választása után, amit a ¬Ott(*LaposKerék*, *Tengely*) eléréséhez választottunk. Hogy elkerüljük az ütközést az *Ott*(*Pótkerék*, *Csomagtartó*)-t védő, az *Eltávolít*(*Pótkerék*, *Csomagtartó*)-ból induló okozati kapcsolattal, az *OtthagyÉjszakára* cselekvést egy rendezési megkötéssel az *Eltávolít*(*Pótkerék*, *Csomagtartó*) elé helyeztük, amit a szaggatott nyíllal jelöltünk.



Összefoglalva, a tervkészítő bebizonyította, hogy az *OtthagyÉjszakára* nem használható fel kerékcserére.

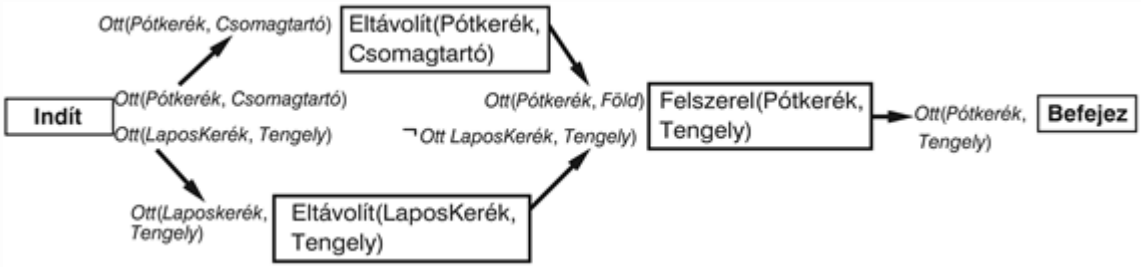
5. Vegyük újra a *Felszerel*(*Pótkerék*, *Tengely*) cselekvés ¬*Ott*(*LaposKerék*, *Tengely*) előfeltételét. Ez alkalommal válasszuk az *Eltávolít*(*LaposKerék*, *Tengely*) cselekvést.

6. Vegyük újra az *Eltávolít*(*Pótkerék*, *Csomagtartó*) cselekvés *Ott*(*Pótkerék*, *Csomagtartó*) előfeltételét, és válasszuk az *Indít* cselekvést, hogy ezt elérjük. Ez alkalommal nincsenek ütközések.

7. Vegyük az *Eltávolít*(*LaposKerék*, *Tengely* cselekvés *Ott*(*LaposKerék*, *Tengely*) előfeltételét, valamint ennek eléréséhez az *Indít* cselekvést. Ez egy teljes és konzisztens tervet ad, vagy más szavakkal egy megoldást, amint azt a 11.10. ábra mutatja.

Created by XMLmind XSL-FO Converter.

11.10. ábra - A kerékcsere probléma végső megoldása. Vegyük észre, hogy az *Eltávolít*(*Pótkerék, Csomagtartó*) és az *Eltávolít*(*LaposKerék, Tengely*) tetszőleges sorrendben végrehajtható a *Felszerel*(*Pótkerék, Tengely*) előtt.



Bár ez a példa nagyon egyszerű, jól szemlélteti a részben rendezett tervkészítés erősségét. Először is az okozati kapcsolatok a keresési tér (fa) egy korai szűkítéséhez metszéséhez vezettek, mert kizártak olyan területeket, amelyek a feloldhatatlan ütközések miatt nem tartalmaznak megoldásokat. Másodszor a 11.10. ábra megoldása egy részben rendezett terv. Ebben az esetben ennek előnye nem nagy, mert csak két lehetséges sorba rendezés létezik, mégis egy ágens számára előnyös lehet ez a rugalmasság, például ha a kereket sűrű forgalomban kellene lecserélni.

A példa szintén rámutat néhány javítási lehetőségre. Például egy próbálkozás duplán történt: az *Indít* az *Eltávolít*(*Pótkerék, Csomagtartó*) cselekvéshez kapcsolódik még mielőtt az ütközés visszalépést váltana ki, és a visszalépés hatására a kapcsolat felbomlik, bár nem vesz részt az ütközésben. A keresés folytatásakor a kapcsolat újra létrejön. Ez tipikus az időrendi visszalépés esetén és elkerülhető, ha függőségvezérelt visszalépést használunk.

3.2. Részben rendezett tervkészítés kötetlen változókkal

Ebben a fejezetben a változókat tartalmazó elsőrendű cselekvés reprezentációra alkalmazott részben rendezett tervkészítés során felmerülő komplikációkkal foglalkozunk. Tegyük fel, hogy a kockavilág (lásd 11.4. ábra) problémát kell megoldani a *Rajta*(*A, B*) nyitott előfeltétellel és a

```
Cselekvés(Mozgat(b, x, y),

Előfeltétel: Rajta(b, x) ∧ Üres(x) ∧ Üres(y)

Következmény: Rajta(b, y) ∧ Üres(x) ∧ ¬Rajta(b, x) ∧ ¬Üres(y))
```

cselekvéssel. Ez a cselekvés teljesíti a *Rajta*(*A, B*) literált, mert a *Rajta*(*b, u*) következmény egyesíthető a *Rajta*(*A, B*)-vel a {*b*/*A*, *y*/*B*} behelyettesítéssel. Alkalmazzuk tehát ezt a behelyettesítést a cselekvésre, aminek az eredménye a

```
Cselekvés(Mozgat(A, x, B),

Előfeltétel: Rajta(A, x) ∧ Üres(x) ∧ Üres(B)

Következmény: Rajta(A, B) ∧ Üres(x) ∧ ¬Rajta(b, x) ∧ ¬Üres(B))
```

Ez az *x* változót kötetlenül hagyja. Ez annyit tesz, hogy a cselekvés annyit mond, hogy mozgasd *A*-t *valahonnan*, de azt nem, hogy honnét. Ez egy másik példa a legkisebb megkötés elvre: elhalaszthatjuk a döntéseket egészen addig, amíg a terv egy másik lépése ezt meghozza nekünk. Tegyük fel például, hogy a *Rajta*(*A, D*) a kiinduló állapotunk. Az *x*-et *D*-vel behelyettesítve az *Indít* cselekvés használható fel, hogy elérjük a *Rajta*(*A, x*)-et. A módszer, hogy több információt kivárunk, mielőtt megválasztjuk *x*-et, gyakran sokkal hatékonyabb, mint az *x* minden értékének kipróbálása és a visszalépés, ha ez sikertelen.

A változók jelenléte az előfeltételekben és a cselekvésekben nehezíti az ütközések detektálását és feloldását. Például amikor a *Mozgat*(*A, x, B*) a terv része lesz, a

$$Mozgat(A, x, B) \xrightarrow{Rajta(A,B)} Befejez$$

okozati kapcsolatra is szükség van. Ha van egy M_2 cselekvés ¬*Rajta*(*A, z*) következménnyel, akkor ez csak akkor okoz ütközést, ha a *z* értéke *B*. Hogy kezeljük ezt a lehetőséget, kiterjesztjük a tervek reprezentációját,

Created by XMLmind XSL-FO Converter.

hogy tartalmazzanak $z \neq X$ alakú **egyenlőtlenségi kényszer**eket (**inequality constraint**s), ahol z egy változó és X egy változó vagy egy konstansszimbólum. Ebben az esetben az ütközést a $z \neq B$ hozzáadásával oldhatjuk fel, ami azt jelenti hogy a terv később B kivételével bármire behelyettesítheti z-t. Bármikor amikor egy behelyettesítést hajtunk végre a tervben, ellenőriznünk kell, hogy az nem ellenkezik-e az egyenlőtlenségi korlátokkal. Például egy x/y behelyettesítés ütközik az $x \neq y$ kényszerrel. Az ilyen ellentétek nem oldhatók fel, ezért a tervkészítő visszalépésre kényszerül.

A 12.6. alfejezetben egy bővebb példát adunk a részben rendezett tervkészítésre, ami a változókat tartalmazó kockavilág probléma.

3.3. Heurisztikák a részben rendezett tervkészítésre

A teljesen rendezett tervkészítéssel összehasonlítva a részben rendezett tervkészítés egyértelmű előnye, hogy a probléma részfeladatokra bontható. A hátránya, hogy nem reprezentálja megfelelően az állapotokat, így nehezebb megbecsülni, hogy a részben rendezett terv milyen messze van a céltól. Jelenleg jóval kevesebb az ismeretanyag arról, hogy hogyan adható pontos heurisztika a részben rendezett tervkészítéshez, mint a teljesen rendezett tervkészítések esetén.

A legkézenfekvőbb heurisztika, hogy megszámoljuk az eltérő nyitott előfeltételeket. Ez javítható azáltal, ha kivonjuk azon nyitott előfeltételek számát, melyek illeszkednek az *Indít* állapot egy literáljaival. Hasonlóan a teljesen rendezett esethez ez túlbecsüli a költséget, ha vannak cselekvések, melyek több célt érnek el, és alulbecsül, ha negatív kölcsönhatások vannak a terv lépései között. A következő alfejezet egy olyan megközelítést mutat be, ami lehetővé teszi, hogy sokkal pontosabb heurisztikákat kapjunk egy relaxált problémából.

A heurisztikus függvényt arra használjuk, hogy kiválasszuk a tovább finomítandó tervet. Ezzel a választással az algoritmus követő állapotokat generál egyetlen kiválasztott nyitott előfeltételből kiindulva. Mint a változókiválasztás a kényszerkielégítési algoritmusban, ez a választás nagy hatással van a hatékonyságra. A kényszerkielégítési probléma **legjobban-korlátozott-változó** (**most-constrained-variable**) heurisztikája alkalmazható a tervkészítő algoritmusokra, és látszólag jól működik. Az alapötlet, hogy válasszuk azt a nyitott feltételt, mely a *lehető legkevesebb* módon elégíthető ki. Ennek a heurisztikának két speciális esete van. Először, ha egy nyitott feltétel nem érhető el egyetlen cselekvéssel sem, akkor a heurisztika kiválasztja. Ez jó ötlet, hiszen a teljesíthetetlenség korai érzékelése sok munkát megspórolhat. Másodszor, ha egy nyitott feltétel csak egyféleképp teljesíthető, akkor célszerű kiválasztani, hiszen ez egy elkerülhetetlen döntés, ami további megkötéseket eredményezhet a későbbi választásokhoz. Habár minden nyitott feltételhez meglehetősen költséges kiszámolni, hogy azok hányféleképpen teljesíthetők, és ez nem is mindig éri meg, a kísérletek azt mutatják, hogy a két speciális eset kezelése jelentős gyorsulást eredményezhet.

4. Tervkészítési gráfok

Az összes javasolt heurisztika a teljesen rendezett és a részben rendezett tervkészítésre pontatlanságokkal terhelt. Ez a fejezet bemutatja, hogy egy speciális adatszerkezet a **tervkészítési gráf** (**planning graph**) felhasználható, hogy jobb heurisztikus becsléseket nyerjünk. Ezek a heurisztikák bármely eddig tárgyalt keresési technikával használhatók. Egy másik lehetőség, hogy a megoldást közvetlenül a tervkészítési gráfból nyerjük ki egy erre kiélezett algoritmussal, mint amilyen például a `GRAPHPLAN`.

A tervkészítési gráf a terv időrendi lépéseinek megfelelő szintekből áll, ahol a 0-dik szint a kiinduló állapot. Minden szint egy literálhalmazt és egy cselekvéshalmazt tartalmaz. A literálok durván azok, melyek igazak *lehetnek* az adott lépésnél, az addig végrehajtott cselekvések függvényében. Ismét durván fogalmazva, a cselekvések azok, melyeknek az előfeltételei teljesülhetnek az adott időlépésben attól függően, hogy aktuálisan mely literálok teljesülnek. Azért mondjuk, hogy „durván”, mert a tervkészítési gráf csak a cselekvések közötti lehetséges negatív kölcsönhatások egy korlátozott részhalmazát tartalmazza, ezért a tervkészítési gráf optimista lehet a literál teljesítéséhez szükséges minimális lépések számával kapcsolatban. Mindazonáltal a tervkészítési gráfban ez a lépésszám egy jó becslés arra, hogy milyen nehéz egy literált teljesíteni a kiindulási állapotból. Még fontosabb, hogy a tervkészítési gráfot úgy definiáltuk, hogy könnyen elkészíthető legyen.

A tervkészítési gráf csak ítéletlogikai tervkészítési problémákra alkalmazható – olyanokra, melyek nem tartalmaznak változókat. Ahogy a 11.1. alfejezetben említettük, mind a `STRIPS`, mind az ADL reprezentáció ítéletlogikára alakítható. A nagyszámú entitást tartalmazó problémákat ez jelentősen felduzzaszthatja a

Created by XMLmind XSL-FO Converter.

cselekvéssémák számában. Ennek ellenére a tervkészítési gráfok hatékony eszköznek bizonyultak a nehéz tervkészítési problémák megoldásában.

A tervkészítési gráfokat egy egyszerű példán mutatjuk be. (Az összetettebb feladatok olyan gráfhoz vezetnek, amelyek nem férnének ki egy oldalra.) A 11.11. ábra a feladatot, míg a 11.12. ábra az ehhez tartozó gráfot mutatja. Az S_0 állapotszintről indulunk, ami a probléma kiinduló állapotának felel meg. Ezt az A_0 cselekvésszinttel folytatjuk, amibe azokat a cselekvéseket helyezzük, melyek előfeltételei az előző szinten teljesülnek. Minden cselekvés össze van kötve az S_0-ban található előfeltételeivel, illetve az S_1-ben található következményeivel, ami ebben az esetben az S_0-ban nem szereplő literálok bevezetését jelenti S_1-ben.

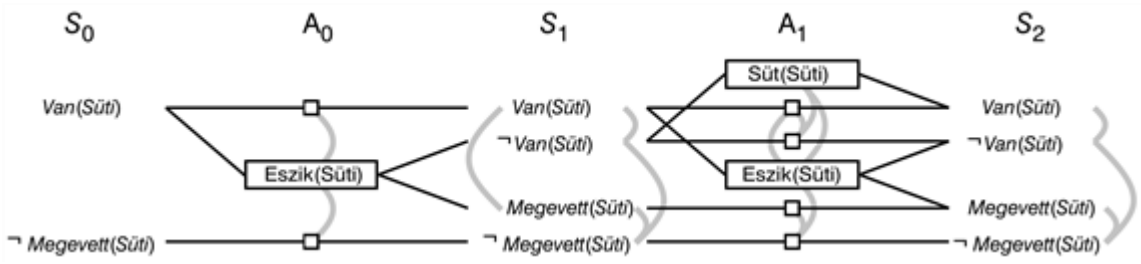
11.11. ábra - A „legyen süti és együnk is” probléma

Kiindulás(*Van*(*Süti*))
Cél(*Van*(*Süti*) ∧ *Megevett*(*Süti*))
Cselekvés(*Eszik*(*Süti*)
ELŐFELTÉTEL: *Van*(*Süti*)
KÖVETKEZMÉNY: ¬ *Van*(*Süti*) ∧ *Megevett*(*Süti*))
Cselekvés(*Süt*(*Süti*)
ELŐFELTÉTEL: ¬ *Van*(*Süti*)
KÖVETKEZMÉNY: *Van*(*Süti*))

A tervkészítési gráfnak a nem cselekvést ugyanúgy kell reprezentálni, mint a cselekvéseket. Ez annyit tesz, hogy a szituációkalkulus keretaxiómáinak megfelelő működésre van szükség, ami alapján egy literál két állapon keresztül igaz maradhat, ha nincs ezt módosító cselekvés. Egy tervkészítési gráfban ezt **megőrző cselekvés**ekkel (**persistence action**s) oldjuk meg. Minden pozitív és negatív *C* literálhoz egy megőrző cselekvést szúrunk be *C* előfeltétellel és *C* következménnyel. A 11.12. ábra az A_0 szinten egyetlen „valós” cselekvést az *Eszik*(*Süti*)-t tartalmazza, két megőrző cselekvéssel, amit kicsi négyzetek jelölnek.

Az A_0 tartalmazza az összes cselekvést, ami az S_0 állapotban előfordulhat, de ugyanilyen fontos, hogy rögzíti a cselekvések közötti ütközéseket is, amelyek megakadályozzák, hogy egyszerre történjenek. A 11.12. ábra szürke vonalai ezeket a **kölcsönös kizárás**i (**mutual exclusion**s vagy **mutex**) kapcsolatokat jelölik. Például az *Eszik*(*Süti*) kölcsönös kizárásban van a *Van*(*Süti*) vagy a ¬*Megevett*(*Süti*) megőrzésével. Hamarosan látjuk, hogy a mutex kapcsolatokat hogyan számíthatjuk.

11.12. ábra - A „legyen süti és együnk is” probléma tervkészítési gráfja az S_2 szintig. A téglalapok az akciókat jelentik (a kicsi négyzetek a megőrző cselekvések), az egyenesek az előfeltételeket és a következményeket jelölik. A kölcsönös kizárásokat ívelt szürke vonalak jelölik.



Az S_1 az összes olyan literált tartalmazza, ami az A_0 szint akcióinak bármely részhalmazát választva elérhető. Szintén tartalmaz mutex kapcsolatokat (szürke vonalakat) jelölve az olyan literálokat, melyek nem teljesülhetnek egyszerre, a választott cselekvésektől függetlenül. Például a *Van*(*Süti*) és a *Megevett*(*Süti*) kölcsönösen kizárják egymást. Az A_0 kiválasztott cselekvéseitől függően az eredmény vagy az egyik vagy a másik lehet, de a kettő egyszerre nem. Más szavakkal az S_1, csakúgy, mint a regressziós állapottér-keresés, több állapotot reprezentál, a kizárási kapcsolatok pedig kényszerek, melyek a lehetséges állapotok halmazát definiálják.

Created by XMLmind XSL-FO Converter.

És ez így megy tovább. Az S_i állapot- és az A_i cselekvésszintek között mozgunk, egészen addig, míg elérünk egy szintet, ahol két egymást követő szint azonos. Ekkor azt mondjuk, hogy a gráf **kiegyenlítődött** (**leveled off**). Minden egymást követő szint azonos, így további kiterjesztésre nincsen szükség.

Egy olyan struktúrát kaptunk, ahol az A_i szint tartalmaz minden S_i-ben alkalmazható cselekvést, a kényszerekkel együtt, melyek megmondják, hogy melyik cselekvéspár nem hajtható végre egyidejűleg. Minden S_i szint tartalmazza az összes literált, ami az A_{i-1} összes lehetséges cselekvéshalmazának hatására teljesül, a kényszerekkel együtt, amelyek a nem lehetséges literálpárokat jelölik ki. Fontos megjegyezni, hogy a tervkészítési gráf elkészítési folyamatához *nem* kell választanunk a cselekvések között, ami kombinatorikus keresést eredményezne. Ezzel szemben, a tervkészítési gráf konstrukciója csak a lehetetlen választásokat rögzíti, mutex kapcsolatok felhasználásával. Egy ilyen tervkészítési gráf elkészítésének bonyolultsága egy alacsonyrendű polinomiális komplexitás a cselekvések számát tekintve, ahol az állapottér exponenciális a literálok számában.

Most mutex kapcsolatokat definiálunk, mind a cselekvések, mind a literálok számára. Egy adott szinten, a mutex kapcsolat akkor érvényes két *cselekvés* között, ha a következő három feltétel valamelyike fennáll:

- *Inkonzisztens hatások:* egy cselekvés negálja egy másik következményét. Például az *Eszik*(*Süti*) és a *Van*(*Süti*) megőrzése inkonzisztens hatású, mert a *Van*(*Süti*) következmény tekintetében ellentétesek.
- *Interferencia:* egy cselekvés egyik következménye a negálása egy másik cselekvés előfeltételének. Például az *Eszik*(*Süti*) interferál a *Van*(*Süti*) megőrzésével, mert negálja az előfeltételét.
- *Versenyhelyzet:* egy cselekvés előfeltétele kölcsönösen kizáró egy másik előfeltételével. Például a *Süt*(*Süti*) és az *Eszik*(*Süti*) mutexek, mert versenyeznek a *Van*(*Süti*) előfeltétel tekintetében.

Két azonos szineten lévő *literál* között mutex kapcsolat áll fenn, ha az egyik a másik negáltja, vagy bármely lehetséges cselekvéspár, amely a két literált elérheti, egymást kölcsönösen kizáró. Ezt az állapotot *inkonzisztens háttérnek* nevezzük. Például a *Van*(*Süti*) és a *Megevett*(*Süti*) kizárják egymást S_1-ben, mert az egyetlen mód a *Van*(*Süti*) teljesítésére egy megőrző cselekvés, ami mutex a *Megevett*(*Süti*) egyetlen elérési lehetőségével, nevezetesen az *Eszik*(*Süti*) cselekvéssel. Az S_2 szinten a két literál már nem mutex, mert új elérési lehetőségek vannak hozzájuk, mint a *Süt*(*Süti*) és a *Megevett*(*Süti*) megőrző cselekvése, melyek nem zárják ki egymást.

4.1. Tervkészítési gráfok heurisztikus becslésekre

Fontos

A tervkészítési gráf, ha egyszer már létrehoztuk, nagyon gazdag információforrás a problémáról. Például *egy literál, ami nem található meg az utolsó szinten, egyetlen tervvel sem érhető el.* Ez a megfigyelés a visszafelé keresésben használható a következőképp: bármely elérhetetlen literált tartalmazó állapot költsége $h(n) = \infty$. Hasonlóan a részben rendezett tervkészítésnél bármely tervnek a költsége egy elérhetetlen nyitott előfeltétellel, $h(n) = \infty$.

Ez az ötlet tovább általánosítható. Minden cél literál elérési költségét becsülhetjük azzal a szinttel, ahol először megjelenik a tervkészítési gráfban. Ezt a cél **szint költségé**nek (**level cost**) nevezzük. A 11.12. ábrán a *Van*(*Süti*) szint költsége 0, a *Megevett*(*Süti*) szint költsége 1. Könnyű megmutatni (lásd 11.9. feladat), hogy ezek a közelítések elfogadhatók az egyedülálló célokra. A becslés azonban lehet, hogy nem túl jó, mert a tervkészítési gráfok több cselekvést megengednek szintenként, míg a heurisztika csak a szintek számát tartalmazza, és nem a cselekvések számát. Ebből kifolyólag a heurisztikák számításában, gyakori a **soros tervkészítési gráf** (**serial planning graph**) használata. A soros gráf szerint egy adott időpillanatban csak egyetlen cselekvés hajtható végre, ami mutexekkel érhető el úgy, hogy a megőrző cselekvések kivételével, minden cselekvés közé mutex kapcsolatot teszünk. A soros tervkészítő gráfból kinyert szintköltségek gyakran egész elfogadható közelítését adják a valós költségeknek.

Három egyszerű megközelítés létezik, hogy célliterálok konjunkcióihoz költséget becsüljünk. A **maximális szint** (**max-level**) heurisztika egyszerűen a célok közüli maximális szintköltségét veszi, ami elfogadható, de nem feltétlen nagyon pontos. A **szintösszeg** (**level sum**) heurisztika a részcél függetlenségi feltételezésből a célok szintköltségeinek összegét adja. Ez nem elfogadható, de nagyon jól működik olyan gyakorlati problémákra, amelyek nagymértékben részekre bonthatók. Ez sokkal pontosabb, mint a 11.2. alfejezetben bemutatott nem-kielégített-célok-heurisztika. A mi feladatunkban a *Van*(*Süti*) $\wedge$ *Megevett*(*Süti*) konjunktív célhoz rendelhető heurisztikus becslés 0 + 1 = 1, míg a helyes válasz 2. Mindezeken túl, ha eltávolítjuk a *Süt*(*Süti*) cselekvést, a

Created by XMLmind XSL-FO Converter.

becslés még mindig 1, de a konjunktív cél ekkor már elérhetetlen. Végezetül, a **halmazszint (set-level)** heurisztika azt a szintet keresi meg, ahol a konjunktív cél összes literálja megjelenik a tervkészítési gráfban, és nincsenek közöttük kölcsönösen kizáró párok. A heurisztika a korrekt 2 értéket adja az eredeti feladatra és végtelent a *Süt*(*Süti*) nélküli feladatra. Ez előnyben van a maximális szint heurisztikával szemben, és kimagaslóan jól működik azon feladatokban, ahol sok kölcsönhatás van a résztervek között.

Lévén egy pontos heurisztika készítésére szolgáló eszköz, a tervkészítési gráf tekinthető úgy, mint egy relaxált probléma, ami hatékonyan megoldható. Hogy a relaxált probléma természetét megértsük, pontosan meg kell értenünk, hogy mit jelent, ha egy *g* literál megjelenik a tervkészítési gráf S_i szintjén. Ideális esetben garanciát szeretnénk arra, hogy létezik egy terv *i* cselekvés szinttel, ami eléri *g*-t, illetve ha *g* nem jelenik meg, akkor nincs is ilyen terv. Sajnos ezt garantálni majdnem ugyanolyan nehéz, mint megoldani az eredeti tervkészítési problémát. A tervkészítési gráf a garancia második részéről gondoskodik (ha *g* nem jelenik meg, akkor nincs terv), de ha *g* megjelenik, akkor a tervkészítési gráf csak annyit ígér, hogy van egy terv, ami *várhatóan* eléri *g*-t, nincsenek „nyilvánvaló" hibák. Egy nyilvánvaló hiba definíció szerint egy olyan hiba, ami úgy detektálható, hogy két cselekvést vagy két literált tekintünk egyszerre, vagy más szavakkal a mutex relációkat vizsgáljuk. Lehetnek összetettebb hibák, melyek három, négy vagy több cselekvést tartalmaznak, de a tapasztalatok azt mutatják, hogy nem éri meg ezekkel foglalkozni. Ez hasonló a kényszerkielégítési problémáknál tanultakkal, hogy a megoldás megkeresése előtt gyakran megéri a 2-konzisztencia számítás, de a 3- vagy a magasabb konzisztencia kiszámítása ritkábban kifizetődő (lásd 5.2. alfejezet).

4.2. A Graphplan algoritmus

Ez a fejezet bemutatja, hogy hogyan nyerhető ki közvetlenül a terv a tervkészítési gráfból, ahelyett hogy azt csak a heurisztikákhoz használjuk. A GRAPHPLAN algoritmusnak (11.13. ábra) két fő lépése van, melyek ciklikusan váltakoznak. Először ellenőrzi, hogy minden célliterál jelen van-e az adott szinten anélkül, hogy mutex kapcsolatok lennének közöttük. Ebben az esetben, az aktuális gráfban *létezhet* megoldás, így az algoritmus megpróbálja azt kinyerni. Ellenkező esetben bővíti a gráfot úgy, hogy az aktuális szinthez a cselekvéseket, a következőhöz az állapotliterálokat adja: ez a folyamat folytatódik addig, amíg megtaláljuk a megoldást, vagy bebizonyosodik, hogy nem létezik megoldás.

11.13. ábra - A GRAPHPLAN algoritmus. A GRAPHPLAN algoritmus egy megoldás kinyerő és egy gráf-bővítő lépés között alternál. A MEGOLDÁS-KINYERÉS a befejezéstől visszafelé keresve ellenőrzi, hogy található-e megoldás. A GRÁF-BŐVÍTÉS adja a cselekvéseket az adott szinthez és az állapotliterálokat a következőhöz.

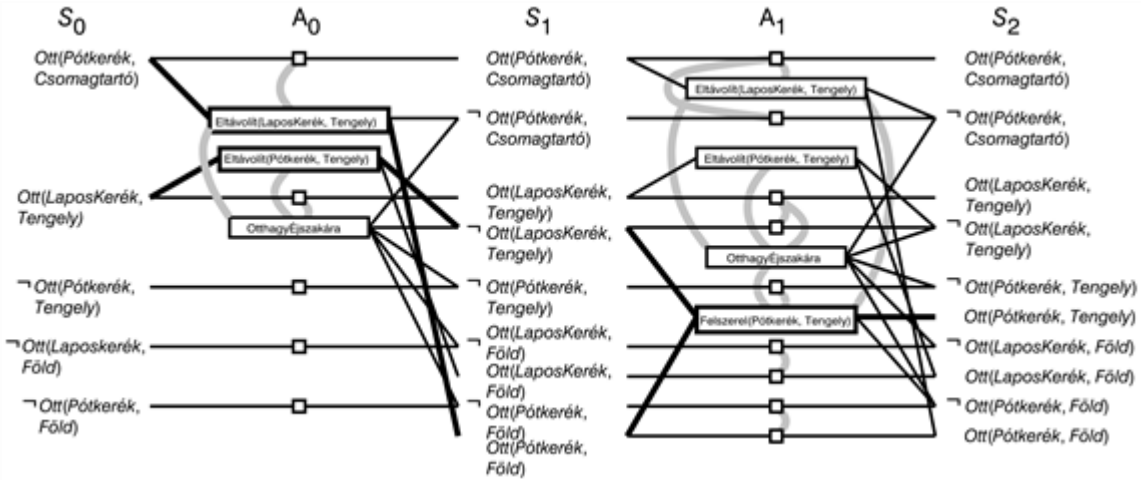
```
function GRAPHPLAN(probléma) returns megoldás vagy kudarc

  gráf ← KIINDULÓ-TERVKÉSZÍTÉSI-GRÁF(probléma)
  célok ← CÉLOK[probléma]
  loop do
    if célok mind nem mutex az utolsó szintjén a gráf-nak then do
      megoldás ← MEGOLDÁS-KINYERÉS(gráf, célok, HOSSZ(gráf))
      if megoldás ≠ kudarc then return megoldás
      else if NINCS-LEHETSÉGES-MEGOLDÁS(gráf) then return kudarc
    gráf ← GRÁF-BŐVÍTÉS(gráf, probléma)
```

Kövessük végig a GRAPHPLAN működését a 11.1. alfejezet kerékcsere problémáján! A teljes gráf a 11.14. ábrán látható. A GRAPHPLAN első lépésben inicializálja a tervkészítési gráfot egy egyszintű (S_0 szint) gráfra, ami a kiindulási állapot öt literálját tartalmazza. Az *Ott*(*Pótkerék*, *Tengely*) célliterál nincs jelen az S_0-ban, ezért nem kell meghívnunk a MEGOLDÁS-KINYERÉST, hiszen biztosak vagyunk benne, hogy még nincs megoldás. Ehelyett a GRÁF-BŐVÍTÉS hozzáad három cselekvést, melyeknek az előfeltétele már teljesül az S_0 szinten (például az összes cselekvés, kivéve a *Felszerel*(*Pótkerék*, *Tengely*) – cselekvést). Hozzáadja továbbá a megőrző-cselekvéseket S_0 összes literáljához. A cselekvések következményeit az S_1 szinthez adjuk hozzá. A GRÁF-BŐVÍTÉS ezek után mutex kapcsolatokat keres, és hozzáadja ezeket a gráfhoz.

Created by XMLmind XSL-FO Converter.

11.14. ábra - A kerékcsere probléma tervkészítési gráfja az S_2 szintre bővítés után. A mutex kapcsolatokat szürke vonalak jelölik. Csak néhány fontos mutexet mutatunk, mert az ábra olvashatatlanná válna, ha az összeset jelölnénk. A megoldást megvastagított vonalak és kiemelések jelölik.



Az *Ott*(*Pótkerék*, *Tengely*) cselekvés még mindig nincs jelen az S_1 állapotban, ezért most sem hívjuk meg a MEGOLDÁS-KINYERÉS lépést. A GRÁF-BŐVÍTÉS hívással a 11.14. ábrán látható tervkészítési gráfot kapjuk. Most, hogy a cselekvések teljes skálája rendelkezésre áll, ideje néhány példát nézni a mutex kapcsolatokra és a hatásaikra:

- *Nem konzisztens hatások:* az *Eltávolít*(*Pótkerék*, *Csomagtartó*) mutex kapcsolatban van az *OtthagyÉjszakára* cselekvéssel, mert az egyik következménye az *Ott*(*Pótkerék*, *Föld*), míg a másiké ennek negáltja.

- *Interferencia:* az *Eltávolít*(*Pótkerék*, *Tengely*) mutex az *OtthagyÉjszakára* cselekvéssel, mert az egyiknek előfeltétele az *Ott*(*Laposkerék*, *Tengely*), míg a másiknak ennek negáltja a következménye.

- *Versenyhelyzet:* a *Felszerel*(*Pótkerék*, *Tengely*) mutex az *Eltávolít*(*Laposkerék*, *Tengely*) cselekvéssel, mert az egyiknek előfeltétele az *Ott*(*Laposkerék*, *Tengely*), míg a másiké ennek negáltja.

- *Inkonzisztens tartó:* az *Ott*(*Pótkerék*, *Tengely*) és az *Ott*(*Laposkerék*, *Tengely*) kölcsönösen kizáróak az S_2-ben, mert az *Ott*(*Pótkerék*, *Tengely*) csak a *Felszerel*(*Pótkerék*, *Tengely*) cselekvéssel érhető el, ami mutex a megőrző cselekvéssel, mely az *Ott*(*Laposkerék*, *Tengely*) egyetlen elérése. A mutex kapcsolatok így azonnal érzékelik a konfliktust, ami abból adódik, hogy két objektumot ugyanazon időpontban azonos helyre próbálunk tenni.

Ez alkalommal visszamegyünk a ciklus kezdetére. A cél minden literálja jelen van S_2-ben, és egyik sem áll kölcsönös kizárásban egyetlen másikkal sem. Ez azt jelenti, hogy egy megoldás létezhet, és a MEGOLDÁS-KINYERÉSmegtalálhatja. Lényegében a MEGOLDÁS-KINYERÉS egy kétértékű kényszerkielégítési problémát old meg, melynek változói a szintek cselekvései, és az értékeik pedig azt jelzik, hogy *benne vannak-e* vagy *nincsenek benne* a tervben. Ehhez egy egyszerű kényszerkielégítési algoritmust használhatunk, vagy definiálhatjuk a MEGOLDÁS-KINYERÉS-t mint egy keresési problémát. Itt a keresés minden állapota a kielégítetlen célok egy halmazát tartalmazza, valamint egy mutatót a tervkészítési gráf egy szintjére. Ezt a keresési problémát a következőképpen definiáljuk:

- A kiindulási állapot a tervkészítési gráf utolsó szintje (S_n) a tervkészítési probléma céljaival egyetemben.

- Az S_i szint állapotában rendelkezésre álló cselekvések összessége, az A_{i-1} cselekvéseinek azon konfliktusmentes részhalmaza, melyek következményei elérik az állapotban lévő célokat. Az eredményként előálló állapot szintje S_{i-1}, céljai pedig a kiválasztott cselekvések előfeltételei. A „konfliktusmentességen” olyan cselekvések halmazát értjük, amelyek között nincs kettő, amelyek kölcsönösen kizárnák egymást, és az előfeltételeik között sem szerepelnek kölcsönösen kizáró párok.

- A cél, hogy elérjünk az S_0 szinten egy állapotot úgy, hogy minden cél teljesüljön.

Created by XMLmind XSL-FO Converter.

- Minden cselekvés költsége 1.

Ehhez a problémához az S_2 szintről indulunk, az *Ott*(*Pótkerék*, *Tengely*) céllal. Az egyetlen választásunk, hogy elérjük ezt a célt, a *Felszerel*(*Pótkerék*, *Tengely*) cselekvés. Ez az S_1 keresési állapothoz visz minket, melynek céljai az *Ott*(*Pótkerék*, *Föld*) és az ¬*Ott*(*Laposkerék*, *Tengely*). Az előbbit az *Eltávolít*(*Pótkerék*, *Csomagtartó*) cselekvéssel érhetjük el, míg a későbbit az *Eltávolít*(*Laposkerék*, *Tengely*) vagy az *OtthagyÉjszakára* cselekvések egyikével. Az *OtthagyÉjszakára* kölcsönösen kizáró kapcsolatban van a *Eltávolít*(*Pótkerék*, *Csomagtartó*)-val, ezért az egyetlen megoldás, hogy az *Eltávolít*(*Pótkerék*, *Csomagtartó*) és az *Eltávolít*(*Laposkerék*, *Tengely*) cselekvéseket választjuk. Ez az S_0 keresési állapotra vezet, az *Ott*(*Pótkerék*, *Csomagtartó*) és az *Ott*(*Laposkerék*, *Tengely*) célokkal. Mindkettő jelen van az állapotban, így megvan a megoldásunk: az *Eltávolít*(*Pótkerék*, *Csomagtartó*) cselekvés és az *Eltávolít*(*Laposkerék*, *Tengely*) az A_0 szinten, melyet az A_1-ben a *Felszerel*(*Pótkerék*, *Tengely*) követ.

Tudjuk, hogy a tervkészítés polinomiális helyigényű, és hogy a tervkészítési gráf elkészítése polinomiális idejű, ezért tudjuk, hogy a megoldás kinyerése legrosszabb esetben kezelhetetlen. Ebből kifolyólag valamifajta heurisztika segítségére lesz szükségünk, hogy a cselekvések közül válasszunk a visszafelé keresés során. A gyakorlatban jól működik egy mohó algoritmus, mely a literálok között a szintköltség alapján választ. Bármely célhalmazra a következő sorrendben járunk el:

1. Elsőként a legmagasabb szintköltségű literált választjuk.

2. Hogy teljesítsük ezt a literált, válasszuk először a legegyszerűbb előfeltételekkel rendelkező cselekvést, azaz válasszuk azt az akciót, melynek előfeltételeire a szintköltségek összege (vagy maximuma) a legkisebb!

4.3. A Graphplan algoritmus leállása

Eddig átsiklottunk a leállás kérdése fölött. Lehetünk-e biztosak abban, hogy amennyiben egy problémának nincs megoldása, a GRAPHPLAN algoritmus nem kerül végtelen ciklusba, iterációnként bővítve a tervkészítési gráfot? A válasz igen, de ennek bizonyítása túlmutat ennek a könyvnek a keretein. Itt csak a fő ötleteket körvonalazzuk, különösen azokat, melyek a tervkészítési gráfok általános tulajdonságaira világítanak rá.

Az első lépés, hogy észrevegyük, hogy a tervkészítési gráfok bizonyos tulajdonságai monoton növekvők vagy csökkenők. „X monoton növekvő” azt jelenti, hogy az X-ek halmaza az *i*-edik szinten az i + 1-edik szint halmazának (nem feltétlenül valódi) részhalmaza. A tulajdonságok a következők:

- *Monoton növekvő literálok.* Ha egy literál megjelenik egy adott szinten, akkor az ezt követő összes szinten megjelenik. Ez a megőrző cselekvések miatt van, azaz, ha egy literál megjelenik, a megőrző cselekvések hatására örökké megmarad.

- *Monoton növekvő cselekvések.* Ha egy cselekvés megjelenik egy adott szinten, akkor az ezt követő összes szinten megjelenik. Ez a literálok növekedésének a következménye, azaz, ha egy cselekvés előfeltételei megjelennek egy szinten, akkor az összes ezt követő szinten rendelkezésre állnak, így a cselekvés is.

- *A kölcsönös kizárások monoton csökkenése.* Ha két cselekvés kölcsönösen kizárja egymást az A_i szinten, akkor az összes ezt *megelőző* szinten, ahol mindketten megjelennek, szintén kölcsönösen kizárják egymást. Ugyanez igaz a literálok közötti kölcsönös kizárásokra. Ez az ábrákon nem mindig látható, mert ezek a következő egyszerűsítést használják: nem ábrázolják sem azokat a literálokat, melyek egy adott S_i szinten nem teljesülnek, sem azokat a cselekvéseket, amelyek egy adott A_i szinten nem végrehajthatók. Láthatjuk, hogy a kölcsönös kizárások monoton csökkenése igaz, ha figyelembe vesszük, hogy a nem látható literálok és cselekvések minden mással kölcsönösen kizárják egymást.

A bizonyítás egy kicsit összetett, de a következő esetekkel kezelhető: ha az *A* és *B* cselekvések kölcsönösen kizárják egymást az A_i szinten, akkor ezt csak a háromféle kizárás egyike okozhatja. Az első kettő – az inkonzisztens hatások, illetve a következtetések – a cselekvések tulajdonságai, így, ha a cselekvések kölcsönösen kizárják egymást az A_i szinten, akkor minden szinten kizárók lesznek. A harmadik eset a versenyhelyzet, az S_i szint feltételeitől függ: ennek a szintnek tartalmaznia kell az A_1 olyan előfeltételét, amely kölcsönösen kizáró a B_1 előfeltétellel. Ez a két előfeltétel akkor lehet kölcsönösen kizáró, ha egymás negáltjai (amely esetben minden szinten kölcsönösen kizárók), vagy ha minden cselekvés, amely teljesíti az egyiket, kölcsönösen kizáró az összes cselekvéssel, mely a másikat érheti el. Mi már tudjuk, hogy a rendelkezésre álló cselekvések monoton növekednek, így indukcióval bizonyítható, hogy a kölcsönös kizárások csökkenők.

Created by XMLmind XSL-FO Converter.

Mivel a cselekvések és a literálok növekednek, a kölcsönös kizárások csökkennek, és mivel csak véges számú cselekvés és literál van, minden tervkészítési gráf szükségszerűen kiegyenlítődik – azaz minden egymást követő szint azonos lesz. Ha hiányzik a probléma egy célja, vagy a célok között vannak kölcsönösen kizárók, miután a gráf kiegyenlítődött, akkor a probléma soha nem oldható meg, így leállíthatjuk a GRAPHPLAN algoritmust, és hibával térhetünk vissza. Ha a gráf kiegyenlítődik, és minden cél jelen van kizárások nélkül, de a MEGOLDÁS-KINYERÉS sikertelen a megoldás megtalálásában, akkor lehet, hogy véges lépésszámban bővítenünk kell a gráfot, de végezetül megállhatunk. A megállás problémaköre ennél összetettebb, amit itt nem tárgyalunk.

5. Tervkészítés ítéletlogikával

A 10. fejezetben láthattuk, hogy a tervkészítés elvégezhető a szituációkalkulus egy tételének bizonyításával. A tétel azt mondja, hogy adott kiinduló állapotra és követőállapot-axiómákra, amelyek a cselekvések következményeit írják le, a cél egy adott cselekvéssorból adódó helyzetben igaz lesz. Ezt a megközelítést már 1969-ben sem találták eléggé hatékonynak érdekes tervek megtalálására. Napjaink fejlesztései az ítéletlogikához készített hatékony magyarázatadó algoritmusokra (lásd 7. fejezet) ismét felkeltették az érdeklődést a tervkészítés logikai magyarázatkészítési megközelítésére.

Az ebben a fejezetben tárgyalt megközelítés a tételbizonyítás helyett egy cselekvéssor **kielégíthetőség**ének (**satisfiability**) vizsgálatán alapul. Ítéletlogikai mondatok modelljét fogjuk megtalálni, melyek a következőképpen néznek ki:

kiinduló állapot ∧ *minden lehetséges cselekvésleírás* ∧ *cél*

A mondat ítéletlogikai szimbólumokat fog tartalmazni, melyek a lehetséges cselekvés-előfordulások megfelelői. A modell, ami kielégíti a mondatot, igaz értéket rendel azokhoz a cselekvésekhez, amelyek részei a helyes tervnek, és hamis értéket a többihez. Egy olyan célkitűzés, mely egy nem helyes terv megfelelője, nem lesz modell, mivel nem lesz konzisztens azzal a feltételezéssel, hogy a cél igaz. Ha a tervkészítési probléma megoldhatatlan, akkor a mondat sem kielégíthető.

5.1. Tervkészítési problémák ítéletlogikai leírása

A STRIPS problémák ítéletlogikai leírásra való fordítását a tudásreprezentációs ciklus egy iskolapéldáján mutatjuk be: egy megfelelő axiómakészlettel indulunk, és úgy találjuk, hogy ezek az axiómák hamis, nem várt modelleket tesznek lehetővé, ezért további axiómákat adunk hozzájuk.

Kezdjük a nagyon egyszerű légi szállítási problémával. A kiinduló állapotban (0. időpont) a P_1 repülő az *SFO* repülőtéren van, míg a P_2 repülő a *JFK*-n. A cél, hogy a P_1 legyen a *JFK*, a P_2 pedig az *SFO* reptéren, azaz a repülőgépeknek helyet kell cserélniük. Először egymástól eltérő ítéletlogikai szimbólumokra van szükségünk az egyes időlépések kijelentéseihez. Az időpillanatok jelöléséhez felső indexeket használunk, mint a 7. fejezetben. Így a kiinduló állapot felírása az

$Ott(P_1, SFO)^0 \wedge Ott(P_2, JFK)^0$

(Emlékezzünk arra, hogy az $Ott(P_1, SFO)^0$ egy atomi szimbólum!) Mivel az ítéletlogika nem használja a zárt világ feltételezést, ezért a kiinduló állapotban definiálnunk kell a *nem* igaz ítéletlogikai állításokat is. Ha néhány állítás nem ismert a kiinduló állapotban, akkor ezek maradhatnak meghatározatlanok (**nyílt világ feltételezés**). Ebben a példában az

$\neg Ott(P_1, JFK)^0 \wedge \neg Ott(P_2, SFO)^0$

kiinduló állapotot specifikáljuk. A célt magát egy megadott időlépéshez kell kapcsolni. Mivel nem tudjuk *a priori*, hogy hány lépésre van szükség a cél eléréséhez, megpróbálhatjuk feltételezni, hogy a cél igaz a kiinduló állapotban, $T = 0$ időpontban. Azaz a következőt állítjuk: $Ott(P_1, JFK)^0 \wedge Ott(P_2, SFO)^0$. Ha ez *sikertelen*, újra próbálkozunk $T = 1$-gyel, és így tovább, amíg a legkisebb megfelelő tervhosszt el nem érjük. T minden értékére a tudásbázis a 0. időponttól a T-ig tartó lépéseket fedő mondatokat fogja tartalmazni. Hogy biztosítsuk a leállást, egy önkényes T_{max} felső korlátot kell állítanunk. Ezt az algoritmust mutatja a 11.15. ábra. Egy másik lehetséges megközelítést, mely elkerüli a többszörös megoldási kísérleteket, a 11.17. feladatban tárgyalunk.

11.15. ábra - A SATPLAN algoritmus. A tervkészítési feladatot egy konjunktív normál formájú mondatra fordítjuk le, amelyben minden cél egy megadott *T* időpillanatban

Created by XMLmind XSL-FO Converter.

teljesül, és *T*-ig minden lépésre tartalmazza az axiómákat. (A fordítás részleteit a szövegben írjuk le.) Ha a kielégíthetőségi algoritmus talál modellt, akkor a tervet a modellben az *igaz* értéket kapott ítéletlogikai szimbólumokhoz tartozó cselekvések kinyerésével kapjuk meg. Ha nem létezik modell, akkor a folyamatot ismételjük, a célt egy lépéssel későbbre mozgatva.

```
function SATPLAN(probléma, T_max) returns megoldás vagy kudarc
  inputs: probléma, egy tervkészítési probléma
          T_max, a terv maximális hossza

  for T = 0 to T_max do
    cnf, leképezés ← SATRA-FORDÍTÁS(probléma, T)
    feladat ← SAT-MEGOLDÓ(cnf)
    if feladat nem üres then
      return MEGOLDÁS-KINYERÉS(feladat, leképezés)
  return kudarc
```

A következő kérdés, hogy hogyan kódoljuk a cselekvés leírásokat az ítéletlogikában. A legegyszerűbb megközelítés, hogy minden cselekvés megjelenésére egy ítéletlogikai szimbólumot vezetünk be. Például a *Repül*(P_1, *SFO*, *JFK*)^0 igaz, ha a P_1 repülőgép az *SFO*-ról a *JFK*-ra repül a 0. időpillanatban. Csakúgy, mint a 7. fejezetben, felírjuk az ítéletlogikai változatait a következő állapotaxiómáknak, melyeket a 10. fejezetben a szituációkalkulushoz fejlesztettünk ki. Adott például az

$$Ott(P_1, JFK)^1 \Leftrightarrow (Ott(P_1, JFK)^0 \wedge \neg(Repül(P_1, JFK, SFO)^0 \wedge Ott(P_1, JFK)^0))$$

$$\vee\ (Repül(P_1, SFO, JFK)^0 \wedge Ott(P_1, SFO)^0) \quad (11.1)$$

axióma. Ennek jelentése, hogy a P_1 repülő a *JFK*-n lesz az 1-es időpillanatban, ha a *JFK*-n volt a 0. időpillanatban, és onnan nem repült el, vagy ha az *SFO*-n volt a 0. időpillanatban, és elrepült a *JFK*-ra. Egy ilyen axiómára minden egyes repülőgéphez, repülőtérhez és időpillanathoz szükségünk van. Mindezeken túl minden egyes hozzáadott repülőtér egy új kiinduló, illetve ott végződő útvonalat ad az összes repülőtérhez, ezért további diszjunkciókat ad az összes axióma jobb oldalához.

Ha ezek az axiómák megvannak, akkor egy kielégíthetőségi algoritmust futtathatunk a terv megtalálására. Kell, hogy legyen olyan terv, ami a célt a $T = 1$ időpillanatban eléri, nevezetesen az a terv, melyben a két repülőgép helyet cserél. Tegyük fel, hogy a tudásbázis a következő:

$$kiinduló\ állapot \wedge követő\ állapot\ axiómák \wedge cél^1 \quad (11.2)$$

mely szerint a cél a $T = 1$ időpillanatban igaz. Ellenőrizhető, hogy az eljárás, melyben a

$$Repül(P_1, SFO, JFK)^0 \text{ és a } Repül(P_2, JFK, SFO)^0$$

igaz, és minden más cselekvésszimbólum hamis, modellje a tudásbázisnak. Eddig itt minden rendben. Vannak más lehetséges modellek, melyeket a kielégíthetőségi algoritmus visszaadhat? Valójában igen. Ezen modellek mindegyike kielégítő terv? Sajnos nem. Vegyük például a

$$Repül(P_1, SFO, JFK)^0, \text{ a } Repül(P_1, JFK, SFO)^0 \text{ és a } Repül(P_2, JFK, SFO)^0$$

cselekvésszimbólumok által specifikált meglehetősen butácska tervet. Ez a terv buta, mert a P_1 repülő az *SFO*-ról indul, ezért a *Repül*(P_1, *JFK*, *SFO*)^0 cselekvés végrehajthatatlan. Mindazonáltal a terv modellje a (11.2) egyenlet mondatának! Ez annyit jelent, hogy megfelel mindennek, amit eddig a problémáról állítottunk. Hogy megértsük miért, egy kicsit közelebbről meg kell vizsgálnunk, hogy a következő állapot axiómák mit mondanak el a cselekvésekről, melyeknek az előfeltételei nem teljesülnek (mint a (11.1) egyenletben). Az axiómák helyesen írják le, hogy semmi sem történik, ha egy ilyen cselekvést végrehajtunk (lásd 11.15. feladat), de nem állítják, hogy egy ilyen cselekvés nem hajtható végre! Hogy elkerüljük az illegális cselekvéseket tartalmazó

Created by XMLmind XSL-FO Converter.

tervek készítését, további **előfeltétel axiómá**kat (**precondition axiom**s) kell felvennünk, melyek biztosítják, hogy egy cselekvés megjelenéséhez az előfeltételeknek teljesülni kell.[6] Például szükségünk van az

$$Repül(P_1, JFK, SFO)^0 \Rightarrow Ott(P_1, JFK)^0$$

axiómára, mert az $Ott(P_1, JFK)^0$ hamis a kiinduló állapotban, így ez az axióma biztosítja, hogy a $Repül(P_1, JFK, SFO)^0$ minden modellben szintén hamis legyen. Az előfeltétel axiómák hozzáadásával már csak pontosan egy modell van, ami teljesíti az összes axiómát, amikor a célt az 1-es időpillanatban érjük el, nevezetesen az a modell, melyben a P_1 repülőgép a *JFK*-ra, a P_2 repülőgép pedig az *SFO*-ra repül. Vegyük észre, hogy ez a megoldás két párhuzamos cselekvést tartalmaz! Csakúgy, mint a GRAPHPLAN vagy a részben rendezett tervkészítés esetén.

További meglepetésekre számíthatunk, amikor egy új repülőteret illesztünk be, a *LAX*-ot (Los Angeles repülőtere). Most minden repülőgéphez minden állapotban két megengedett cselekvés is tartozik. Amikor egy kielégíthetőségi algoritmust futtatunk, úgy találjuk, hogy a $Repül(P_1\ SFO, JFK)^0$, a $Repül(P_2, JFK, SFO)^0$ és a $Repül(P_2, JFK, LAX)^0$-ból álló modell kielégíti az axiómákat. Azaz a követő állapot axiómák és az előfeltétel axiómák megengedik, hogy egy repülőgép egyszerre két célállomásra is repülhessen! A kiinduló állapot a P_2 mindkét repülőútjának előfeltételeit teljesíti, így a követő állapot axiómák szerint a P_2 az *SFO*-n és a *LAX*-on található az 1-es időpillanatban, azaz a cél teljesül. Nyilvánvaló, hogy további axiómákat kell felvennünk, hogy kizárjuk ezeket a hibás megoldásokat. Az egyik megközelítés, hogy **cselekvéskizáró axiómá**kat (**action exclusion axiom**s) veszünk fel, melyek meggátolják az egyidejű cselekvéseket. Pédául teljes kizárást érhetünk el, ha az összes

$$\neg(Repül(P_2, JFK, SFO)^0 \wedge Repül(P_2, JFK, LAX)^0)$$

alakú axiómát felvesszük. Ezek az axiómák biztosítják, hogy semelyik két cselekvés ne eshessen azonos időpontra. Ezek kizárják a hibás terveket, hatásukra minden terv teljesen rendezett lesz. Ez elveszíti a részben rendezett tervek rugalmasságát, és azáltal, hogy megnöveli a terv lépésszámát, a számítási idő is meghosszabbodhat.

A teljes kizárás helyett alkalmazhatunk csupán részleges kizárást, amikor az egyidejű cselekvéseket csak akkor zárjuk ki, ha kölcsönhatásba kerülnek. A feltételek a mutex cselekvések megfelelői: két cselekvés nem hajtható végre egyidejűleg, ha az egyik negálja a másik egy előfeltételét vagy következményét. Például a $Repül(P_2, JFK, LAX)^0$ és a $Repül(P_2, JFK, SFO)^0$ együtt nem következhet be, mert mindkettő negálja a másik előfeltételét. Másrészről azonban a $Repül(P_2, JFK, SFO)^0$ és a $Repül(P_2, JFK, SFO)^0$ együtt is bekövetkezhet, hiszen a járatok nem zavarják egymást. A részleges kizárás a teljes rendezés kényszere nélkül kiküszöböli a hibás terveket.

A kizárási axiómák néha nagyon buta eszköznek tűnnek. Ahelyett hogy kikötnénk, hogy egy gép nem repülhet egyszerre két reptérre, mondhatnánk egyszerűen, hogy egyetlen tárgy sem lehet egyszerre két helyen:

$$\forall p, x, y, t\ x \neq y \Rightarrow \neg(Ott(p, x)^t \wedge Ott(p, y)^t)$$

Ez a tény a követő állapot axiómákkal kombinálva, azt *implikálja*, hogy egy repülő nem repülhet egyszerre két reptérre. Az ilyen tényeket **állapotkorlátozás**oknak (**state constraint**s) nevezzük. Az ítéletlogikában természetesen meg kell adnunk minden egyes állapotkorlátozás összes alappéldányát. A repülés feladatra az állapotkorlátozás elegendő az összes hibás terv kizárásához. Az állapotmegkötések gyakran sokkal tömörebbek, mint a cselekvéskizárási axiómák, ezek azonban a probléma eredeti STRIPS reprezentációjából nem mindig vezethetők le könnyedén.

Összefoglalva tehát a kielégíthetőségi tervkészítés, a kiinduló állapotot, a célt, a követő állapot axiómákat, az előfeltételeket, valamint a cselekvéskizárások vagy az állapotaxiómák egyikét tartalmazó mondathoz tartozó modellek megtalálását jelenti. Megmutatható, hogy az axiómák ezen halmaza elégséges, abban az értelemben, hogy a továbbiakban nincsenek hibás „megoldások”. Az ítéletlogikai mondatot kielégítő bármely modell megfelelő megoldása az eredeti problémának, azaz a terv minden sorba rendezése egy megengedett cselekvéssor, ami a célhoz vezet.

5.2. Az ítéletlogikai leírás bonyolultsága

[6] Figyeljük meg, hogy az előfeltétel axiómák felvétele azt jelenti, hogy nem kell a követő állapot axiómáknál a cselekvésekhez előfeltételeket felvennünk.

Created by XMLmind XSL-FO Converter.

Az ítéletlogikai megközelítés legnagyobb hátránya az eredeti problémából generált tudásbázis mérete. A *Repül*(p, a_1, a_2) cselekvésséma például $T \times |\,Repülők\,| \times |\,Repülőterek\,|^2$ számú különböző állításlogikai szimbólumot eredményez. Általánosságban a cselekvésszimbólumok számának korlátja a $T \times |\,Act\,| \times |\,O\,|^P$, ahol az $|\,Act\,|$ a cselekvéssémák száma, $|\,O\,|$ a problémakör objektumainak száma, valamint P a cselekvés sémák maximális aritása (paramétereinek száma). A klózok száma még nagyobb. 10 időlépés, 12 repülőgép és 30 repülőtér esetén a teljes cselekvéskizárási axiómában 583 millió klóz van.

Mivel a cselekvésszimbólumok száma a cselekvéssémák aritásával exponenciálisan nő, megoldás lehet az aritás csökkentése. Ezt a szemantikus hálóktól (lásd 10. fejezet) kölcsönvett ötlet segítségével tehetjük meg. A szemantikus hálók csak bináris predikátumokat használnak, az ennél több argumentummal rendelkező predikátumokat egy bináris predikátumhalmazra képezzük le, ami külön ad meg minden predikátumot. Ezt az ötletet a *Repül*(P_1, *SFO*, *JFK*)^0 cselekvésszimbólumra alkalmazva, három új szimbólumot kapunk:

$Repül_1(P_1)^0$: a P_1 repülő a 0. időpontban repült

$Repül_2(SFO)^0$: a járat kiinduló állomása az *SFO*

$Repül_3(JFK)^0$: a járat célállomása a *JFK*

Ez a **szimbólumfelosztás**nak (**symbol splitting**) nevezett folyamat kiküszöböli az exponenciális számú szimbólum igényét. Így most csak $T \times |\,Act\,| \times P \times |\,O\,|$ szimbólumra van szükség.

A szimbólum felosztás önmagában lecsökkenti ugyan a szimbólumok számát, de a tudásbázisban szereplő axiómák számát nem csökkenti automatikusan. Azaz, ha minden klózban szereplő cselekvésszimbólumot egyszerűen csak három szimbólum konjunkciójára cserélnénk, a tudásbázis mérete nagyjából azonos maradna. A szimbólumfelosztás valóban csökkenti a tudásbázis méretét, mivel néhány szétbontott szimbólum feleslegessé válik bizonyos axiómákban, és elhagyható. Vegyük például a (11.1) egyenletben szereplő követő állapot axiómát, módosítva, hogy tartalmazza *LAX*-ot, és elhagyva a cselekvés előfeltételeket (melyeket külön előfeltétel axiómákkal fedünk le):

$$Ott(P_1, JFK)^1 \Leftrightarrow (Ott(P_1, JFK)^0 \wedge \neg Repül(P_1, JFK, SFO)^0 \wedge \neg Repül(P_1, JFK, LAX)^0)$$

$$\vee Repül(P_1, SFO, JFK)^0 \vee Repül(P_1, LAX, JFK)^0$$

Az első feltétel szerint a P_1 a *JFK*-n lesz, ha a 0. időpontban ott volt, és nem repült el egyetlen másik városba sem. A második azt mondja, hogy ott lesz, ha mindegy, hogy honnan, de a *JFK*-ra repül. A felosztási szimbólumok használatával, egyszerűen elhagyhatjuk azon argumentumokat, melyek értéke nem számít:

$$Ott(P_1, JFK)^1 \Leftrightarrow (Ott(P_1, JFK)^0 \wedge \neg Repül_1(P_1)^0 \wedge Repül_2(JFK)^0) \vee Repül_1(P_1)^0$$

$$\wedge Repül_3(JFK)^0$$

Vegyük észre, hogy az *SFO* és a *LAX* már nem szerepel az axiómában. Általánosabban a felosztási szimbólumoknak köszönhetően a követő állapot axiómák mérete függetlenné válik a repülőterek számától. Hasonló redukció történik az előfeltétel és a cselekvéskizárási axiómákkal (lásd 11.16. feladat). A korábban bemutatott 10 időlépés, 12 repülő és 30 reptér esetében a teljes cselekvéskizárási axóma mérete 583 millió klózról 9360 klózra csökken.

A módszernek egyetlen hátulütője van: a felosztási szimbólum reprezentáció nem engedi meg a párhuzamos cselekvéseket. Vegyük a *Repül*(P_1, *SFO*, *JFK*)^0 és a *Repül*(P_2, *JFK*, *SFO*)^0 párhuzamos cselekvéseket. A felosztott megadásra alakítva a

$$Repül_1(P_1)^0 \wedge Repül_2(SFO)^0 \wedge Repül_3(JFK)^0 \wedge$$

$$Repül_1(P_2)^0 \wedge Repül_2(JFK)^0 \wedge Repül_3(SFO)^0$$

kifejezést kapjuk. Innentől már nem állapítható meg mi történt! Tudjuk, hogy P_1 és P_2 repült, de nem tudjuk azonosítani a járatok kiinduló és célállomásait. Ez annyit tesz, hogy egy teljes cselekvéskizárási axiómát kell használni, annak korábban jelzett hátrányaival.

A kielégíthetőségen alapuló tervkészítők nagyon nagy tervkészítési problémákat képesek kezelni – például a többtucatnyi dobozból álló kockavilág probléma 30 lépéses tervét. A propozíciós kódolás mérete, valamint a megoldás költsége nagymértékben problémafüggő, de a legtöbb esetben az ítéletlogikai axiómák t árolásához szükséges memóriaméret a szűk keresztmetszet. Ennek a kutatásnak egy érdekes eredménye, hogy a

Created by XMLmind XSL-FO Converter.

visszalépéses algoritmusok (mint a DPLL) gyakran hatékonyabbak voltak a tervkészítési problémák megoldásában, mint a lokális keresési algoritmusok (például a WALKSAT). Ennek oka, hogy az ítéletlogikai axiómák többsége Horn-klóz, ami egység terjesztéstechnikákkal kezelhető hatékonyan. Ez a megfigyelés hibrid algoritmusok kifejlesztéséhez vezetett, melyek a véletlen keresési, a hiba-visszaterjesztési és az egységterjesztési módszereket kombinálják.

6. A tervkészítési módszerek elemzése

A mesterséges intelligencia területén belül, a tervkészítés jelenleg nagy figyelemnek örvend. Ennek egyik oka, hogy egyesíti az eddig bemutatott két nagy területet, a *keresést* és a *logikát*. A tervkészítő tekinthető egy programnak, ami vagy egy megoldást keres, vagy (konstruktívan) bizonyítja egy megoldás létezését. A két terület alapötleteinek keresztezése az utóbbi évtizedben több nagyságrendnyi javulást hozott a hatékonyság tekintetében, és egyúttal megnövelte a tervkészítők ipari alkalmazásokban való felhasználhatóságát. Sajnos még mindig nincs tiszta képünk arról, hogy mely problémákra mely megoldások a legalkalmasabbak. Később valószínűleg új technikák kerülnek majd előtérbe, melyek dominálnak majd a meglévők felett.

A tervkészítési feladat elsősorban a kombinatorikus robbanás kézbentartásáról, kontrollálásról szól. Egy *p* elemi állítást tartalmazó problématérben 2^p állapot lehetséges. Összetett problémakörökben *p* nagyon nagy lehet. Vegyük azt, hogy a problématér objektumainak tulajdonságai (*Pozíció*, *Szín* stb.) és relációi (*Ott*, *Rajta*, *Közte* stb.) vannak. Az állapotok száma, *d* objektummal és hármas relációkkal a problématérben $2^{d\wedge 3}$. Végkövetkeztetésként – a legrosszabb esetet feltételezve – megállapíthatjuk, hogy a tervkészítés reménytelen.

Az ilyen pesszimizmus ellen az „oszd meg és uralkodj” elv hatásos fegyver lehet. A legjobb esetben – amennyiben a probléma teljesen dekomponálható – az „oszd meg és uralkodj” elv exponenciális gyorsulást hozhat. A dekomponálhatóságot azonban a cselekvések negatív kölcsönhatásai megszüntetik. A részben rendezett tervkészítő ezt egy erős reprezentációs megközelítéssel, az okozati kapcsolatokkal kezelte, de sajnos a konfliktusok mindegyike csak egy választással oldható fel (ami a konfliktusban álló cselekvést vagy a kapcsolat elé, vagy mögé sorolja) és ezen választások száma exponenciálisan nő. A GRAPHPLAN algoritmus a gráf felépítésekor kikerüli ezeket a döntéseket úgy, hogy az ütközéseket mutex kapcsolatokkal rögzíti anélkül, hogy a feloldás módjáról döntene. A SATPLAN algoritmus a mutex kapcsolatok hasonló területét fedi le, de ezt egy specifikus adatstruktúra felhasználása helyett az általános CNF alak felhasználásával teszi. Az, hogy ez mennyire jól működik, a felhasznált SAT megoldótól függ.

Néha a probléma hatékonyan megoldható, ha észrevesszük, hogy a negatív kölcsönhatások kizárhatók. Azt mondjuk, hogy egy feladatnak **sorba rendezhető részcéljai** (**serializable subgoals**) vannak, ha létezik a részcéloknak egy olyan rendezése, hogy a tervkészítő sorrendben elérheti őket anélkül, hogy a korábban már elért részcélokat felszámolná. A kockavilágban például, ha a cél egy torony megépítése (pédául az *A* a *B*-n, ami pedig az asztalon elhelyezkedő *C*-n van), akkor a részcélok felülről lefelé sorrendezhetők: ha először elérjük, hogy *C* az asztalon van, akkor a továbbiakban – mialatt a további részcélokat teljesítjük – ennek megszüntetésére soha nem lesz szükség. Egy tervkészítő, ami az alulról felfelé trükköt alkalmazza, a kockavilág bármely problémáját képes megoldani visszalépés nélkül (bár nem mindig a legrövidebb tervet adja).

Egy összetettebb példaként, a NASA Deep Space 1 űrhajóját irányító Távirányító Ágens tervkészítő esetén elhatározták, hogy az űrhajót irányító állítások legyenek sorba rendezhetők. Ez talán nem is annyira meglepő, hiszen az űrhajót a mérnökök úgy tervezték, hogy minél könnyebb legyen irányítani (más megkötések figyelembevételével). A célok sorrendezhetőségét kihasználva a Távirányító Ágens tervkészítő a keresés nagy részét megspórolhatta, így megfelelően gyors volt az űrhajó valós idejű irányítására, amit korábban lehetetlennek tartottak.

A kombinatorikus robbanás kordában tartására több lehetőség is van. Az 5. fejezetben láthattuk, hogy a kényszerkielégíthetőségi problémák (CSP-k) visszalépési számának kontrollálásához számos megoldás létezik, mint például a függőségirányított visszalépés. Ezen technikák mindegyike alkalmazható a tervkészítéshez is. Például a megoldás kinyerése egy tervkészítési gráfból kezelhető egy kétértékű CSP-ként, melynek változói azt jelzik, hogy egy adott cselekvés bekövetkezik-e egy adott időpontban, vagy sem. A CSP megoldható az 5. fejezet bármely algoritmusával, például a min-konfliktusok algoritmussal. Egy közeli rokon, a BLACKBOX rendszerben használt megoldás, a tervkészítési gráfot CNF kifejezésre fordítja, melyből ezután egy SAT megoldóval nyeri ki a tervet. Ez a megközelítés jobb, mint a SATPLAN algoritmus, feltételezhetően azért, mert a tervkészítési gráf már számos lehetetlen állapotot és cselekvést eltávolított a feladatból. A GRAPHPLAN algoritmusnál is jobban működik, valószínűleg azért, mert a kielégíthetőségi keresés, mint a WALKSAT, nagyobb rugalmasságú a GRAPHPLAN algoritmus által használt szigorú visszalépéses keresésnél.

Created by XMLmind XSL-FO Converter.

Nem kétséges, hogy a GRAPHPLAN-hoz, a SATPLAN-hoz és a BLACKBOX-hoz hasonló tervkészítők előremozdították a tervkészítés fejlődését azáltal, hogy megnövelték a tervkészítő rendszerek teljesítményét, és tisztázták a kapcsolódó reprezentációs és komlexitási kérdéseket. Ezek a megoldások azonban eredendően ítéletlogikai megoldások, így korlátozott a problémakör, amelyet képesek kifejezni. (Például néhány tucat objektumot és helyet tartalmazó logisztikai problémának megfelelő CNF kifejezésekhez gigabájtnyi kapacitásra van szükség.) Bár a tervkészítési gráfhoz hasonló struktúrák továbbra is hasznosak lesznek, mint a heurisztikák forrása, valószínűnek tűnik, hogy elsőrendű reprezentációkra és algoritmusokra lesz szükség a további fejlődéshez.

7. Összefoglalás

Ebben a fejezetben a tervkészítési feladatot determinisztikus és teljesen megfigyelhető környezetek esetén definiáltuk. Bemutattuk a tervkészítési feladatok leírására használt főbb reprezentációkat, valamint számos algoritmikus megközelítést a megoldásukra. Idézzük fel a fontosabb állításokat:

- A tervkészítő rendszerek olyan problémamegoldó algoritmusok, melyek állapotok és cselekvések explicit ítéletlogikai (vagy elsőrendű) reprezentációin működnek. Ezek a leírások lehetővé teszik hatékony heurisztikák származtatását, valamint erős és rugalmas algoritmusok kifejlesztését a problémamegoldáshoz.

- A STRIPS nyelv az előfeltételek és következmények segítségével írja le a cselekvéseket, valamint a kiindulási és célállapotokat pozitív literálok konjunkciójaként adja meg. Az ADL nyelv lazít ezeken a megkötéseken, megengedi a diszjunkciót, a negálást és a kvantorokat.

- Az állapottér-keresés történhet előrefelé: **progreszív** (**progression**) vagy hátrafelé: **regresszív** (**regression**). Hatékony heurisztikák származtathatók a részcélok függetlenségét feltételezve és a tervkészítési feladat különböző gyengítéseivel.

- A részben rendezett tervkészítő (RRT) algoritmusok a tervek terében keresnek anélkül, hogy cselekvések teljesen rendezett sorozatára jutnának. A céltól visszafelé működnek a részcélok eléréséhez szükséges cselekvéseknek a tervhez való hozzáadásával. Ezek az algoritmusok különösen hatékonyak az „oszd meg és uralkodj” megközelítéssel kezelhető problémákra.

- Egy **tervkészítési gráf** (**planning graph**) megalkotható inkrementálisan a kiinduló állapotból elindulva. Minden egyes réteg az adott időpillanatban végrehajtható összes cselekvés és literál halmazát tartalmazza, rögzíti a kölcsönös kizárásokat vagy mutexeket, azaz olyan literálok és cselekvések közötti relációkat, melyek egyszerre nem következhetnek be. A tervkészítési gráfok hasznos heurisztikákra vezetnek az állapottér és a részben rendezett tervkészítők esetén, és közvetlenül felhasználhatók a GRAPHPLAN algoritmusban.

- A GRAPHPLAN algoritmus a tervkészítési gráfot dolgozza fel visszafelé keresést alkalmazva a terv kinyeréséhez. Lehetőséget biztosít továbbá még a cselekvések között bizonyos részben rendezésekre.

- A SATPLAN algoritmus a tervkészítési problémát ítéletlogikai axiómákra fordítja, és egy kielégíthetőségi algoritmust használ a helyes tervnek megfelelő modell megtalálására. Számos különböző ítéletlogikai reprezentációt fejlesztettek ki különböző mértékű tömörséggel és hatékonysággal.

- A tervkészítés fő megközelítései mindegyikének vannak nehézségei, és egyelőre nincs egyetértés abban, hogy melyik a legjobb. A versengés és a módszerek ötvözése komoly előrehaladást eredményezett a tervkészítő rendszerek hatékonyságában.

7.1. Irodalmi és történeti megjegyzések

A mesterséges intelligenciában a tervezés az állapottér-keresések vizsgálatából, tételbizonyításból, szabályozástechnikából, illetve a robotika, ütemezés és más területek gyakorlati szükségleteiből ered. Az első számottevő tervkészítő rendszer a STRIPS (Fikes and Nilsson, 1971), ezen hatások kölcsönhatását példázza. A STRIPS-et, az SRI-ben zajló Shakey robot projekt szoftverének tervkészítő komponenseként fejlesztették ki. Ennek teljes szabályozási struktúráját egy eszköz-cél analízist használó állapottér-kereső rendszer, a GPS, azaz az általános problémamegoldó (general problem solver) (Newell és Simon, 1961) mintájára készítették. A STRIPS a QA3 tételbizonyító rendszer egy verzióját (Green, 1969b) használta fel a cselekvések előfeltételeinek teljesítésére. Lifschitz a STRIPS precíz definícióit és elemzéseit adja meg (Lifschitz, 1986). Bylander bizonyította, hogy az egyszerű STRIPS tervkészítés PSPACE-teljes (Bylander, 1992). Fikes és Nilsson

Created by XMLmind XSL-FO Converter.

történelmi áttekintést adnak a STRIPS projektről, és feltárják ennek kapcsolatait az újabb tervkészítési törekvésekkel (Fikes és Nilsson, 1993).

A STRIPS-ben használt cselekvésreprezentációnak sokkal nagyobb hatása volt, mint algoritmikus megközelítésének. Ezt követően majdnem minden tervkészítő rendszer használta a STRIPS nyelv ilyen vagy olyan módosításait. Sajnos a különböző változatok elburjánzása szükségtelenül megnehezítette az összehasonlításokat. Az idő múlásával jobban megértettük az egyes formalizmusok korlátait és hátrányait. A cselekvésleíró nyelv (Action Description Language – ADL) (Pednault, 1986) lazította a STRIPS nyelv néhány megkötését, és lehetővé tette a valósághoz közelibb problémák leírását. Nebel az ADL leírást STRIPS leírásra fordító sémákat vizsgálja (Nebel, 2000). A problématér leíró nyelvet (Problem Domain Description Language – PDDL) (Ghallab és társai, 1998) egy számítógép által érthető szabványos szintaxisként vezették be a STRIPS, az ADL és egyéb nyelvek leírására. A PDDL-t 1998-tól az AIPS konferencián zajló tervkészítési versenyek szabványos nyelveként használták.

Az 1970-es évek elejének tervkészítői általában teljesen rendezett cselekvéssorokkal dolgoztak. A problémadekompozíciót úgy érték el, hogy minden részcélhoz egy résztervet készítettek, majd ezeket valamilyen sorrendben összefűzték. Hamarosan felfedezték, hogy ez a megközelítést, melyet Sacerdoti **lineáris tervkészítés**nek (**linear planning**) nevezett (Sacerdoti, 1975), nem teljes. Nem képes megoldani néhány nagyon egyszerű problémát, mint pédául a Sussman-anomáliát (lásd 11.11. feladat), amit Allen Brown a HACKER rendszerrel végzett kutatásai során fedezett fel (Sussman, 1975). Egy teljes tervkészítőnek meg kell engedni egy mondatban a különböző résztervekből származó cselekvések **összefésülés**ét (**interleaving**). A sorrendezhető részcélok alapötlete (Korf, 1987) pontosan megfeleltethető azon problémák halmazának, melyekre az öszszefésülésre nem alkalmas tervkészítők teljesek.

Az összefésülési probléma egyik megoldása a célregressziós tervkészítés volt, ami egy olyan technika, melyben egy teljesen rendezett terv lépéseit a részcélok közötti konfliktusok elkerülése érdekében újrarendezzük. Ezt Waldinger vezette be (Waldinger, 1975), és Warren WARPLAN rendszerében (Warren, 1974) szintén felhasználták. A WARPLAN-nel kapcsolatban szintén kiemelendő, hogy ez volt az első tervkészítő, melyet logikai programozási nyelven (Prolog) írtak, valamint az egyik legjobb példa arra a kiemelkedő gazdaságosságra, ami a logikai programozással nyerhető: a WARPLAN összesen száz kódsor, azaz csak töredéke az abban az időben ismert hasonló tervkészítők méretének. Az INTERPLAN (Tate, 1975a; 1975b) a Sussman-anomália és hasonló problémák elkerülése érdekében, szintén megengedte a tervlépések tetszőleges összefésülését.

A részben rendezett tervkészítés alapötletei tartalmazzák az ütközések érzékelését (Tate, 1975a) és az elért feltételek kölcsönhatásoktól való védelmét (Sussman, 1975). A részben rendezett tervek (melyeket akkor **feladathálók**nak – **task network** – neveztek) készítését a NOAH tervkészítő (Sacerdoti, 1975; 1977) és Tate NONLIN rendszere vezette be (Tate, 1975b; 1977).[7]

A következő húsz év kutatásaiban a részben rendezett tervkészítés dominált, mialatt azonban a problémakört széles körben nem értették. A TWEAK (Chapman, 1987) ezen időszak tervkészítési munkájának logikai reprezentációja és egyszerűsítése volt. Chapman felírása eléggé tiszta volt ahhoz, hogy bizonyítható legyen a tervkészítési problémák különböző megfogalmazásainak teljessége és követhetetlensége (NP-nehézsége és -eldönthetetlensége). Chapman munkája egy teljes részben rendezett tervkészítő első egyszerű és olvasható leírására vezetett (McAllester és Rosenblitt, 1991). A McAllester és Rosenblitt SNLP-nek (Soderland és Weld, 1991) nevezett algoritmusának implementációja széles körben elterjedt, és elsőként tette számos kutató számára lehetővé a részben rendezett tervkészítők megértését és kutatását. Az ebben fejezetben korábban bemutatott RRT algoritmus az SNLP-n alapul.

Weld csoportja fejlesztette ki az UCPOP-ot is (Penberthy és Weld, 1992), az első tervkészítőt ADL-ben kifejezett problémákra. Az UCPOP a kielégítetlen-célok-száma heurisztikát alkalmazta. Valamivel gyorsabban futott, mint az SNLP, de nagyon ritkán volt képes néhány tucat lépésnél többet tartalmazó tervek megtalálására. Bár javított heurisztikákat is kifejlesztettek az UCPOP módszerhez (Joslin és Pollack, 1994); (Gerevini és Schubert, 1996) az 1990-es években, a gyorsabb módszerek megjelenésével a részben rendezett tervkészítők háttérbe szorultak. Nguyen és Kambhampati a módszer visszatérését javasolták (Nguyen és Kambhampati, 2001): a tervkészítési gráfból megfelelő heurisztikák származtatásával a REPOP tervkészítőjük jobban skálázható, mint a GRAPHPLAN, továbbá ez vetélytársa lett a leggyorsabb állapottér-tervkészítőknek is.

[7] A terminológia kissé zavaros. Sok szerző a **nemlineáris** (**nonlinear**) jelzőt a „részben rendezett” értelemben használja. Ez kissé különbözik Sacerdotti eredeti szóhasználatától, aki ezt az összefésült tervekre használta.

Created by XMLmind XSL-FO Converter.

Avrim Blum és Merrick Furst felpezsdítette a tervkészítés területét GRAPHPLAN rendszerével (Blum és Furst, 1995; 1997), ami több nagyságrenddel gyorsabb volt az akkori részben rendezett tervkészítőknél. Más gráftervező rendszerek, mint az IPP (Koehler és társai, 1997), a STAN (Fox és Long, 1998) és az SGP (Weld és társai, 1998) hamarosan követték. Egy kicsivel korábban Ghallab és Laurelle a tervkészítő gráfhoz nagyon hasonló adatstruktúrát fejlesztettek ki (Ghallab és Laurelle, 1994). Az IXTET részben rendező tervkészítőjük az adatstruktúrát a keresés irányítására szolgáló heurisztikák kinyerésére alkalmazta. Nguyen és társai a tervkészítő gráfokból származtatott heurisztikák nagyon alapos áttekintését adták (Nguyen és társai, 2001). A mi tervkészítő gráfokról szóló leírásunk részben ezen, részben pedig Subbarao Kambhampati jegyzetein alapul. Ahogyan már a fejezetben említettük, többféle módon használható a tervezési gráf a megoldás keresésének irányítására. A 2002-es AIPS tervkészítő győztese, az LPG (Gerevini és Serina, 2002), a WALKSAT által ösztönözve lokális keresési technikát használt a tervkészítési gráf keresésére.

A tervkészítés mint kielégíthetőség és a SATPLAN algoritmus Kautz és Selman javaslata, akiket a kielégíthetőségi problémákra alkalmazott mohó lokális keresés meglepő sikere inspirált (Kautz és Selman, 1992) (lásd 7. fejezet). Kautz és társai szintén kutatták a STRIPS axiómák ítéletlogikai reprezentációjának különböző módjait, és azt találták, hogy a legtömörebb alakok nem feltétlenül vezettek a leggyorsabb megoldásokra (Kautz és társai, 1996). Ernst és társai szisztematikus elemzést hajtottak végre (Ernst és társai, 1997), valamint kifejlesztettek egy automatikus fordítót PDDL problémák ítéletlogikai megadásának generálására. A Kautz és Selman által kifejlesztett BLACKBOX tervkészítő (Kautz és Selman, 1998) a GRAPHPLAN és SATPLAN alapötleteit kombinálja.

Az állapottér-tervkészítőkkel kapcsolatos érdeklődés újjáéledésének úttörője Drew McDermott UNPOP programja (McDermott, 1996), amely elsőként javasolta a törlési listát figyelmen kívül hagyó egyszerűsített problémán alapuló távolság heurisztikát. Az UNPOP név a részben rendezett tervkészítőket illető túlzott figyelem egyik reakciója volt; McDermott szerint más megközelítések nem kapták meg a megérdemelt figyelmet. Bonet és Geffner heurisztikus kereső tervkészítője (Heuristic Search Planner – HSP) és ennek későbbi leszármazottjai (Bonet és Geffner, 1999) elsőként tették az állapottér-keresést alkalmazhatóvá nagyméretű tervkészítési feladatokra. A mai napig legsikeresebb állapottér-kereső Hoffmann FASTFORWARD vagy FF keresője, az AIPS 2000 tervkészítő versenyének győztese (Hoffmann, 2000). Az FF egyszerűsített tervkészítési gráf heurisztikát használ egy nagyon gyors, az előrefelé és a lokális keresést újszerűen ötvöző algoritmussal.

Napjainkban a tervek **bináris döntési diagram** (**binary decision diagram**) alakú reprezentációja hódít, ami egy véges automata tömör leírási módja, melyet a hardververifikációval foglalkozó közösség részletesen tanulmányozott (Clarke és Grumberg, 1987; McMillan, 1993). A bináris döntési diagramok jellemzőinek mint egy tervkészítési probléma megoldásának való megfelelés tulajdonságának bizonyítására számos technika elérhető. Cimatti és társai egy ezen a megközelítésen alapuló tervkészítőt mutattak be (Cimatti és társai, 1998). Más reprezentációkat szintén felhasználtak; például Vossen és társai az egész programozás tervkészítésre való felhasználhatóságát vizsgálja (Vossen és társai, 2001).

A döntőbizotság még nem döntött, de a különböző tervkészítési algoritmusok néhány nagyon érdekes összevetése már elérhető. Helmert a tervkészítési problémák számos osztályát vizsgálja, és megmutatja, hogy az NP-nehéz problémakörökben a megkötésalapú megközelítések, mint a GRAPHPLAN és a SATPLAN bizonyulnak a legjobbnak, míg a keresésalapú megközelítések azon problémakörökben hasznosak, ahol visszalépések nélkül elérhető egy megfelelő megoldás (Helmert, 2001). A GRAPHPLAN és SATPLAN algoritmusoknak nehézséget okoznak a sok objektumot tartalmazó problémák, mert ezek miatt sok cselekvésre van szükségük. Az esetek egy részében a probléma elkerülhető vagy elodázható, ha a cselekvések bizonyos részét csak szükség esetén generáljuk le dinamikusan, ahelyett hogy már a keresés előtt példányosítanánk mindegyiket. Weld a modern tervkészítő algoritmusok két kitűnő áttekintését adja (Weld, 1994; 1999). Érdekes látni az eltelt öt év változásait az áttekintésekben: az első a részben rendezett tervkészítésre fókuszál, míg a második a GRAPHPLAN és SATPLAN algoritmusokra. A *Readings in Planning* (Allen és társai, 1990) a terület legjobb korábbi cikkeinek minden részletre kiterjedő antológiája, beleértve néhány áttekintést is. Yang a részben rendezett tervkészítők könyv méretű áttekintését adja (Yang, 1997).

A tervkészítés kutatása a megjelenésétől kezdve központi kérdése a mesterséges intelligencia kutatásának, így a tervkészítésről született cikkek is számottevő részét képezik a folyóiratok és konferenciák anyagának. Specializált konferenciákat is rendeznek, mint az International Conference on AI Planning Systems (AIPS), az International Workshop on Planning and Scheduling for Space vagy a European Conference on Planning.

7.2. Feladatok

Created by XMLmind XSL-FO Converter.

11.1.

Írja le a különbségeket és hasonlóságokat a problémamegoldás és a tervkészítés között.

11.2.

Adottak a 11.2. ábra axiómái. Melyek az alkalmazható konkrét példányai a *Repül*(*p*, *honnan*, *hova*) cselekvésnek a következő állapotban:

$Ott(P_1, JFK) \land Ott(P_2, SFO) \land Repülő(P_1) \land Repülő(P_1)$

$\land Repülőtér(JFK) \land Repülőtér(SFO)$

11.3.

Nézzük meg, hogyan fordíthatunk le egy STRIPS sémahalmazt a szituációkalkulus követő állapot axiómáira (lásd 10. fejezet).

- Vegyük a *Repül*(*p*, *honnan*, *hová*) sémát. Írja le a *RepülElőfeltétel*(*p*, *honnan*, *hová*, *s*) predikátum logikai definícióját, ami igaz, ha a *Repül*(*p*, *honnan*, *hová*) előfeltételei teljesülnek az *s* szituációban.
- Ezután írjuk fel a *Ott*(*p*, *x*, *s*) követő állapot axiómáját, ami a cselekvéssémával azonos információt tartalmazza, feltételezve, hogy a *Repül*(*p*, *honnan*, *hová*) az ágens számára rendelkezésre álló egyetlen cselekvésséma.
- Most feltételezzük, hogy van egy további utazási lehetőség: *Teleportálás*(*p*, *honnan*, *hová*). Ennek egy további előfeltétele a ¬*Torzult*(*p*), extra következménye pedig a *Torzult*(*p*). Magyarázza meg, hogyan kell módosítani a szituációkalkulus tudásbázisát.
- Végezetül fejlesszen ki egy általános és pontosan specifikált megoldást egy STRIPS sémahalmaz követő állapot axióma halmazra való lefordításához.

11.4.

Egy majom a „majom-és-banán” feladattal találkozik egy laboratóriumban, ahol elérhetetlen banánfürtök lógnak a plafonról. Van egy doboz, ami lehetővé teszi a majom számára a banán elérését, amennyiben felmászik rá. Kiindulásképp a majom az *A*-ban van, a banánok a *B* helyen és a doboz a *C*-n. A majom és a doboz magassága *Alacsony*, de ha a majom a dobozra mászik, magassága *Magas* lesz, megegyezik a banánok magasságával. A majom számára elérhető cselekvések tartalmazzák a *Menj* egyik helyről a másikra, a *Tol* egy tárgyat egyik helyről a másikra, a *Felmászik* egy tárgyra, illetve *Lemászik* egy tárgyról, és a *Megfog*, illetve *Elenged* egy tárgyat cselekvéseket. A megmarkolás a tárgy megfogását eredményezi, ha a majom és a tárgy ugyanazon a helyen és ugyanabban a magasságban vannak.

a. Írja fel a kiinduló állapotot.

b. Írja le a hat cselekvés definícióját STRIPS alakban.

c. Tegyük fel, hogy a majom meg akarja tréfálni a tudósokat, akik teázni mentek, azáltal hogy megragadja a banánokat, de a dobozt az eredeti helyén hagyja. Írja fel ezt mint általános célt (például nem feltételezve, hogy a doboz szükségszerűen a *C* helyen van) a szituációkalkulus nyelvén. Megoldható ez egy STRIPS-szerű rendszerrel?

d. A tolási axiómája valószínűleg helytelen, mert a tárgy túl nehéz, így a pozíció azonos marad, ha a *Tol* operátort alkalmazzuk. Ez elágazási vagy értékelési feladatra ad példát? Javítsa ki a feladat leírását, hogy figyelembe vegye a nehéz tárgyakat.

11.5.

Magyarázza meg, hogy a hátrafelé keresésben a megelőző állapotok generálásának folyamata miért nem igényli, hogy a cselekvés negatív következményeinek megfelelő literálokat is szerepeltessük.

11.6.

Created by XMLmind XSL-FO Converter.

Magyarázza meg, miért vezet relaxált problémára a negatív következmények eldobása egy STRIPS probléma minden cselekvés sémájából.

11.7.

Vizsgálja meg a 3. fejezetben szereplő **kétirányú keresés** (**bidirectional search**) definícióját.

a. A kétirányú állapottér-keresés használata jó ötlet lenne tervkészítéshez?

b. És a részben rendezett tervek terében történő kétirányú keresés?

c. Tervezze meg a részben rendezett tervkészítő azon verzióját, melyben egy cselekvés akkor adható hozzá a tervhez, ha az előfeltételei elérhetők a tervben már szereplő cselekvések következményein keresztül. Magyarázza meg, hogyan kezeljük a konfliktusokat és a rendezési megkötéseket. Szükségszerűen azonos ez az algoritmus az állapottérben történő előrefelé kereséssel?

d. Vegyünk egy részben rendezett tervkészítőt, ami kombinálja a c) részfeladat módszerét azzal az általános módszerrel, ahol a nyitott feltételek eléréséhez cselekvéseket adunk a tervhez. Azonos lesz az így előálló algoritmus a b) részfeladattal?

11.8.

Készítse el a 11.2. ábrán szereplő probléma tervkészítési gráfjának 0., 1. és 2. szintjét.

11.9.

Bizonyítsa be az alábbi tervkészítő gráfokra vonatkozó állításokat:

- Az a literál, ami nem jelenik meg a gráf utolsó szintjén, nem érhető el.
- Egy soros gráfban szereplő literál szintköltsége nem lehet nagyobb, mint az őt elérő optimális terv költsége.

11.10.

Összevetettük az állapottérben előre- és hátrafelé kereső tervkészítőket a részben rendezett tervkészítőkkel úgy, hogy az utóbbit egy tervtérkeresőnek vettük. Magyarázza meg, hogyan tekinthető az előre- és hátrafelé történő állapottér-keresés tervtérkeresőnek, és mondja meg, mik a tervfinomító operátorok.

11.11.

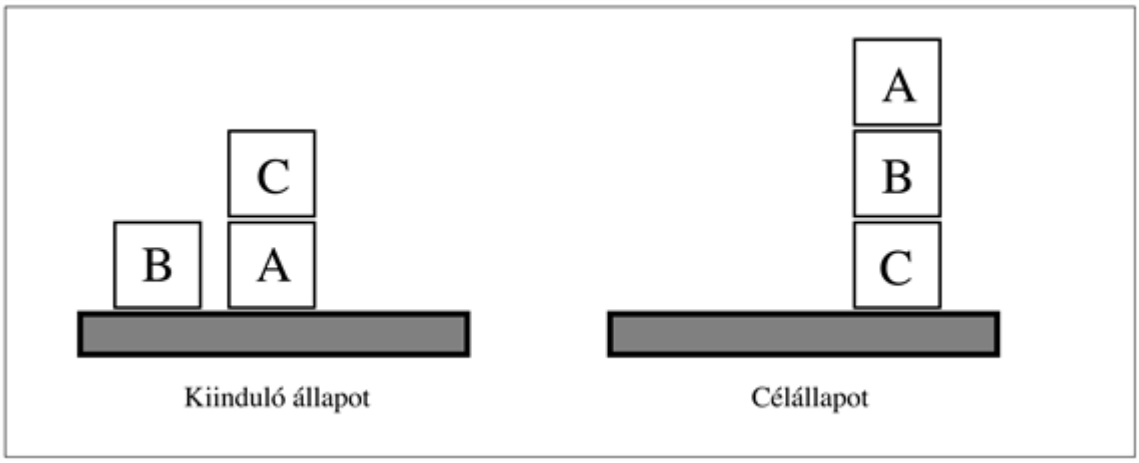
A 11.16. ábra a kockavilág problémát mutatja, ami Sussman-anomáliaként (Sussman anomaly) ismert. A problémát azért tekintették anomáliának, mert az 1970-es évek nem összefésülős tervkészítői nem tudták megoldani. Írjuk fel a probléma definícióját STRIPS alakban, és oldjuk meg, kézzel vagy egy tervkészítő programmal. A nemösszefésülő tervkészítő egy olyan tervkészítő, ami ha két G_1 és G_2 részcélt kap, akkor vagy egy G_1 elérésére szolgáló tervet fűz hozzá egy G_2 elérésére szolgáló tervhez, vagy fordítva. Magyarázza meg, hogy egy nem összefésülő tervkészítő miért nem tudja megoldani a kockavilág problémát.

11.12.

Vegyük a cipő és zokni felvételének problémáját, ahogy a 11.3. alfejezetben definiáltuk. Alkalmazzuk a GRAPHPLAN algoritmust erre a problémára, és mutassuk meg, hogyan áll elő a megoldás. Adjuk hozzá a kabát és a kalap felvételére szolgáló cselekvéseket. Adjuk meg azt a részben rendezett tervet, ami megoldás, és mutassuk meg, hogy ennek 180 különböző sorrendezése van. Mi a 180 különböző sorrendezés reprezentációjához szükséges különböző tervkészítő gráf megoldások minimális száma?

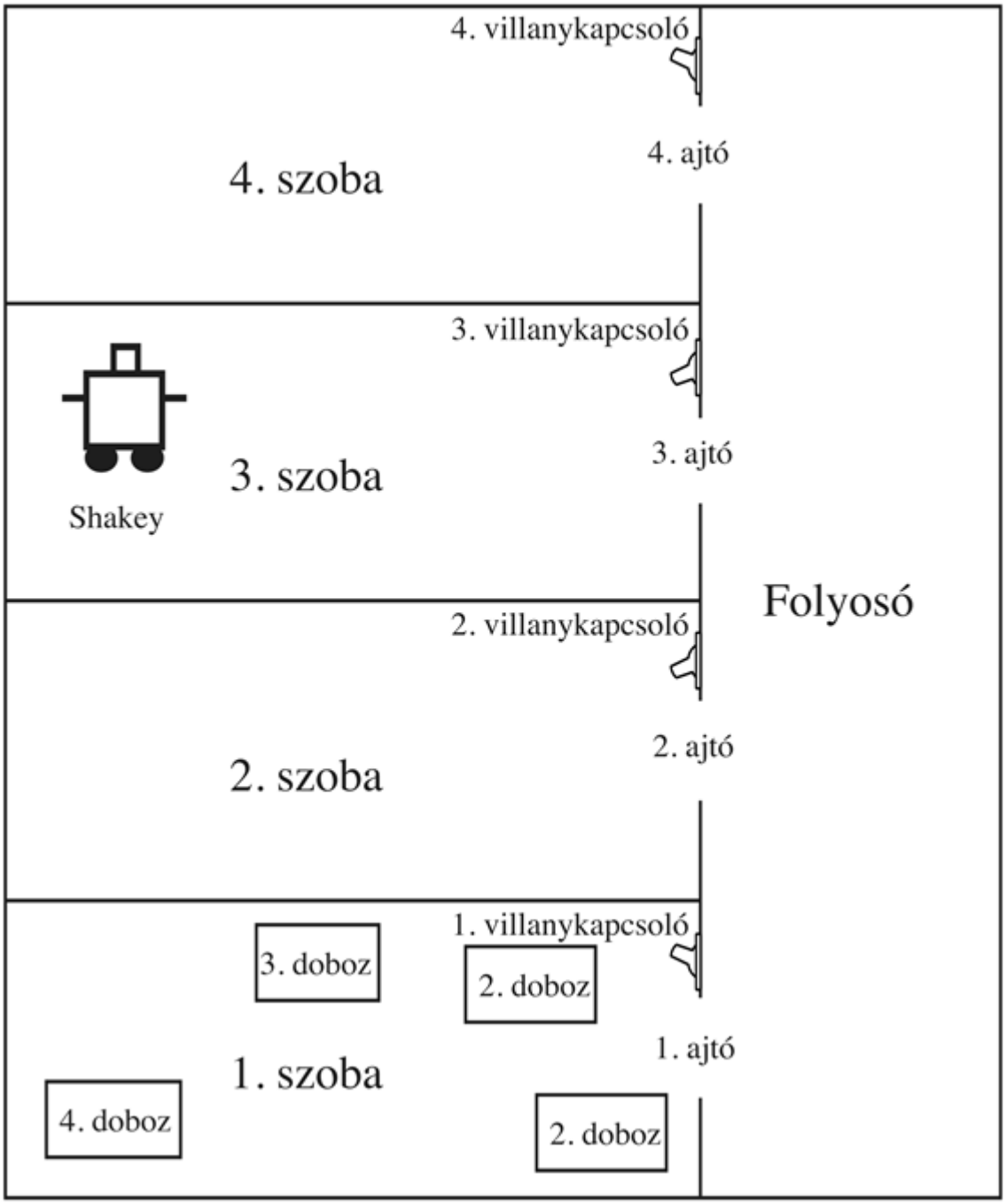
11.16. ábra - A Sussman-anomália kockavilág tervkészítési probléma

Created by XMLmind XSL-FO Converter.



11.17. ábra - Shakey világa. Shakey képes egy szobán belül mozogni, át tud menni a szobák közötti ajtókon, fel tud mászni tárgyakra, el tudja tolni a mozdítható tárgyakat, valamint a villanyt tudja kapcsolni.

Created by XMLmind XSL-FO Converter.



11.13.

Az eredeti `STRIPS` programot a Shakey robot irányítására készítették. A 11.17. ábra a Shakey világának egy változatát mutatja, melyben négy szoba sorakozik egy folyosó mentén, melyek mindegyikének van egy ajtaja és van benne egy villanykapcsoló.

Shakey világának cselekvései tartalmazzák az egyik helyről a másikra való mozgást, mozgatható tárgyak (mint dobozok) eltolását, szilárd testekre (mint dobozokra) való fel- és lemászást, és villanykapcsolók fel-, illetve lekapcsolását. A robot önmagában soha nem volt elég ügyes ahhoz, hogy egy dobozra másszon vagy egy kapcsolót átkapcsoljon, de a `STRIPS` tervkészítő képes a robot képességein túlmutató terveket találni és megadni. Shakey hat cselekvése a következő:

- A *Megy*(*x*, *y*), ami feltételezi, hogy Shakey *x*-ben van, valamint az *x* és *y* helyek ugyanazon szobában vannak. Konvenció alapján a két szoba közötti ajtó mindkét szobában benne van.

Created by XMLmind XSL-FO Converter.

- A *b* doboz eltolása *x* pozícióból ugyanazon szoba *y* pozíciójába: *Tol*(*p*, *x*, *y*). Szükségünk lesz a *Doboz* predikátumra és konstansokra a dobozokhoz.

- Dobozra felmászás: *Felmászik*(*b*); dobozról lemászás: *Lemászik*(*b*). Szükségünk lesz a *Rajta* predikátumra és a *Padló* konstansra.

- A villany felkapcsolása: *Felgyújt*(*s*); lekapcsolása: *Leolt*(*s*). A lámpa fel-, illetve lekapcsolásához Shakey-nek egy dobozon kell állnia a villanykapcsoló mellett.

A 11.17. ábra alapján írja fel Shakey hat cselekvését és a kiinduló állapotot `STRIPS` jelölésekkel. Készítsen tervet Shakey számára, hogy a $Doboz_2$ dobozt a $Szoba_2$ szobába mozgassa.

11.14.

Láttuk, hogy a tervkészítési gráfok, csak ítéletlogikai cselekvéseket képesek kezelni. Mi van, hogyha a tervkészítő gráfokat a célban változókat is tartalmazó problémákra kívánjuk alkalmazni, mint az $Ott(P_1, x) \wedge Ott(P_2, x)$, ahol *x* egy véges terület pozícióiból kerül ki? Hogyan tudna egy ilyen problémát átkódolni, hogy tervkészítő gráfokkal dolgozhasson? (*Segítség:* emlékezzen a részben rendezett tervkészítés *Vége* cselekvésére. Ehhez milyen előfeltételeknek kellene tartozni?)

11.15.

Egészen eddig feltételeztük, hogy a cselekvések csak a megfelelő helyzetekben kerülnek végrehajtásra. Nézzük, hogy az ítéletlogikai követő állapot axiómák (mint a 11.1 egyenlet axiómája) mit mondanak az olyan cselekvésekről, melyek előfeltételei nem teljesülnek.

a. Mutassa meg, hogy az axiómák azt jósolják, hogy semmi sem történik, amikor egy cselekvést olyan állapotban hajtunk végre, melyben az előfeltételei nem teljesülnek.

b. Vegyünk egy *p* tervet, ami tartalmazza a cél eléréséhez szükséges cselekvéseket és illegális cselekvéseket is. Igaz, hogy ebben az esetben

$$kiinduló\ állapot \wedge követő\ állapot\ axiómák \wedge p \models cél$$

c. Egy szituációkalkulusban elsőrendű követő állapot axiómák esetén (mint a 10. fejezetben) lehetséges bizonyítani, hogy egy illegális cselekvéseket tartalmazó terv elérje a célt?

11.16.

A repülőtér problémakörből adott példákkal magyarázza meg, hogy a szimbólumszétválasztás hogyan csökkenti az előfeltétel és a cselekvéskizárási axiómák méretét. Származtasson egy általános képletet, mely az axiómahalmazok méretét adja meg az időlépések száma, a cselekvéssémák száma, ezek aritása, valamint a tárgyak száma függvényében.

11.17.

A 11.15. ábrán szereplő `SATPLAN` algoritmusban a minden kielégíthetőségi algoritmus hívás egy további g^T célt eredményez, ahol a *T* értékkészlete 0 … T_{max}. Tegyük fel, hogy ehelyett a kielégíthetőségi algoritmust csak egyszer hívjuk meg a $g^0 \vee g^1 \vee \ldots \vee g^{T_{max}}$ céllal.

a. Ez mindig ad vissza tervet, amennyiben létezik egy T_{max} hosszúságú vagy annál rövidebb terv?

b. Eredményez ez a módszer új hibás megoldásokat?

c. Vizsgáljuk meg, hogy hogyan lehetne módosítani egy kielégíthetőségi algorimust, például a `WALKSAT` algoritmust, hogy az (amennyiben létezik) rövid megoldásokat találjon, amennyiben hasonló diszjunktív célt kap.

Created by XMLmind XSL-FO Converter.

12. fejezet - Tervkészítés és cselekvés a való világban

Ebben a fejezetben láthatjuk, hogy a kifejezőbb reprezentáció és az interaktívabb ágensarchitektúrák hogyan vezetnek a valós világban is használható tervkészítőkhöz.

Az előző fejezet a tervkészítés legalapvetőbb fogalmait, reprezentációit és algoritmusait vezette be. A való világ feladataihoz, mint a Hubble-űrteleszkóp megfigyeléseinek ütemezésére, gyárak működtetésére vagy hadműveletek logisztikai kezelésére jóval összetettebb tervkészítőket használnak. Ezek kiterjesztik az alaptípusokat mind a reprezentációs nyelv, mind pedig a tervkészítés és a környezet együttműködésének szempontjából. Ez a fejezet bemutatja hogyan. A 12.1. alfejezet az idő- és erőforráskorláttal rendelkező tervkészítést mutatja be. A 12.2. alfejezet az előre definiált résztervekhez történő tervkészítést írja le. A 12.3–12.6. alfejezet olyan ágensarchitektúrákat mutat be, melyeket a bizonytalan környezet kezelésére terveztek. A 12.7. alfejezet bemutatja, hogyan tervezzünk, amikor a környezet más ágenseket is tartalmaz.

1. Idő, ütemezés és erőforrások

A STRIPS reprezentáció azt írja le, hogy *mit* csinálnak a cselekvések, de mivel a reprezentáció a szituációkalkuluson alapul, nem tartalmazza, hogy *milyen hosszú* egy cselekvés, sem azt, hogy *mikor* következik be, kivéve azt, hogy egy másik cselekvés előtt vagy után jön. Néhány feladatkörben azt szeretnénk tudni, hogy a cselekvések mikor kezdődnek és végződnek. A teherszállítási problémakörben például, azt szeretnénk tudni, hogy egy adott csomagot hordozó repülőgép mikor érkezik, nem csak azt, hogy megérkezik, amikor a repülésnek vége.

Az alkalmazások egy általános családjának, az **ütemezési feladat**ok (**job shop scheduling**) családjának a lényege az idő. Az ilyen feladatok munkák elvégzését igénylik, melyek mindegyike cselekvések sorozatából áll, ahol minden cselekvés adott időtartamú és bizonyos erőforrásokat igényelhet. A probléma, hogy egy olyan ütemezést határozzunk meg, ami – az erőforráskorlátok figyelembevétele mellett – minimalizálja az összes feladat elvégzéséhez szükséges összidőt.

A 12.1. ábrán egy ütemezési feladatra láthatunk példát. Ez egy nagyon leegyszerűsített autó-összeszerelési feladat. Két munkánk van, a C_1 és a C_2 autó összeszerelése. Minden munka három cselekvésből áll: a motor beszerelése, a kerekek felszerelése, valamint az eredmény végső ellenőrzése. Először a motort kell berakni (mivel az első kerekek beszerelése megakadályozná, hogy hozzáférjünk a motortérhez), a vizsgálatot pedig értelemszerűen utoljára kell végrehajtani.

12.1. ábra - A két autó összeszerelését tartalmazó ütemezési feladat. A jelölésben az *Időtartam*(*d*) jelentése, hogy egy cselekvés *d* percet vesz igénybe. A *Motor*(E_1, C_1, 60) jelentése az, hogy az E_1 egy motor, ami a C_1 alvázba illeszthető és 60 perc alatt szerelhető be.

Created by XMLmind XSL-FO Converter.

$Kiindulás(Karosszéria(C_1) \wedge Karosszéria(C_2)$
$\wedge Motor(E_1, C_1, 30) \wedge Motor(E_2, C_2, 60)$
$\wedge Kerekek(W_1, C_1, 30) \wedge Kerekek(W_2, C_2, 15))$
$Cél(Kész(C_1) \wedge Kész(C_2))$

$Cselekvés(MotortBeszerel(e, c),$
ELŐFELTÉTEL: $Motor(e, c, d) \wedge Karosszéria(c) \wedge \neg MotorBenne(c)$
KÖVETKEZMÉNY: $MotorBenne(c) \wedge Időtartam(d))$
$Cselekvés(KerekeketFelszerel(w, c),$
ELŐFELTÉTEL: $Kerekek(w, c, d) \wedge Karosszéria(c) \wedge MotorBenne(c)$
KÖVETKEZMÉNY: $KerekekRajta(c) \wedge Időtartam(d))$
$Cselekvés(Átvizsgál(c),$
ELŐFELTÉTEL: $MotorBenne(c) \wedge KerekekRajta(c) \wedge Karosszéria(c)$
KÖVETKEZMÉNY: $Kész(c) \wedge Időtartam(10))$

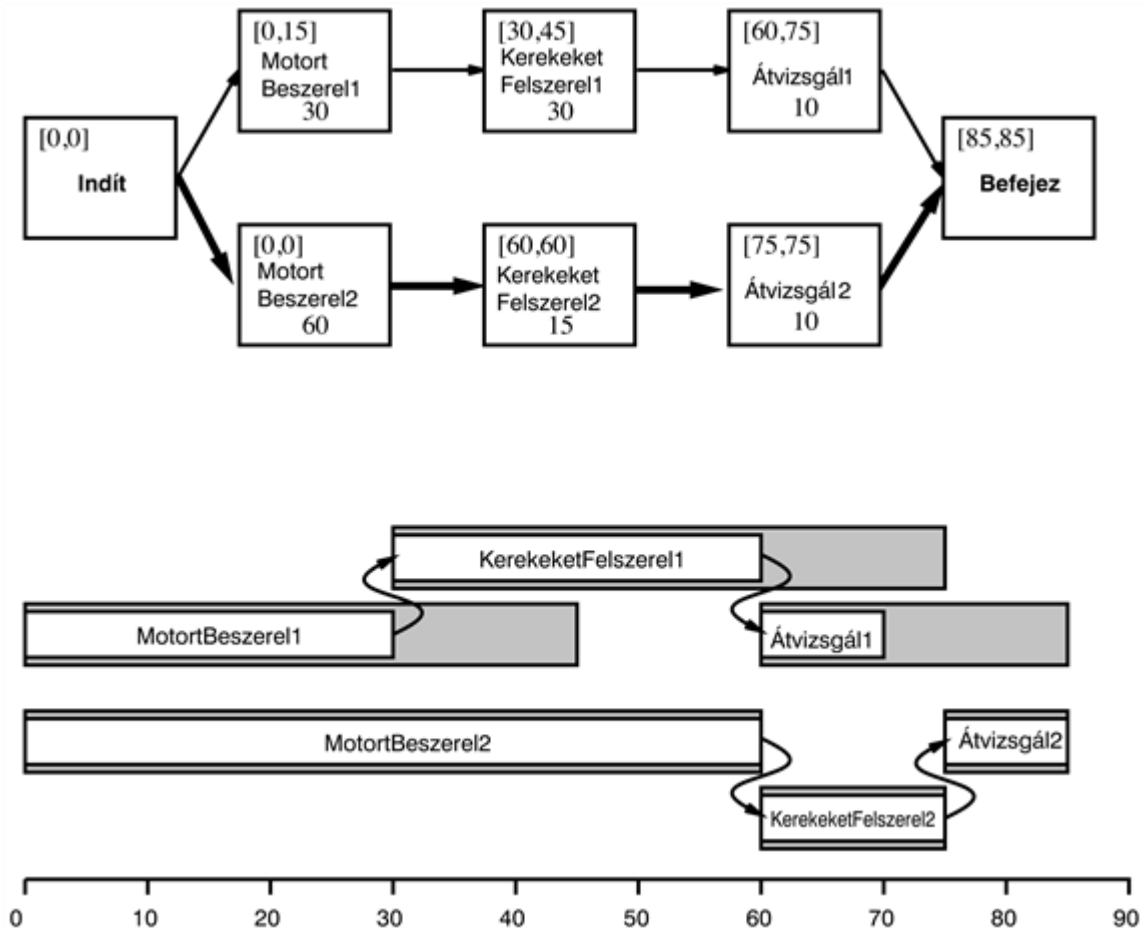
A 12.1. ábrán látható probléma bármely eddig látott tervkészítővel megoldható. A 12.2. ábra (ha a számokkal nem törődünk) a részben rendezett tervkészítő megoldását mutatja. Hogy az inkább egy ütemezési, mintsem egy tervkészítési feladat legyen, minden cselekvéshez meg kell határoznunk, hogy mikor kezdődjön és mikor végződjön. Ez azt jelenti, hogy figyelnünk kell minden cselekvés hosszára, csakúgy, mint a sorrendjére. Az *Időtartam*(*d*) jelölés egy cselekvés kapcsán (ahol *d* csak egy számmal helyettesíthető be) azt jelenti, hogy egy cselekvés elvégzéséhez *d* percre van szükség.

Ha adott a cselekvések egy részleges rendezése az időtartamokkal, ahogy a 12.2. ábra is mutatja, akkor a **kritikus útvonal módszer** (**critical path method** – **CPM**) felhasználható az egyes cselekvések lehetséges kezdési és befejezési időpontjainak meghatározására. Egy részben rendezett tervben található **útvonal**, cselekvések egy lineárisan rendezett sorozata, ami az *Indít*-ból indul és a *Befejez*-ben fejeződik be (például a 12.2. ábra részben rendezett tervében két út található).

A **kritikus útvonal** (**critical path**) az az út, amelynek a teljes ideje a leghosszabb; az útvonal „kritikus”, mert meghatározza a teljes terv hosszát. Más utak rövidítése nem rövidíti a terv egészét, de a kritikus útvonalon lévő bármely cselekvés kezdetének elhalasztása a teljes tervet lassítja. Az ábrán a kritikus útvonal vonalát vastag vonallal jelöljük. Hogy a teljes tervet minimális idő alatt hajtsuk végre, a kritikus útvonal cselekvéseit úgy kell végrehajtani, hogy köztük ne legyen késleltetés. A kritikus útvonalon kívül eső cselekvéseknek van valamennyi mozgásterük — egy időablak, amelyben végrehajthatók. Ezt az időablakot a kezdeti időpont lehető legkorábbi (*ES*) és a lehető legkésőbbi (*LS*) értékével definiáljuk. Az *LS* – *ES* mérőszámot, a **cselekvés mozgástere**nek (**slack**) nevezzük. A 12.2. ábrán láthatjuk, hogy a teljes terv 85 percet vesz igénybe, a kritikus útvonal minden cselekvésének a mozgástere 0 (ez mindig igaz), és a C_1 összeszerelésének minden cselekvése egy 15 perces ablakban indítható. A cselekvések *ES* és *LS* idői a probléma **ütemterv**ét (**schedule**) adják.

12.2. ábra - A 12.1. ábrán szereplő ütemezési probléma megoldása. Az ábra tetején a részben rendezett terv megoldása látható. Minden cselekvés hozza a téglalap alján látható, a legkorábbi és a legkésőbbi kezdési időkkel [*ES*, *LS*] a bal felső sarokban. A két szám között különbség a cselekvés mozgástere, a 0 mozgásterű cselekvések a kritikus útvonalon vannak, melyeket vastag vonallal jelölünk. Az ábra alján ugyanezen megoldás időrendjét láthatjuk. A szürke téglalapok azt az intervallumot jelölik, ami alatt a cselekvés végrehajtható, feltételezve, hogy a sorrendezési megkötéseket betartjuk. A szürke téglalapok nem felhasznált, nem kitöltött területei a mozgásteret jelölik.

Created by XMLmind XSL-FO Converter.



A következő kifejezések az *ES* és *LS* definícióiként szolgálnak, és egyúttal a kiszámításukra szolgáló dinamikus programozási algoritmust is körvonalazzák:

$$ES(Indít) = 0$$

$$ES(B) = \max_{A \prec B} ES(A) + Időtartam(A)$$

$$LS(Befejez) = ES(Befejez)$$

$$LS(A) = \min_{A \prec B} LS(B) - Időtartam(A)$$

Az ötlet az, hogy az *ES*(*Indít*) értékének 0-ra állításával indítunk. Ezután, amint elérünk egy olyan *B* cselekvést, hogy minden cselekvés, ami közvetlenül *B* előtt jön, már rendelkezik *ES* értékekkel, az *ES*(*B*) értéket a közvetlenül megelőző cselekvések közül a legkorábbi befejezési idő maximumára állítjuk be, ahol a legkorábbi befejezési időt úgy definiáljuk, mint a cselekvés kezdési időpontja plusz a cselekvés időtartama. A folyamat addig folytatódik, amíg minden cselekvéshez egy *ES* értéket rendelünk. Az *LS* értékeket hasonlóképpen számítjuk ki a *Befejez* cselekvéstől visszafelé haladva. A részleteket egy későbbi feladat tartalmazza.

A kritikus útvonal algoritmus komplexitása csak $O(Nb)$, ahol N a cselekvések száma és b az egy cselekvésbe be-, illetve kimenő maximális elágazások száma. (Ennek megértéséhez vegyük észre, hogy az *LS* és *ES* számításokat minden cselekvésre csak egyszer hajtjuk végre, és minden számítás legfeljebb b másik cselekvésen megy végig.) Ezért, *ha adott a cselekvések egy részben rendezése,* a minimális időtartamú ütemezés megtalálása elég könnyedén elvégezhető.

1.1. Ütemezés erőforráskorlátokkal

A valós ütemezési problémákat az **erőforrás**okra (**resource**s) vonatkozó korlátozások jelenléte megbonyolítja. Például egy motornak az autóba szereléséhez egy motoremelőre van szükség. Ha csak egyetlen emelőnk van, akkor nem tudjuk egyszerre beszerelni az E_1 motort, a C_1 és az E_2 motort a C_2 autóba, így a 12.2. ábrán bemutatott ütemezés végrehajthatatlan. A motoremelő példa egy **újrahasznosítható erőforrás**ra (**reusable**

Created by XMLmind XSL-FO Converter.

resource). Az erőforrás foglalt, amíg a cselekvés tart, de újra használhatóvá válik annak befejezése után. Vegyük észre, hogy az újrahasznosítható erőforrásokat nem tudjuk a hagyományos előfeltétel és következmény alakú cselekvésleírásokkal kezelni, mert a rendelkezésre álló erőforrás-mennyiség nem változik, miután egy cselekvést lezártunk.[1] Ezért kiterjesztjük a leírásunkat, hogy egy ERŐFORRÁS: $R(k)$-t is tartalmazzon, amelynek jelentése, hogy k egység R típusú erőforrásra van szükség a cselekvéshez. Az erőforrás-szükséglet mind előfeltétel (a cselekvés nem hajtható végre, ha az erőforrás nem érhető el), mind pedig *átmeneti* következmény, abban az értelemben, hogy az R erőforrás rendelkezésre állása k-val csökken a cselekvés időtartama alatt. A 12.3. ábra mutatja, hogyan lehet a motorbeszerelés problémáját kiterjeszteni, hogy az három erőforrást is tartalmazzon: egy motoremelőt a motorok beszereléséhez, egy kerékállomást a kerekek felszereléséhez és két vizsgálóbiztost. A 12.4. ábra a legrövidebb szerelési idejű megoldást mutatja, ez 115 percet igényel. Ez hosszabb, mint az erőforrás-megkötések nélküli 85 perces ütemezés. Vegyük észre, hogy nincs olyan időpillanat, amikor mindkét vizsgálóbiztosra szükség van, ezért az egyik vizsgálóbiztost azonnal egy termelékenyebb posztra helyezhetjük.

Az erőforrások numerikus mennyiségekként történő leírása mint a *Vizsgálóbiztosok*(2), a megnevezett entitások használata helyett, mint a *Vizsgálóbiztos*(I_1) és a *Vizsgálóbiztos*(I_2), jó példa a nagyon általánosan használt **aggregáció**s (**aggregation**) technikára. Az aggregáció központi ötlete, hogy egyedi objektumokat csoportosítsunk mennyiségekké, amikor maguk az objektumok nem megkülönböztethetők a cél szempontjából. Az összeszerelési problémánkban nem számít, hogy *melyik* vizsgálóbiztos vizsgálja az autót, ezért nincs szükség különbségtételre. (Ugyanez az ötlet működik a 3.9. feladat misszionáriusok és kannibálok problémájára.) Az aggregáció a komplexitás csökkentéséhez elengedhetetlen. Vizsgáljuk meg, mi történik, ha egy ütemezés 10 konkurrens *Vizsgálat* cselekvést tartalmaz, de csak 9 vizsgálóbiztos áll rendelkezésre. Ha a vizsgálóbiztosokat mint mennyiségeket reprezentáljuk, a bukást az algoritmus azonnal detektálja, és visszalép, hogy egy másik ütemezéssel próbálkozzon. Ha a vizsgálóbiztosokat önálló személyekként reprezentáljuk, akkor az algoritmus feleslegesen mind a 10! vizsgálóbiztos-kiosztás kipróbálásához visszalép.

12.3. ábra - Két autó összeszerelésének ütemezési feladata erőforrásokkal. Egy összeszerelő állomás, egy kerékszerelő-állomás és két vizsgálóbiztos az összes elérhető erőforrás. Az ERŐFORRÁS:*r* jelölés jelentése, hogy az *r* erőforrást egy cselekvés végrehajtása alatt használjuk, de a cselekvés befejezése után újra szabad.

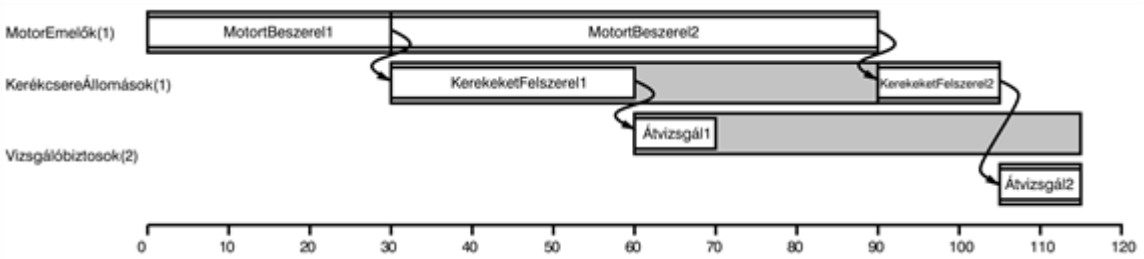


12.4. ábra - A 12.3. ábra erőforrásokat is tartalmazó ütemezési feladatának megoldása. Az ábra bal széle az erőforrásokat sorolja fel, a cselekvések az általuk használt erőforrásokkal egy sorban jelennek meg. Attól függően, hogy melyik összeszerelés

[1] Ezzel szemben a **fogyóeszköz-erőforrás**ok (**consumable resource**s), mint az autó-összeszereléshez használt csavarok, az eredeti keretrendszeren belül kezelhetők; lásd 12.2. feladat.

Created by XMLmind XSL-FO Converter.

használja a motorszerelő-állomást először, két lehetséges ütemezés létezik. Mi az optimális megoldást ábrázoltuk, ami 115 percet vesz igénybe.



Az előnyök ellenére, az erőforrás-megkötések nagyon megbonyolítják az ütemezési feladatokat azáltal, hogy a cselekvések közé további kölcsönhatásokat vezetnek be. Habár a megkötések nélküli ütemezés a kritikus útvonal módszer felhasználásával könnyű, a legkorábbi befejezési idejű erőforrás-megkötéseket tartalmazó ütemezés megtalálása NP-nehéz. Ez a komplexitás gyakran mind a gyakorlatban, mind az elméletben látható. Az 1963-ban közzétett nagy kihívást jelentő feladat – egy optimális ütemezés megtalálása egy 10 gépet, 10 munkát és 100 cselekvést tartalmazó problémához – 23 évig megoldatlan maradt (Lawler és társai, 1993). Sok megközelítést kipróbáltak, köztük az elágazásos megkötést, a szimulált lehűtést, a tabu keresést, a kényszerkielégítést és számos más, a II. részben szereplő technikát. A legegyszerűbb, de népszerű heurisztika a **minimális tartalék** (**minimum slack**) algoritmus. Ez a cselekvéseket mohó jelleggel ütemezi. Minden iterációban megnézi azokat a cselekvéseket, amelyeknek az összes előzményét már ütemeztük, és azt ütemezi, amelyiknek a legkevesebb tartalék ideje van a legkorábbi lehetséges kezdéshez. Ezután frissíti az *ES* és *LS* időértékeket minden egyes érintett cselekvésre, és ezt ismétli.

A heurisztika ugyanazon az ötleten alapul, mint a legjobban korlátozott változó heurisztika a kényszerkielégítésnél. Ez a gyakorlatban gyakran jól működik, de a mi összeszerelési problémánkra egy 130 perces megoldást ad, nem pedig a 12.4. ábrán bemutatott 115 perceset.

Az ebben a fejezetben bemutatott megközelítés a „tervezzünk először, ütemezzünk később” megközelítés: azaz a teljes problémát egy *tervkészítési* fázisra és egy *ütemezési* fázisra bontjuk. A tervezési fázisban a cselekvéseket részben rendezzük, hogy elérjük a probléma céljait, az ütemezési fázisban pedig az időinformációkat illesztjük a tervhez, ezzel biztosítva, hogy a terv kielégítse az erőforrás- és a határidőkorlátokat. Ez a megközelítés nagyon gyakori a valós idejű gyártási és logisztikai problémákban, ahol a tervkészítési fázist leggyakrabban emberi szakértők végzik. Ellenben, amikor sok erőforrás-megkötés van, adódhat, hogy néhány érvényes terv sokkal jobb ütemezéshez vezet, mint mások. Ebben az esetben, a tervkészítési és ütemezési fázist célszerű *integrálni* úgy, hogy a részben rendezett tervkészítés során figyelembe vesszük a cselekvések időtartamait és átfedéseit. A 11. fejezetben bemutatott számos tervkészítő algoritmus kiterjeszthető úgy, hogy ezt az információt is kezelje. Például a részben rendezett tervkészítők az okozati kapcsolatok konfliktusainak detektálásához hasonlóan az erőforráskorlát megsértéseit is detektálni tudják. A heurisztikák módosíthatók, hogy a cselekvések teljes költségén túl, a tervek teljes végrehajtási idejét is becsüljék. Ez napjainkban a kutatás egy aktív területe.

2. Hierarchikus feladatháló tervkészítés

Az egyik legelterjedtebb módszer a komplexitás kezelésére a **hierarchikus dekompozíció** (**hierarchical decomposition**). Az összetett szoftvereket szubrutinok vagy objektum osztályok hierarchiájából építik fel, a hadseregek különböző egységek hierarchiái, a kormányok és cégek igazgatóságokból, osztályokból és alosztályokból állnak. A hierarchikus struktúra fő előnye, hogy a hierarchia minden szintjén egy számítási feladat, egy hadművelet vagy egy adminisztratív feladat az eggyel alatta lévő szint néhány cselekvésére épül, így e cselekvések megfelelő elrendezése a magasabb szinten lévő feladat megoldásához *kis* számítási költséggel jár. Másrészről a nem hierarchikus módszerek a feladatot *nagy*számú, független cselekvésre bontják fel, ami nagy léptékű feladatok esetén egyáltalán nem praktikus. A legjobb esetben – amikor a magas szintű megoldásokhoz mindig kielégítő alacsony szintű megvalósítás tartozik – a hierarchikus megoldások az exponenciális idejű tervkészítő algoritmusokkal szemben lineáris időre vezetnek.

Ez az alfejezet a **hierarchikus feladatháló**kon vagy **HFH**-kon (**hierarchical task networks** – **HTN**) alapuló tervkészítési módszert mutatja be. A megközelítésünk ötvözi a részben rendezett tervkészítés (11.3. alfejezet) alapötleteit, illetve a „HFH-tervkészítés” területét. A HFH-tervkészítésben a kiinduló problémát, amely a feladatot írja le, a végrehajtandó feladat egy nagyon magas szintű leírásának tekintjük, például: építsünk egy házat. A terveket **cselekvésdekompozíció**kkal (**action decompositions**) finomítjuk. Minden

Created by XMLmind XSL-FO Converter.

cselekvésdekompozíció a magas szintű cselekvést alacsonyabb szintű cselekvések részben rendezett halmazára bontja. A cselekvésdekompozíció ezért a cselekvések megvalósítására vonatkozó ismereteket testesíti meg. Például egy ház felépítése az engedélyek megszerzésére, a kivitelező megbízására, az építkezés elvégzésére és a kivitelező kifizetésére redukálható. (A 12.5. ábra egy ilyen dekompozícióra mutat példát.) A folyamat addig folytatódik, amíg csak az **egyszerű cselekvés**ek (**primitive action**s) maradnak a tervben. Az egyszerű cselekvések tipikusan azok a cselekvések, amelyeket az ágens automatikusan végre tud hajtani. Egy általános kivitelezőre a „kertépítés" egy egyszerű cselekvés lehet, mert egyszerűen csak egy kertépítő bevonását jelenti. Egy kertépítő számára azonban az olyan cselekvések tekinthetők egyszerűnek, mint az „ültessen rododendront ide".

A „tiszta" HFH-tervkészítésben a terveket *csak* egymást követő cselekvésdekompozíciókkal állítjuk elő. A HFH ezért a tervkészítést a cselekvésleírások *konkretizálásának* tekinti, szemben az üres cselekvésből kiinduló cselekvésleírás *elkészítésének* folyamatával (ami az állapottér-keresésre és a részben rendezett tervkészítésre is igaz). Végezetül kiderül, hogy minden `STRIPS` cselekvésleírás egy cselekvésdekompozícióra írható át (lásd 12.6. feladat), és a részben rendezett tervkészítés a tiszta HFH-tervkészítés egy speciális esetének tekinthető. Bizonyos feladatokra azonban – különösen „szokatlan" konjunktív célokra — a tiszta HFH-nézőpont eléggé természetellenes, így egy *hibrid* megközelítést részesítünk előnyben, ahol a cselekvésdekompozíciókat mint a részben rendezett tervkészítés tervfinomításait használjuk fel, a nyitott előfeltételek teljesítése és az ütközés feloldásra szolgáló rendezési megkötések hozzáadása mellett. (Annak, hogy a HFH-tervkészítést a részben rendezett tervkészítés kiterjesztésének tekintjük, további előnye, hogy egy teljesen új jelölésrendszer bevezetése helyett ugyanaz használható.) Kezdésként a cselekvések dekomponálását mutatjuk be részletesebben, majd elmagyarázzuk, hogy a részben rendezett tervkészítést hogyan kell módosítani a dekompozíciók kezeléséhez. Végezetül a teljesség, a komplexitás és a használhatóság kérdéseit tárgyaljuk.

2.1. A cselekvésdekompozíciók reprezentációja

A cselekvésdekompozíciós módszerek általános leírásait egy **tervkönyvtár**ban (**plan libary**) tároljuk, ahonnan kinyerhetők és a készülő terv igényeinek megfelelően felhasználhatók. Minden módszer egy *Dekomponál*(*a*, *d*) formájú kifejezés, amelynek jelentése, hogy egy *a* cselekvés dekomponálható a *d* tervbe, mely egy – a 11.3. alfejezetben leírtaknak megfelelő – részben rendezett tervként van megadva.

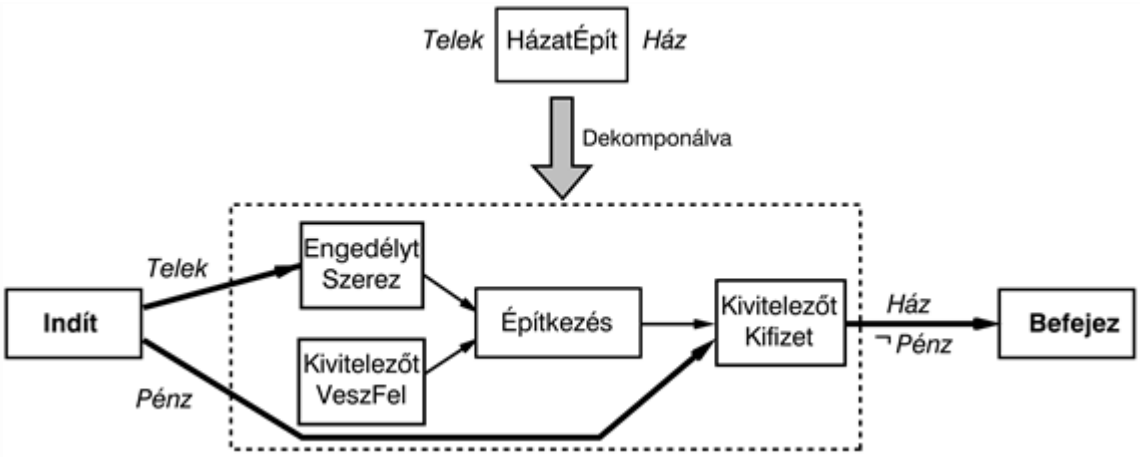
A házépítés egy szép, konkrét példa, ezért ezt használjuk fel a cselekvésdekompozíció bemutatására. A 12.5. ábra a *HázatÉpít* cselekvés egy lehetséges dekompozícióját mutatja négy alacsonyabb szintű cselekvésre. A 12.6. ábra néhány cselekvés leírását tartalmazza erre a feladatkörre, valamint a házépítés dekompozícióját, ahogyan az a tervkönyvtárban megjelenne. A könyvtárban más lehetséges dekompozíciók is szerepelhetnek.

A dekompozíció *Indít* cselekvése szolgáltatja a terv cselekvéseinek összes olyan előfeltételét, melyet más cselekvés nem állít elő. Ezeket **külső előfeltétel**eknek (**external precondition**s) nevezzük. Példánkban a dekompozíció külső előfeltételei a *Telek* és a *Pénz*. Hasonlóképp a *Befejez* előfeltételei a **külső következmény**ek (**external effect**s). Ezek a terv cselekvéseinek összes olyan következményét jelentik, melyeket más cselekvések nem negálnak. Példánkban a *HázatÉpít* külső következményei a *Ház* és a ¬*Pénz*. Néhány HFH-tervkészítő szintén különbséget tesz az **elsődleges következmény**ek (**primary effect**s), mint a *Ház*, és a **másodlagos következmény**ek (**secondary effect**s), mint a ¬*Pénz* között. Mivel mindkét típusú következmény felhasználása ütközésekhez vezethet más cselekvésekkel, csak az elsődleges következményeket használhatjuk a célok elérésére, ami nagymértékben csökkenti a keresési teret.[2]

12.5. ábra - A *HázatÉpít* cselekvés egy lehetséges dekompozíciója

[2] Ez néhány nem várt terv előállítását is megakadályozhatja. Például egy csődeljárás előtt álló személy minden ingó vagyonát eltüntetheti (a ¬*Pénz* elérésével) egy ház megvásárlásával vagy megépítésével. Ez a terv hasznos, mert az aktuális törvény kizárja az elsődleges lakóhely hitelezők általi lefoglalását (legalábbis az USA-ban – *a szerk.*).

Created by XMLmind XSL-FO Converter.



A dekompozíciónak a cselekvés egy *korrekt* megvalósításának kell lennie. A *d* terv egy *a* cselekvést korrekten valósít meg, ha *d* egy teljes és konzisztens részben rendezett terv arra a problémára, ahol *a* előfeltételeiből *a* következményeit kell elérnünk. Ha a dekompozíció egy helyesen működő részben rendezett tervkészítő futásának eredménye, akkor nyilvánvalóan korrekt.

Bármely magas szintű cselekvéshez a tervkönyvtár számos dekompozíciót tartalmazhat, például a *HázatÉpít*-nek lehet egy másik dekompozíciója, amely a folyamatot úgy írja le, hogy a házat az ágens saját kezűleg építi kövekből és malterből. Minden dekompozíciónak egy korrekt tervnek kell lennie, de a magas szintű cselekvésleírásban szereplőkön túl további előfeltételeket és következményeket is tartalmazhat. Például a *HázatÉpít* 12.5. ábrán látható dekompozíciója a *Telek* mellett a *Pénz*-t is megköveteli, és következménye a ¬*Pénz*. Másrészről a saját kezű építés nem igényel pénzt, de szükséges hozzá egy felhasználásra váró *Kő* és *Malter* készlet, valamint eredménye lehet egy *FájósHát*.

Mivel egy magas szintű cselekvésnek, mint amilyen a *HázatÉpít*, számos lehetséges dekompozíciója lehet, elkerülhetetlen, hogy a `STRIPS` cselekvésleírása elrejtse ezen dekompozíciók néhány előfeltételét vagy következményét. A magas szintű cselekvés előfeltételeit a dekompozícióiban szereplő külső előfeltételek *metszete*, míg a következményit a dekompozíciók külső következményeinek metszete adja. Másképpen, a magas szintű előfeltételek és következmények garantáltan részhalmazai minden egyszerű implementáció valós előfeltételeinek és következményeinek.

12.6. ábra - A házépítési probléma cselekvéseinek leírása és a *HázatÉpít* cselekvés részletes dekompozíciója. A leírások a pénzzel kapcsolatban egy egyszerűsített, az építőkkel kapcsolatban egy optimista nézetet alkalmaznak.

Cselekvés(*TelketVásárol*, ELŐFELTÉTEL: *Pénz* KÖVETKEZMÉNY: *Telek* ∧ ¬ *Pénz*)
Cselekvés(*KölcsöntVeszFel*, ELŐFELTÉTEL: *Hitel*, KÖVETKEZMÉNY: *Pénz* ∧ *Adósság*)
Cselekvés(*HázatÉpít*, ELŐFELTÉTEL: *Telek*, KÖVETKEZMÉNY: *Ház*)

Cselekvés(*EngedélytSzerez*, ELŐFELTÉTEL: *Telek*, KÖVETKEZMÉNY: *Engedély*)
Cselekvés(*KivitelezőtVeszFel*, ELŐFELTÉTEL: *Szerződés*)
Cselekvés(*Építkezés*, ELŐFELTÉTEL: *Engedély* ∧ *Szerződés*,
KÖVETKEZMÉNY: *HázFelépítve* ∧ ¬ *Engedély*)
Cselekvés(*KivitelezőtKifizet*, ELŐFELTÉTEL: *Pénz* ∧ *HázFelépítve*,
KÖVETKEZMÉNY: ¬ *Pénz* ∧ *Ház* ∧ ¬ *Szerződés*)
Dekomponál(*HázatÉpít*,
Terv(LÉPÉSEK:{S_1: *EngedélytSzerez*, S_2: *KivitelezőtVeszFel*,
S_3: *Építkezés*, S_4: *KivitelezőtKifizet*}
RENDEZÉSEK:{*Indít* ≺ S_1 ≺ S_2 ≺ S_3 ≺ S_4 ≺ *Befejez*, *Indít* ≺ S_2 ≺ S_3},
KAPCSOLATOK:{*Indít* $\xrightarrow{Telek}$ S_1, *Indít* $\xrightarrow{Pénz}$ S_4,
S_1 $\xrightarrow{Engedély}$ S_3, S_2 $\xrightarrow{Szerződés}$ S_3, S_3 $\xrightarrow{HázFelépítve}$ S_4,
S_4 $\xrightarrow{Ház}$ *Befejez*, S_4 $\xrightarrow{¬Pénz}$ *Befejez*}))

Created by XMLmind XSL-FO Converter.

Az információ elrejtésének két másik formáját is meg kell említenünk. Először, a magas szintű leírás teljes mértékben figyelmen kívül hagyja a dekompozíciók **összes belső következmény**ét (**internal effect**s). Például a *HázatÉpít* általunk javasolt dekompozíciójának ideiglenes belső következménye az *Engedély* és a *Szerződés*.[3] Másodszor, a magas szintű leírás nem specifikálja a cselekvésen belüli időintervallumot, ami alatt a magas szintű előfeltételeknek és következményeknek teljesülniük kell. Például a *Telek* előfeltételnek csak addig kell teljesülnie (a közelítő modellünkben), amíg az *EngedélytSzerez* nem hajtódik végre, és a *Ház* csak a *KivitelezőtKifizet* végrehajtása után igaz.

Az információ elrejtésének ez a módja szükségszerű, ha a hierachikus tervezés célja a komplexitás csökkentése. Szükséges, hogy a magas szintű cselekvésekről anélkül dönthessünk, hogy a számtalan implementációs részlet miatt aggódnánk. Ennek azonban ára van. Konfliktusok léphetnek fel, például egy magas szintű cselekvés belső feltételei, valamint egy másik magas szintű cselekvés belső cselekvései között, miközben a magas szintű leírások alapján nincs mód ezek detektálására. Ennek a problémának komoly hatásai vannak a HFH-tervkészítő algoritmusokra. Dióhéjban, míg a tervkészítő algoritmus az atomi cselekvéseket, mint pontszerű eseményeket kezelheti, a magas szintű cselekvéseknek időbeli kiterjedése van, ami alatt minden mehet tovább.

2.2. A tervkészítő módosítása a dekompozíciók kezeléséhez

Most megmutatjuk, hogyan módosítható a részben rendezett tervkészítő, hogy egy HFH-tervkészítőt is tartalmazzon. Ehhez a részben rendezett tervkészítő állapotátmenet-függvényét (lásd 3.1. szakasz - Példa a részben rendezett tervkészítésre részben) módosítsuk úgy, hogy lehetővé tegye a T aktuális részleges terven a dekompozíciós módszerek alkalmazását. Az új következő terveket úgy alakítjuk ki, hogy először kiválasztunk néhány nem-atomi a' cselekvést a T-ből, majd a tervkönyvtár minden *Dekomponál*(a, d) metódusára, amelyben a θ behelyettesítéssel az a és a' egyenlővé válik, az a'-t behelyettesítjük a d' = SUBST (θ, d)-vel.

A 12.7. ábrán egy példa látható. Az ábra tetején egy ház megszerzésére vezető T terv található. Az a' = *HázatÉpít* magas szintű cselekvést választottuk a dekompozícióra. A 12.5. ábráról a d dekompozíciót választottuk, és a *HázatÉpít* cselekvést ezzel helyettesítettük. A dekompozíciós lépés által létrehozott *Pénz* nyitott feltétel teljesítésére a *KölcsöntVeszFel* járulékos lépést vezettük be. Egy cselekvés helyettesítése annak dekompozíciójával egy kicsit hasonlít a szervátültetéses műtétekhez. Az új résztervet ki kell bontanunk a csomagolásából (az *Indít* és *Befejez* lépések közül), be kell illesztenünk és mindent megfelelően le kell zárnunk. Ezt többféleképpen is megtehetjük. Hogy pontosabbak legyünk, minden lehetséges d' dekompozícióhoz az alábbiakat kell megtennünk:

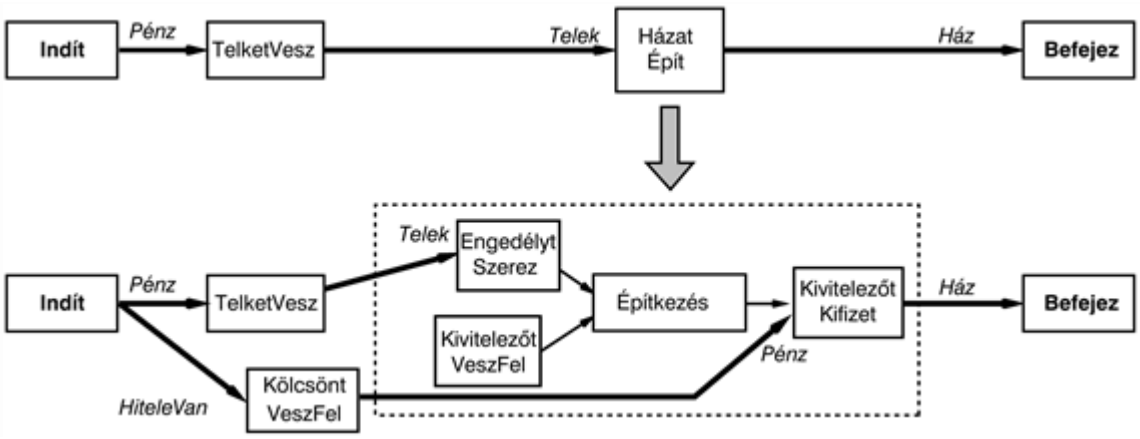
1. Először is az a' cselekvést el kell távolítani a T tervből. Ezután a d' minden s lépésére ki kell választanunk egy cselekvést, amely kitölti az s szerepét, és hozzá kell adnunk a tervhez. Ez lehet az s egy új példányosítása vagy egy meglévő s' lépés a T-ből, ami s-sel azonos. A *Borkészítés* cselekvésdekompozíciója például megkövetelheti a *TelketVesz* cselekvést, de várhatóan használhatjuk ugyanazt a *TelketVesz* cselekvést, mely már szerepel a tervben. Ezt **részfeladat-megosztás**nak (**subtask sharing**) nevezzük.

A 12.7. ábrán nincsenek megosztási lehetőségek, ezért új cselekvéspéldányokat hoztunk létre. Ha egy cselekvést kiválasztottunk, d' minden belső előfeltételét átmásoljuk. Például az *EngedélytSzerez* cselekvés, az építkezés elé rendelt, és létezik egy okozati kapcsolat ezen lépések között, mely szerint az *Engedély* előfeltétele az *Építkezés*-nek. Ez lezárja az a' helyettesítését a $d\theta$ példányosításával.

12.7. ábra - Egy magas szintű cselekvés dekompozíciója egy létező tervben. A *HázatÉpít* cselekvést a 12.5. ábra dekompozíciójával helyettesítjük. A *Telek* külső előfeltételt a már létező *TelketVesz*-ből kiinduló okozati kapcsolat biztosítja. A *Pénz* külső előfeltétel nyitott marad a dekompozíciós lépés után, ezért a *KölcsöntVeszFel* cselekvést szúrjuk be.

[3] Az Építkezés negálja az Engedély-t, egyébként ugyanazon engedély több ház megépítéséhez is használható lenne. Sajnos az Építkezés nem zárja le a Szerződés-t, mert előbb a KivitelezőtKifizet-et kell végrehajtanunk.

Created by XMLmind XSL-FO Converter.



2. A következő lépés, hogy az eredeti tervben szereplő *a*' sorrendezési megkötéseit megfelelően átvezessük a *d*' lépéseihez. Először vegyük $T\ B \prec a$' alakú sorrendezési megkötéseit. Hogyan kellene *B*-t rendezni a *d*' lépéseinek megfelelően? A legkézenfekvőbb megoldás, hogy *B* az összes *d*'-ben szereplő lépés előtt jöjjön, amit úgy érhetünk el, hogy *d*' minden $Indít \prec s$ alakú megkötését a $B \prec s$ megkötéssel helyettesítjük. Másrészről ez a megközelítés túlzottan szigorú lehet! Például a *TelketVesz* a *HázatÉpít* előtt kell következzen, de nincsen szükség arra, hogy a *TelketVesz* a *KivitelezőtVeszFel* előtt jöjjön a kiterjesztett tervben. Egy túlzottan szigorú rendezés felállítása lehetetlenné teheti néhány megoldás megtalálását. Ezért minden rendezési megkötésre a legjobb megoldás, hogy rögzítsük a megkötés *okát*. Ha ezután egy magas szintű cselekvést kibontunk, az új rendezési megkötések a lehető leglazábbra vehetők az eredeti megkötés okával összhangban. Ugyanezek a megfontolások alkalmazhatók, amikor az $a' \prec C$ alakú megkötéseket helyettesítjük.

3. Az utolsó lépés, hogy az okozati kapcsolatokat átvezessük. Ha a $B \xrightarrow{p} a'$ az eredeti terv egy okozati kapcsolata volt, helyettesítsük azt egy okozati kapcsolathalmazzal, amely *B*-ből *d*' összes olyan lépéséhez vezet, amelynek előfeltétele a *d* dekompozíció *Indít* lépése által előállított *p* (például *d*' összes lépése, melyekre *p* egy külső előfeltétel). A példában a $TelketVesz \xrightarrow{Telek} HázatÉpít$ okozati kapcsolatot a $TelketVesz \xrightarrow{Telek} Engedély$ kapcsolattal helyettesítjük. (A dekompozícióban szereplő *KivitelezőtKifizet*, *Pénz* előfeltétele nyitott feltétellé válik, mert az eredeti tervben nincs olyan cselekvés, ami a *Pénz*-t biztosítja a *HázatÉpít*-hez.) Hasonlóképpen a terv összes $a' \xrightarrow{p} C$ okozati kapcsolatát helyettesítsük *d*' bármely, a *d* dekompozíció *Vége* lépéséhez *p*-t szolgáltató lépésből *C*-be mutató okozati kapcsolathalmazzal (például *d*' *p*-t külső következményként tartalmazó lépéséből). Példánkban a $HázatÉpít \xrightarrow{Ház} Befejez$ kapcsolatot a $KivitelezőtKifizet \xrightarrow{Ház} Befejez$ kapcsolattal helyettesítjük.

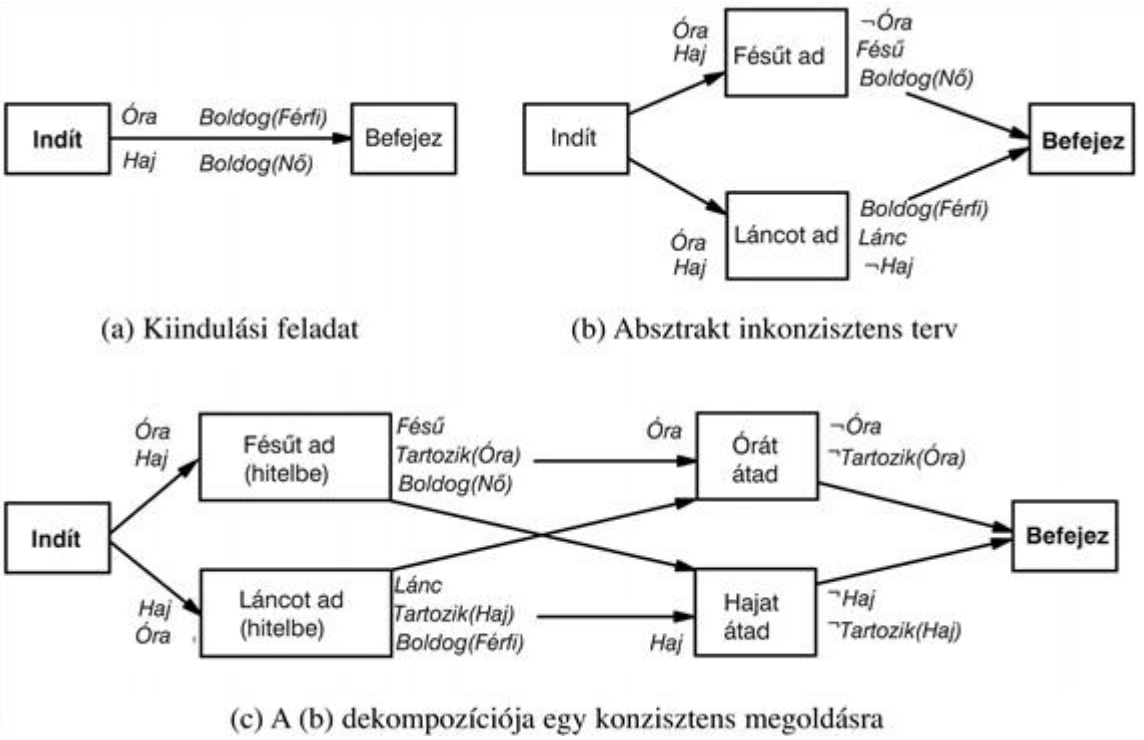
Ez lezárja a részben rendezett tervkészítőknél alkalmazott, a dekompozíciók elkészítéséhez szükséges kiterjesztéseket.[4]

A részben rendezett tervkészítő algoritmushoz további módosítások szükségesek, mert a magas szintű cselekvések *elrejtik az információt* a végső elemi megvalósításukról. Nevezetesen az eredeti részben rendezett tervkészítő algoritmus hibával lép vissza, ha az aktuális terv feloldhatatlan ütközést tartalmaz, vagyis ha egy cselekvés ütközik egy okozati kapcsolattal, de nem helyezhető sem elé, sem pedig mögé. (Erre a 11.9. ábra mutat példát.) Másrészről, a magas szintű cselekvések esetén a feloldhatatlan ütközések néha feloldhatók az ütköző cselekvések *dekompozíciójával* és lépéseik összerendezésével. Erre a 12.8. ábra mutat be egy példát. Így előfordulhat olyan eset, ahol dekompozícióval egy teljes és konzisztens alapterv nyerhető, *még akkor is, hogyha nem létezik teljes és konzisztensen magas szintű terv*. Ez a lehetőség azt jelenti, hogy egy teljes HFH-tervkészítőnek végig kell tekintenie az eredeti részben rendezett tervkészítőhöz található számos metszési lehetőséget. Egyébként használhatjuk bármely metszési eljárást, remélve, hogy nem hagyunk figyelmen kívül lehetséges megoldást.

[4] Vannak további apró módosítások a magas szintű cselekvések konfliktusfeloldásának kezelésére. Az érdeklődő olvasó ezeknek a fejezet végén hivatkozott irodalmakban nézhet utána.

Created by XMLmind XSL-FO Converter.

12.8. ábra - Az O. Henrik történetből kiragadott *Mágusok ajándéka* probléma egy inkonzisztens absztrakt tervet mutat, ami azonban dekomponálható egy konzisztens megoldásra. Az (a) ábra a problémát mutatja be: egy szegény házaspárnak csak két értékes tulajdona van. A férfinek egy aranyórája, a nőnek pedig a gyönyörű hosszú haja. Mindketten azt tervezik, hogy ajándékot vásárolnak a másiknak, hogy az boldog legyen. A férfi úgy dönt, hogy az óráját egy ezüstfésűre cseréli be, míg a nő eladja a haját, hogy aranyláncot vegyen az órához. (Feltételezzük, hogy a „Fésűt ad" cselekvés előfeltétele a *Haj*, mivel ha a feleségnek nincs hosszú haja, a cselekvés nem éri el a kívánt hatást, hogy boldoggá tegye; és hasonlóképpen a „Láncot ad" cselekvésre.) A (b) ábrán szereplő részleges terv inkonzisztens, mert a „Fésűt ad" és „Láncot ad" absztrakt lépések nem sorrendezhetők konfliktus nélkül. (c) Dekomponáljuk a „Fésűt ad" lépést egy „beilleszt terv" metódussal. A dekompozíció első lépésében a férj megszerzi a fésűt, és odaadja feleségének, miközben az órát egy későbbi időpontban adja oda fizetségül. A második lépésben az órát átadja, és a kötelezettséget teljesíti. Egy hasonló módszer dekomponálja a „Láncot ad" lépést. Amíg mindkét odaadó lépést a szállítási lépés elé sorrendezzük, ez a dekompozíció megoldja a feladatot. (Vegyük észre, hogy ez azon múlik, hogy a lánc használata az órához vagy a fésű használata a hajhoz boldogságot okoz még akkor is, ha a tulajdonjogot már elvesztették.)



2.3. Elemzés

Kezdjük a rossz hírrel: a tiszta HFH-tervkészítés (ahol az egyetlen megengedett tervfinomítás a dekompozíció), *a véges állapottér ellenére is,* eldönthetetlen! Ez nagyon elszomorítónak tűnhet, mivel a HFH-tervkészítés lényege, hogy növeljük a hatékonyságot. Ez a nehézség azért lép fel, mert a cselekvésdekompozíciók **rekurzív**ak (**recursive**) (például a sétálás implementálható egy lépés megtételével és utána egy sétálással), így a HFH-tervek tetszőlegesen hosszúra nyúlhatnak. Nevezetesen a legrövidebb HFH-megoldás is tetszőlegesen hosszú lehet, így nincs arra lehetőség, hogy a keresést egy megadott idő után leállítsuk. Ennek ellenére azonban legalább három okunk van a bizakodásra:

1. Kizárhatjuk a rekurziót, amit csak nagyon kevés tervkészítési feladat igényel meg. Ebben az esetben az összes HFH-terv véges vagy megszámlálható hosszúságú.

Created by XMLmind XSL-FO Converter.

2. Korlátozhatjuk azon megoldások hosszát, amelyekre kíváncsiak vagyunk. Mivel az állapottér véges, az olyan terv, amelynek több lépése van, mint az állapottér állapotainak száma, *mindenképpen* tartalmaz ciklust, ami ugyanabba az állapotba többször lép be. Az ilyen HFH-megoldások kizárásával nagyon picit vesztünk, de uraljuk a keresést.

3. Létrehozhatunk egy hibrid megközelítést, ami a részben rendezett és a HFH-tervkészítőket kombinálja. A részben rendezett tervkészítés önmagában elegendő, hogy eldöntsük, létezik-e terv, így a hibrid feladat is nyilvánvalóan eldönthető.

A harmadik megoldással egy kicsit óvatosan kell bánnunk. A részben rendezett tervkészítő különböző módokon tud elemi cselekvéseket összehuzalozni, így olyan megoldásokkal találhatjuk magunkat szembe, amelyek nagyon nehezen érhetők, és amelyeknek nincs meg a HFH-tervek szép hierarchikus rendezése. Egy megfelelő kompromisszum, hogy a hibrid keresést úgy szabályozzuk, hogy a cselekvésdekompozíciókat előnyben részesítjük az új cselekvések hozzáadása előtt, de nem olyan mértékben, hogy tetszőlegesen hosszú HFH-tervek jöjjenek létre, mielőtt egy elemi cselekvést hozzáadnánk. A megvalósítás egyik lehetséges módja egy költségfüggvény alkalmazása, amely engedményt ad a dekompozíció által bevezetett cselekvésekre. Minél nagyobb az engedmény, a keresés annál inkább a tiszta HFH-tervkészítésre hasonlít, és a megoldás annál inkább hierarchikus. A hierarchikus tervek rendszerint sokkal könnyebben végrehajthatók a valós helyzetekben, és könnyebben javíthatók, ha valami elromlik.

A HFH-tervek egy másik fontos tulajdonsága a részfeladat-megosztás lehetősége. Emlékezzünk vissza, hogy a részfeladat-megosztás azt jelenti, hogy ugyanazt a cselekvést használjuk fel a tervdekompozícióban szereplő két különböző lépéshez. Ha megtiltjuk az alfeladat megosztását, akkor a *d*’ dekompozíció minden példányosítása csak egyetlen módon hajtható végre, nem pedig sokféleképp, így jelentősen szűkítjük a keresési teret. Rendszerint ez a szűkítés időt spórol, és a legrosszabb esetben is az optimálisnál csak kicsivel hosszadalmasabb megoldásra vezet. Néhány esetben azonban több problémát is okozhat. Vegyük például az „élvezzük a mézesheteket, és neveljünk fel egy családot” célt. A tervkönyvtár a „házasodj és menj Hawaiira” tervvel állhat elő az első részcélra és a „házasodj és legyenek gyermekeid” tervvel a másodikra. A részfeladat megosztás nélkül a terv két különálló házasodás cselekvést tartalmaz, ami nagymértékben kerülendő.

Egy érdekes példa a részfeladat-megosztás költségeire és előnyeire a fordítók optimalizálásánál fordul elő. Vegyük a tan(*x*) – sin(*x*) kifejezés fordításának problémáját. A legtöbb fordító ezt két külön szubrutin triviális összeolvasztásával éri el: a tan lépések a sin lépések előtt következnek. Vegyük azonban a sin és tan alábbi Taylor soros közelítéseit:

$$\tan(x) \approx x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{17x^7}{315}$$

$$\sin(x) \approx x - \frac{x^3}{6} + \frac{x^5}{120} + \frac{x^7}{5040}$$

Egy részfeladat-megosztást tartalmazó HFH-tervkészítő sokkal hatékonyabb megoldásokat készíthet, mert a sin kiszámításának számos lépésére választhatja a tan meglévő lépéseit. A legtöbb fordító nem alkalmazza ezt a fajta procedúrák közötti megosztást, mert az összes lehetséges megosztott terv figyelembevétele túl sok időt venne igénybe. Ehelyett a legtöbb fordító minden résztervet függetlenül készít el, majd esetleg módosítja az eredményt egy **optimalizáló** használatával.

A cselekvésdekompozíció által bevezetett járulékos bonyolítások tudatában miért hisszük mégis, hogy a HFH-tervkészítés hatékony lehet? A komplexitás valós forrásai a gyakorlatban nehezen analizálhatók, ezért vegyünk egy idealizált esetet. Tegyük fel például, hogy egy *n* cselekvésből álló tervet akarunk létrehozni. Egy nemhierarchikus előrefelé haladó állapottér-tervező költsége, minden állapotban *b* megengedett cselekvéssel, $O(b^n)$. A HFH-tervkészítő esetén feltételezzünk egy nagyon általános dekompozíciós formát: minden nem elemi cselekvésnek, *d* lehetséges dekompozíciója van, mindegyik a következő alacsonyabb szinten *k* cselekvésből áll. Tudni szeretnénk, hogy ebben a struktúrában hány különböző dekompozíciós fa létezik. Ha a kiindulási szinten *n* cselekvés van, akkor a gyökér alatti szintek száma $\log_k n$, így a belső dekompozíciós csomópontok száma $1 + k + k^2 + \ldots k^{\log k\, n-1} = (n - 1)/(k - 1)$. Minden belső csomópontnak *d* lehetséges dekompozíciója van, így összesen $d^{(n-1)/(k-1)}$ lehetséges dekompozíciós fa alkotható. Ezt a képletet megvizsgálva láthatjuk, hogy *d* alacsonyan tartása nagy *k* mellett óriási megtakarításokat eredményezhet. Ha *b* és *d* összemérhető, akkor gyakorlatilag a nemhierarchikus költség *k*-adik gyökét vesszük. Másrészről azonban egy kisszámú, de hosszú dekompozíciót tartalmazó tervkönyvtár megalkotása, bár lehetővé teszi bármely probléma megoldását, nem mindig lehetséges. Másképp mondva, a hosszú makrók, amelyek problémák széles körén alkalmazhatók, kiemelten értékesek.

Created by XMLmind XSL-FO Converter.

Egy másik és talán jobb ok arra, hogy a HFH-tervkészítő hatékonyságában higgyünk, az, hogy a gyakorlatban működik. A nagyméretű alkalmazásokhoz használt tervkészítők majdnem mindegyike HFH-tervkészítő, mert a HFH-tervkészítés lehetőséget ad az emberi szakértő számára, hogy a komplex feladatok végrehajtásához szükséges elengedhetetlen tudást átadhassák, hogy a nagy tervek kicsi számítási ráfordítással elkészíthetők legyenek. Például a HFH-tervkészítést az ütemezéssel ötvöző O-PLAN-t (Bell és Tate, 1985) arra használták, hogy a Hitachi számára készítsen gyártási terveket. Egy tipikus gyártási sor probléma 350 különböző terméket, 35 összeszerelő gépet és több mint 2000 különböző műveletet tartalmaz. A tervkészítő egy 30 napos ütemezést készít, naponta három 8 órás műszakkal és több millió lépéssel.

A HFH-tervkészítés kulcsa így egy tervkönyvtár megalkotása, amely a komplex magas szintű cselekvések megvalósításához tartalmazza az ismert módszereket. A könyvtár megalkotásának egyik módja, hogy a problémamegoldások tapasztalataiból *tanuljunk* módszereket. Egy terv elkészítésének gyötrelmes tapasztalatai után az ágens elmentheti a tervet a könyvtárba mint egy a *feladat (task)* által definiált magas szintű cselekvés megvalósítási módját. Ily módon az ágens egyre kifinomultabbá válik, ahogy a régi módszerekre alapozva új módszerek épülnek. Ennek a tanulási folyamatnak egyik fontos szempontja, hogy az elkészített módszereket *általánosítani* tudja, a tervből csak a kulcslépéseket megtartva, azaz a probléma példányainak specifikus részleteit elhagyva (például az építő nevének és címének elhagyása a tervekről). Az ilyen általánosításra alkalmas módszereket a 19. fejezetben mutatjuk be. Számunkra elképzelhetetlen, hogy hasonló mechanizmusok nélkül az emberek olyan kompetensek lehetnek, mint amilyenek.

3. Tervkészítés és cselekvés nemdeterminisztikus problémakörökben

Eddig csak **klasszikus tervkészítés**i (**classical planning**) környezeteket vettünk figyelembe, amelyek teljesen megfigyelhetők, statikusak és determinisztikusak. Mindezeken túl feltételeztük, hogy a cselekvésleírások megfelelők és teljesek. Ilyen körülmények között egy ágens először tervez, majd „becsukott szemmel” végrehajtja a tervet. Egy bizonytalan környezetben másrészről az ágensnek az érzékeit is fel kell használnia, hogy kövesse a történéseket a terv végrehajtása alatt, valamint szükség esetén módosítsa vagy helyettesítse a tervet, amennyiben valami váratlan dolog történik.

Az ágensnek meg kell küzdenie mind a *nem teljes, hiányos (incomplete),* mind pedig a *nem megfelelő, hibás (incorrect)* információkkal. A nem teljesség abból következik, hogy a világ vagy csak részben megfigyelhető, vagy nemdeterminisztikus, vagy mindkettő. Például az irodaszerszekrény ajtaja vagy be van zárva, vagy nincs. Az egyik kulcsom vagy kinyitja az ajtót, vagy nem, már amennyiben be van zárva. Ráadásul ismereteim e hiányosságainak vagy tudatában vagyok, vagy nem. Ebből kifolyólag a világról alkotott modellem gyenge, de helyes. Másrészről a nem helyesség azért lép fel, mert a világ nem szükségszerűen felel meg a róla alkotott modellemnek. Például én *hihetem*, hogy a kulcsom kinyitja az irodaszerszekrényt, de ebben tévedhetek is, ha a zárakat kicserélték. A helytelen információ kezelési képességének hiányában az ágens olyan buta lehet, mint egy ganajtúró bogár (2.3. szakasz - Mindentudás, tanulás és autonómia részben), ami még az után is megpróbálja a ganajgolyót bedugni a fészkébe, hogy a golyót eltávolították a fogásából.

A teljes és korrekt ismeretek megszerzésének lehetősége attól függ, hogy mennyi nemdeterminisztikusság van a világban. **Korlátos nemdeterminisztikusság** (**bounded indeterminacy**) esetén a cselekvéseknek megjósolhatatlan következményei lehetnek, de a lehetséges következményeket felsorolhatjuk a cselekvésleíró axiómákban. Például amikor feldobunk egy pénzt, jogosan feltételezhetjük, hogy a kimenetele vagy *fej*, vagy *írás* lesz. Egy ágens kezelheti a korlátos nemdeterminisztikusságot úgy, hogy a terveket az összes lehetséges körülményre felkészíti. **Nem korlátos nemdeterminisztikusság** (**unbounded indeterminacy**) esetén azonban az előfeltételek vagy hatások halmaza vagy nem ismert, vagy túl nagy ahhoz, hogy kimerítően felsoroljuk. Ez az eset áll fenn a nagyon összetett vagy dinamikus problémakörökben, mint amilyenek például az autóvezetés, a gazdaságos tervezés vagy a hadműveletek. Egy ágens csak úgy kezelheti a nem korlátos nemdeterminisztikusságot, ha felkészítjük arra, hogy átdolgozza a terveit és/vagy a tudásbázisát. A nem korlátos nemdeterminisztikusság közeli rokonságban van a 10. fejezetben tárgyalt **kvalifikációs problémá**val (**qualification problem**) (az összes előfeltétel felsorolásának lehetetlenségével, amire egy valódivilág-beli cselekvésnek szüksége van, hogy a szándékolt hatást elérje).

A nemdeterminisztikusság kezelésére négy tervkészítő módszer létezik. Az első kettő a korlátos nemdeterminisztikusság, a második kettő pedig a nem korlátos nemdeterminisztikusság esetén használható:

- **Érzékelőmentes tervkészítés** (**sensorless planning**): ez a más néven **alkalmazkodó tervkészítés**nek (**conformant planning**) nevezett módszer normál, szekvenciális terveket készít, melyeket érzékelés nélkül

Created by XMLmind XSL-FO Converter.

kell végrehajtani. Az érzékelésmentes tervkészítő algoritmusnak biztosítania kell, hogy a terv a célt *minden lehetséges körülmény között* elérje függetlenül a valós induló állapottól és a cselekvések aktuális kimeneteleitől. Az érzékelésmentes tervkészítés a **kényszerítés**en (**coercion**) alapul, alapgondolata, hogy a világ erőszakkal egy megadott állapotba vihető még akkor is, ha az ágensnek csak részleges információi vannak az aktuális állapotról. A kényszerítés nem mindig lehetséges, így az érzékelésmentes tervkészítés gyakran alkalmazhatatlan. Az érzékelőmentes feladatmegoldást, amely magában foglalja a hiedelem állapottérben történő keresést, a 3. fejezetben mutattuk be.

- **Feltételes tervkészítés** (**conditional planning**): a más néven **eshetőségi tervkészítés** (**contingency planning**) módszere a korlátos nemdeterminizmust kezeli úgy, hogy egy feltételes tervet készít, amely a különböző eshetőségekhez különböző ágakat tartalmaz. Csakúgy, mint a klasszikus tervkészítésben, az ágens először tervez, majd végrehajtja az elkészített tervet. Az ágens érzékelő cselekvéseket épít a tervbe, hogy ellenőrizze a megfelelő feltételeket, így meghatározhatja, hogy a terv mely részét hajtsa végre. A légi szállítási feladatkörben például olyan terveink lehetnek, amelyek a következőket mondják: „ellenőrizd, hogy az SFO repülőtér használható-e. Ha igen, repülj oda; egyébként repülj Oaklandba”. A feltételes tervkészítéssel a 12.4. alfejezet foglalkozik.

- **Végrehajtás monitorozás és újratervezés** (**execution monitoring and replanning**): ebben a megközelítésben az ágens bármely ezt megelőző tervkészítő technikát felhasználhat (klasszikus, érzékelőmentes vagy feltételes), de használ egy **végrehajtás monitorozás**t (**execution monitoring**), amely eldönti, hogy a terv használható-e az aktuális állapotban, vagy újra kell gondolni. **Újratervezés** (**replanning**) akkor történik, amikor valami hiba lép fel. Ily módon az ágens a nem korlátos nemdeterminisztikusságot is kezelni képes. Például ha az újratervező ágens nem látta előre az SFO lezárásának lehetőségét, észreveheti ezt a szituációt, amikor fellép, és a tervkészítő által új útvonalat találhat a célhoz. Az újratervező ágenseket a 12.5. alfejezetben tárgyaljuk.

- **Folytonos tervkészítés** (**continuous planning**): minden eddig látott tervkészítő arra készült, hogy elérje a célt, és megálljon. A folytonos tervkészítő egy életen keresztüli működésre van tervezve. Kezelni képes a környezetben fellépő nem várt körülményeket még akkor is, ha azok az ágenst egy tervkészítés közepén érik. A **célújraformálás** (**goal formulation**) által képes kezelni a célok elhagyását, illetve új célok keletkezését is. A folytonos tervkészítést a 12.6. alfejezetben tárgyaljuk.

Vegyünk egy példát, hogy megvilágítsuk a különböző ágensek közötti különbségeket. A probléma a következő: a kiinduló állapotban adott egy szék, egy asztal és néhány doboz festék. A színe semminek sem ismert, és olyan állapotot kell elérni, ahol az asztal és a szék színe azonos.

A **klasszikus tervkészítő** ágens nem tudja kezelni ezt a problémát, mert a kiinduló állapot nem teljesen specifikált, azaz nem tudjuk, hogy milyen színűek a bútorok.

Az **érzékelőmentes tervkészítő** ágensnek egy olyan tervet kell találni, ami érzékelés nélkül működik a terv végrehajtása során. A megoldás, hogy bármelyik festékesdobozt nyitjuk ki, a festéket mind az asztalra, mind a székre alkalmazzuk. Így **kényszerítjük** (**coercing**) őket, hogy azonos színűek legyenek (még akkor is, ha az ágens nem tudja, hogy ez milyen szín). A kényszerítés akkor helyénvaló, ha az előzetes információk gyűjtése drága vagy lehetetlen. Például az orvosok gyakran egy széles spektrumú antibiotikumot használnak ahelyett, hogy egy feltételes tervet készítenének, ami egy vérvizsgálatot hajt végre, megvárja az eredményeket, és utána egy specifikus antibiotikumot alkalmaz. Ezt az indokolja, hogy a vérteszt rendszerint nagy költséggel és késedelemmel jár.

A **feltételes tervkészítő** ágens jobb tervet készíthet: először megvizsgálja az asztal és a szék színét, majd ha ezek már eleve azonosak, a terv kész. Ha nem, megnézi a festékesdobozok címkéit, ha ezek közt talál olyan színűt, mint amilyen a színe bármelyik bútordarabnak, akkor ezt a festéket alkalmazza a másikon. Egyébként befesti mindkét darabot bármelyik színnel.

Az **újratervező** ágens a feltételes tervkészítővel azonos tervet készíthet vagy elsőre egy jóval kevesebb ágat tartalmazót, majd amennyiben szükséges, a végrehajtási időben tölti fel a többit. A cselekvésleírások nem megfelelőségét szintén kezelni tudja. Például tegyük fel, hogy a *Fest*(*tárgy*, *szín*) cselekvésnek a *Szín*(*tárgy*, *szín*) determinisztikus hatást tulajdonítjuk. Egy feltételes tervkészítő feltételezi, hogy a következmény teljesül, ha a cselekvést végrehajtottuk, de az újratervező ágens ellenőrizheti a hatást és ha az nem igaz (talán mivel az ágens figyelmetlen volt és kihagyott egy részt), újra tervezhet és újra festhet egy részt. Ehhez a példához az 5. szakasz - Végrehajtás monitorozása és újratervezése részben visszatérünk.

Created by XMLmind XSL-FO Converter.

A **folytonos tervkészítő ágens** a nem várt események kezelésén túl újra is tervezhet, ha mondjuk a „legyen ebéd az asztalon” új célt illesztjük be, azaz a festési tervet el kell halasztani.

A valódi világban az ágensek a különböző megközelítések kombinációit használják. Az autógyártók pótkereket és tartalék légzsákokat árulnak, amelyek fizikai megtestesítői a feltételes tervágaknak, melyeket a defektek vagy az ütközések kezelésére készítettek. Másrészről a legtöbb autóvezető soha nem veszi figyelembe ezeket a lehetőségeket, így a defektekre vagy az ütközésekre mint újratervező ágensek reagálnak. Általánosságban az ágensek csak azokra az eshetőségekre készítenek feltételes tervet, amelyeknek fontos hatásai vannak, és nem elhanyagolható eséllyel okoznak hibát. Így egy autóvezetőnek, aki a Szaharán keresztül kíván utazni, igencsak figyelembe kell vennie a lerobbanás lehetőségét, míg egy áruházba vezető út jóval kevesebb megelőző tervezést igényel.

Az ebben a fejezetben bemutatott ágensek a nemdeterminisztikusság kezelésére készültek, de nem alkalmasak arra, hogy mérlegeljenek a sikeresség valószínűsége és a terv elkészítésének költsége tekintetében. A 16. fejezet további eszközöket biztosít az ilyen problémák kezelésére.

4. Feltételes tervkészítés

Feltételes tervkészítés a bizonytalanság kezelésének egy módja. Módszere, hogy a terv előre meghatározott pontjain ellenőrzi, hogy valójában mi is történik a környezetben. A feltételes tervkészítés bemutatása a teljesen megfigyelhető környezetek esetén a legegyszerűbb, így ezzel az esettel kezdünk. A részlegesen megfigyelhető eset jóval nehezebb, de egyben jóval érdekesebb is.

4.1. Feltételes tervkészítés teljesen megfigyelhető környezetekben

A teljes megfigyelhetőség jelentése, hogy az ágens mindig ismeri az aktuális állapotot. Nemdeterminisztikus környezet esetén azonban, az ágens nem képes megjósolni a cselekvéseinek *kimenetelét*. A feltételes tervkészítő ágens a nemdeterminisztikusságot úgy kezeli, hogy a tervekbe feltételes lépéseket épít be (tervkészítési időben), amelyek ellenőrzik a környezet állapotát (futási időben), hogy a továbbiakról dönthessen. A kérdés tehát, hogyan készíthetők el az ilyen feltételes tervek.

Példaként a **porszívóvilág** (**vacuum word**) problémakört használjuk, amelynek állapotterét a determinisztikus esetre a 2. szakasz - Példaproblémák részben fektettük le. Emlékezzünk vissza, hogy a *Balra*, a *Jobbra* és a *Szív* a rendelkezésre álló cselekvések. Szükségünk lesz néhány propozícióra, hogy definiáljuk az állapotokat: legyen az *OttBal* (*OttJobb*) igaz, ha az ágens a bal (jobb) állapotban van, és legyen a *TisztaBal* (*TisztaJobb*) igaz, ha a bal (jobb) állapot tiszta.[5] Az első feladatunk a `STRIPS` nyelv kiterjesztése, hogy megengedje a nemdeterminisztikusságot. Ennek érdekében megengedjük a cselekvésekben a **diszjunktív következmények**et (**disjunctive effects**), ami azt jelenti, hogy egy cselekvést bármikor végrehajtva annak kettő vagy több különböző kimenetele is lehet. Tegyük fel például, hogy a *Balra* lépés néha sikertelen. Ekkor a

```
Cselekvés(Balra, Előfeltétel:OttJobb, Következmény:OttBal ∧ ¬OttJobb)
```

normál cselekvésleírást módosítanunk kell, hogy diszjunktív következményt is tartalmazzon:

```
Cselekvés(Balra, Előfeltétel:OttJobb, Következmény:OttBal ∨ OttJobb) (12.1)
```

Szintén hasznosnak találjuk a **feltételes következmény**eket (**conditional effect**s), amikor is egy cselekmény következménye függ attól az állapottól, amelyben végrehajtjuk. A feltételes következmények a cselekvés `KÖVETKEZMÉNY` részében jelennek meg a „**when** *<feltétel>*: *<következmény>*” szintaxissal. Például a *Szív* cselekvés modellezésére a

```
Cselekvés(Szív, Előfeltétel:, Következmény:(when OttBal: TisztaBal)

∧ (when OttJobb: TisztaJobb))]
```

[5] Nyilvánvalóan az *OttJobb* akkor és csak akkor igaz, ha a ¬*OttBal* igaz, és fordítva. A két állítás használatának oka főként az olvashatóság javítása.

Created by XMLmind XSL-FO Converter.

kifejezést írnánk fel. A feltételes következmények nem vezetik be a nemdeterminisztikusságot, de segítséget nyújtanak annak modellezésében. Tegyük fel például, hogy egy körmönfont porszívónk van, ami néha, ha mozog, piszkot szór a célnégyzetre, de csak akkor, ha az tiszta. Ez a

```
Cselekvés(Balra, Előfeltétel:OttJobb, Következmény:OttBal

∨ (OttBal ∧ (when TisztaBal: ¬TisztaBal))
```

leírással modellezhető, ami mind diszjunktív, mind pedig feltételes.[6] Hogy feltételes terveket készíthessünk, **feltételes lépés**ekre (**conditional step**s) van szükségünk. Ezeket az „**if** <*teszt*> **then** *terv_A* **else** *terv_B*” szintaxis használatával írjuk le, ahol a <*teszt*> egy kétértékű függvénye az állapotváltozóknak. Az „**if** *OttBal* ∧ *TisztaBal* **then** *Jobbra* **else** *Szív*” például a porszívóvilág egy feltételes lépése lehet. Egy ilyen lépés végrehajtása a kézenfekvő módon történik. A feltételes lépések egymásba ágyazásaival a tervek fák lesznek.

A feltételes tervektől elvárjuk, hogy működjenek, *függetlenül attól, hogy valójában a cselekvés mely kimenetele következik be*. Ezzel a problémával egy másik köntösbe bújtatva már találkoztunk korábban. A kétszemélyes játékokban (lásd 6. fejezet) olyan lépéseket szeretnénk, amelyek *az ellenfél lépéseitől függetlenül* győzelemhez vezetnek. A nemdeterminisztikus tervkészítési problémákat ezért gyakran **természet elleni játék**oknak (**games against nature**) nevezik.

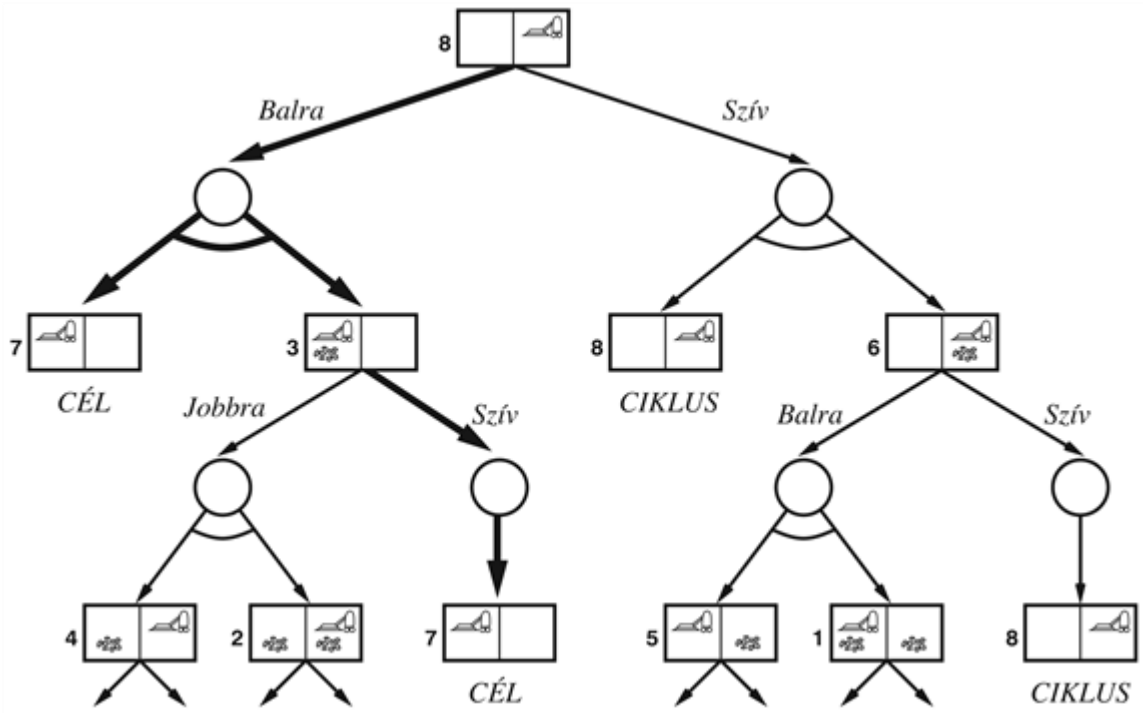
Vegyük a porszívóvilág egy speciális példáját. A kiinduló állapotban a robot a tiszta világ jobb oldali négyzetén van. Mivel a környezet teljesen megfigyelhető, az ágens ismeri a teljes *OttJobb* ∧ *TisztaBal* ∧ *TisztaJobb* állapotleírást. A célállapotban a robot a tiszta világ bal oldali négyzetén van. Ez a feladat elég triviális lenne, ha nem a „dupla-Murphy” porszívóval lenne dolgunk, amely néha piszkot hagy maga után, amikor egy tiszta célnégyzetre lép, és néha bepiszkolja a tiszta négyzetet, ha a *Szívás* cselekvés végrehajtódik.

Ennek a környezetnek a „játékfáját” a 12.9. ábrán mutatjuk be. A cselekvéseket a robot a fa „állapot” csomópontjaiban hajtja végre, majd a körrel jelölt „valószínűségi” csomópontokban a természet dönt a cselekvés kimeneteléről. A megoldás egy részfa, mely (1) minden levelében egy cél csomópontot tartalmaz, (2) minden „állapot” csomóponthoz egy cselekvést specifikál, és (3) minden „valószínűségi” csomópontban tartalmazza az összes kimenetelhez tartozó ágat. Az ábrán a megoldást vastag vonallal jelöltük, ami a [*Balra*, **if** *OttBal* ∧ *TisztaBal* ∧ *TisztaJobb* **then** [] **else** *Szív*] tervnek felel meg. (Mivel állapottér-tervkészítőt használunk, a feltételes lépésekben használt tesztek egyelőre teljes állapot leírások.)

12.9. ábra - A „dupla-Murphy” porszívóvilág keresési fájának első két szintje. Az állapotcsomópontokban és a VAGY csomópontokban cselekvéseket kell választani. A valószínűségi csomópontok, amelyeket körökkel jelöltünk ÉS csomópontok, ahol, ahogy azt a kimenő ágakon szereplő ív is jelöli, minden kimenetelt kezelni kell. A megoldást vastag vonallal jelöltük.

[6] A **when** *TisztaBal*: ¬*TisztaBal* feltételes következmény egy kicsit furcsának tűnhet. Emlékezzünk vissza azonban, hogy itt a *TisztaBal* a cselekvés *előtti,* míg a ¬*TisztaBal* a cselekvés végrehajtása *utáni* helyzetre vonatkozik.

Created by XMLmind XSL-FO Converter.



A játékok pontos megoldásához a **minimax algoritmus**t használjuk (lásd 6.3. ábra). Ehhez a feltételes tervkészítésben tipikusan két módosítás tartozik. Először is, a `MAX` és a `MIN` csomópontok `VAGY` és `ÉS` csomópontokká válhatnak. Nyilvánvalóan a tervnek minden elért állapotban választania kell *valamilyen* cselekvést, de kezelnie kell ezen cselekvés *minden* kimenetelét. Másodszor, az algoritmusnak nemcsak egy lépést kell megadnia, hanem egy feltételes tervet kell készítenie. Egy `VAGY` csomópontban a terv egyszerűen a választott cselekvés, amelyet bármi követhet. Egy `ÉS` csomópontban a terv *if-then-else* lépések egymásba ágyazott sorozata, melyek minden lehetséges kimenetelhez egy résztervet adnak meg. Ezen lépésekben a feltétel vizsgálatokban teljes állapotleírások szerepelnek.[7]

12.10. ábra - Egy algoritmus a nemdeterminisztikus környezetek által generált `ÉS-VAGY` gráfok keresésére. Feltételezzük, hogy az `ÁLLAPOTÁTMENET` függvény a cselekvések egy listáját adja vissza, melyek mindegyike egy lehetséges kimenetel *halmazhoz* tartozik. A cél egy feltételes terv megtalálása, ami bármilyen körülmények között elér egy célállapotot.

[7] Az ilyen terveket **case** szerkezettel is leírhatnánk.

Created by XMLmind XSL-FO Converter.

```
function ÉS-VAGY-GRÁF-KERESÉS(feladat) returns egy feltételes terv vagy kudarc
    VAGY-KERESÉS(KIINDULÓ-ÁLLAPOT[feladat], feladat, [])

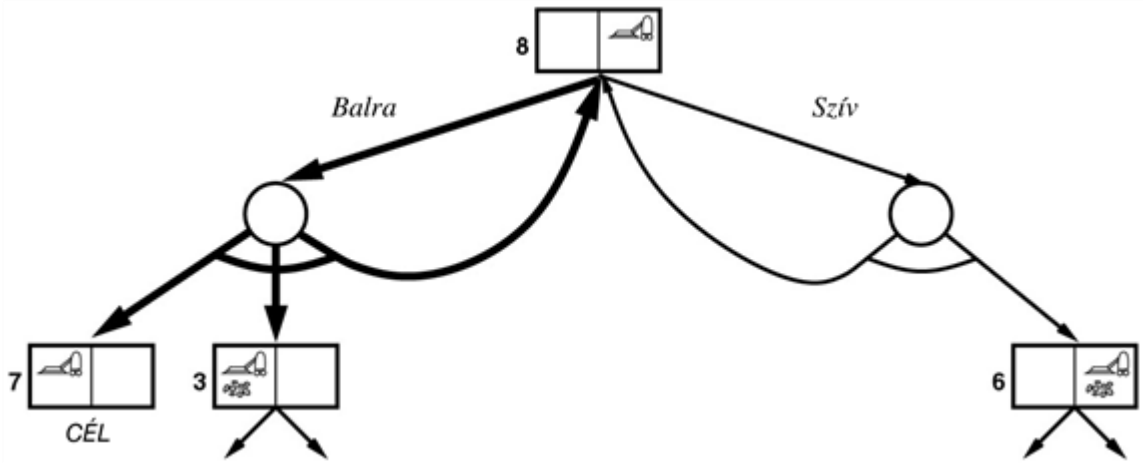
function VAGY-KERESÉS(állapot, [feladat], feladat, []) returns egy feltételes terv vagy kudarc
    if CÉL-TESZT[feladat](állapot) then return üres terv
    if állapot az útvonal-on then return kudarc
    for each cselekvés, állapot_halmaz in KÖVETŐÁLLAPOTOK[feladat](állapot) do
        terv ← ÉS-KERESÉS(állapot_halmaz, feladat, [állapot|útvonal])
        if terv ≠ kudarc then return [cselekvés|terv]
    return kudarc

function ÉS-KERESÉS(állapot_halmaz, feladat, terv) returns egy feltételes terv vagy kudarc
    for each s_i in állapot_halmaz do
        terv_i ← VAGY-KERESÉS(s_i, feladat, terv)
        if terv = kudarc then return kudarc
    return [if s_1 then terv_1 else if s_2 then terv_2 else ... if s_{n-1} then terv_{n-1} else terv_n]
```

Formálisan az eddig definiált keresési tér egy *és-vagy* **gráf.** Az ÉS-VAGY gráfok korábban a 7. fejezetben az ítéletlogikai Horn-klóz következtetésben jelentek meg. Itt az ágak logikai következtető lépések helyett cselekvések, de az algoritmus azonos. A 12.10. ábra az ÉS-VAGY gráfok keresésére egy rekurzív, mélységi kereső algoritmust ad meg.

A bemutatott algoritmusban kulcsfontosságú a nemdeterminisztikus tervkészítési problémákban gyakran felmerülő ciklusok kezelésének módja (például ha egy cselekvésnek néha nincs következménye vagy egy helytelen következmény kijavítható). Ha az aktuális állapot azonos a gyökértől idáig vezető útvonal egy állapotával, akkor hibával tér vissza. Ez nem jelenti, hogy *nincs* megoldás az aktuális állapotból, egyszerűen annyit jelent, hogy *van* egy nemciklikus megoldás, ami elérhető az aktuális állapot korábbi előfordulásából, így az állapot újabb bekövetkezése kihagyható. Ezzel az ellenőrzéssel biztosítjuk, hogy az algoritmus minden véges állapottér esetén leálljon, mivel minden útvonal célt ér, zsákutcába jut, vagy egy állapot ismétlése. Vegyük észre, hogy az algoritmus nem ellenőrzi, hogy az aktuális állapot egy *másik* útvonalon szereplő állapot ismétlése-e. A 12.15. feladat ezt a kérdést járja körül.

12.11. ábra - A „tripla-Murphy” porszívóvilág keresési gráfjának első szintje, ahol a ciklusokat explicit megjelöltük. A probléma összes megoldása ciklikus terv.



Az ÉS-VAGY-GRÁF-KERESÉS által visszaadott tervek feltételes lépéseket tartalmaznak, melyek a teljes állapotleírást megvizsgálják, hogy egy ágról döntsenek. A legtöbb esetben ennél jóval kevésbé kimerítő ellenőrzésekkel is megúszhatjuk. Például a 12.9. ábrán látható megoldás a [*Balra*, **if** *TisztaBal* **then** [] **else** *Szív*] megadással egyszerűen leírható. Ennek oka, hogy a *TisztaBal* teszt elegendő, hogy az ÉS csomópont állapotait két egyelemű halmazba sorolja úgy, hogy a tesztek után az ágens pontosan ismerje az állapotát. Valójában az egyváltozós if-then-else tesztek sorozata mindig elegendő, hogy állapotok egy halmazát egyelemű halmazokra

Created by XMLmind XSL-FO Converter.

ossza, *feltéve,* hogy az állapot teljesen megfigyelhető. Ezért az általánosság teljes megőrzése mellett a teszteket egyváltozós tesztekre szűkíthetjük.

Az utolsó nehézség, ami gyakran felmerül a nemdeterminisztikus feladatkörökben, a következő: a dolgok nem mindig működnek elsőre, így újra kell próbálkozni. Vegyük például a „tripla-Murphy" porszívó példáját, mely (a korábban bemutatott szokások mellett) néha nem mozdul az utasítás ellenére. Például csakúgy, mint a (12.1) egyenletben, a *Balra* cselekvés tartalmazhatja a *OttBal* ∨ *OttJobb* diszjunktív hatást. Ekkor a [*Balra*, **if** *TisztaBal* **then** [] **else** *Szív*] terv már nem garantált, hogy működik. A 12.11. ábra a keresési gráf egy részletét mutatja. Tisztán látható, hogy a továbbiakban nincsenek ciklusmentes megoldások, és az ÉS-VAGY-GRÁF-KERESÉS hibával térne vissza. Létezik azonban egy **ciklikus megoldás** (**cyclic solution**), ami addig próbálgatja a *Balra* lépést, mígnem egyszer működik. Ez a megoldás könnyebben kifejezhető, ha a terv egy részét egy **címké**vel (**label**) jelöljük meg, és a terv ismételgetése helyett erre a címkére hivatkozunk. Így a ciklikus megoldásunk

$[L_1 : \mathit{Balra}, \mathit{if\ OttJobb\ then}\ L_1\ \mathit{else\ if\ TisztaBal\ then}\ [] \ \mathit{else\ Szív}]$

alakú. (A „**while** *OttJobb* **do** *Balra*" kifejezés egy jobb szintaxis a terv ciklikus részére.) Az ÉS-VAGY-GRÁF-KERESÉS-ben szükséges módosításokat a 12.16. feladat dolgozza fel. A megvalósítás kulcsa, hogy az állapottérben egy *L* állapotba visszalépő hurok a tervben egy arra a pontra visszamutató hurkot jelent, ahol az *L* állapotba vezető résztervet végrehajtjuk.

Így már képesek vagyunk feltételeket és ciklusokat tartalmazó programokhoz hasonlatos összetett tervek létrehozására. Sajnos ezek a ciklusok *végtelen* ciklusok lehetnek. Például a tripla-Murphy világ cselekvés reprezentációjában a semmi jelentése, hogy a *Balra* szükségszerűen sikeres. A ciklikus tervek ezért kevésbé előnyösek, mint a ciklus nélküliek, de megoldásnak tekinthetők, amennyiben minden levél egy célállapot, és a terv minden pontjából elérhető egy levél.

4.2. Feltételes tervkészítés részlegesen megfigyelhető környezetekben

Az előző alfejezet teljesen megfigyelhető környezetekkel foglalkozott, amelyek előnye, hogy a feltételes ellenőrzések bármit kérdezhetnek, és biztosak lehetnek a választ kapnak. A valódi világban a részleges megfigyelhetőség jóval gyakoribb. Egy részlegesen megfigyelhető tervkészítési feladat kiinduló állapotában az ágens csak bizonyos dolgokat tud az aktuális állapotról. Ennek a helyzetnek a legegyszerűbb modellezése, ha a kiinduló állapotról annyit mondunk, hogy egy **állapothalmaz**ba tartozik. Az állapothalmaz az ágens kiinduló **hiedelmi állapot**át (**belief state**) írja le.[8]

Tegyük fel, hogy a porszívóvilág ágens tudja, hogy a jobb oldali négyzeten van, és az tiszta, de nem tudja érzékelni a piszok jelenlétét vagy hiányát a többi négyzeten. Ekkor, *legjobb ismeretei szerint* két állapotban lehet: a bal oldali négyzet vagy piszkos, vagy tiszta. Ezt a hiedelmi állapotot a 12.12. ábrán *A*-val jelöltük. Az ábra a „váltakozó dupla-Murphy" porszívóvilág ÉS-VAGY gráfjának egy részét mutatja be, melyben a tiszta négyzetet elhagyó ágens piszkot hagyhat maga után.[9] Ha a világ teljesen megfigyelhető volna, az ágens egy „Mozogj balra és jobbra, és szívd fel a piszkot, amennyiben találsz, amíg mindkét kocka tiszta nem lesz, és a bal oldali kockán vagyok" formájú ciklikus megoldást készíthetne (lásd 12.16. feladat). Sajnos csak lokális szemétérzékeléssel ez a terv végrehajthatatlan, hiszen a „mindkét kocka tiszta" teszt igazságértéke nem határozható meg.

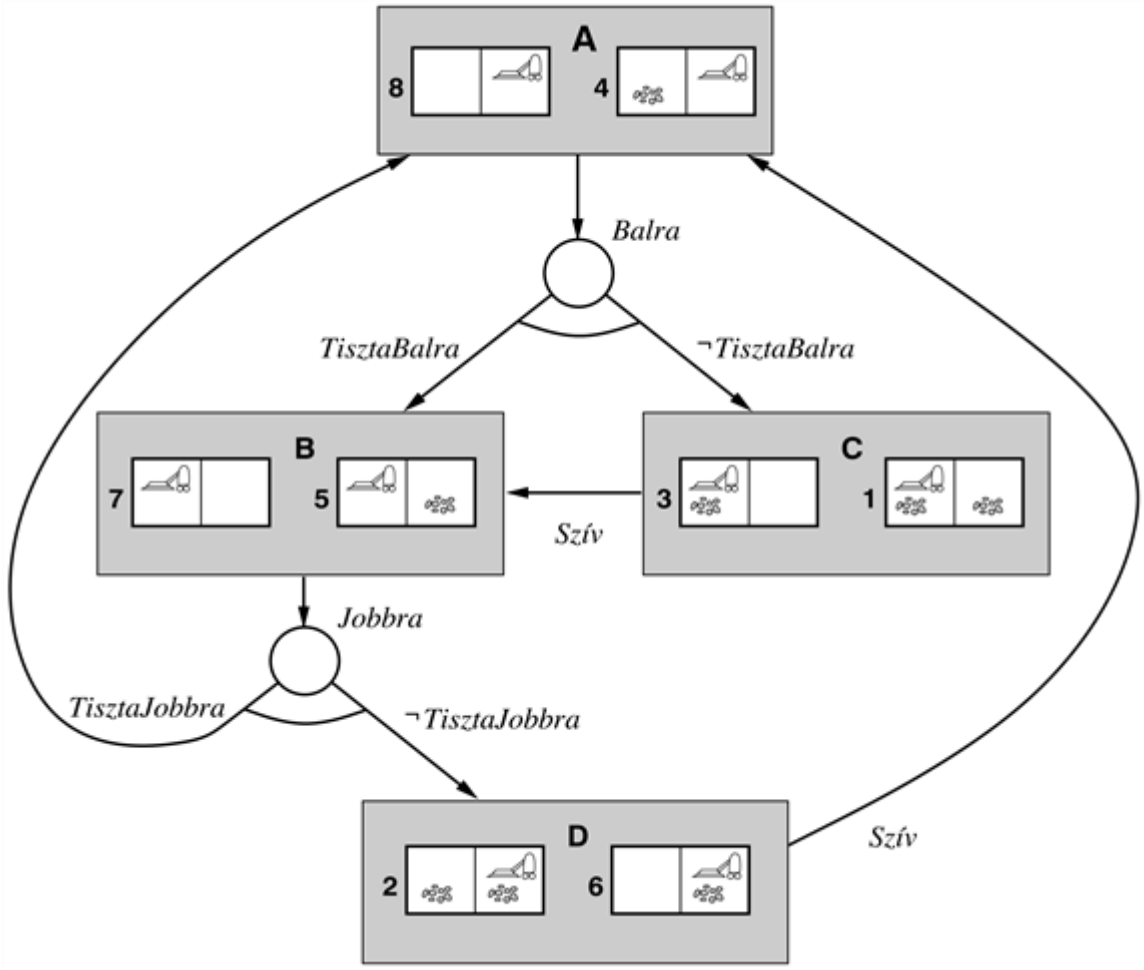
Figyeljük meg az ÉS-VAGY gráf felépítését. Az *A* hiedelmi állapotból a *Balra* mozgás kimenetelét mutatjuk. (A többi cselekvésnek nincs értelme.) Mivel az ágens piszkot hagyhat maga után, a két kiinduló világból négy lehetséges világ adódhat, ahogy az a *B* és *C* állapotokban látható. A rendelkezésre álló érzékelő információk alapján a világ két különálló hiedelmi állapotra osztható.[10] A *B*-ben, az ágens tudása a *TisztaBal*, míg *C*-ben a ¬*TisztaBal*. A piszok feltakarítása *C*-ben az ágenst a *B*-be mozgatja. A *B*-ből a jobbra mozgás vagy hagy piszkot maga után, vagy nem, így az ágensnek azon tudása alapján, hogy *TisztaJobb* igaz (vissza az *A*-ra) vagy hamis (*D* hiedelmi állapot), ismét négy lehetséges világ adódik.

[8] Ezeket a fogalmakat a 3.6. alfejezetben vezettük be, így az olvasó ezt átismételheti a továbblépés előtt.
[9] A kisgyermekes szülők jól ismerhetik ezt a jelenséget. Tisztelet a kivételnek.
[10] Vegyük észre, hogy ezek nem aszerint kerülnek osztályzásra, hogy az ágens piszkot hagy-e maga után, amikor mozdul. A hiedelmi állapottérbeli elágazásokat a különböző tudáslehetőségek, és nem a különböző fizikai kimenetelek okozzák.

Created by XMLmind XSL-FO Converter.

Összegezve a nemdeterminisztikus, részben megfigyelhető környezetek a hiedelmi állapotok egy ÉS-VAGY gráfját eredményezik. Ebből adódóan feltételes tervek találhatók pontosan ugyanazon algoritmusokkal, mint a teljesen megfigyelhető esetben, nevezetesen az ÉS-VAGY-GRÁF-KERESÉS-sel. Ez abból is könnyen megérhető, ha belátjuk, hogy az ágens *hiedelmi* állapota *mindig* teljesen megfigyelhető, azaz mindig tudja, hogy mit tud. A „hagyományos" teljesen megfigyelhető problémamegoldás csak egy speciális eset, melyben minden hiedelmi állapot egy egyelemű halmaz, pontosan egy fizikai állapottal.

12.12. ábra - A „váltakozó dupla-Murphy" porszívóvilág ÉS-VAGY gráfjának egy részét mutatja be, melyben a tiszta négyzetet elhagyó ágens piszkot hagyhat maga után. Az ágens nem tudja érzékelni a más négyzetben levő piszkot.



Ezzel készen vagyunk? Nem egészen! Még meg kell határoznunk, hogy hogyan reprezentáljuk a hiedelmi állapotokat, hogyan működik az érzékelés, és hogy ebben az új helyzetben hogyan írjuk le a cselekvéseket.

A hiedelmi állapotok esetén alapvetően három választásunk van:

1. Teljes állapotleírások halmazai. Például a 12.12. ábra kiinduló hiedelmi állapota az

$\{(OttJobb \wedge TisztaJobb \wedge TisztaBal), (OttJobb \wedge TisztaJobb \wedge \neg TisztaBal)\}$

Ezzel a leírással könnyű dolgozni, de nagyon költséges: ha n kétértékű ítéletállítás definiál egy állapotot, akkor a hiedelmi állapot $O(2^n)$ fizikai állapotleírást tartalmazhat, melyek mindegyike $O(n)$ méretű. Ha az ágens az ítéletállításoknak csak egy töredékét ismeri, exponenciálisan nagy hiedelmi állapotok adódnak — minél kevesebbet tud, annál több lehetséges állapotban lehet.

2. Logikai mondatok, melyek pontosan leírják a hiedelmi állapotban lehetséges világok halmazát. Például a kiinduló állapot az

$OttJobb \wedge TisztaJobb$

Created by XMLmind XSL-FO Converter.

formában adható meg. Nyilvánvaló, hogy bármely hiedelmi állapot pontosan befoglalható egyetlen logikai mondatba. Ha akarjuk, az összes konjuktív állapotleírás diszjunkcióját vehetjük, de a példánk mutatja, hogy ennél tömörebb mondatok létezhetnek.

Az általános logikai mondatok egyik hátulütője, hogy mivel sok ekvivalens, de különböző logikai mondat írhatja le ugyanazt a hiedelmi állapotot, az ismétlődő állapotok ellenőrzése általános tételbizonyító képességeket vár el a gráfkereső algoritmustól. Ezért a mondatok egy *kanonikus* reprezentációját szeretnénk, amelyben minden hiedelmi állapot pontosan ugyanannak a mondatnak felel meg.[11] Egy ilyen reprezentáció, azaz ítéletállítások nevei alapján rendezett literálok konjukcióját használja, melynek egy példája a $\neg OttJobb \wedge TisztaJobb$. Ez a 11. fejezet **nyílt világ feltételezés**ében (**open-world assumption**) egy egyszerű állapotleírás. Nem minden logikai mondat írható fel ilyen alakban (például nincs mód az $OttBal \vee TisztaJobb$ felírására), de számos probléma kezelhető.

3. **Tudás ítéletállítás**ok (**knowledge proposition**s), amelyek az ágens ismereteit írják le. (Ugyanezt lásd a 7.7. alfejezetben.) A kiinduló állapotunk:

```
K(OttJobb) ∧ K(TisztaJobb)
```

ahol *K* jelentése „tudja” és *K*(*P*) jelentése, hogy az ágens tudja, hogy *P* igaz.[12] A tudás ítéletállításokkal zárt világ feltételezést használunk, azaz ha egy állítás nem jelenik meg a listában, akkor hamisnak feltételezzük. Például a $\neg K(TisztaBal)$ és a $\neg K(\neg TisztaBal)$ implicit szerepelnek a fenti mondatban, így az rögzíti a tényt, hogy az ágens érzéketlen a *TisztaBal* igazságértékére.

Kimutatható, hogy a második és harmadik lehetőségek durván azonosak, de mi a harmadik tudás, az ítéletállítás lehetőséget használjuk, mert ez az érzékelés egy erősebb leírását adja, és mert már tudjuk, hogy hogyan írhatunk a **zárt világ feltételezés** (**closed-world assumption**) mellett `STRIPS` kifejezéseket.

Fontos

Mindkét esetben, minden ítéletszimbólum háromféleképpen jelenhet meg: lehet pozitív, negált vagy ismeretlen. Ezért így pontosan 3^n lehetséges hiedelmi állapot adható meg. A hiedelmi állapotok halmaza így a hatványhalmaza (az összes részhalmaz halmaza) a fizikai állapotoknak. Összesen 2^n fizikai állapot van, ezért 2^{2^n} hiedelmi állapot, ami sokkal több, mint 3^n, így a 2. és 3. választás eléggé korlátozottan alkalmas a hiedelmi állapotok leírására. Ez jelenleg használhatatlannak tűnik, hiszen *bármely séma, ami képes minden lehetséges hiedelmi állapot reprezentálására, $O(\log_2(2^{2^n})) = O(2^n)$ bitet igényel, hogy legrosszabb esetben mindegyiket leírhassa*. A mi egyszerű sémánk csak $O(n)$ bitet igényel a hiedelmi állapotok leírásához, mert a kifejezőképességet a tömörségre cseréltük. Nevezetesen, ha egy cselekvés megjelenik, amelynek az előfeltételei ismeretlenek, akkor az eredményként kapott hiedelmi állapot nem lesz pontosan reprezentálható, és a cselekvés kimenetele ismeretlen lesz.

Most arról kell döntenünk, hogy az érzékelés hogyan működik. Itt két választásunk van. Használhatunk **automatikus érzékelés**t (**automatic sensing**), ami annyit tesz, hogy az ágens minden időlépésben az összes elérhető érzetet megkapja. A 12.12. ábrán látható példa a helyzet és a helyi tisztaság meghatározására automatikus érzékelést feltételez. Használhatunk ellenben **aktív érzékelés**t (**active sensing**), ami annyit jelent, hogy az érzékelt információk csak megadott **érzékelési cselekvés**ek (**sensory action**s), (mint *TisztaságEllenőrzés* és *PozícióEllenőrzés*) végrehajtásával nyerhetők. Az érzékelési típusokat sorban tárgyalni fogjuk.

Most használjunk tudás állításokat a cselekvések leírásához. Tegyük fel, hogy az automatikus helyi tisztaságérzékeléssel felruházott váltakozó-dupla-Murphy világ problémában az ágens *Balra* lép. Az erre a világra vonatkozó szabályoknak megfelelően az ágens hagy, vagy talán nem hagy piszkot maga után, ha a négyzet tiszta volt. Mint *fizikai* következmény, ez *diszjunktív* lenne, de mint *tudás* következmény ez egyszerűen törli az ágens *TisztaJobb* tudását. Emellett az ágens a helyi piszokérzékelés miatt így vagy úgy tudni fogja, hogy vajon a *TisztaBal* igaz-e, és tudni fogja, hogy a pozíciója *OttBal*:

```
Cselekvés(Balra, Előfeltétel:OttJobb,
```

[11] Az általános ítéletlogikai mondatok legjobb kanonikus reprezentációja a **bináris döntési diagram** (**binary decision diagram**) vagy **BDD** (Bryant, 1992).

[12] Ez a jelölés ugyanaz, mint amit a 7. fejezetben az áramköralapú ágenseknél használtunk. Néhány szerző ezt a „tudja, hogy *P* igaz-e” értelemben használja. A két értelmezés közötti fordítás kézenfekvő.

Created by XMLmind XSL-FO Converter.

```
Következmény:K(OttBal) ∧ ¬K(OttJobb) ∧

when TisztaJobb: ¬K(TisztaJobb) ∧

when TisztaBal: K(TisztaBal) ∧

when ¬TisztaBal: K(¬TisztaBal) (12.2)
```

Vegyük észre, hogy az előfeltételek és a **when** feltételek egyszerű állítások és nem tudás állítások. Ez úgy van, ahogy lennie kell, hiszen a cselekvések kimenetele függ az aktuális világtól, de hogyan ellenőrizhetnénk ezen feltételek igazságértékét csupán a hiedelmi állapot alapján? Ha az ágens az aktuális hiedelmi állapotban *tudja* mondjuk a *K*(*OttJobb*) állítást, akkor az állításnak igaznak kell lennie az aktuális fizikai állapotban, így egyben a cselekvés is alkalmazható. Ha az ágens nem ismeri az állítást (például *TisztaBal* az **if** feltételt), akkor a hiedelmi állapotnak tartalmaznia kell olyan világokat, melyekben a *TisztaBal* igaz, és olyan világokat, amelyekben a *TisztaBal* hamis. Pontosan ez az, ami miatt egy cselekvés többszörös hiedelmi állapotokat eredményez. Így, ha a kiinduló állapot a (*K*(*OttJobb*) ∧ *K*(*TisztaJobb*)), akkor a *Balra* lépés után a (*K*(*OttBal*) ∧ *K*(*TisztaBal*)) és a (*K*(*OttBal*) ∧ *K*(¬*TisztaBal*)) a két lehetséges hiedelmi állapot. Mindkét esetben ismert a *TisztaBal* értéke, így a *TisztaBal* teszt felhasználható a tervben.

Aktív érzékelés (mint az automatikus érzékelés ellentettje) esetén, az ágens csak kérésre kap új megfigyeléseket. Így a *Balra* lépés után az ágens nem tudja, hogy a bal oldali négyzet piszkos-e, ezért a (12.2) *cselekvésleíró* egyenletben az utolsó két feltételes következmény már nem jelenik meg. Az ágens a *TisztaságEllenőrzés* cselekvéssel derítheti ki, hogy a négyzet piszkos-e:

```
Cselekvés(TisztaságEllenőrzés, Következmény: when OttBal ∧ TisztaBal: K(TisztaBal) ∧

when OttBal ∧ ¬TisztaBal: K(¬TisztaBal) ∧

when OttBal ∧ TisztaJobb: K(TisztaJobb) ∧

when OttBal ∧ ¬TisztaJobb: K(¬TisztaJobb) (12.3)
```

Könnyű megmutatni, hogy aktív érzékelés esetén a *Balra* lépéssel követett *TisztaságEllenőrzés* cselekvés ugyanazt a két hiedelmi állapotot eredményezi, amit az automatikus érzékelés esetén a *Balra* adott. Aktív érzékelés esetén a fizikai cselekvések egy hiedelmi állapotot mindig egyetlen követő hiedelmi állapotra képeznek le. A többszörös hiedelmi állapotok csak az érzékelő cselekvések által jöhetnek létre, melyek specifikus tudást adnak, s ezért lehetővé teszik a feltételvizsgálatok használatát a tervekben.

Az eddigiekben az állapottér `ÉS-VAGY` keresésén alapuló feltételes tervkészítésre mutattunk be egy általános megközelítést. Ez a megközelítés néhány tesztproblémán elég hatékonynak bizonyult, de más feladatokra alkalmatlan. Bizonyítható, hogy a feltételes tervkészítés nagyobb algoritmikus komplexitású, mint a hagyományos. Emlékezzünk vissza, hogy az *NP* osztály definíciója alapján egy megoldásról ellenőrizhető, hogy polinomiális idejű-e. Ez a hagyományos tervekre (legalábbis a polinomiális méretűekre) igaz, így a hagyományos tervkészítés az *NP* osztályba tartozik. A feltételes tervkészítés esetén egy jelöltre ellenőrizni kell, hogy az *összes* lehetséges állapotra a tervben létezik-e *valamilyen*, a célt kielégítő útvonal. Az „összes/valamilyen” összeállítás nem ellenőrizhető polinomiális időben, így a feltételes tervkészítés nehezebb, mint *NP*. Ez csak úgy kerülhető el, hogy a tervkészítési fázisban figyelmen kívül hagyunk néhány lehetséges eshetőséget, és ezeket csak akkor kezeljük, ha valójában fellépnek. A következő alfejezetben ezt a megközelítést elemezzük.

5. Végrehajtás monitorozása és újratervezése

A **végrehajtás-monitorozó** (**execution monitoring**) ágens érzékelőivel ellenőrzi, hogy minden a terv szerint megy-e. Murphy törvénye alapján az egereknek, embereknek és a feltételes tervkészítő ágenseknek még a legjobban elkészített tervei is gyakran sikertelenek. A probléma a nem korlátos nemdeterminisztikusság, azaz néhány váratlan körülmény mindig felmerül, melyre az ágens cselekvés leírásai helytelenek. A valós környezetekben ezért a végrehajtás-monitorozás elengedhetetlen. Kétféle végrehajtás-monitorozást veszünk számba: az egyszerű, de gyenge **cselekvésmonitorozás**t (**action monitoring**), ahol az ágens a környezet vizsgálatával ellenőrzi, hogy a következő cselekvés működni fog, és az összetettebb, de hatékonyabb **tervmonitorozás**t (**plan monitoring**), melyben az ágens a terv teljes hátralevő részét ellenőrzi.

Created by XMLmind XSL-FO Converter.

Az **újratervező** (**replanning**) ágens tudja, hogy váratlan események esetén mit kell tennie: újra meghívja a tervkészítőt, hogy a cél eléréséhez egy új tervet biztosítson. Annak elkerülésére, hogy túl sok időt töltsünk tervkészítéssel, ez rendszerint a régi terv javításával történik, azaz utat keresünk a fennálló nem várt állapotból vissza, a meglévő tervhez.

Példaként térjünk vissza a 12.9. ábra dupla-Murphy porszívóvilágához. Ebben a világban egy tiszta cellára lépés néha bepiszkolja azt. De mi történik, ha az ágens ezt nem tudja vagy nem foglalkozik vele? Akkor egy nagyon egyszerű megoldással áll elő: [*Balra*]. Ha a terv végrehajtásánál érkezéskor nem történik piszkítás, akkor az ágens a cél elérését detektálja. Ellenkező esetben azonban, mivel az implicit *Befejez* lépés *TisztaBal* előfeltétele nem teljesül, az ágens új tervet készít: [*Szív*]. Ennek a tervnek a végrehajtása mindig sikeres.

A végrehajtás-monitorozás és újratervezés együttesen egy általános stratégia, ami mind a teljesen, mind pedig a részben megfigyelhető környezetekre alkalmazható a tervkészítési reprezentációk széles körén, beleértve az állapottér-, a részben rendezett és a feltételes terveket. A 12.13. ábra az állapottér-tervkészítésre mutat be egy egyszerű megközelítést. A tervkészítő ágens egy céllal indít, és kiinduló tervet készít ennek eléréséhez. Ellentétben más tervkészítő ágensekkel, az újratervező ágens követi mind a hátralevő, még végrehajtatlan *terv* részletet, valamint a teljes eredeti tervet *teljes_terv*-et. **Cselekvésmonitorozás**t (**action monitoring**) használ: a *terv* következő cselekvésének végrehajtása előtt az ágens megvizsgálja az érzékelőit, hogy megbizonyosodjon arról, hogy a terv előfeltételei nem váltak-e váratlanul kielégítetlenné. Ha igen, akkor az ágens egy cselekvéssorozat újratervezésével megpróbál a *teljes_terv* egy pontjára visszajutni.

12.13. ábra - Egy cselekvésmonitorozó és -újratervező ágens. Ez szubrutinként a TERVKÉSZÍTŐ teljes állapottér tervkészítő algoritmust használja. Ha a következő cselekvés előfeltételei nem teljesülnek, az ágens a *teljes_terv* lehetséges *p* pontjain iterál olyat keresve, amelyhez a TERVKÉSZÍTŐ útvonalat tud tervezni. Ezt az útvonalat hívjuk *javításnak*. Ha a TERVKÉSZÍTŐ sikeres a javításban, az új terv készítéséhez összefűzi a javítást és a terv *p* utáni részét. Az ágens ezután a terv első lépését adja vissza.

```
function ÚJRATERVEZŐ-ÁGENS(érzékelés) returns egy cselekvés
  static: TB, egy tudásbázis (cselekvésleírásokat tartalmaz)
          terv, egy terv, kezdetben[]
          teljes_terv, egy terv, kezdetben[]
          cél, egy cél

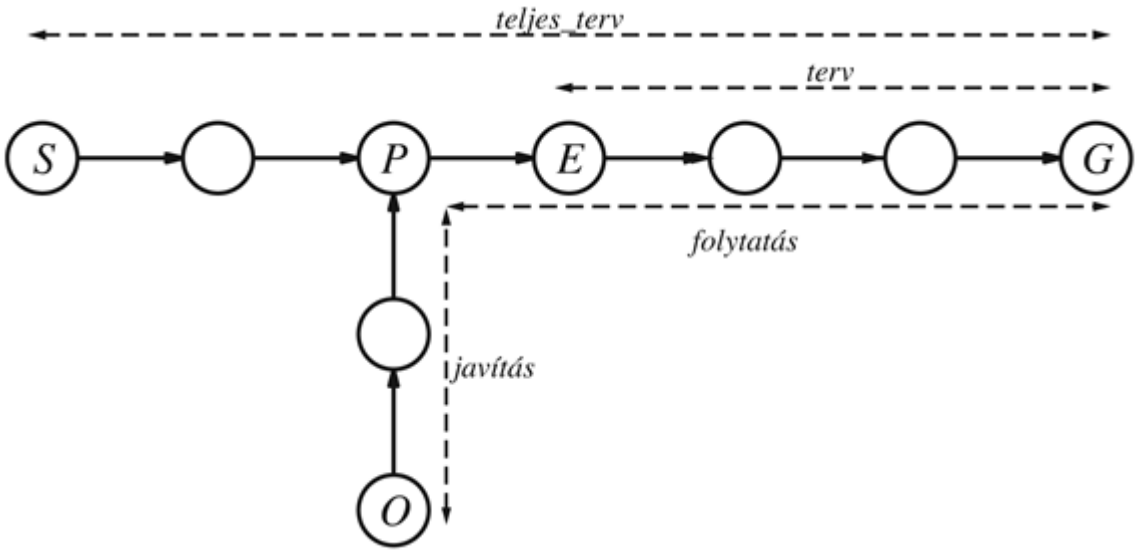
  ÉRTESÍT(TB, ÉRZÉKELŐ-MONDAT-KÉSZÍTÉS(érzékelés, t))
  aktuális ← LEÍRÁS-KÉSZÍTÉS(TB, t)
  if terv = [] then
      teljes_terv ← terv ← TERVKÉSZÍTŐ(aktuális, cél, TB)
  if ELŐFELTÉTELEK(ELŐSZÖR(terv)) aktuálisan nem igaz a TB-ben then
      jelöltek ← RENDEZÉS(teljes_terv, az aktuális-tól vett távolság szerint rendezve)
      keres állapot s a jelöltek amelyekre
          kudarc ≠ javítás ← TERVKÉSZÍTŐ(aktuális, s, TB)
      folytatás ← a teljes_terv s-ből induló farokrésze
      teljes_terv ← terv ← HOZZÁFŰZ (javítás, folytatás)
return RRT(terv)
```

A 12.14. ábra a folyamat sematikus illusztrációja. Az újratervező észreveszi, hogy a *terv* első cselekvésének előfeltételeit az aktuális állapot nem elégíti ki. Ezután meghívja a tervkészítőt, hogy készítsen egy új *javítás*-nak nevezett részterv, amely az aktuális állapotból a *teljes_terv* valamely *s* állapotába vezet vissza. Ebben a példában az *s* történetesen egylépésnyi visszalépést jelent az aktuális hátralévő *terv*-ből. (Ez az oka, hogy a hátralévő terv helyett a teljes tervet figyeljük.) Általánosságban *s*-et a jelen állapothoz a lehető legközelebbre választjuk. Az új terv a *javítás* és a *teljes_terv s*-től hátralevő része — amit *folytatás*-nak nevezünk — összefűzéséből adódik, mellyel az ágens készen áll a végrehajtás folytatására.

Térjünk vissza az asztal és a szék azonos színűre festésének problémájához, ez alkalommal újratervezéssel. Feltételezzük a teljesen megfigyelhető környezetet. A kiinduló állapotban a szék kék, az asztal zöld, valamint 1 doboz kék és 1 doboz zöld festék áll rendelkezésünkre. Ez a következő probléma definícióhoz vezet:

Created by XMLmind XSL-FO Converter.

12.14. ábra - A végrehajtás előtt a tervkészítő egy *teljes_terv*-nek nevezett tervet készít, mely *S*-ből *G*-be vezet. Az ágens az *E*-vel jelölt pontig végrehajtja a tervet. A *terv* hátralevő részének végrehajtása előtt a szokásos módon ellenőrzi az előfeltételeket, és azt találja, hogy valójában az *O* állapotban van, és nem az *E*-ben. Ezután meghívja a tervkészítő algoritmust, hogy készítsen egy *javítás*-t, mely egy terv az *O*-ból az eredeti *teljes_terv* egy *P* pontjához. Az új *terv* ezután a *javítás* és a *folytatás* (az eredeti *teljes_terv* maradék része) összefűzéséből adódik.




```
Kiindulás(Színe(Szék,     Kék)     ∧
Színe(Asztal, Zöld)

  ∧    SzíntTartalmaz(BC,    Kék)    ∧
  FestékDoboz (BC)

  ∧   SzíntTartalmaz(RC,   Piros)   ∧
  FestékDoboz (RC)

Cél(Színe(Szék,  x)  ∧  Színe(Asztal,
x))

Cselekvés(Fest(tárgy, szín),

  Előfeltétel:VanFesték(szín)

  Következmény:Színe(tárgy, szín))

Cselekvés(Nyit(doboz),

  Előfeltétel:FestékDoboz(doboz)    ∧
  Színe(doboz, szín)

  Következmény:Színe(szín))
```

Az ágens TERVKÉSZÍTŐ-jének a következő tervvel kellene előállnia:

```
[Indít; Nyit(BC); Fest(Asztal, Kék); Befejez]
```

Az ágens most készen áll a terv végrehajtására. Tegyük fel, hogy minden jól megy, és az ágens kinyitja a kék festéket és az asztalra keni. Az előző alfejezet ágensei a terv lépésének befejezése után győzelmet kiáltanának. A végrehajtás-monitorozó ágensnek ellenben először ellenőriznie kell a *Befejez* lépés előfeltételét, ami azt

Created by XMLmind XSL-FO Converter.

mondja, hogy a két bútornak azonos színűnek kell lennie. Tegyük fel, hogy az ágens azt érzékeli, hogy a bútorok nem azonos színűek, mivel a festés hiányossága miatt az asztalon egy zöld folt maradt. Az ágensnek ezután keresnie kell egy célpontot a *teljes_terv*-ben, amelyet megcélozhat, és egy javító cselekvéssorozatot kell kidolgoznia, hogy ide jusson. Az ágens észreveszi, hogy az aktuális állapot azonos a *Fest* cselekvés előfeltételeivel, így a *javító* egy üres sorozat, és a *terv* azonos az épp végrehajtott [*Fest, Befejez*] sorozattal. Ezzel az új tervvel a monitorozás végrehajtása folytatódik, és a *Fest* cselekvés újra végrehajtásra kerül. Ez a viselkedés addig ismétlődik, amíg az asztalt teljesen lefestettnek nem érzékeljük. Vegyük észre, hogy a ciklus a terv–végrehajtás–újratervezés folyamat során alakult ki, nem pedig egy explicit ciklus az eredeti tervben.

A cselekvésmonitorozás a végrehajtás-monitorozás nagyon egyszerű módszere, de néha kevésbé intelligens viselkedéshez vezethet. Például tegyük fel, hogy az ágens olyan tervet készít, amelyben az asztalt és a széket is pirosra festi. Ekkor kinyitja a piros festéket, és azt találja, hogy nincs elegendő festék a székhez. A cselekvésmonitorozás nem érzékelné ezt a sikertelenséget, csak *miután* a széket befestettük, amikor is a *VanFesték*(*Piros*) hamissá válik. Amire valójában szükségünk van, az az, hogy akkor érzékeljük a hibát, amikor olyan állapotba jutunk, ahonnan a terv hátralevő része már nem működik. A **tervmonitorozás** (**plan monitoring**) ezt úgy éri el, hogy a teljes hátralevő tervben szereplő előfeltételek sikerességét ellenőrzi, azaz a terv minden lépésének előfeltételeit, kivéve azokat, amelyeket a hátralévő terv egy másik lépésével ér el. A tervmonitorozás a hibás terv végrehajtását a lehető leghamarabb megszakítja, azaz nem várja meg, míg a hiba valójában fellép.[13] Néhány esetben ez az ágenst a teljes bukástól mentheti meg, amikor a hibás terv egy olyan zsákutcába vezetne, melyből a cél már elérhetetlen.

Elég kézenfekvő a tervkészítő algoritmus módosítása, hogy az a terv minden pontján tartalmazza a hátralevő terv sikeréhez szükséges előfeltételeket. Ha a tervmonitorozást kiterjesztjük, hogy az aktuális pont helyett az összes jövőbeni pontra ellenőrizze, hogy az aktuális állapot kielégíti-e a terv előfeltételeit, akkor a tervmonitorozás kihasználhatja a **szerencsés rábukkanás**t (**serendipity**), azaz a véletlenszerű sikert. Ha valaki arra jártában pirosra festi az asztalt ugyanabban az időben, amikor az ágens a széket festi pirosra, akkor a végső terv előfeltételei teljesülnek (a cél teljesül), és az ágens korábban megpihenhet.

Eddig a monitorozást és újratervezést a teljesen megfigyelhető környezetek esetére mutattuk be. A részlegesen megfigyelhető környezetek esetén jóval komplikáltabb problémák léphetnek fel. Először is olyan hibák léphetnek fel, amelyet az ágens nem tud érzékelni. Másodszor „az előfeltételek ellenőrzése” érzékelő cselekvések végrehajtását teheti szükségessé, amelyet tervezni kell vagy a tervkészítési időben, ami a feltételes tervkészítéshez vezet vissza, vagy a végrehajtási időben. A legrosszabb esetben az érzékelési cselekvések végrehajtása összetett tervet igényel, melyhez monitorozás, így további érzékelési cselekvések szükségesek. És ez így megy tovább. Ha az ágens ragaszkodik minden előfeltétel ellenőrzéséhez, akkor elképzelhető, hogy sohasem jut hozzá, hogy valójában *tegyen* is valamit. Az ágensnek csak a fontos változók ellenőrzését kellene szem előtt tartani, melyek jó eséllyel okoznak hibát, és nem túl drága a megfigyelésük. Ez lehetőséget ad az ágensnek, hogy megfelelőképpen reagáljon a fontos fenyegetésekre, de ne vesztegessen időt annak ellenőrizgetésére, hogy összeomlik-e az ég.

Miután bemutattuk a monitorozás és újratervezés módszerét, meg kell kérdeznünk: „Működik?” Ez meglepően becsapós kérdés. Ha a kérdést úgy értelmezzük, hogy „Garantálni tudjuk-e, hogy az ágens mindig, még nem korlátos nemdeterminisztikusság esetén is célt ér?”, akkor a válasz nem, hiszen az ágens figyelmetlenségből zsákutcába juthat, csakúgy, mint a 4.5. alfejezetben bemutatott online keresés. Például lehet, hogy a porszívóágens nem tudja, hogy az elemei kifogyhatnak. Zárjuk ki a zsákutcákat, azaz tételezzük fel, hogy az ágens olyan tervet tud létrehozni, amelyben a célt a környezet *bármely* állapotából eléri. Ha feltételezzük, hogy a környezet valóban nemdeterminisztikus, abban az értelemben, hogy egy ilyen tervnek egy adott végrehajtása esetén mindig van *valamekkora* esélye a sikerre, akkor az ágens szükségszerűen célt ér. Az újratervező ágens képességei ezért megfelelnek a feltételes tervkészítő ágens képességeinek. Valójában módosíthatjuk a feltételes tervkészítőt, hogy csak részleges megoldást készítsen, amely „**if** <*teszt*> **then** *terv_A* **else** *újratervez*” formájú lépéseket tartalmaz. Az említett korlátozásokkal egy ilyen terv megfelelő megoldása lehet az eredeti problémának, és elkészítése szintén sokkal olcsóbb lehet, mint a teljes feltételes tervnek.

Hiba akkor lép fel, ha az ágens ismételt kísérletei is sikertelenek a cél elérésében, azaz valamilyen, számára ismeretlen előfeltétel vagy következmény blokkolja. Például ha az ágens rossz kulcskártyát kap a

[13] A tervmonitorozás okosabbá teszi az ágensünket egy ganajtúró bogárnál (lásd 2.3. szakasz - Mindentudás, tanulás és autonómia részben). Az ágensünk észrevenné, hogy a ganaj már nincs a lábainál, újratervezne, hogy szerezzen egy másikat, és azzal zárná le a lyukat.

Created by XMLmind XSL-FO Converter.

hotelszobájához, akkor nincs az a beillesztési és eltávolítási próbálkozás, ami kinyitná az ajtót.[14] Egy megoldás, hogy ugyanazon terv ismételgetése helyett véletlenszerűen választunk egy lehetséges javítóterv halmazból. Ebben az esetben egy új kulcskártya kérése a recepcióról hasznos alternatív javító terv. Mivel nem biztos, hogy az ágens képes a valóban nemdeterminisztikus esetet és a hiábavalóság esetét megkülönböztetni, egy kis változatosság a javításban általánosságban is jó ötlet.

A helytelen cselekvés problémájára egy másik megoldás a **tanulás**. Néhány próbálkozás után a tanuló ágensnek képesnek kell lennie a cselekvésleírás módosítására, hogy az igényelje, hogy a kulcs nyissa az ajtót. Ezen a ponton az újratervező automatikusan egy új tervvel áll elő, mint egy új kulcs megszerzése. A 21. fejezet ezt a fajta tanulást mutatja be.

Az újratervező ágensnek még ezekkel a lehetséges javításokkal is vannak további hátrányai. Nem hatékony a valós idejű környezetekben, és nincs korlát az újratervezési időre, ezáltal nincs korlát a cselekvésről való döntés idejére sem. Szintén képtelen saját célokat kialakítani vagy a meglévő célokhoz új célokat fogadni, azaz nem lehet hosszú életű egy összetett környezetben. Ezeket a hátrányokat kezeljük a következő fejezetben.

6. Folytonos tervkészítés

Ebben az alfejezetben olyan ágenst tervezünk, amely meghatározatlanul sokáig kitart egy környezetben. Így ez nem egy „problémamegoldó”, ami egyetlen célt szeretne elérni és ezt tervezi, és addig cselekszik, amíg a célt el nem éri, hanem folyamatosan változó célok, tervkészítési és cselekvési fázisok sorozatát éli meg. Ahelyett hogy a tervkészítőt és a cselekvésmonitorozót mint különálló, de egymásnak eredményeket szolgáltató folyamatokat tekintetnénk, a **folytonos tervkészítő ágens**ben (**continuous planning agent**) ezeket egyetlen folyamatként képzeljük el.

Az ágensre mindig úgy gondolunk, mint ami *része egy terv* végrehajtásának, az élete nagy tervének. Cselekvései tartalmazzák a végrehajtásra készen álló terv lépéseinek végrehajtását, a nyitott előfeltételek kielégítésére vagy a konfliktusok feloldására szolgáló tervfinomítást, a végrehajtás során nyert új információk fényében a terv módosítását. Nyilvánvaló, hogy amikor először új célt formál, az ágensnek nincsenek végrehajtásra kész cselekvései, ezért egy kis időt a részleges terv elkészítésével tölt. Nagyon is lehetséges azonban, hogy az ágens a terv teljes befejezése előtt elkezdi a végrehajtást, különösen ha független részcélokat kell elérni. A folytonos tervkészítő ágens folyamatosan monitorozza a világot, az új megfigyelések alapján módosítja ennek modelljét, még akkor is, ha elképzelései még mindig változnak.

Először egy példán haladunk végig, majd bemutatjuk az ágens programját, amelyet `FOLYTONOS-RRT-ÁGENS`-nek nevezünk, mivel a tervezett cselekvések reprezentálására részben rendezett terveket használ. Hogy a leírást egyszerűsítsük, teljesen megfigyelhető környezetet feltételezünk. Ugyanezen technikák kiterjeszthetők a részben megfigyelhető esetre is.

A példánk a kockavilág problémakörből származik (lásd 11.1. alfejezet). A kiinduló állapot a 12.15. (a) ábrán látható. A *Mozgat*(*x*, *y*) cselekvésre van szükségünk, ami az *x* kockát az *y* kockára mozgatja, amennyiben mindketten szabadok. A cselekvés sémája:

Cselekvés(*Mozgat*(*x*, *y*),

Előfeltétel:*Tiszta*(*x*) ∧ *Tiszta*(*y*) ∧ *Rajta*(*x*, *z*)

Következmény:*Rajta*(*x*, *y*) ∧ *Tiszta*(*z*) ∧ ¬*Rajta*(*x*, *z*) ∧ ¬*Tiszta*(*y*)

Az ágensnek először egy célt kell megfogalmaznia. Itt nem tárgyaljuk a célkialakítást, helyette feltételezzük, hogy az ágensnek valahogy megmondták (vagy saját maga eldöntötte), hogy a *Rajta*(*C*, *D*) ∧ *Rajta*(*D*, *B*) célt kell elérnie. Az ágens ehhez a célhoz kezdi a tervezést. Ellentétben minden másik ágensünkkel, amelyek lekapcsolják érzékelőiket, amíg a tervkészítő egy teljes megoldást nem ad a problémára, a folytonos tervkészítő ágens inkrementálisan épít egy tervet, ahol a terv minden bővítése egy korlátos időtartamot vehet igénybe. Minden bővítés után az ágens egy *NoOp* cselekvést ad vissza, és ellenőrzi az érzékeléseit. Feltételezzük, hogy az érzékelt adatok nem változnak, és az ágens gyorsan megalkotja a 12.16. ábrán látható tervet. Vegyük észre, hogy bár mindkét cselekvés előfeltételeit teljesíti az *Indít* állapot, van egy követési megkötés, ami a *Mozgat*(*D*,

[14] A tervjavítás hatástalan ismétlése pontosan megfelel a szöcskeölő darázs viselkedésének (lásd 2.3. szakasz - Mindentudás, tanulás és autonómia részben).

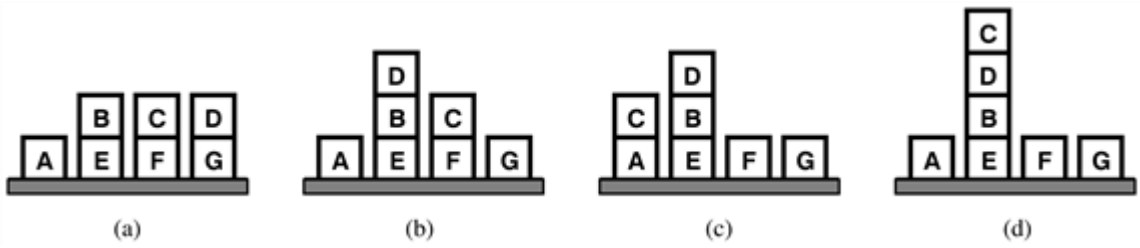
Created by XMLmind XSL-FO Converter.

B) cselekvést a *Mozgat*(*C, D*) elé helyezi. Erre azért van szükség, hogy a *Tiszta*(*D*) igaz maradjon, amíg a *Mozgat*(*D, B*) befejeződik. A folytonos tervkészítési folyamat alatt az *Indít* mindig az aktuális állapot címkéje. Az ágens minden cselekvés után módosítja az állapotot.

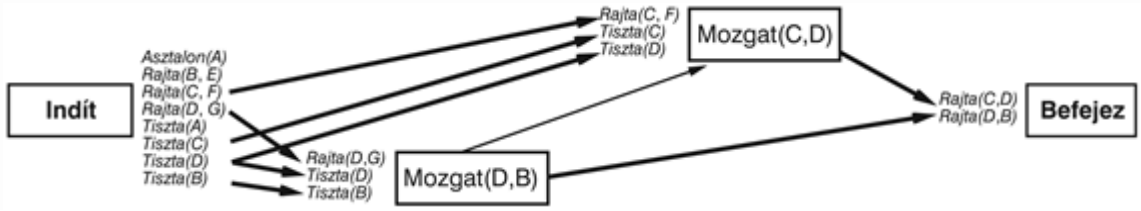
A terv készen áll a végrehajtásra, de még mielőtt az ágens cselekedhetne, a környezet közbeszól. Egy külső ágens (talán az ágens tanítója vált türelmetlenné) a (d) kockát a (b)-re helyezi, a világ most a 12.15. ábra (b) állapotának felel meg. Az ágens ezt érzékeli, észreveszi, hogy a *Tiszta*(*B*) és a *Rajta*(*D, G*) már nem igaz az aktuális állapotban, és ennek megfelelően módosítja az aktuális állapot modelljét. Az okozati kapcsolatok, amelyek a *Mozgat*(*D, B*) cselekvés *Tiszta*(*B*) és *Rajta*(*D, G*) előfeltételeit adták, már nem helyesek, el kell távolítani őket a tervből. Az új tervet a 12.17. ábra mutatja. Minden pillanatban az *Indít* az aktuális állapotot reprezentálja, így ez az *Indít* különbözik az előző ábra hasonló állapotától. Vegyük észre, hogy a terv most nem teljes: a *Mozgat*(*D, B*) előfeltételei közül kettő nyitott, és a *Rajta*(*D, y*) most nem példányosított, mert már nincs okunk feltételezni, hogy a lépés a *G*-ről indul.

Az ágens most kihasználhatja a „segítő” közbeavatkozást, észrevéve, hogy a $Mozgat(D, B) \xrightarrow{Rajta(D,B)} Befejez$ okozati kapcsolat helyettesíthető az *Indít*-ból a *Befejez* állapotba mutató direkt kapcsolattal. Ezt a folyamatot az okozati kapcsolat **kiterjesztés**ének (**extension**) nevezzük, és akkor alkalmazzuk, amikor egy feltétel egy későbbi helyett egy korábbi lépéssel biztosítható, új ütközés okozása nélkül.

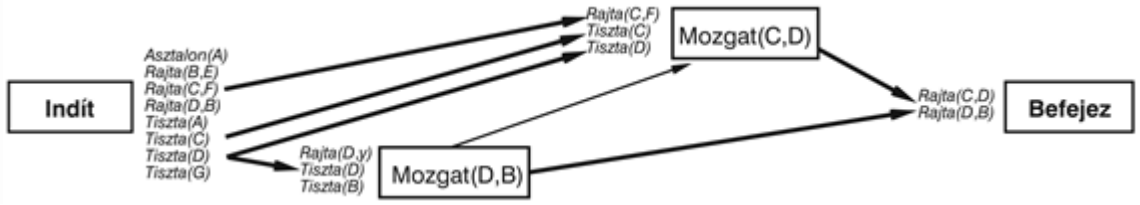
12.15. ábra - Az állapotok sora, amint a folytonos tervkészítő ágens megpróbálja a (d)-nek megfelelő *Rajta*(*C, D*) ∧ *Rajta*(*D, B*) célállapotot elérni. A kiinduló állapot az (a). A (b) állapotnál egy másik ágens közbelépett és (d)-t (b)-re helyezte. A (c) állapotban az ágens a *Mozgat*(*C, D*) cselekvést hajtotta végre, ami sikertelen volt, (c)-t az (a)-ra ejtette. Újrapróbálja a *Mozgat*(*C, D*) cselekvést, amivel eléri a (d) célállapotot.



12.16. ábra - A folytonos tervkészítő ágens által készített kiinduló terv. A terv egyelőre nem különbözik egy átlagos részben rendezett tervkészítő által készítettől.



12.17. ábra - Miután valaki más elmozdította a *D* kockát a *B*-re, a *Tiszta*(*B*) és a *Rajta*(*D, G*) biztosítására szolgáló kapcsolatokat eldobjuk, ami ezt a tervet eredményezi



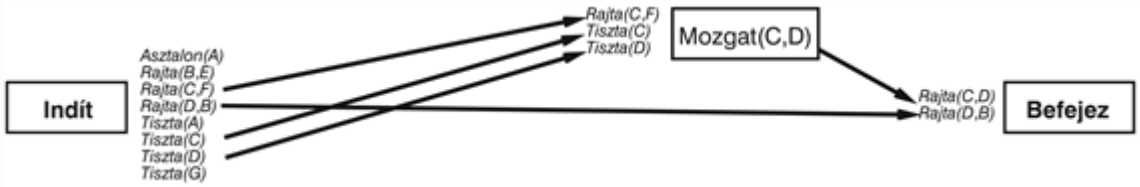
Miután a régi *Mozgat*(*D, B*) és *Befejez* közötti okozati kapcsolatot eltávolítottuk, a *Mozgat*(*D, B*) továbbiakban nem forrása egyetlen okozati kapcsolatnak sem. Ez most egy **redundáns lépés** (**redundant step**). Minden redundáns lépést és bármely ezt biztosító kapcsolatot kitörlünk a tervből. Ez a 12.18. ábrán látható tervet eredményezi.

Created by XMLmind XSL-FO Converter.

Most a *Mozgat*(*C, D*) lépés készen áll a végrehajtásra, hiszen az *Indít* lépés kielégíti minden előfeltételét, nincs szükség más megelőző lépésre, és nem ütközik a terv egyetlen másik kapcsolatával sem. A lépést eltávolítjuk a tervből, és végrehajtjuk. Sajnos az ágens ügyetlen, és a *C*-t az *A*-ra ejti a *B* helyett, ami a 12.15. ábra (c) állapotát eredményezi. Az új tervállapotot a 12.19. ábra mutatja. Vegyük észre, hogy bár most nincsenek cselekvések a tervben, a *Befejez* lépésnek még mindig van egy nyitott feltétele.

Az ágens úgy dönt, hogy a nyitott feltételhez készít tervet. Újra a *Mozgat*(*C, D*) elégíti ki a célfeltételt. Ennek előfeltételeit az *Indít* lépésből induló új okozati kapcsolatok teljesítik. Az új tervet a 12.20. ábra mutatja.

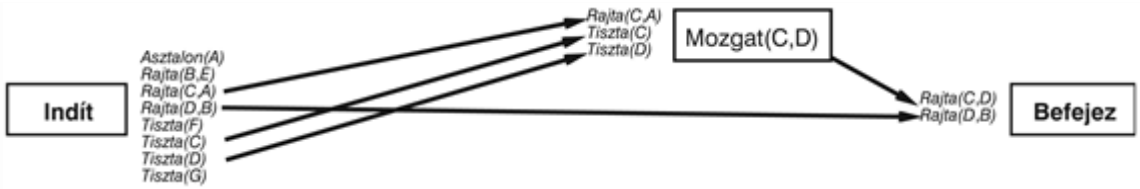
12.18. ábra - A *Mozgat*(*C, D*) által létrehozott kapcsolatot helyettesítettük az *Indít* kapcsolatával, a most már redundáns *Mozgat*(*D, B*) lépést pedig elhagytuk



12.19. ábra - A *Mozgat*(*C, D*) végrehajtása és a tervből való eltávolítása után az *Indít* lépés következményei azt a tényt tükrözik, hogy a *C* a várt *D* helyett az *A* kockára került. A *Rajta*(*C, D*) cél előfeltétel még minding nyitott.



12.20. ábra - A nyitott előfeltételt a *Mozgat*(*C, D*) ismételt hozzáadásával oldjuk fel. Vegyük észre az előfeltételekhez szükséges új kapcsolatokat.



A *Mozgat*(*C, D*) újra készen áll a végrehajtásra. Ez alkalommal működik, ami a célállapotot eredményezi, melyet a 12.15. ábra (d) része mutat. Miután egy lépést elhagyunk a tervből, a *Rajta*(*C, D*) célfeltétel újra nyitottá válik. Mivel az *Indít* lépést frissítettük, hogy tükrözze az új állapotot, a célfeltétel azonnal kielégíthető egy kapcsolattal az *Indít* lépésből. Ez az események normális folyása, ha egy cselekvés sikeres. A végleges tervállapotot a 12.21. ábra mutatja. Mivel az *Indít* lépés az összes célfeltételt kielégíti, és nincsenek hátralevő cselekvések, az ágens szabadon eltávolíthatja a *Befejez* állapot céljait, és új célokat fogalmazhat meg.

12.21. ábra - A *Mozgat*(*C, D*) végrehajtása és tervből eltávolítása után a *Rajta*(*C, D*) megmaradt nyitott feltételt az *Indít* lépésből induló okozati kapcsolat hozzáadásával oldjuk fel. A terv most teljes.



Ebből a példából látható, hogy a folytonos tervkészítés nagyban hasonlít a részben rendezett tervkészítéshez. Minden iterációban az algoritmus talál valami **tervsérülés**nek (**plan flaw**) nevezett javítanivalót a tervben, és megjavítja. A részben rendezett tervkészítő algoritmus egy sérüléseltávolító algoritmusnak tekinthető, ahol a

Created by XMLmind XSL-FO Converter.

kétféle sérülés a nyílt előfeltétel és az okozati ütközés lehet. A folytonos tervkészítő ágens másrészről sérülések egy jóval szélesebb körét kezeli:

- *Hiányzó cél:* az ágens eldöntheti, új célt vagy célokat ad a *Befejez* állapothoz. (Folytonos tervkészítésben értelmesebb lenne a *Befejez* állapot nevét *Örökkévalóság*-ra és az *Indít*-ét *Aktuális*-ra változtatni, de ragaszkodunk a hagyományokhoz.)

- *Nyílt előfeltétel:* egy okozati kapcsolatot ad a nyílt előfeltételhez, egy meglévő vagy egy új cselekvést választva (mint a részben rendezett tervkészítőben).

- *Okozati ütközések:* adott az $A \xrightarrow{p} B$ ok-okozati kapcsolat és a $\neg p$ következményű *C* cselekvés. Válasszunk az ütközés feloldásához egy rendezési megkötést vagy egy változómegkötést (mint a részben rendezett tervkészítésben).

- *Nem támogatott kapcsolat:* ha van egy $Start \xrightarrow{p} A$ ok-okozati kapcsolat, ahol a *p* nem igaz az *Indít* állapotban, akkor távolítsuk el a kapcsolatot. (Ez megóv minket attól, hogy végrehajtsunk egy olyan cselekvést, melynek előfeltételei hamisak.)

- *Redundáns cselekvések:* ha egy *A* cselekvés nem eredményez ok-okozati kapcsolatokat, akkor távolítsuk el a kapcsolataival együtt. (Ez lehetővé teszi számunkra, hogy kihasználjuk a szerencsés helyzeteket.)

- *Végrehajtatlan cselekvés:* ha egy (a *Befejez*-től eltérő) *A* cselekvésnek, melynek az előfeltételeit az *Indít* kielégíti, nincs (az *Indít*-tól eltérő) cselekvés elé sorolva, és nem ütközik okozati kapcsolatokkal, akkor távolítsuk el *A*-t és ok-okozati kapcsolatait, majd adjuk vissza mint a végrehajtandó cselekvést.

- *Szükségtelen történelmi cél:* ha nincsenek nyílt előfeltételek és nincsenek a tervben cselekvések (azaz minden ok-okozati kapcsolat közvetlen az *Indít*-ból a *Befejez* állapotba mutat), akkor elértük az aktuális célhalmazt. Távolítsuk el a célokat és az ezekbe mutató kapcsolatokat, hogy új célokat tegyünk lehetővé.

A 12.22. ábra a `Folytonos-RRT-Ágens`-t mutatja be. Ennek ciklusa az „érzékelés, eltávolítás a folyamatból, cselekvés”. Tudásbázisában egy állandó tervet tart, és minden lépésben eltávolít egy folyamatot ebből a tervből. Ezután cselekszik (bár gyakran ez a cselekvés a *NoOp*) és ismétli a ciklust. Ez az ágens az 5. szakasz - Végrehajtás monitorozása és újratervezése részben tárgyalt újratervező ágenshez felsorolt problémák nagy részét képes kezelni. Nevezetesen valós időben cselekszik, kezeli a kedvező helyzeteket, saját célokat fogalmaz meg, és kezeli a váratlan eseményeket, melyek a jövőbeli terveket érintik.

12.22. ábra - A `FOLYTONOS-RRT-ÁGENS`, egy folytonos, részben rendezett tervkészítő ágens. Egy érzékelés után az ágens eltávolít egy lépéssorozatot a folytonosan javított tervéből, és visszaad egy cselekvést. Gyakran számos cselekvéssor-eltávolító tervkészítési lépésre van szükség, mialatt csak a *NoOp*-ot adja vissza, mielőtt készen állna egy valós cselekvés végrehajtására.

```
function FOLYTONOS-RRT-ÁGENS(érzékelés) returns egy cselekvés
    static: terv, egy terv, kezdetben csak Indít, Befejez

    cselekvés ← NoOp (alapértelmezés)
    KÖVETKEZMÉNYEK[Start] = MÓDOSÍTÁS(KÖVETKEZMÉNYEK[Start], érzékelés)
    TÖRÉS-ELTÁVOLÍTÁS(terv) // várhatóan egy cselekvés módosítása
    return cselekvés
```

7. Multiágens tervkészítés

Eddig **egyágenses környezet**ekkel (**single-agent enviroment**s) foglalkoztunk, amelyben az ágensünk egyedül van. Amikor más ágensek is jelen vannak a környezetben, az ágensünk egyszerűen hozzáveheti őket a környezetről alkotott modelljéhez, az alapvető algoritmusainak módosítása nélkül. Sok esetben azonban ez gyenge teljesítményhez vezetne, mert többi ágenssel való foglalkozás nem azonos a környezet kezelésével.

Created by XMLmind XSL-FO Converter.

Nevezetesen, a természet (feltételezhetően) közömbös az ágens szándékait illetően, míg más ágensek nem.[15] Ez az alfejezet ezen kérdések kezelésére a multiágens tervkészítést mutatja be.

A 2. fejezetben láthattuk, hogy a multiágens környezetek lehetnek **együttműködő**k (**cooperative**) vagy **versengő**k (**competitive**). Egy nagyon egyszerű kooperatív példával kezdünk: egy páros teniszcsapat tervkészítésével. Olyan tervek alkothatók, amelyek a csapat mindkét játékosának a cselekvéseit meghatározzák. Az ilyen tervek hatékony elkészítéséhez használható technikákat mutatjuk be. A hatékony tervkészítés hasznos, de nem garantálja a sikert. Az ágenseknek egyet kell érteniük a felhasznált tervben! Ez valamilyen **koordináció**t (**coordination**) feltételez, amit valószínűleg **kommunikáció**val (**communication**) érhetünk el.

7.1. Kooperáció: közös célok és tervek

A páros teniszt játszó ágenscsapatoknak közös céljuk van, a meccs megnyerése, ami különböző részcélokat eredményez. Tegyük fel, hogy a játék egy pontján a közös céljuk az átütött labda visszaadása úgy, hogy legalább egyikőjük a hálót védi. Ezt a gondolatot egy **multiágens tervkészítési** (**multiagent planning**) problémával írhatjuk le, amint azt a 12.23. ábra is mutatja.

12.23. ábra - A páros tenisz probléma. Két ágens együtt játszik, és négy pozíció egyikében lehetnek: [*Bal, Alapvonal*], [*Jobb, Alapvonal*], [*Bal, Háló*] és [*Jobb, Háló*]. A labda visszaadható, ha pontosan egy játékos van a megfelelő helyen.

Ágensek(*A*, *B*)
Kiindulás(*Ott*(*A*, [*Bal*, *Alapvonal*])), *Ott*(*B*, [*Jobb*, *Háló*]) ∧
    Megközelít(*Labda*, [*Jobb*, *Alapvonal*])) ∧ *Partner*(*A*, *B*) ∧ *Partner*(*B*, *A*)
Cél(*Visszaadott*(*Labda*) ∧ *Ott*(*ágens*, [*x*, *Háló*]))
Cselekvés(*Ütés*(*ágens*, *Labda*),
    ELŐFELTÉTEL:*Megközelít*(*Labda*, [*x*, *y*] ∧ *Ott*(*ágens*, [*x*, *y*]) ∧ *Partner*(*ágens*, *partner*)
        ∧ ¬ *Ott*(*partner*, [*x*, *y*])
    KÖVETKEZMÉNY:*Visszaadott*(*Labda*))
Cselekvés(*Menj*(*ágens*, [*x*, *y*]),
    ELŐFELTÉTEL:*Ott*(*ágens*, [*a*, *b*])
    KÖVETKEZMÉNY:*Ott*(*ágens*, [*x*, *y*]) ∧ ¬ *Ott*(*ágens*, [*a*, *b*]))

Ez a jelölésrendszer két újdonságot vezet be. Először az *Ágens*(*A*, *B*) deklarálja, hogy két ágensünk van, *A* és *B*, akik a tervben szerepelnek. (Ebben a problémában a szemben álló játékosokat nem ágensnek vesszük.) Másodszor minden cselekvés az ágenst explicit módon, mint egy paramétert említi, mert pontosan követnünk kell, melyik ágens mit csinál.

A multiágens tervkészítési probléma megoldása egy **együttes terv** (**joint plan**), amely minden ágenshez tartalmaz cselekvéseket. Az együttes terv egy megoldás, ha a célt elérjük azáltal, hogy mindegyik ágens végrehajtja a számára kijelölt cselekvéseket. A következő terv a tenisz probléma egy megoldása :

```
Terv1:

A: [Megy(A, [Jobb, Alapvonal]), Üt(A, Labda)]

B: [NoOp(B), NoOp(B)]
```

Ha mindkét ágens ugyanazzal a tudásbázissal rendelkezne, és ez lenne az egyetlen megoldás, akkor minden rendben lenne; minden ágens meghatározná a megoldást, és együttesen végrehajtanák. Az ágensek számára sajnálatos (és hamarosan látni fogjuk, hogy ez miért sajnálatos), hogy van egy másik terv is, ami az elsőhöz hasonlóan kielégíti a célt:

```
Terv 2:
```

[15] Nem biztos, hogy az Egyesült Királyság lakói, ahol egy pikniknek már pusztán a tervezése is esőt garantál, egyetértenek ezzel.

Created by XMLmind XSL-FO Converter.

```
A: [Megy(A, [Bal, Háló]), NoOp(A)]

B: [Megy(B, [Jobb, Alapvonal]), Üt(B, Labda)]
```

Ha az *A* a 2. tervet választja, és *B* az 1-et, akkor senki sem fogja visszaadni a labdát. Fordítva, ha *A* az 1-et és *B* a 2-t választja, akkor várhatóan összeütköznek egymással, senki nem adja vissza a labdát, és a háló is fedezetlenül marad. Ezért a helyes együttes terv létezése nem jelenti, hogy a célt el is érjük. Az ágenseknek szükségük van egy **koordinációs** (**coordination**) mechanizmusra, hogy *ugyanazt* az együttes tervet érjék el, sőt mi több, az ágensek közös tudással is kell rendelkezzenek (lásd 10. fejezet) arról, hogy melyik együttes tervet hajtják végre.

7.2. Többtestű tervezés

Ez az alfejezet a megfelelő együttes tervek megalkotására fókuszál, elodázva egyelőre a koordinációs kérdéskört. Ezt **többtestű tervkészítés**nek (**multibody plannining**) nevezzük, és tulajdonképpen az a tervkészítési probléma, mellyel egy egyedülálló centralizált ágens áll szemben, mely cselekvéseket diktálhat minden fizikai entitásnak. Egy valóban multiágens esetben ez lehetővé teszi minden ágens számára, hogy felderítse melyek a lehetséges együttes tervek, melyek együttes végrehajtása sikeres lenne.

Megközelítésünkben a többtestű tervkészítést a 11.3. alfejezetben bemutatott részben rendezett tervkészítésre alapozzuk. Az egyszerűség kedvéért teljes megfigyelhetőséget feltételezünk. Van még egy kérdés, ami nem merül fel az egyedülálló ágens esetén: a környezet a továbbiakban nem igazán **statikus** (**static**), mert a többi ágens cselekedhet, amíg egy bizonyos ágens gondolkozik. Ezért foglalkoznunk kell a **szinkronizáció**val (**synchronization**). Az egyszerűség kedvéért feltételezzük, hogy minden cselekvés végrehajtási ideje azonos, illetve hogy az összetett terv minden pontján a cselekvések végrehajtása szimultán történik.

Bármely időpontban minden ágens pontosan egy cselekvést hajt végre (beleértve a *NoOp*(*Üres*) *operátor*t is). Ezen konkurens cselekvések halmazát **együttes cselekvés**nek (**joint action**) nevezzük. A tenisz problémában (7.1. szakasz - Kooperáció: közös célok és tervek részben) például a ⟨ *NoOp*(*A*), *Üt*(*B*, *Labda*)⟩ az *A* és *B* ágenseknek egy együttes cselekvése. Egy együttes terv együttes cselekvések részben rendezett gráfja. Például a tenisz probléma második terve az együttes cselekvések következő sorozatával írható le:

```
⟨Megy(A, [Bal, Háló]), Megy(B, [Jobb, Alapvonal])⟩

⟨NoOp(A), Üt(B, Labda)⟩
```

Végrehajthatnánk a tervkészítést a hagyományos részben rendezett tervkészítő algoritmussal, azt az összes lehetséges együttes cselekvés halmazára futtatva. Az egyetlen probléma a halmaz mérete: 10 cselekvéssel és 5 ágenssel 10^5 együttes cselekvést kapunk. Elég unalmas lenne az összes cselekvés előfeltételét és következményét helyesen definiálni, és nem lenne hatékony egy ilyen nagy halmazzal tervezni.

Egy másik lehetőség az együttes cselekvések implicit definiálása úgy, hogy minden egyes cselekvéshez leírjuk, hogy milyen kölcsönhatásban áll a többi lehetséges cselekvéssel. Ez egyszerűbb lenne, hiszen a legtöbb cselekvés független a legtöbb másiktól, ezért csak azon néhány cselekvést kellene listáznunk, amelyek valójában kölcsönhatásban vannak. Ezt a hagyományos `STRIPS` vagy ADL cselekvésleírások egy új tulajdonsággal való kiterjesztésével tehetjük meg: ez a **konkurens cselekvések listája** (**concurrent action list**). Ez nagyon hasonlít a cselekvés leírásában szereplő előfeltételhez, kivéve, hogy az állapotváltozók leírása helyett ez cselekvéseket ír le, amelyeket egyszerre kell, vagy éppen nem szabad egyszerre végrehajtani. Például a *Találat* cselekvés a következőképpen írható le:

```
Cselekvés(Üt(A, Labda),

Konkurens:¬Üt(B, Labda)

Előfeltétel:Megközelít(Labda, [x, y]) ∧ Ott(A, [x, y])

Következmény:Visszaadott(Labda))
```

Itt egy tiltó egyidejűségi megkötésünk van, azaz az *Üt* cselekvés végrehajtása alatt egy másik ágens nem hajthat végre egy másik *Üt* cselekvést. Hasonlóképpen *megkövetelhetünk* egyidejű cselekvést. Például amikor két ágensre van szükség egy üdítőkkel teli hűtőtáska teniszpályához cipeléséhez. Ezen cselekvés leírása azt fejezi ki, hogy az *A* ágens nem hajthatja végre a *Cipel* cselekvést, hacsak nincs egy másik *B* ágens, aki egyidejűleg egy *Cipel* cselekvést hajt végre ugyanazon hűtőtáskával:

Created by XMLmind XSL-FO Converter.

Cselekvés(*Cipel*(*A*, *hűtőtáska*, *itt*, *ott*),

Konkurens:*Cipel*(*B*, *hűtőtáska*, *itt*, *ott*)

Előfeltétel:*Ott*(*A*, *itt*) ∧ *Ott*(*Hűtőtáska*, *itt*) ∧ *Hűtőtáska*(*hűtőtáska*)

Következmény:*Ott*(*A*, *ott*) ∧ *Ott*(*hűtőtáska*, *ott*)

∧ ¬*Ott*(*A*, *itt*) ∧ ¬*Ott*(*hűtőtáska*, *itt*))

Ezen reprezentáció segítségével a részben rendezett tervkészítőhöz nagyon közeli tervkészítő hozható létre. Mindössze három különbség van:

1. Az $A \prec B$ temporális rendezési relációt kiegészítendő megengedjük az $A = B$-t és az $A \preceq B$ -t, ami „egyidejűséget”, illetve „megelőzőséget vagy egyidejűséget” jelent.

2. Amikor egy új cselekvés egyidejű cselekvéseket igényel, példányosítani kell ezen cselekvéseket a terv új vagy létező cselekvéseinek felhasználásával.

3. A tiltottan egyidejű cselekvések a megkötések újabb forrásai. Minden megkötést fel kell oldani az ütköző cselekvések egymás elé vagy mögé rendelésével.

Ez a reprezentáció a többtörzsű problémakörökhöz alkalmazott, részben rendezett tervkészítővel ekvivalens. Kiterjeszthetnénk ezt a megközelítést az utolsó két fejezet finomításaival – HFH-ek részleges megfigyelhetőség, feltételek, végrehajtás-monitorozás és újratervezés –, de ez túlmutat a könyv célkitűzésein.

7.3. Koordináló mechanizmusok

A legegyszerűbb módszer, amivel ágensek egy csoportja biztosíthatja az egyetértést egy együttes tervben, hogy egy **konvenció**t (**convention**) fogadnak el az együttes cselekvés megkezdése előtt. A konvenció az együttes tervek közötti választásra vonatkozó bármely olyan megkötés, amely túlmutat az alapmegkötésen, amely szerint egy együttes tervnek működni kell, ha minden ágens alkalmazza. Például a „maradj a saját térfeleden” konvenció a páros partnereket a második terv kiválasztására sarkallja, míg az „egy játékos mindig a hálónál marad” konvenció az egyes tervhez vezetné őket. Néhány konvenció, mint az út megfelelő oldalán való vezetés, olyan széles körben elterjedt, hogy ezeket már **társadalmi törvényszerűség**ekként (**social law**s) kezeljük. Az emberi nyelvek szintén konvencióknak tekinthetők.

Az előző alfejezet konvenciói alkalmazási terület specifikusak és a cselekvésleírások megkötéseivel implementálhatók, amelyekkel kizárjuk a konvenciók megszegéseit. Egy általánosabb megközelítés, hogy problémakör-független konvenciókat használjunk. Például ha mindegyik ágens ugyanazon többtestű tervkészítő algoritmust futtatja ugyanazon bemenetekkel, követi azt a konvenciót, miszerint az elsőként megtalált, használható összetett tervet hajtsa végre, miközben biztos benne, hogy a többi ágens ugyanezzel a választással él. Egy sokkal robusztusabb, de jóval költségesebb stratégia lenne az összes összetett terv elkészítése, majd az ebből való választás (mondjuk annak kiválasztása, amelynek leírt reprezentációja ábécé-sorrendben az első).

Konvenciók szintén adódhatnak a folyamatok evolúciós fejlődésével. Például a közösségben élő rovarok nagyon kidolgozott összetett terveket hajtanak végre, melyeket a kolónia egyedeinek közös genetikus összeállítása biztosít. Az azonosságot szintén nyomatékosítja a tény, hogy a konvencióktól való eltérés csökkenti az evolúciós alkalmasságot, így bármely alkalmazható összetett terv stabil egyensúlyi helyzetet eredményezhet. Hasonló megfontolások alkalmazhatók az emberi nyelv fejlődésére, ahol nem az a fontos, hogy egyes személyek milyen nyelvet beszélnek, hanem az a tény, hogy minden személy ugyanazt a nyelvet beszéli.

Egy utolsó példa a madarak gyülekezési viselkedése. Megfelelő szimulációt nyerhetünk, ha minden egyes madár ágens (melyet néha madaroidnak vagy **roid**nak (**boid**) nevezünk) a következő három szabályt követi valamilyen kombináció alapján:

1. Különválás: távolodjunk el a szomszédoktól, amikor túl közel kerülünk;

2. Kohézió: kerüljünk átlagos távolságra a szomszédainktól;

3. Irányultság: kerüljünk a szomszédokhoz képest átlagos orientációba (haladási irányba).

Created by XMLmind XSL-FO Converter.

Ha minden madár azonos szabályrendszert hajt végre, a csapat repülése egy **kialakuló viselkedés**t (**emergent behavior**) mutat, hasonlóan egy nagyjából azonos sűrűségű pszeudomerev test viselkedéséhez, amely nem bomlik fel az idő során. Mint a bogaraknál, itt sincs szükség arra, hogy minden ágens ismerje a teljes összetett tervet, ami a többi ágens cselekvését modellezi.

A konvenciók ahelyett, hogy minden egyes új problémához kifejlesztenénk őket, tipikusan egyedi multiágens tervkészítő problémák egy egész csoportjához készülnek. Ez rugalmatlansághoz és hibához vezethet, amit néha láthatunk, ha a páros tenisz játszmában a labda nagyjából azonos távolságra van a két partner között. Egy alkalmazható konvenció hiányában az ágensek **kommunikál**hatnak (**communication**), hogy egy közös tudást érjenek el az alkalmazható összetett tervből. Például a teniszpáros egy játékosa felkiálthat „Enyém!” vagy „Tiéd!”, hogy jelezze a preferált tervet. A kommunikációs mechanizmusokat nagyobb mélységben a 22. fejezetben tárgyaljuk, ahol megfigyelhetjük, hogy a kommunikáció nem szükségszerűen szóbeli információcserét jelent. Például egy játékos úgy is közölheti az általa előnyben részesített összetett tervét a másikkal, hogy végrehajtja annak kezdő részletét. A teniszjátszma problémánkban, ha az *A* ágens elindul a hálóhoz, akkor a *B* ágens rákényszerül, hogy visszamenjen az alapvonalhoz megütni a labdát, mivel a második terv az egyetlen összetett terv, ami *A* hálóhoz mozgásával kezdődik. Ez az irányítási megközelítés, amelyet gyakran a **terv felismerés**ének (**plan recognition**) nevezünk, akkor működik, ha egyetlen cselekvés (vagy cselekvések rövid sorozata) elegendő, hogy egyértelműen azonosítsuk az összetett tervet.

Azt a terhet, amely biztosítja, hogy az ágensek egy sikeres összetett tervhez jussanak, vagy az ágensek készítőire, vagy magukra az ágensekre helyezhetjük. Az első esetben mielőtt az ágensek tervezni kezdenének, az ágens készítőjének bizonyítania kell, hogy az ágensek szabályai és stratégiái sikeresek lesznek. Az ágensek önmagukban lehetnek reaktívak, amennyiben ez működik az őket körülvevő környezetben, valamint ha nem kell, hogy pontos modelljük legyen a többi ágensről. Az utóbbi esetben az ágensek megfontoltak: bizonyítaniuk vagy demonstrálniuk kell, hogy a tervük hatékony lesz, figyelembe véve a többi ágens megfontolásait. Például egy *A* és *B* logikai ágenst tartalmazó környezetben mindketten rendelkezhetnek a következő definícióval:

$$\forall p, s \; Megfelelő(p, s) \Leftrightarrow KözösTudás(\{A, B\}, Eléri(p, s, Cél))$$

Ez azt mondja, hogy bármely *s* szituációban, a *p* terv egy alkalmazható összetett terv, ha az ágensek közösen tudják, hogy a *p* eléri a célt. További axiómákra van szükségünk, hogy biztosítsuk egy adott összetett terv végrehajtására vonatkozó **közös szándék** (**joint intention**) együttes ismeretét. Az ágensek csak ebben az esetben kezdhetnek cselekedni.

7.4. Versengés

A többágenses környezeteknek nem mindegyike áll együttműködő ágensekből. Az egymásnak ellentmondó működésű ágensek **verseny**ben (**competition**) vannak egymással. Erre egy példát jelentenek a két játékost igénylő, de zérus összegű játékok, mint amilyen a sakk. A 6. fejezetben láthattuk, hogy a sakkozó ágensnek figyelembe kell vennie az ellenfél következő néhány lehetséges lépését is. Azaz egy versengő környezetben levő ágensnek (a) fel kell ismernie, hogy vannak más ágensek, (b) meg kell határoznia a többi ágens lehetséges terveit, (c) meg kell határoznia, hogy a többi ágens hogyan tervezi a saját terveivel való kölcsönhatást, és (d) el kell döntenie, hogy ezek fényében mi a legjobb cselekvés. A versengés, hasonlóan az együttműködéshez, a többi ágens tervének modelljét igényli. Másrészről, egy versengő környezetben nincs elhatározás egy összetett terv mellett.

A 12.4. alfejezet felvázolta a játékok és a feltételes tervkészítési problémák közötti analógiát. A 12.10. ábrán szereplő feltételes tervkészítő algoritmus olyan tervet készít, amely a környezetről a legrosszabb esetet feltételezve is működik, így olyan versenyhelyzetekben is alkalmazható, ahol az ágenst csak a siker vagy a bukás érdekli. Amikor az ágens és ellenfelei a terv *költségeivel* is foglalkoznak, akkor a minimax megoldás a helyénvaló. Eddig nagyon kevés munkát fordítottak a minimax módszer olyan módszerekkel való kombinálására, mint a részben rendezett tervkészítés vagy a HFH-tervkészítés, ami túlmutat a 6. fejezetben használt állapottér-keresési modellen. A versengés kérdésére a játékelmélettel foglalkozó 17.6. alfejezetben még visszatérünk.

8. Összefoglalás

Created by XMLmind XSL-FO Converter.

Ez a fejezet a valódi világban alkalmazott tervkészítés és cselekvés nehézségeivel foglalkozott. Összefoglaljuk a fő megállapításokat.

- A legtöbb cselekvés **erőforrás**okat (**resource**s) használ, például pénzt, benzint vagy nyersanyagokat. Ezen erőforrásokat készletekre vonatkozó mérőszámokként célszerű kezelni, mintsem hogy megpróbáljunk következtetni mondjuk a világ minden egyes pénzérméjéről és papírpénzéről. A cselekvések generálhatnak és elfogyaszthatnak erőforrásokat, és rendszerint olcsó és hatékony, ha a további finomítások végrehajtása előtt a részleges terveket ellenőrizzük, hogy az erőforrásokra vonatozó korlátozásokat kielégítik-e.

- Az idő az egyik legfontosabb erőforrás. Ez speciális ütemező algoritmusokkal kezelhető, vagy az ütemezést integrálhatjuk a tervkészítésbe.

- A **hierarchikus feladat háló** (**hierarchical task network**) (**HFH**) tervkészítés lehetővé teszi az ágens számára, hogy dekompozíciós szabályok formájában tanácsokat fogadjon el a feladatkör tervezőjétől. Ez lehetővé teszi, hogy nagyon nagy terveket készítsünk, amelyekre számos valódivilág-beli alkalmazáshoz szükség van.

- A hagyományos tervkészítő algoritmusok teljes és helyes információt és determinisztikus, teljesen megfigyelhető környezetet feltételeznek. A legtöbb problémakörre nem igaz ez a feltételezés.

- A hiányos információ kezelhető azáltal, hogy érzékelő cselekvéseket tervezünk, amelyekkel megszerezzük a szükséges információt. A **feltételes terv**ek (**conditional plan**s) lehetővé teszik az ágens számára, hogy a végrehajtás során érzékelje a környezetét, hogy eldönthesse, mely ágát követi a tervnek. Néhány esetben **érzékelőmentes** (**sensorless**) vagy **alkalmazkodó tervkészítés** (**conformant planning**) használható az érzékelést nem igénylő terv elkészítéséhez. Mind az érzékelőmentes, mind pedig a feltételes tervek előállíthatók a **hiedelmi állapot**ok (**belief state**s) terében történő kereséssel.

- A helytelen információk kielégítetlen előfeltételeket eredményeznek a cselekvések és a tervek számára. A **végrehajtás-monitorozás** (**execution monitoring**) a terv sikeres befejezése érdekében érzékeli az előfeltételek megszegéseit.

- Az **újratervező ágens** (**replanning agent**) végrehajtás-monitorozást használ és szükség esetén javításokat szúr be.

- A **folytonos tervkészítő** (**continuous planning**) ágens előrehaladása során új célokat alkot és valós időben reagál.

- **Multiágens** (**multiagent**) tervezésre van szükség, amikor a környezetben más ágensek is találhatók, melyekkel együttműködni vagy versengeni kell, vagy amelyeket koordinálni kell. A **többtestű** (**multibody**) tervkészítés az együttes cselekvésleírások hatékony dekompozícióját felhasználva együttes terveket alkot, de ki kell egészíteni valamilyen koordinációval, amennyiben két együttműködő ágensnek egyet kell érteni abban, hogy melyik együttes tervet hajtsák végre.

8.1. Irodalmi és történeti megjegyzések

A folytonos idővel való tervkészítést elsőként DEVISER vetette fel (Vere, 1983). Az időnek a tervekben való szisztematikus reprezentációjával Dean és társai dolgoztak a FORBIN rendszerben (Dean és társai, 1990). A NONLIN+ (Tate és Whiter, 1984) és a SIPE (Wilkins, 1988; 1990) kezelni tudta a korlátos erőforrások különböző tervlépésekhez való hozzárendelését. Az O-PLAN (Bell és Tate, 1985) HFH-tervkészítő az idő- és erőforrás-korlátozások egységes és általános reprezentációját tartalmazta. A szövegben említett Hitachi-alkalmazás kiegészítéseképpen, az O-PLAN módszert a Price Waterhouse-ban a szoftverbeszerzési tervkészítéshez alkalmazták, míg a Jaguar Autógyárban a hátsó tengely összeszerelési tervének készítésénél. Számos hibrid tervkészítő és ütemező rendszert alkalmaztak: az ISIS (FOX és társai, 1982; Fox, 1990) egy ütemezési feladatban alkalmazták a Westinghouse-nál, a GARI (Descotte és Latombe, 1985) mechanikus alkatrészek gépesítését és készítését tervezte, a FORBIN-t gyár vezérlésre használták, végezetül a NONLIN+ hajózási logisztikai tervkészítéshez alkalmazták.

Az 1980-as években megindult elméleti munka tetőzése után, napjainkban ismét a temporális tervkészítés kerül előtérbe, amikor az új algoritmusok és a megnövekedett feldolgozási teljesítmény lehetővé tette a gyakorlati felhasználást. A két tervkészítő a SAPA (Do és Kambhampati, 2001) és a T4 (Haslum és Geffner, 2001) előrefelé történő állapottér keresést alkalmaztak, amit az időtartammal és erőforrásokkal rendelkező cselekvések

Created by XMLmind XSL-FO Converter.

kezeléséhez egy kifinomult heurisztikával egészítettek ki. Egy másik lehetőség a nagyon költséges cselekvésleíró nyelvek használata, melyeket kézzel írt és alkalmazási terület specifikus heurisztikákkal irányíthatunk, amint az az ASPEN (Fukunaga és társai, 1997), a HSTS (Jonsson és társai, 2000) és az IxTeT (Ghallab és Laruelle, 1994) esetén is történik.

Az ütemezés az űrkutatásban nagyon hosszú múltra tekint vissza. A T-SCHED (Drabble, 1990) rendszert az UOSAT-II műhold küldetésvezérlő parancssorozatainak ütemezésére használták. Az O-PLAN-en alapuló OPTIMUM-AIV (Aarup és társai, 1994) és a PLAN-ERS1 (Fuchs és társai, 1990) az űrrepülőgép-összeszerelésben a megfigyelési tervkészítésben használták fel az Európai Űrkutatási Ügynökségnél (ESA).

A SPIKE rendszert (Johnston és Adorf, 1992) a NASA-nál a Hubble-űrteleszkóp megfigyelés-tervezéséhez használták, míg az űrsikló földi feldolgozásütemező rendszer (Deale és társai, 1994) emberi erőforrás ütemezést végez egészen 16 000 munkás-műszak értékig. A Remote Ágens (Muscettola és társai, 1998) lett az első önálló tervkészítő és ütemező, mely űrsiklót vezérelt, mialatt a Deep Space One űrszonda fedélzetén repült 1999-ben. Az emberi erőforrás ütemezés irodalmát Vaessens és társai (Vaessens és társai, 1996) foglalták össze, az elméleti eredményeket Martin és Shmoys (Martin és Shmoys, 1996) jelentette meg.

A STRIPS programban található a **macrops** – elemi lépések sorozatából álló „makróoperátorok” (**macro operators**) – tanulásra szolgáló eszköztár, az első hierarchikus mechanizmusnak tekinthető (Fikes és társai, 1972). A hierarchiát szintén alkalmazták a LAWALY rendszerben (Siklossy és Dreussi, 1973). Az ABSTRIPS rendszer (Sacerdoti, 1974) az **absztrakciós hierarchia** (**abstraction hierarchy**) ötletét vezette be, ahol a magasabb szintű tervezés megengedte a cselekvések alacsonyabb szintű előfeltételeinek figyelmen kívül hagyását, hogy egy működő terv általános struktúráját származtathassa. Austin Tate PhD-tézise (Tate, 1975b) és Earl Sacerdoti (Sacerdoti, 1977) munkája a HTM tervkészítés modern formájának alapötleteit vezette be. Számos gyakorlati tervkészítő, beleértve az O-PLAN és a SIPE tervkészítőket, HFH-tervkészítő. Yang (Yang, 1990) a HFH-tervkészítést hatékonnyá tevő cselekvések jellemzőit tárgyalja. Erol, Hendler és Nau (Erol és társai, 1994, 1996) egy teljes hierarchikus dekompozíciót alkalmazó tervkészítőt mutat be, valamint számos komplexitásbeli eredményt a tiszta HFH-tervkészítőkhöz. Más szerzők (Ambros-Ingerson és Steel, 1988); Young és társai (Young és társai, 1994); Barrett és Weld (Barett és Weld, 1994); Kambhampati és társai (Kambhampati és társai, 1998) az ebben a fejezetben bemutatott hibrid megközelítést javasolták, melyben a dekompozíciók mindössze a részben rendezett tervkészítéshez használható finomítások egy másik formáját jelentik.

A STRIPS-ben szereplő makróoperátorokkal kezdődően a hierarchikus tervkészítés egyik célja a korábbi, általánosított tervek formájában rendelkezésre álló tervkészítési tapasztalat újrahasznosítása. A **magyarázatalapú tanulás** (**explanation-based learning**) technikáját, amit részleteiben a 19. fejezet mutat be, számos rendszerben, mint a SOAR Laird és társai (Laird és társai, 1986) és PRODIGY (Carbonell és társai, 1989) alkalmazták a korábban elkészített tervek általánosítására. Egy másik megközelítés, hogy a korábban kiszámított terveket eredeti formájukban tároljuk el, majd az új hasonló feladatokban az eredeti feladat analógiájára újrahasznosítjuk őket. Ezt a megközelítést **esetalapú tervkészítés**nek (**case-based planning**) (Carbonell, 1983; Alterman, 1988; Hammond, 1989) nevezzük. Kambhampati (Kambhampati, 1994) érvelése szerint az esetalapú tervkészítést a finomító tervkészítéshez hasonlóan kellene vizsgálni. Emellett lefekteti az esetalapú részben rendezett tervkészítés formális alapjait.

A valós környezetek kiszámíthatatlanságát és részleges megfigyelhetőségét elsőként tervkészítési technikákat használó robotikai projektekben – beleértve a Shakey (Fikes és társai, 1972) és a FREDDY (Michie, 1974) projekteket – ismerték fel. A problémakör több figyelmet kapott McDermott nagy hatású *Tervezés és cselekvés (Planning and Acting)* c. cikkének megjelenését követően (McDermott, 1978a).

A korai, feltételeket és ciklusokat nem tartalmazó tervkészítők nem tükrözték pontosan a feltételes tervkészítés megközelítésmódját, de ennek ellenére néha kényszerítően reagáltak a környezet bizonytalanságára. Sacerdoti NOAH rendszere kényszerítést használt a „kulcsok és dobozok” (keys and boxes) probléma megoldásában, egy olyan tervkészítési kihívásban, ahol a tervkészítő keveset tud a kiinduló állapotról. Mason (Mason, 1993) kijelentette, hogy a robotikai tervkészítés nélkülözni tudja és nélkülöznie is kell az érzékelést, és bemutatott egy érzékelőmentes tervet, ami a kiindulási pozíciótól *függetlenül* képes egy eszközt csuklómozdulatok sorozatával az asztal egy megadott pozíciójába mozgatni. Ezt az ötletet a robotika kontextusában tárgyaljuk (lásd 25.17. ábra).

Goldman és Boddy (Goldman és Boddy, 1996) bevezették az **alkalmazkodó tervkészítés** (**conformant planning**) fogalmát, az érzékelőmentes tervkészítőkre, melyek a bizonytalanságot a környezet ismert állapotokba való kényszerítésével kezelik, megjegyezve, hogy az érzékelőmentes tervek gyakran hatékonyak

Created by XMLmind XSL-FO Converter.

még akkor is, ha az ágens rendelkezik érzékelőkkel. Az első megfelelően hatékony alkalmazkodó tervkészítő Smith és Weld (Smith és Weld, 1998) Conformant Graphplan (CGP) rendszere volt. Ferraris és Giunchiglia (Ferraris és Giunchiglia, 2000), valamint Rintanen (Rintanen, 1999) függetlenül fejlesztettek ki SATPLAN-alapú alkalmazkodó tervkészítőket. Bonet és Geffner (Bonet és Geffner, 2000) a hiedelmi állapotok terén végzett heurisztikus keresésalapú alkalmazkodó tervkészítőt mutat be, olyan ötletekre alapozva, melyeket először 1960-ban fejlesztettek ki a részlegesen megfigyelhető Markov döntési folyamatokhoz (Partially Observable Markov Decision Process – POMDP) (lásd 17. fejezet). Jelenleg a leggyorsabb hiedelmi állapot alkalmazkodó tervkészítők, mint például a HSCP (Bertoli és társai, 2001a) bináris döntési diagramokat (BDD) (Bryant, 1992) használnak a hiedelmi állapotok leírására, valamint öt nagyságrenddel gyorsabbak, mint a CGP.

A WARPLAN-C (Warren, 1976), ami a WARPLAN egy változata, volt az első feltételes cselekvéseket használó tervkészítők egyike. Olawski és Gini (Olawski és Gini, 1990) a feltételes tervkészítés fő problémáit vetette fel.

A fejezetben bemutatott feltételes tervkészítési megközelítés a Jimenez és Torras (Jimenez és Torras, 2000), valamint Hansen és Zilberstein (Hansen és Zilberstein, 2001) által kifejlesztett ciklikus ÉS-VAGY gráfok hatékony keresési algoritmusán alapul. Bertoli és társai (Bertoli és társai, 2001b) egy BDD-alapú megközelítést mutatnak be, ami ciklikus feltételes terveket készít. A C-BURIDAN rendszer (Draper és társai, 1994) egy folytonos tervkészítést végez valószínűségi kimenetelű cselekvésekre, melynek problémája a POMDP vezérlése során is felmerült (lásd 17. fejezet). Szoros kapcsolat van a feltételes tervkészítés és az automatizált programkészítés között. A 9. fejezet számos ilyen hivatkozást tartalmaz. A két területet egymástól függetlenül tárták fel, aminek oka az az óriási költségkülönbség, ami egy gépi utasítás végrehajtása, valamint robot- vagy manipulátorcselekvések végrehajtása között fennáll. Linden (Linden, 1991) a két terület közötti tudásmegosztást szorgalmazza.

Történelmi távlatból már látható, hogy hogyan vezetett a két fő klasszikus tervkészítő algoritmus kibővített verziókhoz a bizonytalan környezetek kezelése céljából. A keresésalapú technikák a hiedelmi állapottér keresésre vezettek (Bonet és Geffner, 2000). A SATPLAN algoritmus a sztochasztikus SATPLAN (Majercik és Littman, 1999) algoritmusra, valamint a kvantorokat alkalmazó logikai tervezésre vezetett (Rintanen, 1999). A részben rendezett tervkészítésből az UWL (Etzioni és társai, 1992), a CNLP (Peot és Smith, 1992) és a CASSANDRA (Pryor és Collins, 1996) származik. A GRAPHPLAN, az érzékeléses GRAPHPLAN (sensory GRAPHPLAN-SGP) (Weld és társai, 1998) algoritmusra vezetett, de a teljesen valószínűségi GRAPHPLAN algoritmus még kifejlesztésre vár.

A végrehajtás monitorozás első fő megjelenése a PLANEX (Fikes és társai, 1972), ami egy STRIPS tervkészítővel működött a Shakey robot irányításán. A PLANEX háromszög táblákat használt – ami egy szükségszerűen hatékony tárolási mechanizmus a terv előfeltételeinek tárolására a terv minden pontján –, hogy teljes újratervezés nélkül helyreállhasson egy részleges végrehajtási hiba után. A végrehajtás Shakey-modelljét a 25. fejezetben tárgyaljuk tovább. A NASL tervkészítő (McDermott, 1978a) a tervkészítési problémát egyszerűen, mint egy komplex cselekvés végrehajtásának specifikációját tekintette, így a tervezés és végrehajtás teljesen egységessé vált. Az összetett cselekvések levezetésére tételbizonyítást használ.

A SIPE (System for Interactive Planning and Execution Monitoring) (Wilkins, 1988, 1990) volt az első tervkészítő, ami az újratervezés problémáját szisztematikusan kezelte. Különböző problémakörök demonstrációs feladataiban is felhasználták, például az űrsiklóhordozó fedélzeti terv készítésére vagy emberi erőforrás ütemezésére egy ausztrál sörgyárban. Egy másik vizsgálat a SIPE rendszert többemeletes épületek tervezésében alkalmazta, ami az egyik legbonyolultabb problémakör, amit tervkészítő valaha kezelt.

Az IPEM (Intergrated Planning, Execution, and Monitoring) (Ambros-Ingerson és Steel, 1988) volt az első rendszer, ami integrálta a részben rendezett tervkészítést és a végrehajtást, hogy folytonos tervkészítő ágensre jusson. Az általunk tárgyalt folytonos, részben rendezett tervkészítő ágens (CONTINUOUS-POP-AGENT) az IPEM a PUCCINI (Golden, 1998) és a CYPRESS (Wilkins és társai, 1995) tervkészítők ötleteit kombinálja.

Az 1980-as évek közepén néhányan úgy gondolták, hogy a részben rendezett tervkészítés és a kapcsolódó technikák soha nem futhatnak eléggé gyorsan, hogy egy valódivilág-beli ágens számára használható viselkedést biztosítsanak (Agre és Chapman, 1987). Ezek helyett a **reaktív tervkészítő** (**reactive planning**) rendszereket vezették be. Ezek alapesetben reflexszerű ágensek, esetenként belső állapottal, melyek számos különböző feltétel-cselekvés szabályábrázolás segítségével implementálhatók. Brooks (Brooks, 1986) bennfoglalási architektúrája (lásd. 7. és 25. fejezet) rétegzett véges állapotú állapotgépeket használt, lábbal és kerékkel ellátott robotok mozgásának és akadályok elkerülésének a szabályozására. Pengi (Agre és Chapman, 1987) egy (teljesen megfigyelhető) videojátékot tudott játszani, az aktuális célok „látott” reprezentációja, az ágens belső állapota, valamint kétértékű áramkörök kombinálásával.

Created by XMLmind XSL-FO Converter.

Az „univerzális tervek” (Schoppers, 1987) a reaktív tervkészítés keresőtábla módszereként került kifejlesztésre, de mint kiderült, ez a Markov döntési folyamatokban használt **eljárásmód** (**policy**) ötletének újrafelfedezése volt. Egy univerzális terv (vagy egy eljárásmód) bármely állapotban az adott állapotban végrehajtandó cselekvésre mutató leképezést tartalmazza. Ginsberg (Ginsberg, 1989) lelkes támadást indított az univerzális tervek ellen, ami kezelhetetlenségi eredményeket is tartalmazott a reaktív tervkészítési probléma néhány alakjára. Erre Schoppers 1989-ben egy hasonlóan lelkes választ adott (Schoppers, 1989).

Mint a legtöbb esetben a hibrid megközelítés feloldja az ellentmonást. Jól tervezett hierarchiák használatával a HFH tervkészítők, mint a PRS (Georgeff és Lansky, 1987) és a RAP (Firby, 1996), valamint a folytonos tervkészítő ágensek használható válaszidőket és összetett hosszú távú terveket érhetnek el számos problémakörben.

A multiágens tervkészítés napjainkban vált népszerűvé, bár hosszú történelme van. Konolige (Konolige, 1982) a multiágens tervkészítés elsőrendű logikai leírását adta meg, míg Pednault (Pednault, 1986) egy `STRIPS`-alakú leírást készített. A kölcsönös szándék ötlete, ami elengedhetetlen, ha az ágensek együttes tervet hajtanak végre, a kommunikációs viselkedés kutatásából ered (Cohen és Levesque, 1990; Cohen és társai, 1990). Az általunk bemutatott többtestű részben rendezett tervkészítés Boutilier és Brafman (Boutilier és Brafman, 2001) munkáján alapul.

Az itt szereplő, a multiágens tervkészítésben az együttműködést célzó munka bemutatása éppcsak a jéghegy csúcsa. Durfee és Lesser (Durfee és Lesser, 1989) azt tárgyalja, hogy a feladatok hogyan oszthatók meg az ágensek között egyezkedés útján. Kraus és társai (Kraus és társai, 1991) egy Diplomacyt – egy egyezkedést, együttműködést, felbomlást és ügyeskedést igénylő táblajátékot – játszó rendszert mutatnak be. Stone (Stone, 2000) bemutatja, hogy az ágensek hogyan tudnak csapattagként együttműködni a robotfoci versengő, dinamikus, részben megfigyelhető környezetében. A (Weiss, 1999) a multiágens rendszerek egy könyvhosszúságú áttekintése. Az 7.3. szakasz - Koordináló mechanizmusok rész moid modellje Reynoldsnak köszönhető (Reynolds, 1987), aki ennek a *Batman visszatér* (*Batman returns*) c. film denevérseregeire és pingvincsapatára való alkalmazásával Oscar-díjat is nyert.

8.2. Feladatok

12.1.

Vizsgáljuk meg figyelmesen a 12.1. alfejezetben az idő és az erőforrások reprezentációját.

a. Miért jobb, ha egy cselekvés következménye az *Időtartam*(*d*), mint ha a cselekvésnek egy `IDŐTARTAM`:*d* alakú külön mezője lenne? (*Segítség:* vegyük figyelembe a feltételes következményeket és a diszjunktív következményeket.)

b. Az `ERŐFORRÁS`: *m* miért egy külön mező a cselekvésben, nem pedig egy következmény?

12.2.

Egy **fogyóeszköz** (**consumable resource**) egy olyan erőforrás, ami (részlegesen) felhasználásra kerül a cselekvésben. Például a motor beszerelése az autóba csavarokat igényel. Miután a csavarokat felhasználtuk, már nem használhatók további rögzítésekhez.

a. Magyarázza meg, hogyan módosítsuk a 12.3. ábra reprezentációját úgy, hogy 100 kiinduló csavarunk legyen az E_1 motor 40 csavart, az E_2 pedig 50 csavart igényel. A + és – szimbólumok felhasználhatók az erőforrásokat leíró következményliterálokban.

b. Mutassa meg, hogy a részben rendezett tervkészítésben alkalmazott okozati kapcsolatok és cselekvések közötti **ütközés** (**conflict**) definícióját hogyan kell módosítani, hogy a fogyóeszközöket kezelje.

c. Néhány cselekvés, például a gyár ellátása csavarokkal vagy egy autó feltankolása, *növelheti* az erőforrások készletét. Egy erőforrás monoton nem növekvő, ha egyetlen cselekvés sem növeli. Magyarázza meg, hogyan használható ez a tulajdonság a keresési tér megmetszéséhez.

12.3.

Adjon dekompozíciót a 12.7. ábra *KivitelezőtVeszFel* és *EngedélytSzerez* lépéseire, és mutassa meg, hogy a dekomponált résztervek hogyan állnak össze egy teljes tervvé.

Created by XMLmind XSL-FO Converter.

12.4.

Adjon példát a házépítési problémakörben két olyan absztrakt résztervre, ami nem egyesíthető egy teljes tervvé lépések megosztása nélkül. (*Segítség:* azok a pontok, ahol egy ház fizikai részei csatlakoznak, egyúttal a résztervek várható csatlakozási kölcsönhatási pontjai is.)

12.5.

Sokan mondják, hogy a HFH-tervkészítés előnye, hogy „tegyünk egy körutat Los Angelesből New Yorkba, és vissza” típusú problémákat is meg tud oldani, amelyeket nehéz nem HFH-jelölésekkel kifejezni, hiszen a kiinduló és a célállapotok azonosak lennének (*Ott*(*LA*)). Tud mondani megfelelő reprezentációt, illetve meg tudja-e oldani ezt a problémát HFH-k nélkül?

12.6.

Mutassa meg, hogy az eredeti `STRIPS` cselekvésleírás hogyan írható át HFH-dekompozícióvá. Használja az *Eléri*(*p*) cselekvést a *p* feltétel eléréséhez szükséges cselekvés jelölésére.

12.7.

A hagyományos programozási nyelvekben számos művelet modellezhető olyan cselekvéssel, amely megváltoztatja a világ állapotát. Például egy értékadási operátor egy memóriaterület tartalmát másolja, míg egy nyomtatás operátor megváltoztatja a kimenet állapotát. Egy, ezekből a műveletekből álló programot tervnek tekinthetünk, melynek célját a program specifikációja definiálja. A felkészítő algoritmusok ezért felhasználhatók egy adott specifikációjú program elkészítéséhez.

a. Írjon egy operátorsémát az értékadási operátorhoz (egy változó értékének átadása egy másik változónak). Vegye figyelembe, hogy az eredeti érték felülíródik!

b. Mutassa meg, hogy az objektumok létrehozása hogyan használható a felkészítő által két változó értékének egy átmeneti változó felhasználásával történő felcserélésére.

12.8.

Tekintsük a következő indoklást: egy bizonytalan kiinduló állapotot megengedő keretrendszerbe a **diszjunktív következmény**ek (**disconjuctive effect**s) csak jelölési könnyebbséget jelentenek, és nem a kifejezőképesség növelésére szolgálnak. Egy $P \vee Q$ diszjunktív következményt tartalmazó *a* cselekvéssémát mindig helyettesíthetünk a **when** *R*: $P \wedge$ **when** $\neg R$: *Q* feltételes következményekkel, ami cserében két hagyományos cselekvésre redukálható. Az *R* kifejezés egy a kiinduló állapotban ismeretlen véletlen állítást takar, melyhez nem tartozik érzékelő cselekvés. Helyes ez az indoklás? Vegyünk külön két esetet, egyet, melyben csak egy *a* cselekvésséma szerepel a tervben, és egy másikat, melyben több mint egy példány van.

12.9.

Miért nem tudja a feltételes tervkészítés kezelni a nem korlátos nemdeterminisztikusságot?

12.10.

A kockavilágban két `STRIPS` cselekvést kellett bevezetnünk, hogy az *Üres* predikátumot helyesen megtartsuk, a *Mozgat* és az *AsztalraTesz* cselekvéseket. Mutassuk meg, hogy a feltételes következmények hogyan használhatók mindkét eset egyetlen cselekvéssel történő leírására.

12.11.

A feltételes következményeket a porszívóvilág *Szív* cselekvésére mutattuk be, ahol az, hogy melyik kocka lesz tiszta, attól függ, hogy mely kockán áll a robot. Tud mondani olyan ítéletlogikai változókat, melyek a porszívóvilág állapotait definiálják úgy, hogy a *Szív* cselekvésnek *feltétel nélküli* leírása legyen? Adja meg a *Szív*, a *Balra* és a *Jobbra* cselekvések leírását az állításaik felhasználásával, és mutassa meg, hogy ezek kielégítők a világ összes lehetséges állapotának leírásához.

12.12.

Created by XMLmind XSL-FO Converter.

Adja meg a *Szív* cselekvés teljes leírását a dupla-Murphy porszívó problémára, ami néha piszkot hagy maga után, amikor egy tiszta célterületre mozog, és néha piszkot rak le, ha a *Szív* cselekvést egy tiszta négyzeten hajtjuk végre.

12.13.

Keressen egy megfelelően piszkos szőnyeget, amely akadálymentes és porszívózza ki. Rajzolja le a porszívó által választott útvonalat, amilyen pontosan csak tudja. Indokolja a választ hivatkozva a fejezetben tárgyalt tervkészítési módszerekre.

12.14.

A következő idézetek samponos flakonok hátuljáról származnak. Azonosítsa mindegyiket, mint egy feltétel nélküli, feltételes vagy végrehajtás monitorozó tervet. (a) „Mosás. Öblítés. Ismételje." (b) „Tegye a sampont a fejére, és hagyja ott néhány percig. Öblítsen és ismételje, ha szükséges." (c) „Amennyiben az irritáció nem szűnik, forduljon orvoshoz."

12.15.

A 12.10. ábrán szereplő `ÉS-VAGY-GRÁF-KERESÉS` algoritmus csak a gyökér és az aktuális állapot közötti útvonalat ellenőrzi le ismétlődő állapotokra. Mindehhez tegyük fel, hogy az algoritmus eltárolt *minden* meglátogatott állapotot, és összeveti ezzel a listával (lásd például 3.19. ábra `GRÁF-KERESÉS`). Határozza meg, hogy milyen információt kellene tárolni, és azt, hogy az algoritmus hogyan használja ezt fel, amikor ismétlődő állapotot talál. (*Segítség:* szüksége lesz arra, hogy különbséget tegyen azon állapotok, melyekhez egy sikeres részterv készült korábban, és azon állapotok között, melyekre nem talált résztervet.) Magyarázza meg, hogyan használhatunk **címké**ket (**label**s), hogy elkerüljük a résztervek duplikálását.

12.16.

Mutassa meg precízen, hogy hogyan módosítjuk az `ÉS-VAGY-GRÁF-KERESÉS` algoritmust, hogy ciklikus tervet generáljon, amennyiben ciklusmentes terv nem létezik. Ehhez három problémával kell megküzdenie: a terv lépéseinek címkézése, hogy egy ciklikus terv vissza tudjon mutatni a terv egy korábbi pontjára, a `VAGY-KERESÉS` módosításával, hogy egy ciklikus terv megtalálása után ciklusmentes tervet keressen, és a terv leírásának olyan bővítésével, mely jelzi, hogy a terv ciklikus. Mutassa meg, hogyan működik az algoritmus (a) a tripla-Murphy porszívóvilágban, (b) a váltakozó dupla-Murphy porszívóvilágban. Használhat számítógépes implementációt az eredmények ellenőrzéséhez. A (b) eset terve felírható egy hagyományos ciklusszintaxis használatával?

12.17.

Specifikálja teljesen a hiedelem állapot módosító folyamatot részben megfigyelhető környezetekre. Azaz a módszert, ami az új hiedelmi állapot leírást számítja (mint ismeret állítások listáját) az aktuális hiedelmi állapot leírásból és a feltételes következményeket tartalmazó cselekvésleírásokból.

12.18.

Írjon a (12.2.) egyenlettel analóg cselekvésleírásokat a *Jobbra* és a *Szív* cselekvésekre. Adja meg a leírását a *PozícióEllenőrzés* cselekvésnek, a (12.3.) egyenletnek megfelelően. Ismételje ezt meg a 12.11. feladat alternatív állítás halmazának felhasználásával.

12.19.

Tekintse meg az 6. szakasz - Folytonos tervkészítés rész listáját, ami felsorolja, hogy mit nem tud az újratervező ágens végrehajtani. Vázoljon fel egy algoritmust néhány probléma kezelésére.

12.20.

Vegyük a következő feladatot: egy páciens olyan tünetekkel érkezik az orvoshoz, amelyet kiszáradás vagy *D* fertőzés okozhat (de nem mindkettő). Két lehetséges cselekvés van: *Iszik*, ami feltétel nélkül kúrálja a kiszáradást, és *Gyógyszert szed*, amely a *D* fertőzést kúrálja ki, de káros mellékhatása van, ha a páciens kiszáradt. Írja fel a probléma leírást PDDL-ben, és az összes releváns világ számbavétele mellett ábrázolja az érzékelőmentes tervet, ami megoldja a problémát.

Created by XMLmind XSL-FO Converter.

12.21.

Az előző feladat gyógyszerszedési problémájához adja hozzá a *Vizsgál* cselekvést, melynek feltételes következménye a *TenyészetNövekedés*, amennyiben a *Fertőzés* igaz, és minden esetben van egy érzékelési következménye: *Ismert*(*TenyészetNövekedés*). Ábrázolja a feltételes tervet, ami megoldja a problémát és minimalizálja a *GyógyszertSzed* cselekvés felhasználását.

Created by XMLmind XSL-FO Converter.

13. fejezet - Bizonytalanság

Ebben a fejezetben meglátjuk, mit kell egy ágensnek tennie, ha nem minden kristálytiszta.

1. Cselekvés bizonytalan tudás esetén

Fontos

A III. és IV. részben leírt logikai ágensek ismeretelméleti kijelentéseket tesznek az állítások igaz, hamis vagy ismeretlen voltáról. Ha egy ágens elegendő mennyiségű tényt ismer a környezetére vonatkozóan, akkor a logikai megközelítés révén olyan terveket tud származtatni, amelyek biztosítják a sikert. Ami nagyon jó. Sajnos azonban *az ágensek szinte soha nem férnek hozzá a környezetüket érintő teljes igazsághoz.* Vagyis az ágenseknek **bizonytalanság** (**uncertainty**) közepette kell működniük. A 7. fejezet wumpus világának ágense például csak olyan szenzorokkal rendelkezik, amelyek lokális információt szolgáltatnak; a világ nagy része nem figyelhető meg közvetlenül. A wumpus ágens gyakran képtelen felfedezni, hogy két négyzet közül melyik tartalmaz csapdát. Ha a két négyzet az aranyhoz vezető úton van, akkor az ágensnek kockáztatnia kell, és rá kell lépnie az egyikre.

A világ a valóságban nagyságrendekkel bonyolultabb a wumpus világnál. Egy logikai ágens nem biztos, hogy képes kimerítő és helyes leírást adni arról, hogy a cselekedetei mennyire lesznek eredményesek. Tegyük fel például, hogy az ágens ki szeretne vinni valakit a repülőtérre egy bizonyos járathoz. Ehhez egy A_{90} nevű tervet tekint, amely szerint a járat indulása előtt 90 perccel kell elindulnia otthonról és normális sebességgel kell vezetnie. Mindemellett, még ha a repülőtér csak 15 kilométerre van is, az ágens nem juthat a következő határozott kijelentésre: az A_{90} terv alapján időben kiérünk a repülőtérre, hanem csak egy gyengébb következtetésre: az A_{90} terv alapján időben kiérünk a repülőtérre, ha az autóm nem romlik el vagy nem fogy ki belőle az üzemanyag, továbbá, ha nem történik velünk baleset, valamint a hídon sem lesz baleset, és a gép nem száll fel korábban, továbbá ha nem lesz földrengés… E feltételek egyike sem vezethető le, így a terv sikere sem következtethető ki biztosan. Mindez a 10. fejezetben említett **minősítési problémá**ra (**qualification problem**) mutat példát.

Ha egy logikai ágens nem tud arra a következtetésre jutni, hogy valamely cselekvéssorozat eléri a célját, akkor képtelen lesz cselekedni. A feltételes tervezés bizonyos mértékben úrrá lehet a bizonytalanságon, de csak akkor, ha az ágens érzékelési tevékenységei képesek megszerezni a szükséges információt, továbbá ha nincs túl sok különböző eshetőség. Egy másik lehetőség az lehetne, ha az ágenst egy egyszerű, de hibás elmélettel ruháznánk fel, amely *biztosan* lehetővé tenné valamely terv származtatását; egy ilyen terv az esetek *többségében* valóban működik is, ugyanakkor problémák lépnek fel, ha az események az ágens elméletének ellentmondanak. Sőt az ágens elméletének pontossága és használhatósága közötti egyensúly kezelése maga is megköveteli a bizonytalanság végiggondolását. Összegezve, egyetlen tisztán logikai ágens sem lesz képes arra a következtetésre jutni, hogy az A_{90} tervet kell követnie.

Fontos

Mindazonáltal tegyük fel, hogy az A_{90} ténylegesen a helyes és követendő terv. De mit is értünk ezen a kijelentésen? Ahogy már a 2. fejezetben kifejtettük, ez a kijelentés azt takarja, hogy a végrehajtható összes lehetséges terv közül az A_{90} az, amelyik a környezetre vonatkozó adott információ mellett várhatóan az ágens teljesítményének legnagyobb mértékét biztosítja. A teljesítmény foka magában foglalja a járathoz időben való kiérést a repülőtérre, továbbá a repülőtéren való hosszú, értelmetlen várakozás és a gyorshajtás miatti megbírságolás elkerülését. Az ágens rendelkezésére álló információ ezen kimenetelek közül egyiket sem szavatolja az A_{90} esetén, de bizonyos mértékű hihetőségét nyújtja annak, hogy azok be fognak következni. Más tervek, mint például az A_{120}, megnövelné az ágens hitét abban, hogy időben kiérnek a repülőtérre, de egyben megnövelné a hosszú várakozás valószínűségét is. *A helyes cselekedet – az **ésszerű döntés** (**rational decision**) – ezért függ mind a különböző célok viszonylagos fontosságától, mind pedig a megvalósulásuk valószínűségétől és mértékétől.* Az alfejezet hátralevő részében ezeket a gondolatokat szeretnénk hangsúlyozni, előkészítvén az ebben és a következő fejezetekben bemutatott bizonytalan következtetés és az ésszerű döntés általános elméleteinek a kifejlesztését.

1.1. Bizonytalan tudás kezelése

Created by XMLmind XSL-FO Converter.

Ebben az alfejezetben a bizonytalan tudás természetét vizsgáljuk meg közelebbről. Egy egyszerű diagnosztikai példán keresztül mutatjuk be az érintett fogalmakat. A diagnosztika – legyen az orvosi, gépjármű-javítási vagy bármi egyéb – olyan feladat, amely szinte mindig tartalmaz bizonytalanságot. Hogy lássuk a logikai megközelítés sikertelenségét, próbáljuk meg egy fogorvosi diagnosztikai rendszer szabályait meghatározni elsőrendű logika segítségével. Tekintsük a következő szabályokat:

$\forall p\ Tünet(p, Fogfájás) \Rightarrow Betegség(p, Lyuk)$

A gond az, hogy ez a szabály rossz. Nincs minden fogfájós betegnek lyukas foga; néhányuknak lehet ínysorvadása vagy tályoga, vagy valamilyen más problémája.

$\forall p\ Tünet(p, Fogfájás) \Rightarrow Betegség(p, Lyuk) \vee Betegség(p, Ínysorvadás) \vee Betegség(p, Tályog)\ldots$

Sajnos, ahhoz, hogy igazzá tegyük a szabályt, közel végtelen sok okot kellene hozzátennünk. Megpróbálhatjuk a szabályt ok-okozati szabállyá formálni:

$\forall p\ Betegség(p, Lyuk) \Rightarrow Tünet(p, Fogfájás)$

De még ez a szabály sem lesz helyes, hiszen nem minden lyukas fog okoz fájdalmat. Az egyetlen módja, hogy kijavítsuk a szabályt, az az, hogy logikailag teljessé tesszük: a bal oldalt ki kell bővíteni minden olyan lehetséges okkal, amely lyukas fogak esetén fogfájást okoz. De még ebben az esetben is figyelembe kell venni a diagnózis megállapításánál annak lehetőségét, hogy a betegnek egymástól teljesen függetlenül is lehet fogfájása és lyukas foga.

Próbálkozásaink az elsőrendű logika alkalmazására olyan területeken, mint az orvosi diagnosztika, három fő okból is kudarcot vallanak:

- **Lustaság (laziness)**: túl nagy munkát jelent az ok és okozatok teljes eseményhalmazának felsorolása, amely elengedhetetlenül szükséges annak biztosítására, hogy egy szabály mindig érvényes legyen. Ezenkívül túl körülményes az eredményül kapott hatalmas méretű szabályok használata.

- Az **elméleti tudatlanság (theoretical ignorance)**: az orvosi tudományterület elmélete nem teljes.

- A **gyakorlati tudatlanság (practical ignorance)**: még ha ismerjük is a teljes szabályrendszert, egy bizonyos beteggel kapcsolatban akkor is lehetünk bizonytalanok, mert előfordulhat, hogy még nem fejeződött be, vagy egyáltalán nem végezhető el az összes szükséges vizsgálat.

Arról van szó, hogy a fogfájás és a lyuk közötti kapcsolat egyik irányban sem feltétlen logikai következmény. Ez jellemző az orvosi területekre, valamint más szakértői területekre (jog, üzlet, tervezés, autójavítás, kertészkedés, időpontok összeegyeztetése stb.). Az ágens tudása legjobb esetben is csak egy **bizonyos mértékű meggyőződés**t vagy **hiedelmet (degree of belief)** nyújthat az adott kijelentésekkel kapcsolatban. A meggyőződési értékek kezelésére az elsődleges eszközünk a **valószínűség-számítás (probability theory)** lesz, amely egy 0 és 1 közötti számszerű meggyőződési mértéket rendel az egyes mondatokhoz. (A bizonytalan következtetésre alkalmas néhány más lehetséges módszerről a 14.7. alfejezetben olvashatunk.)

Fontos

*A valószínűség lehetőséget nyújt a lustaságunkból vagy tudáshiányunkból fakadó bizonytalanság kifejezésére **(összegzésére).*** Nem lehetünk biztosak abban, hogy mi kínoz egy bizonyos beteget, ugyanakkor hihetjük, hogy mondjuk 80% az esély arra – azaz 0,8 a valószínűsége annak –, hogy a betegnek van lyukas foga, ha fogfájásra panaszkodik. Azaz, várakozásaink szerint az összes, az ágens tudása alapján a jelenlegi szituációtól megkülönböztethetetlen eset 80%-ában a páciensnek van lyukas foga. Ezt a valószínűséget statisztikai adatokból vonhatjuk le – az eddig vizsgált fogfájós betegek 80%-ának lyukas volt a foga –, vagy valamilyen általános szabályból, illetve tények kombinációjából származtathatjuk. A 80% azokat az eseteket foglalja magában, amikor minden olyan tényező jelen van, amely hatására a lyuk fogfájással jár, valamint amikor a betegeknek teljesen függetlenül van fogfájása és lyukas foga is. A hiányzó 20%-ban pedig a fogfájás összes többi lehetséges oka található, amelyek megerősítéséhez vagy tagadásához túl lusták vagy tudatlanok vagyunk.

Egy adott kijelentéshez rendelt 0 valószínűség annak az egyértelmű meggyőződésnek felel meg, hogy a mondat állítása hamis, míg az 1 valószínűség egyenértékű azzal a határozott meggyőződéssel, hogy a mondat állítása

Created by XMLmind XSL-FO Converter.

igaz. A 0 és 1 közötti valószínűségek a mondat igazságtartalmában való hit közbenső mértékeinek felelnek meg. Az állítás *valójában* persze vagy igaz, vagy hamis. Fontos tehát megjegyeznünk, hogy a meggyőződés mértéke és az igazságtartalom mértéke különböző fogalmak. A 0,8 valószínűség nem jelent „80%-ban igaz”-at, hanem egy 80%-os mértékű meggyőződést – vagyis egy igen erős elvárást (reményt) az állítás igazságával szemben. Következésképpen a valószínűség-számítás a logikával azonos lételméleti állásfoglalást hordoz – nevezetesen, hogy a világban a tények vagy érvényesek, vagy nem. Az igazság mértéke, mint a meggyőződés mértékének az ellentéte, a **fuzzy logika** (**fuzzy logic**) tárgya, amelyet a 14.7. alfejezetben tárgyalunk.

A logikában egy olyan állítás, mint „a páciensnek van lyukas foga” az interpretációjától és a világtól függően vagy igaz, vagy hamis; csak akkor igaz, ha az a tény, amelyre hivatkozik, megfelel a tényállásnak. A valószínűség-számításban egy olyan mondat, mint „0,8 annak a valószínűsége, hogy a betegnek lyukas a foga” az ágens meggyőződését fejezi ki, és nem közvetlenül a valóságra vonatkozik. Ez a meggyőződés az ágens addigi észleléseitől függ. Az észlelések alkotják azt a **tény**t vagy **tényállás**t (**evidence**), amelyen a valószínűségi kijelentések alapulnak. Tegyük fel például, hogy az ágens kihúz egy lapot egy megkevert kártyapakliból. Mielőtt ránéz a lapra, 1/52 valószínűséget kell rendelnie a mellé az állítás mellé, hogy a kihúzott lap a pikk ász. Megnézés után ugyanennek a valószínűsége vagy 0, vagy 1. Vagyis, egy kijelentéshez rendelt valószínűség inkább annak felel meg, hogy egy logikai állítás (vagy annak tagadása) következik-e a tudásbázisból, mint annak, hogy igaz-e vagy sem. Ahogy a tudásbázishoz hozzáadott állítások megváltoztatják az abból levonható következtetéseket, ugyanúgy a valószínűség is megváltozik, amikor több tény birtokába jutunk.[1]

Ebből következően minden valószínűségi kijelentésnek hivatkoznia kell azokra a tényekre, amelyek alapján az adott valószínűség az állításhoz lett rendelve. Amint egy ágens új észlelések birtokába jut, ezek figyelembevételével módosítja a valószínűségek becslését. Mielőtt tények birtokába jutunk, **előzetes**, illetve **a priori** (**prior**) vagy **feltétel nélküli** (**unconditional**) valószínűségről beszélünk, a tények birtokában pedig **utólagos**, illetve **a posteriori** (**posterior**) vagy **feltételes** (**conditional**) valószínűségről. Az ágens a legtöbb esetben rendelkezni fog bizonyos tényekkel az érzékelései hatására, és érdekelt lesz az általa felügyelt kimenetelek a posteriori valószínűségeinek kiszámításában.

1.2. Bizonytalanság és racionális döntések

A bizonytalanság megjelenése gyökeresen megváltoztatja azt a módot, ahogy az ágens a döntéseit meghozza. Egy logikai ágensnek általában van valamilyen célja, és bármely tervet végrehajt, amely biztosítja a cél elérését. Azon az alapon választ ki vagy utasít vissza egy cselekvést, hogy az eléri a célt vagy sem, függetlenül attól, hogy más cselekvések mire vezetnek. Más lesz a helyzet azonban, amint a bizonytalanság belép a képbe. Gondoljuk végig újra az A_{90} tervet a repülőtérre jutás szempontjából. Tegyük fel, hogy ez 95%-os valószínűséggel biztosítja a sikert. Ésszerű döntést jelent ez? Nem feltétlenül: lehetnek más tervek, például az A_{120}, amelynek nagyobb esélye van a sikerre. Ha életbevágó az, hogy ne késsük le a járatot, akkor célszerűbb megkockáztatni egy hosszabb várakozást a repülőtéren. Mi a helyzet például az A_{1440} tervvel, amely a felszállás előtt 24 órával való indulást javasolja. A legtöbb esetben ez nem jó választás, mert bár majdnem biztosra vehetjük az időben való érkezést, sajnos a tűrhetetlenül hosszú várakozást is.

Ilyen típusú döntések meghozatala előtt az ágensnek fel kell állítania egy **preferencia-sorrend**et (**preferences**) a különböző tervek lehetséges **kimenetel**ei (**outcome**) között. Egy adott kimenetel teljesen határozott állapotot jelent, magában foglalva, hogy az ágens időben érkezik a repülőtérre vagy sem, és hogy milyen hosszú lesz a repülőtéri várakozás. A preferenciák figyelembevételével történő leírásra és következtetésre a **hasznosságelmélet**et (**utility theory**) fogjuk használni. (A **hasznosság** – **utility** – fogalmán a továbbiakban az értendő, hogy „milyen a haszon minősége”, és nem valamifajta közhasznúság, mint az elektromos művek vagy vízművek esetében.) A hasznosságelmélet szerint minden állapotnak van egy hasznavehetőségi mértéke vagy haszna az ágens számára, és az a számára több hasznot hozó állapotokat fogja előnyben részesíteni.

Egy állapot hasznossága viszonylagos az ágens számára, akinek preferencia-sorrendjét a hasznosságfüggvény hivatott képviselni. A 6. fejezet játszmáiban szereplő jutalmak például hasznosságfüggvények. Sakkozás közben annak az állapotnak a hasznossága, amelyben Fehér játszmát nyer, nyilvánvalóan igen magas a fehér bábukkal játszó ágens számára, ugyanakkor igen alacsony a feketével játszó számára. Vagy bizonyos játékosok (beleértve a szerzőket is) igen elégedettek lehetnek egy világbajnok elleni döntetlennel, ugyanakkor más játékosok (beleértve a korábbi világbajnokot) nem. Az ízlést vagy a személyes választást nem lehet megindokolni: azt gondolhatjuk, hogy egy ágens, aki a műfagyit előnyben részesíti a csokireszelékkel szemben különös, sőt félre

[1] Ez teljesen eltér attól, hogy a világ megváltozása következtében egy állítás igazzá vagy hamissá válik. Ha valószínűségeket akarunk használni egy változó világ kezelésére, akkor ugyanolyan mechanizmusokra – állapot, távolság és események – lesz szükségünk, mint amelyet a 10. fejezetben használtunk a logikai reprezentációban. Ezeket a mechanizmusokat a 15. fejezetben tárgyaljuk.

Created by XMLmind XSL-FO Converter.

van vezetve, de azt nem mondhatjuk, hogy irracionális. A hasznosságelmélet még az önzetlenséget is megengedi, egyszerűen azáltal, hogy az ágens a saját hasznosságát növelő tényezők közé beleveszi mások jólétét.

A preferenciák sorrendje, amit a hasznossággal fejezhetünk ki, az ésszerű döntések általános elméletében valószínűségekkel van kombinálva. Ezt hívjuk **döntéselmélet**nek (**decision theory**):

```
döntéselmélet = valószínűség-elmélet + hasznosságelmélet
```

Fontos

A döntéselmélet alapgondolata szerint *egy ágens akkor és csak akkor racionális, ha olyan cselekvést választ, amely az adott cselekvés összes lehetséges kimenetelére átlagolt legmagasabb várható hasznot hozza.* Ezt hívják a **Maximális Várható Haszon** (**MVH**, **Maximum Expected Utility**, **MEU**) elvének. Az elv működését a 6. fejezetben láthattuk, amikor az ostáblajáték optimális lépését vizsgáltuk röviden. A későbbiekben látni fogjuk, hogy ez az elv teljesen általános érvényű.

1.3. Egy döntéselméleti ágens tervezése

A 13.1. ábra egy, a cselekvései kiválasztásához döntéselméleti módszereket alkalmazó ágens felépítését mutatja. Egy absztrakt szinten az ágens megegyezik a 7. fejezetben leírt logikai ágens felépítésével. Az elsődleges különbség közöttük az, hogy a döntéselméleti ágens pillanatnyi állapotra vonatkozó tudása bizonytalan; az ágens **meggyőződési** vagy **hiedelemállapot**a (**belief state**) a világ összes lehetséges aktuális állapotainak a valószínűségeit megjeleníti. Az idő előrehaladtával az ágens egyre több tényt gyűjt össze, és meggyőződési állapota is változik. Adott meggyőződési állapotra alapozva az ágens valószínűségi becsléseket tud adni az egyes cselekedetek kimeneteleire vonatkozóan, következésképpen ki tudja választani a legnagyobb várható hasznossággal bíró lépést. Ez és a következő fejezet a valószínűségi információ általános megjelenítésére és az azon alapuló számításokra összpontosít. A 15. fejezet a meggyőződési állapot reprezentálásának és aktualizálásának, valamint a környezet becslésének speciális módszereivel foglalkozik. A 16. fejezet a hasznosságelméletet taglalja mélységeiben, míg a 17. fejezetben a komplex döntések módszereit építjük fel.

13.1. ábra - Döntéselméleti ágens, amely racionális cselekvéseket választ ki. A lépéseket a következő öt fejezetben részletezzük.

function DE-ÁGENS(*észlelés*) **returns** egy *cselekvés*
  static: *meggyőződési állapot*, a világ állapotára vonatkozó valószínűségi meggyőződések halmaza
    cselekvés, az ágens cselekvése

  a *meggyőződési állapot* aktualizálása *cselekvés* és *érzékelés* alapján
  cselekvések kimeneteleihez tartozó valószínűségek kiszámítása,
    adott műveletek leírása és pillanatnyi *meggyőződési állapot*
  a legmagasabb várható haszonnal járó *cselekvés* kiválasztása
    adott kimenetelekhez tartozó valószínűségek és a hasznosságra vonatkozó információ
  return *cselekvés*

2. Valószínűségi alapfogalmak

Miután felállítottuk a racionális ágens általános keretét, szükségünk lesz egy formális nyelvre a bizonytalan tudás leírásához és a következtetéshez. Minden olyan jelölésnek, amely a meggyőződésünk fokának leírására szolgál, képesnek kell lennie két fő dolog kezelésére: az egyik a kijelentések jellege, amelyekhez meggyőződési mértéket akarunk rendelni, a másik pedig a meggyőződés mértékének az ágens tapasztalatától való függése. A valószínűség-elmélet itt bemutatott változata az ítéletlogika egy kiterjesztését használja állításaihoz. A tapasztalattól való függés az a priori valószínűségi kijelentések és a feltételes valószínűségi állítások szintaktikai megkülönböztetésében tükröződik. Az a priori valószínűségi kijelentéseket akkor alkalmazzuk, mielőtt még tények birtokába jutnánk, míg a feltételes valószínűségi kijelentések explicit módon tartalmazzák a megszerzett tényeket.

Created by XMLmind XSL-FO Converter.

2.1. Állítások

A hiedelmi mértékeket mindig **állítás**okhoz (**proposition**s) rendeljük – amelyek ez és ez a helyzet típusú kijelentések. Az állítások leírására eddig két formális nyelvet – az ítéletlogikát és az elsőrendű logikát – használtuk. Ez az alfejezet egy olyan nyelvet ír le, amelyet a valószínűség-elmélet jellegzetesen használ, és amely valamelyest kifejezőbb az ítéletlogikánál. (A 14.6. alfejezet azokat a módszereket taglalja, amelyek megadják, hogy milyen hiedelmi mértékek tulajdoníthatók az elsőrendű logika egyes kijelentéseinek.)

A nyelv alapeleme a **valószínűségi** vagy **véletlen változó** (**random variable**), ami úgy tekinthető, mint ami egy kezdetben ismeretlen „állapotú" világ egy „részére" vonatkozik. Például a *Lyuk* a bal alsó bölcsességfogam esetleges lyukasságát mutatja. A véletlen változók a kényszerkielégítési problémáknál megismert CSP-változókhoz és az ítéletlogikánál használt ítéletszimbólumhoz hasonló szerepet játszanak.

A véletlen változókat mindig nagybetűvel kezdjük. (Ugyanakkor az ismeretlen véletlen változókat változatlanul kis- és egybetűs nevekkel fogjuk jelölni, például: $P(a) = 1 - P(\neg a)$.)

Minden valószínűségi változóhoz tartozik egy **értéktartomány** (**domain**), amelyből az értékeit veheti. Például a *Lyuk* tartománya az ⟨ *igaz*, *hamis*⟩ lehetne.[2] (Az értékeket kisbetűs nevekkel fogjuk jelölni.) Az állítások legegyszerűbb fajtája azt jelenti ki, hogy a valószínűségi változó valamilyen konkrét értéket vesz fel a tartományon belül. Például a *Lyuk = igaz* azt reprezentálja, hogy nekem valóban lyukas a bal alsó bölcsességfogam.

A véletlen változók – a CSP-változókhoz hasonlóan – tipikusan három csoportba sorolhatók a tartomány fajtájától függően:

- (**Boole-típusú**) **logikai véletlen változó**k (**Boolean random variable**s), mint a *Lyuk*, amelyeknek az ⟨ *igaz*, *hamis*⟩ a tartománya. Az olyan állításokat, mint a *Lyuk = igaz* gyakran rövidítve, csak a kis kezdőbetűs nevével – *lyuk* – fogjuk jelölni, míg a *Lyuk = hamis* állítás rövidített jelölése: ¬*lyuk*.
- **Diszkrét véletlen változó**k (**discrete random variable**s), amelyek speciális esetben logikai változók is lehetnek, egy *megszámlálható* tartományból vesznek fel értéket. Például az *Időjárás* tartománya a ⟨ *napos*, *esős*, *felhős*, *havazik*⟩ lehet. A tartomány értékeinek egymást kizáróknak és összességében kimerítőknek (teljeseknek) kell lenniük. Ha ez nem okozhat félreértést, akkor a *havazik* rövidítés fogja jelölni például az *Időjárás = havazik* állítást.
- **Folytonos véletlen változó**k (**continuous random variable**s), amelyek valós értéket vehetnek fel. A tartomány lehet akár a teljes valós tengely, akár annak egy részhalmaza, mint a [0, 1] intervallum. Például az az állítás, hogy $X = 4{,}02$ azt jelenti ki, hogy az X véletlen változó értéke pontosan 4,02. A véletlen változókra vonatkozó állítások egyenlőtlenségek is lehetnek, mint például $X \leq 4{,}02$.

Néhány kivételtől eltekintve, mi a diszkrét esetre fogunk koncentrálni.

Az összetett állítások létrehozásához az olyan elemi állítások, mint a *Lyuk = igaz* vagy a *Fogfájás = hamis*, bármely szokásos logikai kapcsolat felhasználásával kombinálhatók. Például a *Lyuk = igaz* ∧ *Fogfájás = hamis* egy olyan állítás, amelyhez valamilyen hihetőségi (hihetetlenségi) mértéket rendelhetünk. Ahogy az előző bekezdésben leírtuk, a fenti állítást úgy is jelölhetjük, hogy *fogszuvasodás* ∧ ¬*fogfájás*.

2.2. Elemi események

Az **elemi esemény** (**atomic event**) jelölés hasznos a valószínűség-elmélet alapjainak megértésében. Egy elemi esemény a világ – amely tekintetében az ágens bizonytalan – állapotának egy *teljes* leírását jelenti. Úgy is tekinthetjük, mint a világot alkotó összes változóhoz való konkrét érték hozzárendelését. Például, ha a világomat csak a *Lyuk* és a *Fogfájás* logikai változók alkotják, akkor pontosan négy különböző elemi esemény létezik; amelyek közül a *Lyuk = hamis* ∧ *Fogfájás = igaz* egy esemény.[3]

[2] Egyesek elvárása szerint a tartományt halmazként kellene megadni: {*igaz, hamis*}. Mi állítások *n*-eseként írjuk le, mivel a későbbiekben ez megkönnyíti egy rendezés hozzárendelését.

[3] A valószínűség-elmélet számos formája az elemi eseményt, más néven **mintá**t (**sample point**) egy primitívnek tekinti. A véletlen változót pedig, mint egy függvényt definiálja, amelynek bemenete egy elemi esemény, kimenetként pedig a megfelelő tartomány egy értékét adja. Ez a megközelítés talán általánosabb, de ugyanakkor kevésbé intuitív.

Created by XMLmind XSL-FO Converter.

Az elemi eseményeknek van néhány fontos tulajdonsága:

- Az elemi események *egymást kölcsönösen kizáró* események – legfeljebb egyikük lehet igaz. Például, nem lehet egyszerre igaz a *lyuk* ∧ *fogfájás*, valamint a *lyuk* ∧ ¬*fogfájás*.
- Az összes elemi esemény halmaza kimerítő – legalább az egyiknek igaznak kell lennie. Azaz az összes elemi esemény egyesítése logikailag egyenértékű az *igaz* állítással.
- Minden egyes elemi esemény maga után vonja következményként az összes állítás igazságát vagy hamisságát függetlenül attól, hogy azok egyszerűek vagy összetettek. Ez a logikai kapcsolatok szokásos szemantikájának alkalmazása révén mutatható meg (lásd 7. fejezet). Például a *lyuk* ∧ ¬*fogfájás* elemi esemény következménye a *fogszuvasodás* igaz volta és a *lyuk* ⇒ *fogfájás* hamissága.
- Bármely állítás logikailag egyenértékű azon elemi események diszjunkciójával, amelyekből az állítás következik. Például a *lyuk* állítás ekvivalens a *lyuk* ∧ *fogfájás* és a *lyuk* ∧ ¬*fogfájás* elemi események egyesítésével.

A 13.4. feladat a fenti tulajdonságok bizonyítását célozza.

2.3. A priori valószínűség

Az *a* állításhoz tartozó **feltétel nélküli** (**unconditional**) vagy **a priori valószínűség** (**prior probability**) azt a meggyőződési mértéket jelenti, amely *bármely más információ hiányában* az állításhoz kapcsolható; jelölése *P*(*a*). Például ha 0,1 annak az a priori valószínűsége, hogy van lyukas fogam, akkor

P(*Lyuk = igaz*) = 0,1 vagy *P*(*lyuk*) = 0,1-et írhatunk.

Fontos megjegyeznünk, hogy *P*(*a*) csak akkor használható, ha nincs semmilyen más információ a birtokunkban. Amint ismertté válik valamilyen új információ, a továbbiakban már *a* adott új információ melletti *feltételes* valószínűségével kell következtetnünk. A feltételes valószínűségekkel a következő alfejezet foglalkozik.

Bizonyos esetekben előfordulhat, hogy beszélni szeretnénk egy véletlen változó összes lehetséges értékének valószínűségéről. Ilyen esetekben a **P**(*Időjárás*) kifejezés használható, amely az időjárás minden egyes állapotához rendelt valószínűségi értékekből képzett *vektort* jelöli. Következésképpen, ahelyett hogy az alábbi négy egyenletet írnánk le

P(*Időjárás = napos*) = 0,7

P(*Időjárás = esős*) = 0,2

P(*Időjárás = felhős*) = 0,08

P(*Időjárás = havazik*) = 0,02

elegendő egyszerűen azt írnunk, hogy

P(*Időjárás*) = ⟨0,7, 0,2, 0,08, 0,02⟩

Az ilyen kijelentés az *Időjárás* véletlen változó előzetes **valószínűség-eloszlás**át (**probability distribution**) definiálja.

Olyan kifejezéseket is használni fogunk, mint a **P**(*Időjárás*, *Lyuk*), hogy egy véletlen változóhalmaz összes lehetséges kombinációjának valószínűségeit jelölni tudjuk.[4] Ekkor a **P**(*Időjárás*, *Lyuk*) egy 4 × 2-es valószínűségi táblázatot jelent. Ez az *Időjárás* és *Lyuk* **együttes valószínűség-eloszlás**a (**joint probability distribution**).

Hasznos lehet az is, ha világot leíró véletlen változók teljes halmazáról gondolkozunk. Az olyan együttes valószínűség-eloszlást, amely lefedi a teljes halmazt **teljes együttes valószínűség-eloszlás**nak (**full joint probability distribution**) nevezzük. Például, ha a világ csak a *Lyuk*, a *Fogfájás* és az *Időjárás* változókból áll, akkor a teljes együttes valószínűség-eloszlást a

[4] Az általános jelölési szabály szerint az eloszlás a nagybetűs változók összes értékét tartalmazza. Azaz, a **P**(*Időjárás*, *lyuk*) a valószínűségek egy olyan négyelemű vektorát jelenti, amelyben minden egyes időjárás-állapot valószínűsége szerepel *Lyuk = igaz* mellett.

Created by XMLmind XSL-FO Converter.

```
P(Lyuk, Fogfájás, Időjárás)
```

adja meg. Ez az együttes valószínűség-eloszlás egy 16 elemű, 2 × 2 × 4-es táblázattal reprezentálható. A teljes együttes valószínűség-eloszlás minden egyes elemi esemény valószínűségét, és így *a kérdéses világgal kapcsolatos összes bizonytalanságot meghatározza.* A 13.4. alfejezetben látni fogjuk, hogy a teljes együttes valószínűség-eloszlás alapján bármely valószínűségi kérdés megválaszolható.

Folytonos változók esetén az eloszlás nem foglalható össze táblázatos formában, mivel a lehetséges értékek száma végtelen. Ehelyett annak valószínűsége, hogy egy valószínűségi változó egy adott *x* értéket vesz fel, általában *x* egy paraméterezett függvényeként definiálható. Például az *X* véletlen változó jelölje a holnapi hőmérséklet maximumát Berkeleyben. Ezzel a

```
P(X = x) = U[18, 26](x)
```

kijelentés azt a hiedelmet fejezi ki, hogy *X* egyenletes eloszlást mutat 18 és 26 °C között. (Néhány hasznos folytonos valószínűségi változó definícióját az A) függelékben találjuk meg.) A folytonos valószínűségi változókra vonatkozó valószínűségi eloszlást **valószínűség-sűrűségfüggvény**nek (**probability density function**) nevezzük. A sűrűségfüggvények jelentése különbözik a diszkrét eloszlásokétól. Például a korábbiakban megadott hőmérsékleteloszlásból kiindulva $P(X = 20{,}5) = U[18, 26](20{,}5) = 0{,}125/°C$ adódik. Ez *nem* azt jelenti, hogy annak az esélye, hogy a holnapi maximális hőmérséklet *pontosan* 20,5 °C lesz 12,5%; ennek a valószínűsége természetesen 0. Technikailag ez azt jelenti, hogy annak a valószínűsége, hogy a kérdéses hőmérséklet a 20,5 °C egy kicsiny környezetébe fog esni, határértékét tekintve egyenlő azzal, hogy a 0,125-öt elosztjuk a szakasz °C-ban megadott szélességével:

$$\lim_{dx \to 0} P(20{,}5 \le dx \le X \le 20{,}5 + dx) / dx = 0{,}125 / °C$$

Néhány szerző a diszkrét eloszlások és a sűrűségfüggvények jelölésére más szimbólumot használ; mi *P*-vel fogjuk jelölni mindkettőt, mivel ritkán lehet ezeket összekeverni, és az egyenletek általában azonos formájúak. Jegyezzük meg ugyanakkor, hogy míg a valószínűségek mértékegység nélküli számok, a sűrűségfüggvényeknek van mértékegysége, a fenti példában 1/°C.

2.4. Feltételes valószínűség

Amint az ágens bizonyos tények birtokába jut a korábban ismeretlen, a tartományra jellemző véletlen változóra vonatkozóan, az a priori valószínűségek többé nem használhatók. Ehelyett a **feltételes** (**conditional**) vagy **a posteriori** (**posterior**) valószínűségeket használhatjuk. Jelölése $P(a|\ b)$, ahol *a* és *b* tetszőleges állítás lehet.[5] Értelmezése: *a* valószínűsége, ha *b*-t és csak *b*-t tudjuk. Például,

```
P(Lyuk| Fogfájás) = 0,8
```

azt jelenti, hogy ha egy betegnél megfigyeltük, hogy fogfájása van, és semmilyen más információnk nincs vele kapcsolatban, akkor annak a valószínűsége, hogy szuvas a foga 0,8. Egy *P*(*lyuk*) típusú a priori valószínűség tekinthető a *P*(*lyuk*|) feltételes valószínűség speciális esetének, ahol a feltételt a „semmi bizonyíték” jelenti.

A feltételes valószínűségek megadhatók feltétel nélküliek segítségével. A definíció

$$P(a|b) = \frac{P(a \wedge b)}{P(b)} \qquad (13.1)$$

minden $P(b) > 0$ esetén igaz. Az egyenletet írhatjuk

```
P(a ∧ b) = P(a| b)P(b)
```

alakban is, amelyet **szorzatszabály**nak (**product rule**) hívunk. A szorzatszabályt talán könnyebb megjegyezni: ez abból következik, hogy *a* és *b* együttes teljesüléséhez, szükséges, hogy *b* igaz legyen, valamint hogy *a* is igaz legyen *b* feltétele mellett. A szabályt megadhatjuk fordítva is:

$P(a \wedge b) = P(b|\ a)P(a)$

[5] A „| ” operátor a lehető legkevesebb előzményt jelenti, azaz a $P(a \wedge b|c \vee d)$ $P((a \wedge b)|(c \vee d))$-vel egyenértékű.

Created by XMLmind XSL-FO Converter.

Bizonyos esetekben könnyebb a konjunkciók feltétel nélküli (a priori) valószínűségeit használni, azonban mi az esetek többségében feltételes valószínűségeket fogunk alkalmazni valószínűségi következtetéseink eszközeként.

A **P** jelölést használhatjuk feltételes eloszlásokra is. $\mathbf{P}(X|\ Y)$ a $P(X = x_i|\ Y = y_i)$ értékeit adja meg minden lehetséges i-re, j-re. Annak példájaként, hogy ez mennyivel tömörebbé teszi a jelölést, képzeljük el a szorzatszabály alkalmazását minden olyan esetre, ahol az a és b állítások X és Y bizonyos értékeit veszik fel. A következő egyenleteket fogjuk kapni:

$$P(X = x_1 \wedge Y = y_1) = P(X = x_1|\ Y = y_1) = P(Y = y_1)$$

$$P(X = x_1 \wedge Y = y_2) = P(X = x_1|\ Y = y_2) = P(Y = y_2)$$

Mindezt összefoglalhatjuk egyetlen egyenletben is:

$$P(X,\ Y) = P(X|\ Y)P(Y)$$

Ne felejtsük, hogy ez egy olyan egyenlethalmaz jelölésére szolgál, amely kapcsolatba hozza a táblázatok megfelelő elemeit, és *nem* a táblázatok mátrix szorzásáról van szó. Csábító, de helytelen a feltételes valószínűségeket bizonytalansággal kiegészített logikai implikációkként tekinteni. Például a $P(a|\ b) = 0{,}8$ állítást *nem lehet* úgy értelmezni, hogy amikor csak igaz b, $P(a)$ 0,8-del egyenlő. Ez két okból is téves: először is $P(a)$ mindig a előzetes valószínűségét, és nem a valamilyen tény megléte esetén alkalmazandó utólagos valószínűségét jelöli; másrészről pedig a $P(a|b) = 0{,}8$ csak akkor alkalmazható, ha b az *egyetlen* tény, amelynek birtokában vagyunk. Ha ismerünk egy további c információt is, akkor a hihetőségi mértékét $P(a|\ b \wedge c)$ fogja adni, amely akár független is lehet $P(a|\ b)$-tól. Például c megadhatja közvetlenül azt is, hogy a igaz vagy hamis. Ha megvizsgálunk egy beteget, aki fogfájásra panaszkodik, és találunk egy lyukas fogat, akkor a további, *lyuk* információnak jutottunk birtokába, és így természetesen, arra a következtetésre fogunk jutni, hogy $P(lyuk|\ fogfájás \wedge lyuk) = 1{,}0$.

3. Valószínűségi axiómák

Az eddigiekben az állításokra, az a priori, valamint a feltételes valószínűségi kijelentésekre vonatkozó szintaxist definiáltunk. A továbbiakban meg kell fogalmaznunk a valószínűségi kijelentések valamilyen szemantikáját is. Ezt azokkal az alapvető axiómákkal kezdjük, amelyek a valószínűségi skálát és annak végpontjait határozzák meg:

1. Minden valószínűség 0 és 1 közé esik. Bármely a kijelentésre

$$0 \leq P(a) \leq 1$$

2. A biztosan igaz (azaz érvényes) állítások valószínűsége 1, a biztosan hamis (azaz kielégíthetetlen) állításoké pedig 0.

$$P(igaz) = 1 \quad P(hamis) = 0$$

Ezután egy olyan axiómára van szükségünk, amely a logikailag összefüggő állításokat kapcsolja össze. A legegyszerűbb, ha a diszjunkcióra vonatkozó valószínűséget az alábbiak szerint definiáljuk:

3. A diszjunkció valószínűsége

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b)$$

Ez utóbbi szabályt könnyű megjegyezni arról, hogy azok az esetek, ahol a igaz és azok az esetek, ahol b igaz, együtt természetesen lefedik azokat az eseteket, ahol $a \vee b$ igaz; ugyanakkor a két halmazba eső eseteket összeadva kétszer számoljuk azokat, amelyek a halmazok közös részébe esnek, így ki kell vonnunk $P(a \wedge b)$-t.

Ezt a három axiómát gyakran nevezik **Kolmogorov-axiómák**nak (**Kolmogorov's axioms**) az orosz matematikus Andrei Kolmogorov emlékére, akinek a nevéhez fűződik az ezen alapaxiómákból kiinduló valószínűség-elmélet felépítése. Vegyük észre, hogy az axiómák csak az a priori valószínűségekkel, és nem a feltételes valószínűségekkel foglalkoznak; ennek oka, hogy az utóbbiakat már a korábbiakban definiáltuk a (13.1) egyenlet szerint.

Fontos

Created by XMLmind XSL-FO Converter.

Honnan származnak a valószínűségek?

Vég nélküli viták folytak a valószínűségi számok forrása és értelme körül. A **frekvencionistab(frequentist**) álláspont szerint ezek az értékek csak *kísérletekből* származhatnak: ha 100 ember vizsgálata során az derül ki, hogy 10-nek szuvas a foga, akkor azt mondhatjuk, hogy a szuvas fog valószínűsége körülbelül 0,1. E szemlélet szerint az az állítás, miszerint „a lyuk valószínűsége 0,1" azt jelenti, hogy ha végtelen sok mintát megvizsgálnánk, határértékét tekintve 0,1 lenne annak a résznek a hányada, ahol ezt tapasztalnánk. Tetszőleges, de véges számú minta esetén pedig megbecsülhetjük a valós arányt, valamint kiszámíthatjuk azt is, hogy várhatóan mekkora lesz a becslésünk pontossága.

Az **objektivista** (**objectivist**) megközelítés szerint a valószínűségek inkább az univerzum valóságos jellemzőinek tekinthetők – ahol egy jellemző a dolgok hajlama bizonyos típusú viselkedésre –, mint egy megfigyelő meggyőződési foka leírásának. Például az, hogy egy nem hamis pénzdarab feldobásánál 0,5-es valószínűséggel fej jön ki, magának a pénzdarabnak a hajlandósága. E szemlélet szerint a frekvencionista mérések ezen hajlandóságok megfigyelésére tett kísérletek. A legtöbb fizikus egyetért abban, hogy a kvantumjelenségek valójában valószínűségiek, de makroszkopikus szinten – például a pénzfeldobásnál – a bizonytalanság a kezdeti feltételek elhanyagolása miatt lép fel, ami látszólag nem konzisztens a hajlandóságalapú szemlélettel.

A **szubjektivista** (**subjectivist**) szemlélet szerint a valószínűségek inkább az ágens meggyőződésének jellemzésére szolgálnak, és nincs külső fizikai jelentőségük. E szerint az orvos vagy az elemző aszerint találhatja ki a számokat, hogy mondhassa véleményem szerint a lyuk valószínűsége 0,1 körül várható. Kifejlesztettek olyan további, még megbízhatóbb módszereket is – mint például az 3.1. szakasz - A valószínűségi axiómák használata részben részletezett fogadási rendszereket –, amelyekkel az emberek valószínűség-hozzárendelését szerették volna kiismerni. A **vonatkoztatási osztály** (**reference class**) probléma jól illusztrálja a szubjektivitás jelenlétét. Képzeljünk el egy frekvencionista orvost, aki meg akarja állapítani annak az esélyeit, hogy egy páciens egy bizonyos betegségben szenved. Ehhez az orvos más, de fontos szempontokból (kor, tünet, esetleg nem tekintetében) azonos betegeket akar figyelembe venni, majd megnézni közöttük az ilyen betegségben szenvedők arányát. Azonban, ha az orvos mindent figyelembe vesz, amit ismer a beteggel kapcsolatban – a grammra pontos testsúlyát, a haja színét, az anyja leánykori nevét stb. – arra az eredményre fog jutni, hogy nincs még egy olyan ember, aki pontosan megegyezne a beteggel, és így nincs olyan vonatkoztatási osztály sem, amelyről kísérleti eredményeket lehetne gyűjteni. Mindez nyugtalanító kérdése a tudományfilozófiának.

Laplace **közömbösségi elve** (**principle of indifference**) (Laplace, 1816) szerint a tény tekintetében szintaktikailag „szimmetrikus" állításokhoz azonos valószínűséget kell rendelni. Erre vonatkozóan történtek különböző finomítási javaslatok, amelyek Carnap és más filozófusok azon törekvéseiben csúcsosodtak ki, hogy egy olyan precíz **induktív logikát** (**inductive logic**) fejlesszenek ki, amely képes bármely állítás helyes valószínűségét tetszőleges megfigyelések alapján meghatározni. Jelenlegi felfogásunk szerint nem létezik különálló induktív logika; inkább úgy gondoljuk, hogy minden ilyen logika egy szubjektív a priori valószínűségi eloszláson alapszik, amelynek hatása a megfigyelések számának növekedésével csökken.

3.1. A valószínűségi axiómák használata

A valószínűségi axiómákból nagyon sok hasznos tény vezethető le. Például a tagadásra vonatkozó jól ismert szabály következik, ha $\neg a$-val helyettesítjük b-t a harmadik axiómában:

$P(a \vee \neg a) = P(a) + P(\neg a) - P(a \wedge \neg a)$ (a 3. axiómából, $b = \neg a$ helyettesítéssel)

$P(igaz) = P(a) + P(\neg a) - P(hamis)$ (logikai ekvivalencia alapján)

$1 = P(a) + P(\neg a)$ (a 2. axiómából)

$P(\neg a) = 1 - P(a)$ (algebrai műveletekkel)

A következtetés harmadik sora önmagában is hasznos tény és kiterjeszthető a Boole-típusú logikai esetből az általános diszkrét esetre. Legyen $\langle d_1, \ldots, d_n \rangle$ a D diszkrét változó értelmezési tartománya. Könnyű megmutatni (13.2. feladat), hogy

Created by XMLmind XSL-FO Converter.

$$\sum_{i=1}^{n} P(D = d_i) = 1$$

Azaz, egyetlen változó tetszőleges valószínűségi eloszlása összegzésének 1-et kell adnia.[6] Igaz az is, hogy változók tetszőleges *halmazának* bármely *együttes* valószínűségi eloszlását összegezve, eredményül szintén 1-et kapunk: ennek belátására elegendő egyetlen többdimenziós, az eredeti változók értelmezési tartományai Descartes-szorzataként létrehozott tartomány felett értelmezett változót létrehoznunk.

Fontos

Emlékezzünk, hogy bármely *a* állítás egyenértékű minden olyan elemi esemény diszjunkciójával, ahol *a* igaz; nevezzük az események ezen halmazát **e**(*a*)-nak. Emlékezzünk arra is, hogy az elemi események egymást kölcsönösen kizárják, így a 2. axióma szerint az ilyen események konjunkciója nulla. Következésképpen a 3. axiómából a következő egyszerű összefüggésre juthatunk: *Egy állítás valószínűsége megegyezik azon elemi események valószínűségeinek összegével, ahol az állítás igaz*, azaz

$$P(a) = \sum_{e_i \in e(a)} P(e_i) \qquad (13.2)$$

Ez az egyenlet egyszerű módszert kínál bármilyen állítás valószínűségének meghatározására, ha adott az összes elemi esemény valószínűségét meghatározó teljes együttes valószínűség-eloszlás (lásd 13.4. alfejezet). A következő alfejezetekben a valószínűségek kezelésének további szabályait fogjuk származtatni. Mindenekelőtt azonban az axiómák megalapozottságát vizsgáljuk meg.

3.2. Ami a valószínűségi axiómákat indokolja

A valószínűségi axiómák tekinthetők az ágens által elhihető valószínűségi hiedelmek korlátainak. Ez valamennyire hasonlít a logikai esethez, amikor például egy logikai ágens nem hihet egyszerre *A*-ban, *B*-ben és ¬(*A* ∧ *B*)-ben. De van itt egy további bonyodalom is. A logikai esetben az együttes bekövetkezés definíciójának jelentése szerint a fenti állítások közül legalább egynek *hamisnak kell lennie a valóságban,* vagyis egy ágens számára ésszerűtlen mindháromban hinni. A valószínűségeket tekintve azonban a kijelentések nem közvetlenül a valóságra vonatkoznak, hanem az ágens saját ismereteire. Akkor miért nem hihet egy ágens a meggyőződések következő – a harmadik axiómát egyértelműen megsértő valószínűség-hozzárendelések – halmazában?

```
P(a) = 0,4 P(a ∧ b) = 0,0

P(b) = 0,3 P(a ∨ b) = 0,8 (13.3)
```

Az effajta kérdés évtizedekig vita tárgya volt azok között, akik a valószínűségek használatát tartották egyedül jogosnak a meggyőződési mértékek kifejezésére, és azok között, akik más megközelítési módokat javasoltak. Itt most egy olyan érvelést mutatunk be a valószínűségi axiómákkal kapcsolatban, amelyet először Bruno de Finetti fejtett ki 1931-ben.

De Finetti érvelésének kulcsa a meggyőződési fok és a cselekvések közötti összefüggés. A gondolat lényege, hogy ha egy ágensnek van valamilyen szintű meggyőződése egy *a* állítással kapcsolatban, akkor képesnek kell lennie olyan tételeket megadni, amelyek mellett mindegy, hogy az *a* állításra, vagy ellene fogad. Képzeljük el úgy, mint egy játszmát két ágens között: az 1. ágens szerint az én hiedelmem az *a* eseményben 0,4. A 2. ágens ezután szabadon dönthet, hogy *a* mellett vagy ellen fogad akkora téttel, amely a nyilatkozott hiedelemfokkal konzisztens. Vagyis a 2. ágens választhatja azt, hogy arra fogad, hogy *a* esemény bekövetkezik, 4 dollárt téve az 1. ágens 6 dollárjával szemben, vagy tehet 6 dollárt az 1. ágens 4 dollárja ellenében arra fogadva, hogy *a* nem fog bekövetkezni.[7] Ha egy ágens meggyőződésében nem tükröződik pontosan a világ ismerete, akkor számíthatunk arra, hogy hosszú távon veszíteni fog a szemben álló ágenssel szemben, akinek a hiedelmei pontosabb képet mutatnak a világ állapotáról.

---

[6] Folytonos változóknál az összegzést integrállal kell helyettesíteni: $\int_{-\infty}^{\infty} P(X = x)\, dx = 1$

[7] Érvelhetünk azzal, hogy az ágens döntését a bankviszonyok fogják meghatározni, azaz 1 dollár elvesztése nincs kiegyensúlyozva 1 dollár azonos valószínűség mellett való megnyerésével. Kellően kicsire választva azonban a tétet, ez a probléma elkerülhető. Savage (Savage, 1954) analízise az egész problémát elkerüli.

Created by XMLmind XSL-FO Converter.

Fontos

De Finetti ennél sokkal erősebb állítást is bizonyított: *ha az 1. ágens olyan meggyőződési mértékekben hisz, amelyek megsértik a valószínűség-számítás axiómáit, akkor létezik olyan stratégia a 2. ágens számára, amely* **biztosítja**, *hogy az 1. ágens* **minden** *alkalommal veszíteni fog.* Tehát ha elfogadjuk, hogy egy ágensnek a valószínűségeknek megfelelően kell feltennie a pénzét, akkor azt is el kell fogadnunk, hogy ésszerűtlen olyan meggyőződéseket vallani, amelyek megsértik a valószínűségi axiómákat.

Azt gondolhatjuk, hogy ez a fogadósdi nagyon körmönfont. Mi van például, ha valaki megtagadja, hogy fogadjon? Elrontja-e ez az egész érvelést? Minderre az a válasz, hogy a fogadásos játék elvont modellje annak a döntési helyzetnek, amelyben minden ágens minden pillanatban *elkerülhetetlenül* részt vesz. Minden cselekvés (beleértve a tétlenséget is) egyfajta fogadás, és a következmény tekinthető a fogadás nyereményének. Épp úgy nem utasíthatjuk vissza a fogadást, mint ahogy nem állíthatjuk meg az idő múlását.

Nem mutatjuk be de Finetti elméletének bizonyítását, de mutatunk rá egy *példát. Tegyük fel, hogy az 1. ágens meggyőződése megfelel a* (*13.3*) *egyenletnek. A* 13.2. ábra azt mutatja, hogy ha a 2. ágens azt választja, hogy 4 dollárt tesz a-ra, 3 dollárt b-re és 2 dollárt $\neg(a \vee b)$-re, akkor az 1. ágens mindig veszíteni fog, függetlenül attól, hogy a vagy b fog bekövetkezni.

13.2. ábra - Mivel az 1. ágens meggyőződése nem következetes, ezért a 2. ágens tud úgy tételeket tenni, amely *a* és *b* kimenetelétől függetlenül biztosítja, hogy az 1. ágens veszítsen

1. ágens		2. ágens		Kimenetel az 1. ágens számára			
Állítás	Meggyőződés	Fogadás	Tét	$a \wedge b$	$a \wedge \neg b$	$\neg a \wedge b$	$\neg a \wedge \neg b$
a	0,4	a	4–6 között	–6	–6	4	4
b	0,3	b	3–7 között	–7	3	–7	3
$a \vee b$	0,8	$\neg (a \vee b)$	2–8 között	2	2	2	–8
				–11	–1	–1	–1

Más, erősen filozofikus érveléseket is előterjesztettek a valószínűségek használata mellett, amelyek közül a legnevezetesebb Cox és Carnap nevéhez fűződik (Cox, 1946; Carnap, 1950). Lévén a világ olyan, amilyen, a gyakorlati példák sokszor minden bizonyítéknál erősebbek. Így a valószínűség-számításon alapuló következtető rendszerek sikere minden érvelésnél hatékonyabb volt a hitetlenek megtérítésében. A következőkben azt vizsgáljuk meg, hogy az axiómák hogyan használhatók a következtetésben.

4. Teljes együttes valószínűség-eloszláson alapuló következtetés

Ebben az alfejezetben a **valószínűségi következtetés** (**probabilistic inference**) egy egyszerű módszerét fogjuk leírni – hogy hogyan határozhatók meg az állítások a posteriori valószínűségekre vonatkozó megfigyelt bizonyítékok alapján. „Tudásbázisként”, a teljes együttes valószínűség-eloszlást fogjuk használni, amelyből az összes kérdésre adandó válasz levezethető. Menet közben számos hasznos, a valószínűségeket tartalmazó egyenletek kezelésére alkalmas módszert is bevezetünk.

Kezdjük egy egészen egyszerű példával, egy olyan tartománnyal, amely mindössze három logikai változóból áll. Ezek: *Fogfájás*, *Lyuk*, *Beakadás* (a fogorvos kellemetlen acélszondája beleakad a fogamba). A teljes együttes eloszlás a 13.3. ábra szerinti 2 × 2 × 2-es táblázatból fog állni.

13.3. ábra - A *Fogfájás, Lyuk, Beakadás* világ egy teljes együttes valószínűség-eloszlása

Created by XMLmind XSL-FO Converter.

	fogfájás		¬ *fogfájás*	
	beakadás	¬ *beakadás*	*beakadás*	¬ *beakadás*
lyuk	0,108	0,012	0,072	0,008
¬ *lyuk*	0,016	0,064	0,144	0,576

Vegyük észre, hogy az együttes valószínűség-eloszlásban – a valószínűségi axiómáknak megfelelően – a valószínűségek összege 1. Hasonlóképpen, a (13.2) egyenlet közvetlen módot nyújt bármely, egyszerű vagy összetett állítás valószínűségének a meghatározására: egyszerűen azokat az elemi eseményeket kell meghatároznunk, amelyekben az állítás igaz, majd összegeznünk kell a hozzájuk rendelt valószínűségeket. Például, hat olyan elemi esemény van, amelyben a *lyuk* ∨ *fogfájás* igaz:

$$P(lyuk \vee fogfájás) = 0,108 + 0,012 + 0,072 + 0,008 + 0,016 + 0,064 = 0,28$$

Az egyik általános feladat a változók egy részhalmaza vagy egyetlen változó fölötti valószínűségi eloszlás kifejezése. Például az első sor bejegyzéseinek összege a *lyuk* feltétel nélküli vagy **peremeloszlás**át[8] (**marginal probability**) adja:

$$P(lyuk) = 0,108 + 0,012 + 0,072 + 0,008 = 0,2$$

Ezt a folyamatot **marginalizálás**nak (**marginalization**) vagy **kiátlagolás**nak (**summing out**) hívjuk, mivel a *Lyuk*-on kívüli változókat „kiátlagoljuk”. A következő általános behatárolási szabályt fogalmazhatjuk meg a változók tetszőleges **Y** és **Z** halmaza esetén:

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y}, \mathbf{z}) \qquad (13.4)$$

Azaz, egy **Y** feletti eloszlás megkapható, ha az összes többi változót kiátlagoljuk az **Y**-t tartalmazó együttes eloszlásokból a szorzat szabályban

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y} \mid \mathbf{z}) P(\mathbf{z}) \qquad (13.5)$$

Ezt a szabályt **feltételfeloldás**nak (**conditioning**) hívjuk. A marginalizálás és a feltételfeloldás szabályai előnyöseknek fognak bizonyulni mindenfajta valószínűségi kifejezéseket tartalmazó következtetésben.

A legtöbb esetben valamely változó, bizonyos másokra vonatkozó tények esetén fennálló, *feltételes* valószínűségének kiszámítása fog bennünket érdekelni. A feltételes valószínűségek meghatározásához, először is alkalmazzuk a (13.1) egyenletet, hogy egy feltétel nélküli valószínűségen alapuló kifejezésre jussunk, majd ezen kifejezés értékét meghatározzuk a teljes együttes eloszlásból. Például a lyuk valószínűségét a fogfájás tény fennállása esetére az alábbiak szerint határozhatjuk meg:

$$P(lyuk \mid fogfájás) = \frac{P(lyuk \wedge fogfájás)}{P(fogfájás)} = \frac{0,108 + 0,012}{0,108 + 0,012 + 0,016 + 0,064} = 0,6$$

Ellenőrzésképpen kiszámíthatjuk annak valószínűségét, hogy nincs lyuk, de tudjuk, hogy fogfájás igen:

$$P(\neg lyuk \mid fogfájás) = \frac{P(\neg lyuk \wedge fogfájás)}{P(fogfájás)} = \frac{0,016 + 0,064}{0,108 + 0,012 + 0,016 + 0,064} = 0,4$$

Vegyük észre, hogy a fenti két számításban az $1/P(fogfájás)$ konstans értékű marad függetlenül attól, hogy a *Lyuk* mely értékét számítjuk. Valójában ez egy, a $\mathbf{P}(Lyuk \mid fogfájás)$-t **normalizálás** (**normalization**) konstansnak tekinthető, amely biztosítja, hogy a feltételes valószínűségek összege 1 lesz. A valószínűségekkel foglalkozó fejezetekben az ilyen konstansokat α-val fogjuk jelölni. E jelölés segítségével a két előző kifejezés egybefoglalható:

[8] Azért hívják így, mert a biztosítási matematikusok/statisztikusok általános gyakorlata szerint a megfigyelt gyakoriságokat a biztosítási táblázatok szélére (margójára) szokták írni.

Created by XMLmind XSL-FO Converter.

$$\mathbf{P}(Lyuk \mid fogfájás) = \alpha \mathbf{P}(Lyuk, fogfájás)$$

$$= \alpha[\mathbf{P}(Lyuk, fogfájás, beakadás) + \mathbf{P}(Lyuk, fogfájás, \neg beakadás)]$$

$$= \alpha[\langle 0{,}108,\ 0{,}016\rangle + \langle 0{,}012,\ 0{,}064\rangle = \alpha\langle 0{,}12,\ 0{,}08\rangle = \langle 0{,}6,\ 0{,}4\rangle$$

A normalizálás sok valószínűségi számításnál hasznos rövidítésnek bizonyul.

A példából levezethető egy általános következtetési eljárás. Ragaszkodjunk ahhoz az esethez, ahol a keresés egy változót érint. A következő jelölésekre lesz még szükségünk: jelölje X a keresés változóját (a példában ez a *Lyuk*), **E** jelentse a tény változók halmazát (a példában a *Fogfájás* az egyetlen ilyen), **e** ezek megfigyelt értékét mutatja és a többi, meg nem figyelt változót **Y** foglalja magában (a példa esetében csak a *Beakadás* tartozik ide). $\mathbf{P}(X \mid \mathbf{e})$-t keressük, és a következőképpen számíthatjuk ki:

$$\mathbf{P}(X \mid \mathbf{e}) = \alpha \mathbf{P}(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y}) \qquad (13.6)$$

ahol az összegzést az összes lehetséges **y** fölött végezzük (azaz az **Y** meg nem figyelt változók értékeinek összes lehetséges kombinációja esetén). Vegyük észre, hogy az X, **E** és **Y** változó együttesen a változók teljes halmazát alkotja az adott tárgytartományban, így $\mathbf{P}(X, \mathbf{e}, \mathbf{y})$ egyszerűen a teljes valószínűségi eloszlás egy valószínűségi részhalmaza. Az algoritmust a 13.4. ábra mutatja.

13.4. ábra - A valószínűségi következtetés egy olyan algoritmusa, amely valamely teljes együttes valószínűségi eloszlás bejegyzéseit veszi számba

function FELSOROL-EGYÜTTES-KÉRDEZÉS(X, **e**, **P**) **returns** egy X feletti valószínűségi eloszlás
  inputs: X, a lekérdezés változója
    e, az **E**-ben levő változók megfigyelt értékei
    P, az $\{X\} \cup \mathbf{E} \cup \mathbf{Y}$ változók együttes eloszlása /* **Y** = *rejtett változók* */

  $\mathbf{Q}(X) \leftarrow X$ fölötti valószínűség-eloszlás, kezdetben üres
  for each X x_i értékére **do**
    $\mathbf{Q}(x_i) \leftarrow$ FELSOROL-EGYÜTTES(x_i, **e**, **Y**, [], **P**)
  return NORMALIZÁL($\mathbf{Q}(X)$)

---

function FELSOROL-EGYÜTTES(x, **e**, *változók*, *értékek*, **P**) **returns** egy valós szám
  if ÜRES?(*változók*) **then return** $\mathbf{P}(x, \mathbf{e}, értékek)$
  $Y \leftarrow$ ELSŐ(*változók*)
  return Σ_y FELSOROL-EGYÜTTES(x, **e**, MARADÉK(*változók*), $[y|értékek]$, **P**)

Az eljárás ciklusa mind X, mind Y összes értéke fölött fut, számba véve minden olyan elemi eseményt, amely **e** rögzített értéke mellett lehetséges, az együttes valószínűségi tábla alapján összegzi ezek valószínűségeit, és normalizálja az eredményeket.

Ha adott a teljes együttes valószínűségi eloszlás, akkor a `FELSOROL-EGYÜTTES-KÉRDEZÉS` egy teljes algoritmus, amely alkalmas minden diszkrét változóra vonatkozó valószínűségi lekérdezés megválaszolására. Azonban nem minősíthető jónak: egy n db Boole-típusú logikai változó által leírt tartományban $O(2^n)$ méretű bemeneti táblára van szüksége, és a tábla feldolgozása is $O(2^n)$ időt igényel. Valós problémák esetén nem csupán három, hanem több száz vagy több ezer véletlen változó figyelembevételére van szükség. Nagyon hamar teljesen kivitelezhetetlenné válik a szükséges hatalmas számú valószínűség definiálása – nem létezik ugyanis olyan tapasztalat, amellyel a táblázat egyes bejegyzései külön-külön becsülhetők lennének.

Ezért a táblázatos formában megadott teljes együttes valószínűségi eloszlás a gyakorlatban nem használható eszköz következtető rendszerek felépítésére (mindemellett a fejezet végén található történeti megjegyzésekben találkozhatunk egy ilyen módszeren alapuló valós alkalmazással). E technikát úgy kell tekintenünk, mint egy elméleti alapot, amelyre a hatékonyabb módszerek épülhetnek. A fejezet további részeiben bemutatjuk azokat az alapötleteket, amelyekre szükségünk lesz a 14. fejezet valós rendszereinek kifejlesztéséhez.

Created by XMLmind XSL-FO Converter.

5. Függetlenség

Bővítsük ki a 13.3. ábrán látható teljes együttes valószínűségi eloszlást egy negyedik, *Időjárás* változóval. Ezzel a teljes együttes valószínűségi eloszlás **P**(*Fogfájás*, *Beakadás*, *Lyuk*, *Időjárás*) lesz 32 bejegyzéssel (mivel az *Időjárás* változónak négy értéke lehet). A teljes eloszlás a 13.3. ábrán mutatott táblázat négy „példányát” – minden egyes időjárástípusra egyet – fogja magában foglalni. Természetes kérdésnek tűnik, hogy vajon ezeknek a táblázatpéldányoknak mi az egymáshoz és az eredeti háromváltozós táblához való viszonyuk? Hogyan függ össze például a *P*(*fogfájás*, *beakadás*, *lyuk*, *Időjárás* = *felhős*) a *P*(*fogfájás*, *beakadás*, *lyuk*)-kal?

A megválaszolás egyik lehetséges módja a szorzatszabály alkalmazása:

$$P(\text{fogfájás}, \text{beakadás}, \text{lyuk}, \text{Időjárás} = \text{felhős}) = P(\text{Időjárás} = \text{felhős} | \text{fogfájás}, \text{beakadás}, \text{lyuk}) P(\text{fogfájás}, \text{beakadás}, \text{lyuk})$$

Amennyiben eltekintünk az isteni hatalommal bíró páciensektől, nem gondolhatjuk komolyan, hogy valakinek a fogászati problémái befolyásolják az időjárást. Ezért ésszerűnek tűnik a következő kijelentés:

$$P(\text{Időjárás} = \text{felhős} | \text{fogfájás}, \text{beakadás}, \text{lyuk}) = P(\text{Időjárás} = \text{felhős}) \quad (13.7)$$

Ebből a következőre következtethetünk:

$$P(\text{fogfájás}, \text{beakadás}, \text{lyuk}, \text{Időjárás} = \text{felhős}) = P(\text{Időjárás} = \text{felhős}) P(\text{fogfájás}, \text{beakadás}, \text{lyuk})$$

Hasonló egyenlet írható fel a **P**(*Fogfájás*, *Beakadás*, *Lyuk*, *Időjárás*) eloszlás minden egyes bejegyzéséhez. Az általános egyenletet pedig a következőképpen írhatjuk fel:

$$P(\text{Fogfájás}, \text{Beakadás}, \text{Lyuk}, \text{Időjárás}) = P(\text{Fogfájás}, \text{Beakadás}, \text{Lyuk}) P(\text{Időjárás})$$

Ebből következően a négyváltozós, 32 elemes táblázat létrehozható egy 8 és egy 4 elemes táblázatból. A dekompozíció sematikus vázlatát a 13.5. (a) ábra mutatja.

Azt a tulajdonságot, ami alapján a (13.7) egyenletet felírtuk **függetlenség**nek (más szóval **marginális függetlenség**nek vagy **abszolút függetlenség**nek) (**independence**, **marginal independence**, **absolute independence**) nevezzük. Esetünkben az időjárás független bárkinek a fogászati problémáitól. Az *a* és *b* állítások függetlensége felírható:

$$P(a | b) = P(a) \text{ vagy } P(b | a) = P(b) \text{ vagy } P(a \wedge b) = P(a) P(b) \quad (13.8)$$

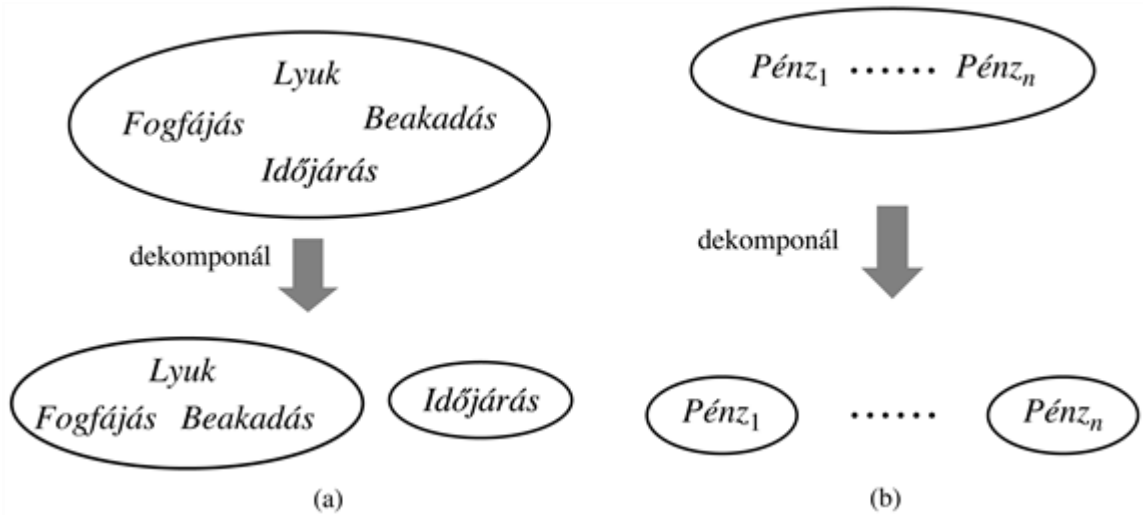
A fenti képletek egymással ekvivalensek (13.7. feladat). Az *X* és *Y* változók közötti függetlenség a következőképpen fejezhető ki (még egyszer felhívjuk a figyelmet, hogy ezek ekvivalensek):

$$P(X | Y) = P(X) \text{ vagy } P(Y | X) = P(Y) \text{ vagy } P(X, Y) = P(X) P(Y)$$

A függetlenségi állítások általában a tartománnyal kapcsolatos ismereteken alapulnak. Mint korábban láthattuk, ezek jelentősen lecsökkenthetik a teljes együttes eloszlás meghatározásához szükséges információ mennyiségét. Ha a változók teljes halmaza szétbontható független részhalmazokra, akkor a teljes együttes eloszlás felbontható ezen részhalmazok felett értelmezett, egymástól független együttes eloszlásokra. Például *n* független pénzfeldobás eredményének együttes eloszlása **P**(C_1, …, C_n) felírható *n* egyváltozós **P**(C_i) eloszlás szorzataként. Még kézzelfoghatóbban: a fogászat és a meteorológia függetlensége jó dolog, különben a fogászati gyakorlat mély meteorológiai ismereteket igényelne, és fordítva.

13.5. ábra - Két példa nagyméretű együttes eloszlások kisebb eloszlásokra való felbontására az abszolút függetlenség alapján. (a) Az időjárás és a fogászati problémák függetlenek. (b) A pénzfeldobások függetlenek.

Created by XMLmind XSL-FO Converter.



Amennyiben rendelkezésre állnak, akkor a függetlenségi állítások segíthetnek lecsökkenteni a tartományleírások méretét, valamint a következtetési feladat bonyolultságát. Sajnos azonban a változók teljes halmazait csak nagyon ritkán lehet függetlenség alapján egyértelműen szétválasztani. A függetlenség nem lesz igaz az olyan esetekben, amikor bármilyen, akár indirekt kapcsolat is, de fennáll két változó között. Ezen kívül, a független részhalmazok mérete is lehet egészen nagy – például a fogászat problémaköre egymással összefüggő betegségek tucatjait és tünetek százait takarhatja. Az ilyen típusú problémák kezelésére a függetlenség közvetlen kimondásánál kifinomultabb módszerekre lesz szükségünk.

6. A Bayes-tétel és használata

Az 2.3. szakasz - A priori valószínűség részben definiáltuk a **szorzatszabály**t (**product rule**), és rámutattunk, hogy ez a konjunkció kommutativitása miatt két alakban írható:

$$P(a \wedge b) = P(a \mid b)P(b)$$

$$P(a \wedge b) = P(b \mid a)P(a)$$

A jobb oldalak egyenlőségéből $P(a)$-val való osztás után következik, hogy

$$P(b|a) = \frac{P(a|b)P(b)}{P(a)} \qquad (13.9)$$

Ezt az egyenletet **Bayes-szabály**ként (**Bayes' rule**) ismerjük (nevezik Bayes-törvénynek vagy Bayes-tételnek is).[9] Ez az egyszerű egyenlet az alapja az összes korszerű valószínűségi következtetést alkalmazó MI-rendszernek. A többértékű változókat tartalmazó általánosabb eset a **P** jelölésekkel a következő átírással adható meg:

$$\mathbf{P}(Y|X) = \frac{\mathbf{P}(X|Y)\mathbf{P}(Y)}{\mathbf{P}(X)}$$

amelyet ismételten úgy kell értelmezni, hogy egy egyenlethalmazt képvisel, amely egyenletek mindegyike a változók meghatározott értékeire vonatkozik. Lesznek olyan esetek, amikor egy még általánosabb felírást használhatunk, valamilyen **e** háttértény feltételével:

$$\mathbf{P}(Y|X,\mathbf{e}) = \frac{\mathbf{P}(X|Y,\mathbf{e})\mathbf{P}(Y|\mathbf{e})}{\mathbf{P}(X|\mathbf{e})}$$

6.1. Bayes tételének alkalmazása: egyszerű eset

[9] Az angol irodalomban Bayes' rule az elnevezése. Strunk és White *The Elements of Style* c. könyvének 1. oldalán található 1. szabály értelmében a Bayes' helyett inkább a Bayes's lenne a helyes jelölés, azonban az előbbit elterjedtebben használják. (Szerencsére ilyen probléma a magyarban nincs. *A szerk.*)

Created by XMLmind XSL-FO Converter.

A Bayes-tétel első pillantásra nem tűnik túl használhatónak. Egyetlen feltételes valószínűség kiszámításához három kifejezés – egy feltételes és két feltétel nélküli valószínűség – megadása szükséges.

A gyakorlatban a Bayes-tétel jól használható, mivel gyakran rendelkezünk a fenti három kifejezésre vonatkozó jó valószínűségi becsléssel, miközben a negyediket kell kiszámítanunk. Olyan feladatoknál, mint az orvosi diagnosztika, gyakran ismerjük az ok-okozati kapcsolatok feltételes valószínűségeit, miközben egy diagnózist szeretnénk felállítani. Az orvos tudja azt, hogy az agyhártyagyulladás az esetek mondjuk 50%-ában nyakmerevedést okoz a betegeknél. Az orvos ezenfelül ismer néhány feltétel nélküli tényt is: annak előzetes valószínűsége, hogy egy beteg agyhártyagyulladást kap, 1/50 000, míg annak előzetes valószínűsége, hogy egy betegnek merev a nyaka 1/20. Jelölje *s* azt az állítást, hogy a betegnek megmerevedett a nyaka, valamint *m* azt az állítást, hogy a betegnek agyhártyagyulladása van. Ekkor

$$P(s|m) = 0,5$$
$$P(m) = 1/50000$$
$$P(s) = 1/20$$
$$P(m|s) = \frac{P(s|m)P(m)}{P(s)} = \frac{0,5 \times 1/50000}{1/20} = 0,0002$$

Vagyis a nyakmerevedésről panaszkodó 5000 beteg közül várhatóan csak egynek lesz agyhártyagyulladása. Vegyük észre, hogy annak ellenére, hogy az agyhártyagyulladásnak igen gyakori tünete (0,5 valószínűséggel) a nyakmerevedés, annak valószínűsége, hogy egy nyakmerevedéses betegnek ténylegesen agyhártyagyulladása van, mégis csekély. Ez abból következik, hogy a nyakmerevedés a priori valószínűsége sokkal nagyobb, mint az agyhártyagyulladásé.

A 13.4. alfejezetben bemutattunk egy eljárást, amely segítségével elkerülhető a tény valószínűségének (példánkban $P(s)$) megbecsülése, úgy, hogy a lekérdezett változó minden egyes értékéhez (itt m és $\neg m$) egy utólagos valószínűséget számítunk ki, majd az eredményeket normalizáljuk. Hasonló eljárás alkalmazható, ha a Bayes-tételt használjuk. Ismert, hogy

$$P(M|s) = \alpha\langle P(s|m)P(m), P(s|\neg m)P(\neg m)\rangle$$

Következésképpen ahhoz, hogy ezt a megközelítést használni tudjuk, $P(s)$ helyett $P(s|\neg m)$-et kell tudnunk becsülni. Nincs ingyenebéd – van, hogy ez könnyebb és van, hogy nehezebb. A normalizált Bayes-tétel általános alakja

$$P(Y|X) = \alpha P(X\backslash Y)P(Y) \quad (13.11)$$

ahol *a* a normalizáló konstanst jelöli, amely segítségével a $\mathbf{P}(Y|X)$ elemeinek összegét 1-gyé tudjuk tenni.

Fontos

A Bayes-tétellel kapcsolatban magától értetődő kérdés, hogy miért ismerhetjük a feltételes valószínűséget az egyik irányból, a másik irányból pedig nem. Az agyhártyagyulladás tartományban a doktor tudhatja, hogy a nyakmerevedésből csak minden 5000-dik esetben következik agyhártyagyulladás, vagyis a doktor a tünetektől a kiváltó ok felé, azaz **diagnosztikai** (**diagnostic**) irányban rendelkezik mennyiségi információval. Egy ilyen orvosnak nincs szüksége a Bayes-tétel alkalmazására. Sajnálatos módon azonban a *diagnosztikai tudás gyakran sokkal törékenyebbnek bizonyul az ok-okozati összefüggéseknél.* Ha például hirtelen agyhártyagyulladás-járvány tör ki, akkor az agyhártyagyulladás előzetes valószínűsége, $P(m)$ megnő. Az az orvos, aki a járványt megelőző statisztikai adatok alapján számította ki $P(m|s)$-t, nem fogja tudni, hogyan változtassa meg az agyhártyagyulladásra vonatkozó értéket, míg az a doktor, aki a másik három érték segítségével számítja $P(m|s)$-t, látni fogja, hogy ennek értéke $P(m)$-mel arányosan meg fog nőni. Még fontosabb azonban, hogy a $P(s|m)$ okozati információ értékét a járvány *nem befolyásolja,* hiszen ez kizárólag az agyhártyagyulladás lefolyásától függ. Ez a fajta közvetlen ok-okozati és modellalapú tudás alkalmazása teszi a valószínűségi rendszereket döntően robusztussá, amely a valódi világban való felhasználhatósághoz szükséges.

6.2. A Bayes-tétel alkalmazása: több együttes tény figyelembevétele

Created by XMLmind XSL-FO Converter.

Láttuk, hogy a Bayes-tétel hasznos lehet az egyetlen tény – például nyakmerevedés – feltételezése melletti valószínűségi kérdések megválaszolásánál. Nevezetesen, megmutattuk, hogy a valószínűségi információ gyakran *P*(*okozat*| *ok*) formában áll rendelkezésre. Mi történik azonban akkor, ha kettő vagy több tény van a birtokunkban? Például milyen következtetésre juthat a fogorvos, ha az az undok acélszondája lyukra akad a beteg fájó fogában? Ha ismerjük a teljes együttes eloszlást (lásd 13.3. ábra), a válasz kiolvasható:

$$P(Lyuk | fogfájás \wedge beakadás) = \alpha\langle 0{,}018,\ 0{,}016\rangle \approx \langle 0{,}871,\ 0{,}129\rangle$$

Azt is tudjuk ugyanakkor, hogy ez a megközelítés nagyszámú változó esetén nem használható.

Megpróbálkozhatunk a Bayes-tétel alkalmazásával is, átfogalmazva a kérdést:

$$P(Lyuk | fogfájás \wedge beakadás) = \alpha P(fogfájás \wedge beakadás | Lyuk) P(Lyuk) \quad (13.12)$$

Ahhoz, hogy ez az átfogalmazás működjön, a *Lyuk* minden értékére ismernünk kell a *fogfájás* ∧ *beakadás* együttes bekövetkezésének feltételes valószínűségét. Noha ez két tényváltozó esetén használható lehet, nagyszámú változó esetén már nem alkalmazható. Ha *n* figyelembe veendő tényváltozónk van (röntgen, diéta, szájhigiénia stb.), akkor a megfigyelt értékek lehetséges kombinációinak száma 2^n, amely esetek mindegyikénél ismernünk kell a feltételes valószínűséget. Ennyi erővel akár vissza is térhetünk a teljes együttes valószínűség-eloszlás használatához. Ez vezetett arra, hogy a kutatók a valószínűség-számítás helyett közelítő módszereket kezdtek el használni több tény együttes figyelembevételénél, mert bár az így kapott válaszok pontatlanok, de kevesebb számolást igényelnek.

A fenti út követése helyett inkább további állításokat kell keresnünk a tárgytartományról, amelyek lehetővé teszik a kifejezések egyszerűsítését. A 13.5. alfejezetben bevezetett **függetlenség** (**independence**) fogalma kínálja a megoldás kulcsát, azonban finomítást igényel. Kellemes lenne, ha a *Fogfájás* és a *Beakadás* függetlenek lennének, de nem azok: ha a szonda megakad a fogban, akkor az valószínűleg lyukas, ami várhatóan fájdalmat okoz. Mindemellett, ezek a változók *függetlenek,* amennyiben a *lyuk megléte vagy hiánya adott tény.* Mindkettő a lyuk közvetlen következménye, azonban egyiknek sincs közvetlen hatása a másikra: a fogfájás a fogidegek állapotától függ, míg a szonda pontosságát a fogorvos szakértelme határozza meg, amelyben nincs szerepe a fogfájásnak.[10] Ezt a tulajdonságot matematikailag a következőképpen írhatjuk le

$$P(fogfájás \wedge beakadás | Lyuk) = P(fogfájás | Lyuk) P(beakadás | Lyuk) \quad (13.13)$$

Ez az egyenlet a *Lyuk* ténye esetén a *fogfájás* és *beakadás* között fennálló **feltételes függetlenség**et (**conditional independence**) fejezi ki. (13.13)-at (13.12)-be behelyettesítve megkapjuk a lyuk valószínűségét:

$$P(Lyuk | fogfájás \wedge beakadás) = \alpha P(fogfájás | Lyuk) P(beakadás) \mid Lyuk) P(Lyuk)$$

Ezzel ugyanarra az információigényre jutottunk, mint a minden egyes tényt külön használó következtetésnél: a keresés változójának **P**(*Lyuk*) a priori valószínűségét, valamint minden egyes okozat saját okának fennállása esetén igaz feltételes valószínűségét kell ismernünk.

Két változó, *X* és *Y* egy adott *Z* harmadik melletti feltételes függetlenségének általános definíciójáta következő egyenlet adja:

$$P(X, Y | Z) = P(X | Z) P(Y | Z)$$

A fogorvostartományban, például, logikusnak tűnik a *Fogfájás* és *Beakadás* között feltételes függetlenséget feltételezni a *Lyuk* ténye esetén:

$$P(Fogfájás, Beakadás | Lyuk) = P(Fogfájás | Lyuk) P(Beakadás | Lyuk) \quad (13.14)$$

Vegyük észre, hogy ez a kijelentés valamelyest erősebb, mint amit a (13.13) egyenlet fejez ki. Ez utóbbi csak a *Fogfájás* és *Beakadás* bizonyos értékeihez rendel függetlenséget. Éppúgy, mint a (13.8) egyenletben az abszolút függetlenségnél, a következő ekvivalens kifejezések is használhatók:

$$P(X | Y, Z) = P(X | Z) \text{ és } P(Y | X, Z) = P(Y | Z)$$

A 13.5. alfejezetben megmutattuk, hogy a teljes függetlenség bizonyíthatósága esetén a teljes együttes valószínűségi eloszlás szétbontható sokkal kisebb részekre. Ugyanez lesz igaz feltételes függetlenség esetén is. Példának okáért, a (13.14) szerinti kijelentés fennállása esetén, a felbontás a következőt jelentheti:

[10] Feltételezve, hogy a beteg és a fogorvos két különböző személy.

Created by XMLmind XSL-FO Converter.

$P(Fogfájás, Beakadás, Lyuk)$

$= P(Fogfájás, Beakadás | Lyuk) P(Lyuk)$ (szorzatszabály)

$= P(Fogfájás | Lyuk) P(Beakadás | Lyuk) P(Lyuk)$ (13.14)-et használva

Fontos

Ezzel a módszerrel az eredeti nagy táblázat felbontható három kisebbre. Az eredeti táblázat hét független számot tartalmaz (2^3–1, mivel a számok összegének 1-et kell adnia). A kisebb táblázatokban öt független szám található (mindegyik feltételes valószínűségi eloszlásra $2 \times (2^1–1)$, valamint $2^1–1$ a *Lyuk* előzetes valószínűségi eloszlására). Ez nem tűnik túl nagy győzelemnek, azonban, ha a *Lyuk* ténye mellett n tünet bizonyul függetlennek, akkor a reprezentáció $O(n)$ nagyságrendben növekszik $O(2^n)$ helyett. Következésképpen *a feltételes függetlenségi kijelentések lehetővé teszik nagy valószínűségi rendszerek kezelhetőségét, sőt, a feltételes függetlenségi kijelentések gyakrabban rendelkezésre is állnak, mint az abszolút függetlenségre vonatkozók.* A *Lyuk* fogalmilag **szétválaszt**ja (**separate**) a *Fogfájás*-t és a *Beakadás*-t, mivel mindkettőnek közvetlen következménye. Nagy valószínűségi tartományok feltételes függetlenségen keresztül lazán kapcsolódó részhalmazokra történő szétbontása a modern MI történetének legnagyobb eredményei közé tartozik.

A fogorvosi eset jó példa az olyan, rendszeresen bekövetkező mintára, ahol egyetlen ok közvetlenül befolyásol számos olyan okozatot, amelyek az ok fennállása esetén feltételesen függetlenek. A teljes valószínűségi eloszlás az alábbiak szerint írható fel:

$$\mathrm{P}(Ok, Okozat_1, \ldots, Okozat_n) = \mathrm{P}(Ok) \prod_i \mathrm{P}(Okozat_i | Ok)$$

Az ilyen valószínűségi eloszlásokat **naiv Bayes**-modellnek hívjuk – „naiv”, mert gyakran használják (egyszerűsítésként) olyan esetekben is, ahol az „okozati” változók valójában *nem* függetlenek az „ok” fennállása esetén. (A naiv Bayes-modellt néha **Bayes-osztályozó**nak (**Bayes classifier**) is hívják némileg meggondolatlanul, ami arra sarkallta az igazi Bayes-következtetést alkalmazókat, hogy a naiv Bayes-modellt **együgyű Bayes**- (**idiot Bayes**) modellnek hívják.) A naiv Bayes-modellek a gyakorlatban akkor is meglepően jól működnek, ha a valószínűségi feltételezés nem igaz. A 20. fejezetben olyan módszereket írunk le, amelyekkel a naiv Bayes-eloszlások megtanulhatók a megfigyelésekből.

7. A wumpus világ újralátogatása

E fejezet gondolatai közül sok eredményesen kombinálható a wumpus világ valószínűségi következtetési problémáinak megoldására. (A wumpus világ teljes leírása a 7. fejezetben található.) A wumpus világban fellépő bizonytalanságot az ágens érzékelésének a világra vonatkozóan a csak helyi információt szolgáltató részlegessége okozza. Például a 13.6. ábra egy olyan szituációt ábrázol, amelyben mindhárom elérhető négyzet – [1, 3], [2, 2] és [3, 1] – tartalmazhat csapdát. Egyszerű logikai következtetéssel nem dönthető el, hogy melyik a legvalószínűbben biztonságos négyzet, így a logikai ágens kénytelen véletlenszerűen választani. Látni fogjuk ugyanakkor, hogy egy valószínűségi ágens sokkal sikeresebb, mint egy logikai ágens.

A célunk az lesz, hogy mindhárom négyzetre kiszámítsuk a csapda valószínűségét. (A példa erejéig elfelejtjük a wumpust és az aranyat.) A wumpus világ idevágó tulajdonságai a következők: (1) egy csapda szellőt okoz a szomszédos négyzetekben, és (2) az [1,1]-en kívül minden négyzet 0,2 valószínűséggel tartalmaz csapdát. Első lépésként a számításhoz szükséges véletlen változókat kell meghatároznunk:

- Az ítéletlogikához hasonlóan, itt is minden négyzethez egy ($C_{i,j}$) logikai változóra van szükségünk, amelynek értéke akkor és csak akkor igaz, ha az [i,j] négyzet tartalmaz csapdát.

- Vannak továbbá $S_{i,j}$ logikai változóink is, amelyek igaz értéke egyértelműen az [i,j] négyzetben tapasztalható szellőre utal; példánkban csak a megfigyelt – esetünkben az [1, 3], [2, 2] és [3, 1] négyzetekre – vonatkozó változókat használjuk.

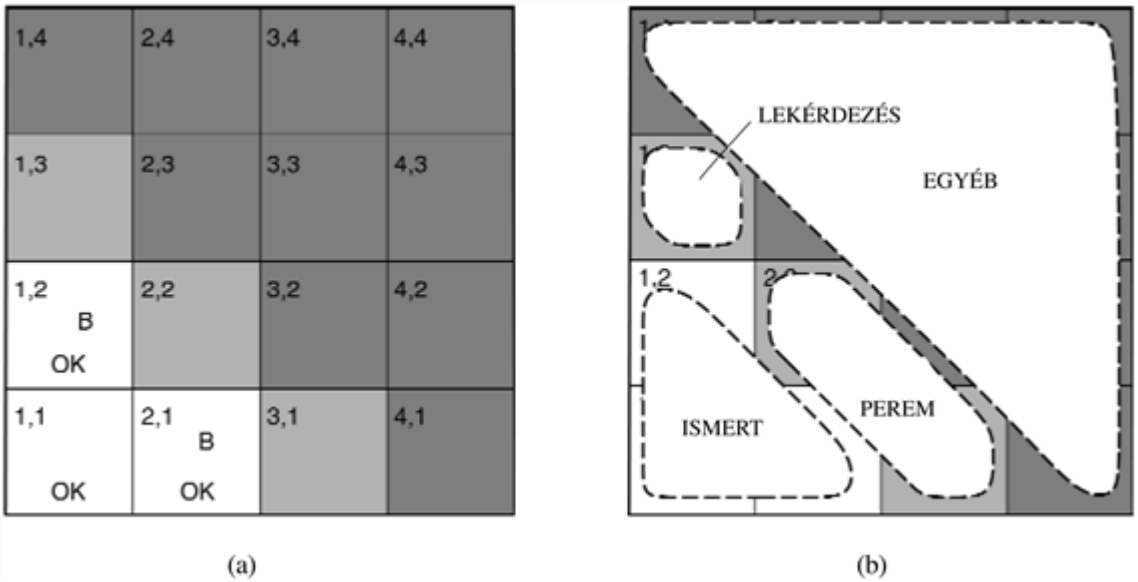
A következő lépés a $\mathbf{P}(C_{1,1}, \ldots, C_{4,4}, S_{1,1}, S_{1,2}, S_{2,1})$ teljes valószínűségi eloszlás meghatározása. A szorzatszabály alkalmazásával

$P(C_{1,1}, \ldots, C_{4,4}, S_{1,1}, S_{1,2}, S_{2,1}) = P(S_{1,1}, S_{1,2}, S_{2,1} | C_{1,1}, \ldots, C_{4,4}) P(C_{1,1}, \ldots, C_{4,4})$

Created by XMLmind XSL-FO Converter.

adódik.

13.6. ábra - (a) Az ágens beragad, miután [1, 2]-ben és [2, 1]-ben is szellőt észlel – nincs feltárható biztonságos hely. (b) A négyzetek szétosztása *Ismert, Perem* és *Egyéb* csoportokba az [1, 3]-ra vonatkozó lekérdezéshez.



A dekompozíció nagyon könnyűvé teszi az együttes valószínűségi értékek meghatározását. Az első rész a szellő feltételes valószínűség konfigurációját adja adott csapda konfigurációnál; ennek értéke 1, ahol a szellő szomszédos egy csapdával, minden más esetben 0. A második tag a csapdakonfiguráció előzetes valószínűsége. A négyzetek a többi négyzettől függetlenül 0,2 valószínűséggel tartalmaznak csapdát; következésképpen

$$\mathrm{P}(C_{1,1},\ldots,C_{4,4}) = \prod_{i,j=1,1}^{4,4} \mathrm{P}(C_{i,j})$$

Egy *n* csapdát tartalmazó konfigurációnál ennek értéke $0{,}2^n \times 0{,}8^{16-n}$.

A 13.6. (a) ábra szerinti helyzetnél a bizonyíték magában foglalja minden meglátogatott négyzetre az ott megfigyelt szellőt (vagy annak hiányát) azzal a ténnyel együtt, hogy a meglátogatott négyzetek nem tartalmaznak csapdát. Ezeket az $s = \neg s_{1,1} \wedge s_{1,2} \wedge s_{2,1}$ és $ismert = \neg c_{1,1} \wedge \neg c_{1,2} \wedge \neg c_{2,1}$ rövidítésekkel fogjuk jelölni. A mi számunkra a $\mathbf{P}(C_{1,3} | \ ismert, s)$ típusú lekérdezések megválaszolása érdekes: adott megfigyelések mellett mennyire valószínű, hogy az [1, 3] csapdát tartalmaz?

A válaszhoz követhetjük a (13.6) egyenlet szerinti szabványos megközelítést, amelyet a `FELSOROL-EGYÜTTES-KÉRDEZÉS` eljárás valósít meg, azaz a teljes együttes valószínűségi eloszlás bejegyzései szerinti összegzést. Jelölje *Ismeretlen* azt az összetett változót, amelyet a nem *Ismert* csoportba tartozó négyzetek és a lekérdezett [1, 3] négyzet $C_{i,j}$ változói alkotnak. Ezzel a (13.6) egyenletből

$$\mathrm{P}(C_{1,3} | ismert, s) = \alpha \sum_{ismeretlen} \mathrm{P}(C_{1,3}, ismeretlen, ismert, s)$$

A teljes együttes valószínűségeket korábban már meghatároztuk, így készen is vagyunk – kivéve, ha a számítási bonyolultság kérdésével is törődünk. 12 ismeretlen négyzetünk van; így az összegzés $2^{12} = 4096$ tagból fog állni. Általánosságban, az összegzés exponenciálisan nő a négyzetek számával.

Az intuíció azt súgja, hogy itt valamit még figyelmen kívül hagyunk. És tényleg, bárki megkérdezheti, hogy a többi négyzet nem lényegtelen-e? A [4, 4] tartalma nincs hatással arra, hogy az [1, 3] tartalmaz-e csapdát! Ez az intuíció valóban helyes. Jelölje *Perem* azokat a – lekérdezés változójától különböző – változókat, amelyek a meglátogatott négyzetekkel szomszédosak, esetünkben a [2, 2] és a [3, 1]. Továbbá *Egyéb* legyen a többi ismeretlen négyzet változója; példánkban – ahogy a 13.6. (b) ábra is mutatja – 10 ilyen négyzet van. A kulcs meglátás az, hogy megfigyelt szellők *feltételesen függetlenek* a többi változótól, ha adottak az *ismert*, a *perem* és a *lekérdezés* változók. A többi, ahogy mondják, már csak egy kis algebra.

Created by XMLmind XSL-FO Converter.

A fenti meglátás használatához a lekérdezést olyan alakra hozzuk, amelyben a szellő az összes többi változó feltétele mellett van megadva, majd az egészet egyszerűsítjük a feltételes függetlenség alapján:

$$\begin{aligned}
&\mathrm{P}(C_{1,3} \mid ismert, s) \\
&= \alpha \sum_{ismeretlen} \mathrm{P}(s \mid C_{1,3}), ismert, ismeretlen,) \mathrm{P}(C_{1,3}, ismert, ismeretlen) \\
&\qquad \text{(szorzószabály)} \\
&= \alpha \sum_{perem} \sum_{egyéb} \mathrm{P}(s \mid ismert, C_{1,3}, perem, egyéb) \mathrm{P}(C_{1,3}, ismert, perem, egyéb) \\
&= \alpha \sum_{perem} \sum_{egyéb} \mathrm{P}(s \mid ismert, C_{1,3}, perem) \mathrm{P}(C_{1,3}, ismert, perem, egyéb)
\end{aligned}$$

ahol az utolsó lépés használja ki a feltételes függetlenséget. E kifejezés első tagja nem függ a többi változótól, így az összegzés belülre vihető:

$$\mathrm{P}(C_{1,3} \mid ismert, s) = \alpha \sum_{perem} \mathrm{P}(s \mid ismert, C_{1,3}, perem) \sum_{egyéb} \mathrm{P}(C_{1,3}, ismert, perem, egyéb)$$

Függetlenség esetén a (13.15) egyenlethez hasonlóan az előző kifejezés tényezőkre bontható, majd a tényezők sorrendje átrendezhető:

$$\begin{aligned}
&\mathrm{P}(C_{1,3} \mid ismert, s) \\
&= \alpha \sum_{perem} \mathrm{P}(s \mid ismert, C_{1,3}, perem) \sum_{egyéb} \mathrm{P}(C_{1,3}) C(ismert) C(perem) C(egyéb) \\
&= \alpha C(ismert) \mathrm{P}(C_{1,3}) \sum_{perem} \mathrm{P}(s \mid ismert, C_{1,3}, perem) C(perem) \sum_{egyéb} C(egyéb) \\
&= \alpha' \mathrm{P}(C_{1,3}) \sum_{perem} \mathrm{P}(s \mid ismert, C_{1,3}\ perem) C(perem)
\end{aligned}$$

ahol az utolsó lépésben $C(ismert)$-et belefoglaltuk a normalizáló konstansba, és kihasználtuk, hogy $\sum_{egyéb} C(egyéb)$ értéke 1.

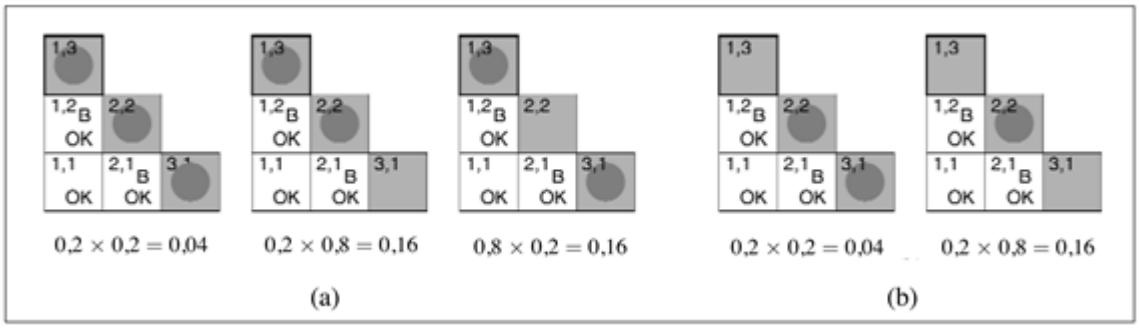
Most már csak négy tag van a $C_{2,2}$ és $C_{3,1}$ peremváltozók feletti összegzésben. A függetlenség és feltételes függetlenség miatt egyáltalán nem szükséges figyelembe vennünk a többi négyzetet. Vegyük észre, hogy a **P**(*s*| *ismert*, $C_{1,3}$, *perem*) kifejezés értéke 1, ha a határ megegyezik a szellő érzékelésekkel és minden más esetben 0. Azaz, $C_{1,3}$ minden értékére azon határváltozók *logikai modelljei* fölött végezzük az összegzést, amelyek az megfelelnek az ismert tényeknek. (Érdemes ezt összehasonlítani a 7.5. ábra modelljei szerinti számítással.) A modelleket és a hozzájuk kapcsolódó *C*(*perem*) a priori valószínűségeket a 13.7. ábra mutatja. Ezzel

$$P(C_{1,3} \mid ismert,\ s) = \alpha' \langle 0{,}2(0{,}04 + 0{,}16 + 0{,}16),\ 0{,}8(0{,}04 + 0{,}16) \rangle \approx \langle 0{,}31,\ 0,\ 69 \rangle$$

Azaz, az [1, 3] (és a szimmetria miatt a [3, 1]) nagyjából 31% valószínűséggel tartalmaz csapdát. Hasonló számításokból (amelyet az olvasó könnyen elvégezhet) a [2, 2]-re 86%-os valószínűséggel adódik csapda. A wumpus ágensnek határozottan el kell kerülnie a [2, 2]-t!

13.7. ábra - $C_{2,2}$ és $C_{3,1}$ peremváltozók, az egyes modellek *C*(*perem*) értékét mutató konzisztens modelljei: (a) három, két vagy három csapdát jelző modell $C_{1,3}$ = *igaz* mellett, és (b) két, egy vagy két csapdát jelző modell $C_{1,3}$ = *hamis* mellett.

Created by XMLmind XSL-FO Converter.



Ebben az alfejezetben azt mutattuk meg, hogy még a látszólag bonyolult problémák is pontosan megfogalmazhatók valószínűség-elméleti alapokon, és meg is oldhatók egyszerű algoritmusok segítségével. Hogy *hatékony* megoldásokat kapjunk, a szükséges összegzéseket a függetlenségekre és feltételes függetlenségekre alapozva egyszerűsítjük. Ezek az összefüggések gyakran egybeesnek a probléma részproblémákra való felbontására vonatkozó természetes felfogásunkkal. A következő fejezetben ilyen összefüggések formális megjelenítésére alkalmazunk módszereket, valamint olyan algoritmusokat fejlesztünk ki, amelyek hatékonyan használhatók e reprezentációk alapján elvégzett valószínűségi következtetések végrehajtására.

8. Összefoglalás

Ez a fejezet arra mutat rá, hogy a valószínűség megfelelő módja a bizonytalanságra vonatkozó következtetéseknek.

- Bizonytalanság következik mind a lustaságból, mind pedig a tudatlanságból. Összetett, dinamikus vagy hozzáférhetetlen világok esetén mindez elkerülhetetlen.
- A bizonytalanság azt jelenti, hogy a továbbiakban sok olyan egyszerűsítés, amely deduktív következtetés során megengedhető, hamis eredményre vezet.
- A bizonytalanság az ágens képtelenségét fejezi ki arra, hogy egyértelmű döntést hozzon egy állítás igazságát illetően. A valószínűségek az ágens meggyőződésének fokát fejezik ki.
- Az alapvető valószínűségi állítások magukban foglalják az egyszerű és az összetett állításokra vonatkozó **a priori** és **feltételes valószínűség**eket (**prior** and **conditional probabilities**).
- A **teljes együttes valószínűség-eloszlás** (**full joint probability distribution**) valószínűséget rendel a véletlen változó minden teljes körű érték-hozzárendeléséhez. A teljes együttes valószínűség-eloszlás általában sokkal nagyobb méretű annál, semhogy létrehozzuk vagy használjuk.
- A valószínűségi axiómák kényszereket jelentenek az állításokhoz ésszerűen hozzárendelt valószínűségek értékére vonatkozóan. Az axiómákat megsértő ágens bizonyos körülmények között ésszerűtlenül fog viselkedni.
- Ha rendelkezésre áll a teljes valószínűségi eloszlás, akkor a lekérdezések megválaszolását egyszerűen a lekérdezések állításához tartozó elemi események bejegyzéseinek összeadásával elvégezhetjük.
- A véletlen változók részhalmazai között fennálló **abszolút függetlenség** (**absolute independence**) maga után vonja a teljes együttes valószínűségi eloszlás kisebb együttes valószínűségi eloszlásokból álló szorzattényezőkre bontásának lehetőségét. Ez nagymértékben csökkentheti a bonyolultságot, azonban a gyakorlatban ritkán fordul elő.
- A **Bayes-tétel** (**Bayes' rule**) lehetővé teszi, hogy ismert feltételes valószínűségekből számítsunk ki ismeretleneket, általában az okozat irányába következtetve. Nagy mennyiségű tény rendelkezésre állása esetén a Bayes-tétel alkalmazása ugyanolyan számításigény-robbanáshoz vezet, mint a teljes együttes valószínűségi eloszlás.
- A tartományon belüli közvetlen oksági kapcsolatok okozta **feltételes függetlenség** (**conditional independence**) lehetővé teszi, hogy a teljes valószínűségi eloszlást kisebb, feltételes valószínűségi eloszlások

Created by XMLmind XSL-FO Converter.

szorzataként írhassuk fel. A **naiv Bayes-** (**naive Bayes**) modell adott okváltozó mellett feltételezi az okozati változók feltételes függetlenségét, a modell mérete pedig az okozati változók számával lineárisan nő.

- A wumpus világ ágense ki tudja számítani a világ meg nem figyelt eseményeinek valószínűségét is, és azokat használva jobb következtetésekre tud jutni, mint egy egyszerű logikai ágens.

8.1. Irodalmi és történeti megjegyzések

Bár a szerencsejátékok már i. e. 300 körül is ismertek voltak, az esélyek és a valószínűségek matematikai analízise jóval későbbre tehető. Mahaviracarya Indiában nagyjából az i. e. 9. században kezdte meg a téma vizsgálatát. Európában az első kísérletek csak az olasz reneszánsz idejére, kb. 1500-ra tehetők. Az első jelentős következetes elemzés Girolamo Cardano nevéhez fűződik (1565), amely azonban 1663-ig nem került nyilvánosságra. Ekkorra Blaise Pascalnak a valószínűségek szisztematikus kiszámítására vonatkozó felfedezése (Pierre Fermat-val levelezve 1654-ben) megalapozta a valószínűség-számítást, mint a matematika széles körben és eredményesen vizsgált ágát. Az első valószínűségekkel kapcsolatos tankönyv a *De Ratiociniis in Ludo Aleae* volt (Huygens, 1657). Pascal bevezette a feltételes valószínűség fogalmát is, amelyről szintén szó van Huygens könyvében. Thomas Bayes anglikán lelkész (1702–1761) nevéhez fűződik a később róla elnevezett szabály, amely feltételes valószínűségek számítására alkalmas. Eredménye csak halála után jelent meg nyomtatásban (Bayes, 1763). Kolmogorov volt az első (Kolmogorov, 1950; első megjelenés németül 1933), aki a valószínűség-számítást szigorúan axiomatikus keretek között tárgyalta. Rényi (Rényi, 1970) olyan axiómarendszert vezetett be, amely nem a feltétel nélküli, hanem a feltételes valószínűség fogalmára épül.

Pascal többféleképpen használta a valószínűség fogalmát, így igényelte mind az objektivista értelmezést – amely szerint a valószínűség a világhoz tartozó, szimmetrián vagy relatív gyakoriságon alapuló sajátság –, mind pedig a valószínűség meggyőződési mértéken alapuló szubjektív megközelítését: előbbit a szerencsejátékokra vonatkozó valószínűség elemzéseiben, míg az utóbbit a híres pascali fogadás érvelésében Isten létezése vagy nemléte kapcsán. Mindemellett, Pascal nem látta világosan a két megközelítés közötti különbséget. Ezt jól érthetően először James Bernoulli (1654–1705) fogalmazta meg.

Leibniz nevéhez fűződik a valószínűség klasszikus fogalmának – mint a számba vett azonos esélyű esetek arányának – leírása. Ugyanezt a fogalmat használta Bernoulli is, bár igazából Laplace (1749–1827) vezette be a gyakorlatba. A fogalom azonban a frekvencionista, illetve a szubjektív értelmezési mód között nem tud különbséget tenni. Az esetek tekinthetők azonos esélyűeknek akár a természetes fizikai szimmetriák miatt, akár pedig egyszerűen azért, mert nincs olyan információnk, amely arra mutatna, hogy az egyik bekövetkezésének nagyobb az esélye, mint a másikénak. Ez utóbbit, amikor egyéni megfontolások alapján igazoljuk az egyenlő valószínűségek hozzárendelését, a *pártatlanság elvének* (*principle of indifference*) hívjuk (Keynes, 1921).

A valószínűség objektivista és szubjektivista értelmezése közötti vita a 20. században kiélesedett. A relatív gyakoriság szószólói Kolmogorov (Kolmogorov, 1963), R. A. Fisher (Fisher, 1922) és Richard von Mises (von Mises, 1928) voltak. Karl Popper (Popper, 1959; első megjelenés németül 1934) hajlam (propensity) értelmezése a mögöttes fizikai szimmetriát látja a relatív gyakoriságban. Frank Ramsey, Bruno de Finetti, R. T. Cox, Leonard Savage és Richard Jeffrey úgy értelmezték a valószínűséget, mint meghatározott egyének meggyőződésének mértékét (Ramsey, 1931; de Finetti, 1937; Cox, 1946; Savage, 1954; Jeffrey, 1983). A hiedelem mértékére vonatkozó elemzésük szorosan kötődött a hasznossághoz és a viselkedéshez, különösen ahhoz a szándékhoz, hogy fogadásokat kössünk. Rudolf Carnap, Leibnizet és Laplace-t követve, a szubjektív valószínűség egy másfajta értelmezését javasolta: nem mint az adott egyén meggyőződését, hanem mint azt a hiedelemfokot, amelyet egy eszményi személynek *kell* hinnie egy adott *a* állításról **e** tény fennállása esetén. Carnap megpróbált Leibniz-nél és Laplace-nál tovább menni, és ezt a **megerősítés**i (**confirmation**) mérték fogalmat, mint *a* és **e** között fennálló logikai kapcsolatot, matematikailag is pontosan leírni. A kapcsolat tanulmányozása révén akarta a megszokott deduktív logika (Carnap, 1948; 1950) mintájára, a matematika egy **induktív logiká**nak (**inductive logic**) nevezett diszciplináját megalapítani. Carnap képtelen volt az induktív logikát lényegesen kiterjeszteni az ítéletkalkuluson túlra, és Putnam mutatta meg, hogy alapvető nehézségek hiúsítják meg az aritmetikát is kifejezni képes nyelvek irányába történő szabatos kiterjesztést (Putnam, 1963).

A vonatkoztatási osztály kérdése szorosan kötődik az induktív logika megtalálására tett törekvésekhez. Az elégséges méretű legjobban specifikus vonatkoztatási osztály kiválasztására való törekvést Reichenbach javasolta (Reichenbach, 1949). Erőfeszítések történtek kifinomultabb irányelvek kialakítására, amelyekkel el lehet kerülni a Reichenbach-szabályból következő, Henry Kyburg által észlelt ellentmondásokat (Kyburg, 1977; 1983), azonban ezek a törekvések leginkább *ad hoc* jellegűek maradtak. Egy újabb, Bacchus, Grove, Halpern és Koller nevéhez fűződő munka (Bacchus és társai, 1992) Carnap módszereit kiterjeszti az elsőrendű elméletekre, kiküszöbölve ezáltal sok, a vonatkoztatási osztályokkal kapcsolatos nehézséget.

Created by XMLmind XSL-FO Converter.

A bayesi valószínűségi következtető rendszereket a hatvanas évek óta használják az MI-ben, különösen az orvosi diagnosztikában. Nemcsak azért, hogy meglevő tényekből következtetéseket vonjanak le, hanem azért is, hogy az információérték-elmélet (lásd 16.6. alfejezet) segítségével további kérdéseket és vizsgálatokat válasszanak, amikor a meglevő tények nem meggyőzők (Gorry, 1968; Gorry és társai, 1973). Az egyik rendszer felül is múlta az emberi szakértőket a heveny hasi megbetegedések diagnózisában (de Dombal és társai, 1974). Mindemellett azonban ezekkel a korai bayesi rendszerekkel igen sok gond is volt. Lévén, semmilyen elméleti modellel nem rendelkeztek azokról a körülményekről, amelyeket diagnosztizáltak, sebezhetők voltak mindazon esetekben, amikor olyan helyzetekre jellemző adatok tűntek fel, amelyeket csak kisszámú mintahalmaz képviselt (de Dombal és társai, 1981). Még ennél is alapvetőbb, hogy mivel nem volt tömör leíró módszerük (mint amilyet a 14. fejezetben ismertetünk) a feltételes függetlenség megjelenítésére és használatára, erősen függtek a hatalmas mennyiségű valószínűségi adat gyűjtésétől, tárolásától és feldolgozásától. E nehézségek miatt 1970-től a nyolcvanas évek közepéig a bizonytalanságkezelés valószínűségi módszerei kegyvesztettek lettek az MI-ben. A nyolcvanas évek vége óta bekövetkezett fejlődést a következő fejezet írja le.

Az együttes eloszlások naiv bayesi megközelítését az alakfelismeréssel foglalkozó irodalom alaposan tanulmányozta az 1950-es évektől kezdve (Duda és Hart, 1973). A megközelítést Maron munkájának megjelenése (Maron, 1961) óta gyakran használták – sokszor akaratlanul – az adatbányászat területén is. A módszer valószínűségi megalapozását, amelyet részletesen a 13.18. feladat ír le, Robertson és Sparck Jones tisztázta (Robertson és Jones, 1976). A naiv Bayes-következtetésnek a még olyan tartományokban is jelentkező meglepő sikerére, ahol a függetlenségi feltételezés egyértelműen sérül, Domingos és Pazzani ad magyarázatot (Domingos és Pazzani, 1977).

A valószínűség-számítás témakörében nagyon sok kiváló tankönyv létezik, beleértve a Chung és Ross által írtakat (Chung, 1979; Ross, 1988). Morris DeGroot két könyve közül az egyik (DeGroot, 1989) a bayesi álláspont felől közelítve ad közös bevezetést a valószínűség-számításba és a statisztikába, a másik pedig mélyebb ismereteket tartalmaz (DeGroot, 1970). Richard Hamming tankönyve matematikailag megalapozott bevezetést nyújt a valószínűség-számítás elméletébe a fizikai szimmetrián alapuló hajlammegközelítés felől (Hamming, 1991). Hacking és Hald a valószínűség fogalmának korai történetét tárják elénk (Hacking, 1975; Hald, 1990), Bernstein pedig szórakoztató népszerűsítő áttekintést ad a kockázatelemzés történetéről (Bernstein, 1996).

8.2. Feladatok

13.1.

Mutassa meg az alapelvek segítségével, hogy

$P(a \mid b \wedge a) = 1$

13.2.

A valószínűség-számítás axiómáit használva bizonyítsa be, hogy diszkrét véletlen változók tetszőleges valószínűségi eloszlása összegének 1-et kell adnia.

13.3.

Ésszerű lehet-e egy ágens számára a következő három meggyőződésben hinni: $P(A) = 0{,}4$, $P(B) = 0{,}3$ és $P(A \vee B) = 0{,}5$? Ha igen, milyen valószínűségsávot tart ésszerűnek $A \wedge B$-re vonatkozóan? Készítsen a 13.2. ábrához hasonló táblázatot, és mutassa meg, hogyan támasztja ez alá az Ön érvelését. Ezután készítsen egy másik változatot, amelynél $P(A \vee B) = 0{,}7$. Magyarázza el, miért racionális egy ekkora valószínűség hozzárendelése, mindamellett hogy a táblázat mutat egy olyan esetet, ami veszteség, és három olyat, ami ki van egyenlítve. (*Segítség*: milyen valószínűségi hozzárendelés mellett kötelezte el magát az 1. ágens a négy esetben, de különösen a veszteség esetén?)

13.4.

Ez a kérdés az 2.1. szakasz - Állítások részben tárgyaltaknak megfelelően, az elemi események tulajdonságaival foglalkozik.

a. Bizonyítsa be, hogy az összes lehetséges elemi esemény egyesítése logikailag megegyezik az *igaz* állítással. (*Segítség*: bizonyítékként alkalmazzon indukciót a véletlen változók számára vonatkozóan.)

Created by XMLmind XSL-FO Converter.

b. Bizonyítsa be, hogy bármely állítás logikailag ekvivalens azon elemi események diszjunkciójával, amelyek maguk után vonják annak igazságát.

13.5.

Képzeljük el a szokványos 52 lapos kártyacsomagból osztott ötlapos póker játszmák tartományát, annak feltevésével, hogy az osztó becsületes.

a. Hány elemi eseményt tartalmaz az együttes valószínűség-eloszlás (azaz hányféle 5 lapos leosztás létezik egy kézben)?

b. Mekkora a valószínűsége az egyes elemi eseményeknek?

c. Mekkora a valószínűsége a felül ászos színsornak (royal flush, egyszínű ász, király, dáma, bubi és tízes)? És négy egyforma lapnak?

13.6.

Számítsa ki az alábbiakat, ha adott a 13.3. ábra szerinti teljes együttes valószínűségi eloszlás.

a. $P(fogfájás)$

b. $\mathbf{P}(Lyuk)$

c. $\mathbf{P}(Fogfájás \mid lyuk)$

d. $\mathbf{P}(Lyuk \mid fogfájás \vee beakadás)$

13.7.

Mutassa meg, hogy a függetlenség (13.8) egyenlet szerinti három felírási módja ekvivalens egymással.

13.8.

Az évenként esedékes orvosi ellenőrzése után a doktor rossz és jó híreket mond. A rossz hír az, hogy komoly betegséget mutattak ki önnél, és a teszt 99%-ban megbízható (azaz, annak a valószínűsége, hogy a teszt kimutatja a meglevő betegséget, 0,99, és ugyanígy, annak a valószínűsége, hogy a teszt negatív lesz, amennyiben nem áll fenn a betegség, szintén 0,99). A jó hír az, hogy ez a betegség nagyon ritka, amelyet az Ön korában minden 10 000 emberből csak egy kap meg. Miért jó hír az, hogy a betegség ritka? Mekkora az esélye annak, hogy Ön tényleg beteg?

13.9.

Sokszor célszerűbb adott állítások hatását bizonyos rögzítettnek feltételezett általános háttértények feltételezésével végiggondolni, mint az információ teljes hiányában. Az alábbi kérdések a szorzatszabály és a Bayes-tétel általános alakjainak bizonyítását célozzák, valamely **e** háttértény feltételezése mellett.

a. Bizonyítsa be az általános szorzatszabály valamely feltétel fennállása esetén felírható alakját:

$$P(X, Y \mid e) = P(X \mid Y, e)P(Y \mid e)$$

b. Bizonyítsa be a Bayes-tétel (13.10) egyenlet szerinti, valamilyen feltétel fennállása esetére felírt változatát.

13.10.

Mutassa meg, hogy a

$$P(A,B \mid C) = P(A \mid C)\ P(B \mid C)$$

állítás mindkét alábbi állítással egyenértékű

$$P(A \mid B,C) = P(A \mid C) \text{ és } P(B \mid A,C) = P(B \mid C)$$

13.11.

Created by XMLmind XSL-FO Converter.

Tegyük fel, hogy kapott egy n darab egyforma pénzérmét tartalmazó zsákot. Azt is tudjuk, a pénzek közül n–1 szabályos, azaz egyik oldalán fej, a másikon írás található, ugyanakkor van egy darab hamis, amelynek mindkét oldala fej.

a. Tegyük fel, hogy benyúl a zsákba, majd véletlenszerűen kiemel egy pénzdarabot. Ezt feldobva fejet kap eredményül. Mekkora a (feltételes) valószínűsége annak, hogy a kivett pénzdarab a hamis?

b. Tegyük fel, hogy k-szor folytatva a pénz feldobását, ezután is minden alkalommal fejet kap. Most mennyi lesz a feltételes valószínűsége annak, hogy a kivett pénzdarab a hamis?

c. Tegyük fel, hogy a k-szori feldobással akarta eldönteni, hogy a hamis vagy egy jó pénzérmét emelt ki a zsákból. A döntési eljárás HAMIS-at ad, ha mind a k feldobás alkalmával a fej van fölül, egyébként pedig JÓ-t. Mekkora annak a (feltétel nélküli) valószínűsége, hogy az eljárás tévedni fog?

13.12.

Ebben a feladatban befejezheti az agyhártyagyulladásra vonatkozó példa normalizálását. Először is vegyen fel egy megfelelő értéket $P(S|\ \neg M)$-re, és ennek segítségével számítsa ki $P(M|\ S)$ és $P(\neg M|\ S)$ normalizálatlan értékeit (azaz ne vegye figyelembe $P(S)$-t a Bayes-tétel kifejezésében), majd normalizálja az értékeket úgy, hogy összegük 1 legyen.

13.13.

Ez a feladat azt vizsgálja, hogy a feltételes függetlenség fennállása hogyan befolyásolja a valószínűségi becslésekhez szükséges információ mennyiségét.

a. Tegyük fel, hogy $P(h|\ e_1,e_2)$-t szeretnénk kiszámítani úgy, hogy nincs feltételes függetlenségre vonatkozó információnk. Az alábbiak közül melyik az az adathalmaz, amelyikre a számításokhoz szükségünk van?

i. $P(E_1,\ E_2),\ P(H),\ P(E_1|\ H),\ P(E_2|\ H)$

ii. $P(E_1,\ E_2),\ P(H),\ P(E_1,\ E_2|\ H)$

iii. $P(H),\ P(E_1|\ H),\ P(E_2|\ H)$

d. Tegyük fel, hogy tudjuk, hogy $\mathbf{P}(E_1|\ H, E_2) = \mathbf{P}(E_1|\ H)$, H, E_1, E_2 minden értékére. Hogyan változik az előbbi kérdésre adott válasz?

13.14.

X, Y és Z legyenek logikai véletlen változók. A $\mathbf{P}(X, Y, Z)$ együttes eloszlás nyolc lehetséges esetét jelöljük a, …, h-val. Fejezze ki az „X és Y adott Z mellett feltételesen függetlenek” állítást az a, …, h-t összekapcsoló egyenletek segítségével. Hány lesz ezek közül *redundanciamentes*?

13.15.

(A (Pearl, 1988)-ból átvéve.) Képzeljük el, hogy Ön egy éjszakai taxis cserbenhagyásos baleset tanúja Athénban. Athénban minden taxi kék vagy zöld. Ön azt vallja eskü alatt, hogy a taxi kék volt. Egy széles körű vizsgálat azt mutatja, hogy ilyen gyenge fényben a kék és zöld szín közötti tévesztés valószínűsége 75%. Kiszámítható-e, hogy milyen színű taxi volt a legvalószínűbb? (*Segítség*: gondosan különböztessük meg azt az állítást, hogy a taxi *kék*, és azt, hogy a taxi *kéknek tűnik*.)

Mi van akkor, ha minden 10 athéni taxiból 9 zöld?

13.16.

(A (Pearl, 1988)-ból átvéve.) Három fegyenc, A, B és C celláikba vannak zárva. Közismert, hogy egyiküket másnap ki fogják végezni, míg a másik kettő kegyelmet kap. Azt, hogy kit végeznek ki, csak a börtönigazgató tudja. A fegyenc egy kéréssel fordul a börtönőrhöz: „Legyen szíves, kérdezze meg az igazgatótól, hogy kit fognak holnap kivégezni, és a barátaim közül az egyiknek, B-nek vagy C-nek árulja el, hogy reggel kegyelmet fog kapni.” Az őr beleegyezik, majd később azzal jön vissza, hogy B-nek szólt a kegyelemről.

Created by XMLmind XSL-FO Converter.

Mekkorák *A* esélyei a kivégzésre a történtek után? (Ne csak erőteljes kézlegyintéssel, hanem *matematikailag megalapozottan* válaszoljon.)

13.17.

Írja meg azt az általános, naiv Bayes-eloszlást használó algoritmust, amely alkalmas **P**(*Ok*| **e**) alakú keresések megválaszolására. Fel kell tételeznie azt is, hogy az **e** bizonyíték az érintett változók *tetszőleges részhalmazához* rendel értékeket.

13.18.

A szövegosztályozás olyan feladat, amely egy adott dokumentumot – a benne levő szöveg alapján – előre rögzített kategóriák valamelyikébe sorol be. Erre gyakran használnak naiv Bayes-modellt. Ezekben a modellekben a lekérdezés változója a dokumentumkategória és az „okozati” változók a nyelv egyes szavainak megléte vagy hiánya. Feltételezzük azt is, hogy a szavak egymástól függetlenül, a dokumentumkategóriára jellemző sűrűséggel jelennek meg a szövegekben.

a. Magyarázza el pontosan, hogyan lehet egy ilyen modellt létrehozni, ha „tanító mintaként” adott egy, már osztályozott dokumentumhalmaz.

b. Magyarázza el pontosan, hogyan történik egy új dokumentum besorolása.

c. Elfogadható-e a függetlenség feltételezése? Részletezze.

13.19.

A wumpus világ vizsgálatánál abból a tényből indultunk ki, hogy a négyzetek a többi négyzet tartalmától függetlenül 0,2 valószínűséggel tartalmaznak csapdát. Ehelyett most tegyük fel, hogy az [1, 1]-en kívüli N négyzetben pontosan $N/5$ csapda van egyenletes eloszlásban véletlenszerűen elhelyezve. Függetlenek-e továbbra is a $C_{i,j}$ és a $C_{k,l}$ változók? Hogyan fog most kinézni a **P**($C_{1,1}$, …, $C_{4,4}$) együttes eloszlás? Számolja újra az [1, 3] és a [2, 2] négyzetekben található csapda valószínűségét.

Created by XMLmind XSL-FO Converter.

14. fejezet - Valószínűségi következtetés

Ebben a fejezetben ismertetjük, hogyan építhetők olyan következtető rendszerek, amelyek hálós modelleket használnak a bizonytalanságnak a valószínűség-számítás törvényeinek megfelelő kezelésére.

A 13. fejezetben bemutattuk a valószínűség-számítás szintaxisát és szemantikáját. Külön hangsúlyoztuk a függetlenségi és a feltételes függetlenségi kapcsolatok fontosságát a világ valószínűségi reprezentációjának az egyszerűsítésében. Ez a fejezet egy szisztematikus módszert vezet be az ilyen relációk explicit reprezentálására, a **Bayes-hálók** formájában. Ezeknek a hálózatoknak definiáljuk a szintaxisát és szemantikáját, és megmutatjuk, hogyan használhatók fel bizonytalan tudás megragadására természetes és hatékony módon. Ezt követően ismertetjük, hogyan végezhető el a valószínűségi következtetés, ami bár legrosszabb esetben exponenciális komplexitású, a gyakorlatban sok esetben hatékonyan alkalmazható. Leírunk továbbá különböző közelítő következtető algoritmusokat, amelyek gyakran akkor is alkalmazhatók, amikor egzakt következtetés nem lehetséges. Megvizsgálunk olyan módszereket is, hogy hogyan alkalmazható a valószínűség-számítás egy objektumokat és relációkat is tartalmazó világra – azaz az *ítéletlogikával* szemben az *elsőrendű* reprezentációkra. Végül áttekintjük a bizonytalan következtetés egyéb alternatív megközelítéseit.

1. A tudás reprezentálása bizonytalanság esetén

A 13. fejezetben láttuk, hogy az együttes valószínűség-eloszlás függvény alapján minden kérdés megválaszolható a modellezett tárgytartománnyal kapcsolatban, de a változók számának növekedésével ez általában kivitelezhetetlenné válik. Továbbá az elemi események valószínűségének megadása is igen mesterkélt és nehéz is lehet, hacsak nem áll rendelkezésre nagy mennyiségű adat a valószínűségek statisztikai becsléséhez.

A változók közötti függetlenségi és feltételes függetlenségi relációkkal kapcsolatban azt is láttuk, hogy ezek nagyban csökkenthetik az együttes valószínűség-eloszlás függvény megadásához szükséges valószínűségek számát. Ez a fejezet bevezet egy **Bayes-hálónak** (**Bayesian network**[1]) nevezett adatstruktúrát a változók közötti függőség leírásához, és *bármely* együttes valószínűség-eloszlás függvény tömör megadásához.

A Bayes-háló egy irányított gráf, amelyben minden *csomóponthoz* számszerű valószínűségi információk vannak csatolva. A teljes megadás a következő:

1. A háló csomópontjait valószínűségi változók egy halmaza alkotja. A változók lehetnek diszkrétek vagy folytonosak.

2. Irányított élek (nyilak) egy halmaza összeköt bizonyos csomópontpárokat. Ha létezik nyíl az X csomóponttól az Y csomópontig, azt mondjuk, hogy az X a *szülője* az Y-nak.

3. Minden X_i csomóponthoz tartozik egy $P(X_i | \mathit{Szülők}(X_i))$ feltételes valószínűség-eloszlás, ami számszerűen megadja a szülők hatását a csomóponti változóra.

4. A gráf nem tartalmaz irányított kört (azaz irányított, körmentes gráf – Directed, Acyclic Graph, DAG).

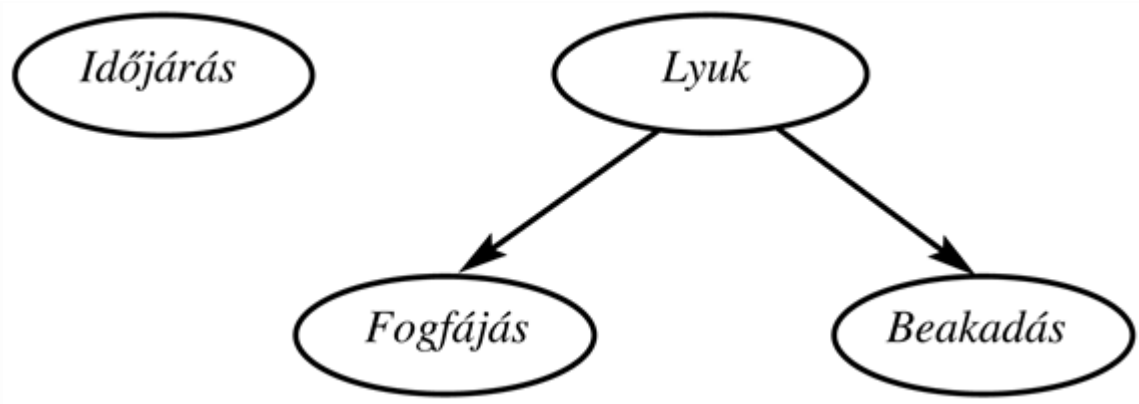
A háló topológiája – a csomópontok és élek halmaza – megadja a tárgyterületen fennálló feltételes függetlenségi kapcsolatokat, hogy mily módon, azt hamarosan pontosan kifejtjük. Egy helyesen létrehozott hálóban az X csomópontot az Y csomóponttal összekötő nyíl *intuitív* jelentése rendszerint az, hogy az *X-nek közvetlen befolyása van* az Y-ra. A tárgyterület szakértője számára általában könnyű eldönteni, hogy milyen *közvetlen befolyások* teljesülnek egy adott területen – valójában sokkal könnyebb, mint a megfelelő valószínűségeket megadni. Ha pedig a Bayes-háló topológiája kész, már csak az egyes változókhoz tartozó feltételes valószínűség-eloszlásokat kell meghatározni a szülőkkel mint feltételekkel. Látni fogjuk, hogy a topológia és a feltételes eloszlások együttese elegendő, hogy megadja (implicit módon) az összes változó feletti együttes valószínűség-eloszlás függvényt.

[1] Az angol nyelvű szakirodalomban a **Bayes-háló** a legelterjedtebb megnevezés, de számos más is létezik: **bizonyosságháló** (**belief network**), **valószínűségi háló** (**probabilistic network**), **okozati háló** (**causal network**), **tudásháló** (**knowledge map**). A statisztikában a **gráfos modell** (**graphical model**) bővebb modellosztályt jelent, ami a Bayes-hálót is magában foglalja. A Bayes-hálók egy kiterjesztésének neve a **döntési háló** (**decision network**) vagy **hatásdiagram** (**influence diagram**) – lásd 16. fejezet.

Created by XMLmind XSL-FO Converter.

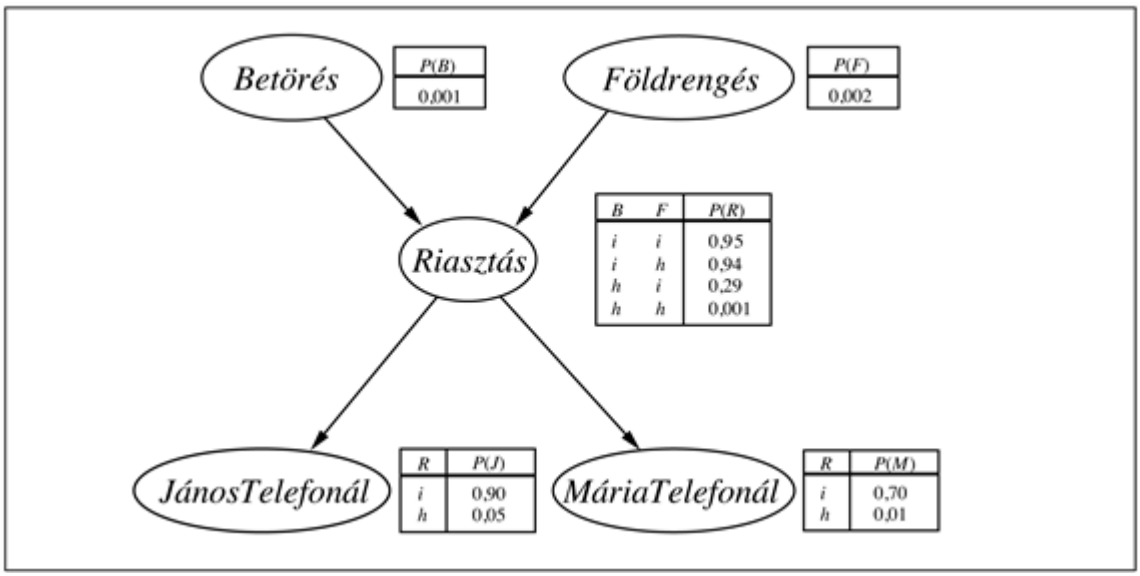
Idézzük fel a 13. fejezetben leírt egyszerű világot, amely a *Fogfájás*, a *Lyuk*, a *Beakadás* és az *Időjárás* változót tartalmazza. Úgy érveltünk, hogy az *Időjárás* független az összes többi változótól; továbbá, hogy a *Fogfájás* és a *Beakadás* feltételesen függetlenek a *Lyuk* ismeretében. Ezeket a viszonyokat a 14.1. ábrán bemutatott Bayes-háló reprezentálja. Formálisan, a *Fogfájás* és a *Beakadás* feltételes függetlenségét a *Lyuk* ismeretében a *Fogfájás* és a *Beakadás* közötti nyíl *hiánya* jelzi. Szemléletesen, a háló azt a tényt fejezi ki, hogy a *Lyuk* közvetlen oka a *Fogfájás*-nak és a *Beakadás*-nak, további közvetlen okozati kapcsolat azonban nem létezik a *Fogfájás* és a *Beakadás* között.

14.1. ábra - Egy egyszerű Bayes-háló, amelyben az *Időjárás* független a többi három változótól, a *Fogfájás* és a *Beakadás* pedig feltételesen függetlenek a *Lyuk* ismeretében



Most vizsgáljuk meg a közvetkező példát, ami csak egy árnyalatnyival összetettebb. Otthonunkban egy új betörésjelzőt szereltek fel. Ez megbízhatóan észleli a betöréseket, de időnként kisebb földrengések esetén is jelez. (Ez a példa Judea Pearltől származik, aki Los Angeles-i lakos; innen ered a földrengések iránti különös érdeklődése.)[2] Két szomszédunk is van, János és Mária, akik megígérték, hogy felhívnak a munkahelyünkön, ha meghallják a riasztónkat. János mindig felhív minket, ha meghallja a riasztást, de néha összekeveri a telefoncsörgést a riasztó csengésével, és ekkor is telefonál. Mária viszont, mivel szereti hangosan hallgatni a zenét, néha meg sem hallja a riasztót. Mi tehát a hívások bekövetkezte vagy hiánya alapján szeretnénk megbecsülni a betörés valószínűségét. Ezt az egyszerű problémát a 14.2. ábrán látható Bayes-háló írja le.

14.2. ábra - Egy tipikus Bayes-háló, amely a topológiát és a feltételes valószínűségi táblákat (FVT) is mutatja. Az FVT-kben *B*, *F*, *R*, *J* és *M* szerepel *Betörés*, *Földrengés*, *Riasztás*, *JánosTelefonál* és *MáriaTelefonál* helyett.



[2] Judea Pearl meghatározó szerepet játszott a Bayes-hálók egzakt matematikai hátterének és gyakorlati alkalmazhatóságának a vizsgálatában. (*A ford.*)

Created by XMLmind XSL-FO Converter.

Egyelőre hagyjuk figyelmen kívül az ábrán szereplő feltételes eloszlásokat, és összpontosítsunk a háló topológiájára. A betörés háló esetén a topológia azt mutatja, hogy a betörés és a földrengés közvetlenül befolyásolja a riasztó megszólalásának valószínűségét, ellenben János vagy Mária hívásának bekövetkezése csak magán a riasztón múlik. A háló így tartalmazza azon feltevéseinket, hogy ők közvetlenül nem vesznek észre betöréseket, nem vesznek észre kisebb földrengéseket, és nem egyeztetnek hívás előtt.

Megfigyelhető, hogy a hálóban nincsenek olyan csomópontok, amelyek azt írnák le, hogy Mária éppen hangos zenét hallgat, vagy hogy a telefon cseng, és megzavarja Jánost. Ezeket a tényezőket a *Riasztás* csomóponttól a *JánosTelefonál* és *MáriaTelefonál* csomópontig tartó élekhez rendelt bizonytalanság foglalja magában. Ebben mind a lustaság, mind pedig a tudatlanság tetten érhető: rengeteg munka volna olyan tényezők meghatározása, amelyek valószínűbbé vagy kevésbé valószínűvé válnának adott esetekben, és nincs semmilyen ésszerű módszerünk, hogy ezekre vonatkozóan érdemi információt szerezzünk. A valószínűségek valójában a lehetséges körülmények egy *potenciálisan végtelen* halmazát összegzik, amikor is a riasztó elmulaszt megszólalni (magas páratartalom, elektromos hálózat hibája, lemerült elem, elvágott drót, döglött egér beragadva a csengőbe…), vagy, hogy János és Mária elmulaszt értesíteni minket (ebédelni mentek, szabadságon vannak, időlegesen megsüketültek, egy áthaladó helikopter…). Ezen a módon egy kis ágens is képes megbirkózni egy igen bonyolult világgal, legalábbis közelítőleg. A közelítés mértéke tetszőlegesen javítható további releváns információk bevezetésével.

Most folytassuk a 14.2. ábrán mutatott feltételes eloszlásokkal. Az ábrán minden eloszlás mint **feltételes valószínűségi táblázat** – **FVT** (**conditional probability table**, **CPT**) van feltüntetve. (A táblázatos formát diszkrét változók esetén lehet használni; más reprezentációkat, amelyek már folytonos változóknál is használhatók, a 14.2. alfejezetben írunk le.) Az FVT-táblázatban minden sor az egyes csomóponti értékek feltételes valószínűségét tartalmazza az adott sorhoz tartozó **szülői feltétel** (**conditioning case**) esetén. A szülői feltétel a szülő csomópontok értékeinek egy lehetséges kombinációja (egyfajta elemi esemény, ha úgy tetszik). Az egyes sorokban szereplő számok összegének 1-et kell adnia, mivel az adott változó összes lehetséges értéke szerepel a bejegyzésekben. Bináris változók esetén, ha ismerjük, hogy az igaz érték valószínűsége p, a hamis érték valószínűségének $1 - p$-nek kell lennie, így gyakran a második számot elhagyjuk, ahogyan ezt a 14.2. ábrán is tettük. Általánosabban, egy k bináris szülővel rendelkező bináris változó esetén 2^k valószínűség adható meg tetszés szerint. Szülő nélküli csomópontok esetén a táblázat csak egyetlen sort tartalmaz, a változó egyes értékeinek a priori valószínűségeit.

2. A Bayes-hálók szemantikája

Az előző alfejezetben bemutattuk a hálót, de annak jelentését nem. A Bayes-hálók szemantikáját kétféle módon lehet megérteni. Az első szerint a háló az együttes valószínűség-eloszlás függvény egy leírása. A második szerint a háló feltételes függetlenségekről szóló állítások együttesét írja le. A két szemlélet ekvivalens, de az első inkább abban segít, hogyan *hozzunk létre* egy hálót, míg a második a következtetési eljárások tervezését segíti.

2.1. Az együttes valószínűség-eloszlás függvény leírása

A Bayes-háló a tárgytartomány teljes leírását adja meg. A benne található információk segítségével az együttes valószínűség-eloszlás függvény bármely bejegyzése kiszámítható.[3] Az együttes valószínűség-eloszlás függvény egy általános bejegyzése egy teljes – minden egyes változóhoz történő – hozzárendelés konjunkciójának a valószínűsége, úgymint $P(X_1 = x_1 \wedge \ \ldots \ \wedge \ X_n = x_n)$. Erre a továbbiakban a $P(x_1, \ldots, x_n)$ rövidítést fogjuk használni. Egy bejegyzés értékét a következő egyenlőség adja meg:

$$P(x_1,\ldots,x_n) = \prod_{i=1}^{n} P(x_i \mid \textit{szülők}(X_i)) \qquad (14.1)$$

ahol a *szülők*(X_i) a *Szülők*(X_i)-ben szereplő változók adott értékeinek együttesét jelöli. Így az együttes valószínűség-eloszlás függvényt leíró táblázat minden bejegyzése a Bayes-hálóban szereplő, feltételes valószínűségi táblák (FVT) megfelelő elemeinek a szorzata. Ezzel az FVT-k valójában az együttes valószínűség-eloszlás függvény dekomponált leírását adják meg. Ennek illusztrálására kiszámítjuk annak az eseménynek a valószínűségét, hogy a riasztó megszólal, de nem volt sem betörés, sem földrengés, azonban János és Mária is telefonál. A kezdőbetűkkel jelölve a változókat:

[3] A diszkrét változókra utaló szóhasználat ellenére az állítások analógiai folytonos változók esetén is fennállnak. (*A ford.*)

Created by XMLmind XSL-FO Converter.

$$P(j \wedge m \wedge a \wedge \neg b \wedge \neg e)$$

$$= P(j|a)P(m|a)P(a|\neg b \wedge \neg e)P(\neg b)P(\neg e)$$

$$= 0,90 \times 0,70 \times 0,001 \times 0,999 \times 0,998 = 0,00062$$

A 13.4. alfejezetben bemutattuk, hogy az együttes valószínűség-eloszlás függvény alapján a tárgytartománnyal kapcsolatos bármely kérdés megválaszolható. Ha egy Bayes-háló leírja az együttes valószínűség-eloszlás függvényt, akkor ez alapján bármely kérdés megválaszolható, összegezve a releváns együttes bejegyzéseket. A 14.4. alfejezet elmagyarázza, hogy ez hogyan végezhető el, azonban sokkal hatékonyabb módszereket is ismertet.

Egy módszer Bayes-hálók építésére

A (14.1) egyenlet definiálja, hogy mit is jelent egy adott Bayes-háló. Azonban arra nézve nem ad felvilágosítást, hogyan *építhetünk* olyan Bayes-hálót, hogy az általa meghatározott együttes valószínűség-eloszlás függvény az adott tárgytartomány megfelelő leírása legyen. Most megmutatjuk, hogy a (14.1) egyenlet tartalmaz bizonyos feltételes függetlenségi relációkat, amelyeket a tudásmérnök felhasználhat a háló topológiájának meghatározásánál. Elsőként írjuk fel az együttes valószínűség-eloszlás függvényt feltételes valószínűségek szorzataként, felhasználva a szorzatszabályt (lásd 13. fejezet):

$$P(x_1, \ldots, x_n) = P(x_n|x_{n-1}, \ldots, x_1)P(x_{n-1}, \ldots, x_1)$$

Majd ismételten alkalmazzuk ezt a lépést, minden egyes együttes valószínűséget felbontva egy feltételes valószínűségre és egy kisebb együttes valószínűségre. Végezetül egyetlen hosszú szorzatot kapunk:

$$P(x_1,\ldots,x_n) = P(x_n|x_{n-1},\ldots,x_1)P(x_{n-1}|x_{n-2},\ldots,x_1)\cdots P(x_2|x_1)P(x_1)$$
$$= \prod_{i=1}^{n} P(x_i|x_{i-1},\ldots,x_1)$$

Ez a **láncszabály**ként (**chain rule**) ismert azonosság valószínűségi változók bármely halmazára fennáll.

Összehasonlítva ezt a (14.1.) egyenlettel láthatjuk, hogy az együttes valószínűség-eloszlás függvény megadása ekvivalens azzal az általános állítással, hogy a háló minden X_i változójára

$$P(X_i|X_{i-1}, \ldots, X_1) = P(X_i|Szülők(X_i)) \quad (14.2)$$

feltéve, hogy $Szülők(X_i) \subseteq \{X_{i-1}, \ldots, X_1\}$. Ez utóbbi feltétel a csomópontok bármely olyan sorszámozásával teljesíthető, ami konzisztens a gráf struktúrájából adódó implicit részleges rendezéssel.

Fontos

A (14.2) egyenlet szerint, egy Bayes-háló csak abban az esetben lehet helyes reprezentációja a tárgytartománynak, ha az adott szülők mellett, minden csomópont feltételesen független a csomópontot sorrendezésben őt megelőzőktől. Így a tárgytartomány struktúrájának megfelelő Bayes-háló megépítése során, minden egyes csomóponthoz úgy kell a szülőket megválasztanunk, hogy ez a feltétel teljesüljön. Szemléletesen ez azt jelenti, hogy az X_i csomópont szülei halmazának tartalmaznia kell az $\{X_1, \ldots, X_{i-1}\}$ közül mindazokat a csomópontokat, amelyek *közvetlenül befolyásolják* X_i-t. Például tételezzük fel, hogy a 14.2. ábrán látható hálót már teljesen befejeztük, csupán a *MáriaTelefonál* szüleit kell még megválasztanunk. A *MáriaTelefonál*-t egyértelműen befolyásolja, hogy történt-e *Betörés* vagy *Földrengés*, de nem közvetlenül. A helyzettel kapcsolatos ismereteink alapján tudhatjuk, hogy ezek az események csak a riasztó által befolyásolhatják Mária telefonálással kapcsolatos viselkedését. Hasonlóan, ha a riasztó állapota ismert, akkor János hívásának bekövetkezte vagy elmaradása már nincs hatással Mária telefonálására. Formálisan fogalmazva, úgy véljük, hogy a következő feltételes függetlenség teljesül:

$$P(MáriaTelefonál|JánosTelefonál, Riasztás, Földrengés, Betörés)$$

$$= P(MáriaTelefonál|Riasztás)$$

Tömörség és a csomópontok sorrendje

Created by XMLmind XSL-FO Converter.

Amellett hogy egy teljes és nem redundáns reprezentációja a tárgytartománynak, egy Bayes-háló gyakran sokkal *tömörebb,* mint az együttes valószínűség-eloszlás függvény. Ez a tulajdonsága teszi használhatóvá a sokváltozós tárgyterületek kezelésében. A Bayes-háló tömörsége a **lokálisan strukturált** (**locally structured**) (vagy **ritka** – **sparse**) rendszerek egy igen általános tulajdonságának példája. Egy lokálisan strukturált rendszerben egy komponens csak korlátos számú más komponenssel van kapcsolatban közvetlenül, függetlenül a komponensek teljes számától. Lokális struktúrákhoz általában inkább a lineáris, mint az exponenciális komplexitásnövekedés kapcsolható. A Bayes-háló esetében jogos azt feltételezni, hogy a legtöbb tárgytartomány esetén egy valószínűségi változót csak *k* számú más változó befolyásol, ahol *k* konstans. Ha bináris változókat tételezünk fel az egyszerűség kedvéért, akkor az egy csomóponthoz tartozó feltételes valószínűségi tábla megadásához legfeljebb 2^k érték szükséges, így a teljes háló $n2^k$ értékkel megadható. Ezzel szemben az együttes valószínűség-eloszlás függvény 2^n értéket tartalmaz. Egy konkrét példával élve, ha 30 csomópontunk van ($n = 30$), és mindegyiknek legfeljebb 5 szülője van ($k = 5$), akkor a Bayes-háló 960 számot igényel, míg az együttes valószínűség-eloszlás függvény több mint egy milliárdot.

Léteznek olyan tárgytartományok, ahol a változók mindegyikét közvetlenül befolyásolhatja az összes többi, így a háló teljesen összekötött. A feltételes valószínűségi táblák megadása ekkor ugyanakkora mennyiségű információt igényel, mint az együttes valószínűség-eloszlás függvény megadása. Bizonyos tárgytartományokban léteznek olyan gyenge függőségek, amiket feltétlenül modellezni kell egy új kapcsolat felvételével. De ha ezek a függőségek igen gyengék, akkor lehet, hogy nem éri meg a háló komplexitását megnövelni a pontosság kismértékű növelésének érdekében. Például a betöréses hálónkkal kapcsolatban kifogásolható az, hogyha földrengés van, akkor Mária és János akkor sem telefonálna, ha hallanák a riasztót, mivel feltételezik, hogy a földrengés okozta. Az, hogy hozzákapcsoljuk-e a *Földrengés*-t a *MáriaTelefonál*-hoz és a *JánosTelefonál*-hoz (és így megnöveljük a táblákat) azon múlik, hogy mennyire fontos pontosabb valószínűségeket kapni, és mennyire költséges meghatározni az extra információt.

Fontos

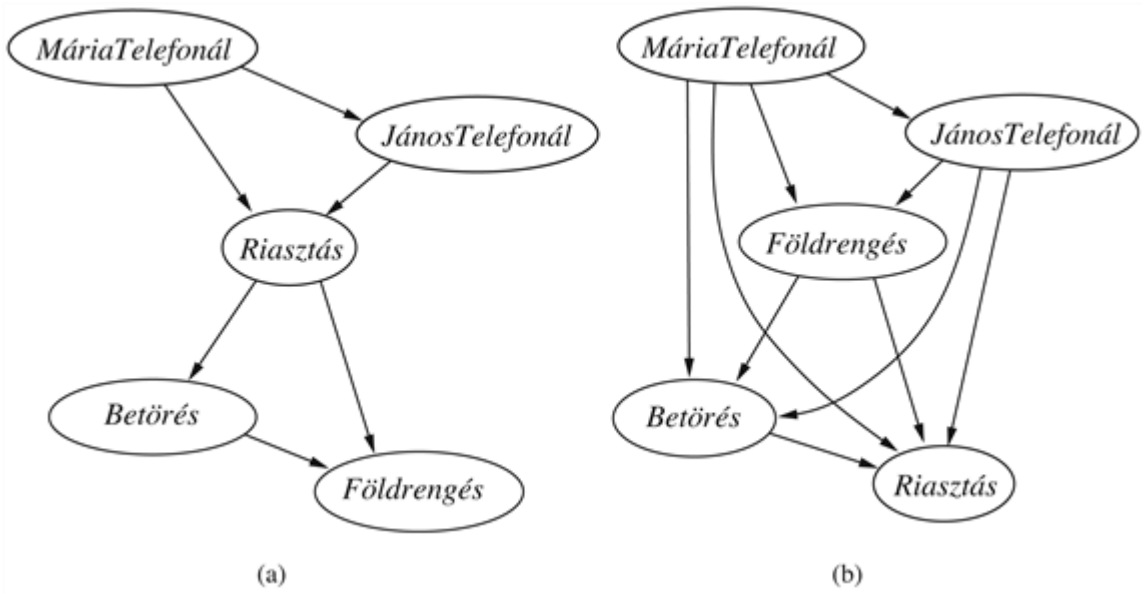
Azonban még egy lokálisan strukturált problémánál sem egyszerű lokálisan strukturált Bayes-hálót felépíteni. Ugyanis nemcsak azt követeljük meg, hogy minden egyes változót csak néhány másik befolyásoljon közvetlenül, hanem azt is, hogy a háló topológiája valóban mutassa azokat a közvetlen hatásokat a szülők megfelelő megválasztásával. A konstrukciós eljárásunk működése miatt előbb a „közvetlen befolyásolókat” kell a hálóhoz adni, ha azt szeretnénk, hogy szülőknek tudjuk őket választani az általuk befolyásolt csomópontnál. Ezért *a helyes sorrend a csomópontok hozzáadásánál az, hogy először az „alapvető okokat” adjuk a hálóhoz, majd a változókat, amelyeket befolyásolnak,* és ezt addig folytatjuk, amíg el nem érjük a „leveleket”, amelyeknek már nincs közvetlen okozati hatása más változókra.

Mi történik, ha történetesen rossz sorrendet választunk? Vizsgáljuk meg újra a betöréses példát. Tételezzük fel, hogy a változókat a következő sorrendben adjuk a hálóhoz: *MáriaTelefonál*, *JánosTelefonál*, *Riasztás*, *Betörés*, *Földrengés*. Ekkor egy kicsit bonyolultabb hálót kapunk (14.3. (a) ábrán). Az eljárás a következő:

- *MáriaTelefonál* hozzáadása: szülők nincsenek.
- *JánosTelefonál* hozzáadása: ha Mária telefonál, az valószínűleg azt jelenti, hogy a riasztó megszólalt, ami természetesen valószínűbbé teszi, hogy János telefonáljon. Így a *JánosTelefonál*-nak szükségszerűen szülője a *MáriaTelefonál*.

14.3. ábra - A háló struktúrája függ a hozzáadás sorrendjétől. A csomópontokat mindegyik hálóhoz felülről lefelé haladva adtuk hozzá.

Created by XMLmind XSL-FO Converter.



- *Riasztás* hozzáadása: nyilvánvaló, ha mindketten telefonálnak, akkor valószínűbb, hogy a riasztó megszólalt, mintha csak egyikük, vagy egyikük sem. Így mind a *JánosTelefonál*, mind a *MáriaTelefonál* szükséges mint szülő.

- *Betörés* hozzáadása: ha ismerjük a riasztó állapotát, akkor a Jánostól vagy Máriától jövő telefonhívások léte vagy hiánya csupán azt jelezhetik, hogy cseng-e a telefonunk, vagy hogy Mária zenéje hangosra van-e állítva, de a betörésről nem nyújtanak további információt. Azaz

$$P(\text{Betörés} \mid \text{Riasztás}, \text{JánosTelefonál}, \text{MáriaTelefonál}) = P(\text{Betörés} \mid \text{Riasztás})$$

Így csak a *Riasztás* szükséges mint szülő.

- *Földrengés* hozzáadása: ha a riasztó bekapcsolt, akkor valószínűbb, hogy bekövetkezett egy földrengés (mivel a riasztó valamiféle földrengésészlelő). De ha tudjuk, hogy betörés történt, akkor ez megmagyarázza a riasztást, és a földrengés valószínűsége csak picit nagyobb a normálisnál. Így a *Betörés* és a *Riasztás* is szükséges mint szülő.

Fontos

A kiadódó hálónak kettővel több éle van, mint az eredetinek a 14.2. ábrán, és hárommal több valószínűség meghatározását igényli. Ami még rosszabb, hogy néhány kapcsolódás megfoghatatlan viszonyt reprezentál, ami nehéz és nem természetes valószínűségi ítéleteket igényel, például a *Földrengés* feltételes valószínűségének a megbecslését, a *Betörés* és a *Riasztás* feltételekkel. Ez a jelenség igen általános, és kapcsolódik az okozati és diagnosztikai modelleknek a 8. fejezetben bevezetett megkülönböztetéséhez. Ha úgy próbálunk meg egy diagnosztikai modellt megépíteni, hogy a kapcsolatok okozat-ok irányúak (mint például a *MáriaTelefonál*-tól a *Riasztás*-ig vagy a *Riasztás*-tól a *Betörés*-ig kapcsolat), akkor végül egyébként független okok között kell függést meghatároznunk (és gyakran a különállóan bekövetkező okozatok között is). *Ha ragaszkodunk az okozati modellhez, akkor kevesebb értéket kell megadnunk, és az értékeket általában könnyebb meghatározni*. Orvosi tárgytartományon például Tversky és Kahneman demonstrálta, hogy orvos szakértők szívesebben hoznak valószínűségi ítéleteket okozati, mint diagnosztikai összefüggések esetén (Tversky és Kahneman, 1982).

A 14.3. (b) ábra a változók igazán szerencsétlen sorrendjét mutatja: *MáriaTelefonál*, *JánosTelefonál*, *Földrengés*, *Betörés*, *Riasztás*. Ez a háló 31 önálló valószínűség meghatározását igényli – pontosan annyit, amennyi a teljes együttes valószínűség-eloszlás függvény megadása. Azonban fontos felismerni, hogy a három háló bármelyike képes *pontosan ugyanannak az együttes valószínűség-eloszlás függvénynek a reprezentálására*. A két utóbbi egyszerűen csak nem reprezentálja az összes feltételes függetlenségi relációt, és így rengeteg szükségtelen érték meghatározására kényszerül.

Created by XMLmind XSL-FO Converter.

2.2. Feltételes függetlenségi relációk Bayes-hálókban

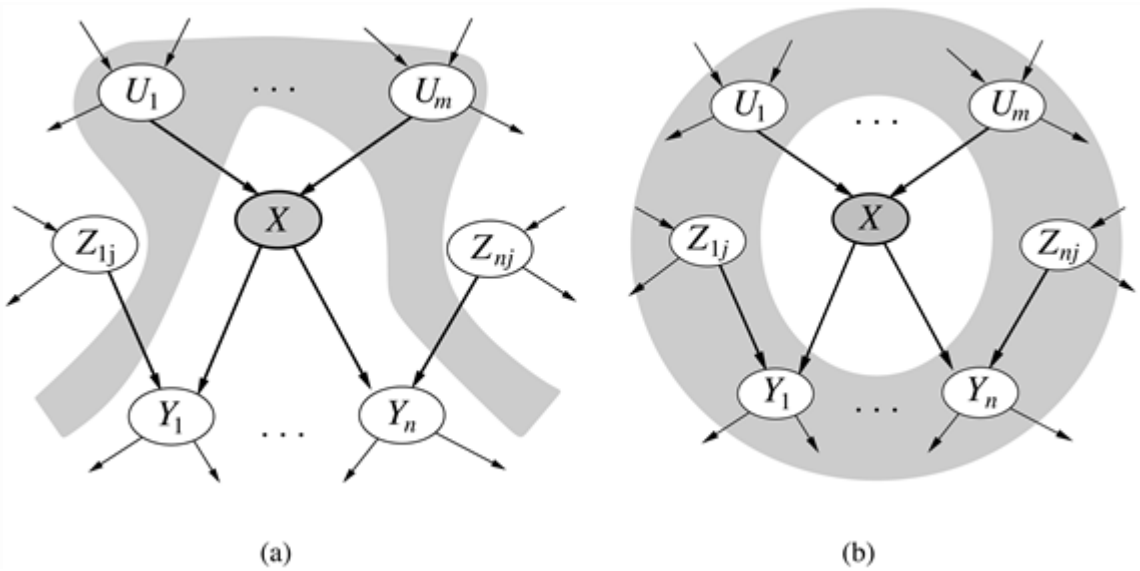
A Bayes-hálókra egy „numerikus” szemantikát adtunk meg a teljes együttes eloszlás reprezentációjának a szempontjából, mint a (14.1) egyenletben. Ezt a szemantikát alkalmazva a Bayes-hálók konstrukciós módszereinek a származtatásánál azt a következményt kaptuk, hogy egy csomópont feltételesen független az őt megelőzőktől, ha a csomópont szülői adottak. Kiderül, hogy más módon is eljárhatunk. Elindulhatunk egy „topológiai” szemantikától, ami a gráf által kódolt feltételes függetlenségi relációkat adja meg, és ezekből származtatjuk a „numerikus” szemantikát. A topológiai szemantikát a következő két egymással ekvivalens meghatározás bármelyike rögzíti:[4]

1. Egy csomópont feltételesen független a nem **leszármazott**aitól (non-**descendant**s), feltéve, hogy a szülei adottak. Például a 14.2. ábrán *JánosTelefonál* független a *Betörés*-től és a *Földrengés*-től, ismerve a *Riasztás* értékét.

2. Egy csomópont feltételesen független az összes többi csomóponttól a hálózatban, a szülei, gyermekei és gyermeki szüleinek az ismeretében – azaz a **Markov-takaró**jának (**Markov blanket**) ismeretében. Például, a *Betörés* független *JánosTelefonál*-tól és *MáriaTelefonál*-tól, a *Betörés*-t és *Földrengés*-t ismerve.

Ezeket a meghatározásokat mutatja be a 14.4. ábra. Ezekből a feltételes függetlenségi állításokból és az FVT-kből a teljes együttes eloszlást rekonstruálni lehet; így a „numerikus” szemantika és a „topológiai” szemantika ekvivalens.

14.4. ábra - (a) Az *X* csomópont feltételesen független a nem leszármazottaitól (Z_{ij}-ktől) a szülei (a szürke területen látható U_i-k) ismeretében. (b) Az *X* csomópont feltételesen független a háló összes többi csomópontjától a Markov-takarójának ismeretében (a szürke terület).



3. Feltételes eloszlások hatékony reprezentációja

Még ha a szülők maximális száma, *k* meglehetősen kicsi is, egy csomópont feltételes valószínűségi táblájának kitöltése akár $O(2^k)$ számú értéket és az összes lehetséges feltételes esetet figyelembe véve is nagy szakértelmet igényelhet. Valójában azonban az a legrosszabb eset, amikor a kapcsolat a szülők és a gyermek között teljesen önkényes. Általában az ilyen kapcsolatok egy **kanonikus eloszlás**sal (**canonical distribution**) írhatók le,

[4] Létezik egy további általános topológiai kritérium a **d-elválasztás** (**d-separation**) annak eldöntésére, hogy a csomópontok egy **X** halmaza független-e egy másik **Y** halmaztól egy harmadik **Z** halmaz feltétel esetén. A kritérium elég bonyolult, és nem szükséges a fejezetben az algoritmusok származtatásánál, ezért nem tárgyaljuk. A részletek megtalálhatók Russell és Norvig, valamint Pearl munkáiban (Russell és Norvig, 1995; Pearl, 1988). Shachter egy sokkal szemléletesebb módszert ad a d-elválasztások meghatározásához (Shachter, 1998).

Created by XMLmind XSL-FO Converter.

amelyek valamilyen szabványos mintát követnek. Ilyen esetekben a teljes tábla megadható a mintázat és esetleg néhány paraméter meghatározásával – sokkal könnyebben, mint exponenciális számú paraméter megadásával.

A legegyszerűbb példát a **determinisztikus csomópont**ok (**deterministic node**s) szolgáltatják. Egy determinisztikus csomópont értékét a szüleinek az értéke teljesen meghatározza, mindenfajta bizonytalanságtól mentesen. A reláció lehet egy logikai kapcsolat – például ha a szülőcsomópontok azt jelentik, hogy *Kanadai*, *Egyesült Államokbeli* és *Mexikói*, a gyermekcsomópont pedig azt, hogy *Észak-Amerikai*, akkor a közöttük lévő kapcsolat egyszerűen a szülők diszjunkciója. A reláció lehet numerikus is – például ha a szülőcsomópontok egy gépkocsi különböző árai különböző forgalmazóknál, a gyermekcsomópont pedig az az ár, amit egy legolcsóbbat kereső vevő végezetül fizetne, akkor a gyermekcsomópont értéke a szülők értékeinek a minimuma. Másik példa, ha a szülőcsomópontok az egy tóba bejövő vízmennyiségek (folyók, vízlevezetők, csapadék) és az onnan eltávozó vízmennyiségek (folyók, párolgás, elszivárgás), a gyermek pedig a tó szintjének a megváltozása, akkor a gyermek értéke a kifolyó és befolyó szülők értékeinek a különbsége.

Bizonytalan relációkat gyakran jellemezhetünk úgynevezett „zajos” logikai relációkkal. A mintapélda erre az úgynevezett **zajos-VAGY** (**noisy-OR**) reláció, ami a logikai VAGY reláció általánosítása. Ítéletlogikában kijelenthetjük, hogy a *Láz* akkor és csak akkor igaz, ha a *Megfázás* vagy az *Influenza* vagy a *Malária* igaz. A zajos-VAGY modell megenged bizonytalanságot, hogy egyes szülők okozhatják-e a gyermekek igaz értékét – az okozati kapcsolat a szülő és gyermek között *gátolt* lehet, és így lehet, hogy a páciens meg van fázva, de nincs láza. A modell két feltevésre épül. Elsőként feltételezi, hogy az összes lehetséges ok fel van sorolva. (Ez nem annyira szigorú megkötés, mint amilyennek tűnik, mivel mindig létrehozhatunk egy úgynevezett **szivárgáscsomópont**ot (**leak node**), ami „vegyes okokat” fed le.) Másodikként felteszi, hogy bármely szülő gátlása független a többi szülő gátlásától: például akármi is gátolja, hogy a *Malária* lázat okozzon, ez független attól, hogy mi gátolja az *Influenzá*-t, hogy lázat okozzon. Ezekkel a feltevésekkel a *Láz* akkor és csak akkor *hamis*, ha az összes *igaz* értékű szülő gátolt, aminek a valószínűsége a gátlás-valószínűségek szorzata. Tételezzük fel, hogy ezek az önálló gátlási valószínűségek a következők:

$$P(\neg láz \mid megfázás, \neg influenza, \neg malária) = 0{,}6$$

$$P(\neg láz \mid \neg megfázás, influenza, \neg malária) = 0{,}21$$

$$P(\neg láz \mid \neg megfázás, \neg influenza, malária) = 0{,}1$$

Ekkor – ennyi információból és a zajos-VAGY feltevésből – a teljes FVT-t fel lehet építeni. A következő táblázat azt mutatja, hogy hogyan:

Megfázás	*Influenza*	*Malária*	*P*(*Láz*)	*P*(¬*Láz*)
H	H	H	0,0	1,0
H	H	I	0,9	**0,1**
H	I	H	0,8	**0,2**
H	I	I	0,98	$0{,}02 = 0{,}2 \times 0{,}1$
I	H	H	0,4	**0,6**
I	H	I	0,94	$0{,}06 = 0{,}6 \times 0{,}1$
I	I	H	0,88	$0{,}12 = 0{,}6 \times 0{,}2$
I	I	I	0,988	$0{,}012 = 0{,}6 \times 0{,}2 \times 0{,}1$

Általánosságban, a zajos logikai relációk, amelyekben egy változó k számú szülőtől függ, $O(k)$ paraméterrel írhatók le, a teljes feltételes valószínűség-eloszlás táblázathoz tartozó $O(2^k)$ helyett. Ez sokkal könnyebbé teszi a becslést és a tanulást. Például a CPSC-háló (Pradhan és társai, 1994) zajos-VAGY- és zajos-MAX-eloszlásokat használ a betegségek és tünetek közötti kapcsolatok modellezésére. 448 csomópont és 906 él esetén ez csak 8254 értéket igényel a 133 931 430 helyett, ami egy teljes FVT-ket használó háló esetén lenne szükséges.

3.1. Bayes-hálók folytonos változókkal

Számos valós problémában fordulnak elő folytonos mennyiségek, mint a magasság, tömeg, hőmérséklet és pénz; valójában a statisztika nagy része olyan valószínűségi változókkal foglalkozik, amelyek értéktartománya

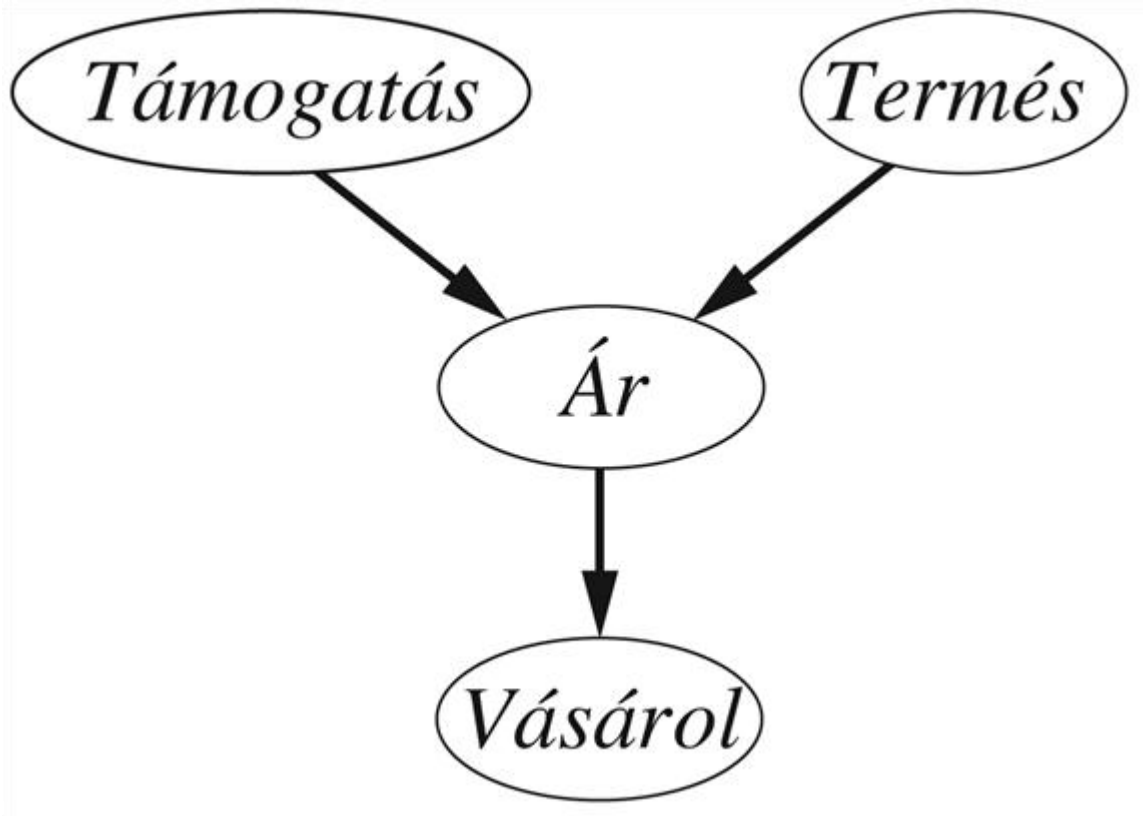
Created by XMLmind XSL-FO Converter.

folytonos. Definíció szerint, a folytonos változóknak végtelen számú értéke lehet, így lehetetlen feltételes valószínűségeket megadni minden egyes értékre. Egy lehetséges módszer a folytonos változók kezelésére, ha elkerüljük őket **diszkretizálás**sal (**discretization**) – azaz felosztjuk a lehetséges értékeket intervallumok adott halmaza szerint. Például, a hőmérsékletet feloszthatjuk (<0°C), (0°C–100°C) és (>100°C) intervallumokra. A diszkretizálás néha adekvát megoldás, de gyakran eredményezi a pontosság jelentős romlását, valamint nagyon nagy FVT-ket. Egy másik megoldás, ha a valószínűség sűrűségfüggvények alapvető családjaiból választunk (lásd A) függelék), amelyek véges számú **paraméter**rel megadhatók. Például, a Gauss- (vagy normál) eloszláshoz $N(\mu, \sigma^2)(x)$ a μ átlag és σ^2 szórásnégyzet tartozik mint paraméterek.

Egy diszkrét és folytonos változókat is tartalmazó hálót **hibrid Bayes-háló**nak (**hybrid Bayesian network**) nevezünk. Egy hibrid háló megadásához két újfajta eloszlást kell megadnunk: feltételes eloszlást folytonos változóhoz diszkrét és/vagy folytonos szülők esetén; továbbá feltételes eloszlást diszkrét változókhoz folytonos szülők esetén. Fontoljuk meg a 14.5. ábra egyszerű példáját, amelyben a vásárló valamilyen gyümölcsöt vásárol az ára függvényében, ami viszont a termés mennyiségétől függ és attól, hogy éppen van-e állami támogatás. Az *Ár* változó folytonos, a szülei pedig folytonosak és diszkrétek; a *Vásárol* változó diszkrét, és van egy folytonos szülője.

Az *Ár* változóhoz meg kell adnunk a **P**(*Ár*| *Termés*, *Támogatás*) eloszlást. A diszkrét szülőt explicit felsorolással kezeljük – azaz megadjuk mind a *P*(*Ár*| *Termés*, *támogatás*), mind a *P*(*Ár*| *Termés*, ¬*támogatás*) valószínűségeket. A *Termés* kezeléséhez megadjuk, hogy a *c* ár feletti eloszlása hogyan függ a *t Termés* folytonos értékétől. Máshogy fogalmazva, az ár eloszlásának a *paramétereit* a termés *t* értékének függvényében adjuk meg.

14.5. ábra - Egy egyszerű háló diszkrét (*Támogatás* és *Vásárol*) és folytonos (*Termés* és *Ár*) változókkal



A leggyakoribb választás a **lineáris Gauss-eloszlás** (**linear Gaussian**), amelyben a gyermek Gauss-eloszlású, ahol a μ várható érték lineárisan változik a szülő értékével, és ahol a δ szórás rögzített. Két eloszlásra van szükségünk, a *támogatás* és a ¬*támogatás* esetére különböző paraméterekkel:

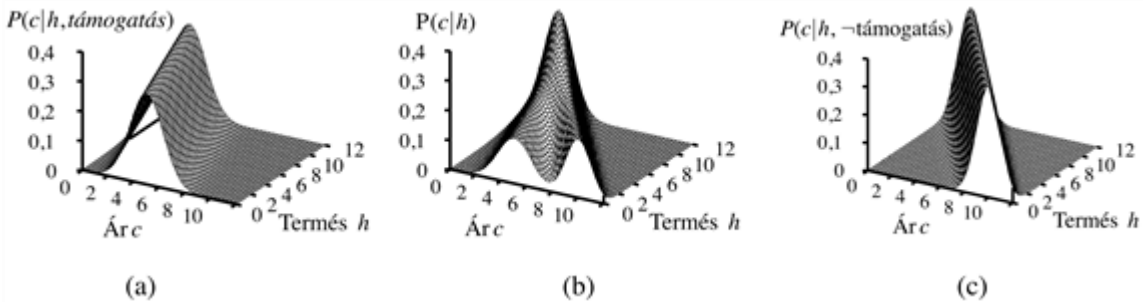
Created by XMLmind XSL-FO Converter.

$$P(c|t, támogatás) = N(a_i t + b_i, \sigma_i^2)(c) = \frac{1}{\sigma_i \sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{c-(a_i t+b_i)}{\sigma_i}\right)^2}$$

$$P(c|t, \neg támogatás) = N(a_h t + b_h, \sigma_h^2)(c) = \frac{1}{\sigma_h \sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{c-(a_h t+b_h)}{\sigma_h}\right)^2}$$

Ebben a példában ekkor, az *Ár* feltételes eloszlását a lineáris normális eloszlás kiválasztásával és az a_i, b_i, σ_i, a_h, b_h *és* σ_h paraméterekkel adhatjuk meg. A 14.6. (a) és (b) ábra mutatják ezt a két kapcsolatot. Figyeljük meg, hogy mindegyik esetben a meredekség negatív, mivel az ár csökken a kínálat növekedésével. (Természetesen a linearitás feltevése azt is jelenti, hogy az ár egy bizonyos pontnál negatív lesz; a lineáris modell csak akkor ésszerű, ha a termés mennyiségét egy szűk tartományra korlátozzuk.) A 14.6. (c) ábra a $P(c|t)$ eloszlást mutatja, átlagolva a *Támogatás* két lehetséges értéke felett feltételezve, hogy mindegyik a priori valószínűsége 0,5. Ez mutatja, hogy még igen egyszerű modellekkel is, elég érdekes eloszlások reprezentálhatók.

14.6. ábra - Az (a) és (b) grafikonok az *Ár* valószínűség-eloszlását mutatják a *Termés* függvényében és a *Támogatás* igaz és hamis értéke mellett. A (c) diagram a P(*Ár*| *Termés*) eloszlást mutatja, ami a két aleset összegzéseként adódik.



A lineáris normális feltételes eloszlásnak vannak bizonyos speciális tulajdonságai. Egy csak folytonos, lineáris normális feltételes eloszlású változókat tartalmazó háló együttes eloszlása egy többváltozós normális eloszlás az összes változó felett (lásd 14.5. feladat).[5] [A többváltozós normális eloszlás egy felület több mint egy dimenzióban, aminek van egy csúcsa az átlagnál (*n* dimenzióban), értéke pedig ettől távolodva minden irányban csökken.] Ha diszkrét változókat adunk a hálóhoz (feltéve, hogy egyetlen diszkrét változó sem gyermeke egy folytonos változónak), a háló egy **feltételes Gauss**- (**conditional Gaussian**) vagy FG-eloszlást definiál: bármilyen értéket is rendelünk a diszkrét változókhoz, a folytonos változók feletti eloszlás egy többváltozós Gauss-eloszlás lesz.

Most a folytonos szülővel rendelkező diszkrét változók eloszlásával kezdünk foglalkozni. Fontoljuk meg például a *Vásárol* csomópontot a 14.5. ábrán. Ésszerűnek tűnik azt feltételezni, hogy a vásárló vásárol, ha az ár alacsony, nem vásárol, ha magas, a vásárlás valószínűsége pedig folytonosan változik egy közbenső régióban. Máshogy fogalmazva, a feltételes eloszlás hasonló egy „elmosódott” („soft”) küszöbfüggvényhez. Az elmosódott küszöbök létrehozására egy módszer a standard normális eloszlás *integrál*jának a használata:

$$\Phi(x) = \int_{-\infty}^{x} N(0,1)(x)dx$$

Ekkor a *Vásárlás* valószínűsége az *Ár* ismeretében ez lehet

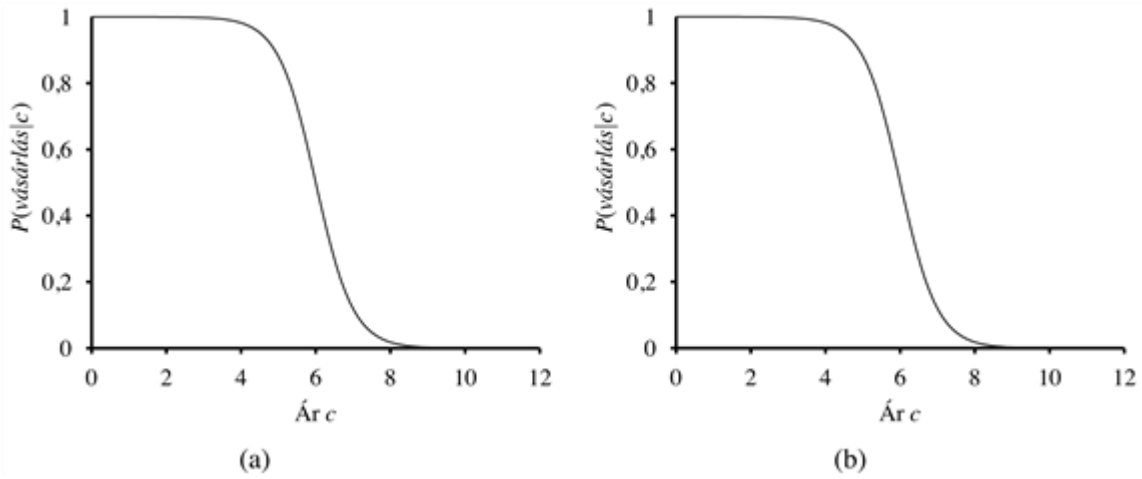
$$P(vásárol|Ár = c) = \Phi((- c = \mu)/\sigma)$$

ami azt jelenti, hogy az ár küszöbe μ körül van, és a küszöbrégió szélessége arányos δ-val, illetve, hogy a vásárlás valószínűsége csökken, ahogy az ár növekszik.

14.7. ábra - (a) A *Vásárlás* valószínűségének probit eloszlása az *Ár* ismeretében, μ = 6,0 és σ = 1,0 mellett. (b) Logit eloszlás hasonló paraméterekkel.

[5] Következésképpen lineáris normális hálókban a következtetés időigénye legrosszabb esetben is csak $O(n^3)$, függetlenül a háló topológiájától. A 14.4. alfejezetben látni fogjuk, hogy diszkrét változós hálóban a következtetés NP-nehéz.

Created by XMLmind XSL-FO Converter.



Ezt a **probit eloszlás**t (**probit distribution**) a 14.7. (a) ábra illusztrálja. Alakja azzal az érveléssel igazolható, hogy az alapul szolgáló döntési folyamatnál létezik egy pontos küszöb, de ennek pontos helyét egy véletlen normális eloszlású zaj befolyásolja. A probit modell egy alternatívája a **logit eloszlás** (**logit distribution**), amely a **szigmoid függvény**t (**sigmoid function**) használja egy elmosódott küszöb előállításához:

$$P(\text{vásárol} \mid \text{Ár} = c) = \frac{1}{1 + \exp(-2(\frac{-c+\mu}{\sigma})}$$

Ezt a 14.7. (b) ábra mutatja. A két eloszlás hasonlóan néz ki, de valójában a logit sokkal lassabban tart a határértékekhez. A probit gyakran jobban illeszkedik a valódi helyzetekhez, de a logit esetenként matematikailag könnyebben kezelhető, például széles körben használatos a neurális hálókban (lásd 20. fejezet). Mind a probit, mind a logit általánosítható több folytonos szülőre a szülők értékeinek lineáris kombinációját véve. Többértékű diszkrét gyermekre történő kiterjesztéseket a 14.6. feladatban taglalunk.

4. Egzakt következtetés Bayes-hálókban

Az alapvető feladat bármely valószínűségi következtető rendszer számára az, hogy kiszámítsa a **célváltozó**k (**query variable**s) egy halmazának a posteriori valószínűség-eloszlását egy adott megfigyelt **esemény** (**event**) esetén – azaz **bizonyítékváltozó**k (**evidence variable**s) egy halmazához történő érték-hozzárendelés esetén. A 13. fejezetben bevezetett jelölést fogjuk használni: X a célváltozót jelöli; $\mathbf{E}$ a bizonyítékváltozók $E_1, \ldots, E_m$ halmazát, $\mathbf{e}$ pedig a megfigyelt eseményt; $\mathbf{Y}$ fogja jelölni az (olykor **rejtett változó**knak (**hidden variable**s) nevezett) nem-bizonyítékváltozók $Y_1, \ldots, Y_k$ halmazát. Így a változók teljes halmaza $\mathbf{X} = \{X\} \cup \mathbf{E} \cup \mathbf{Y}$. Egy jellemző lekérdezés a $\mathbf{P}(X \mid \mathbf{e})$ a posteriori eloszlásra irányul.[6]

A riasztós hálóban például megfigyelhetnénk azt az eseményt, ahol *JánosTelefonál* = *igaz* és *MáriaTelefonál* = *igaz*. Ekkor megkérdezhetnénk mondjuk a betörés megtörténtének a valószínűségét:

$$\mathbf{P}(\text{Betörés} \mid \text{JánosTelefonál} = \text{igaz}, \text{MáriaTelefonál} = \text{igaz}) = \langle 0{,}284,\ 0{,}716 \rangle$$

Ebben a fejezetben az a posteriori valószínűségek kiszámítására szolgáló egzakt algoritmusokat tárgyalunk, és átgondoljuk ennek a feladatnak a komplexitását. Kiderül, hogy általános esetben ez kivitelezhetetlen, ezért a 14.5. alfejezet közelítő következtetési módszereket ismertet.

4.1. Következtetés felsorolással

[6] Feltesszük, hogy a célváltozó nincs a bizonyítékváltozók között; amúgy X a posteriori eloszlása a megfigyelt értékre 1 valószínűséget ad. Az egyszerűség kedvéért azt is feltettük, hogy a kérdés egyetlen változóra irányul. Az algoritmusaink könnyen kiterjeszthetők több változó együttesének lekérdezésére.

Created by XMLmind XSL-FO Converter.

A 13. fejezet elmagyarázta, hogy bármely feltételes valószínűség kiszámítható a teljes együttes eloszlás tagjainak összegzésével. Pontosabban, egy **P**(*X*| **e**) lekérdezés megválaszolható a (13.6) egyenlet felhasználásával, amit a jobb követhetőség kedvéért itt megismétlünk:

$$\mathbf{P}(X|e) = \alpha \mathbf{P}(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y})$$

Fontos

Továbbá, mint ismeretes, a Bayes-háló a teljes együttes eloszlás egy teljes reprezentációját nyújtja. Még pontosabban, a (14.1) egyenlet azt mutatja, hogy az együttes eloszlás $P(x, \mathbf{e}, \mathbf{y})$ tagjai felírhatók a hálóból származó feltételes valószínűségek szorzataiként. Ezért *egy lekérdezés megválaszolható a Bayes-háló felhasználásával, kiszámítva a hálóból származó feltételes valószínűségek szorzatainak az összegét*.

A 13.4. ábrán a `FELSOROL-EGYÜTTES-KÉRDEZÉS` algoritmust adtuk meg a teljes együttes eloszlásból felsorolással történő következtetésre. Az algoritmus bemenetként fogadja a **P** teljes együttes eloszlást, és értékeket néz meg benne. Egyszerű módosítással elérhető, hogy az algoritmus egy *bn* Bayes-hálót fogad bemenetként, és az együttes bejegyzéseket a *bn* megfelelő FVT-bejegyzéseinek összeszorzásával „nézi meg”.

Vegyük a **P**(*Betörés*| *JánosTelefonál* = *igaz*, *MáriaTelefonál* = *igaz*) lekérdezést. A rejtett változók ennél a kérdésnél a *Földrengés* és a *Riasztás*. A (13.6) egyenletből, a kifejezések rövidítése miatt a kezdőbetűket használva, azt kapjuk, hogy[7]

$$\mathbf{P}(B|j,m) = \alpha \mathbf{P}(B, j, m) = \alpha \sum_{e} \sum_{a} \mathbf{P}(B, e, a, j, m)$$

A Bayes-hálók szemantikája – (14.1) egyenlet – pedig az FVT-bejegyzések felhasználásával egy kifejezést ad. Az egyszerűség kedvéért ezt csak a *Betörés* = *igaz* esetére adjuk meg:

$$P(b|j,m) = \alpha \sum_{e} \sum_{a} P(b)P(e)P(a|b,e)P(j|a)P(m|a)$$

Ennek a kifejezésnek a kiszámításához négy tagot kell összeadnunk, amelyek mindegyikét öt szám összeszorzásával kapjuk. Legrosszabb esetben, amikor majdnem minden változó felett összegezni kell, az algoritmus komplexitása egy *n* bináris változós háló esetén $O(n2^n)$.

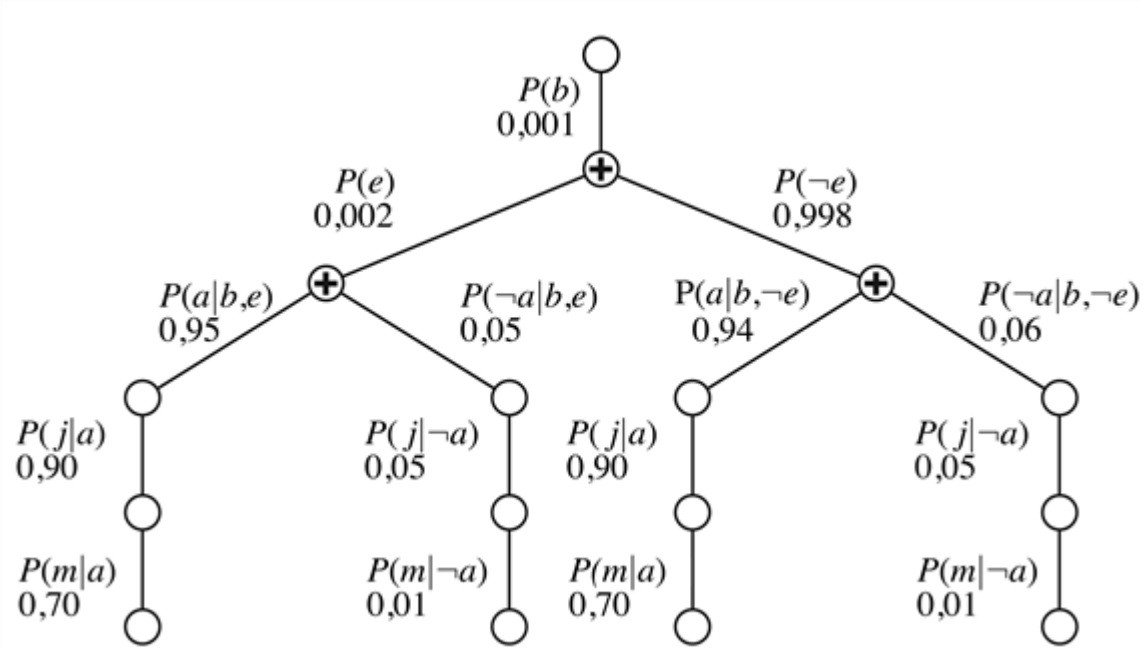
A következő egyszerű megfigyeléssel egy javításhoz juthatunk: a $P(b)$ tag állandó, és ki lehet vinni az *a* és *e* feletti összegzések elé, a $P(e)$ tag pedig kivihető az *a* feletti összegzés elé. Így azt kapjuk, hogy

$$P(b|j,m) = \alpha P(b) \sum_{e} P(e) \sum_{a} P(a|b,e) P(j|a) P(m|a) \qquad (14.3)$$

14.8. ábra - A (14.3) egyenletben szereplő kifejezés struktúrája. A kiértékelés fentről lefelé halad, összeszorozva az értékeket az egyes útvonalak mentén, és összegezve a „+” csomópontoknál. Vegyük észre, a *j*-hez és az *m*-hez tartozó útvonalak megismétlődését.

[7] Egy olyan kifejezés mint a $\sum_e P(a,e)$ a $P(A = a, E = e)$ -nek az összegzését jelenti minden lehetséges *e* értékre. Ez többértelműséget rejt magában, mivel a $P(e)$ jelölést használják mind a $P(E = igaz)$, mind a $P(E = e)$ jelentés kifejezésére, azonban a szándékolt jelentésnek a szövegkörnyezetből ki kell derülnie; egy összegzés esetében az utóbbiról van szó.

Created by XMLmind XSL-FO Converter.



Ez a kifejezés egy ciklussal értékelhető ki, sorban végighaladva a változókon, miközben összeszorozzuk az FVT-bejegyzéseket. Minden összegzésnél, a változó lehetséges értékei szerint is iterálnunk kell. A számítás struktúráját a 14.8. ábra mutatja. Felhasználva a 14.2. ábra értékeit, azt kapjuk hogy $P(b|\, j,m) = \alpha \times 0{,}00059224$. A $\neg b$-hez tartozó számítást átfutva az $\alpha \times 0{,}0014919$-et eredményez; így

14.9. ábra - A felsoroló algoritmus Bayes-hálós lekérdezések megválaszolására

```
function FELSOROLÁS-KÉRDEZÉS(X, e, bn) returns egy X feletti eloszlás
  inputs: X, a lekérdezés változója
          e, az E változók megfigyelt értékei
          bn, egy valószínűségi háló {X} ∪ E ∪ Y /*Y = rejtett változók*/

  Q(X) ← egy eloszlás X felett, kezdetben üres
  for each x_i értékére X-nek do
      terjeszd ki e-t X-nek az x_i értékével
      Q(x_i) ← FELSOROL-MINDET(VÁLTOZÓ[bn], e)
  return NORMALIZÁL(Q(X))

function FELSOROL-MINDET(változók, e) returns egy valós szám
  if ÜRES?(változók) then return 1.0
  Y ← ELSŐ(változók)
  if Y értéke y e-ben
      then return P(y|szülők(Y)) × FELSOROL-MINDET(MARADÉK(változók), e)
      else return Σ_y P(y|szülők(Y)) × FELSOROL-MINDET(MARADÉK(változók), e_y)
        ahol e_y ki lett terjesztve Y = y-nal
```

$P(B|\, j,m) = \alpha \langle 0{,}00059224,\ 0{,}0014919 \rangle \approx \langle 0{,}284,\ 0{,}716 \rangle$

Azaz a betörés valószínűsége, ha mindkét szomszéd telefonál körülbelül 28%.

A 14.8. ábra a (14.3) egyenlet kiértékelési folyamatát mutatja, kifejezés fa alakban. A `FELSOROL-KÉRDEZ` algoritmus a 14.9. ábrán ilyen fákat mélységi rekurzióval értékel ki. Így a `FELSOROL-KÉRDEZ` tárkomplexitása csak lineáris a változók számában – az algoritmus így hatékonyan, anélkül összegez a teljes együttes eloszlás felett, hogy azt explicit módon megkonstruálná. Sajnos az algoritmus időkomplexitása egy n bináris változós hálónál minden esetben $O(2^n)$ – ami jobb, mint az egyszerű megközelítéshez tartozó $O(n2^n)$, de még mindig elég

Created by XMLmind XSL-FO Converter.

nyomasztó. Egy figyelemre méltó dolog a 14.8. ábra fájával kapcsolatban, hogy nyilvánvalóvá teszi a *megismételt alkifejezéseket,* amelyeket az algoritmus kiértékel. A $P(j \mid a)\, P(m \mid a)$ és a $P(j \mid \neg a)\, P(m \mid \neg a)$ szorzatok kétszer kerülnek kiszámításra, *e* minden értékénél. A következő fejezet egy általános módszert ír le, ami elkerüli az ilyen felesleges újraszámítások elvégzését.

4.2. A változó eliminációs algoritmus

A felsoroló algoritmus lényegesen javítható a 14.8. ábra által szemléltetett ismétlődő számítások kiküszöbölésével. Az ötlet egyszerű: egyszer végezzük el a számítást, és őrizzük meg az eredményeket későbbi használatra. Ez egyfajta dinamikus programozás. Ennek a megközelítésnek több változata is létezik; mi a legegyszerűbbet, a **változó eliminációs (variable elimination)** algoritmust mutatjuk be. A változó eliminálás a (14.3) egyenlet típusú kifejezéseket *jobbról balra* (azaz a 14.8. ábrán *lentről felfelé*) sorrendben értékeli ki. A köztes eredményeket eltároljuk, és bármely változó feletti összegzés a kifejezésnek csak azon része felett történik meg, amely függ ettől a változótól.

Szemléltessük ezt a folyamatot a betörés háló esetén. Értékeljük ki a következő kifejezést:

$$\mathbf{P}(B \mid j, m) = \alpha \underbrace{P(B)}_{B} \sum_{e} \underbrace{P(e)}_{E} \sum_{a} \underbrace{P(a \mid B, e)}_{A} \underbrace{P(j \mid a)}_{J} \underbrace{P(m \mid a)}_{M}$$

Vegyük észre, hogy a kifejezés minden részét megjelöltük a kapcsolódó változó nevével; ezeket **tényező**knek (**factor**s) nevezzük. A lépések a következők:

- Az *M* tényező, $P(m \mid a)$ nem igényel *M* feletti összegzést (mivel *M* értéke már rögzített). Eltároljuk a valószínűséget *a* minden értékére egy kételemű vektorban,

$$\mathbf{f}_M(A) = \begin{pmatrix} P(m \mid a) \\ P(m \mid \neg a) \end{pmatrix}$$

(Az $\mathbf{f}_M$ azt jelöli, hogy **f** létrehozásában *M*-et felhasználtuk.)

- Hasonlóan, tároljuk a *J*-hez tartozó tényezőt egy kételemű $\mathbf{f}_J(A)$ vektorban.
- Az *A*-hoz tartozó tényező, $\mathbf{P}(a \mid B, e)$, ami egy 2 × 2 × 2-es mátrix lesz, $\mathbf{f}_A(A, B, E)$.
- Most ki kell összegeznünk *A*-t ennek a három tényezőnek a szorzatából. Ez egy 2 × 2-es mátrixot eredményez, aminek indexei már csak *B* és *E* felett futnak. A mátrix nevében *A*-nál egy felülvonással jelezzük, hogy *A* már ki lett összegezve:

$$\begin{aligned} \mathbf{f}_{\bar{A}JM}(B, E) &= \sum_{a} \mathbf{f}_A(a, B, E) \times \mathbf{f}_J(a) \times \mathbf{f}_M(a) \\ &= \mathbf{f}_A(a, B, E) \times \mathbf{f}_J(a) \times \mathbf{f}_M(a) \\ &+ \mathbf{f}_A(\neg a, B, E) \times \mathbf{f}_J(\neg a) \times \mathbf{f}_M(\neg a) \end{aligned}$$

Az itt használt szorzást **pontonkénti szorzás**nak (**pointwise product**) nevezik, és hamarosan ismertetjük.

- Hasonló módon dolgozzuk fel *E*-t: kiösszegezzük *E*-t $\mathbf{f}_E(E)$ és $\mathbf{f}_{\bar{A}JM}(B, E)$ szorzatából:

$$\mathbf{f}_{\bar{E}\bar{A}JM}(B) = \mathbf{f}_E(e) \times \mathbf{f}_{\bar{A}JM}(B, e) + \mathbf{f}_E(\neg e) \times \mathbf{f}_{\bar{A}JM}(B, \neg e).$$

Most pedig kiszámíthatjuk az eredményt egyszerűen összeszorozva *B* tényezőjét (azaz $\mathbf{f}_B(B) = \mathbf{P}(B)$-t) az $\mathbf{f}_{\bar{E}\bar{A}JM}(B)$ kiadódott mátrixszal:

$$\mathbf{P}(B \mid j, m) = \alpha\, \mathbf{f}_B(B) \times \mathbf{f}_{\bar{E}\bar{A}JM}(B)$$

A 14.7. (a) feladatban azt kérjük, hogy ellenőrizze, helyes eredményt ad-e ez a folyamat.

A lépések sorozatát megvizsgálva láthatjuk, hogy két alapvető művelet szükséges: tényezőpárok pontonkénti összeszorzása és egy változó kiösszegzése tényezők szorzatából.

Created by XMLmind XSL-FO Converter.

A pontonkénti szorzás nem mátrixszorzás és nem is egy elemenkénti szorzás. Két, $\mathbf{f}_1$ és $\mathbf{f}_2$ tényezőnek a pontonkénti szorzata egy új $\mathbf{f}$ tényezőt eredményez, amelynek változóit az $\mathbf{f}_1$ és $\mathbf{f}_2$ változóinak az *uniója* adja. Tételezzük fel, hogy a két tényezőben az $Y_1, \ldots, Y_k$ változók közösek. Ekkor azt kapjuk, hogy

$$f(X_1 \ldots X_j, Y_1 \ldots Y_k, Z_1 \ldots Z_l) = f_1(X_1 \ldots X_j, Y_1 \ldots Y_k) f_2(Y_1 \ldots Y_k, Z_1 \ldots Z_l)$$

Ha minden változó bináris, akkor $\mathbf{f}_1$-nek és $\mathbf{f}_2$-nek 2^{j+k} és 2^{k+l} eleme van, pontonkénti szorzatuknak pedig 2^{j+k+l}. Például az $\mathbf{f}_1(A,B)$ és az $\mathbf{f}_2(B,C)$ tényezők esetén a lentebb mutatott valószínűség-eloszlással, az $\mathbf{f}_1 \times \mathbf{f}_2$ pontonkénti szorzata $\mathbf{f}_3(A, B, C)$-vel adott:

A	B	$\mathbf{f}_1(A, B)$	B	C	$\mathbf{f}_2(B, C)$	A	B	C	$\mathbf{f}_3(A, B, C)$
I	*I*	0,3	*I*	*I*	0,2	*I*	*I*	*I*	0,3 × 0,2
I	*H*	0,7	*I*	*H*	0,8	*I*	*I*	*H*	0,3 × 0,8
H	*I*	0,9	*H*	*I*	0,6	*I*	*H*	*I*	0,7 × 0,6
H	*H*	0,1	*H*	*H*	0,4	*I*	*H*	*H*	0,7 × 0,4
						H	*I*	*I*	0,9 × 0,2
						H	*I*	*H*	0,9 × 0,8
						H	*H*	*I*	0,1 × 0,6
						H	*H*	*H*	0,1 × 0,4

Egy változó kiösszegzése tényezők szorzataiból szintén egyértelmű számítás. Az egyetlen észreveendő trükk az, hogy bármely tényező, ami *nem* függ a kiösszegzendő változótól, az az összegzésen kívülre mozgatható. Például:

$$\sum_e \mathbf{f}_E(e) \times \mathbf{f}_A(A, B, e) \times \mathbf{f}_J(A) \times \mathbf{f}_M(A) = \mathbf{f}_J(A) \times \mathbf{f}_M(A) \times \sum_e \mathbf{f}_E(e) \times \mathbf{f}_A(A, B, e)$$

Most a pontonkénti szorzat kerül kiszámításra az összegzésen belül, a változót pedig kiösszegezzük a kiadódó mátrixból:

$$\mathbf{f}_J(A) \times \mathbf{f}_M(A) \times \sum_e \mathbf{f}_E(e) \times \mathbf{f}_A(A, B, e) = \mathbf{f}_J(A) \times \mathbf{f}_M(A) \times \mathbf{f}_{E,A}(A, B)$$

Vegyük észre, hogy mátrixokat *nem* szorzunk, ameddig nem kell kiösszegeznünk egy változót a kiadódó szorzatból. Annál a pontnál csak azokat a mátrixokat szorozzuk összse, amelyek magukban foglalják a kiösszegzendő változót. A pontonkénti szorzás és a kiösszegzés eljárások megléte esetén maga a változó eliminálás algoritmusa, amint azt a 14.10. ábra mutatja, igen egyszerűen felírható.

Nézzünk még egy lekérdezést: $P(JánosTelefonál \mid Betörés = igaz)$. Szokásos módon, az első lépés az egymásba ágyazott összegzések felírása:

$$P(J|b) = \alpha P(b) \sum_e P(e) \sum_a P(a|b,e) P(J|a) \sum_m P(m|a)$$

Ha ezt a kifejezést jobbról balra kiértékeljük, valami érdekeset veszünk észre: $\sum_m P(m|a)$ definíció szerint 1-gyel egyenlő. Így ezt eleve kihagyhattuk; a változó M a kérdésre *irreleváns*. Máshogy fogalmazva, a $P(JánosTelefonál \mid Betörés = igaz)$ lekérdezés eredményét nem változtatja meg a *MáriaTelefonál* eltávolítása a hálóból. Általában, bármely levélcsomópontot eltávolíthatunk, ami nem célváltozó vagy bizonyítékváltozó. Eltávolítás után lehetnek újabb levélcsomópontok, amelyek szintén irrelevánsak lehetnek.

Fontos

Ezt az eljárást folytatva végül szükségszerűen arra jutunk, hogy *minden változó, ami nem őse a célváltozónak vagy egy bizonyítékváltozónak, irreleváns a lekérdezésre*. A változó eliminálló algoritmus ezért az összes ilyen változót eltávolíthatja a lekérdezés kiértékelése előtt.

14.10. ábra - A változó eliminálás algoritmus Bayes-hálós lekérdezések megválaszolására

Created by XMLmind XSL-FO Converter.

```
function ELIMINÁLÁS-KÉRDEZÉS(X, e, bn) returns egy X feletti eloszlás
  inputs: X, a lekérdezés változója
          e, a bizonyíték mint esemény
          bn, egy P(X_1, ..., X_n) együttes eloszlást megadó Bayes-háló

  tényezők ← []; változók ← FORDÍT(VÁLTOZÓK[bn])
  for each változó in változók do
      tényezők ← [TÉNYEZŐ-LÉTREHOZÓ(változó, e)|változók]
      if változó rejtett then tényezők ← KIÖSSZEGEZ(változó, tényezők)
  return NORMALIZÁL(PONTONKÉNT-SZORZAT(tényezők))
```

4.3. Az egzakt következtetés komplexitása

Megmutattuk, hogy a változóeltávolítás hatékonyabb, mint a felsorolás, mivel elkerüli a számítások megismétlését (ahogyan az irreleváns változókat is kiejti). A változó eliminálás idő- és tárigényét az algoritmus működése alatt létrejövő legnagyobb tényező mérete befolyásolja a legerőteljesebben. Ezt viszont a változók eliminálásának sorrendje és a háló struktúrája határozza meg.

Fontos

A 14.2. ábra betörős hálója a hálóknak azon családjához tartozik, ahol a háló bármely két csomópontja között legfeljebb egyetlen irányítatlan út létezik. Ezeket **egyszeresen összekötött** (**singly connected**) hálóknak vagy **polifá**knak (**polytree**) nevezzük, amelyeknek van egy különösen kellemes tulajdonságuk: *az egzakt következtetés idő- és tárkomplexitása a polifákban a háló méretében lineáris*. Itt a méret az FVT-bejegyzések számával van definiálva; ha minden egyes csomópont szüleinek a száma egy konstanssal korlátozott, akkor a komplexitás a csomópontok számában is lineáris. Ezek az eredmények bármely – a háló topológiai sorrendjével konzisztens – sorrendezés esetén fennállnak (lásd 14.7. feladat).

Fontos

A **többszörösen összekötött** (**multiply connected**) hálóban (lásd például 14.11. (a) ábra) a változó eliminálás legrosszabb esetben exponenciális idő- és tárkomplexitású lehet, még akkor is, ha a csomópontonkénti szülők száma korlátos. Ez nem meglepő, figyelembe véve, hogy *a Bayes-hálókban való következtetés NP-nehéz, mivel ez speciális alesetként tartalmazza az ítéletlogikai következtetést is*. Valójában megmutatható (lásd 14.8. feladat), hogy a probléma ugyanannyira nehéz, mint kiszámítani egy ítéletlogikai formula esetén a formulát kielégítő értékadások *számát*. Ez azt jelenti, hogy ez #P-nehéz („számosság-P-nehéz”), azaz szigorúan nehezebb, mint az NP-teljes problémák.

Szoros kapcsolat áll fenn a Bayes-hálókban történő következtetés komplexitása és a kényszerkielégítési problémák (CSP) komplexitása között. Amint az 5. fejezetben tárgyaltuk, egy diszkrét CSP megoldásának nehézsége ahhoz kapcsolódik, hogy a kényszergráf mennyire „faszerű”. Olyan mértékek, mint a **hiperfaszélesség** (**hypertree width**), amelyek a CSP megoldási komplexitását határolják, közvetlenül alkalmazhatók Bayes-hálóknál is. Ezenfelül a változó eliminálás algoritmusa általánosítható úgy, hogy kényszerkielégítési problémákat is és Bayes-hálókat is megoldjon.

4.4. Csoportosító algoritmusok

A változó eliminálás algoritmusa egyszerű és hatékony egyedi lekérdezések megválaszolására. Azonban ha a háló összes változójának az a posteriori eloszlását szeretnénk kiszámítani, akkor kevésbé hatékony is lehet. Például egy polifa hálóban $O(n)$ egyenként $O(n)$ költségű lekérdezést kell kiadni, összességében $O(n^2)$ időköltséggel. **Csoportosító** (**clustering**) eljárásokkal (amit **egyesítési fa** – **join tree** – algoritmusnak is neveznek) ez az idő $O(n)$-re csökkenthető. Emiatt ezeket az algoritmusokat gyakran használják kereskedelmi Bayes-hálós eszközökben.

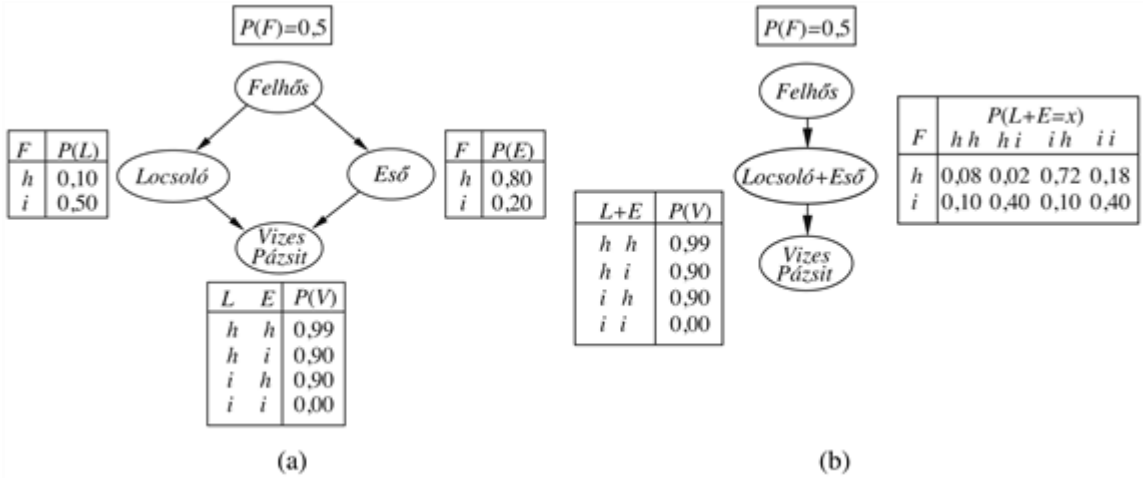
A csoportosítás alapötlete, hogy a háló önálló csomópontjait egyesítjük, klasztercsomópontokat formálva úgy, hogy a kiadódó háló polifa legyen. Például a 14.11. (a) ábrán mutatott, többszörösen összekötött háló egy polifává konvertálható a *Locsoló* és az *Eső* csomópontok egy *Locsoló+Eső*-nek nevezett **klasztercsomópont**ba

Created by XMLmind XSL-FO Converter.

(**cluster node**) való összevonásával, ahogy azt a 14.11. (b) ábra mutatja. A két bináris csomópontot egyetlen megacsomópont váltotta fel, ami négy lehetséges értéket vesz fel: *hh*, *hi*, *ih*, *ii*. A megacsomópontnak csak egy szülője van, a bináris *Felhős* változó, így két feltételes eset létezik.

A háló polifává konvertálása után, egy speciális célú következtetési algoritmust alkalmazunk. Alapvetően az algoritmus egyfajta kényszerterjesztés (lásd 5. fejezet), ahol a kényszerek biztosítják, hogy a szomszédos csoportokban megegyeznek a közös változók a posteriori eloszlásai. Gondos nyilvántartással ez az algoritmus képes $O(n)$ időben – ahol n most a módosított háló mérete – kiszámolni a háló összes nem bizonyíték csomópontjának a posteriori eloszlását. Azonban a probléma NP-nehéz volta nem tűnt el: ha a háló exponenciális idő- és tárigényű a változó eliminálás esetén, akkor a csoportosított hálóhoz tartozó FVT-k megkonstruálása exponenciális idő- és tárigényű lesz.

14.11. ábra - (a) Egy többszörösen összekötött háló feltételes valószínűségi táblákkal. (b) Egy többszörösen összekötött háló csoportosított ekvivalense.



5. Közelítő következtetés Bayes-hálókban

Az egzakt következtetés nagy, többszörösen összekötött hálókban való kezelhetetlensége miatt, elengedhetetlen közelítő következtetési módszerek átgondolása. Ez a fejezet véletlen mintavételi, **Monte Carlo**-módszereknek is nevezett, eljárásokat ír le, amelyek közelítő válaszokat nyújtanak, ahol a pontosság a generált minták számától függ.

Az utóbbi években a Monte Carlo-algoritmusok széles körben elterjedtek a számítástudományban olyan mennyiségek megbecslésére, amelyeket nehéz egzakt módon kiszámolni. Például a 4. fejezetben leírt szimulált lehűtés algoritmus egy Monte Carlo-eljárás optimalizációs problémákra. Ebben a fejezetben a mintavételezésnek az a posteriori valószínűségek kiszámításában történő alkalmazásában vagyunk érdekeltek. Algoritmusok két nagy családját írjuk le: a közvetlen mintavételezést és a Markov-láncos mintavételezést. Két másik megközelítést – a variációs módszert és a „hurkos” („loopy”) terjesztést – a fejezet végi megjegyzésekben említjük meg.

5.1. Közvetlen mintavételezési módszerek

Az alapelem minden mintavételi algoritmusban a minták generálása egy ismert valószínűség-eloszlásból. Például egy szabályos érme felfogható egy *Érme* valószínűségi változónak $\langle$ *fej*,*írás*$\rangle$ értékekkel és **P**(*Érme*) = $\langle$ 0,5, 0,5$\rangle$ a priori eloszlással. A mintavétel ebből az eloszlásból pontosan megfelel egy érme dobálásának: 0,5 valószínűséggel *fej*-et ad, és 0,5 valószínűséggel *írás*-t. Ha rendelkezésre áll a [0, 1] tartományba eső véletlen számoknak egy forrása, akkor bármely egyváltozós eloszlásból egyszerű dolog mintavételezni (lásd 14.9. feladat).

A véletlen mintavételezési folyamat legegyszerűbb fajtája Bayes-hálók esetén a hálóból generál olyan eseményeket, amelyekhez nem kapcsolódik bizonyíték. Az ötlet az, hogy mintavételezzünk minden változót egymás után, topológiai sorrendben. A valószínűség-eloszlás, amiből az értéket mintavételezzük, feltételesen

Created by XMLmind XSL-FO Converter.

függ a változó szüleihez már hozzárendelt értékektől. Ez az algoritmus látható a 14.12. ábrán. A működését a 14.11. (a) ábrán látható hálón szemléltetjük, feltételezve egy [*Felhős*, *Locsoló*, *Eső*, *VizesPázsit*] sorrendezést:

1. Sorsoljunk a **P**(*Felhős*) = ⟨ 0,5, 0,5⟩ eloszlásból; tegyük fel, hogy *igaz*-at kapunk.

2. Sorsoljunk a **P**(*Locsoló*| *Felhős* = *igaz*) = ⟨ 0,1, 0,9⟩ eloszlásból; tegyük fel, hogy *hamis*-at kapunk.

3. Sorsoljunk a **P**(*Eső*| *Felhős* = *igaz*) = ⟨ 0,8, 0,2⟩ eloszlásból; tegyük fel, hogy *igaz*-at kapunk.

4. Sorsoljunk a **P**(*VizesPázsit*| *Locsoló* = *hamis*, *Felhős* = *igaz*) = ⟨ 0,9, 0,1⟩ eloszlásból; tegyük fel, hogy *igaz*-at kapunk.

Ebben az esetben a `PRIOR-MINTA` a következő eseményt adja [*igaz*, *hamis*, *igaz*, *igaz*].

Könnyen látható, hogy a `PRIOR-MINTA` a háló által meghatározott a priori együttes eloszlásból generál mintákat. Először, legyen $S_{PS}(x_1, \ldots, x_n)$ annak a valószínűsége, hogy egy adott eseményt legenerál a `PRIOR-MINTA` algoritmus. *Pusztán a mintavételi folyamat alapján* felírható, hogy

$$S_{PS}(x_1 \ldots x_n) = \prod_{i=1}^{n} P(x_i \mid \text{szülők}(X_i))$$

mivel minden mintavételi lépés a szülő értékektől függ.

14.12. ábra - Egy mintavételező algoritmus, amely eseményeket generál egy Bayes-hálóból

function PRIOR-MINTA(*bn*) **returns** egy esemény a *bn* által definiált a priori eloszlásból
  inputs: *bn*, egy $\mathbf{P}(X_1, \ldots, X_n)$ együttes eloszlást megadó Bayes-háló

  $\mathbf{x} \leftarrow$ egy esemény n elemmel
  for $i = 1$ **to** n **do**
    $x_i \leftarrow$ egy véletlen minta $\mathbf{P}(X_i \mid \text{szülők}(X_i))$ szerint
  return x

Ez a kifejezés ismerős, mivel – ahogy ezt a (14.1) egyenlet kimondja – ez egyben a valószínűsége is az együttes eloszlás Bayes-hálós reprezentációja szerinti eseménynek is. Arra jutottunk tehát, hogy

$$S_{PS}(x_1 \ldots x_n) = P(x_1 \ldots x_n)$$

Ez az egyszerű tény nagyon könnyűvé teszi lekérdezések – minták felhasználásával történő – megválaszolását.

Bármilyen mintavételi algoritmusban a válasz kiszámítása a generálás során előálló minták megszámlálása alapján történik. Tételezzük fel, hogy N teljes mintánk van, és jelölje $N(x_1, \ldots, x_n)$ az $x_1, \ldots, x_n$ esemény gyakoriságát. Azt várjuk, hogy ez a gyakoriság határértékben konvergáljon a várható értékéhez a mintavételi valószínűség szerint:

$$\lim_{N \to \infty} \frac{N_{PS}(x_1, \ldots, x_n)}{N} = S_{PS}(x_1, \ldots, x_n) = P(x_1, \ldots, x_n) \qquad (14.4)$$

Például gondoljuk meg az előbb generált eseményt: [*igaz*, *hamis*, *igaz*, *igaz*]. Ennek az eseménynek a mintavételi valószínűsége:

$$S_{PS}(\textit{igaz}, \textit{hamis}, \textit{igaz}, \textit{igaz}) = 0{,}5 \times 0{,}9 \times 0{,}8 \times 0{,}9 = 0{,}324$$

Így, N nagy értékeinél azt várjuk, hogy a minták 32,4%-a ez az esemény legyen.

A következőkben, amikor a közelítő egyenlőség jelet („≈”) használjuk, pontosan ilyen értelemben használjuk – azaz a becsült valószínűség egzakttá válik a nagy mintaszámú határesetben. Az ilyen becsléseket **konzisztens**nek (**consistent**) nevezzük. Például bármely részlegesen meghatározott $x_1, \ldots, x_m$ esemény valószínűségének egy konzisztens becslése a következőképpen kapható meg, ha $m \leq n$:

Created by XMLmind XSL-FO Converter.

$$P(x_1, \ldots, x_n) \approx N_{PS}(x_1, \ldots, x_n)/N \quad (14.5)$$

Azaz, az esemény valószínűsége megbecsülhető a mintavételi folyamat által generált összes teljes esemény azon hányadával, amelyek illeszkednek a részlegesen meghatározott eseményre. Például ha generálunk 1000 mintát a locsolós hálóból, és 511-ben közülük az *Eső* = *igaz*, akkor az eső becsült valószínűsége $\hat{P}(Eső = igaz) 0,511$.

Elutasító mintavételezés Bayes-hálókban

Az **elutasító mintavételezés** (**rejection sampling**) általános módszer minták előállítására egy nehezen mintavételezhető eloszlásból, felhasználva egy könnyen mintavételezhető eloszlást. Legegyszerűbb formájában feltételes valószínűségek kiszámítására – azaz $P(X|\mathbf{e})$ meghatározására használható fel. Az ELUTASÍTÓ-MINTAVÉTELEZÉS algoritmusa a 14.13. ábrán látható. Először a háló által megadott a priori eloszlásból generál mintákat, majd elutasítja azokat, amelyek nem illeszkednek a bizonyítékhoz. Végül, a $\hat{P}(X = x|\mathbf{e})$ becslés megkapható az $X = x$ előfordulásainak megszámlálásával a megmaradt mintában.

Legyen $\hat{\mathbf{P}}(X|\mathbf{e})$ az algoritmus által kiadott becsült eloszlás. Az algoritmus definíciója szerint fennáll, hogy

$$\hat{\mathbf{P}}(X|\mathbf{e}) = \alpha \mathbf{N}_{PS}(X, \mathbf{e}) = \frac{\mathbf{N}_{PS}(X, \mathbf{e})}{N_{PS}(\mathbf{e})}$$

A 14.5 egyenletből kapjuk, hogy

$$\hat{\mathbf{P}}(X|\mathbf{e}) \approx \frac{\mathbf{P}(X, \mathbf{e})}{P(\mathbf{e})} = \mathbf{P}(X|\mathbf{e})$$

Azaz az elutasító mintavétel az igazi valószínűség konzisztens becslését adja.

14.13. ábra - Az elutasító mintavétel algoritmusa, ami Bayes-hálós lekérdezéseket válaszol meg bizonyítékok esetén

function ELUTASÍTÓ-MINTAVÉTELEZÉS(X, **e**, bn, N) **returns** $P(X|\mathbf{e})$ egy becslése
  inputs: X, a lekérdezés változója
    e, a bizonyíték mint esemény
    bn, egy Bayes-háló
    N, a generálandó minták száma
  local variables: **N**, X értékeihez tartozó számlálók vektora, kezdetben nulla

  for $j = 1$ **to** N **do**
    $\mathbf{x} \leftarrow$ PRIOR-MINTA(bn)
    if **x** konzisztens **e**-vel **then**
      $\mathbf{N}[x] \leftarrow \mathbf{N}[x] + 1$, ahol x az X értéke **x**-ben
  return NORMALIZÁL($\mathbf{N}[X]$)

Folytatva a 14.11. (a) ábra szerinti példánkat, tételezzük fel, hogy meg szeretnénk becsülni a $\mathbf{P}(Eső|\ Locsoló = igaz)$-t, 100 minta felhasználásával. A 100 általunk generált mintából, tegyük fel, hogy 73-ban teljesül, hogy *Locsoló* = *hamis*, és így elutasított, míg 27-ben fennáll, hogy *Locsoló* = *igaz*; a 27-ből 8-ban *Eső* = *igaz*, 19-ben pedig *Eső* = *hamis*. Így a

$$P(Eső|\ Locsoló = igaz) \approx \text{Normalizál}(\langle 8, 19 \rangle) = \langle 0,296, 0,704 \rangle$$

A helyes érték $\langle 0,3, 0,7 \rangle$. A begyűjtött minták szaporodásával a becslés konvergálni fog a helyes értékhez. A becslés szórása az egyes valószínűségeknél arányos lesz $1/\sqrt{n}$ -nel, ahol n a becsléshez felhasznált minták száma.

Az elutasító mintavétel legnagyobb hibája, hogy nagyon sok mintát utasít el! Az **e** bizonyítékkal konzisztens minták aránya exponenciálisan egyre kevesebb, ahogy a bizonyítékváltozók száma nő, így az eljárás egyszerűen használhatatlan komplex problémákban.

Created by XMLmind XSL-FO Converter.

Vegyük észre, hogy az elutasító mintavétel nagyon hasonló a feltételes valószínűségek becsléséhez, ha közvetlenül a valódi világból történik a becslés. Például a $P(Eső | VörösAzÉgEste = igaz)$ becslésénél egyszerűen megszámoljuk, milyen gyakran esik az után, hogy az előző este az ég vörös volt – figyelmen kívül hagyva azokat az estéket, amikor az ég nem volt vörös. (Itt a világ maga játssza a mintageneráló algoritmus szerepét.) Nyilvánvalóan, ha az ég nagyon ritkán vörös, akkor ez hosszú időt vehet igénybe, és ez az, ami az elutasító mintavétel gyengesége.

Valószínűségi súlyozás

A **valószínűségi súlyozás** (**likelihood weighting**) elkerüli az elutasító mintavételezés gyengeségét azáltal, hogy csak az **e** bizonyítékkal konzisztens eseményeket generál. Az algoritmus működésének a leírásával kezdjük; majd megmutatjuk, hogy helyesen működik – azaz, hogy konzisztens valószínűség becsléseket generál. A `VALÓSZÍNŰSÉGI-SÚLYOZÁS` (lásd 14.14. ábra) rögzíti az **E** bizonyítékváltozók értékeit, és csak a maradék X és **Y** változókat mintavételezi. Ez garantálja, hogy minden generált esemény konzisztens a bizonyítékkal. Azonban nem minden esemény egyenlő. Mielőtt megállapítanánk a számlálási eredményeket a célváltozó eloszlásában, minden eseményt súlyozunk azzal a valószínűséggel, amely megadja, hogy az esemény mennyire van összhangban a bizonyítékkal. Ezt a valószínűséget az egyes bizonyítékváltozók feltételes valószínűségeinek a szorzatával mérjük, a szüleik ismeretében. Szemléletesen, azoknak az eseményeknek, ahol a bizonyíték valószínűtlennek tűnik, kisebb súlyt kell adni.

Alkalmazzuk az algoritmust a 14.11. (a) ábrán látható háló esetén a $\mathbf{P}(Eső | Locsoló = igaz, VizesPázsit = igaz)$ kérdésre. A folyamat a következőképpen halad: először a w súlyt 1,0-ra állítjuk. Aztán generálunk egy eseményt:

1. Sorsoljunk a $\mathbf{P}(Felhős) = \langle 0{,}5, 0{,}5 \rangle$ eloszlásból; tegyük fel, hogy *igaz*-at kapunk.

2. A *Locsoló* egy bizonyítékváltozó *igaz* értékkel. Ezért beállítjuk, hogy

$$w \leftarrow w \times P(Locsoló = igaz | Felhős = igaz) = 0{,}1$$

3. Sorsoljunk a $\mathbf{P}(Eső | Felhős = igaz) = \langle 0{,}8, 0{,}2 \rangle$ eloszlásból; tegyük fel, hogy *igaz*-at kapunk.

3. A *VizesPázsit* egy bizonyítékváltozó *igaz* értékkel. Ezért beállítjuk, hogy

$$w \leftarrow w \times P(VizesPázsit = igaz | Locsoló = igaz, Eső = igaz) = 0{,}099$$

Itt a `SÚLYOZOTT-MINTA` az [*igaz*, *igaz*, *igaz*, *igaz*] eseményt adja ki 0,099 súllyal, és ezt az $Eső = igaz$ esetnél vesszük számításba. A súly alacsony, mivel az esemény egy felhős napot ír le, amikor valószínűtlen, hogy a locsoló be van kapcsolva.

Hogy megértsük, miért is működik a valószínűségi súlyozás, azzal kezdjük, hogy megvizsgáljuk a `SÚLYOZOTT-MINTA` S_{WS} mintavételi eloszlását. Emlékezzünk, hogy az **E** bizonyítékváltozók értéke (**e**) rögzített. A többi változót **Z**-vel jelöljük, azaz $\mathbf{Z} = \{X\} \cup \mathbf{Y}$. Az algoritmus **Z** minden változóját mintavételezi, szülei adott értékei mellett:

$$S_{WS}(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(z_i | szülők(Z_i)) \qquad (14.6)$$

Vegyük észre, hogy a $Szülők(Z_i)$ tartalmazhat mind rejtett, mind bizonyítékváltozókat is. A $P(\mathbf{z})$ a priori eloszlástól eltérően az S_{WS} eloszlás szentel valamennyi figyelmet a bizonyítéknak is: a mintavételezett értékeket minden Z_i esetén befolyásolják a Z_i ősei között levő bizonyítékok. Másfelől, S_{WS} kevesebb figyelmet szentel a bizonyítékoknak, mint a $P(\mathbf{z} | \mathbf{e})$ valódi a posteriori eloszlás, mivel a mintavételezett értékek minden Z_i esetén *figyelmen kívül hagyják* azokat a bizonyítékokat, amelyek Z_i-nek nem ősei.[8]

A w valószínűségi súly pótolja ki a különbséget a valódi és a kívánt mintavételi eloszlás között. Egy adott **z**-ből és **e**-ből álló **x** minta súlya az összes bizonyítékváltozó valószínűségének szorzata a szülei értékei mellett (amelyek közül néhány vagy akár az összes a Z_i-k között lehet):

[8] Ideálisan, a $P(\mathbf{z} | \mathbf{e})$ valódi a posteriori eloszlással megegyező mintavételi eloszlást szeretnénk használni, hogy az összes bizonyítékot figyelembe vegyük. Ez azonban nem tehető meg hatékonyan. Ha lehetne, akkor a kívánt valószínűséget tetszőleges pontossággal közelíthetnénk polinomiális számú mintával. Megmutatható, hogy nem létezhet ilyen polinom idejű közelítő séma.

Created by XMLmind XSL-FO Converter.

$$w(\mathbf{z},\mathbf{e}) = \prod_{i=1}^{m} P(e_i | szülők(E_i)) \qquad (14.7)$$

A (14.6) és (14.7) egyenleteket összeszorozva láthatjuk, hogy egy minta *súlyozott* valószínűsége különösen kényelmes alakú

$$S_{WS}(\mathbf{z},\mathbf{e})w(\mathbf{z},\mathbf{e}) = \prod_{i=1}^{l} P(z_i | szülők(Z_i)) \prod_{i=1}^{m} P(e_i | szülők(E_i)) = P(\mathbf{z},\mathbf{e}) \qquad (14.8)$$

mivel a két szorzat lefedi az összes változót a hálóban, lehetővé téve, hogy az együttes valószínűségre a (14.1) egyenletet használjuk.

14.14. ábra - A valószínűségi súlyozás algoritmusa Bayes-hálóban történő következtetéshez

function VALÓSZÍNŰSÉGI-SÚLYOZÁS(X, **e**, bn, N) **returns** $P(X|\mathbf{e})$ egy becslése
  inputs: X, a lekérdezés változója
    e, a bizonyíték mint esemény
    bn, egy Bayes-háló
    N, a generálandó minták száma
  local variables: **W**, X értékeihez tartozó súlyozott számlálók vektora, kezdetben nulla

  for $j = 1$ **to** N **do**
    x, $w \leftarrow$ SÚLYOZOTT-MINTA(bn)
    W[x] $\leftarrow$ **W**[x] + w, ahol x az X értéke **x**-ben
  return NORMALIZÁL(**W**[X])

---

function SÚLYOZOTT-MINTA(bn, **e**) **returns** egy esemény és egy súly

  x $\leftarrow$ egy esemény n elemmel; $w \leftarrow 1$
  for $i = 1$ **to** n **do**
    if X_i értéke szerepel **e**-ben
      then $w \leftarrow w \times P(X_i = x_i | szülők(X_i))$
      else $x_i \leftarrow$ egy véletlen minta $\mathbf{P}(X_i = x_i | szülők(X_i))$ szerint
  return **x**, w

Most már könnyen megmutatható, hogy a valószínűségi súlyozásos mintavétel konzisztens. Az X bármely konkrét x értékére a becsült a posteriori valószínűség a következőképpen számítható ki:

$$\hat{P}(x|\mathbf{e}) = \alpha \sum_{\mathbf{y}} N_{WS}(x,\mathbf{y},\mathbf{e}) w(x,\mathbf{y},\mathbf{e}) \qquad \text{a VALÓSZÍNŰSÉGI-SÚLYOZÁS-ból}$$

$$\approx \alpha' \sum_{\mathbf{y}} S_{WS}(x,\mathbf{y},\mathbf{e}) w(x,\mathbf{y},\mathbf{e}) \qquad \text{nagy } N\text{-ekre}$$

$$\approx \alpha' \sum_{\mathbf{y}} P(x,\mathbf{y},\mathbf{e}) \qquad \text{a (14.8) egyenlet miatt}$$

$$\alpha' P(x,\mathbf{e}) = P(x|\mathbf{e})$$

Így a valószínűségi súlyozás konzisztens becsléseket szolgáltat.

Mivel a valószínűségi súlyozás az összes generált mintát felhasználja, sokkal hatékonyabb lehet, mint az elutasításos mintavétel. Azonban a teljesítménye leromlik, amint a bizonyítékváltozók száma növekszik. Ez azért következik be, mert a legtöbb mintának nagyon kis súlya lesz, és így a súlyozott becslést főként a minták azon csekély töredéke határozza meg, amelyek egy elenyésző valószínűségnél jobban illeszkednek a bizonyítékokhoz. A probléma még erőteljesebben jelentkezik, ha a bizonyítékváltozók később fordulnak elő a változó sorrendben, mivel ekkor a minták olyan szimulációk, amelyek kevés hasonlóságot mutatnak a bizonyítékok által sugalmazott valósághoz.

Created by XMLmind XSL-FO Converter.

5.2. Következtetés Markov-lánc szimulációval

Ebben a fejezetben a **Markov lánc Monte Carlo** (**MCMC**, **Markov chain Monte Carlo**) algoritmust ismertetjük, hogy Bayes-hálókban következtethessünk. Először leírjuk, mit is csinál az algoritmus, majd elmagyarázzuk, hogy miért is működik, és miért van ilyen bonyolult neve.

Az MCMC algoritmus

Az előző két mintavételi algoritmustól eltérően, amelyek az egyes eseményeket a semmiből generálják, az MCMC minden eseményt az azt megelőző esemény véletlen módosításával generál. Ezért a hálót hasznos úgy elképzelni, mint aminek van egy konkrét *jelenlegi állapota,* ami minden változóra meghatároz egy értéket. A következő állapot generálása egy X_i nem bizonyítékváltozóhoz tartozó érték véletlenszerű mintavételezésével történik, *az X_i Markov-takarójába tartozó változók jelenlegi értékeinek feltétele mellett*. (Emlékezzünk vissza az 2.2. szakasz - Feltételes függetlenségi relációk Bayes-hálókban részre: egy változó Markov-takarója a szüleiből, gyermekeiből és gyermekei szüleiből áll.) Az MCMC így véletlen bolyongást végez az állapottérben – a lehetséges teljes érték hozzárendelések terében –, egyszerre egy változót billentve át, de rögzítetten tartva a bizonyítékváltozókat.

Gondoljuk meg a **P**(*Eső*| *Locsoló* = *igaz*, *VizesPázsit* = *igaz*) kérdést a 14.11. (a) ábrán látható háló esetén. A *Locsoló* és a *VizesPázsit* bizonyítékváltozók rögzítettek a megfigyelt értékeikre, míg a rejtett *Felhős* és *Eső* változók véletlenszerűen inicializáltak – mondjuk az egyik *igaz*-ra, a másik *hamis*-ra. Így a kezdeti állapot [*igaz*, *igaz*, *hamis*, *igaz*]. Ekkor a következő lépéseket hajtjuk végre ismétlődően:

1. A *Felhős*-t mintavételezzük a Markov-takarójába eső változók jelenlegi értékeinek ismeretében: ebben az esetben a **P**(*Felhős*| *Locsoló* = *igaz*, *Eső* = *hamis*) szerint mintavételezünk. (Hamarosan megmutatjuk, hogyan számítható ki ez az eloszlás.) Tegyük fel, hogy az eredmény *Felhős* = *hamis*. Ekkor az új állapot [*hamis*, *igaz*, *hamis*, *igaz*].

2. Az *Eső*-t mintavételezzük a Markov-takarójába eső változók jelenlegi értékeinek ismeretében: ebben az esetben a **P**(*Eső*| *Felhős* = *hamis*, *Locsoló* = *igaz*, *VizesPázsit* = *igaz*) szerint mintavételezünk. Tegyük fel, hogy ennek eredménye *Eső* = *igaz*. Ekkor az új állapot [*hamis*, *igaz*, *igaz*, *igaz*].

A folyamat során meglátogatott minden egyes állapot egy olyan minta, ami hozzájárul az *Eső* célváltozó megbecsléséhez. Ha a folyamat 20 állapotot látogatott meg, ahol az *Eső* igaz, és 61 állapotot, ahol az *Eső* hamis, akkor a kérdésre a választ a `NORMALIZÁL` (⟨ 20, 60⟩) = ⟨ 0,25, 0,75⟩ adja. A teljes algoritmus a 14.15. ábrán látható.

Miért működik az MCMC?

Fontos

Most megmutatjuk, hogy az MCMC konzisztens becsléseket szolgáltat az a posteriori valószínűségekre. A fejezet anyaga eléggé technikai jellegű, de az alapállítás egyszerű: *a mintavételi folyamat egy olyan „dinamikus egyensúlyban” állapodik meg, amelyben az egyes állapotokban töltött idő hosszú távon számolt hányada pontosan az a posteriori valószínűséggel arányos.* Ez a figyelemre méltó tulajdonság a speciális **átmenetvalószínűség** (**transition probability**) miatt áll fent, amely szerint a folyamat egyik állapotból a másikba lép át, ahogyan ezt a feltételes eloszlást a mintavételezett változó Markov-takarója meghatározza.

14.15. ábra - Az MCMC algoritmus Bayes-hálóban történő közelítő következtetéshez

Created by XMLmind XSL-FO Converter.

```
function MCMC-KÉRDEZÉS(X, e, bn, N) returns P(X|e) egy becslése
  local variables: N[X], X értékeihez tartozó számlálók vektora, kezdetben nulla
                   Z, a nem bizonyíték változók bn-ben
                   x, a háló jelenlegi állapota, kezdetben e-re állítva

  a Z-beli változók kezdeti értékét x-ben válasszuk meg véletlenszerűen
  for j = 1 to N do
      for each Z_i Z-ben do
          sorsoljuk Z_i értékét x-ben P(Z_i|mb(Z_i))-ből, ahol az MB(Z_i) értéke x szerinti
          N[x] ← N[x] + 1, ahol x az X értéke x-ben
  return NORMALIZÁL(N[X])
```

Legyen $q(\mathbf{x} \rightarrow \mathbf{x}')$ az a valószínűség, hogy a folyamat **x** állapotból **x′** állapotba lép át. Ez az átmenet-valószínűség definiálja az úgynevezett **Markov-lánc**ot (**Markov chain**) az állapottéren. (A Markov-láncok a 15. és 17. fejezetben is kiemelkedő szerepet játszanak.)

Tegyük fel most, hogy t lépésnyit futtatjuk a Markov-láncot, és legyen $\pi_t(\mathbf{x})$ annak a valószínűsége, hogy a rendszer a t időpillanatban az **x** állapotban van. Hasonlóan, legyen $\pi_{t+1}(\mathbf{x}')$ annak a valószínűsége, hogy a rendszer $t + 1$ időpillanatban **x′** állapotban van. Ismerve $\pi_t(\mathbf{x})$-et a $\pi_{t+1}(\mathbf{x}')$ kiszámítható úgy, hogy minden olyan állapotra, amiben a rendszer t időpillanatban lehet, összegezzük az állapot valószínűségének és az ebből az állapotból az **x′**-be való átlépés valószínűségének szorzatát:

$$\pi_{t+1}(\mathbf{x}') = \sum_{\mathbf{x}} \pi_t(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}')$$

Akkor mondjuk, hogy egy lánc elérte a **stacionárius eloszlás**át (**stationary distribution**), ha $\pi_t = \pi_{t+1}$. Jelöljük ezt a stacionárius eloszlást π-vel; az ezt definiáló egyenlet így

$$\pi(\mathbf{x}') = \sum_{\mathbf{x}} \pi_t(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') \text{ minden } \mathbf{x}'\text{-re} \qquad (14.9)$$

Ha a q átmenet-valószínűség eloszlás eleget tesz néhány alapvető feltevésnek,[9] akkor bármely q esetén pontosan egyetlen π eloszlás elégíti ki ezt az egyenletet.

A (14.9) egyenlet úgy tekinthető, mint ami azt állítja, hogy az egyes állapotokból a „kilépések” várható értéke (azaz a jelenlegi „populáció”) egyenlő az összes állapotbóli „belépések” várható értékével. Ez a kapcsolat nyilvánvalóan teljesíthető, ha a várható átlépések mindkét irányban ugyanakkorák bármely állapotpár esetén. Ez a tulajdonság a **teljes egyensúly** (**detailed balance**):

$$\pi(x)q(x \rightarrow x') = \pi(x')q(x' \rightarrow x) \text{ minden } x, x'\text{-re} \quad (14.10)$$

Megmutathatjuk, hogy a teljes egyensúly maga után vonja a stacionaritást, egyszerűen az **x**-ek felett összegezve a (14.10) egyenletben. Azt kapjuk, hogy

$$\sum_{\mathbf{x}} \pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}') q(\mathbf{x}' \rightarrow \mathbf{x}) = \pi(\mathbf{x}') \sum_{\mathbf{x}} q(\mathbf{x}' \rightarrow \mathbf{x}) = \pi(\mathbf{x}')$$

ahol az utolsó lépés abból következik, hogy **x′**-ből biztosan történik átmenet.

Most megmutatjuk, hogy az MCMC-KÉRDEZ mintavételi lépésében megadott $q(\mathbf{x} \rightarrow \mathbf{x}')$ átmenet-valószínűség eleget tesz a teljes egyensúly egyenletének, ahol a stacionárius eloszlás a $P(\mathbf{x}|\ \mathbf{e})$, a rejtett változók valódi a posteriori eloszlása. Ezt két lépésben teszszük meg. Először, egy Markov-láncot definiálunk, amelyben minden változót az *összes* többi változóval feltételesen mintavételezünk, és megmutatjuk, hogy ez eleget tesz a teljes egyensúly egyenletének. Majd egyszerűen megállapítjuk, hogy Bayes-hálóknál ez ekvivalens azzal a feltételes

[9] A q által definiált Markov-láncnak **ergodikus**nak (**ergodic**) kell lennie – azaz lényegében, minden állapotnak elérhetőnek kell lennie bármely másik állapotból, és nem lehetnek szigorúan periodikus ciklusok.

Created by XMLmind XSL-FO Converter.

mintavételezéssel, hogy a feltétel a változó Markov-takarója (lásd 2.2. szakasz - Feltételes függetlenségi relációk Bayes-hálókban részben).

Legyen X_i a mintavételezett változó, és jelölje $\bar{\mathbf{X}}_i$ az összes többi rejtett változót. A jelenlegi állapotban az értékük x_i és $\bar{\mathbf{x}}_i$. Ha az X_i-nek új x_i' értéket mintavételezünk feltételesen az összes többi változóval, beleértve a bizonyítékot is, akkor

$$q(\mathbf{x} \rightarrow \mathbf{x}') = q((x_i, \bar{\mathbf{x}}_i) \rightarrow (x_i', \bar{\mathbf{x}}_i)) = P(x' | \bar{\mathbf{x}}_i, \mathbf{e})$$

Ezt az átmenet-valószínűséget nevezik **Gibbs-mintavételező**nek (**Gibbs sampler**), ami az MCMC egy nagyon kényelmes alakja.

Most megmutatjuk, hogy a Gibbs-mintavételező teljes egyensúlyban van a helyes a posteriori eloszlása szerint:

$$\begin{aligned}
&\pi(\mathbf{x})q(\mathbf{x} \rightarrow \mathbf{x}') = P(\mathbf{x}|\mathbf{e})P(x_i' | \bar{\mathbf{x}}_i, \mathbf{e}) = P(x_i | \bar{\mathbf{x}}_i, \mathbf{e})P(x_i' | \bar{\mathbf{x}}_i, \mathbf{e}) \\
&= P(x_i | \bar{\mathbf{x}}_i, \mathbf{e})P(\bar{\mathbf{x}}_i | , \mathbf{e})P(x_i' | \bar{\mathbf{x}}_i, \mathbf{e}) \qquad \text{(alkalmazva a láncszabályt az első tagon)} \\
&= P(x_i | \bar{\mathbf{x}}_i, \mathbf{e})P(x_i' | \bar{\mathbf{x}}_i, \mathbf{e}) \qquad \text{(alkalmazva a láncszabály megfordítását)} \\
&= \pi(\mathbf{x}')q(\mathbf{x}' \rightarrow \mathbf{x})
\end{aligned}$$

Ahogy az 2.2. szakasz - Feltételes függetlenségi relációk Bayes-hálókban részben kimondtuk, egy változó független az összes többi változótól a Markov-takarójának a feltételében; így:

$$P(x_i' | \bar{\mathbf{x}}_i, \mathbf{e}) = P(x_i' | mb(X_i))$$

ahol $mb(X_i)$ jelöli az X_i Markov-takarójában, $MB(X_i)$-ben lévő változók értékeit. Amint a 14.10. feladat mutatja, egy változó valószínűsége a Markov-takarójának ismeretében arányos a változó szüleivel vett feltételes valószínűségének és az egyes gyermekek azok szüleivel vett feltételes valószínűségeinek a szorzatával:

$$P(x_i' | mb(X_i)) = \alpha P(x_i' | szülők(X_i)) \times \prod_{Y_j \in Gyermekei(X_i)} P(y_j | szülők(Y_j)) \qquad (14.11)$$

Így egy változó átléptetéséhez az X_i gyermekeinek számával megegyező számú szorzás szükséges.

Itt csupán az MCMC-nek egy egyszerű változatát tárgyaltuk, nevezetesen a Gibbs-mintavételezőt. Legáltalánosabb formájában az MCMC hatékony módszer valószínűségi modellekkel való számolásra, és számos változatát fejlesztették ki, közöttük a 4. fejezetben bemutatott szimulált lehűtés algoritmust, a 7. fejezetben a sztochasztikus kielégíthetőség algoritmust, valamint a 15. fejezetben a Metropolis–Hastings-mintavételezőt.

6. Elsőrendű reprezentációk valószínűségi kiterjesztése

Fontos

A 8. fejezetben kifejtettük az elsőrendű logika reprezentációs előnyeit az ítéletlogikához képest. Az elsőrendű logika objektumok és közöttük lévő relációk létezését veszi alapul, és képes tényeket kifejezni a tárgyterület *néhány* vagy *összes* objektumáról. Ez gyakran vezet olyan reprezentációkra, amelyek összehasonlíthatatlanul tömörebbek, mint az ekvivalens ítéletlogikai leírások. Márpedig a Bayes-hálók alapvetően az ítéletlogikához kapcsolódnak: a változók halmaza rögzített és véges, és minden változónak rögzített értékkészlete van. Ez a tény korlátozza a Bayes-hálók felhasználhatóságát. *Ha sikerül találnunk egy módszert arra, hogy az elsőrendű reprezentációk kifejezőerejét a valószínűség-számítással kombináljuk, várható, hogy ugrásszerűen megnöveljük a kezelhető problémák körét.*

Fontos

Created by XMLmind XSL-FO Converter.

Ennek a célnak az eléréséhez a következő alapvető felismerés szükséges: az ítéletlogika szemszögéből nézve a Bayes-háló elemi események feletti valószínűségeket határoz meg, amelyek mindegyike a hálózat minden változójára meghatároz egy értéket. Azaz, az ítéletlogika terminológiája szerint egy elemi esemény egy **modell** vagy egy **lehetséges világ**. Az elsőrendű felfogás szerint egy modell (az értelmezésével együtt) meghatározza az objektumok egy tárgykörét, a közöttük fennálló kapcsolatokat, és a tudásbázisbeli konstansok és predikátumok egy megfeleltetését a modellbeli objektumokra és kapcsolatokra. Ezért *egy elsőrendű valószínűségi tudásbázisnak az összes lehetséges elsőrendű modell valószínűségét definiálnia kell.* Legyen $\mu(M)$ a tudásbázis által az M modellhez rendelt valószínűség. Ekkor bármely ϕ elsőrendű állításnak a $P(\phi)$ valószínűségét a szokásos módon kapjuk, összegezve azon lehetséges világok felett, ahol ϕ igaz:

$$P(\phi) = \sum_{M:\phi \text{ igaz } M\text{-ben}} \mu(M) \qquad (14.12)$$

Eddig rendben is volna. Azonban van egy probléma: az elsőrendű modellek halmaza végtelen. Ez azt jelenti, hogy (1) az összegzés kivitelezhetetlen, és (2) egy teljes, konzisztens eloszlás meghatározása a világok egy végtelen halmazán igen bonyolult lehet.

Fogadjunk el ezért bizonyos korlátozásokat, legalábbis időlegesen. Nevezetesen, hogy tervezzünk egy olyan korlátozott nyelvet, amelyhez csak véges modell tartozik. Ez számos módon megtehető. Itt a **relációs valószínűségi model**leket (**relational probability model**s, **RPM**) vagy **RVM**-eket ismertetjük, amelyek a szemantikus hálók (lásd 10. fejezet) és objektumrelációs adatbázisok elméletéből is vettek át elemeket. Egyéb megközelítések az irodalmi és történeti megjegyzésekben szerepelnek.

Az RVM-ek megengednek konstans szimbólumokat, amelyek objektumokat neveznek meg. Például legyen *KovácsProf* egy professzor neve, *János* pedig egy diák neve. Minden objektum az osztály egy példánya: például *KovácsProf* egy *Professzor* és *János* egy *Diák*. Feltesszük, hogy minden konstans szimbólum osztálya ismert.

Függvényszimbólumaink kétfelé lesznek osztva. Az első fajtába tartozók, az **egyszerű függvény**ek (**simple function**s), egy objektumot nem egy másik strukturált objektumra képeznek le, hanem egy értékre egy rögzített értéktartományból, pontosan úgy, ahogy egy valószínűségi változónál. Például az *Intelligencia*(*János*) és a *Finanszírozás*(*KovácsProf*) lehet *magas* vagy *alacsony*; a *Siker*(*János*) és a *Hírnév*(*KovácsProf*) lehet *igaz* vagy *hamis*. Függvényszimbólumok nem alkalmazhatók olyan értékekre, mint az *igaz* vagy a *hamis*, így nem lehetséges egyszerű függvények egymásba ágyazása. Ezzel elkerüljük a végtelenség egyik forrását. Egy adott objektumon alkalmazott egyszerű függvény értéke lehet megfigyelt vagy ismeretlen; reprezentációnkban ezek lesznek az elemi valószínűségi változók.[10]

Megengedünk emellett **komplex függvény**eket (**complex function**s) is, amelyek objektumokat képeznek le más objektumokra. Például a *Konzulens*(*János*) lehet *KovácsProf*. Minden komplex függvénynek van egy megadott értéktartománya és értékkészlete, amik osztályok. Például a *Konzulens* értéktartománya a *Diák* és értékkészlete a *Professzor*. A függvények csak helyes osztályokra alkalmazhatók; például a *KovácsProf Konzulens*-e nem definiált. Komplex függvények egymásba ágyazhatók: a *TanszékVezető*(*Konzulens*(*János*)) lehet *MórProf*. Egyelőre feltesszük, hogy minden komplex függvény értéke ismert az összes konstans szimbólumra. Mivel a *TB* véges, ez maga után vonja, hogy komplex függvények alkalmazásának minden láncolata elvezet a véges számú objektum egyikéhez.[11]

Az utolsó elem, amire szükségünk van, az a valószínűségi információ. Minden egyszerű függvényhez specifikáljuk a szülők egy halmazát, csakúgy, mint a Bayes-hálóknál. A szülők lehetnek ugyanannak az objektumnak más egyszerű függvényei; például egy *Professzor*-nak a *Finanszírozás*-a függhet a *Hírnév*-től. A szülők lehetnek még *kapcsolódó* objektumok egyszerű függvényei – például egy diák *Sikere* függhet a diák *Intelligenciá*-jától és a diák konzulensének *Hírnév*-től. Ezek valóban *univerzálisan kvantifikált* állítások egy osztály minden objektumának szüleiről. Így írhatjuk azt, hogy

$$\forall x \; x \in Diák \Rightarrow Szülők(Siker(x)) = \{Intelligencia(x), \; Hírnév(Konzulens(x))\}$$

(Kevésbé formálisan a 14.16. (a) ábrához hasonló diagrammokat rajzolhatunk.) Most megadjuk a gyermek feltételes valószínűség eloszlását, ahol a feltételt a szülei jelentik. Például mondhatjuk, hogy:

[10] Nagyon hasonló szerepet töltenek be, mint a változómentes atomi mondatok, amelyek a 9.1. alfejezetben leírt ítéletlogikai állításokra való visszavezetési folyamatban generálódnak.

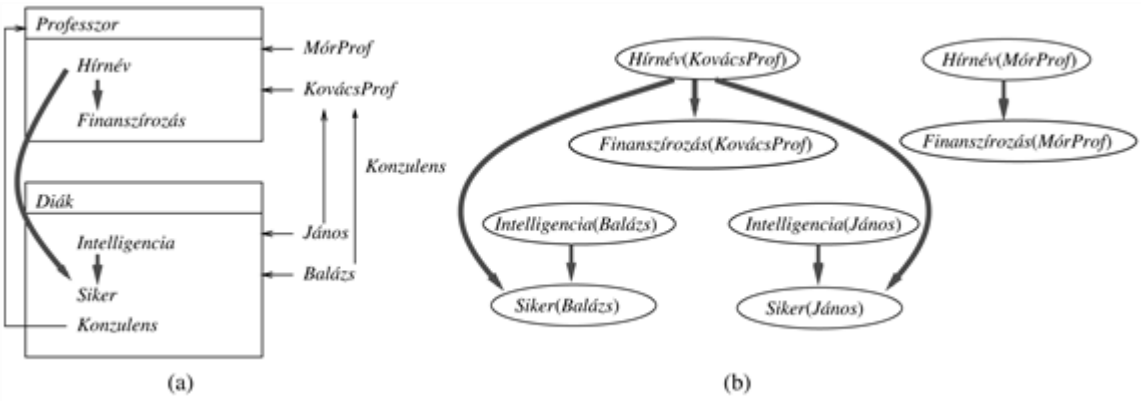
[11] Ez a megkötés azt jelenti, hogy nem használhatunk olyan komplex függvényeket, mint *Apa* vagy *Anya*, amelyek potenciálisan végtelen láncokat eredményezhetnének egy ismeretlen objektummal végződve. Ezt a megkötést később újra meggondoljuk.

Created by XMLmind XSL-FO Converter.

$\forall x \; x \in Diák \Rightarrow P(Siker(x) = igaz \mid \{Intelligencia(x) = magas, Hírnév(Konzulens(x)) = igaz) = 0,95$

Csakúgy, mint a szemantikus hálókban, csatolhatunk feltételes eloszlásokat magához az osztályhoz is, így a példányok **öröklik** (**inherit**) a függéseket és a feltételes valószínűségeket az osztálytól.

14.16. ábra - (a) Egy két osztályt (*Professzor* és *Diák*) leíró RVM. Két professzor és két diák van, mindkét diák konzulense *KovácsProf*. (b) Az (a)-beli RVM Bayes-háló ekvivalense.



Az RVM nyelv szemantikája felteszi, hogy minden konstans szimbólum egy különálló objektumra hivatkozik – az **egyedi név feltételezés**t (**unique names assumption**) a 10. fejezetben írtuk le. Ezen feltevés mellett és a korábban felsorolt megkötésekkel, meg lehet mutatni, hogy minden RVM valószínűségi változók egy rögzített, véges halmazát generálja, amelyek mindegyike egy egyszerű függvény egy konstans szimbólumra alkalmazva. Ekkor – feltéve, hogy a szülő-gyermek függések *körmentesek* – megkonstruálhatunk egy ekvivalens Bayes-hálót. Azaz az RVM és a Bayes-háló azonos valószínűségeket határoz meg minden lehetséges világra. A 14.6. (b) ábra mutatja azt a Bayes-hálót, ami a 14.16. (a) ábra RVM-jéhez tartozik. Vegyük észre, hogy az RVM-ben szereplő *Konzulens* kapcsolatok nincsenek jelen a Bayes-hálóban. Ennek oka, hogy rögzítettek és ismertek. Azonban *közvetetten* feltűnnek a háló topológiájában; például a *Siker*(*János*) szülője a *Hírnév*(*KovácsProf*), mivel a *Konzulens*(*János*) értéke a *KovácsProf*. Általában, az objektumok között fennálló relációk meghatározzák az ezen objektumok tulajdonságai között fennálló függőségeket.

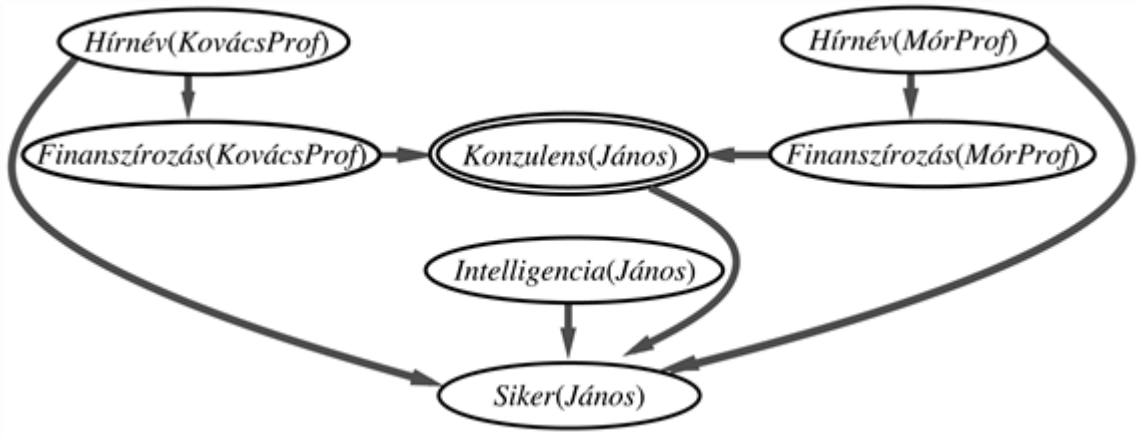
Az RVM-ek kifejezési erejének megnövelésére számos módszer kínálkozik. Megengedhetünk **rekurzív függés**eket (**recursive dependencies**) a változók között, hogy bizonyosfajta visszatérő kapcsolatokat kezelni tudjunk. Például tegyük fel, hogy a gyorséttermi ételektől való függőséget egy *McGén* okozza. Ekkor minden *x*-re *McGén*(*x*) függ a *McGén*(*Apa*(*x*)) és *McGén*(*Anya*(*x*))-től, amelyek viszont függnek a *McGén*(*Apa*(*Apa*(*x*))) és a *McGén*(*Anya*(*Apa*(*x*)))-től és így tovább. Bár az ilyen tudásbázisok végtelen változót tartalmazó Bayes-hálókhoz tartoznak, fixpont-egyenletekből olykor megoldásokhoz juthatunk. Például a kiszámítható a *McGén* egyensúlyi eloszlása, az öröklődés adott feltételes valószínűsége mellett. A rekurzív tudásbázisok egy másik igen fontos családja a 15. fejezetben leírt **időbeli valószínűségi model**leket (**temporal probability model**s) tartalmazza. Ezekben a modellekben a t időpillanatbeli állapot tulajdonságai a $t - 1$ időpillanatbeli állapot tulajdonságaitól függnek és így tovább.

Az RVM-ek szintén kiterjeszthetők, hogy megengedjenek **kapcsolati bizonytalanság**ot (**relational uncertainty**) is – azaz bizonytalanságot a komplex függvények értékeinél. Például lehet, hogy nem tudjuk, kicsoda a *Konzulens*(*János*). Ekkor a *Konzulens*(*János*) egy valószínűségi változó lesz, aminek lehetséges értékei *KovácsProf* és *MórProf*. A megfelelő hálót a 14.17. ábra mutatja.

Szintén jelen lehet **azonossági bizonytalanság** (**identity uncertainty**); például esetleg nem tudjuk, hogy *Mária* és *KovácsProf* ugyanaz a személy-e. Azonossági bizonytalanság esetén az objektumok és az állítások száma változhat a lehetséges világokban. Annak a világnak, ahol *Mária* és *KovácsProf* ugyanaz a személy eggyel kevesebb objektuma van, mint annak a világnak, ahol ők különböző személyek. Ez a következtetési eljárást bonyolultabbá teszi, de a (14.12) egyenlet által lefektetett alapelv érvényben marad: bármely kijelentés valószínűsége jól definiált és kiszámítható. Az azonossági bizonytalanság különösen fontos robotok és beágyazott érzékelőrendszerek esetén, amelyeknek több objektumot is követniük kell. Erre a problémára a 15. fejezetben viszszatérünk.

Created by XMLmind XSL-FO Converter.

14.17. ábra - Egy Bayes-háló részlete, ami egy olyan RVM-hez tartozik, amelyben a *Konzulens*(*János*) ismeretlen, de vagy *KovácsProf*, vagy *MórProf*. A konzulens választása az egyes professzorok finanszírozásától függ. Vegyük észre, hogy a *Siker*(*János*) most *mind a két* professzor *Hírnev*-étől függ, bár a *Konzulens*(*János*) értéke meghatározza, hogy valójában melyiknek is van hatása.



Vizsgáljuk meg most a következtetés kérdését. Világos, hogy a következtetés elvégezhető az ekvivalens Bayes-hálóban, feltéve, hogy az RVM nyelvet úgy korlátozzuk, hogy az ekvivalens háló véges és rögzített struktúrájú. Ez analóg azzal a módszerrel, ahogy elsőrendű logikai következtetést végezhetünk el ítéletlogikai következtetéssel, az ekvivalens ítéletlogikai tudásbázisban (lásd 9.1. alfejezet). Ahogy a logikai esetben is, az ekvivalens háló túlságosan nagy ahhoz, hogy megkonstruáljuk, különösen ahhoz, hogy kiértékeljük. A sűrű összekötöttség szintén probléma (lásd 14.12. feladat). A közelítő algoritmusok, mint az MCMC (lásd 14.5. alfejezet), ezért igen hasznosak RVM-kben való következtetésre.

Amikor az MCMC-t egy egyszerű RVM-tudásbázisának az ekvivalens Bayes-hálójára alkalmazzuk, ahol a tudásbázis nem tartalmaz kapcsolati vagy azonossági bizonytalanságot, az algoritmus a lehetséges világok teréből mintavételez, amit az objektumok egyszerű függvényeinek értékei határoznak meg. Világosan látható, hogy ez a megközelítés kiterjeszthető a kapcsolati és az azonossági bizonytalanságok kezelésére is. Ebben az esetben a lehetséges világok közötti átmenet lehet, hogy egy egyszerű függvény értékét vagy egy komplex függvényt változtat meg, ami a függési struktúra megváltozásához is vezethet. Az átmenetek szintén megváltoztathatják a konstans szimbólumok közötti azonossági relációkat. Így az MCMC elegáns módszernek tűnik következtetések elvégzésére igen nagy kifejezőerejű, elsőrendű valószínűségi tudásbázisokban.

A kutatás ezen a területen még a kezdeteknél tart, de az egyre nyilvánvalóbb, hogy az elsőrendű valószínűségi érvelés az MI-rendszerek óriási hatékonyságnövekedését eredményezi a bizonytalanság kezelése kapcsán. A potenciális alkalmazások között jelen van a számítógépes látás, a számítógépes nyelvészet, az információ-visszakeresés és a szituációértékelés. Ezen területek mindegyikén az objektumok halmaza – és így valószínűségi változók halmaza – nem ismert előre, így a tisztán „kijelentéseken” alapuló módszerek, mint például a Bayes-hálók, nem képesek a helyzetet teljesen leírni. Ezen módszereket kiegészítették a modellek tere feletti kereséssel, de az RVM-ek egyetlen modellben teszik lehetővé a következtetést ilyen típusú bizonytalanság esetén is.

7. Egyéb módszerek a bizonytalan környezetben történő következtetéshez

Más tudományágak (például fizika, genetika, közgazdaságtan) régóta a valószínűségszámítást részesítették előnyben a bizonytalanság modellezésére.[12] Pierre Laplace 1819-es megfogalmazásában „A valószínűség-számítás semmi egyéb, mint a józan ész számokban megfogalmazva”. James Maxwell 1850-ben tett kijelentése szerint „a világ igazi logikája a valószínűségek kalkulusa, ami figyelembe veszi a valószínűség nagyságát, ahogy azt minden gondolkodó ember megteszi vagy meg kellene tennie”.

[12] A bizonytalan ismereteket és/vagy bizonytalan következtetési formákat tartalmazó következtetéseket a továbbiakban egységesen „bizonytalansági következtetéseknek” nevezzük. (*A ford.*)

Created by XMLmind XSL-FO Converter.

Ezt a régi tradíciót látva talán meglepő, hogy az MI a valószínűség-számítás helyett számos alternatívával kísérletezett. Az 1970-es évekbeli első szakértői rendszerek teljesen figyelmen kívül hagyták a bizonytalanságot, és szigorúan logikai következtetést használtak, de hamar kiderült, hogy ez a legtöbb valós problémánál nem alkalmazható. A szakértői rendszerek következő generációja (különösen az orvosi tárgytartományokban) valószínűségi technikákat használt. A kezdeti eredmények biztatók voltak, de nem voltak skálázhatók nagyobb problémákra a teljes együttes valószínűség-eloszlás megadásához szükséges exponenciális számú érték miatt. (Hatékony Bayes-hálós algoritmusok ekkor még nem voltak ismertek.) Ennek eredményeképpen a valószínűségi megközelítések elvesztették vonzerejüket 1975 és 1988 között, és számos egyéb alternatívát próbáltak ki:

- Egy elterjedt nézet szerint a valószínűség-számítás alapvetően numerikus jellegű, míg az emberi ítélethozatal sokkal „kvalitatívabb”. Az bizonyos, hogy nem vagyunk tudatában, hogy különböző fokú bizonyossággal végzünk numerikus számításokat. (Ahogyan a logikai egyesítés elvégzésének sem vagyunk tudatában, mégis úgy tűnik, hogy képesek vagyunk valamilyenfajta logikai érvelésre.) Lehetséges, hogy a bizonyosság valamilyen numerikus fokát közvetlenül tárolják az idegsejtek kapcsolatainak és aktivációinak erősségei. Ebben az esetben ezen erősségek tudatos elérésének a nehézsége nem meglepő. Azt is el kell ismerni, hogy a kvalitatív következtetési mechanizmus a valószínűség-számítási alapokra is felépíthető, így a „nincsenek számok” érv a valószínűségek ellen nem meggyőző. Mindazonáltal néhány kvalitatív módszer igen hasznosnak tűnhet a sajátos tulajdonságaik miatt. Az egyik legjobban tanulmányozott módszer az **alapértelmezéseken alapuló következtetés** (**default reasoning**), ami nem úgy kezeli a következtetetéseket, mint amikben „bizonyos fokig hiszünk”, hanem mint „amikben addig hiszünk, amíg nincs jobb indok valami mást hinni”. Az alapértelmezéseken alapuló következtetés a 10. fejezetben szerepel.

- **Szabályalapú** (**rule-based**) megközelítésekkel is megpróbáltak bizonytalanságot kezelni. Ezek a módszerek a logikai szabályrendszerek sikereit akarták felhasználni, de hozzáadva minden szabályhoz valamilyen „kiagyalt” *ad hoc* tényezőt, ami a bizonytalanságot kezeli. Ezeket az eljárásokat az 1970-es évek közepén fejlesztették ki, és igen nagy számú szakértői rendszer alapjául szolgáltak, különösen orvosi és egyéb területeken.

- Az **ismerethiány** (**ignorance**) (ami a bizonytalansággal bizonyos értelemben szembeállítható) kérdéseit eddig még nem érintettük. Fontoljuk meg egy érme feldobását. Ha tudjuk, hogy az érme szabályos, akkor ésszerűnek tűnik, hogy a fej valószínűsége 0,5. Ha tudjuk, hogy az érme szabálytalan, de nem tudjuk, milyen módon, akkor 0,5 az egyetlen ésszerűnek tűnő valószínűség. Nyilvánvaló, hogy a két eset különböző, de a valószínűségük láthatóan nem különbözteti meg őket. A **Dempster– Shafer-elmélet** (**Dempster–Shafer theory**) **intervallumérték**eket (**interval-valued**) használ a bizonyosság fokozataira, így reprezentálja az ágens tudását egy állítás valószínűségéről. Más, másodrendű valószínűségeket használó módszereket is tárgyalunk majd.

- A valószínűség-számítás ugyanazt a létélméleti feltevést használja, mint a logika: az események vagy igazak, vagy hamisak a világban, még ha az ágens bizonytalan is, melyik is áll fenn. A **fuzzy logika** (**fuzzy logic**) kutatói egy másik ontológiát javasoltak, ami megengedi a **meghatározatlanság**ot (**vagueness**) is: azt, hogy egy esemény lehet „valamennyire” igaz. A meghatározatlanság és a bizonytalanság valójában ortogonális témakörök, ahogy azt látni fogjuk.

A következő három alfejezet ezen megközelítések mindegyikét egy kicsit részletesebben mutatja be. Nem tartalmaznak részletes technikai leírásokat, de a megadott hivatkozások segítséget nyújtanak további részletek eléréséhez.

7.1. Bizonytalansági következtetés szabályalapú eljárásokkal

A szabályalapú rendszerek a logikai következtés gyakorlati és szemléletes rendszerein való korai munkákból alakultak ki. A logikai rendszerek általában, és a logikai szabályalapú rendszerek speciálisan, rendelkeznek az alábbi három tulajdonsággal:

- **Lokalitás** (**locality**)**:** logikai rendszerekben, ha van $A \Rightarrow B$ formájú szabály, akkor a B kikövetkeztethető, ha az A bizonyíték adott, *bármely más szabályra való tekintet nélkül*. Valószínűségi rendszerekben a Markov-takaróban rendelkezésre álló *összes* tényt figyelembe kell venni.

- **Leválasztás** (**detachment**): ha egyszer találtunk egy logikai bizonyítást B állítás kikövetkeztetéséhez, az állítást tetszés szerint felhasználhatjuk, a származtatásától függetlenül. Vagyis **leválasztható** (**detached**) a

Created by XMLmind XSL-FO Converter.

megokolásaitól. Valószínűségek esetén azonban a bizonyosság bizonyítékának a forrása fontos az elkövetkező következtetésekben.

- **Igazságfüggvény** (**truth-functionality**): a logikában az összetett kifejezések igazságértéke kiszámítható az alkotóinak igazságértékéből. Valószínűségek kombinálása nem lehetséges ezen a módon, csak szigorú, teljes függetlenségi feltételezések mellett.

Számos kísérlet történt olyan bizonytalansági következtetések kifejlesztésére, ahol ezek az előnyös tulajdonságok megmaradnak. Az elképzelés szerint a bizonyosság mértékét hozzárendeljük állításokhoz és szabályokhoz, és kifejlesztünk tisztán lokális sémákat a bizonyosságmértékek kombinálására és terjesztésére. A sémák egyben igazságfüggvények, például az $A \vee B$ bizonyosságmértéke, az A- és B-beli bizonyosságok függvénye.

Fontos

A rossz hír a szabályalapú rendszerek számára, hogy a *lokalitás, a leválasztás és az igazságfüggvény egyszerűen nem alkalmas a bizonytalansági következtetés számára.* Vizsgáljuk először az igazságfüggvényt. Legyen F_1 az az esemény, hogy fejet kapunk egy szabályos érme feldobásakor, legyen I_1 az az esemény, hogy ugyanekkor írást kapunk, és legyen F_2 az az esemény, hogy fejet kapunk egy szabályos érme második feldobásakor. Világos, hogy mindhárom esemény valószínűsége 0,5, és egy igazságtartó rendszernek ugyanazt az értéket kell hozzárendelnie bármely kettő konjunkciójához. De láthatjuk, hogy a konjunkció valószínűsége függ maguktól az eseményektől is, nem csak a valószínűségektől.

$P(A)$	$P(B)$	$P(A \vee B)$
$P(F_1) = 0{,}5$	$P(F_1) = 0{,}5$ $P(I_1) = 0{,}5$ $P(F_2) = 0{,}5$	$P(F_1 \vee F_1) = 0{,}50$ $P(F_1 \vee I_1) = 1{,}00$ $P(F_1 \vee F_2) = 0{,}75$

A helyzet még rosszabb, ha bizonyításokat láncolunk. Az igazságfüggvényeken alapuló rendszerekben léteznek $A \mapsto B$ alakú **szabály**ok (**rule**s), amelyek lehetővé teszik, hogy kiszámítsuk B-beli bizonyosságunk értékét, mint annak függvényét, hogy mennyire hiszünk a szabályban, és mennyire hiszünk A-ban. Ekkor természetesen mind előre-, mind hátrahaladó rendszereket is ki lehet fejleszteni. A szabálybeli bizonyosságunkat konstansnak tételezzük fel, amit általában a tudásmérnök határoz meg, például $A \mapsto_{0,9} B$.

Gondoljuk a 14.11. (a) ábrán látható vizes-pázsit szituációra. Ha képesek szeretnénk lenni mind okozati, mind diagnosztikai következtetés elvégzésére, a két alábbi szabályra lesz szükségünk:

```
Eső ↦VizesPázsit és VizesPázsit ↦ Eső
```

Ha nem vigyázunk, ez a két szabály visszacsatolásos módon fog működni úgy, hogy az *Eső*-re vonatkozó bizonyíték megnöveli a *VizesPázsit*-hoz tartozó bizonyosságot, ami viszont még inkább megnöveli az *Eső*-höz tartozó bizonyosságot. Nyilvánvaló, hogy bizonytalansági következtető rendszereknek nyilván kell tartaniuk, hogy milyen utakon terjedt a bizonyíték.

Az okok közötti következtetés (vagy kimagyarázás) szintén bonyodalmas. Gondoljuk meg, mi is történik, ha a következő két szabályunk van:

```
Locsoló ↦ VizesPázsit és VizesPázsit ↦ Eső
```

Tételezzük fel, hogy látjuk a locsolót bekapcsolva. Előreláncolással a szabályainkon keresztül ez megnöveli azt a bizonyosságot, hogy a pázsit vizes, ami viszont megnöveli az esőhöz tartozó bizonyosságot. De ez képtelenség: a locsoló bekapcsolt állapota megmagyarázza a pázsit vizességét, így *csökkentenie* kell az esőhöz tartozó bizonyosságot. Egy igazságfüggvényen alapuló rendszer úgy viselkedik, mintha a *Locsoló*↦*Eső* szabályban is hinne.

Created by XMLmind XSL-FO Converter.

Ezen nehézségek mellett hogyan lehetséges, hogy az igazságfüggvényen alapuló rendszereket valaha is hasznosnak tartották? A megoldás abban rejlik, hogy korlátozták a feladatokat körét, és a szabálybázist gondosan úgy tervezték meg, hogy nemkívánatos kölcsönhatások ne következzenek be. A leghíresebb igazságfüggvényen alapuló rendszer bizonytalansági következtetésre a **bizonyossági tényező** (**certainty factor**) modell, amit a MYCIN orvosi diagnosztikai program számára fejlesztettek ki, és széles körben használták szakértői rendszerekben az 1970-es és az 1980-as években. A bizonyossági tényezők majdnem minden alkalmazása vagy tisztán diagnosztikai (mint a MYCIN), vagy tisztán okozati szabálybázist tartalmazott. Továbbá a tények csak a szabálybázis „gyökereinél" léptek be, és a legtöbb szabálybázis egyszeresen összekötött volt. Heckerman megmutatta, hogy ezen feltételek mellett a bizonyossági tényezőkön alapuló következtetés kicsit eltérő változata ekvivalens a polifabeli Bayes-következtetéssel (Heckerman, 1986). Más körülmények között, a bizonyossági tényezők katasztrofálisan helytelen bizonyosságot eredményezhetnek a tények túlzott figyelembevétele miatt. Amint a szabályhalmaz naggyá válik, a szabályok között a nemkívánatos interakciók egyre gyakoribbak, és a felhasználók azt találták, hogy számos szabálynak kell „hangolni" a bizonyossági faktorát, amikor új szabályokkal bővítenek. Szükségtelen mondani, hogy ez a megközelítés már nem ajánlott.

7.2. Az ismerethiány reprezentálása: a Dempster–Shafer-elmélet

A **Dempster–Shafer-elmélet**et (**Dempster–Shafer theory**) azzal a céllal hozták létre, hogy megkülönböztethetővé váljon a **bizonytalanság** (**uncertainty**) és az **ismerethiány** (**ignorance**). Ez az elmélet egy állításnak nem a valószínűségét számítja ki, hanem helyette azt, hogy mennyi annak a valószínűsége, hogy a bizonyíték támogatja az állítást. A bizonyosságnak ezt a mértékét **bizonyosságfüggvény**nek (**belief function**) nevezik, és *Bel*(*X*)-szel jelölik.

Most visszatérünk a pénzérmés példánkhoz, a bizonyosságfüggvény bemutatásához. Tételezzük fel, hogy egy rossz kinézetű alak odamegy önhöz és felajánlja, hogy fogadjon 1000 forintba, hogy a következő esetben fej lesz az eredmény. Mivel az érme lehet szabályos, de lehet szabálytalan is, milyen bizonyosságot kell ahhoz az eseményhez rendelni, hogy a következő eredmény fej lesz? A Dempster–Shafer-elmélet szerint mivel nincs bizonyíték egyik esetre sem, így azt kell mondani, hogy a bizonyosság $Bel(Fej) = 0$ és $Bel(\neg Fej) = 0$. Ez szkeptikussá teszi a Dempster–Shafer-elméletet használó következtető rendszereket, aminek van egy intuitív vonzereje. Most tegyük fel, hogy rendelkezésre áll egy szakértő, aki 90%-os bizonyossággal tanúsítja, hogy az érme szabályos (azaz 90%-ban biztos, hogy $P(Fej) = 0{,}5$). Ekkor a Dempster–Shafer-elmélet azt adja, hogy $Bel(Fej) = 0{,}9 \times 0{,}5 = 0{,}45$, és hasonlóan $Bel(\neg Fej) = 0{,}9 \times 0{,}5 = 0{,}45$. Ekkor még mindig 10% az a „hiány", amit a tényeink nem fednek le. „Dempster szabálya" (Dempster, 1968) megmutatja, hogy hogyan kombináljunk tényeket, hogy új *Bel* értéket kapjunk, Shafer munkája pedig ezt egy teljes számítási modellként fogalmazta meg.

Mint az alapértelmezéses következtetésnél, itt is probléma a bizonyosság és a cselekvések összekapcsolása. Valószínűségek esetén a döntéselmélet szerint, ha $P(Fej) = 0{,}5$ és $P(\neg Fej) = 0{,}5$, akkor (feltéve, hogy nyerni és veszteni 1000 forintot pontosan egymás ellentettjei) a következtető számára közömbös lesz a tét elfogadása vagy elutasítása. Egy Dempster–Shafer-következtető esetén $Bel(\neg Fej) = 0$, és így nincs oka elfogadni, de ugyanakkor $Bel(Fej) = 0$, és így nincs oka vonakodni sem. Így látható, hogy ebben az esetben a Dempster–Shafer-következtető ugyanarra a következtetésre jutott, hogy hogyan cselekedjen. Sajnos a Dempster–Shafer-elmélet számos más esetben is megenged nem definiált döntéseket, amikor a valószínűségi következtetés konkrét választást eredményez. Valójában a hasznosság fogalma a Dempster–Shafer-modellben még nem kellően tisztázott.

Dempster–Shafer-elmélet egyik értelmezése, hogy egy valószínűségi intervallumot definiál – az intervallum a *Fej* esetén a szakértő tanúságtétele előtt [0, 1], míg utána [0,45, 0,55]. Az intervallum nagysága hasznos segítséget adhat annak megítélésében, hogy mikor van szükségünk további tényekre: elárulhatja, hogy a szakértő tanúságtétele akkor fog segíteni, ha nem tudjuk, hogy szabályos-e az érme, de nem fog segíteni, ha már tudjuk, hogy az érme szabályos. Azonban nincsenek világos irányelvek ennek megtételére vonatkozóan, mivel nincs világos értelmezés arra, hogy mit is jelent az intervallum szélessége. A Bayes-megközelítésben ez a fajta következtetés könnyen elvégezhető, ha megvizsgáljuk azt, hogy újabb tények begyűjtésével hogyan változna egy bizonyosság. Például a fejek megjelenésével kapcsolatos bizonyosságunkat jelentősen befolyásolná, ha tudjuk, hogy az érme szabályos, és aszimmetrikus súly észrevétele jelentősen befolyásolná azon bizonyosságunkat, hogy az érme szabályos. Egy teljes Bayes-modell magában foglalna ehhez hasonló tényezőkre vonatkozó valószínűségi becsléseket is, megengedve, hogy a „tudatlanságunkat" bizonyosságaink azon megváltozásának mértékével fejezzük ki, amelyek jövőbeli megfigyelések esetén következnének be.

Created by XMLmind XSL-FO Converter.

7.3. A meghatározatlanság reprezentálása: fuzzy halmazok és logikák

Fontos

A **fuzzy halmazok elmélete** (**fuzzy set theory**) egy eszköz annak specifikálására, hogy egy objektum milyen mértékben illeszkedik egy bizonytalan leíráshoz. Például fontoljuk meg azt az állítást, hogy „Nóri magas”. Igaz ez akkor, ha Nóri 175 cm magas? A legtöbb ember vonakodna „igent” vagy „nemet” mondani, és inkább azt választaná, hogy „olyasmi”. Látható, hogy ez nem egy külső világra vonatkozó bizonytalanság – mi biztosak vagyunk Nóri magasságában. A probléma az, hogy a „magas” nyelvi kifejezés nem az objektumok éles kettéosztását jelenti – a magasságnak *fokozatai* vannak. Ezért *a legtöbb szerző szerint a fuzzy halmazelmélet egyáltalán nem használható bizonytalansági következtetésekre.* Ehelyett a fuzzy halmazelmélet a *MagasSzemély*-t egy fuzzy predikátumnak tekinti, aminek az igazságértéke inkább egy 0 és 1 közötti érték, mint hogy *igaz* vagy *hamis* lenne. A „fuzzy halmaz” név a predikátum értelmezéséből származik, ahogyan implicite definiálja a halmaz tagjait – egy halmazt, aminek nincsenek éles határai.

A **fuzzy logika** (**fuzzy logic**) egy eljárás olyan logikai kifejezésekkel való következtetésre, amelyek fuzzy halmazbeli tagsági állításokat írnak le. Például annak az összetett kifejezésnek, hogy *MagasSzemély*(*Nóri*) ∨ *Nehéz*(*Nóri*) van fuzzy igazságértéke, ami a kifejezésben lévő komponensek igazságértékeinek a függvénye. Egy összetett kifejezés *T* fuzzy igazságértékének a kiértékelésére a következő alapvető szabályok szolgálnak:

$$T(A \wedge B) = \min(T(A), T(B))$$

$$T(A \vee B) = \max(T(A), T(B))$$

$$T(\neg A) = 1 - T(A)$$

A fuzzy logika tehát egy igazságfüggvényen alapuló rendszer, és ez komoly nehézségeket okoz. Tegyük fel például, hogy $T(MagasSzemély(Nóri)) = 0{,}6$ és $T(Nehéz(Nóri)) = 0{,}4$. Ekkor azt kapjuk, hogy $T(MagasSzemély(Nóri) \wedge Nehéz(Nóri)) = 0{,}4$, ami elfogadhatónak tűnik, de az is adódik, hogy $T(MagasSzemély(Nóri) \wedge \neg MagasSzemély(Nóri)) = 0{,}4$, ami már nem. Világos, hogy a probléma amiatt lép fel, hogy az igazságfüggvényen alapuló megközelítés nem képes kezelni a pozitív és negatív korrelációkat az állítás részei között.

A **fuzzy szabályozás** (**fuzzy control**) egy metodológia olyan szabályozási rendszerek alkotására, amelyekben a valós értékű bemenetek leképezését a kimeneti paraméterekre fuzzy szabályok reprezentálják. A fuzzy szabályozás nagyon sikeresnek bizonyult olyan kereskedelmi termékekben, mint az automatikus sebességváltók, a videokamerák és a villanyborotvák. A kritikusok [lásd például (Elkan, 1993)] úgy érveltek, hogy ezek az alkalmazások azért sikeresek, mert kicsi a szabálybázisuk, nem végeznek többlépéses következtetést, és hangolható paramétereik vannak, amelyek állításával a rendszer teljesítménye javítható. Az a tény, hogy fuzzy műveleteket használnak, mellékes a sikereik szempontjából; a magyarázat egyszerűen az, hogy egy tömör és szemléletes módszert biztosítanak egyenletesen interpoláló, valós értékű függvények megadására.

Voltak arra irányuló kísérletek, hogy a fuzzy logikát a valószínűség-számítás keretein belül értelmezzék. Az egyik elgondolás az, hogy az olyan kijelentéseket, mint „Nóri magas” úgy fogjuk fel, mint egy folytonos rejtett változó, Nóri valódi *Magasság*-ának diszkrét megfigyeléseit. A valószínűségi modell meghatározza a $P(\text{„Megfigyelő állítja, hogy Nóri magas”} \mid Magasság)$ valószínűséget, például az 3.1. szakasz - Bayes-hálók folytonos változókkal részben leírt **probit eloszlás** (**probit distribution**) felhasználásával. Ekkor az a posteriori eloszlás Nóri magassága felett a megszokott módon számolható ki, például ha a modell része egy hibrid Bayes-hálónak. Egy ilyen megközelítés természetesen nem igazságfüggvényen alapuló. Például a feltételes eloszlás

$$P(\text{„Megfigyelő állítja, hogy Nóri magas”} \mid Magasság, Súly)$$

kapcsolatot enged meg a magasság és a súly között, ahogyan a megfigyelést okozzák. Így nagyon valószínűtlen, hogy valakit, aki két és fél méter magas és kilencven kilogramm, „magas és nehéz”-nek nevezzünk, bár a „két és fél méter” „magas”-nak és a „kilencven kilogramm” „nehéz”-nek minősül.

A fuzzy predikátumoknak szintén adható egy valószínűségi értelmezés a **valószínűségi halmaz**okat (**random set**s) felhasználva – azaz olyan valószínűségi változókat, amiknek lehetséges értékei objektumok halmazai. Például a *MagasSzemély* egy valószínűségi halmaz, amelynek lehetséges értékei emberek halmazai. A *P*(*Magas*

Created by XMLmind XSL-FO Converter.

= S_1) valószínűség, ahol S_1 az embereknek valamely konkrét halmaza, annak a valószínűsége, hogy pontosan ezt a halmazt minősítené „magas”-nak a megfigyelő. Ekkor a „Nóri magas” valószínűsége minden olyan halmaz valószínűségének az összege, amelyben Nóri elemként jelen van.

Mind a hibrid Bayes-hálós megközelítés, mind a valószínűségi halmaz megközelítés úgy tűnik, hogy modellezi a meghatározatlanság egyes vetületeit bizonyosságok felhasználása nélkül. Mindazonáltal számos megválaszolatlan kérdés van a nyelvi megfigyelések és folytonos mennyiségek helyes reprezentációjával kapcsolatban, amely kérdésekkel a fuzzy kutatói közösségen kívül a többség nem foglalkozik.

8. Összefoglalás

Ez a fejezet a **Bayes-hálók**at (**Bayesian network**s) mutatta be, amelyek egy jól kidolgozott reprezentációt jelentenek a bizonytalan tudás számára. A Bayes-hálók nagyjából hasonló szerepet töltenek be, mint amit az ítéletlogika a biztos tudás számára.

- A Bayes-háló egy irányított körmentes gráf, aminek csomópontjai valószínűségi változókhoz tartoznak; minden csomóponthoz tartozik egy feltételes eloszlás, ahol a feltételt a csomópont szülei jelentik.
- A Bayes-hálók tömör módot adnak a tárgyterület **feltételes függetlenségi** (**conditional independence**) kapcsolatainak a reprezentálására.
- Egy Bayes-háló egy teljes együttes eloszlást specifikál; egy együttes bejegyzést a megfelelő lokális feltételes eloszlásokbeli bejegyzések szorzata határoz meg. Egy Bayes-háló gyakran exponenciálisan kisebb méretű, mint a teljes együttes eloszlás.
- Számos feltételes eloszlást lehet tömören reprezentálni eloszlások kanonikus családjaival. A **hibrid Bayes-hálók** (**hybrid Bayesian network**s), amelyek mind diszkrét, mind folytonos változókat tartalmaznak, sokféle kanonikus eloszlást használnak.
- A következtetés Bayes-hálókban a célváltozók egy halmaza valószínűség-eloszlásának kiszámítását jelenti, feltéve a tényváltozók egy halmazát. Az egzakt következtetési algoritmusok, mint a **változó eliminálás** (**variable elimination**) feltételes valószínűségek szorzatainak összegét értékelik, amilyen hatékonyan csak lehet.
- **Polifá**kban (**polytree**s) (egyszeresen összekötött hálókban) az egzakt következtetés számítási ideje a háló méretével lineárisan nő. Általános esetben a probléma kezelhetetlen.
- Sztochasztikus közelítő számítási technikák, mint a **valószínűségi súlyozás** (**likelihood weighting**) és a **Markov lánc Monte Carlo** (**Markov chain Monte Carlo**) módszerek elfogadható becsléseket adnak a hálóbeli valódi a posteriori valószínűségekre, és sokkal nagyobb hálókkal is megbirkóznak, mint az egzakt algoritmusok.
- A valószínűség-számítás kombinálható az elsőrendű logikából vett reprezentációs ötletekkel, hogy nagyon hatékony rendszereket hozzunk létre bizonytalanság esetén történő következtetéshez. A relációs valószínűségi modellek (RVM-ek) eleget tesznek olyan reprezentációs megkötéseknek, amelyek egy olyan jól definiált valószínűség eloszlást garantálnak, amely egy ekvivalens Bayes-hálóval kifejezhető.
- Számos egyéb rendszert is javasoltak a bizonytalanság esetén történő következtetéshez. Nagy általánosságban azt lehet mondani, hogy az **igazságfüggvény**en (**truth-functional**) alapuló rendszerek az ilyen érveléshez nem a legmegfelelőbb eszközök.

8.1. Irodalmi és történeti megjegyzések

A hálók felhasználása valószínűségi információk reprezentálására Sewall Wright munkájával kezdődött el a 20. század elején a genetikai öröklődés analízisével és az állatok növekedési tényezőivel kapcsolatban (Wright, 1921; 1934). Az ő hálóinak egyike látszik ennek a könyvnek a borítóján. I. J. Good, Alan Turinggal együttműködve, olyan valószínűségi reprezentációkat és Bayes következtetési módszereket fejlesztett ki,

Created by XMLmind XSL-FO Converter.

amelyeket a modern Bayes-hálók előfutárának lehet tekinteni – bár a cikkre nem hivatkoznak gyakran ebben a kontextusban (Good, 1961).[13] Ugyanez a cikk az eredeti forrása a zajos-VAGY modellnek.

A döntési problémák **hatásdiagram** (**influence diagram**) reprezentációját – ami a valószínűségi változók reprezentációjához egy irányított körmentes gráfot tartalmazott – döntéselemzések során használták az 1970-es évek vége felé (lásd 16. fejezet), de a kiértékelésre csak felsorolást használtak. Judea Pearl fejlesztette ki az üzenetváltó eljárást, a fa hálókban (Pearl, 1982a) és az egyszeresen összekötött (polifa) hálókban (Kim és Pearl, 1983) történő következtetésre, és szemben az akkor népszerű bizonyossági tényezős rendszerekkel, a diagnosztikai helyett a kazuális valószínűségi modellek létrehozásának fontossága mellett érvelt. Az első szakértői rendszer, ami Bayes-hálókat használt, a CONVINCE volt (Kim, 1983; Kim és Pearl, 1987). A későbbi rendszerek között találjuk a mozgatóidegekkel kapcsolatos rendellenességek diagnosztizálására szolgáló MUNIN (Andersen és társai, 1989) és a patológiás helyzetek felderítésére szolgáló PATHFINDER rendszert (Heckerman, 1991). Messze a legtöbbet használt Bayes-hálós rendszerek a Microsoft Windows diagnóziskijavítás moduljai (például a Nyomtató varázsló) (Breese és Heckerman, 1996) és az Office segéd a Microsoft Office-ban (Horvitz és társai, 1998).

Általános Bayes-hálókban történő egzakt következtetésre Pearl fejlesztett ki egy csoportosításon alapuló algoritmust, felhasználva egy konverziót változók csoportjai feletti irányított polifába, amelyben üzenetváltások biztosítják a csoportok közös változói feletti konzisztenciát. Egy hasonló módszer, amit David Spiegelhalter és Steffen Lauritzen fejlesztett ki (Spiegelhalter, 1986; Lauritzen és Spiegelhalter, 1988), egy irányítatlan Markov-hálóba vivő konverzión alapul. Az eljárást a HUGIN rendszerben valósították meg, ami egy hatékony és széles körben használt eszköz a bizonytalansági következtetések esetén (Andersen és társai, 1989). Ross Shachter, a hatásdiagramok kutatói közösségéből, egy egzakt módszert fejlesztett ki, ami a háló célirányos csökkentésén alapul a posteriori megőrző átalakításokat felhasználva.

A fejezetben leírt változó eliminációs módszer Shachter módszeréhez áll legközelebb, amiből a szimbolikus valószínűségi következtetés (SzVK) módszere alakult ki (Shachter, 1990). Az SzVK megkísérli optimalizálni az olyan kifejezési fák kiértékelését, mint amilyen a 14.8. ábrán látható. Az általunk ismertetett algoritmus a Zhang és Poole által kifejlesztetthez áll a legközelebb (Zhang és Poole, 1994; 1996). Irreleváns változók eltávolítására szolgáló kritériumokat Geiger és társai, illetve Lauritzen és társai fejlesztettek ki (Geiger és társai, 1990; Lauritzen és társai, 1990); az általunk megadott kritérium ezeknek egy egyszerű speciális esete. Rina Dechter (Dechter, 1999) mutatta meg, hogy a változó eliminálás ötlete lényegében megegyezik a **nem folytatólagos dinamikus programozás**sal (**nonserial dynamic programming**) (Bertele és Brioschi, 1972). Ez egy algoritmikus megközelítés, amit Bayes-hálókban előálló következtetési problémák széles körére lehet sikerrel alkalmazni. Ilyen probléma például megfigyelések egy adott halmazához a legvalószínűbb magyarázat megkeresése. Ez összekapcsolja a Bayes-hálós algoritmusokat a kényszerkielégítési problémákat megoldó kapcsolódó módszerekkel, és közvetlen módszereket szolgáltat az egzakt következtetés komplexitására a háló hiperfájának a szélessége alapján.

Folytonos valószínűségi változók jelenlétét Bayes-hálókban Pearl, illetve Shachter és Kenley vizsgálta (Pearl, 1988; Shachter és Kenley, 1989); ezek a cikkek kizárólag folytonos változókat tartalmazó hálókat tárgyaltak, amelyek lokális függési modelljei lineáris normális eloszlásúak. Diszkrét változók bevezetését Lauritzen és Wermuth vizsgálta (Lauritzen és Wermuth, 1989), amit a cHUGIN rendszerben meg is valósítottak (Olesen, 1993). A probit eloszlást először Finney vizsgálta (Finney, 1947), szigmoid eloszlásnak nevezve. Az eloszlást széles körben használják diszkrét választási jelenségek modellezésére, és kiterjeszthető több mint két választási lehetőség kezelésére (Daganzo, 1979). A logit eloszlás használatához Bishop ad indoklást (Bishop, 1995).

Cooper mutatta meg, hogy a következtetés általános problémája tetszőleges Bayes-hálóban NP-teljes (Cooper, 1990), Paule Dagum és Mike Luby pedig azt, hogy ennek a közelítő megoldása is NP-teljes (Dagum és Luby, 1993). A tárkomplexitás szintén komoly probléma mind a csoportosító, mind a változó eliminációs módszereknél. A **vágóhalmaz feltételezés**eken (**cutset conditioning**) alapuló eljárás, amit a kényszerkielégítési problémákra az 5. fejezetben mutattunk be, elkerüli az exponenciális méretű táblák létrehozását. Egy Bayes-hálóban a vágóhalmaz a csomópontok azon halmaza, amelyek értékeinek rögzítése esetén a fennmaradó csomópontok függései egy fagráfra egyszerűsödnek, ami már lineáris időben és térben megoldható. A lekérdezést összegzéssel adhatjuk meg a vágóhalmaz minden lehetséges értékadása mellett, így az összes tárigény még mindig lineáris (Pearl, 1988). Darwiche (Darwiche, 2001) egy rekurzívan feltételező algoritmust ad meg, ami tetszőleges tár/idő egyensúlyt tesz lehetővé.

[13] I. J. Good vezető statisztikus volt Turing kódfeltörő csapatában a második világháború során. A *2001: űrodüsszeia* c. művében Arthur C. Clarke Goodnak és Minskynek tulajdonította azokat az áttöréseket, amelyek a HAL számítógép kifejlesztéséhez vezettek (Clarke, 1968a).

Created by XMLmind XSL-FO Converter.

Gyors közelítő algoritmusok Bayes-hálókra való kifejlesztése nagyon aktív terület, a statisztika, a számításelmélet és a fizika határterületén. Az elutasító mintavételezés módszere egy általános, a statisztikában régóta ismert technika; Bayes-hálóknál Max Henrion alkalmazta először, aki **logikai mintavételezés**nek (**logic sampling**) nevezte (Henrion, 1988). A valószínűségi súlyozás, amit Fung és Chang (Fung és Chang, 1989), illetve Shachter és Peot fejlesztettek ki (Shachter és Peot, 1989), egy példája a **fontossági mintavétel** (**importance sampling**) jól ismert statisztikai módszerének. A valószínűségi súlyozás nagymérvű felhasználását az orvosi diagnosztikában Shwe és Cooper jelenti (Shwe és Cooper, 1991). Cheng és Druzdzel a valószínűségi súlyozás adaptív verzióját közlik (Cheng és Druzdzel, 2000), ami akkor is jól működik, ha a bizonyíték a priori valószínűsége nagyon alacsony.

A Markov lánc Monte Carlo (MCMC) módszerek a Metropolis algoritmussal kezdődtek, ami Metropolisnak köszönhető (Metropolis és társai, 1953), ami a 4. fejezetben leírt szimulált lehűtés algoritmusának szintén a forrása. A Gibbs-mintavételezőt Geman és Geman fejlesztette ki irányítatlan Markov-hálókban történő következtetésre (Geman és Geman, 1984). Az MCMC Bayes-hálókban történő alkalmazása Pearlnek köszönhető (Pearl, 1987). A Gilks és társai által összegyűjtött cikkek MCMC-alkalmazások széles skáláját ölelik fel, számos közülük a jól ismert `BUGS` csomaggal lett kifejlesztve (Gilks és társai, 1996).

A közelítési módszereknek két nagyon fontos családja létezik, amelyeket nem tárgyaltunk a fejezetben. Az első család a **variációs közelítés** (**variational approximation**) módszere, ami mindenfajta komplex számítás egyszerűsítésére felhasználható. Az alapötlet az, hogy az eredeti problémának előállítjuk egy olyan redukált verzióját, ami annyira egyszerű, hogy dolgozni lehessen vele, de amennyire lehet hasonló az eredeti problémához. A redukált probléma leírása tartalmazza a **λ variációs paramétereket** (**variational parameters**), amelyeket úgy állítunk be, hogy minimalizáljon egy D távolságfüggvényt az eredeti és a redukált problémák között, gyakran a $\partial D/\partial\lambda = 0$ egyenletrendszer megoldásával. Számos esetben pontos felső és alsó korlátok kaphatók. Variációs módszereket régóta használnak a statisztikában (Rustagi, 1976). A statisztikus fizikában a **mezőátlag** (**mean field**) módszer egy speciális variációs módszer, amelyben a modellt alkotó egyes változókat teljesen függetlennek tételezzük fel. Ezt az ötletet felhasználva sikerült kezelni nagy irányítatlan Markov-hálókat (Peterson és Anderson, 1987; Parisi, 1988). A variációs módszerek Bayes-hálókra történő alkalmazásának matematikai megalapozását Saul és társai fejlesztették ki, és ők kaptak a szigmoid hálókra pontos alsó korlát közelítéseket a mezőátlag módszerrel (Saul és társai, 1996). Jaakkola és Jordan úgy terjesztették ki a módszert (Jaakkola és Jordan, 1996), hogy mind az alsó, mind a felső korlát kinyerhető legyen. A variációs módszereket Jordan tekintette át (Jordan, 1999).

A közelítő algoritmusok második családja Pearl fagráfon futó üzenetváltásos algoritmusán alapszik (Pearl, 1982a). Az algoritmust általános hálókra is lehet alkalmazni, ahogy Pearl javasolta (Pearl, 1988). Az eredmények lehetnek helytelenek, és lehet, hogy az algoritmus nem áll le, de számos esetben a szolgáltatott értékek közel vannak a helyes értékekhez. Ez az úgynevezett **bizonyosságterjesztés**es (**belief propagation**) vagy **hurkos terjesztés**es (**loopy propagation**) megközelítés kevés figyelmet kapott, mindaddig, amíg McEliece és társai fel nem figyeltek (McEliece és társai, 1998) arra, hogy az üzenetváltás többszörösen összekötött Bayes-hálókban pontosan megegyezik a **turbó dekódolás** (**turbo decoding**) algoritmus által végzett számítással (Berrou és társai, 1993), ami jelentős áttörést eredményezett a hibajavító kódok tervezésében. Ebből az a következtetés vonható le, hogy a bizonyosságterjesztés gyors is és pontos is a nagyon nagy méretű és nagyon sűrűn összekötött dekódolásra használt hálókban, és ezért lehet, hogy általánosabban is hasznos lehet. Murphy és társai egy kísérletről számoltak be, ahol ez valóban működött (Murphy és társai, 1999). Yedidia és társai további kapcsolatokra mutattak rá a bizonyosságterjesztés és a statisztikus fizika között (Yedidia és társai, 2001).

A valószínűség és az elsőrendű nyelvek közötti kapcsolatot Carnap tanulmányozta (Carnap, 1950). Gaifman, illetve Scott és Krauss megadott egy nyelvet, amelyben valószínűségeket elsőrendű mondatokhoz lehet kapcsolni, és amelyre a modellek valószínűségi mértékek voltak lehetséges világokon (Gaifman, 1964; Scott és Krauss, 1966). Az MI-n belül ezt az ötletet az ítéletlogikára Nilsson, a predikátumlogikára pedig Halpern fejlesztette ki (Nilsson, 1986; Halpern, 1990). A tudásreprezentációs kérdések első kiterjedt vizsgálatát ilyen nyelvek esetén Bacchus végezte (Bacchus, 1990), Wellman és társainak a tanulmánya a korai implementációkat tekinti át (Wellman és társai, 1992), amelyek egy ekvivalens (ítéletlogikai) Bayes-háló létrehozásán alapulnak. Napjainkra a kutatók elkezdték megérteni a *teljes* tudásbázisok fontosságát, azaz olyan tudásbázisok fontosságát, amelyek akár a Bayes-hálók, egy egyértelmű együttes eloszlást definiálnak az összes lehetséges világ felett. Az erre szolgáló eljárások a logikai programozás valószínűségi verzióin (Poole, 1993; Sato és Kameya, 1997) vagy szemantikus hálókon (Koller és Pfeffer, 1998) alapultak. A fejezetben ismertetett típusú relációs valószínűségi modelleket Pfeffer vizsgálta részletesebben (Pfeffer, 2000). Pasula és Russell mind a relációs, mind az azonossági bizonytalanságot vizsgálta az RVM-ekben és az MCMC következtetés felhasználásában (Pasula és Russell, 2001).

Created by XMLmind XSL-FO Converter.

Amint a 13. fejezetben kifejtettük, a korai valószínűségi rendszerek kegyvesztettek lettek az 1970-es években, helyt adva az alternatív módszerek megjelenésének. A bizonyossági tényezőket a MYCIN orvosi szakértői rendszerbeli felhasználásra fejlesztették ki (Shortliffe, 1976), amit egyrészt mérnöki megoldásnak is, másrészt az emberi ítélethozatal modelljének is javasolták, bizonytalanság esetében. A *Rule-Based Expert Systems* gyűjtemény (Buchanan és Shortliffe, 1984) teljes áttekintést ad a MYCIN-ről és leszármazottairól (lásd még [Stefik, 1995]). David Heckerman megmutatta, hogy a bizonyossági tényezős számítások kicsit módosított verziója már helyes valószínűségeket ad bizonyos esetekben, de más esetekben a bizonyítékok hibát okozó, túlzott figyelembevételére vezet (Heckerman, 1986). A PROSPECTOR szakértői rendszer (Duda és társai, 1979) szabályalapú megközelítést használt, amiben a szabályokat (ritkán tartható) globális függetlenségi feltételezésekkel igazolták.

A Dempster–Shafer-elmélet Arthur Dempster publikációjával kezdődött, amiben a pontszerű valószínűségi értékek általánosítását javasolta intervallumértékekre, és szabályokat javasolt a kombinálásukra (Dempster, 1968). Glenn Shafer későbbi munkája vezetett el a Dempster–Shafer-elmélet és a valószínűség-számítás egymással versengő szemléletéhez (Shafer, 1976). Ruspini a Dempster–Shafer-elmélet és a valószínűség-számítás kapcsolatát elemezte (Ruspini és társai, 1992). Shenoy a Dempster–Shafer bizonyosságfüggvény alapján egy eljárást javasolt a döntéshozatalra (Shenoy, 1989).

A fuzzy halmazokat Lotfi Zadeh dolgozta ki válaszul arra a nehézségre (Zadeh, 1965), hogy egzakt bemeneti értékeket adjunk az intelligens rendszerek számára. Zimmermann leírása (Zimmermann, 2001) részletes bevezetést nyújt a fuzzy halmazok elméletébe; a (Zimmermann, 1999) pedig fuzzy alkalmazásokról szóló cikkek gyűjteménye. Amint korábban említettük, a fuzzy logikát gyakran tekintik a valószínűség-számítás versenytársának, pedig valójában eltérő kérdésekkel foglalkozik. A **lehetőségelmélet**et (**possibility theory**) (Zadeh, 1978) a bizonytalanság kezelésére vezették be fuzzy rendszerekben, és sok közös vonása van a valószínűség-számítással. Dubois és Prade alapos áttekintést nyújt a lehetőségelmélet és a valószínűség-számítás kapcsolatáról (Dubois és Prade, 1994).

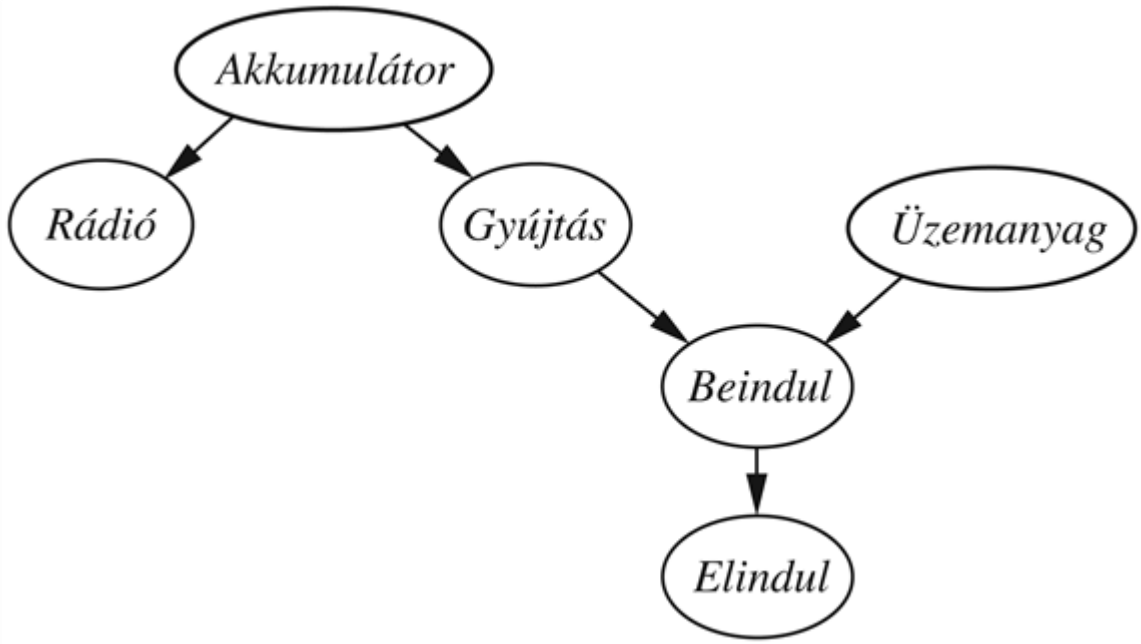
A valószínűség-számítás MI-n belüli feltámadása főként a Bayes-hálók felfedezésén múlt, ami módszert adott a feltételes függetlenségek reprezentálására és kihasználására. Ez az újjászületés igen küzdelmes volt; Peter Cheeseman harcias *In defense of Probability* (Cheeseman, 1985) és a későbbi *An Inquire into Computer Understanding* (Cheeseman, 1988, megjegyzésekkel) c. cikke ízelítőt ad a vitából. A logika művelőinek egyik legfőbb kifogása az volt, hogy a valószínűség-számítás miatt szükségesnek vélt számítások önelemzéssel nem érhetők el, és egy nem reális pontossági szintet tételeznek fel a bizonytalan tudásunkban. A **kvalitatív Bayes-hálók** (**qualitative probabilistic networks**) (Wellman, 1990a) lehetőséget adnak a Bayes-hálók tisztán kvalitatív absztrakciójára, csupán a változók közötti hatások pozitív és negatív jellegét kihasználva. Wellman megmutatta, hogy számos esetben ennyi információ is elegendő az optimális döntéshozatalhoz a valószínűségi értékek precíz meghatározása nélkül. Adnan Darwiche és Matt Ginsberg munkája kivonatolja a valószínűség-számítás elméletében szerepelő feltételesség és a tények kombinálásának alapvető tulajdonságait, és megmutatja, hogyan lehet ezeket alkalmazni logikai és alapértelmezési következtetésekben (Darwiche és Ginsberg, 1992).

A fejezetben leírt szívbetegség-kezelő rendszer Lucastól származik (Lucas, 1996). A Bayes-hálók más területen történő alkalmazásai között találjuk egyebek között a Microsoftnál végzett fejlesztéseket a felhasználó céljainak kikövetkeztetésére (Horvitz és társai, 1998) és a kéretlen elektronikus levelek szűrésére (Sahami és társai, 1998), az Electric Power Research Institute fejlesztését áramgenerátorok figyelésére és a NASA fejlesztését időkritikus információk megjelenítésére a Mission Controlnál Houstonban (Horvitz és Barry, 1995).

Az MI-n belüli bizonytalansági következtetéssel kapcsolatos számos fontos korai publikáció megtalálható a *Readings in Uncertain Reasoning* (Shafer és Pearl, 1990) és az *Uncertainty in Artifical Intelligence* (Kanal és Lemmer, 1986) antológiákban. A Bayes-hálók témakörének fejlődésében a legfontosabb egyedülálló publikáció vitathatatlanul a *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* (Pearl, 1988). Több kiváló munka tartalmazza az újabb anyagokat, mint például (Lauritzen, 1996; Jensen, 2001; Jordan, 2003). A valószínűségi következtetéssel kapcsolatos új eredmények mind az MI fő irányait meghatározó főbb lapokban, mint pl. az *Artificial Intelligence*-ben, mind a specializáltabb folyóiratokban, mint amilyen az *International Journal of Approximate Reasoning* jelennek meg. A **gráfos modell**ekről (**graphical model**s), amely modellosztály magában foglalja a Bayes-hálókat is, számos cikk statisztikai újságokban jelenik meg. Az *Uncertainty in Artificial Intelligence* (UAI), a *Neural Information Processing Systems* (NIPS) és az *Artificial Intelligence and Statistics* (AISTATS) konferenciák kiadványai kiváló forrásai a legfrissebb kutatásoknak.

Created by XMLmind XSL-FO Converter.

14.18. ábra - Egy gépkocsi elektromos és motorikus rendszereinek részeit leíró Bayes-háló. Minden változó bináris, és az *igaz* érték jelzi a megfelelő funkció helyes működését.



8.2. Feladatok

14.1.

Tekintsük a 14.18. ábrán látható gépkocsi-diagnosztikai hálót.

a. Egészítse ki a hálót a *JegesIdő* és az *IndítóMotor* bináris változókkal.

b. Adjon meg ésszerű feltételes valószínűségi táblákat az összes csomópontra.

c. Hány független értéket tartalmaz a teljes együttes valószínűség-eloszlás függvény 8 bináris csomópont esetén, feltételezve, hogy nincs közöttük feltételes függetlenségi reláció?

d. Hány független valószínűségi értéket tartalmaznak az ön hálójának táblázatai?

e. A *Beindul* feltételes valószínűség eloszlása megadható egy **zajos-ÉS** eloszlással. Írja le ezt az eloszláscsaládot általában és kapcsolatát a zajos-VAGY eloszláshoz.

14.2.

A helyi nukleáris erőműben van egy riasztó, ami érzékeli, ha a hőmérsékletmérő egy adott küszöbértéket meghalad. A mérő a reaktormag hőmérsékletét méri. Tekintse a következő bináris változókat: R (riasztó hangja), H_R (hibás riasztó), H_M (hibás mérő) és a következő többértékű változókat: M (mérő) és H (egy aktuális belső hőmérséklet).

a. Rajzoljon fel egy Bayes-hálót a tárgytartományra, feltételezve, hogy valószínűbb, hogy a mérő akkor hibásodik meg, ha a hőmérséklet túl magas.

b. A hálózata polifa lett?

c. Tételezze fel, hogy a hőmérsékletnek csak két lehetséges értéke van: normális és magas; és hogy a mérő helytelen hőmérsékletet mér az esetek x%-ában, ha működik, és y%-ában, ha hibás. Adja meg az M feltételes valószínűség-tábláit.

Created by XMLmind XSL-FO Converter.

d. Tételezze fel, hogy a riasztó tökéletesen működik, hacsak nem hibás, mely utóbbi esetben egyáltalán nem riaszt. Adja meg az *R*-hez tartozó feltételes valószínűség-táblát.

e. Tételezze fel, hogy a mérő és a riasztó működik, és a riasztó megszólal. Fejezze ki annak a valószínűségét, hogy a reaktormag hőmérséklete magas a háló feltételes valószínűségeinek a függvényében.

14.3.

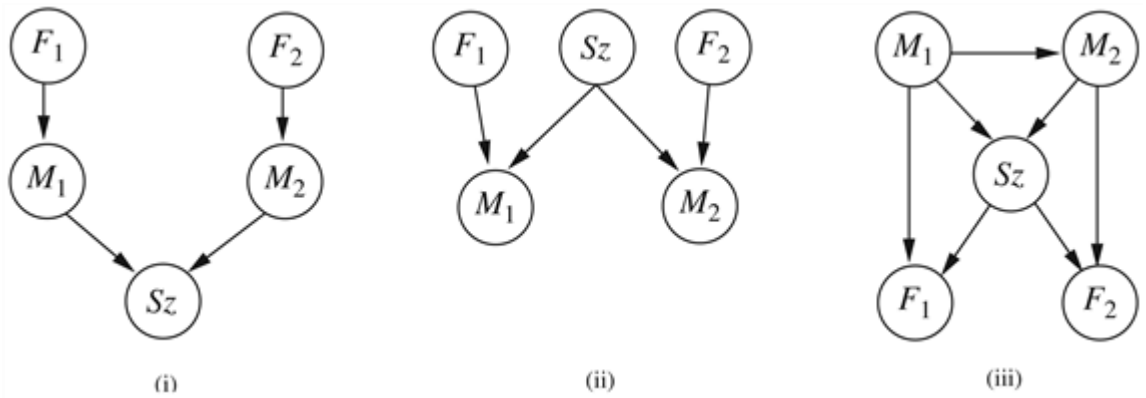
Két csillagász – távcsöveiket használva – a világ különböző részein M_1 és M_2 méréseket végeznek az égbolt egy kis részén látható csillagok számáról (*Sz*). Általában egy kis *e* valószínűsége a hibának, ami egy csillagnyi eltérést jelent. Bármelyik távcsővel megtörténhet az is (egy sokkal kisebb *f* valószínűséggel), hogy nincs rendesen fókuszálva (F_1 és F_2 események), amely esetben a tudós 3 vagy több csillaggal kevesebb csillagot számlál (ha *Sz* kisebb mint 3, akkor egyetlen csillagot sem észlel). Tekintsük a 14.19. ábrán látható három hálót.

a. Melyik Bayes-háló reprezentálja helyesen (de nem feltétlenül hatékonyan) a fenti ismereteket?

b. Melyik a legjobb háló? Adjon magyarázatot.

c. Adjon meg egy ésszerű feltételes valószínűségi táblát a $\mathbf{P}(M_1 | Sz)$ értékeire, amikor $M_1 \in \{0, 1, 2, 3, 4\}$ és $Sz \in \{1, 2, 3\}$. A feltételes eloszlás minden értékét az *e* és az *f* paraméterek függvényében fejezze ki.

14.18. ábra - Három lehetséges háló a távcsőproblémára



d. Tételezze fel, hogy $M_1 = 1$ és $M_2 = 3$. Mik a csillagok lehetséges számai, ha a priori nem korlátozzuk *Sz* értékét?

e. Mi a csillagok *legvalószínűbb* száma ezen megfigyelések esetén? Magyarázza el ennek kiszámítását, vagy ha nem lehetséges kiszámítani, magyarázza el, milyen további információkra van szükség, és hogy az miként befolyásolná az eredményt.

14.4.

Tekintsük a 14.19. (ii) ábrán látható hálót, és tegyük fel, hogy a két teleszkóp azonosan működik. Legyen M_1, $M_2 \in \{0, 1, 2, 3, 4\}$ és $Sz \in \{1, 2, 3\}$ a 14.3. feladatban megadott szimbolikus FVT-kkel. A felsoroló algoritmus felhasználásával számolja ki a következő valószínűség-eloszlást: $\mathbf{P}(Sz | M_1 = 2, M_2 = 2)$.

14.5.

Tekintsük a lineáris Gauss lokális eloszlású hálók családját, amit az 3.1. szakasz - Bayes-hálók folytonos változókkal rész illusztrál.

a. Egy kétváltozós hálóban legyen X_1 az X_2 szülője, X_1-nek legyen normális a priori eloszlása, $\mathbf{P}(X_2 | X_1)$ pedig egy lineáris normális eloszlás. Mutassa meg, hogy a $P(X_1, X_2)$ együttes eloszlás egy többváltozós Gauss-eloszlás, és számolja ki ennek a kovarianciamátrixát.

b. Bizonyítsa be indukcióval, hogy egy általános lineáris Gauss-háló $X_1, \ldots, X_n$-en vett együttes eloszlása szintén többváltozós Gauss-eloszlás.

Created by XMLmind XSL-FO Converter.

14.6.

Az 3.1. szakasz - Bayes-hálók folytonos változókkal részben definiált probit eloszlás egy bináris gyermek valószínűség-eloszlását írja le, egyetlen folytonos szülő esetén.

a. Hogyan terjeszthető ki a definíció több folytonos szülő esetére?

b. Hogyan terjeszthető ki, hogy *többértékű* gyermekváltozóra is alkalmazható legyen? Gondolja végig mind a két esetet, amikor a gyermek értékei sorrendezettek (ahogyan a vezetésnél a sebességfokozat megválasztása a sebesség, a lejtés, a kívánt gyorsulás stb. függvénye), és amikor nem állíthatók sorba (mint például a busz, a vonat vagy a kocsi választása munkába menetelnél). (*Segítség*: gondoljunk a lehetséges értékek két csoportra osztására, hogy bináris változót szimuláljunk.)

14.7.

Ez a feladat a 14.10. ábrán látható változó eliminációs algoritmusra irányul.

a. A 14.4. alfejezet a változó eliminálást alkalmazza arra a lekérdezésre, hogy

```
P(Betörés| JánosTelefonál = igaz, MáriaTelefonál = igaz)
```

Végezze el a jelzett számításokat, és ellenőrizze, hogy helyes-e a válasz.

b. Számolja össze az elvégzett aritmetikai műveletek számát, és hasonlítsa össze a felsorolási algoritmus által elvégzettek számával.

c. Tegyük fel, hogy a háló egy láncot alkot: bináris változók szekvenciáját, ahol $Szülők(X_i) = \{X_{i-1}\}$ $i = 2, ..., n$ esetén. Mi a $\mathbf{P}(X_1 | X_n = igaz)$ kiszámításának a komplexitása a felsorolást használva? És mi, ha változó eliminálást használunk?

d. Bizonyítsa be, hogy a változó eliminálás futásának komplexitása egyszeresen összekötött hálóban lineáris a háló méretében bármely, a háló struktúrájával konzisztens változó sorrendezés mellett.

14.8.

Vizsgáljuk meg az egzakt következtetés komplexitását általános Bayes-hálókban.

a. Bizonyítsa be, hogy bármely 3-SAT probléma redukálható egy egzakt következtetésre egy olyan Bayes-hálóban, amely a konkrét problémát reprezentálja, és így az egzakt következtetés NP-nehéz. (*Segítség*: gondoljon egy olyan hálóra, amiben minden kijelentésszimbólumhoz, minden klózhoz és a klózok minden konjunkciójához rendre egy-egy csomópont tartozik.)

b. A 3-SAT problémában a kielégítő érték-hozzárendelések számának meghatározása #P-teljes. Bizonyítsa be, hogy az egzakt következtetés legalább ennyire nehéz.

14.9.

Tekintsük egy véletlen minta egy megadott egyváltozós eloszlásból történő generálásának a problémáját. Feltételezhetjük, hogy rendelkezésre áll egy véletlenszám-generátor, ami egyenletes eloszlás szerint 0 és 1 közötti véletlen számokat ad.

a. Legyen X egy diszkrét valószínűségi változó $P(X_i = x_i) = p_i$ $i \in \{1, ..., k\}$ tömegfüggvénnyel. Az X **eloszlásfüggvény**e (**cumulative distribution**) minden lehetséges j-re megadja annak valószínűségét, hogy $X \in \{x_1, ..., x_j\}$.

   Magyarázza el, hogyan számolható ki az eloszlásfüggvény $O(k)$ időben, és hogyan generálhatunk vele egy X eloszlása szerinti mintát. Ez utóbbi megtehető kevesebb mint $O(k)$ időben?

b. Most tegyük fel, hogy N mintát szeretnénk generálni X szerint, ahol $N >> k$. Magyarázza el, hogyan tehető ez úgy meg, hogy a mintánkénti várható futási idő *konstans* (azaz független k-tól).

c. Most tekintsünk egy folytonos értékű változót egy parametrikus eloszlással (például normálissal). Hogyan generálhatunk mintákat egy ilyen eloszlásból?

Created by XMLmind XSL-FO Converter.

d. Tegyük fel, hogy egy folytonos értékű változót szeretne lekérdezni, és egy olyan mintavételező algoritmust használ a következtetésre, mint a VALÓSZÍNŰSÉGISÚLYOZÁS. Hogyan kellene módosítania a lekérdezés-válaszadás folyamatát?

14.10.

Egy változó **Markov-takaró**ját (**Markov blanket**) az 2.2. szakasz - Feltételes függetlenségi relációk Bayes-hálókban definiáltuk.

a. Bizonyítsa be, hogy a változó független a háló összes többi változójától, ha Markov-takarója ismert.

b. Vezesse le a (14.11) egyenletet.

14.11.

Tekintsük a $\mathbf{P}(Eső \mid Locsoló = igaz, VizesPázsit = igaz)$ lekérdezést a 14.11. (a) ábra szerint, és annak az MCMC módszerrel való megválaszolását.

a. Hány állapota van a Markov-láncnak?

Számítsa ki a Q **állapotátmenet-mátrix**ot (**transition matrix** Q), ami tartalmazza az összes $q(\mathbf{y} \rightarrow \mathbf{y}')$ értéket minden $\mathbf{y}$-ra és $\mathbf{y}'$-ra.

b. Mit reprezentál $\mathbf{Q}^2$, az állapotátemeneti mátrix négyzete?

c. Mi lesz $\mathbf{Q}^n$, ha $n \rightarrow \infty$?

d. Magyarázza el, hogy hajtsunk végre valószínűségi következtetést Bayes-hálókban, ha $\mathbf{Q}^n$ elérhető. Hatékony módja ez a következtetésnek?

14.12.

Három focicsapat, *A*, *B* és *C* játszik egymás ellen. Minden meccsen két csapat vesz részt, az eredmény pedig győzelem, vereség vagy döntetlen lehet. Minden csapatnak egy rögzített, ismeretlen fokozatú játékszintje van – ami egy 0 és 3 közötti egész érték –, és egy meccs eredménye a két csapat játékszintje közötti különbségtől függ sztochasztikusan.

a. Hozzon létre egy relációs valószínűségi modellt a tárgyterületre, és javasoljon értékeket az összes szükséges valószínűség-eloszlásra.

b. Hozzon létre egy ekvivalens Bayes-hálót.

c. Tegyük fel, hogy az első két meccsen *A* megveri *B*-t, *C*-vel pedig döntetlent játszik. Egy tetszőleges egzakt következtetést felhasználva, számítsa ki a harmadik meccs a posteriori eloszlását.

d. Tegyük fel, hogy *n* csapat vesz részt a bajnokságon, és minden eredményünk megvan, kivéve az utolsót. Hogyan változik az utolsó meccs megjóslásának a komplexitása az *n* függvényében?

e. Vizsgálja meg az MCMC alkalmazását ezen a problémán. Milyen gyorsan konvergál az MCMC a gyakorlatban, és hogyan skálázható fel?

Created by XMLmind XSL-FO Converter.

15. fejezet - Időbeli valószínűségi következtetés

Ebben a fejezetben megpróbáljuk értelmezni a jelent, megérteni a múltat és esetleg megjósolni a jövőt, még akkor is, ha igencsak kevéssé kristálytiszta.

A bizonytalan környezetben lévő ágenseknek – a logikai ágensekhez hasonlóan – képesnek kell lenniük környezetük aktuális állapotának nyomon követésére. Ezt a feladatot nehezebbé teszi a részleges és zajos érzékelés, és az a bizonytalanság, ahogyan a környezet az idő előrehaladtával változik. Az ágens a legjobb esetben is a jelenlegi helyzetnek csak egy valószínűségi értékeléséhez képes hozzájutni. Ez a fejezet olyan reprezentációkat és következtetési algoritmusokat ír le, amelyek lehetővé teszik ezt az értékelést, a 14. fejezetben bemutatott ötletekre építve.

Az alapvető megközelítés leírása a 15.1. alfejezetben szerepel: egy változó világot úgy modellezünk, hogy a világ állapotának minden vonatkozására, *minden időpillanatban* egy valószínűségi változót használunk. Ezen változók közti kapcsolatok írják le az állapot fejlődését. A 15.2. alfejezet meghatározza az alapvető következtetési feladatokat, és leírja az időbeli modellekhez tartozó következtető algoritmusok általános struktúráját. Ezután három különbözőfajta modellt írunk le: a **rejtett Markov-modell**eket (**hidden Markov model**), a **Kalman-szűrő**ket (**Kalman filter**s) és a **dinamikus Bayes-háló**kat (**dynamic Bayesian network**s) (ami mint speciális aleseteket magában foglalja a rejtett Markov-modelleket és a Kalman-szűrőket). Végül a 15.6. alfejezet bemutatja, miként alkotják az időbeli valószínűségi modellek a modern beszédfelismerő rendszerek magját. Ezen modellek mindegyikének létrehozásában központi szerepet tölt be a tanulás, de a tanulási algoritmusok részletes vizsgálata a VI. részre marad.

1. Idő és bizonytalanság

A valószínűségi következtetésre szolgáló technikáinkat **változatlan** (**static**) világok esetén fejlesztettük ki, amelyekben minden egyes valószínűségi változónak egyetlen rögzített értéke van. Például egy gépkocsi javításánál feltesszük, hogy ami meghibásodott, az a diagnosztizálás alatt is hibás marad; a feladatunk a gépkocsi állapotának kikövetkeztetése a megfigyelt bizonyítékokból, amelyek szintén változatlanok maradnak.

Most vegyünk fontolóra egy másféle problémát: egy cukorbeteg páciens kezelését. Akárcsak a gépkocsijavítás esetén, rendelkezünk bizonyítékokkal: a jelenlegi inzulinadagok, az élelmiszer-bevitel, a vércukorszint mérésének eredményei és egyéb testi tünetek. A feladat a páciens jelenlegi állapotának értékelése, beleértve az aktuális vércukorszintet és inzulinszintet. Ezeknek az információknak az alapján az orvos (vagy a páciens) döntést hoz a páciens élelmiszer-beviteli és inzulinadagjáról.

A gépkocsijavítás esetétől eltérően, itt a probléma *dinamikai* vonatkozásai alapvetőek. A vércukorszintek és méréseik idővel gyorsan változhatnak, amit befolyásol a páciens aktuális élelmiszer-bevitele és inzulinadagja, az anyagcsere aktivitása, a napszak és egyéb tényezők. Ahhoz, hogy a bizonyítékok időbeli alakulásából a jelenlegi állapotot megbecsüljük, és egy kezelés kimenetelét megjósoljuk, modelleznünk kell ezeket a változásokat.

Ugyanezek a szempontok számos más esetben is jelentkeznek, egy nemzet gazdasági aktivitásának közelítő és részleges statisztikák alapján történő követésétől, beszédszekvenciák zajos és többértelmű akusztikai mérésekből történő megértéséig. Adódik a kérdés: hogyan lehet az ezekhez hasonló dinamikai helyzeteket modellezni?

1.1. Állapotok és megfigyelések

Az alapvető megközelítés, amit követünk, hasonló a 10. fejezetben leírt szituációkalkulus alapjául szolgáló ötlethez: a változás folyamatát pillanatfelvételek sorozatának tekinthetjük, amelyek mindegyike egy adott pillanatban írja le a világ állapotát. Minden pillanatfelvétel vagy **időpont** (**time slice**) valószínűségi változók egy halmazát tartalmazza, amelyek némelyike megfigyelhető, némelyike pedig nem.

Az egyszerűség kedvéért a továbbiakban fel fogjuk tenni, hogy a változóknak ugyanaz a részhalmaza minden időpontban megfigyelhető (bár az elkövetkezőkben ez teljes mértékben sehol sem szükségszerű). A t időpillanatban nem megfigyelhető változók halmazának a jelölésére $\mathbf{X}_t$-t fogjuk használni, és $\mathbf{E}_t$-t a

Created by XMLmind XSL-FO Converter.

megfigyelhető változók halmazának a jelölésére. A t időpontbeli megfigyelés $\mathbf{E}_t = \mathbf{e}_t$ az értékek valamely $\mathbf{e}_t$ halmazára.

Gondoljuk át a következő leegyszerűsített példát: tegyük fel, hogy egy titkos, földalatti létesítmény biztonsági őrei vagyunk. Szeretnénk tudni, hogy vajon aznap esik-e, de a külvilághoz való egyetlen hozzáférésünket az jelenti, hogy reggelenként látjuk, hogy az igazgató esernyővel vagy esernyő nélkül jön be. Minden egyes t napon az $\mathbf{E}_t$ halmaz így egyetlen bizonyítékváltozót tartalmaz, az U_t-t (van-e esernyő), és $\mathbf{X}_t$ az R_t-t, az egyetlen állapotváltozót tartalmazza (esik-e). Más problémákhoz változók nagyobb halmaza tartozhat. A cukorbetegség példájában a bizonyítékváltozók lehetnek a $MértVércukor_t$ és a $Pulzusszám_t$, és az állapotváltozók lehetnek a $Vércukor_t$ és a $Gyomortartalom_t$.[1]

Az időpontok közötti intervallum szintén problémafüggő. A cukorbetegség követésénél az alkalmas intervallum inkább egy óra, mint egy nap. Ebben a fejezetben általában egy rögzített véges intervallumot tételezünk fel, ami azt jelenti, hogy az időpillanatok egész számokkal felcímkézhetők. Feltesszük, hogy az állapotsorozat $t = 0$-nál kezdődik; és jelentéktelen okokból feltesszük, hogy a bizonyítékok $t = 1$-nél kezdenek beérkezni, és nem $t = 0$-nál. Így az esernyős világot az R_0, R_1, R_2, ... állapotváltozók és az U_1, U_2, ... bizonyítékváltozók reprezentálják. Az $a : b$ jelölést fogjuk használni az egészek a-tól b-ig tartó sorozatának jelölésére (a határokat beleértve), és $\mathbf{X}_{a:b}$ jelöli a változók megfelelő halmazát $\mathbf{X}_a$-tól $\mathbf{X}_b$-ig. Például $U_{1:3}$ az U_1, U_2, U_3 változóknak felel meg.

1.2. Stacionárius folyamatok és a Markov-feltétel

Egy adott problémánál az állapotváltozók és a bizonyítékváltozók halmazának meghatározása után a következő lépés a változók közötti függőségek megadása. Követhetnénk a 14. fejezetben megállapított eljárást, valahogyan sorrendezve a változókat, és kérdéseket feltéve az elődöktől való feltételes függetlenségre, adott szülői halmaz esetén. Egy nyilvánvaló választás, hogy a változókat a természetes idősorrendjük szerint sorrendezzük, mivel az ok általában megelőzi a hatást, és a változókat lehetőleg az ok-okozati sorrendjük szerint vesszük.

Azonban hamar beleütközhetünk egy akadályba: a változók halmaza nem korlátos, mivel minden időpontra tartalmazza az állapot- és bizonyítékváltozókat. Valójában ez két problémát is felvet: (1) korlátlan számú feltételes valószínűségi táblát kell megadnunk minden változóra, minden időpillanatban; (2) ezek korlátlan számú szülőt tartalmazhatnának.

Az első problémát megoldja annak feltételezése, hogy a világ állapotának a változásait egy **stacionárius folyamat** (**stationary process**) okozza – azaz egy változási folyamat, amit olyan törvények határoznak meg, amik maguk nem változnak az idővel. (Ne keverjük össze a *stacionaritást* a *statikussággal:* egy *statikus* folyamatban maga az állapot nem változik.) Az esernyős világban ekkor a $\mathbf{P}(U_t|Szülők(U_t))$ feltételes valószínűség, hogy az esernyő feltűnik, minden t-re azonos. A stacionaritás feltevésénél ezért csak egy „reprezentatív” időpillanathoz tartozó változók feltételes valószínűségeit kell megadnunk.

A második problémát, a potenciálisan végtelen számú szülő kezelését, az úgynevezett **Markov-feltétel** (**Markov assumption**) elfogadása oldja meg – azaz, hogy a jelenlegi állapot a korábbi állapotoknak csak *véges* történetétől függ. Ennek a feltevésnek eleget tévő folyamatokat elsőként Andrei Markov orosz matematikus tanulmányozta részletesen. Ezeket a folyamatokat **Markov-folyamat**oknak (**Markov process**es) vagy **Markov-lánc**oknak (**Markov chain**s) neveznek. (A Markov-folyamatok feloszthatók aszerint, hogy az állapottér folytonos vagy diszkrét, illetve hogy az idő folytonos vagy diszkrét. Markov-láncoknak a diszkrét idejű és/vagy diszkrét állapotterű Markov-folyamatokat nevezik – *a ford*.) A Markov-folyamatok különböző jellemzőkkel rendelkezhetnek; a legegyszerűbb az **elsőrendű Markov-folyamat** (**first-order Markov process**), amelyben a jelenlegi állapot csak az előző állapottól függ, és nem függ egyetlen korábbitól sem. Máshogy fogalmazva, egyetlen állapot ismeretére van szükség, hogy a jövő a múlttól feltételesen független legyen az állapot ismeretében. A jelölésünket felhasználva, a megfelelő feltételes valószínűségi állítás azt mondja ki, hogy minden t-re

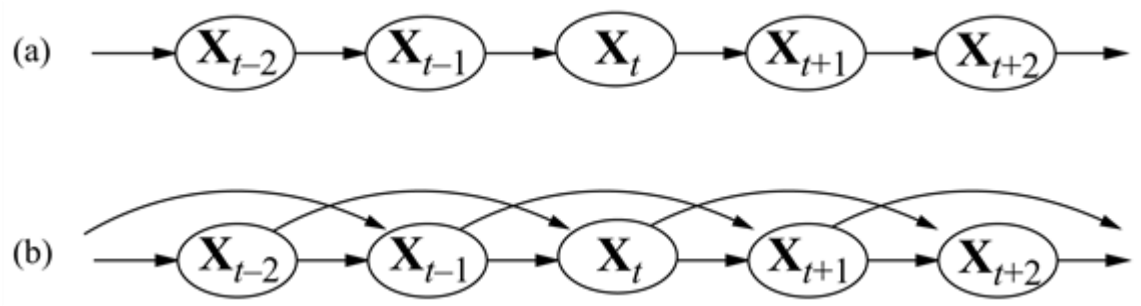
$$P(X_t | X_{0:t-1}) = P(X_t | X_{t-1}) \quad (15.1)$$

Így egy elsőrendű Markov-folyamatban az állapotok időbeli változását leíró szabályokat teljes mértékben tartalmazza a $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{1:t})$ feltételes eloszlás, amit az elsőrendű folyamat **állapotátmenet-modell**jének (**transition model**) nevezünk.[2] A 15.1. ábra elsőrendű és másodrendű Markov-folyamatokhoz tartozó Bayes-hálóstruktúrákat mutat.

[1] Vegyük észre, hogy a $Vércukor_t$ és a $MértVércukor_t$ két különböző változó, így kezeljük az aktuális mennyiségek zajos mérését.
[2] Az állapotátmenet-modell a valószínűségi analógja a 7. fejezet logikai frissítő áramkörének és a 10. fejezet követő állapot axiómáinak.

Created by XMLmind XSL-FO Converter.

15.1. ábra - (a) Egy elsőrendű Markov-folyamathoz tartozó Bayes-hálóstruktúra. A reprezentált Markov-folyamatban az állapotot az X_t változók definiálják. (b) Egy másodrendű Markov-folyamat.



Az $\mathbf{X}_t$ állapotváltozók szüleinek a korlátozásán túl korlátoznunk kell az $\mathbf{E}_t$ bizonyítékváltozók szüleit is. Tipikusan feltesszük, hogy a bizonyítékváltozók egy t időpillanatban csak az aktuális állapottól függnek:

$$P(E_t | X_{0:t}, E_{0:t-1}) = P(E_t | X_t) \quad (15.2)$$

A $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$ feltételes eloszlást **érzékelő modell**nek (**sensor model**) nevezik (vagy néha **megfigyelési modell**nek [**observational model**]), mivel leírja, hogy az „érzékelőket” – azaz a bizonyítékváltozókat – hogyan befolyásolja a világ aktuális állapota. Vegyük észre a függés irányát: a „nyíl” az állapottól az érzékelő értékére mutat, mivel a világ állapota *okozza* azt, hogy az érzékelők bizonyos értékeket vegyenek fel. Az esernyős világban például az eső okozza az esernyő feltűnését. (A következtetési folyamat természetesen a másik irányban halad; a modellezett függések iránya és a következtetés iránya közötti különbségtétel a Bayes-hálók egyik fő előnye.)

Az állapotátmenet-modellhez és az érzékelő modellhez még meg kell adnunk egy $\mathbf{P}(\mathbf{X}_0)$ a priori eloszlást a 0. időpontbeli állapotok felett. Ez a három eloszlás, kombinálva a (15.1) és a (15.2) egyenletekben megfogalmazott feltételes függetlenségi állításokkal, biztosítja számunkra a teljes együttes eloszlás meghatározását az összes változó felett. Bármely véges t-re azt kapjuk, hogy

$$\mathbf{P}(\mathbf{X}_0, \mathbf{X}_1, \ldots, \mathbf{X}_t, \mathbf{E}_1, \ldots, \mathbf{E}_t) = \mathbf{P}(\mathbf{X}_0)\prod_{i=1}^{t} \mathbf{P}(\mathbf{X}_i|\mathbf{X}_{i-1})\mathbf{P}(\mathbf{E}_i|\mathbf{X}_i)$$

A függetlenségi állítások egy nagyon egyszerű Bayes-hálóstruktúrának felelnek meg, ami az egész rendszert leírja. A 15.2. ábrán látható a háló struktúrája az esernyős példa esetén, beleértve az állapotátmenet- és az érzékelő modellekhez tartozó feltételes eloszlásokat.

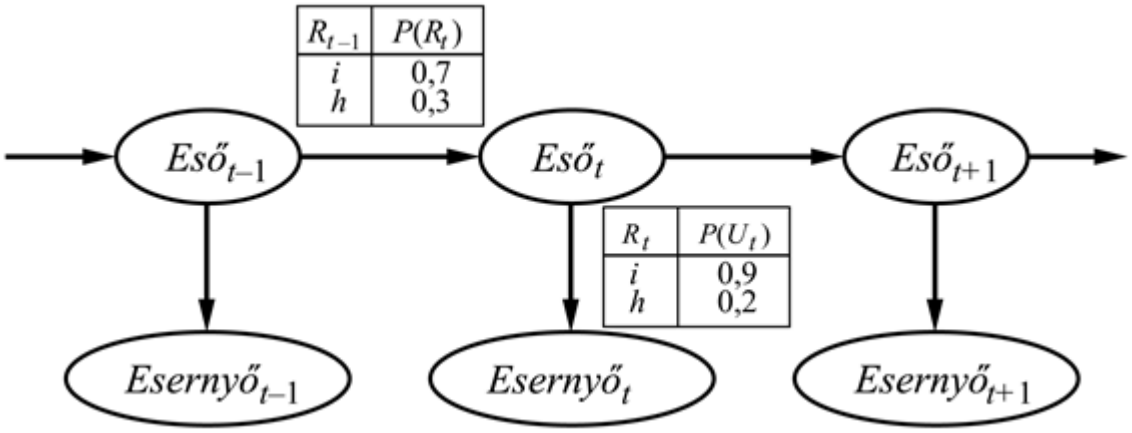
Az ábrán lévő struktúra elsőrendű Markov-folyamat feltételezésen alapul, mivel feltesszük, hogy az eső valószínűsége csak attól függ, hogy az előző nap esett-e. Egy ilyen feltevésnek a helyénvalósága magától a tárgyterülettől függ. Az elsőrendű Markov-feltétel kimondja, hogy az állapotváltozók az *összes* olyan információt tartalmazzák, amely a következő időpontbeli valószínűség-eloszlások megadásához szükséges. Néha ez a feltevés pontosan teljesül – például ha egy részecske **véletlen bolyongás**t (**random walk**) végez az x tengelyen, ±1-gyel változtatva meg a pozícióját minden időpontban, ekkor az x koordinátát használva állapotként, egy Markov-lánc adódik. Gyakran a feltevés csak közelítő, mint amikor az eső jóslása csak az alapján történik, hogy esett-e az előző napon. Két lehetséges javítás létezik, ha a közelítés túlságosan pontatlannak bizonyul:

1. A Markov-folyamat rendjének a megnövelése. Például létrehozhatnánk egy másodrendű modellt, felvéve egy $Eső_{t-2}$-t mint az $Eső_t$ szülőjét, ami lehet, hogy valamivel pontosabb predikciót adna (például Palo Altóban nagyon ritkán esik több mint két napig egyfolytában).

2. Az állapotváltozók halmazának megnövelése. Például felvehetnénk az $Évszak_t$ változót, hogy ez lehetővé tegye számunkra az esős évszakok történeti feljegyzéseinek beépítését, vagy hozzáadhatnánk a $Hőmérséklet_t$, a $Páratartalom_t$ és a $Légnyomás_t$ változókat, hogy az eső feltételeinek fizikai modelljeit felhasználhassuk.

Created by XMLmind XSL-FO Converter.

15.2. ábra - Az esernyős világot leíró Bayes-hálóstruktúra és feltételes eloszlások. Az álapotátmenet-modell a $P(Eső_t|Eső_{t-1})$ feltételes valószínűség-eloszlás, az érzékelő modell a $P(Esernyőt_t|Eső_t)$.



A 15.1. feladat annak megmutatását kéri, hogy az első megoldás – a folyamat rendjének a megnövelése – mindig átfogalmazható az állapotváltozók halmazának megnövelésére, változatlanul hagyva a rendet. Vegyük észre, hogy állapotváltozók hozzáadása javíthatja a rendszer előrejelző erejét, de megnöveli a predikciós *követelményeket* is: ekkor már az új változókat is jósolni kell. Így a változóknak egy „önmagában elégséges” halmazát keressük, ami valójában azt jelenti, hogy meg kell értenünk a modellezett folyamat „fizikáját”. A folyamat pontos modellezése iránti követelmény nyilvánvalóan mérsékeltebb, ha új érzékelőket vehetünk fel (például a hőmérséklet és a nyomás mérésére), amelyek közvetlenül az új állapotváltozókról szolgáltatnak információt.

Gondoljuk meg például egy X–Y síkon véletlenszerűen sétáló robot követésének a problémáját. Egy javaslat az állapotváltozók egy elégséges halmazára ekkor a pozíció és a sebesség lehet: Newton törvényeit felhasználva kiszámolható az új pozíció, és a sebesség megjósolhatatlanul változik. Ha azonban a robot akkumulátorról üzemel, akkor az elem lemerülésének tipikusan van egy szisztematikus hatása a sebesség megváltozására. Mivel ez viszont függ attól, hogy mennyi energia használódott el az összes korábbi manőverben, a Markov-tulajdonság sérül. A Markov-tulajdonságot helyreállíthatjuk egy $Akkumulátor_t$ feltöltöttségi szintnek mint az $\mathbf{X}_t$-t alkotó állapotváltozók egyikének a felvételével. Ez segít a robot mozgásának a jóslásában, de ugyanakkor megköveteli az $Akkumulátor_t$ jóslását az $Akkumulátor_{t-1}$-ből és a sebességből. Bizonyos esetekben ez megbízhatóan megtehető; a pontosság azonban javulna egy, az akkumulátor feltöltöttségét mérő új érzékelő felvételével.

2. Következtetés időbeli modellekben

Egy általános időbeli modellstruktúra kidolgozása után, most már megfogalmazhatjuk a megoldandó alapvető következtetési feladatokat:

- **Szűrés** (**filtering**) vagy **ellenőrző megfigyelés** (**monitoring**). Ez a **bizonyossági állapot** (**belief state**) kiszámításának a feladata – ami a jelenlegi állapot feletti a posteriori eloszlás, az adott időpontig vett összes bizonyíték ismeretében. Azaz szeretnénk kiszámítani a $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$ mennyiséget, feltéve, hogy a bizonyítékok folyamatos sorozatban érkeznek kezdve a $t = 1$ időponttól. Az esernyős példában ez az aznapi eső valószínűségének a kiszámítását jelentené, az esernyőhordozó eddigi összes megfigyelésének az ismeretében. A szűrés az, amit egy racionális ágensnek el kell végeznie ahhoz, hogy a jelenlegi állapotot követni tudja, és így racionális döntéseket hozhasson (lásd 17. fejezet). Kiderül, hogy majdnem azonos számítás szolgáltatja a bizonyítéksorozat megfigyelésének a **valószínűség**ét (**likelihood**), $P(\mathbf{e}_{1:t})$-t.

- **Előrejelzés** (**prediction**). Ez egy *jövőbeli* állapot feletti a posteriori eloszlás kiszámításának a feladata, az adott időpontig vett összes bizonyíték ismeretében. Azaz, szeretnénk kiszámítani a $\mathbf{P}(\mathbf{X}_{t+k}|\mathbf{e}_{1:t})$ mennyiséget valamely $k > 0$ esetén. Az esernyős példában ez jelentheti az eső valószínűségének a kiszámítását három napra előre, az esernyőhordozó eddigi összes megfigyelésének az ismeretében. Az előrejelzés hasznos a cselekedetek lehetséges sorozatainak a kiértékelésében.

- **Simítás** (**smoothing**) vagy **visszatekintés** (**hindsight**). Ez egy múltbeli állapot feletti a posteriori eloszlás kiszámításának a feladata, a jelen időpontig vett összes bizonyíték ismeretében. Azaz szeretnénk kiszámítani

Created by XMLmind XSL-FO Converter.

a $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ mennyiséget valamely $0 \leq k < t$ esetén. Az esernyős példában ez jelentheti az eső valószínűségének a kiszámítását múlt szerdára, ha ismerjük az esernyőhordozónak a mai napig történő összes megfigyelését. A visszatekintés az állapotnak egy jobb becslését adja, mint ami akkor elérhető volt, mivel több bizonyítékot használ fel.

- **Legvalószínűbb magyarázat (most likely explanation).** A megfigyelések egy sorozatának ismeretében lehet, hogy szeretnénk megtalálni azt az állapotsorozatot, ami a leginkább valószínű, hogy az adott megfigyeléseket generálta. Azaz szeretnénk kiszámítani az $\text{argmax}_{\mathbf{x}_{1:t}} P(\mathbf{x}_{1:t}|\mathbf{e}_{1:t})$ értékét. Például ha az esernyő feltűnik az első három nap mindegyikén, és hiányzik a negyediken, akkor a legvalószínűbb magyarázat az, hogy az első három napon esett és a negyediken nem esett. Az erre a feladatra szolgáló algoritmusok számos alkalmazásban hasznosak, ideértve a beszédfelismerést – ahol a cél a szavak legvalószínűbb sorozatának a megtalálása hangok sorozatának ismeretében – és egy zajos csatornán továbbított bináris szekvenciák rekonstrukcióját.

Ezeken a feladatokon túl szükség van még módszerekre az állapotátmenet- és érzékelő modellek megfigyelésekből történő *megtanulására*. Csakúgy, mint a statikus Bayes-hálóknál, a dinamikus Bayes-hálós tanulás elvégezhető mint a következtetés melléktermék. A következtetés egy becslést ad arra, hogy milyen átmenetek következtek be valójában, és milyen állapotok generálták az érzékelők értékeit, és ezek a becslések felhasználhatók a modell frissítésére. A frissített modellek jobb becsléseket adnak, és a folyamat a konvergálásig iterálódik. A teljes folyamat a várhatóérték-maximálás vagy **EM algoritmus** egy esete (lásd 20.3. alfejezet). Figyelemre méltó részlet, hogy a tanulás teljes simítós következtetést igényel szűrés helyett, mivel ez jobb becsléseket ad a folyamat állapotára. Lehet, hogy a szűréssel való tanulás nem konvergál helyesen; gondoljunk például a gyilkosságok felderítésének a megtanulására: a visszatekintés *mindig* szükséges annak kikövetkeztetéséhez a megfigyelhető bizonyítékok alapján, hogy mi történt a gyilkosság helyszínén.

Az előző szakaszban felsorolt négy következtetési feladatot megoldó algoritmusok először leírhatók általános szinten, függetlenül az alkalmazott modell konkrét típusától. Az egyes modellekhez igazodó javításokat a következő fejezetekben ismertetjük.

2.1. Szűrés és előrejelzés

Kezdjük a szűréssel. Megmutatjuk, hogy ez menet közben egyszerűen, online módon elvégezhető: a t-edik időpillanatig tartó szűrés eredményét ismerve, a $t + 1$-edik időpillanatra az eredmény könnyen kiszámítható az új $\mathbf{e}_{t+1}$ bizonyítékból. Azaz

$$P(X_{t+1}|e_{1:t+1}) = f(e_{t+1}, P(X_t|e_{1:t}))$$

valamely f függvénnyel. Ezt a folyamatot gyakran nevezik **rekurzív becslés**nek (**recursive estimation**). Tekinthetjük a számítást úgy, mint ami valójában két részből áll: elsőként a jelenlegi állapot eloszlását terjesztjük előre t-ről $t + 1$-re; azután fogjuk frissíteni, felhasználva az új $\mathbf{e}_{t+1}$ bizonyítékot. Ez a kétlépéses folyamat elég egyszerűen alakul:

$$P(X_{t+1}|e_{1:t+1}) = P(X_{t+1}|e_{1:t}, e_{t+1}) \text{ (felosztva a bizonyítékot)}$$

$$= \alpha\, P(e_{t+1}|X_{t+1}, e_{1:t})\, P(X_{t+1}|e_{1:t}) \text{ (a Bayes-szabályt használva)}$$

$$= \alpha\, P(e_{t+1}|X_{t+1})\, P(X_{t+1}|e_{1:t}) \text{ (a bizonyíték Markov-tulajdonsága miatt)}$$

Itt és végig a fejezetben, az α egy normalizációs konstans, ami a valószínűségek 1-re összegzését biztosítja. A második tényező $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$ reprezentálja a következő állapot egylépéses előrejelzését, és az első tényező ezt frissíti az új bizonyítékkal. Vegyük észre, hogy $\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})$ közvetlenül kinyerhető az érzékelő modellből. Most a jelenlegi $\mathbf{X}_t$ állapotot feltételnek véve a következő állapot egylépéses előrejelzéséhez jutunk:

$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) = \alpha \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t, \mathbf{e}_{1:t})\, P(\mathbf{x}_t|\mathbf{e}_{1:t}) = \alpha \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)\, P(\mathbf{x}_t|\mathbf{e}_{1:t})$$

(kihasználva a Markov-tulajdonságot)(15.3)

Az összegzésen belül az első tényező egyszerűen az állapotátmenet-modell és a második a jelenlegi állapot eloszlása. Így megvan a kívánt rekurzív képlet. A szűrt $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$ becslésre úgy gondolhatunk, mint egy $\mathbf{f}_{1:t}$

Created by XMLmind XSL-FO Converter.

„üzenetre”, amit előre terjesztünk végig a sorozaton, minden átmenetnél az új megfigyeléssel módosítva és frissítve. A folyamat

$f_{1:t+1} = \alpha$ Előre$(f_{1:t}, e_{t+1})$

ahol az ELŐRE a (15.3) egyenlet által leírt frissítést hajtja végre.

Fontos

Ha az összes állapotváltozó diszkrét, minden frissítés ideje állandó (azaz *t*-től független), és a tárigény is állandó. (Ezek az állandók természetesen függnek az állapottér méretétől és a tárgyalt időbeli modell konkrét típusától.) *A frissítés idő- és tárigényének állandónak kell lennie, ha egy korlátos memóriájú ágensnek követnie kell az aktuális állapot eloszlását a megfigyelések egy korlátlan sorozata esetén.*

Mutassuk be a szűrési folyamatot két lépésen keresztül az alap esernyős példában (lásd 15.2. ábra). Feltesszük, hogy a biztonsági őrünknek van valamilyen a priori hite, hogy a 0. napon esett-e, éppen azelőtt, hogy a megfigyelések sorozata elkezdődik. Tegyük fel, hogy ez $\mathbf{P}(R_0) = \langle 0{,}5,\ 0{,}5 \rangle$. Most a két megfigyelést a következőképpen dolgozzuk fel:

- Az 1. napon az esernyő feltűnik, így U_1 = *igaz*. Az előrejelzés t = 0-ról t = 1-re

$$\mathbf{P}(R_1) = \sum_{r_0} \mathbf{P}(R_1 | r_0) = \langle 0{,}7, 0{,}3 \rangle \times 0{,}5 + \langle 0{,}3, 0{,}7 \rangle \times 0{,}5 = \langle 0{,}5, 0{,}5 \rangle$$

és a t = 1-beli bizonyítékkal való frissítés azt adja, hogy

$$P(R_1 | u_1) = \alpha P(u_1 | R_1)\ P(R_1) = \alpha \langle 0{,}9,\ 0{,}2 \rangle \langle 0{,}5,\ 0{,}5 \rangle$$

$$= \alpha \langle 0{,}45,\ 0{,}1 \rangle \approx \langle 0{,}818,\ 0{,}182 \rangle$$

- A 2. napon az esernyő feltűnik, így U_2 = *igaz.* Az előrejelzés t = 1-ről t = 2-re

$$\mathbf{P}(R_2 | u_1) = \sum_{r_1} P(R_2 | r_1) P(r_1 | u_1) = \langle 0{,}7, 0{,}3 \rangle \times 0{,}818 + \langle 0{,}3, 0{,}7 \rangle \times 0{,}182 \approx \langle 0{,}627, 0{,}373 \rangle$$

a t = 2-beli bizonyítékkal való frissítés pedig azt adja, hogy

$$P(R_2 | u_1, u_2) = \alpha P(u_2 | R_2)\ P(R_2 | u_1) = \alpha \langle 0{,}9,\ 0{,}2 \rangle\ \langle 0{,}627,\ 0{,}373 \rangle$$

$$= \alpha \langle 0{,}565,\ 0{,}075 \rangle \approx \langle 0{,}883,\ 0{,}117 \rangle$$

Érzésünk szerint az eső valószínűsége azért növekedett az 1. napról a 2. napra, mert az eső folytatódik. A 15.2. (a) feladat ennek a tendenciának a további vizsgálatát kéri.

Az előrejelzés feladatát tekinthetjük egyszerűen szűrésnek új bizonyíték hozzáadása nélkül. Valójában a szűrési folyamat már magában foglal egy egylépéses előrejelzést és könnyen származtatható a következő rekurzív számítás a $t + k + 1$ állapot előrejelzésére a $t + k$ előrejelzéséből:

$$\mathbf{P}(\mathbf{X}_{t+k+1} | \mathbf{e}_{1:t}) = \sum_{\mathbf{x}_{t+k}} \mathbf{P}(\mathbf{X}_{t+k+1} | \mathbf{x}_{t+k}) P(\mathbf{x}_{t+k} | \mathbf{e}_{1:t}) \qquad (15.4)$$

Természetesen ez a számítás csak az állapotátmenet-modellt tartalmazza és nem az érzékelő modellt.

Érdekes meggondolni, hogy mi történik, ahogy egyre távolabbra próbálunk előre jelezni a jövőben. A 15.2. (b) feladat megmutatja, hogy az eső előrejelzett eloszlása a $\langle 0{,}5,\ 0{,}5 \rangle$ fix ponthoz konvergál, ami után a továbbiakban állandó marad. Ez az állapotátmenet-modell által meghatározott Markov-folyamat **stacionárius eloszlás**a (**stationary distribution**) (lásd még 5.2. szakasz - Következtetés Markov-lánc szimulációval rész). Rengeteg ismeret gyűlt fel az ilyen eloszlások tulajdonságairól és a **keverési idő**ről (**mixing time**) – ami nagyjából a fix pont eléréséig eltelt idő. Gyakorlati szempontból ez bukásra ítél minden olyan kísérletet, ami a keverési idő kis hányadánál nagyobb számú lépés múlva kísérli meg az aktuális állapot előrejelzését. Minél bizonytalanabb az állapotátmenet-modell, annál rövidebb a keverési idő, és a jövő annál homályosabb.

Created by XMLmind XSL-FO Converter.

A szűrésen és az előrejelzésen túl, felhasználhatjuk az előrehaladó rekurziót egy bizonyítéksorozat $P(\mathbf{e}_{1:t})$ **valószínűség**ének a kiszámítására is. Ez egy hasznos mennyiség, ha különböző időbeli modelleket szeretnénk összehasonlítani, amelyek ugyanazt a bizonyítéksorozatot állíthatták elő; például a 15.6. alfejezetben különböző szavakat hasonlítunk össze, amelyek ugyanazt a hangsort állíthatták elő. Ehhez a rekurzióhoz felhasználjuk az $\boldsymbol{\ell}_{1:t} = \mathbf{P}(\mathbf{X}_t, \mathbf{e}_{1:t})$ valószínűségi (likelihood) üzenetet. Egyszerű feladat azt megmutatni, hogy

$\boldsymbol{\ell}_{1:t+1} = \text{Előre}(\boldsymbol{\ell}_{1:t}, e_{t+1})$

Kiszámítva $\boldsymbol{\ell}_{1:t}$-t, az aktuális valószínűséget az $\mathbf{X}_t$ feletti összegzéssel kapjuk:

$$L_{1:t} = P(\mathbf{e}_{1:t}) = \sum_{\mathbf{x}_t} \boldsymbol{\ell}_{1:t}(\mathbf{x}_t) \qquad (15.5)$$

2.2. Simítás

Ahogy korábban meghatároztuk, a **simítás** (**smoothing**) múltbeli állapotok feletti eloszlás kiszámításának a folyamata a jelenig tartó bizonyítékok ismeretében; azaz $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ számítása valamely $0 \leq k < t$ esetén (lásd 15.3. ábra). Ez legkényelmesebben két részletben végezhető el – a bizonyíték k-ig és a bizonyíték k + 1-től t-ig,

$P(X_k|e_{1:t}) = P(X_k|e_{1:k}, e_{k+1:t})$

$= \alpha\ P(X_k|e_{1:k})\ P(e_{k+1:t}|X_k, e_{1:k})$ (a Bayes-szabályt használva)

$= \alpha\ P(X_k|e_{1:k})\ P(e_{k+1:t}|X_k)$ (a feltételes függetlenséget használva)

$= \alpha\ f_{1:k}\ b_{k+1:t}$ (15.6)

ahol definiáltuk a $\mathbf{b}_{k+1:t} = \mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k)$ „visszafelé” üzenetet, analóg módon az $\mathbf{f}_{1:k}$ előre üzenethez. Az $\mathbf{f}_{1:k}$ előre üzenetet előre szűréssel lehet kiszámolni 1-től k-ig, a (15.3) egyenlet alapján. A $\mathbf{b}_{k+1:t}$ visszafelé üzenetről pedig kiderül, hogy egy egyszerű rekurzív folyamattal kiszámítható, ami t-től visszafelé halad:

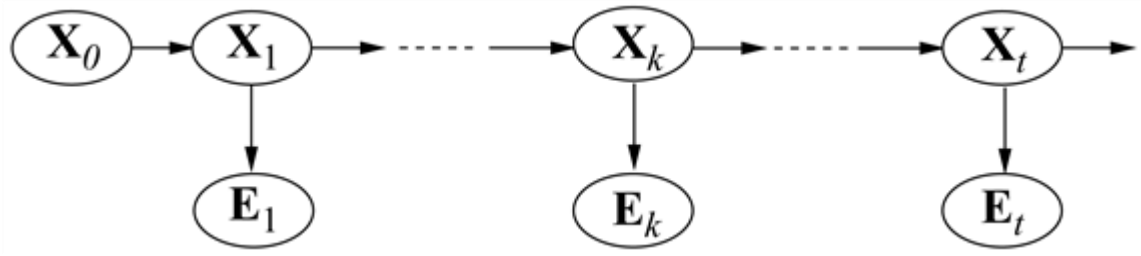
$$\begin{aligned}
\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) &= \sum_{\mathbf{x}_{k+1}} \mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k, \mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) && \text{(feltételnek véve } \mathbf{X}_{k+1}\text{ - et)} \\
&= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t}|\mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) && \text{(a feltételes függetlenség miatt)} \\
&= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}, \mathbf{e}_{k+2:t}|\mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) \\
&= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}|\mathbf{x}_{k+1})P(\mathbf{e}_{k+2:t}|\mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) && (15.7)
\end{aligned}$$

ahol az utolsó lépés az $\mathbf{e}_{k+1}$ és az $\mathbf{e}_{k+2:t}$ feltételes függetlenségéből következik az $\mathbf{x}_{k+1}$ melletti feltétellel. Az összegzésben a három tényezőből az első és a harmadik közvetlenül a modellből megkapható, a második pedig a „rekurzív hívás”. Az üzenetjelölést használva kapjuk, hogy

$b_{k+1:t} = \text{Hátra}(b_{k+2:t}, e_{k+1:t})$

ahol a HÁTRA a (15.7) egyenlet által leírt frissítést hajtja végre. Ahogy az előrefelé rekurziónál, az egyes frissítésekhez szükséges idő- és tárigény állandó, és így t-től független itt is.

15.3. ábra - A simítás egy múltbeli k időpontban az állapot a posteriori $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ eloszlását számítja ki egy teljes, 1-től t-ig terjedő megfigyelés sorozat ismeretében



Created by XMLmind XSL-FO Converter.

Most láthatjuk, hogy a (15.6) egyenlet mindkét tényezője kiszámítható időbeli rekurzióval, az egyik előrefelé fut 1-től k-ig, és a szűrés (15.3) egyenletét használja, a másik visszafelé fut t-től k+1-ig, és a (15.7) egyenletet használja. Figyeljünk arra, hogy a viszszafelé fázist $\mathbf{b}_{t+1:t} = \mathbf{P}(\mathbf{e}_{t+1:t}|\mathbf{X}_t) = \mathbf{1}$-gyel inicializáljuk, ahol **1** egy egyesekből álló vektor. (Mivel $\mathbf{e}_{t+1:t}$ egy üres sorozat, a megfigyelésének valószínűsége 1.)

Alkalmazzuk most ezt az algoritmust az esernyős példára, kiszámítva az eső valószínűségének simított becslését t = 1-nél, az első és második nap megfigyeléseinek ismeretében. A (15.6) egyenlet szerint ezt az adja meg, hogy

$$P(R_1|u_1, u_2) = \alpha\ P(R_1|u_1)\ P(u_2|R_1) \quad (15.8)$$

A korábban leírt előrefelé szűrési folyamatból már tudjuk, hogy az első tényező $\langle 0{,}818, 0{,}182\rangle$. A második tényezőt a (15.7) egyenletbeli visszafelé rekurziót alkalmazva számíthatjuk ki:

$$\mathbf{P}(u_2|R_1) = \sum_{r_2} P(u_2|r_2)P(r_2)\mathbf{P}(r_2|R_1) = (0{,}9\times 1\times\langle 0{,}7,\ 0{,}3\rangle) + (0{,}2\times 1\times\langle 0{,}3,\ 0{,}7\rangle) = \langle 0{,}69,\ 0{,}41\rangle$$

Ezt visszahelyettesítve a (15.8) egyenletbe azt kapjuk, hogy a simított becslés az esőre az 1. napon

$$P(R_1|u_1, u_2) = \alpha\ \langle 0{,}818,\ 0{,}182\rangle\ \times\ \langle 0{,}69,\ 0{,}41\rangle\ \approx\ \langle 0{,}883,\ 0{,}117\rangle$$

Így ebben az esetben a simított becslés nagyobb, mint a szűrt becslés (0,818). Ennek oka az, hogy az esernyő a 2. napon valószínűbbé teszi, hogy esett a 2. napon, és mivel az eső tartósan szokott esni, ez még valószínűbbé teszi, hogy esett az 1. napon.

Mind az előrefelé, mind a visszafelé haladó rekurzió lépésenként állandó időt igényel; így a simítás időkomplexitása $\mathbf{e}_{1:t}$ bizonyíték esetén $O(t)$. Ez egy konkrét k időlépésbeli simítás komplexitása. Ha a teljes szekvenciát szeretnénk simítani, nyilvánvaló módszer, hogy egyszerűen az egész simítási folyamatot lefuttatjuk minden egyes simítandó időpillanatra. Ez $O(t^2)$ időkomplexitást eredményez. Jobb megközelítés a dinamikai programozás nagyon egyszerű alkalmazásával $O(t)$-re csökkenti a komplexitást. A megoldás kulcsa megjelent az esernyős példa előző elemzésében, ahol az előrefelé szűrés folyamatának eredményeit újra fel tudtuk használni. A lineáris idejű algoritmus kulcsa így az előrefelé szűrés eredményeinek a tárolása az egész sorozatnál. Aztán a visszafelé rekurziót futtatjuk t-től 1-ig, kiszámítva a szűrt becslést minden k lépésnél a kiszámolt $\mathbf{b}_{k+1:t}$ visszafelé üzenetből és a tárolt $\mathbf{f}_{1:k}$ előrefelé üzenetből. Az algoritmus, amit találóan **előre-hátra algoritmus**nak (**forward-backward algorithm**) neveznek, a 15.4. ábrán látható.

15.4. ábra - Az előre-hátra algoritmus, ami állapotok egy sorozatának az a posteriori valószínűségeit számítja ki adott megfigyelési sorozat ismeretében. Az ELŐRE és a HÁTRA műveleteket rendre a (15.3) és (15.7) egyenletek definiálják.

```
function ELŐRE-HÁTRA(ev, prior) returns valószínűségeloszlások egy vektora
  inputs: ev, a bizonyíték (megfigyelés) értékek egy vektora az 1, ..., t lépéshez
          prior: a kezdő állapot P(X_0) a priori eloszlása
  local variables: fv, az előrefelé üzenetek vektora a 0, ..., t lépéshez
                   b, a hátrafelé üzenet egy reprezentációja, kezdetben csupa 1
                   sv, a simított becslések vektora az 1, ..., t lépéshez

  fv[0] ← prior
  for i = 1 to t do
      fv[i] ← ELŐRE(fv[i − 1], ev[i])
  for i = t downto 1 do
      sv[i] ← NORMALIZÁL(fv[i] × b)
      b ← HÁTRA(b, ev[i])
  return sv
```

A figyelmes olvasó észrevehette, hogy a 15.3. ábrán látható Bayes-hálóstruktúra egy polifa (polytree) a 14. fejezet szóhasználatában. Ez azt jelenti, hogy a csoportosító algoritmus egy nyilvánvaló alkalmazása szintén lineáris idejű algoritmust eredményez, ami a teljes sorozatra kiszámítja a simított becslést. Mára már világossá

Created by XMLmind XSL-FO Converter.

vált, hogy az előre-hátra algoritmus valójában a csoportosító eljárásokban használt polifa terjesztési algoritmusnak egy speciális esete (bár a kettőt egymástól függetlenül fejlesztették ki).

Az előre-hátra algoritmus alkotja a gerincét azon számítási módszereknek, amelyeket számos zajos megfigyelések sorozatával foglalkozó alkalmazásban használnak, a beszédfelismeréstől a repülőgépek radarkövetéséig. Ahogy leírtuk, két gyakorlati hátránya van. Az első, hogy a tárigénye túl nagy lehet azoknál az alkalmazásoknál, ahol az állapottér nagy, és a sorozatok hosszúak, mivel $O(|\mathbf{f}|t)$ méretű tárat használ, ahol $|\mathbf{f}|$ az előrefelé üzenet reprezentációjának a mérete. A tárigény $O(|\mathbf{f}|\log t)$-re csökkenthető az időkomplexitásnak egy $\log t$ tényezővel történő egyidejű megnövelése árán, ahogy a 15.3. feladat mutatja. Bizonyos esetekben (lásd 15.3. alfejezet) egy állandó tárigényű algoritmus használható időbüntetés nélkül.

Az alapalgoritmus második hátránya, hogy a folyamatos (online) működéshez módosítást igényel. Ekkor ugyanis a korábbi időpontokhoz simított becsléseket kell kiszámítanunk, amint folyamatosan új megfigyelések érkeznek a sorozat végéhez. A leggyakoribb követelmény az **állandó időkülönbségű simítás** (**fixed-lag smoothing**), ami a $\mathbf{P}(\mathbf{X}_{t-d}|\mathbf{e}_{1:t})$ simított becslés kiszámítását jelenti egy rögzített d-re. Azaz a simítást a jelenlegi t időpillanat előtt d lépéssel lévő időpontra végezzük el; t növekedését a simításnak is követnie kell. Nyilvánvaló, hogy lefuttathatjuk az előre-hátra algoritmust a d lépéses „ablakban”, amint egy új bizonyíték adódik, de ez nem tűnik hatékonynak. A 15.3. alfejezetben látni fogjuk, hogy az állandó időkülönbségű simításnál egy frissítés bizonyos esetekben állandó idő alatt megtehető, függetlenül a d időkülönbségtől.

2.3. A legvalószínűbb sorozat megtalálása

Tegyük fel, hogy a biztonsági őr esernyősorozata a munkája első öt napjában az [*igaz, igaz, hamis, igaz, igaz*]. Mi a legvalószínűbb időjárás-sorozat, ami ezt megmagyarázza? Az esernyő hiánya a 3. napon azt jelenti, hogy nem esett, vagy azt, hogy az igazgató elfelejtette elhozni? Ha a 3. napon nem esett, talán a 4. napon sem esik (mivel az időjárás nem változik gyorsan), de ekkor az igazgató puszta óvatosságból hozta az esernyőt. Összességében 2^5 lehetséges időjárás-sorozatot választhatunk. Létezik-e módszer a legvalószínűbb sorozat megkeresésére, az összes felsorolása nélkül?

Az egyik megközelítés, amit kipróbálhatunk a következő lineáris idejű algoritmus: használjuk a simító algoritmust, hogy megtaláljuk az időjárás a posteriori eloszlását minden időpillanatban; majd állítsuk elő a sorozatot minden egyes lépésnél az a posteriori szerinti legvalószínűbb időjárást felhasználva. Egy ilyen megközelítést kötelező gyanakvással kell fogadnia az olvasónak, mivel a szűrés által kiszámított a posteriori eloszlások az *egyes* időpontok feletti eloszlások, ezzel szemben a legvalószínűbb *sorozat* megtalálásához az összes időpont feletti *együttes* valószínűségeket kell figyelembe venni. Az eredmények valójában nagyon különbözők lehetnek (lásd 15.4. feladat).

Lineáris idejű algoritmus azonban *létezik* a legvalószínűbb sorozat megtalálására, de kicsit több gondolkozást igényel. Ugyanazon a Markov-tulajdonságon alapul, mint ami hatékony algoritmusokat eredményezett a szűrésre és a simításra. A problémáról való gondolkodás legkönnyebb módja, hogy ha minden sorozatot egy *útvonalnak* tekintünk egy gráfban, aminek a csomópontjai az egyes időpillanatokban a lehetséges *állapotok*. Egy ilyen gráf látható az esernyős problémára a 15.5. (a) ábrán. Most gondoljuk meg a gráfon keresztül vezető legvalószínűbb út megkeresésének a problémáját, ahol egy út valószínűsége (a likelihood érték) az útvonal menti átmenetek valószínűségeinek és az egyes állapotokban adott megfigyelések valószínűségeinek a szorzata. Koncentráljunk most azokra az útvonalakra, amelyek elérik az $Eső_5 = igaz$ állapotot. A Markov-tulajdonság miatt fennáll, hogy az $Eső_5 = igaz$ állapotba vezető legvalószínűbb útvonal tartalmazza azt a legvalószínűbb útvonalat, amely valamelyik 4. időpontbeli állapotba vezet, és amit egy átmenet követ az $Eső_5 = igaz$-ba; a 4. időpontbeli állapot pedig, ami része lesz az $Eső_5 = igaz$-ba vezető útnak, az az állapot, amelyik maximalizálja ennek az útvonalnak a valószínűségét.

Fontos

Másképpen fogalmazva, van egy *rekurzív kapcsolat az* $\mathbf{x}_{t+1}$ *állapotokba vezető legvalószínűbb útvonalak és az* $\mathbf{x}_t$ **állapotokba vezető legvalószínűbb útvonalak között**. Ez a kapcsolat olyan egyenletként írható fel, amely az útvonalak valószínűségeit kapcsolja össze:

$$\max_{\mathbf{x}_1 \ldots \mathbf{x}_t} \mathbf{P}(\mathbf{x}_1, \ldots, \mathbf{x}_t, \mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) = \alpha \mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \max_{\mathbf{x}_t} (\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{x}_t) \max_{\mathbf{x}_1 \ldots \mathbf{x}_{t-1}} P(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}, \mathbf{x}_t | \mathbf{e}_{1:t})) \quad (15.9)$$

Created by XMLmind XSL-FO Converter.

A (15.9) egyenlet *megegyezik* a szűrés (15.3) egyenletével, azzal a kivétellel, hogy

1. az $\mathbf{f}_{1:t} = \mathbf{P}(\mathbf{X}_t \mathbf{e}_{1:t})$ előre üzenet helyett a következő üzenet szerepel:

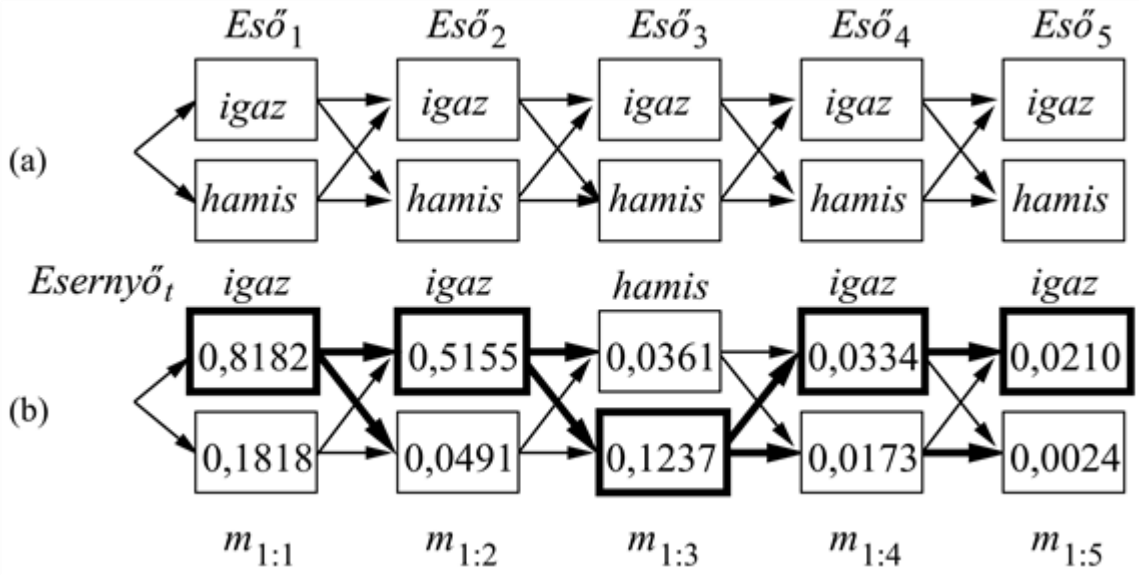
$$\mathbf{m}_{1:t} = \max_{\mathbf{x}_1 \ldots \mathbf{x}_{t-1}} \mathbf{P}(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}, X_t \,|\, \mathbf{e}_{1:t})$$

ami az $\mathbf{x}_t$ állapotba vezető legvalószínűbb út valószínűségeit adja meg; és

2. a (15.3) egyenletbeli $\mathbf{x}_t$ feletti összegzés helyett a (15.9) egyenletben az $\mathbf{x}_t$ feletti maximalizálás szerepel.

Így a legvalószínűbb állapotsorozat kiszámítása hasonló a szűréshez: előrefelé végigfut a sorozaton, minden időpontban kiszámítva az **m** üzeneteket a (15.9) egyenlet szerint. A számítás menetét a 15.5. (b) ábra mutatja. Végül ez kiadja az *összes* végső állapothoz vezető legvalószínűbb útvonal valószínűségét. Így már könnyen kiválasztható a teljes hosszúságú legvalószínűbb sorozat (a vastagon szedettel kiemelt végső állapottól). Az aktuális sorozat azonosításához – szemben azzal, amikor csak a valószínűségét számítjuk ki – az algoritmusnak minden állapotból mutatókat kell nyilvántartania a legjobb hozzá vezető állapothoz (vastagon szedettel látható); a sorozat a mutatóknak a legjobb végső állapottól való visszafelé követésével határozható meg.

15.5. ábra - (a) Az *Eső*$_t$ lehetséges állapotsorozatait tekinthetjük egy átvezető útnak egy gráfon, aminek csomópontjai a lehetséges állapotok az egyes időpontokban. (Az állapotokat négyzetes csomópont jelzi, hogy félreérthetetlenül megkülönböztessük őket egy Bayes-háló csomópontjaitól.) (b) A Viterbi-algoritmus működése az [*igaz, igaz, hamis, igaz, igaz*] megfigyelési sorozatra. Minden *t* időpontra feltüntettük az $\mathbf{m}_{1:t}$ üzenet értékeit, ami minden egyes *t* időpontbeli állapothoz megadja a legjobb, benne végződő sorozat valószínűségét. Minden egyes állapothoz egy vastag nyíl is vezet, ami a legjobb elődjét jelzi, a megelőző sorozat valószínűségének és az átmenet valószínűségének a szorzata szerint. A vastag nyilak visszafelé követése az $\mathbf{m}_{1:5}$-beni legvalószínűbb állapotból pedig megadja a legvalószínűbb sorozatot.



Az előzőleg leírt algoritmust **Viterbi-algoritmus**nak nevezik a megalkotója után. Hasonlóan a szűrési algoritmushoz, ennek a komplexitása is lineáris *t*-ben, a sorozat hosszában. Azonban eltérően a szűréstől, a tárigénye szintén lineáris *t*-ben. Ez azért van így, mert a Viterbi-algoritmusban mutatókkal kell nyilvántartani az egyes állapotokhoz vezető legjobb sorozatot.

3. Rejtett Markov-modellek

Az előző fejezet időbeli valószínűségi következtetésre szolgáló algoritmusokat vezetett be egy általános keretben, ami független volt az állapotátmenet- és az érzékelő modellek speciális formájától. Ebben és a

Created by XMLmind XSL-FO Converter.

következő két alfejezetben, olyan konkrétabb modelleket és alkalmazásokat tárgyalunk meg, amelyek bemutatják az alapalgoritmusok erejét, és bizonyos esetekben további tökéletesítéseket tesznek lehetővé.

A **rejtett Markov-modell**ekkel (**RMM**) (**hidden Markov model**, **HMM**) kezdjük. Egy RMM egy olyan időbeli valószínűségi modell, amelyben a folyamat állapotát egyetlen diszkrét valószínűségi változó írja le. [A modell karakterisztikus tulajdonsága tehát a rejtett állapot, azaz a (sztochasztikus) állapotfejlődés és annak passzív (sztochasztikus) megfigyelése. Az index időértelmezése helyett bármely szekvenciális értelmezés lehetséges, gyakori például az indexnek mint egy egydimenziós pozíciónak az értelmezése is – *a ford.*] A változó lehetséges értékei a világ lehetséges állapotai. Az előző fejezetben leírt esernyős példa ezért egy RMM, mivel csak egyetlen állapotváltozója van az Esőt. Az RMM keretei között további állapotváltozók csak úgy adhatók hozzá az időbeli modellhez, hogy az összes állapotváltozót egyetlen „megaváltozóba” kombináljuk, amelynek az értékei az egyes állapotváltozók értékeinek minden lehetséges kombinációja. Látni fogjuk, hogy az RMM-ek korlátozott struktúrája lehetővé teszi az összes alapalgoritmus egy nagyon egyszerű és elegáns mátrixos átfordítását.[3] A 15.6. alfejezet bemutatja az RMM-ek használatát a beszédfelismerésben.

3.1. Egyszerűsített mátrix algoritmusok

Egyetlen, diszkrét X_t állapotváltozó esetén konkrét forma adható az állapotátmenet-modell, az érzékelő modell és az előre és hátra üzenetek reprezentációira. Jelöljük az X_t állapotváltozó értékeit 1, …, S egészekkel, ahol S a lehetséges állapotok száma. A $\mathbf{P}(X_t \mid X_{t-1})$ állapotátmenet-modell ekkor egy $\mathbf{T}$ $S \times S$ mátrix, ahol

$$T_{ij} = P(X_t = j \mid X_{t-1} = i)$$

Azaz $\mathbf{T}_{ij}$ az átmenet valószínűsége i-ből j-be. Például az esernyős világban az állapotátmenet-mátrix

$$\mathbf{T} = \mathbf{P}(X_t \mid X_{t-1}) = \begin{pmatrix} 0,7 & 0,3 \\ 0,3 & 0,7 \end{pmatrix}$$

Az érzékelő modellt szintén mátrixalakra hozzuk. Ebben az esetben mivel az E_t bizonyítékváltozó értéke ismert, mondjuk e_t, így a modellnek csak azt a részét használjuk, ami az e_t megjelenésének valószínűségét meghatározza. Minden t időpontra konstruálunk egy $\mathbf{O}_t$ diagonális mátrixot, aminek az átlóbeli elemeit a $P(e_t|X_t = i)$ értékek adják, a többi értéke pedig 0. Például az esernyős világban az 1. napon az $U_1 = igaz$, így a 15.2. ábra szerint azt kapjuk, hogy

$$\mathbf{O}_1 = \begin{pmatrix} 0,9 & 0 \\ 0 & 0,2 \end{pmatrix}$$

Most oszlopvektort használva az előre és hátra üzenetek reprezentálására, a számítások egyszerű mátrix-vektor műveletekké válnak. A (15.3) előrefelé egyenlet ekkor

$$f_{1:t+1} = \alpha O_{t+1} T^{\top} f_{1:t} \quad (15.10)$$

és a (15.7) visszafelé egyenlet pedig

$$b_{k+1:t} = T O_{k+1} b_{k+2:t} \quad (15.11)$$

Ezekből az egyenletekből látható, hogy az előre-hátra algoritmus (lásd 15.4. ábra) időkomplexitása egy t hosszúságú sorozatra történő alkalmazásnál $O(S^2t)$, mivel minden lépés egy S elemű vektor szorzását igényli egy $S \times S$ mátrixszal. A tárigény pedig $O(St)$, mivel az előrefelé fázis t darab S méretű vektort tárol el.

Amellett hogy a mátrixos jelölés az RMM-ek esetében a szűrés és a simítás algoritmusaira egy elegáns leírásmódot kínál, ez javított algoritmusokra is jelez lehetőségeket. Az első az előre-hátra algoritmus egy egyszerű változata, ami simítás elvégzését teszi lehetővé *állandó* tárigény mellett, a sorozat hosszától függetlenül. Az ötlet az, hogy egy konkrét k időpillanatban a simítás mind az $\mathbf{f}_{1:k}$ előre, mind a $\mathbf{b}_{k+1:t}$ hátra üzenetek egyidejű jelenlétét igényli a (15.6) egyenlet szerint. Az előre-hátra algoritmus ezt úgy biztosítja, hogy az előrefelé fázisban tárolja az **f**-eket, hogy a hátrafelé fázisban elérhetők legyenek. Egy másik módszer szerint ez úgy érhető el, hogy egyetlen menetben mind **f**-et, mind **b**-t ugyanabban az irányban terjesztjük. Például az **f** „előre” üzenet terjeszthető hátrafelé, ha átrendezzük a (15.10) egyenletet, hogy a másik irányban működjön:

[3] A vektor- és mátrixművelekben járatlan olvasó számára hasznos lehet az A) függeléket átnézni, mielőtt a fejezetben továbbhaladna.

Created by XMLmind XSL-FO Converter.

$$\mathbf{f}_{1:t} = \alpha'(\mathbf{T}^{\top})^{-1}\mathbf{O}_{t+1}^{-1}\mathbf{f}_{1:t+1}$$

A módosított szűrési algoritmus úgy működik, hogy először lefut egy szabványos előre fázis, kiszámítva $\mathbf{f}_{t,t}$-t (elfelejtve az összes közbenső eredményt), majd lefut egy visszafelé fázis együtt **b**-re és **f**-re, kiszámítva a simított becslést minden időpontban. Mivel mindegyik üzenetnek csak egy példánya szükséges, a tárigény állandó (azaz független *t*-től, a sorozat hosszától). Az algoritmusra egyetlen jelentős megkötés vonatkozik: az állapotátmenet-mátrixnak invertálhatónak kell lennie, és az érzékelő modellben nem lehetnek nullák – azaz minden megfigyelésnek lehetségesnek kell lenni minden állapotban.

A második terület, ahol a mátrixjelölés javítást jelez, a menet közben történő (online) simítás állandó időkülönbözettel. Az a tény, hogy a simítás elvégezhető állandó tárigénnyel azt jelzi, hogy léteznie kell egy hatékony rekurzív algoritmusnak menet közbeni simításra – azaz olyan algoritmusnak amelynek az időkomplexitása független az időkülönbözet hosszától. Tegyük fel, hogy az időkülönbözet d; azaz a simítást $t - d$ időpontban végezzük, ahol t a jelenlegi időpont. A (15.6) egyenlet szerint ki kell számolnunk a $t - d$ időpontra az

$$\alpha\ f_{1:t-d}\ b_{t-d+1:t}$$

értékét. Majd amikor az új megfigyelés beérkezik, a $t - d + 1$ időpontra kell kiszámítanunk az

$$\alpha\ f_{1:t-d+1}\ b_{t-d+2:t+1}$$

értéket. Hogyan tehető ez meg inkrementálisan? Először is, az $\mathbf{f}_{1:t-d+1}$ kiszámítható az $\mathbf{f}_{1:t-d}$-ből a szabványos szűrés műveletét használva a (15.3) egyenlet szerint.

A visszafelé üzenet inkrementális kiszámítása trükkösebb, mivel nincs egyszerű kapcsolat a régi $\mathbf{b}_{t-d+1:t}$ visszafelé üzenet és az új $\mathbf{b}_{t-d+2:t+1}$ visszafelé üzenet között. Ehelyett, a régi $\mathbf{b}_{t-d+1:t}$ visszafelé üzenet és a sorozat kezdő $\mathbf{b}_{t+1:t}$ visszafelé üzenete közötti kapcsolatot fogjuk megvizsgálni. Ennek eléréséhez alkalmazzuk a (15.11) egyenletet d alkalommal, és azt kapjuk, hogy

$$\mathbf{b}_{t-d+1:t} = \left(\prod_{i=t-d+1}^{t} \mathbf{TO}_i\right)\mathbf{b}_{t+1:t} = \mathbf{B}_{t-d+1:t}\mathbf{1} \qquad (15.12)$$

ahol a $\mathbf{B}_{t-d+1:t}$ mátrix a **T** és az **O** mátrixok sorozatának a szorzata. A **B**-t felfoghatjuk egy „transzformációs operátornak”, ami egy későbbi visszafelé üzenetet egy korábbiba alakít. Egy hasonló egyenlet áll fenn az új visszafelé üzenetekre a következő megfigyelés beérkezése után:

$$\mathbf{b}_{t-d+2:t+1} = \left(\prod_{i=t-d+2}^{t+1} \mathbf{TO}_i\right)\mathbf{b}_{t+2:t+1} = \mathbf{B}_{t-d+2:t+1}\mathbf{1} \qquad (15.13)$$

Ha a (15.12) és a (15.13) egyenletben megvizsgáljuk a szorzatokat, láthatjuk, hogy egyszerű kapcsolat van közöttük: a második szorzathoz el kell „osztani” az első szorzatot $\mathbf{TO}_{t-d+1}$-gyel, és megszorozni az új utolsó elemmel $\mathbf{TO}_{t+1}$-gyel. Mátrixjelöléssel ekkor a régi és az új **B** mátrixok között egy egyszerű kapcsolat van:

$$\mathbf{B}_{t-d+2:t+1} = \mathbf{O}_{t-d+1:t}^{-1}(\mathbf{T})^{-1}\mathbf{B}_{t-d+1:t}\mathbf{TO}_{t+1} \qquad (15.14)$$

Ez az egyenlet a **B** mátrix egy inkrementális frissítését adja, amely viszont (a (15.3) egyenlet által) lehetővé teszi az új $\mathbf{b}_{t-d+2:t+1}$ visszafelé üzenet kiszámítását. A teljes algoritmus, ami az **f** és **B** tárolását és frissítését igényli, a 15.6. ábrán látható.

15.6. ábra - Egy állandó *d* lépésnyi időkülönbözettel simító algoritmus, folyamatos működésű (online) algoritmusként megvalósítva: egy új időpontbeli megfigyelésre kiadja az új simított becslést

Created by XMLmind XSL-FO Converter.

function ÁLLANDÓ-IDŐKÜLÖNBSÉGŰ-SIMÍTÁS(e_t, *rmm*, *d*) **returns** egy eloszlás $\mathbf{X}_{t-d}$ felett
  inputs: e_t, a jelenlegi bizonyíték a t időpontban
    rmm, egy rejtett Markov-modell $S \times S$ méretű **T** állapotátmenet-mátrixszal
    d, az időkülönbség nagysága a simításnál
  static: t, a jelenlegi idő, kezdetben 1
    f, egy valószínűség-eloszlás, a $\mathbf{P}(X_t|e_{1:t})$ előrefelé üzenet, kezdetben PRIOR[*rmm*]
    B, a d lépésnyit hátrafelé transzformáló mátrix, kezdetben az egységmátrix
    $e_{t-d:t}$, a bizonyítékok kétirányú listája $t-d$-től t-ig, kezdetben üres
  local variables: $\mathbf{O}_{t-d}$, $\mathbf{O}_t$, diagonális mátrixok az érzékelő modell információjával

  adjuk e_t-t az $e_{t-d:t}$ végéhez
  $\mathbf{O}_t \leftarrow$ egy $\mathbf{P}(e_t|X_t)$-t tartalmazó diagonális mátrix
  if $t > d$ **then**
    f $\leftarrow$ ELŐRE(**f**, e_t)
    töröld e_{t-d-1}-et az $e_{t-d:t}$ elejéről
    $\mathbf{O}_{t-d} \leftarrow$ egy $\mathbf{P}(e_{t-d}|X_{t-d})$-t tartalmazó diagonális mátrix
    $\mathbf{B} \leftarrow \mathbf{O}_{t-d}^{-1}\mathbf{T}^{-1}\mathbf{B}\mathbf{T}\mathbf{O}_t$
  else $\mathbf{B} \leftarrow \mathbf{B}\mathbf{T}\mathbf{O}_t$
  $t \leftarrow t + 1$
  if $t > d$ **then return** NORMALIZÁL(**f** $\times$ **B1**) **else return** semmi

4. Kalman-szűrők

Képzeljünk el egy kismadarat, amint a dzsungel sűrűjében repül szürkületkor: a mozgásának csupán rövid, szakaszos felvillanásait pillanthatjuk meg; minden igyekezetünkkel azt próbáljuk megjósolni, hogy hol van most a madár, és legközelebb hol fog felbukkanni, hogy nehogy szem elől tévesszük. Vagy képzeljük azt, hogy a második világháborúban radarkezelők vagyunk, egy gyenge, mozgó radarjelet kémlelve, ami 10 másodpercenként jelenik meg a képernyőn. Vagy, kissé még távolabbra visszalépve, képzeljük magunkat Kepler helyébe, ahogy a bolygók mozgását próbálja rekonstruálni igen pontatlan szögmérések sokaságából, amit rendszertelen és pontatlanul mért intervallumokban rögzítettek. Mindegyik esetben egy fizikai rendszer állapotát próbáljuk megbecsülni (helyet és sebességet például) időben egymást követő zajos megfigyelésekből. A problémát megfogalmazhatjuk egy időbeli valószínűségi modellben való következtetésként, ahol az állapotátmenet-modell a mozgás fizikáját írja le, az érzékelő modell pedig a mérési folyamatot. Ez a fejezet azokat a speciális reprezentációkat és következtetési algoritmusokat vizsgálja, amelyeket ilyen típusú problémákra fejlesztettek ki; a tárgyalt eljárást **Kalman-szűrés**nek (**Kalman filtering**) nevezik, a kidolgozója Rudolf E. Kalman[4] után.

Világos, hogy több *folytonos* változóra lesz szükségünk a rendszer állapotának megadásához. Például a madár repülése megadható egy (X, Y, Z) pozícióval és egy ($\dot{X}$, $\dot{Y}$, $\dot{Z}$) sebességgel minden egyes időpillanatban. Szükségünk lesz még alkalmas feltételes sűrűségfüggvényekre az állapotátmenet- és érzékelő modellekhez; a 14. fejezethez hasonlóan, **lineáris Gauss-eloszlás**okat (**linear Gaussian**) fogunk használni. Ez azt jelenti, hogy a következő $\mathbf{X}_{t+1}$ állapot a jelenlegi $\mathbf{X}_t$ állapot lineáris függvénye, amihez még egy Gauss-zaj adódik, amely feltétel a gyakorlatban igen elfogadhatónak bizonyul. Tekintsük például a madár X koordinátáját, pillanatnyilag figyelmen kívül hagyva a többi koordinátát. Legyen a megfigyelések közötti intervallum Δ, és tételezzünk fel állandó sebességet; ekkor a pozíció frissítésére az adódik, hogy

$$X_{t+\Delta} = X_t + \dot{X}\Delta$$

Ha Gauss-zajt adunk hozzá, akkor egy lineáris Gauss-féle állapotátmenet-modellt kapunk:

$$P(X_{t+\Delta} = x_{t+\Delta} \mid X_t = x_t, \dot{X}_t = \dot{x}_t) = N(x_t + \dot{x}_t\Delta, \sigma)(x_{t+\Delta})$$

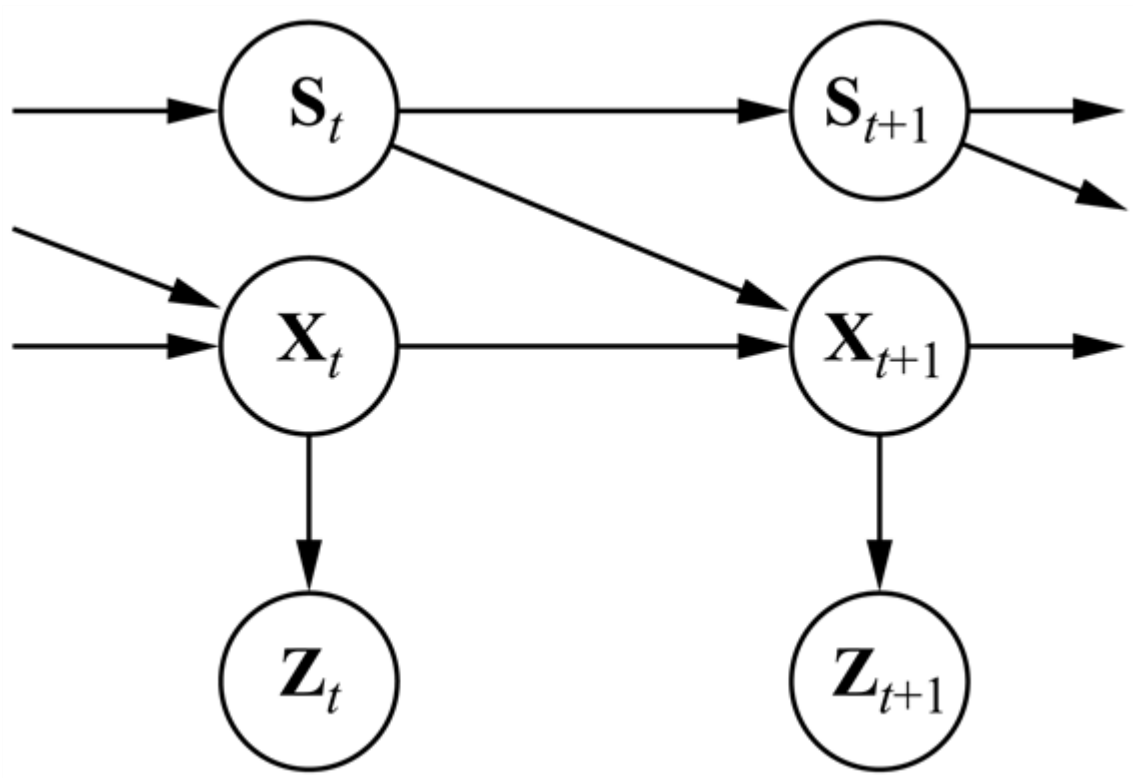
Az $\mathbf{X}_t$ pozíció és $\dot{\mathbf{X}}_t$ sebességvektorok alkotta rendszerhez tartozó Bayes-hálóstruktúra a 15.7. ábrán látható. Vegyük észre, hogy ez a lineáris Gauss-modellnek egy nagyon speciális alakja; az általános alakot a fejezetben

[4] Rudolf E. Kalman (Kálmán Rudolf) magyar származású matematikus.

Created by XMLmind XSL-FO Converter.

később írjuk le, ami alkalmazások nagyon széles körét fedi le az első bekezdés egyszerű mozgásos példáin túl. Az olvasónak hasznos lehet az A) függelékben átnézni a Gauss-eloszlások egyes matematikai tulajdonságait; a jelenlegi céljainkhoz a legfontosabb, hogy egy **többváltozós Gauss-eloszlás**t (**multivariate Gaussian**) d változó esetén egy d elemű $\boldsymbol{\mu}$ átlag és egy $d \times d$-s $\boldsymbol{\Sigma}$ kovarianciamátrix ad meg.

15.7. ábra - Egy $\mathbf{X}_t$ hely, $\dot{\mathbf{X}}_t$ sebesség és $\mathbf{Z}_t$ helymegfigyelés alkotta lineáris dinamikus rendszerhez tartozó Bayes-hálóstruktúra



4.1. Gauss-eloszlások frissítése

A 14. fejezetben hivatkoztunk a lineáris Gauss-eloszlások családjának egy alaptulajdonságára: az eloszláscsalád zárt a standard Bayes-hálóbeli műveletekre. Most ezt az állítást pontosítjuk az időbeli valószínűségi modellben végzett szűrés esetére. A megkövetelt tulajdonságok megfelelnek a szűrés (15.3) egyenletben megadott kétlépéses számításának:

1. Ha a jelenlegi $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$ eloszlás Gauss-eloszlás és a $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)$ állapotátmenet-modell lineáris Gauss-modell, akkor az egylépéses előrejelzés eloszlása

$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}) = \int_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t) P(\mathbf{x}_t|\mathbf{e}_{1:t}) d\mathbf{x}_t \qquad (15.15)$$

szintén Gauss-eloszlás.

2. Ha az előrejelzés $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$ eloszlása Gauss-eloszlás és a $\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})$ érzékelő modell lineáris Gauss-modell, akkor az új bizonyítékkal, mint feltétellel, a frissített eloszlás

$$P(X_{t+1}|e_{1:t+1}) = \alpha P(e_{t+1}|X_{t+1}) P(X_{t+1}|e_{1:t}) \quad (15.16)$$

szintén Gauss-eloszlás.

Így a Kalman-szűrés ELŐRE művelete fogad egy $_t$ átlaggal és $_t$ kovarianciamátrixszal meghatározott $\mathbf{f}_{1:t}$ Gauss előre üzenetet, és előállít egy új többváltozós $\mathbf{f}_{1:t+1}$ Gauss előre üzenetet $\boldsymbol{\mu}_{t+1}$ átlaggal és $\boldsymbol{\Sigma}_{t+1}$ kovarianciamátrixszal. Így, ha egy $\mathbf{f}_{1:0} = \mathbf{P}(\mathbf{X}_0) = N(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ Gauss-eloszlással indulunk, egy lineáris Gauss-modellel való szűrés az állapotokon Gauss-eloszlást eredményez minden időpontban.

Created by XMLmind XSL-FO Converter.

Fontos

Ez tetszetős és elegáns eredménynek tűnik, de miért is olyan fontos? Ennek a magyarázata az, hogy a most tárgyalt esethez hasonló néhány speciális esetet kivéve, *a szűrés folytonos vagy hibrid (diszkrét és folytonos) hálókkal olyan állapoteloszlásokat generál, amelyek reprezentációja az idővel korlátlanul nő*. Ezt az állítást általában nem könnyű bizonyítani, de a 15.5. feladat egy egyszerű példán mutatja be, hogy mi történik.

4.2. Egy egyszerű egydimenziós példa

Azt állítottuk, hogy a Kalman-szűrés ELŐRE művelete egy Gauss-eloszlást egy új Gauss-eloszlásba visz át. Ez új átlag- és kovarianciamátrix kiszámítását jelenti az előző átlagból és kovarianciamátrixból. Az általános (többváltozós) eset frissítési szabályának származtatása igen sok lineáris algebrai lépést igényel, így egyelőre a nagyon egyszerű egyváltozós esetnél maradunk, és később adjuk meg az eredményeket az általános esetre. A számítások még az egyváltozós esetre is unalmasak kissé, de úgy érezzük, hogy érdemes látni őket, mivel a Kalman-szűrő hasznossága olyan szorosan kötődik a Gauss-eloszlások matematikai tulajdonságaihoz.

A tárgyalt időbeli modell egyetlen folytonos X_t állapotváltozós **véletlen bolyongást (random walk)** ír le egy zajos Z_t megfigyeléssel. Egy példa lehet erre a „vásárlói bizalom” mutatója, amit modellezhetünk úgy, mint ami havonként egy véletlen Gauss-eloszlású változáson megy át, és amelyet egy véletlen vásárlói kérdőív mér fel, ami szintén Gauss-mintavételi zajt okoz. Az a priori eloszlást gaussinak tételezzük fel σ_0^2 szórásnégyzettel:

$$P(x_0) = \alpha e^{-\frac{1}{2}\left(\frac{(x_0-\mu_0)^2}{\sigma_0^2}\right)}$$

(Az egyszerűség kedvéért ebben a fejezetben ugyanazt az α szimbólumot fogjuk használni minden normalizációs állandó jelölésére.) Az állapotátmenet-modell egyszerűen a jelenlegi állapot egy Gauss-eloszlású módosítását jelenti állandó σ_x^2 szórásnégyzettel:

$$P(x_{t+1}|x_t) = \alpha e^{-\frac{1}{2}\left(\frac{(x_{t+1}-x_t)^2}{\sigma_x^2}\right)}$$

Az érzékelő modell egy Gauss-eloszlású zajt tételez fel σ_z^2 szórásnégyzettel:

$$P(z_t|x_t) = \alpha e^{-\frac{1}{2}\left(\frac{(z_t-x_t)^2}{\sigma_z^2}\right)}$$

Most, az a priori $P(X_0)$ eloszlás ismeretében kiszámíthatjuk az egylépéses előrejelzés eloszlását felhasználva a (15.15) egyenletet:

$$P(x_1) = \int_{-\infty}^{\infty} P(x_1|x_0)P(x_0)dx_0 = \alpha\int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(\frac{(x_1-x_0)^2}{\sigma_x^2}\right)} e^{-\frac{1}{2}\left(\frac{(x_0-\mu_0)^2}{\sigma_0^2}\right)} dx_0 = \alpha\int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(\frac{\sigma_0^2(x_1-x_0)^2+\sigma_x^2(x_0-\mu_0)^2}{\sigma_0^2\sigma_x^2}\right)} dx_0$$

Az integrál igen bonyolultnak néz ki. A továbbhaladás lehetőségét annak észrevétele jelenti, hogy a kitevő két olyan kifejezés összege, amelyek négyzetesek x0-ban, és így az maga is négyzetes x0-ban. Egy egyszerű trükk, amit teljes négyzetté kiegészítésként (completing the square) ismerünk, lehetővé teszi bármely $ax_0^2 + bx_0 + c$ átírását egy $a\left(x_0 - \frac{-b}{2a}\right)^2$ négyzetes tag és egy $c - \frac{b^2}{4a}x_0$ x_0-tól független maradék tag összegévé. A maradék tag az integrálból kivihető, így azt kapjuk, hogy

$$P(x_1) = \alpha e^{-\frac{1}{2}\left(c-\frac{b^2}{4a}\right)}\int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(a(x_0-\frac{-b}{2a})^2\right)} dx_0$$

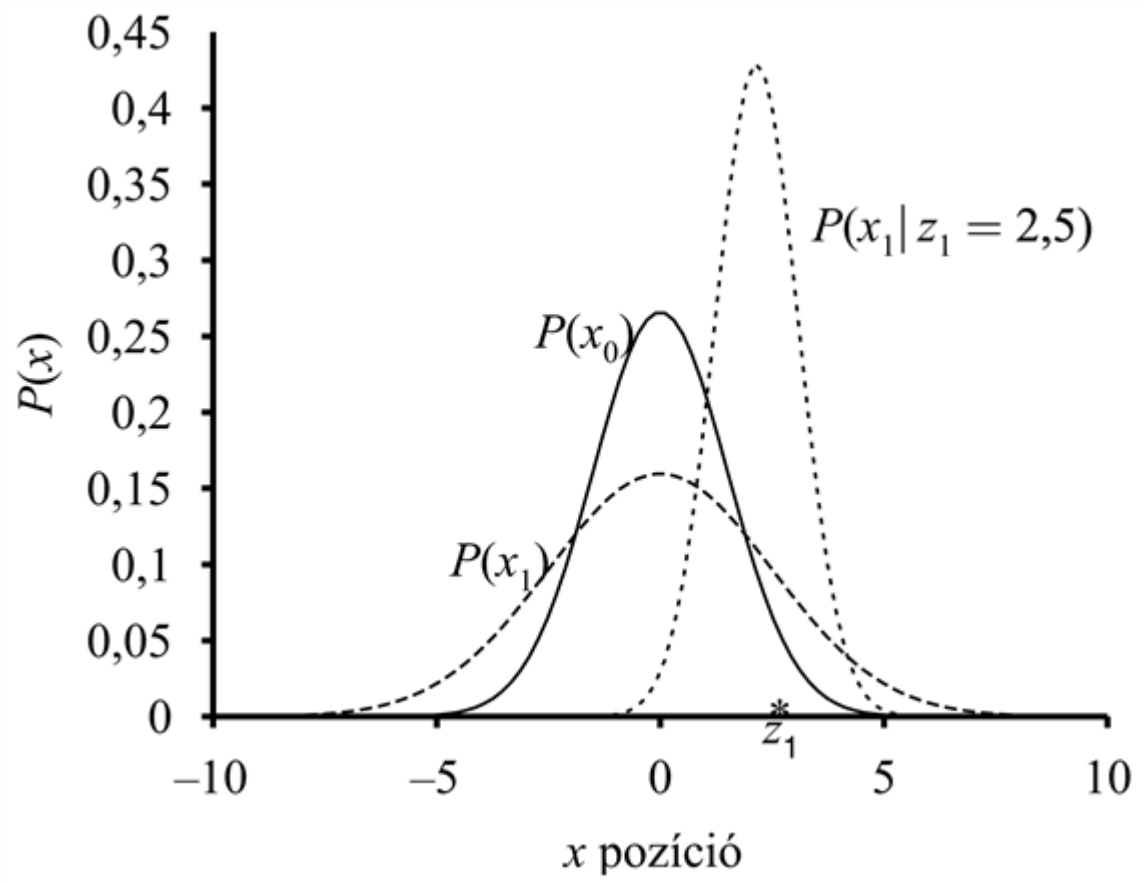
Created by XMLmind XSL-FO Converter.

Most az integrál pontosan egy Gauss-eloszlás teljes tartomány feletti integrálja, ami egyszerűen 1-et ad. Így csupán a maradék tag maradt a négyzetesből.

A második kulcslépés annak észrevétele, hogy a maradék tagnak x_1-ben négyzetesnek kell lennie; valóban, egyszerűsítés után azt kapjuk, hogy

$$P(x_1) = \alpha e^{-\frac{1}{2}\left(\frac{(x_1-\mu_0)^2}{\sigma_0^2+\sigma_x^2}\right)}$$

15.8. ábra - A Kalman-szűrés frissítési ciklusának a lépései egy véletlen bolyongás esetén. A véletlen bolyongás a priori eloszlása $\mu_0 = 0{,}0$ és $\sigma_0 = 1{,}0$, az átmenet bizonytalansága $\sigma_x = 2{,}0$, az érzékelő bizonytalansága $\sigma_z = 1{,}0$, az első megfigyelés pedig $z_1 = 2{,}5$ (az *x* tengelyen csillaggal jelölve).Vegyük észre, ahogy a $P(x_1)$ előrejelzés ellapul a $P(x_0)$-hoz képest az átmenet bizonytalansága miatt. Vegyük azt is észre, hogy az a posteriori $P(x_1|z_1)$ kissé balra helyezkedik el a z_1 megfigyeléstől, mivel az átlag az előrejelzés és a megfigyelés súlyozott átlaga.



Azaz az egylépéses előrejelzés eloszlása Gauss-eloszlás, ugyanazzal a μ_0 átlaggal, a szórásnégyzet pedig egyenlő az eredeti σ_0^2 szórásnégyzetnek és az átmenet σ_x^2 szórásnégyzetének az összegével. Egy pillanatnyi belegondolás után ez szemléletesen is elfogadhatónak tűnik.

A frissítés lépésének befejezéséhez még szükséges, hogy az első időpontbeli megfigyelést, x_1-t feltételként vegyük számításba. A (15.16) egyenletből erre az adódik, hogy

$$P(x_1|z_1) = \alpha P(z_1|x_1)P(x_1) = \alpha e^{-\frac{1}{2}\left(\frac{(z_1-x_1)^2}{\sigma_z^2}\right)} e^{-\frac{1}{2}\left(\frac{(x_1-\mu_0)^2}{\sigma_0^2+\sigma_x^2}\right)}$$

Created by XMLmind XSL-FO Converter.

Most ismét kombináljuk a kitevőket, és négyzetté egészítsük ki (15.6 feladat), azt kapva, hogy

$$P(x_1|z_1) = \alpha e^{-\frac{1}{2}\left(\frac{(x_1 - \frac{(\sigma_0^2+\sigma_x^2)z_1 + \sigma_z^2\mu_0}{\sigma_0^2+\sigma_x^2+\sigma_z^2})}{(\sigma_0^2+\sigma_x^2)\sigma_z^2/(\sigma_0^2+\sigma_x^2+\sigma_z^2)}\right)} \qquad (15.17)$$

így egy frissítési ciklus után egy új Gauss-eloszlásunk van az állapotváltozóra.

A (15.17) egyenletben a Gauss-alakból láthatjuk, hogy az új átlag és szórás kiszámítható a régi átlagból és szórásból a következőképpen:

$$\mu_{t+1} = \frac{(\sigma_t^2 + \sigma_x^2)z_{t+1} + \sigma_z^2\mu_t}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2}$$

$$\sigma_{t+1}^2 = \frac{(\sigma_t^2 + \sigma_x^2)\sigma_z^2}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2} \qquad (15.18)$$

A 15.8. ábrán látható egy frissítési ciklus az állapotátmenet- és az érzékelő modell konkrét értékei esetén.

Az előző egyenletpár pontosan ugyanazt a szerepet játssza, mint az *általános szűrés egyenlete* (15.3) vagy az *RMM-szűrés egyenlete* (15.10). A Gauss-eloszlások speciális tulajdonsága miatt azonban az egyenleteknek van néhány további érdekes tulajdonsága. Először is, hogy az új μ_{t+1} átlag kiszámítása értelmezhető úgy, mint egyszerűen az új z_{t+1} megfigyelés és a régi μ_t átlag *súlyozott átlaga*. Ha a megfigyelés megbízhatatlan, akkor σ_z^2 nagy, és több figyelmet szentelünk a régi átlagnak; ha a régi átlag megbízhatatlan ($_t^2$ nagy), vagy a folyamat nehezen megjósolható (σ_t^2 nagy), akkor több figyelmet szentelünk a megfigyelésnek. Másodszor, vegyük észre, hogy a σ_{t+1}^2 szórásnégyzet frissítése *független a megfigyeléstől*. Ezért előre kiszámíthatjuk, hogy mi lesz a szórásnégyzetértékek sorozata. Harmadszor, a szórásnégyzetértékek sorozata gyorsan konvergál egy adott értékhez, ami csak σ_x^2 -től és σ_z^2 -től függ, ezáltal lényegesen egyszerűsítve az elkövetkező számításokat (lásd 15.7. feladat).

4.3. Az általános eset

Az előző levezetés szemléletesen bemutatta a Gauss-eloszlásoknak azt az alaptulajdonságát, ami a Kalman-szűrés működését lehetővé teszi: azt a tényt, hogy az exponens négyzetes alakú. Ez nem csak az egyváltozós esetre igaz; a teljes többváltozós Gauss-eloszlás alakja:

$$N(\boldsymbol{\mu}, \boldsymbol{\Sigma})(\mathbf{x}) = \alpha e^{-\frac{1}{2}\left((\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right.}$$

A kitevőben lévő tényezők összeszorzása láthatóvá teszi, hogy a kitevő is négyzetes függvénye az **x**-ben lévő x_i valószínűségi változóknak. Ahogy az egyváltozós esetben, a szűrési frissítés megőrzi az állapoteloszlás gaussi voltát.

Elsőként definiáljuk a Kalman-szűrésnél használt általános időbeli modellt. Mind az állapotátmenet-modell, mind az érzékelő modell lineáris transzformációt enged meg additív Gauss-zajjal. Így azt kapjuk, hogy

$$P(x_{t+1}|x_t) = N(Fx_t, \Sigma_x)(x_{t+1})$$

$$P(z_t|x_t) = N(Hx_t, \Sigma_z)(z_t) \quad (15.19)$$

ahol az **F** és a $\boldsymbol{\Sigma}_x$ mátrixok a lineáris állapotátmenet-modellt és az átmeneti zaj kovarianciáját írják le, a **H** és a $\boldsymbol{\Sigma}_z$ pedig az érzékelő modell megfelelő mátrixai. Ekkor az átlag és a kovariancia frissítésének az egyenletei a maguk rémisztő valóságában:

$$\mu_{t+1} = F\mu_t + K_{t+1}(z_{t+1} - HF\mu_t)$$

Created by XMLmind XSL-FO Converter.

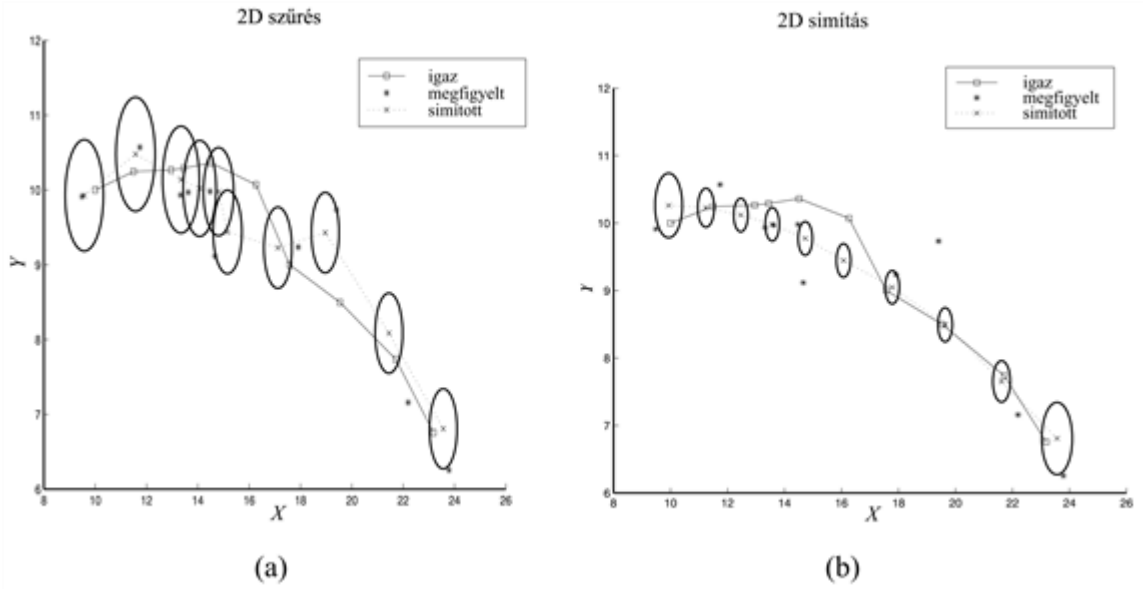
$$\Sigma_{t+1} = (I - K_{t+1})(F\Sigma_t F^\top + \Sigma_x) \quad (15.20)$$

ahol a $\mathbf{K}_{t+1} = (\mathbf{F}\boldsymbol{\Sigma}_t\mathbf{F}^\top + \boldsymbol{\Sigma}_x)\mathbf{H}^\top(\mathbf{H}(\mathbf{F}\boldsymbol{\Sigma}_t\mathbf{F}^\top + \boldsymbol{\Sigma}_x)\mathbf{H}^\top + \boldsymbol{\Sigma}_z)^{-1}$ mennyiség neve a **Kalman-erősítésmátrix** (**Kalman gain matrix**). Akár hihető, akár nem, ezeknek az egyenleteknek lehetséges egy szemléletes értelmezése. Például, gondoljuk meg a $\boldsymbol{\mu}$ állapotátlag-becslés frissítését. Az $\mathbf{F}\boldsymbol{\mu}_t$ tag a t+1 időpontban előre jelzett állapot, így $\mathbf{HF}\boldsymbol{\mu}_t$ az előre jelzett megfigyelés. Ezért a $\mathbf{z}_{t+1} - \mathbf{HF}\boldsymbol{\mu}_t$ tag az előre jelzett megfigyelés hibáját reprezentálja. Ez van megszorozva a $\mathbf{K}_{t+1}$ tényezővel az előre jelzett állapot korrigálásához; így a $\mathbf{K}_{t+1}$ annak mértéke, hogy mennyire kell figyelembe venni az új megfigyelést az előrejelzéshez képest. Ahogyan a (15.18) egyenletben, az a tulajdonság most is fennáll, hogy a szórásnégyzet frissítése független a megfigyeléstől. A $\boldsymbol{\Sigma}_t$ és a $\mathbf{K}_t$ értékek sorozata ezért előre (offline) is kiszámolható, és a követés közben szükséges konkrét számítások igen mérsékeltek.

Hogy bemutassuk az egyenletek működését egy X–Y síkon mozgó tárgy követésének a problémájára alkalmaztuk őket. Az állapotváltozók az $\mathbf{X} = \left(X, Y, \dot{X}, \dot{Y}\right)^\top$, így az $\mathbf{F}$, $\boldsymbol{\Sigma}_x$, $\mathbf{H}$ és $\boldsymbol{\Sigma}_z$ 4 × 4-es mátrixok. A 15.9. (a) ábrán látható az igazi pályagörbe, a zajos megfigyelések sorozata és a Kalman-szűréssel becsült pályagörbe végig a kovarianciával, amit az egységnyi szórás méretű körvonalak jeleznek. A szűrési folyamat jól teljesít az aktuális mozgás követésében, és ahogy várható, a szórásnégyzet gyorsan beáll egy rögzített értékre.

Ahogyan szűrésre, úgy *simításra* is származtathatók egyenletek lineáris Gauss-modell esetén. A simítás eredményei a 15.9. (b) ábrán láthatók. Vegyük észre, hogy a helyzetbecslés szórásnégyzete gyorsan lecsökken, a pályagörbe végét leszámítva (itt miért nem?), és hogy a simítással becsült pályagörbe sokkal egyenletesebb.

15.9. ábra - (a) A Kalman-szűrés eredménye egy *X*–*Y* síkon mozgó objektumnál, feltüntetve a valódi (balról jobbra haladó) pályagörbét, a zajos megfigyelések egy sorozatát és a Kalman-szűrés alapján becsült pályagörbét. A helybecslések szórásait az oválisok jelzik. (b) A Kalman-simítás eredménye ugyanarra a megfigyelési sorozatra.



4.4. A Kalman-szűrés alkalmazhatósága

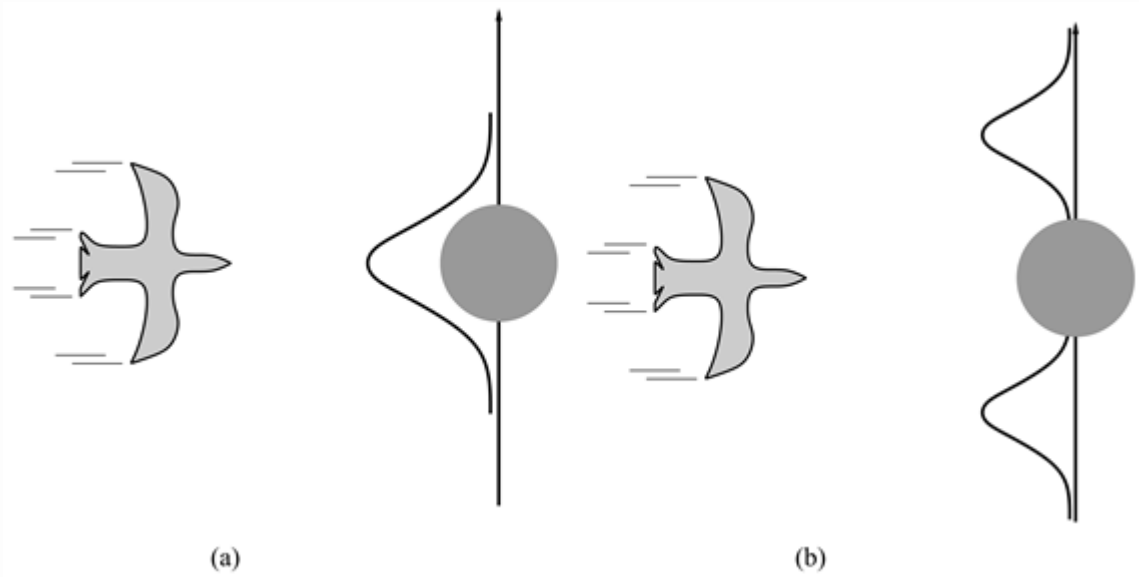
A Kalman-szűrést és kiterjesztéseit alkalmazások széles körében használják. A „klaszszikus” alkalmazás légi járművek és rakéták radar követése. Kapcsolódó alkalmazások között van a tengeralattjárók és földi járművek akusztikai követése, és járművek és emberek képi követése. Egy kissé elvontabb szemlélettel a Kalman-szűrőket részecskék buborékkamrás pályagörbéjének rekonstrukciójára, valamint óceáni áramlások felszíni műholdképek alapján történő rekonstrukciójára használják. Az alkalmazások köre sokkal szélesebb, mint csupán mozgások követése: bármely rendszerre alkalmazható, ami folytonos állapotváltozókkal és zajos megfigyelésekkel jellemezhető. Ilyen rendszerek magukban foglalnak zúzdákat, vegyi üzemeket, nukleáris reaktorokat, növényi ökoszisztémákat és nemzetgazdaságokat.

Created by XMLmind XSL-FO Converter.

A Kalman-szűrés alkalmazhatóságának ténye egy rendszerre nem jelenti, hogy az eredmények érvényesek vagy hasznosak lesznek. A feltevések – lineáris Gauss-féle állapotátmenet-modell és érzékelő modell – igen erősek. A **kiterjesztett Kalman-szűrő** (**KKSZ**) (**extended Kalman filter**, **EKF**) megpróbál úrrá lenni a modellezett rendszer nemlineáris tulajdonságain. Egy rendszer nemlineáris, ha az állapotátmenet-modell nem írható le, mint az állapotvektor mátrixszorzata, ahogyan ez a (15.19) egyenletben szerepel. A KKSZ úgy működik, hogy rendszert $\mathbf{x}_t$-ben *lokálisan* lineárisként modellezi az $\mathbf{x}_t = \boldsymbol{\mu}_t$ környezetében, ahol $\boldsymbol{\mu}_t$ a jelenlegi állapoteloszlás átlaga. Ez jól működik „sima”, „jó magaviseletű” rendszereknél, és lehetővé teszi a követőnek, hogy egy olyan Gauss-állapoteloszlást tartson nyilván és frissítsen, ami elfogadható közelítése az igazi a posteriori eloszlásnak.

Azonban mit is jelent, hogy egy rendszer „nem sima” vagy „rossz magaviseletű”? Technikailag ez azt jelenti, hogy jelentős nemlinearitás van jelen a rendszer viselkedésében abban a régióban, ami „közel” van a jelenlegi $\boldsymbol{\mu}_t$ átlaghoz (a $\boldsymbol{\Sigma}_t$ kovarianciamátrix szerint). Ennek a tulajdonságnak a nem technikai megértéséhez gondoljunk arra a példára, amikor egy dzsungelben szálló madár követésével próbálkoztunk. A madár nagy sebességgel egyenesen egy fatörzs felé tart. A Kalman-szűrő, akár reguláris, akár kiterjesztett a madár pozíciójára, csak Gauss-előrejelzést adhat, és ennek a Gaussnak az átlaga a fatörzs közepére esik, ahogyan a 15.10. (a) ábrán látható. A madár egy elfogadható modellje azonban egy elkerülő műveletet jelezne előre egyik vagy másik oldalra, ahogy az a 15.10. (b) ábrán látható. Egy ilyen modell erősen nemlineáris, mivel a madár döntése nagyon eltérő a fatörzshöz vett pontos pozíciójának függvényében.

15.10. ábra - Egy fa felé repülő madár (felülnézetben). (a) Egy Kalman-szűrő előrejelzése a madár helyzetére, ami egyetlen Gauss-eloszlás az akadály közepére illesztve. (b) Egy valósághűbb modell számításba veszi a madár elkerülő manővereit, és azt jelzi előre, hogy az egyik vagy a másik oldalon fog elszállni.



Ilyen példák kezeléséhez nyilvánvalóan egy kifejezőbb nyelvre van szükségünk a modellezett rendszer viselkedésének a reprezentálásához. A szabályozáselmélet területén, ahol olyan problémák, mint például egy repülőgép elkerülő manőverezése ugyanilyen típusú bonyodalmakat vetnek fel, a megszokott megoldás a **váltó Kalman-szűrő** (**switching Kalman filter**). Ebben a megközelítésben, több Kalman-szűrő fut párhuzamosan, mindegyik a rendszer különböző modelljét használva – például egy az egyenes repülésre, egy az éles balra fordulásra és egy az éles jobbra fordulásra. Az előrejelzéseknek egy súlyozott összegét használjuk, ahol a súly attól függ, hogy mennyire illeszkednek az egyes szűrők az aktuális adatokhoz. A következő fejezetben látni fogjuk, hogy ez egyszerűen az általános dinamikus Bayes-háló modell egy speciális esete, amit a 15.7. ábrán látható hálóból egy diszkrét „manőver” állapotváltozó hozzáadásával nyerünk. A váltó Kalman-szűrőket a 15.5. feladat tárgyalja tovább.

5. Dinamikus Bayes-hálók

Egy **dinamikus Bayes-háló** vagy **DBH** (**dynamic Bayesian network**, **DBN**) egy olyan Bayes-háló, ami egy a 15.1. alfejezetben leírt típusú időbeli valószínűségi modellt reprezentál. (Ahogy korábban is említettük az index időbeli értelmezése helyett bármely szekvenciális értelmezés lehetséges, gyakori például az indexnek mint egy

Created by XMLmind XSL-FO Converter.

egydimenziós helyzeti pozíciónak az értelmezése is – *a ford.*) A DBH-kra már láttunk példákat, a 15.2. ábrán az esernyős hálót és a 15.7. ábrán a Kalman-szűrő hálót. Általában egy DBH minden egyes szeletének tetszőleges számú állapotváltozója ($\mathbf{X}_t$) és bizonyítékváltozója ($\mathbf{E}_t$) lehet. Az egyszerűség kedvéért fel fogjuk tenni, hogy a változók és kapcsolataik szeletről szeletre pontosan ismétlődnek, és hogy a DBH egy elsőrendű Markov-folyamatot reprezentál, így minden változónak csak a saját szeletében vagy a közvetlenül megelőző szeletben lehetnek szülei.

Fontos

Nyilvánvaló, hogy minden rejtett Markov-modell reprezentálható mint egy DBH, egyetlen állapotváltozóval és egyetlen bizonyítékváltozóval. Az is fennáll, hogy minden diszkrét változós DBH reprezentálható mint egy RMM; ahogyan a 15.3. alfejezetben megmutattuk, a DBH összes állapotváltozója összekombinálható egyetlen állapotváltozóvá, aminek az értékei az egyes állapotváltozók értékeinek az összes lehetséges együttese. Azonban ha minden RMM egy DBH, és minden DBH átfordítható egy RMM-be, akkor mi a különbség? A különbség abban rejlik, hogy *egy komplex rendszer állapotának az őt alkotó változókra történő dekomponálásával a DBH képes kihasználni az időbeli valószínűségi modell ritkaságát*. Tegyük fel például, hogy egy DBH-nak 20 bináris állapotváltozója van, amelyek mindegyikének három szülője van az előző szeletben. Ekkor a DBH állapotátmenet-modellje $20 \times 2^3 = 160$ valószínűségi értéket tartalmaz, míg a hozzátartozó RMM-nek 2^{20} állapota és ezért 2^{40}, azaz durván egybillió átmenet-valószínűség értéke van az állapotátmenet-mátrixban. Ez legalább három ok miatt is rossz: először, hogy az RMM maga sokkal több tárat igényel, másodszor, hogy a hatalmas állapotátmenet-mátrix az RMM-következtetést sokkal költségesebbé teszi, harmadszor, hogy ilyen hatalmas számú paraméter megtanulásának problémája a tiszta RMM-modelleket alkalmazhatatlanná teszi nagy problémák esetén. A DBH-k és az RMM-ek közötti kapcsolat nagyjából analóg a hagyományos Bayes-hálók és a teljes táblázatos együttes eloszlások közötti kapcsolathoz.

Már megmutattuk, hogy minden Kalman-szűrő reprezentálható egy DBH-ban folytonos változókkal és lineáris Gauss feltételes eloszlásokkal (lásd 15.7. ábra). Az előző fejezet végén tárgyaltak miatt azzal is tisztában kell lenni, hogy *nem* minden DBH reprezentálható egyetlen Kalman-szűrő modellel. Egy Kalman-szűrőnél az aktuális állapoteloszlás mindig egy egyedülálló többváltozós Gauss-eloszlás – azaz egyetlen „dudor” egy konkrét helyen. A DBH-k ezzel szemben tetszőleges eloszlást képesek modellezni. Számos valós alkalmazásnál ez a rugalmasság elengedhetetlen. Gondoljuk meg például a kulcsaim jelenlegi helyét. Lehetnek a zsebeimben, az éjjeli szekrényen, a konyhapulton vagy a bejárati ajtóban himbálódzva. Egyetlen Gauss-dudornak, ami ezt az összes helyet magában foglalja, jelentős valószínűséget kell rendelni ahhoz, hogy a kulcsok az előszobában a levegőben vannak. A valós világ olyan aspektusai, mint céltudatos ágensek, akadályok és zsebek „nemlinearitásokat” vezetnek be, amelyek diszkrét és folytonos változók kombinációját igénylik, hogy egy elfogadható modellhez jussunk.

5.1. DBH-k létrehozása

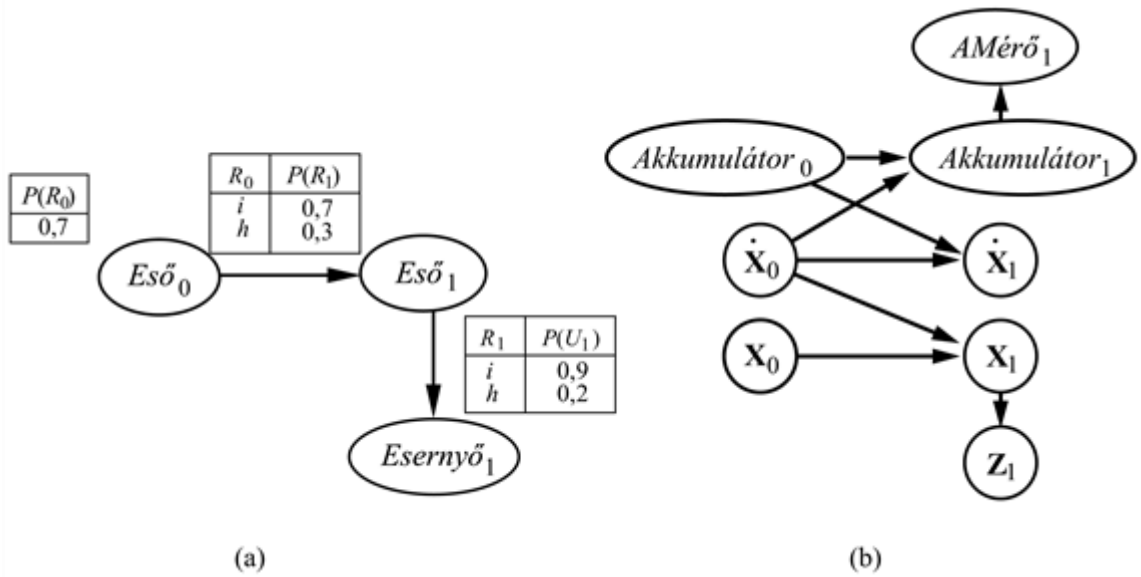
Egy DBH létrehozásához háromfajta információt kell megadni: a $\mathbf{P}(\mathbf{X}_0)$ a priori eloszlást az állapotváltozók felett; a $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{X}_t)$ állapotátmenet-modellt; és a $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$ érzékelő modellt. Az állapotátmenet- és érzékelő modell megadásához, meg kell adni az egymást követő szeletek közötti és az állapot- és bizonyítékváltozók közötti kapcsolatok topológiáját. Mivel az állapotátmenet- és érzékelő modelleket stacionáriusnak – minden t-re azonosnak – tételezzük fel, a legkényelmesebb egyszerűen az első szeletre megadni őket. Például az esernyős világ teljes DBH-specifikációját megadja az a három csomópontos háló, ami a 15.11. (a) ábrán látható. Ebből a specifikációból a teljes (egyik irányban végtelen) DBH megkonstruálható az első szelet megfelelő másolásával.

Gondoljunk most át egy érdekesebb példát: egy akkumulátorhajtású, az X–Y síkon mozgó robot követesét, ahogyan azt a 15.1. alfejezetben bevezettük. Elsőként szükségünk van állapotváltozókra, amelyek mind az $\mathbf{X}_t = (X_t, Y_t)$ pozíció-, mind az $\dot{\mathbf{X}}_t = (\dot{X}_t, \dot{Y}_t)$ sebességkomponenseket magukban foglalják. Feltételezünk valamilyen pozícióméró módszert – akár egy rögzített kamerát vagy egy felszíni GPS-t (Global Positioning System) –, ami a $\mathbf{Z}_t$ méréseket eredményezi. A pozíció a következő időpontban a jelenlegi pozíciótól és sebességtől függ, ugyanúgy, mint a standard Kalman-szűrő modellben. A sebesség a következő időpontban a jelen sebességtől és az akkumulátor állapotától függ. Felveszünk még egy $Akkumulátor_t$ változót az aktuális akkumulátor töltési szintjének reprezentálására, aminek szülői az előző akkumulátorszint és a sebesség, és felveszünk még egy $AMérő_t$ változót, ami az akkumulátor töltési szintjét méri. Ez szolgáltatja a 15.11. (b) ábrán látható alapmodellt.

Created by XMLmind XSL-FO Converter.

Érdemes részletesebben megvizsgálni az $AMérő_t$-hez tartozó érzékelőmodell tulajdonságait. Tegyük fel az egyszerűség kedvéért, hogy mind az $Akkumulátor_t$, mind az $AMérő_t$ diszkrét értékeket vehetnek fel 0-tól 5-ig – igen hasonlóan egy tipikus laptop számítógép akkumulátormérőjéhez. Ha a mérő mindig pontos akkor a $\mathbf{P}(AMérő_t|Akkumulátor_t)$ FVT-nek végig 1,0 valószínűségeket kell tartalmaznia az „átlójában” és 0,0 valószínűségeket máshol. A valóságban a mérésekben a zaj mindig felbukkan. Folytonos mérésekre, egy kis szórású Gauss-eloszlást lehetne ehelyett használni.[5] A diszkrét változóinknál a Gauss-eloszlást egy olyan eloszlással közelíthetjük, amelyben a hiba valószínűsége a megfelelő módon csökken, így nagy hiba valószínűsége igen kicsi. A **Gauss-hibamodell** (**Gaussian error model**) kifejezést fogjuk használni mind a folytonos, mind a diszkrét változatokra.

15.11. ábra - (a) Az esernyős DBH-hoz tartozó a priori eloszlás, az állapotátmenet-modell és az érzékelő modell megadása. Az összes következő szelet feltevésünk szerint az 1. szelet másolata. (b) Egy *X*–*Y* síkon való robotmozgáshoz tartozó egyszerű DBH.



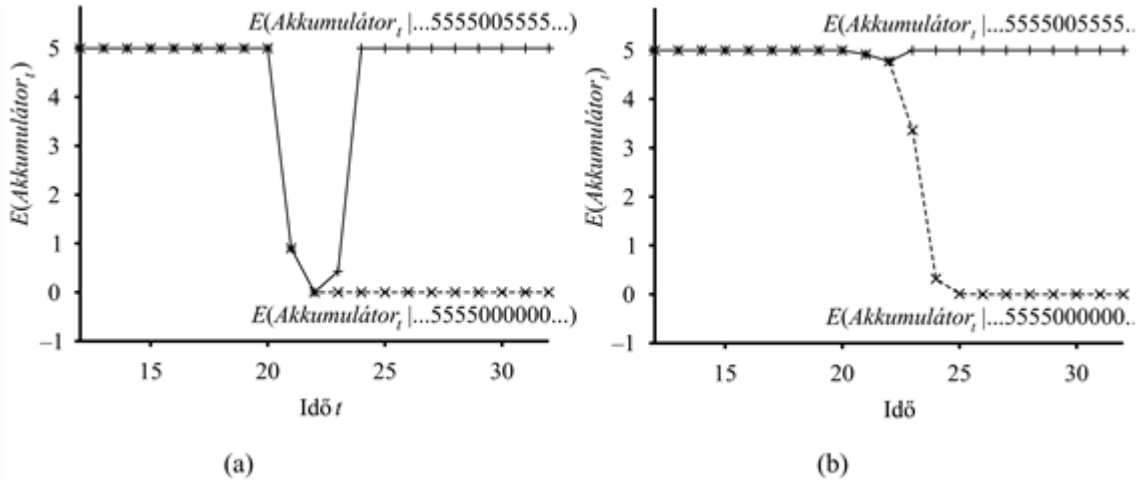
A robotikában bárki, aki egy kevés gyakorlattal rendelkezik a számítógépes folyamatvezérlésben vagy az automatikus érzékelés más eseteiben, rögtön tanúskodik arról a tényről, hogy a kis mennyiségű mérési hiba gyakran a legkevésbé fontos a problémák között. Ugyanis a valódi érzékelők elromlanak, és amikor egy érzékelő elromlik, akkor nem küld szükségszerűen egy jelet azzal, hogy „Ó, egyébiránt az adat, amit küldök, egy halom zagyvaság.” Ehelyett egyszerűen küldi a zagyvaságot. A meghibásodás legegyszerűbb formáját **átmeneti hibá**nak (**transient failure**) nevezik, amikor az érzékelő alkalmanként elhatározza, hogy valami zagyvaságot küld. Például az akkumulátor szintérzékelőjének lehet olyan szokása, hogy nullát küld, amikor valaki meglöki a robotot, még akkor is, ha az akkumulátor teljesen fel van töltve.

Lássuk, mi történik, amikor egy átmeneti hiba bekövetkezik egy Gauss-hibamodellnél, ami nem tudja kezelni az ilyen hibákat. Tételezzük fel például, hogy a robot csendesen üldögél, és 20 egymást követő mérés az akkumulátoron 5-öt ad. Ezután az akkumulátormérő időleges kiesése miatt a következő mérés eredménye $AMérő_{21} = 0$. Mire kell következtetnünk egy egyszerű Gauss-hibamodellből az $Akkumulátor_{21}$-gyel kapcsolatban. A Bayes-szabály szerint a válasz függ mind a $\mathbf{P}(AMérő_{21} = 0|Akkumulátor_{21})$ érzékelő modelltől, mind a $\mathbf{P}(Akkumulátor_{21}|AMérő_{1:20})$ előrejelzéstől. Ha egy nagy érzékelési hiba valószínűsége szignifikánsan kisebb, mint az $Akkumulátor_{21} = 0$-ba való átmenet valószínűsége, még ha ez utóbbi nagyon valószínűtlen is, akkor az a posteriori eloszlás nagy valószínűséget fog rendelni a lemerült akkumulátorhoz. Egy második nulla értékű mérési eredmény $t = 22$-nél ezt a konklúziót majdnem biztossá teszi. Ha az átmeneti hiba aztán eltűnik, és a jelzett érték visszaáll 5-re $t = 23$-tól, akkor az akkumulátorszint becslése gyorsan vissza fog állni 5-re, szinte varázsütésre. Ezt az eseménysort mutatja be a 15.12. (a) ábra felső görbéje, ami az $Akkumulátor_t$ várható értékét mutatja idő szerint egy diszkrét Gauss-hibamodellt használva.

[5] Szigorúan véve egy Gauss-eloszlás problematikus, mivel pozitív valószínűséget rendel nagy negatív töltési szintekhez. A **béta-eloszlás** gyakran jobb választás az olyan változókra, amelyeknek tartománya korlátos.

Created by XMLmind XSL-FO Converter.

15.12. ábra - (a) Felső görbe: az Akkumulátort várható értékének a pályagörbéje egy olyan megfigyelési sorozatnál, ami csak 5-ösöket tartalmaz, kivéve a 0-kat a t = 21 és a t = 22 időpontokban, egy egyszerű Gauss-hibamodell felhasználásával. Alsó görbe: a pályagörbe, amikor a megfigyelések 0-val folytatódnak t = 21-től. (b) Ugyanez a kísérlet az átmenetihiba-modellel elvégezve. Vegyük észre, hogy az átmeneti hibát jól kezeli, de a végleges hiba túlzott pesszimizmust eredményez.



Fontos

A helyreállás ellenére előfordul (t = 22-nél), hogy a robot meg van győződve, hogy az akkumulátora lemerült; feltehetően ekkor ki kell küldenie egy segítség jelet, és le kell kapcsolnia. Sajnos a túlegyszerűsített érzékelő modellje tévútra vezette. Hogyan lehet ezt kijavítani? Gondoljunk egy ismerős példára az emberek mindennapos vezetési gyakorlatából: éles kanyarokban vagy meredek hegyeken néha felvillan az „üzemanyagtartály üres” figyelmeztető lámpa. A segélykérő telefon keresése helyett inkább az szokott az ember eszébe jutni, hogy az üzemanyagmérő gyakran igen nagy hibákat ad, amikor az üzemanyag lötyög a tartályban. A történet tanulsága a következő: *az érzékelő hibájának helyes kezeléséhez a rendszer érzékelő modelljének tartalmaznia kell a meghibásodás lehetőségét*.

Egy érzékelő legegyszerűbb hibamodellje egy bizonyos valószínűséggel megengedi, hogy az érzékelő teljesen helytelen értéket adjon vissza, függetlenül a világ valódi állapotától. Például ha az akkumulátormérő 0-t visszaadva hibázik, azt mondhatjuk, hogy

$$P(AMérő_t = 0 | Akkumulátor_t = 5) = 0{,}03$$

ami feltételezhetően sokkal nagyobb, mint az a valószínűség, amit az egyszerű Gauss-hibamodell rendelne hozzá. Hívjuk ezt az **átmenetihiba-modell**nek (**transient failure model**). Hogyan segít ez, amikor a 0 jelzéssel kerülünk szembe? Feltételezve, hogy az az *előre jelzett* valószínűség, hogy az akkumulátor lemerült, az eddigi jelzések alapján sokkal kisebb, mint 0,03, akkor az $AMérő_{21} = 0$ megfigyelés legjobb magyarázata az, hogy az érzékelő időlegesen meghibásodott. Szemléletesen úgy képzelhetjük el a bizonyosságot az akkumulátor feltöltöttségéről, mint aminek van egy bizonyos mennyiségű „tehetetlensége”, ami segít úrrá lenni a mérőjelzés időleges zavarain. A 15.12. (b) ábra felső görbéjén látható, hogy az átmenetihiba-modell képes átmeneti hibákat kezelni a bizonyosság katasztrofális változásai nélkül.

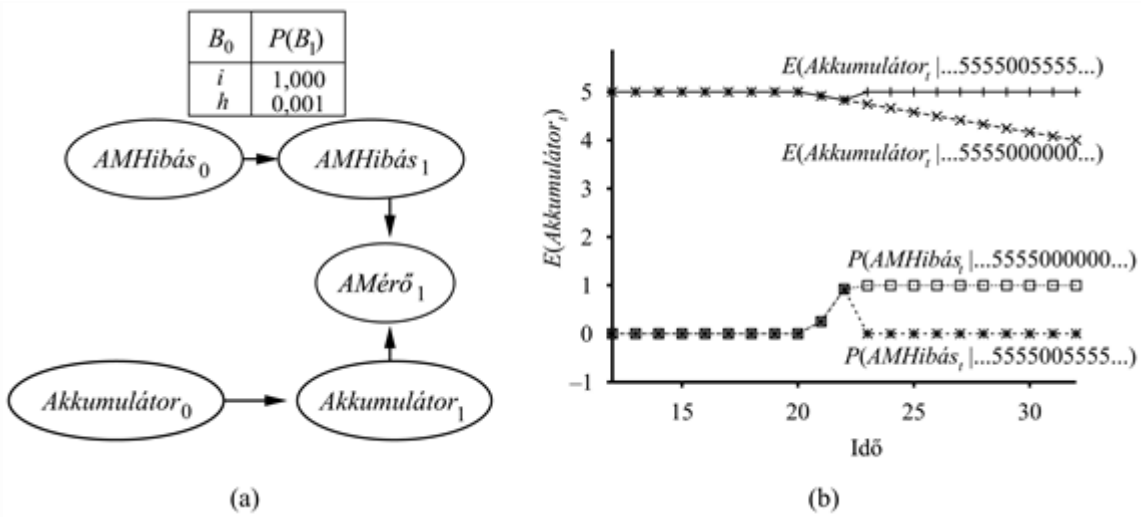
Ennyit hát az időleges zavarokról. Mi a helyzet az érzékelő állandó meghibásodásával? Sajnos, az ilyenfajta meghibásodások túlságosan is gyakoriak. Ha az érzékelő húsz 5-ös jelzést ad, amit húsz 0-s jelzés követ, akkor az előző bekezdésben leírt átmenetihiba-modell azt fogja eredményezni, hogy a robot fokozatosan azt kezdi hinni, hogy az akkumulátora lemerült, amikor valójában az is lehet, hogy az érzékelő hibásodott meg. A 15.12. (b) ábrán látható alsó görbe mutatja a bizonyosság „pályagörbét” erre az esetre. A t = 25 időpontra – 5 darab 0 jelzésre – a robot meg van győződve, hogy az akkumulátora üres. Nyilvánvaló, hogy jobban szeretnénk, ha a robot azt hinné, hogy az akkumulátor mérője romlott el – ha valóban ez a valószínűbb esemény.

Created by XMLmind XSL-FO Converter.

Nem meglepő, hogy egy tartós hiba kezeléséhez szükségünk lesz egy **tartóshiba-modell**re (**persistent failure model**), ami leírja, hogyan működik az érzékelő normális körülmények között és meghibásodás után. Ennek eléréséhez ki kell egészítenünk a rendszer rejtett állapotát egy további változóval, mondjuk egy *AMHibás*-sal, ami megadja az akkumulátormérő állapotát. A hiba tartós megmaradását egy nyíllal kell modellezni, ami az $AMHibás_0$-t az $AMHibás_1$-hez kapcsolja. Ennek a **megmaradási nyíl**nak (**persistence arc**) olyan FVT-je van, ami minden időlépésben megengedi a meghibásodást egy kis valószínűséggel, mondjuk 0,001-gyel, de leírja, hogy ha egy érzékelő elromlott, akkor az rossz is marad. Amikor az érzékelő rendben van, akkor az *AMérő* érzékelő modellje megegyezik az átmenetihiba-modellel; amikor az érzékelő rossz, az *AMérő* mindig 0, függetlenül az akkumulátor aktuális töltöttségétől.

Az akkumulátor érzékelőjének tartóshiba-modellje a 15.13. (a) ábrán látható. A teljesítménye két adatsoron (az időleges zavarjelnél és a tartós meghibásodáson) a 15.13. (b) ábrán látható. Számos dolgot érdemes megfigyelni ezeken a görbéken. Először is, hogy az időleges zavarjelnél az érzékelő hibájának a valószínűsége jelentősen megemelkedik a második 0 jelzés után, de azonnal visszaesik, ahogy egy 5-öst megfigyelt. Másodszor, hogy tartós hibánál az érzékelő hibájának a valószínűsége gyorsan megemelkedik 1 közelébe, és ott marad. Végül, hogy amikortól az érzékelőt hibásnak hiszi, a robot csak azt tételezheti fel, hogy az akkumulátor „normális” sebességgel veszti el töltését, ahogy az $E(Akkumulátor_t|...)$ fokozatosan csökkenő szintje ezt mutatja.

15.13. ábra - (a) Az érzékelő állapotváltozóját bemutató DBH-részlet. Ez a változó szükséges az akkumulátormérő végleges hibájának a modellezésére. (b) Felső görbék: az Akkumulátort várható értékének a pályagörbéje az „átmeneti hibájú” és a „tartós hibájú” megfigyelési sorozatoknál. Alsó görbék: az AMHibás valószínűségének pályagörbéi a két megfigyelési sorozatnál.



Eddig csupán felületesen érintettük a komplex folyamatok reprezentálásának problémáját. Az állapotátmenet-modellek változatossága óriási, magában foglalva egymástól olyan távol eső témákat, mint például az emberi endokrin rendszer (hormonális működés) és az autópályákon közlekedő járművek modellezését. Az érzékelő modellezés egymagában is hatalmas terület, de még az olyan nehezen megfogható jelenségek, mint például az érzékelő elállítódása, hirtelen dekalibrálása és a külső körülményeknek (mint az időjárásnak) a hatásai az érzékelő jelzésein is kezelhetők explicit reprezentációkkal a dinamikus Bayes-hálóknál.

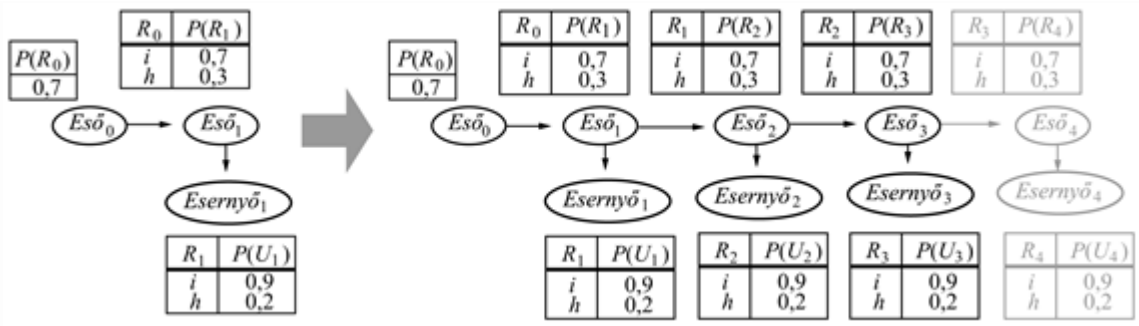
5.2. Egzakt következtetés DBH-kban

Miután felvázoltunk néhány ötletet komplex folyamatok DBH-kkal történő reprezentálásáról, most a következtetés kérdését vizsgáljuk meg. Bizonyos értelemben ezt a kérdést már megválaszoltuk: a dinamikus Bayes-hálók Bayes-hálók, és nekünk már vannak algoritmusaink Bayes-hálóban történő következtetésre. Adott megfigyeléssorozat esetén a teljes Bayes-hálós reprezentáció felépíthető a szeletek megismétlésével, ameddig a háló elég nagy nem lesz a megfigyelések befogadásához, ahogy a 15.14. ábrán látható. Ezt a technikát **kibontás**nak (**unrolling**) nevezik. (Technikailag a DBH ekvivalens egy egy irányban végtelen hálóval, amit végtelen számú kibontással kapunk. Az utolsó megfigyelés után hozzáadott szeleteknek nincs hatása a következtetésre a megfigyelési periódusban, ezért elhagyhatók.) Ha a DBH-t már kibontottuk, alkalmazhatók a következtető algoritmusok: változó eliminálás, egyesítési fa módszere stb. (lásd 14. fejezet).

Created by XMLmind XSL-FO Converter.

Sajnos a kibontás naiv alkalmazása nem volna valami hatékony. Ha szűrést vagy simítást szeretnénk elvégezni megfigyelések $\mathbf{e}_{1:t}$ hosszú sorozatán, akkor a kibontott háló tárigénye $O(t)$ lenne, és korlát nélkül nőne, amint még több megfigyelést adnánk hozzá. Ráadásul ha egyszerűen újra lefuttatjuk a következtető algoritmust minden időpontban, amikor megfigyelés érkezik, a következtetési ideje frissítésenként szintén $O(t)$ szerint fog növekedni.

15.14. ábra - Egy dinamikus Bayes-háló kibontása: a szeletek ismétlődnek befogadva a megfigyelési sorozatot (a halvány csomópontok). További szeleteknek már nincs hatása a megfigyelési perióduson belüli következtetésre. Ha a DBH-t kibontottuk, bármely következtetési algoritmus használható – változó eliminálás, egyesítési fa módszerek és így tovább –, ami a 14. fejezetben szerepel.



A 15.2. alfejezetre visszatekintve látjuk, hogy állandó idő- és tárigény elérhető a szűrési frissítésnél, ha a számítások rekurzív módon végezhetők. Lényegében a szűrési frissítés (15.3) egyenlete az előző időponthoz tartozó állapotváltozók *kiösszegzését (summing out)* végzi el, hogy megkapja az új időponthoz tartozó eloszlást. Változók kiösszegzése pedig pontosan az, amit a **változóeliminálás** (**variable elimination**) algoritmusa elvégez (lásd 14.10. ábra), és megmutatható, hogy a változó eliminálás időrendi változókon futva pontosan a szűrés rekurzív frissítésének (15.3) egyenlet szerinti működését utánozza. A módosított algoritmus egyszerre legfeljebb két szeletet tart a memóriában: kezdve a 0. szelettel, hozzáadjuk az 1. szeletet, kiösszegezzük a 0. szeletet, aztán hozzáadjuk a 2. szeletet, aztán kiösszegezzük az 1. szeletet és így tovább. Ezzel a módszerrel elérhető az állandó idő- és tárigény a szűrési frissítésnél. (Ugyanez a teljesítmény elérhető az egyesítési fa algoritmusának megfelelő módosításaival.) A 15.10. feladat kéri ennek a ténynek az ellenőrzését az esernyős hálóra.

Eddig tartottak a jó hírek, most következnek a rosszak: megmutatható, hogy egy frissítés idő- és tárkomplexitásának az „állandója” majdnem minden esetben exponenciális az állapotváltozók számában. Az történik ugyanis, hogy amint a változó eliminálás halad, a tényezők megnőnek, magukban foglalva az összes állapotváltozót (vagy pontosabban azokat az állapotváltozókat, amelyeknek van szülőjük az előző időszeletben). A maximális tényezőméret $O(d^{n+1})$, a frissítés költsége pedig $O(d^{n+2})$.

Fontos

Természetesen ez sokkal kevesebb, mint az RMM frissítésének költsége, ami $O(d^{2n})$, de nagyszámú változónál még mindig kivitelezhetetlenül nagy. Ezt a súlyos tényt nehéz elfogadni. Ugyanis a következőt jelenti: *annak ellenére, hogy a DBH-k felhasználhatók olyan nagyon komplex időbeli folyamatok reprezentálására, amelyek számos, ritkán összekapcsolt változóval rendelkeznek, nem vagyunk képesek hatékonyan és pontosan következtetni ezekről a folyamatokról*. A DBH-modell, ami az összes változó feletti a priori együttes eloszlást reprezentálja, faktorizálható az őt alkotó FVT-kre, de az a posteriori együttes eloszlás a megfigyelési szekvencián vett feltétellel – azaz az előre üzenet – általában már *nem* faktorizálható. Eddig senki sem talált megoldást erre a problémára, annak ellenére, hogy megoldást a tudomány és mérnöki tervezés fontos területei tudnák nagyon jól hasznosítani. Így közelítő módszerekre kell szorítkoznunk.

5.3. Közelítő következtetés DBH-kban

A 14. fejezet két következtető módszert írt le: a valószínűségi súlyozást (lásd 14.14. ábra) és a Markov lánc Monte Carlo módszert (MCMC, lásd 14.15. ábra). A kettő közül az első igazítható legkönnyebben a DBH-keretbe. Látni fogjuk azonban, hogy egy gyakorlati módszer kialakulásához a standard valószínűségi súlyozó algoritmuson számos javításra van szükség.

Created by XMLmind XSL-FO Converter.

Fontos

Emlékezzünk vissza, hogy a valószínűségi súlyozás a működése során a nem bizonyítékváltozókat mintavételezi azok topológiai sorrendjében, minden mintát azzal a valószínűséggel súlyozva, amennyire egyezik a megfigyelt bizonyítékváltozókkal. Ahogyan az egzakt közelítő algoritmusok esetében is, a valószínűségi súlyozást közvetlenül alkalmazhatnánk egy kibontott DBH-ra, de ugyanazok a problémák jelentkeznének, nevezetesen, hogy a megfigyeléssorozat növekedésével a frissítésenkénti idő- és tárigény növekedne. A probléma az, hogy az alapalgoritmus az egyes mintákat egymás után sorban végigfuttatja a háló teljes hosszában. Ehelyett egyszerűen az összes N mintát együtt futtathatnánk a DBH-n keresztül, szeletenként egyszerre haladva. (Azaz a mintavételeket az újragenerálás költségének elkerülése miatt nyilvántartanánk és fokozatosan frissítenénk – *a ford.*) A módosított algoritmus illeszkedik a szűrési algoritmus általános mintájához, az N mintával mint előrefelé üzenettel. Az első kulcsgondolat az, hogy ekkor *a mintákat magukat használjuk az állapoteloszlás egy közelítő reprezentációjaként*. Ez eleget tesz a frissítésenkénti „állandó” időigénynek, bár az állandó értéke a minták számától függ, aminek elég nagynak kell lenni a valódi a posteriori eloszlás elfogadható közelítéséhez. Szintén nincs szükség a DBH kibontására, mivel csak az aktuális és a következő szeletet kell memóriában tartani.

A valószínűségi súlyozás tárgyalásakor a 14. fejezetben rámutattunk, hogy az algoritmus pontossága leromlik, ha a bizonyítékváltozók „lentebbiek” a mintavételezett változóknál, mivel ekkor a minták generálására nincsen hatással a bizonyíték. Egy DBH tipikus struktúráját megnézve – mondjuk az esernyős DBH-t a 15.14. ábrán – látjuk, hogy a korai állapotváltozók mintavételezésénél valóban nem hasznosulnak a későbbi bizonyítékok. Valójában, még tüzetesebben megnézve azt láthatjuk, hogy az *állapotváltozók egyikének sincs egyetlen bizonyítékváltozó sem* az ősei között! Így, bár az egyes minták súlya függeni fog a bizonyítékoktól, a generált minták aktuális halmaza teljesen független lesz a bizonyítékoktól. Például még ha a főnök mindennap el is hozza az esernyőjét, a mintavételi folyamat ekkor is hallucinálhat véget nem érő kánikulát. A gyakorlatban ez azt jelenti, hogy a minták azon aránya, ami elfogadhatóan közel marad az események valódi sorához, exponenciálisan csökken t-vel, a megfigyelés sorozat hosszával; máshogy fogalmazva egy adott pontossági szint fenntartásához, a minták számát exponenciálisan növelni kell t függvényében. Mivel egy valós időben működő szűrő algoritmus csak rögzített számú mintát tud használni, a gyakorlatban az történik, hogy igen kis számú frissítési lépés után a hiba határtalanul megnő.

Fontos

Világos, hogy jobb megoldásra van szükségünk. A második kulcsgondolat az, hogy *a minták halmazát az állapottér nagy valószínűségű részeire koncentráljuk*. Ez megtehető a megfigyelések szerint nagyon alacsony súlyú minták eldobásával, miközben megsokszorozzuk a nagy súlyúakat. Ezen a módon a minták populációja elfogadhatóan közel marad a valósághoz. Ha úgy gondolunk a mintákra, mint egy eszközre az a posteriori eloszlás modellezéséhez, akkor van értelme több mintát használni az állapottér azon részeiben, ahol az a posteriori valószínűség nagyobb.

Az algoritmusok egy családját, a **részecskeszűrés**t (**particle filtering**) pontosan ennek elvégzésére tervezték. A részecskeszűrés a következőképpen működik: először egy N mintájú populációt alkotunk a 0. időpontbeli $\mathbf{P}(\mathbf{X}_0)$ a priori eloszlás mintavételezéséből. Ezután minden időpontban egy frissítési ciklus ismétlődik:

- Minden mintát előreterjesztünk mintavételezve az $\mathbf{x}_{t+1}$ következő állapot értéket feltéve, hogy a minta aktuális értéke $\mathbf{x}_t$ adott, és felhasználva a $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)$ állapotátmenet-modellt.
- Minden mintát súlyozunk azzal a $P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1})$ valószínűséggel, amit az új bizonyítékhoz rendel.
- A populációt újra mintavételezzük, hogy egy N mintájú új populációt generáljunk. Minden új mintát a jelenlegi populációból választunk ki; egy konkrét minta kiválasztásának a valószínűsége arányos a súlyával. Az új minták súlyozatlanok.

Az algoritmus részletesen a 15.15. ábrán látható, és működését az esernyős DBH esetén a 15.16. ábra szemlélteti.

15.15. ábra - A részecskeszűrés algoritmus megvalósítása egy rekurzív frissítési művelettel és állapottal (ami a minták halmaza). A mintavételező lépések mindegyike az aktuális szelet változóinak a mintavételezését jelenti egy topológiai sorrendben,

Created by XMLmind XSL-FO Converter.

hasonlóan, mint a PRIOR-MINTA-ban. A SÚLYOZOTT-MINTA-VISSZATÉTELLEL művelet megvalósítható *O*(*N*) várható futási idővel.

function RÉSZECSKE-SZŰRÉS (**e**, *N*, *dbh*) **returns** egy mintahalmaz a következő időpontra
  inputs: **e**, az új beérkező bizonyíték
    N: a fenntartani kívánt mintaszám
    dbh: egy DBH egy $\mathbf{P}(\mathbf{X}_0)$ a priori eloszlással, $\mathbf{P}(\mathbf{X}_1|\mathbf{X}_0)$ állapotátmenet-modellel és $\mathbf{P}(\mathbf{E}_1|\mathbf{X}_1)$ érzékelő modellel
  static: *S*, a minták egy *N* méretű vektora, kezdetben $\mathbf{P}(\mathbf{X}_0)$-ból generált
  local variables: *W*, a súlyok egy *N* méretű vektora

  for $i = 1$ **to** N **do**
    $S[i] \leftarrow$ sorsolt minta $\mathbf{P}(\mathbf{X}_1|\mathbf{X}_0 = S[i])$-ből
    $W[i] \leftarrow \mathbf{P}(\mathbf{e}|\mathbf{X}_1 = S[i])$
  $S \leftarrow$ SÚLYOZOTT-MINTA-VISSZATÉTELLEL(N, S, W)
  return *S*

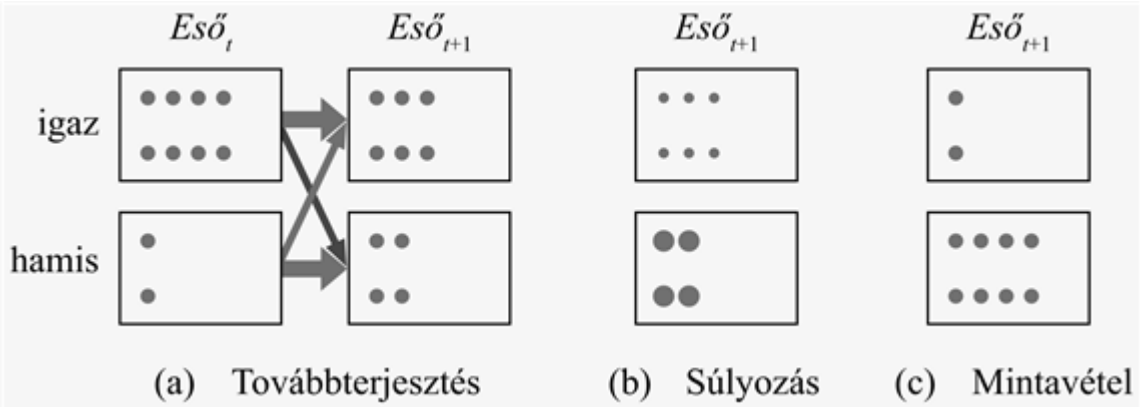
Megmutatjuk, hogy ez az algoritmus konzisztens – így helyes valószínűségeket ad, ahogy *N* a végtelenbe tart – meggondolva, hogy mi történik egy frissítési ciklus alatt. Feltesszük, hogy a mintapopuláció a *t* időpontbeli $\mathbf{f}_{1:t}$ előre üzenet egy helyes reprezentációjaként indul. Az $N(\mathbf{x}_t|\mathbf{e}_{1:t})$ jelölést használva azon minták számára, amelyek az $\mathbf{e}_{1:t}$ megfigyelések feldolgozása után az $\mathbf{x}_t$ állapotban vannak, azt kapjuk, hogy

$$N(x_t|e_{1:t}) \;/N = P(x_t|e_{1:t}) \quad (15.21)$$

nagy *N*-ekre. Most minden mintát előreterjesztünk mintavételezve az állapotváltozókat a $t + 1$ időponban az adott *t* időponthoz tartozó mintabeli értékek mellett. Az egyes $\mathbf{x}_t$ állapotokból az $\mathbf{x}_{t+1}$ állapotot elérő minták száma az átmenet-valószínűség és az $\mathbf{x}_t$ populáció számának a szorzata; így az $\mathbf{x}_{t+1}$ állapotot elérő minták teljes száma

$$N(\mathbf{x}_{t+1}|\mathbf{e}_{1:t}) = \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1}|\mathbf{x}_t) N(\mathbf{x}_t|\mathbf{e}_{1:t})$$

15.16. ábra - A részecskeszűrés frissítési ciklusa az esernyő DBH-nál *N* = 10 esetén, bemutatva minden egyes állapot mintapopulációját. (a) A *t* időpontban 8 minta jelez *Eső*-t és 2 ¬*Eső*-t. Mindegyiket továbbterjesztjük a következő állapotátmenet-modell szerinti mintavételezésével. A *t* + 1 időpontban 6 minta jelez *Eső*-t és 4 ¬*Eső*-t. (b) A *t* + 1 időpontban ¬*Esernyő* a megfigyelés értéke. Mindegyik mintát súlyozzuk a mintának a megfigyelésre vett feltételes valószínűségével, amit a körök nagysága jelez. (c) Egy új 10-es mintahalmaz generálódott az aktuális halmazból történő súlyozott sorsolással, 2 olyan mintát eredményezve, ami *Eső*-t jelez, és 8 olyat, ami ¬*Eső*-t.



Created by XMLmind XSL-FO Converter.

Most minden egyes mintát súlyozunk a $t + 1$ időpontbeli bizonyíték valószínűségével. Egy $\mathbf{x}_{t+1}$ állapotban lévő minta $P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1})$ súlyt kap. Az $\mathbf{x}_{t+1}$ állapotban lévő minták teljes súlya az $\mathbf{e}_{t+1}$ megfigyelése után ezért

$$W(x_{t+1}|e_{1:t+1}) = P(e_{t+1}|x_{t+1})N(x_{t+1}|e_{1:t})$$

Most a mintavételi lépés jön. Mivel minden egyes minta a súlyával arányos valószínűséggel ismétlődik, az újramintavételezés után az $\mathbf{x}_{t+1}$ állapotban lévő minták száma arányos az $\mathbf{x}_{t+1}$ állapotban lévő minták teljes súlyával az újramintavételezés előtt:

$$\begin{aligned}
&N(x_{t+1}|e_{1:t+1})/N = \alpha W(x_{t+1}|e_{1:t+1}) \\
&= \alpha P(e_{t+1}|x_{t+1})N(x_{t+1}|e_{1:t}) \\
&= \alpha P(e_{t+1}|x_{t+1})\sum_{x_t} P(x_{t+1}|x_t)N(x_t|e_{1:t}) \\
&= \alpha NP(e_{t+1}|x_{t+1})\sum_{x_t} P(x_{t+1}|x_t)P(x_t|e_{1:t}) \qquad \text{(15.21 miatt)} \\
&= \alpha' P(e_{t+1}|x_{t+1})\sum_{x_t} P(x_{t+1}|x_t)P(x_t|e_{1:t}) \\
&= P(x_{t+1}|e_{1:t+1}) \qquad \text{(15.3 miatt)}
\end{aligned}$$

Azaz a mintapopuláció egy frissítési ciklus után helyesen reprezentálja a $t + 1$ időpontbeli előrefelé üzenetet.

A részecskeszűrés így *konzisztens,* de *hatékony-e?* A gyakorlat azt mutatja, hogy a válasz igen: a részecskeszűrés, úgy tűnik, az igazi a posteriori eloszlás egy jó közelítését tartja fenn állandó számú mintát használva. Azonban jelenleg nincs elméleti garancia; a részecskeszűrés jelenleg aktívan kutatott terület. Számos javítást és változatot javasoltak, és az alkalmazások köre is egyre bővül. Mivel ez egy mintavételező algoritmus, a részecskeszűrés könnyen felhasználható hibrid és folytonos DBH-kban, lehetővé téve az alkalmazását akár olyan területeken is, mint komplex mintázatok követése videofelvételben (Isard és Blake, 1996) és tőzsdei előrejelzés (de Freitas és társai, 2000).

6. Beszédfelismerés

Ebben az alfejezetben az időbeli valószínűségi modellek egyik legfontosabb alkalmazását, a **beszédfelismerés**t (**speech recognition**) tekintjük át. A feladat egy beszélő által elmondott szósorozat azonosítása az adott akusztikus jelből. A beszéd az emberek közötti kommunikációnak a domináns modalitása, és a megbízható számítógépes beszédfelismerés felmérhetetlenül fontos lenne. Még ennél is hasznosabb volna a **beszédmegértés** (**speech understanding**) – az elhangzott beszéd *jelentésének* a meghatározása. Ennek tárgyalására a 22. fejezetig kell várnunk.

A beszéd jelenti az első találkozásunkat a valódi érzékelők szolgáltatta adatok nyers, tisztítatlan világával. Ezek az adatok zajosak, a szó szoros értelmében: a *zaj* lehet háttérzaj és lehet a digitalizálási folyamat okozta melléktermék; eltérések lehetnek a szavak kiejtési módjában még ugyanannál a beszélőnél is; különböző szavak hangzása ugyanaz lehet és így tovább. Ezen okok miatt a beszédfelismerésre mint valószínűségi következtetési problémára kezdtek el tekinteni.

A legáltalánosabb szinten a valószínűségi következtetési problémát a következőképpen definiálhatjuk. Legyen a *Szavak* egy valószínűségi változó az összes lehetséges szószekvencia felett, ami elhangozhat, és legyen a *jel* a megfigyelt akusztikusjel-szekvencia. Ekkor az elhangzottak legvalószínűbb értelmezése a *Szavak* azon értéke, ami maximalizálja a $P(szavak|jel)$ értéket. Amint az gyakori eset, a Bayes-szabály alkalmazása hasznos:

$$P(szavak|jel) = \alpha P(jel|szavak)\ P(szavak)$$

A $P(jel|szavak)$ alkotják az ún. **akusztikai modell**t (**acoustic model**). Ez írja le a szavak hangzását – a „ceiling (mennyezet)” egy lágy „c”-vel kezdődik és ugyanúgy hangzik, mint a „sealing (pecsételés/tömítés/fókavadászat)”. (Az **azonos hangzású** szavakat **homofon** (**homophone**) szavaknak nevezzük.) A $P(szavak)$ értékei képezik az ún. **nyelvi modell**t (**language model**). Ez adja meg az a priori valószínűségét minden egyes kijelentésnek – például hogy a „magas mennyezet (high ceiling)” sokkal valószínűbb szókapcsolat, mint a „magas pecsételés (high sealing)”.

Created by XMLmind XSL-FO Converter.

A beszédfelismerő rendszerekben használt nyelvi modellek gyakran igen egyszerűek. A fejezetben később leírt **bigram** (**bigram**) **modell** minden lehetséges esetre megadja annak a valószínűségét, hogy egy szó egy másik szót követ. Az akusztikai modell sokkal bonyolultabb. Az alapjait egy fontos felfedezés alkotja a **fonológia** (**phonology**) (a nyelvi hangok tanulmányozása) területéről, nevezetesen, hogy az emberiség minden nyelve egy 40–50 hangból álló korlátos választékkal él, amelyeket **beszédhang**oknak (**phone**s) hívunk. Egy beszédhang nagyjából az a hang, amely egy egyedi mással- vagy magánhangzónak felel meg. A helyzetet komplikálja valamelyest, hogy az olyan betűkombinációk, mint „th” vagy „ng” egyedi beszédhangot eredményeznek, viszont egyes betűk többféle beszédhanghoz vezetnek, a környezetüktől függően (például az „a” betű a „rat” és a „rate” szóban). Az angol nyelv összes beszédhangja, példákkal együtt, a 15.17. ábrán látható. A **fonéma** (**phoneme**) a legkisebb hangi egység, ami önálló jelentéssel bír egy adott nyelv használói számára. Például az angolban a „t” beszédhang a „stick” szóban ugyanaz a fonéma, mint a „t” beszédhang a „tick” szóban, de a thai nyelvben ezek megkülönböztethetők, mint két különböző fonéma.

A beszédhangok létezése lehetővé teszi az akusztikai modell két részre bontását. Az első rész a **kiejtés**t (**pronunciation**) kezeli, és minden egyes szóra megad egy valószínűség-eloszlást az összes lehetséges beszédhangsor felett. Például a „ceiling” szó kiejtése [s iy l ih ng], vagy néha [s iy l ix ng], vagy esetleg [s iy l en]. A beszédhangok közvetlenül nem megfigyelhetők, így nagyjából a beszéd egy olyan rejtett Markov-modellel reprezentálható, aminek X_t állapotváltozója a t időpontban kimondott beszédhang.

Az akusztikai modell második része a beszédhangok akusztikus jelként való megvalósulási módjával foglalkozik: azaz a rejtett Markov-modell E_t bizonyítékváltozója adja meg az akusztikus jel t időpontban megfigyelt jellemzőit, és az akusztikus modell adja meg az $P(E_t|X_t)$ feltételes valószínűséget, ahol X_t az aktuális beszédhang. A modellnek lehetőséget kell adni eltérésekre a hangmagasságban, a sebességben és a hangerőben, és **jelfeldolgozás**i (**signal processing**) technikákon alapul, hogy olyan jelreprezentációt biztosítson, ami megfelelően robusztus módon kezeli ezeket a fajta eltéréseket.

15.17. ábra - A DARPA fonetikus ábécé, avagy ARPAbet, amely az amerikai angolban használt beszédhangokat sorolja fel. Számos alternatív felírás létezik, köztük az International Phonetic Alphabet (IPA), mely az összes ismert nyelv beszédhangját tartalmazza.

Magánhangzók		**B–N mássalhangzók**		**P–Z mássalhangzók**	
Beszédhang	Példa	Beszédhang	Példa	Beszédhang	Példa
[iy]	beat	[b]	**b**et	[p]	**p**et
[ih]	b**i**t	[ch]	**Ch**et	[r]	**r**at
[ey]	b**e**t	[d]	**d**ebt	[s]	**s**et
[ae]	b**a**t	[f]	**f**at	[sh]	**sh**oe
[ah]	b**u**t	[g]	**g**et	[t]	**t**en
[ao]	**bought**	[hh]	**h**at	[th]	**th**ick
[ow]	b**oa**t	[hv]	**h**igh	[dh]	**th**at
[uh]	b**oo**k	[jh]	**j**et	[dx]	bu**tt**er
[ey]	b**ai**t	[k]	**k**ick	[v]	**v**et
[er]	B**er**t	[l]	**l**et	[w]	**w**et
[az]	b**uy**	[el]	bott**le**	[wh]	**wh**ich
[oy]	b**oy**	[m]	**m**et	[y]	**y**et
[axr]	din**er**	[em]	bott**om**	[z]	**z**oo
[aw]	d**ow**n	[n]	**n**et	[zh]	mea**s**ure
[ax]	**a**bout	[en]	butt**on**		
[ix]	ros**e**s	[ng]	si**ng**		
[aa]	c**o**t	[eng]	washi**ng**	[-]	*(Szünet)*

A fejezet hátralevő része a modelleket és az algoritmusokat írja le lentről felfelé haladva, az akusztikus jelekkel és a beszédhangokkal kezdve, majd az egyedi szavakkal és végül teljes sorozatokkal folytatva. Befejezésül áttekintjük, hogy ezek a modellek hogyan taníthatók, és mennyire működik jól a kapott rendszer.

6.1. Beszédhangok

Created by XMLmind XSL-FO Converter.

A hanghullámok olyan periodikus nyomásváltozások, amelyek a levegőben terjednek. A hang egy mikrofonnal mérhető, aminek membránját elmozdítja a nyomásváltozás, és egy folytonosan változó áramot generál. Az áramerősséget – ami a hanghullám amplitúdójához tartozik – egy analóg-digitális konverter méri diszkrét időintervallumonként a **mintavételi frekvencia** (**sampling rate**) szerint. Beszéd esetén a tipikus mintavételi frekvencia 8 és 16 kHz (azaz 8000-től 16 000-ig másodpercenként) között van. (Jó minőségű zenei felvételeket mintavételeznek 44 kHz vagy még magasabb frekvenciával.) A mérés pontosságát az egyes mintavételi pontokban a **kvantálási tényező** (**quantization factor**) határozza meg; a beszédfelismerők tipikusan 8–12 bitet használnak. Ez azt jelenti, hogy a legkevésbé igényes rendszer, amely 8 kHz-es frekvenciával mintavételez, és 8 bites kvantálást használ, közel fél megabájtot igényel egy egyperces beszéd tárolásához. Ilyen nagy mennyiségű jelinformáció esetén kivitelezhetetlen létrehozni és használni a $P(jel|beszédhang)$ eloszlásokat, így az akusztikus jelnek tömörebb leírásait kell kifejlesztenünk.

Elsőként azt vesszük figyelembe, hogy bár a beszédben a hangfrekvencia akár több kHz-es is lehet, a jel tartalmának a *megváltozásai* sokkal ritkábban fordulnak elő, talán nem több mint 100 Hz-cel. Ezért a beszédrendszerek a jel tulajdonságait hosszabb időintervallumonként, úgynevezett **keret**enként (**frame**s) összegzik. Egy 10 ms-os (80 minta 8 kHz-es mintavétel mellett) keret elég rövidnek tűnik ahhoz, hogy csak néhány rövid idejű jelenség mosódjon el az összegzési folyamat miatt. Az egyes kereteken belüli folyamatot egy **jellemző** (**feature**) vektorral ábrázoljuk. Például jellemezni akarjuk az energia szintjét számos frekvenciasáv mindegyikében. Más fontos jellemzők a keret globális energiaszintje és a megelőző kerettől számított (energia)különbség. A jellemzők kiemelése a beszédjelből hasonlít egy kicsit arra, ahogy a zenekar játékát figyelve megjegyezzük: „Most a vadászkürtök hangosan és a hegedűk halkan szólnak.” A 15.18. ábrán láthatjuk az átalakítás lépéseit a nyers hangtól a keretek sorozatáig. Figyeljük meg, hogy a keretek átlapolódnak; ez megvéd minket attól, hogy pont a keret határán fellépő fontos akusztikus események információját elveszítsük.

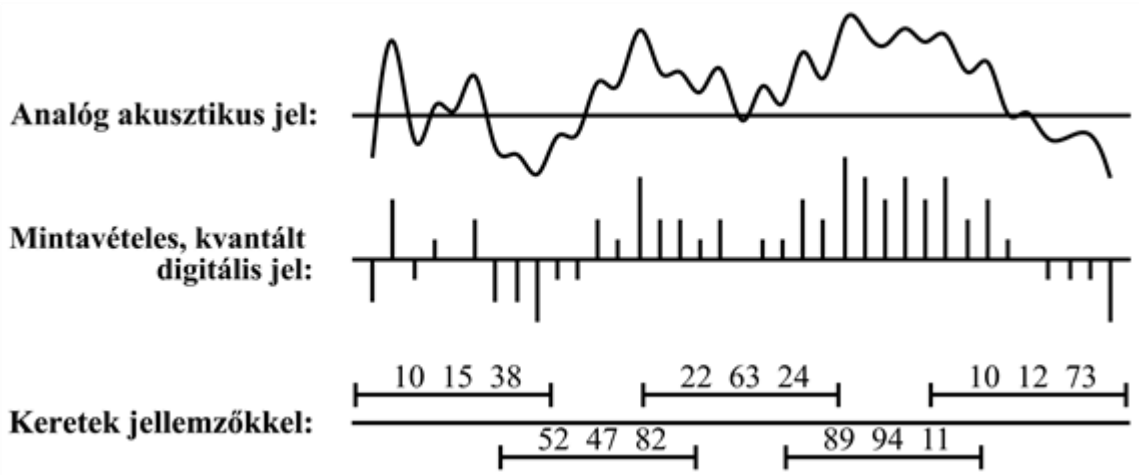
A példánkban csupán három jeggyel rendelkező keretet mutattunk. Valós rendszerek jellemzők tucatjaival vagy akár százaival dolgoznak. Például n darab jellemző esetén, amikor mindegyik 256 lehetséges értéket vehet fel, egy keret egy ponttal adható meg egy n-dimenziós térben, és 256^n lehetséges keret létezik. $n > 2$ esetén kivitelezhetetlen volna a $P(jellemzők|beszédhang)$ eloszlásnak egy explicit táblázatként történő ábrázolása, így további tömörítésre van szükségünk. Két lehetséges megközelítés létezik:

- A **vektorkvantálás** (**VK**) (**vector quantization**, **VQ**) módszere felosztja az n dimenziós teret mondjuk 256 partícióra, C1-től C256-ig címkézve. Ekkor az egyes kereteket inkább egy egyedi címkével, mint egy n számból álló vektorral lehet jellemezni. Így a $P(VQ|beszédhang)$ eloszlás táblázatos formája 256 valószínűséget tartalmaz minden egyes beszédhangra. Nagy rendszerekben a vektorkvantálás már nem népszerű.

- A jellemzők terének diszkretizálása helyett parametrikus folytonos eloszlásokat használhatunk a $P(jellemzők|beszédhang)$ leírásához. Például használhatunk minden beszédhanghoz egy Gauss-eloszlást, beszédhangonként különböző átlaggal és kovarianciamátrixszal. Ez jól működik, ha az egyes beszédhangok akusztikus megvalósulása a jellemzők terében egyetlen területen csoportosul. A gyakorlatban egy hanghoz több területen való csoportosulás is tartozik, és **Gauss-eloszlások keverék**ét (**mixture of Gaussians**) kell használni. Egy keverék k egyedi eloszlás súlyozott összege, így a $P(jellemzők|beszédhang)$ eloszláshoz k súly, k n elemű átlagvektor és k n^2 méretű kovarianciamátrix tartozik, ami $O(kn^2)$ paraméter minden beszédhangra.

15.18. ábra - Az akusztikus jel átalakítása keretek sorozatává; mindegyik keretet három akusztikus jellemzővel írunk le

Created by XMLmind XSL-FO Converter.



Természetesen bizonyos információ elvész a feldolgozás során, ami a teljes beszédjeltől egy VK címkéig vagy a keverékparaméterek egy halmazáig tart. A jelfeldolgozás művészete abban rejlik, hogy úgy válasszuk meg a jellemzőket és a tartományokat (vagy a Gauss-eloszlásokat), hogy a hasznos információ vesztesége minimális legyen. Egy adott beszédhang számos módon kiejthető: hangosan vagy halkan, gyorsan vagy lassan, magasan vagy mély hangon, csendben vagy háttérzajban és beszélők sok milliója által, mindegyikőjük különböző kiejtésével és eltérő hangképző szerveivel.[6]

Még két további finomítást kell elvégeznünk az eddig leírt egyszerű modellen. Az első a beszédhangok időbeli szerkezetével foglalkozik. Normális beszédben a legtöbb beszédhangnak a terjedelme 50–100 ezredmásodperc vagy 5–10 keret. A $P(\textit{jellemzők}|\textit{beszédhang})$ valószínűségi modell ugyanaz ezen keretek mindegyikére, pedig a legtöbb beszédhangnak jelentős belső struktúrája van. Például a [t] egyike a **zárhang**oknak (**stop consonant**s), amelyekben a levegő áramlása megszűnik egy rövid időre egy határozott kibocsátás előtt. Az akusztikus jelet megvizsgálva azt látjuk, hogy egy [t]-nek a kezdete hangtalan, közepén egy kis kirobbanással és (általában) egy sziszegéssel a végén. A beszédhangoknak ez a belső struktúrája egy háromállapotú modellel ragadható meg; minden beszédhangnak van egy kezdeti, közép- és végállapota, és minden állapotnak megvan a saját eloszlása a jegyek felett.

A második finomítás azzal a környezettel foglalkozik, amelyben a beszédhang elhangzik. Egy adott beszédhang hangzását megváltoztathatják a környező beszédhangok.[7] Emlékezzünk arra, hogy a beszédhangok az ajkak, a nyelv és az álkapocs mozgása és a levegőnek a hangképző szerveken történő kényszerített átáramlása révén keletkezik. Ezen komplex mozgások koordinálásánál, amik másodpercenként öt vagy még több beszédhangot produkálnak, az agy már akkor elindít a második beszédhanghoz tartozó műveletet, mielőtt az első befejeződött volna, így megváltoztatva az egyik vagy mindkét beszédhangot. Például a „sweet (édes)” kiejtésénél az ajkak kerekítettek az [s] előállítása közben számítva a következő [w]-re. Ezeket a **koartikulációs hatás**okat (**coarticulation effect**s) részlegesen megragadja a **hármashangzó** (**triphone**) modell, amelyben az egyes beszédhangok akusztikai modellje függhet az előző és következő beszédhangtól. Így a [w] a „sweet”-ben úgy írható, hogy [w(s,iy)], azaz [w] egy [s] bal- és egy [iy] jobb-környezetben.

A háromállapotú és hármashangzó modellek együttes hatása megnöveli az időbeli folyamat lehetséges állapotainak a számát az eredeti beszédhang készletbeli n beszédhangról $3n^3$-ra (az ARPAbet-ben $n \approx 50$). A tapasztalatok azt mutatják, hogy a megnövelt pontosság bőven ellensúlyozza a következtetés és a tanulás megnövekedett költségeit.

6.2. Szavak

Minden egyes szóra gondolhatunk úgy, mint ami meghatároz egy külön $\mathbf{P}(X_{1:t}|\textit{szó})$ valószínűségi eloszlást, ahol X_i megadja az i-edik keretben a beszédhang állapotát. Tipikusan ezt az eloszlást két részre osztjuk. A **kiejtési modell** (**pronunication model**) egy eloszlást ad meg a beszédhangsorok felett (figyelmen kívül hagyva az ütemet és a kereteket), majd a **beszédhangmodell** (**phone model**) leírja, ahogyan a beszédhangok leképződnek a keretek szekvenciájára.

[6] A duális probléma – a **beszélőazonosítás** (**speaker identification**) pontosan ennek fordítottja, a közös elemeket hagyja el, és az egyéni különbségeket tartja meg, majd megpróbálja a különbségeket az egyéni beszédmodellekhez illeszteni.
[7] Ebben az értelemben a beszéd „beszédhangmodelljét” inkább hasznos közelítésnek kell tekinteni, és nem megváltoztathatatlan törvénynek.

Created by XMLmind XSL-FO Converter.

Gondoljunk arra a szóra, hogy „tomato (paradicsom)”. Gershwin szerint ezt úgy ejtik ki, hogy [t ow m ey t ow] (Gershwin, 1937), én viszont úgy ejtem, hogy [t ow m aa t ow]. A 15.19. ábra felső részén látható egy olyan állapotátmenet-modell, ami gondoskodik erről a változatról. A modellen keresztül csak két út létezik, egy, ami a [t ow m ey t ow] beszédhang szekvenciához tartozik, és a másik a [t ow m aa t ow] szekvenciához. Egy út valószínűsége az útvonalat alkotó nyilakon szereplő valószínűségek szorzata:

$$P([towmeytow]|\text{„tomato”}) = P([towmaatow]|\text{„tomato”}) = 0,5$$

A fonetikai változékonyság második forrása a **koartikuláció** (**coarticulation**). Például a [t] beszédhangnál a nyelv a szájpadlásnál helyezkedik el, az [ow]-nál viszont a száj alján. Gyors beszédnél a nyelv sokszor közbülső pozícióba kerül, melynek eredménye inkább a [t ah], és nem a [t ow]. A 15.19. ábra alsó része a „tomato” kiejtésének egy bonyolultabb modelljét mutatja, amely ezt a koartikulációs hatást figyelembe veszi. Ebben a modellben négy különböző út van, és azt kapjuk, hogy:

$$P([towmeytow]|\text{„tomato”}) = P([towmaatow]|\text{„tomato”}) = 0,1$$

$$P([tahmeytow]|\text{„tomato”}) = P([tahmaatow]|\text{„tomato”}) = 0,4$$

Hasonló modelleket lehetne konstruálni minden felismerni kívánt szóhoz.

A háromállapotú beszédhangmodell állapotátmenet-diagramja a 15.20. ábrán látható. A modell egy konkrét beszédhanghoz tartozik, az [m]-hez, de az összes beszédhangnak hasonló topológiájú modellje van. Az ábrán láthatók az egyes beszédhang állapotokhoz kapcsolódó akusztikai modellek is, feltételezve, hogy a jelet egy VK címke reprezentálja. Például a modell szerint $P(E_t = C_1|X_t = [\text{m}]_{\text{Kezdet}}) = 0,5$. Vegyük észre a hurkokat az ábrán; például az $[\text{m}]_{\text{Közép}}$ állapot 0,9 valószínűséggel fennmarad, ami azt jelenti, hogy az $[\text{m}]_{\text{Közép}}$ állapot várható időtartama 10 keret. A modellünkben az egyes beszédhangok hossza független a többi beszédhang hosszától; egy kifinomultabb modell képes lenne a gyors és lassú beszédet megkülönböztetni.

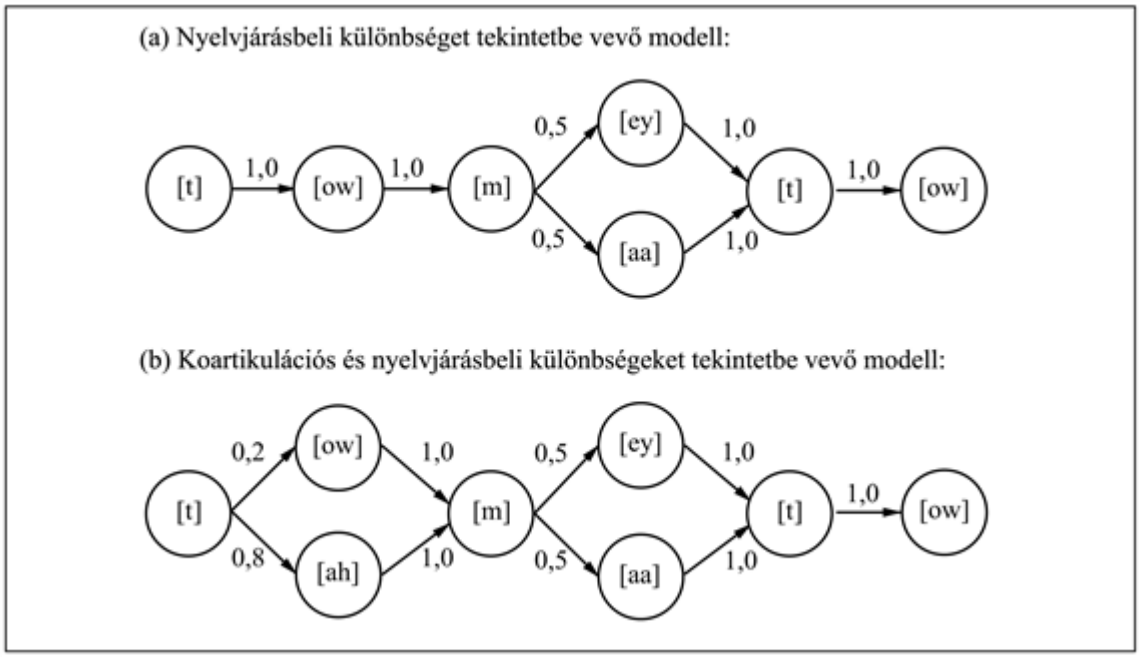
Hasonló modelleket hozhatunk létre minden egyes beszédhangra, akár a hármashangzó környezet hatását is figyelembe véve. Minden szómodell, amikor a beszédhangmodellekkel kombináljuk, egy RMM teljes specifikációját adja. A modell megadja a beszédhang állapotok közötti, keretről keretre történő átmeneteknek a valószínűségeit csakúgy, mint az akusztikus jegyek valószínűségeit minden egyes beszédhang állapothoz.

Ha **egyedülálló szavak**at (**isolated words**) szeretnénk felismerni – azaz egyértelmű határokkal rendelkező és mindennemű környezeti összefüggés nélkül kiejtett szavakat –, akkor azt a szót kell megkeresnünk, amelyik maximalizálja azt, hogy

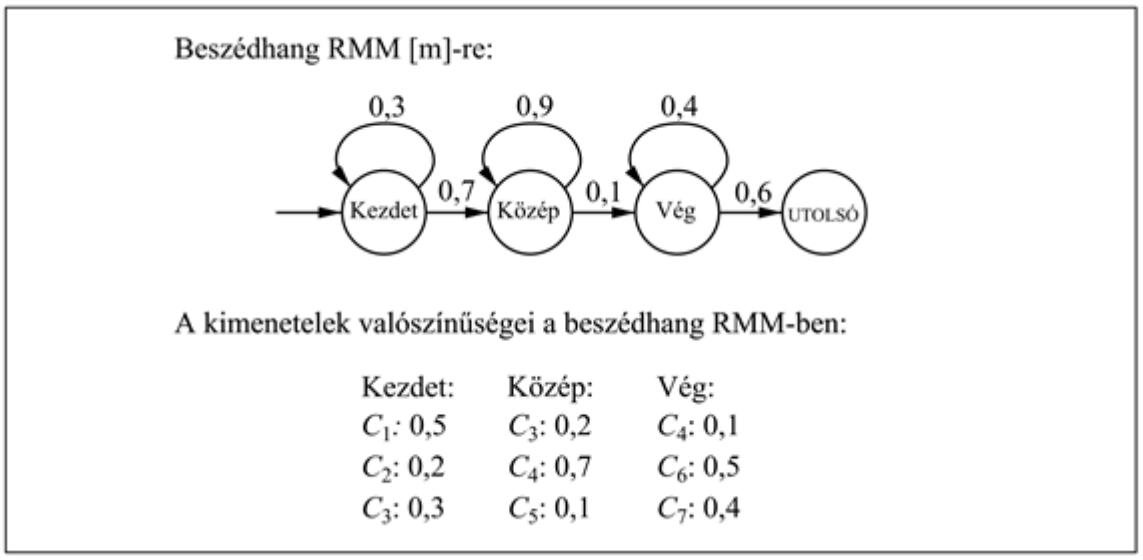
$$P(szó|e_{1:t}) = \alpha P(e_{1:t}|szó)\,P(szó)$$

15.19. ábra - A „tomato” szó két kiejtési modellje. Mindegyik modellt egy átmenetdiagramként ábrázoljuk, amiben az állapotokat körrel jelöljük, a megengedett átmeneteket pedig nyilakkal, rajtuk a kapcsolódó valószínűségekkel. (a) Egy nyelvjárásbeli különbségeket is tekintetbe vevő modell. A 0,5-ös értékek a két szerző preferált kiejtésein alapuló becslések. (b) Egy olyan modell, ami az első magánhangzón egy koartikulációs hatást is figyelembe vesz, megengedve az [ow] vagy az [ah] beszédhangokat.

Created by XMLmind XSL-FO Converter.



15. ábra - Az [m] háromállapotú beszédhang egy RMM-je. Mindegyik állapotnak számos lehetséges kimenetele lehet, különálló valószínűségekkel. A C_1, …, C_7 VK-címkék önkényesen lettek megválasztva.



$$P(szó|e_{1:t}) = \alpha P(e_{1:t}|szó)\,P(szó)$$

A $P(szó)$ a priori valószínűség valódi szöveges adatból kapható meg. A $P(e_{1:t}|szó)$ pedig az akusztikus jegyek sorozatának a valószínűsége a szómodell szerint. Ilyen valószínűségek kiszámítását a 15.2. alfejezet tárgyalta; nevezetesen a (15.15) egyenlet egy egyszerű rekurzív számítást definiál, lineáris költséggel t-ben és a Markov-lánc állapotainak számában. A legvalószínűbb szó megtalálásánál ezt a számítást minden lehetséges szóra elvégezhetjük, megszorozzuk az a priori valószínűséggel, és e szerint választjuk ki a legjobb szót.

6.3. Mondatok

Az emberi kommunikációban való részvételhez egy gépnek a **folytonos beszéd**et (**continuous speech**) kell felismernie, és nem pusztán egyedülálló szavakat. Azt gondolhatnánk, hogy a folytonos beszéd nem több, mint szavak sorozata, amelyek mindegyikére alkalmazhatjuk az algoritmust az előző fejezetből. Ez a megközelítés két ok miatt is bukásra van ítélve. Először is, már láttuk (a 2.3. szakasz - A legvalószínűbb sorozat megtalálása részben), hogy a legvalószínűbb szavak sorozata nem a legvalószínűbb együttes szósorozat. Például a *Take the*

Created by XMLmind XSL-FO Converter.

Money and Run (Fogd a pénzt és fuss) c. filmben a banki alkalmazott félreérti Woody Allen irkafirka bankrablási üzenetét, és azt „I have a gub (Nálam van a slukker)”-nak olvassa (az „I have a gun, vagyis „Nálam van a stukker” helyett – *a ford.*). Egy jobb nyelvi modell az „I have a gun”-t javasolná, mint aminek sokkal nagyobb a valószínűsége, még ha a legutolsó szó inkább „gub”-nak és nem „gun”-nak olvasható. A második probléma, amivel folytonos beszéd esetén szembesülnünk kell, a **szegmentálás** (**segmentation**) – annak az eldöntése, hogy hol van egy szónak a vége, és hol kezdődik a következő. Mindenki, aki megkísérelt egy idegen nyelvet megtanulni, méltányolni fogja a probléma nehézségét: először úgy tűnik, hogy a beszédben a szavak egybefolynak, majd fokozatosan tanulunk meg egyedi szavakat kiemelni a hangok egyvelegéből. Ebben az esetben az első benyomás korrekt: a spektrográfiai elemzés azt mutatja, hogy a folyamatos beszédben a szavak tényleg szünetek nélkül *követik* egymást. A szavak határait annak ellenére tanuljuk meg megkülönböztetni, hogy a szavakat nem választják szét csendszakaszok.

Kezdjük a nyelvi modellel, aminek a beszédfelismerésben az a feladata, hogy valószínűséget rendeljen minden egyes szósorozathoz. Legyen $w_1 \ldots w_n$ az n szóból álló szósorozat, és legyen w_i a szósorozat i-edik szava, akkor a szósorozat-valószínűséget az alábbi módon írhatjuk:[8]

$$P(w_1 \ldots w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1 w_2)\ldots P(w_n|w_1 \ldots w_{n-1}) =$$
$$= \prod_{i=1}^{n} P(w_i|w_1 \ldots w_{i-1})$$

Ezen kifejezések többsége igen bonyolult és nehezen becsülhető vagy számítható. Szerencsére a képletet egyszerűbb kifejezéssel is közelíthetjük, miközben a nyelvi modell zömét mégis megragadjuk. Az egyik egyszerű, közkedvelt és hatékony eljárás a **bigram** (**bigram**) **modell**. Ez a modell a $P(w_i|w_1 \ldots w_{i-1})$-et a $P(w_i|w_{i-1})$-gyel közelíti. Más szóval, feltételezi, hogy a szósorozatokra teljesül az elsőrendű Markov-feltétel.

A bigram modell nagy előnye, hogy a modell egyszerűen tanítható az egyes szópárok megszámolásával szósorozatok egy reprezentatív korpuszában és a valószínűségek ezen számok alapján történő megbecslésével. Így például, ha az „a” a mintakorpuszban 10 000-szer fordul elő, és a „gun” azt 37-szer követi, akkor $P(gun_i|a_{i-1})$ = 37/10 000, ahol a $\hat{P}$ a becsült valószínűség. Az ilyen tanítás befejeztével elvárnánk, hogy az „I have” és az „a gun” valószínűsége relatíve magas legyen, az „I has” és az „an gun” valószínűsége viszont alacsony. A 15.21. ábra a jelen könyv (eredeti angol nyelvű kiadásának) szavaiból számított néhány bigram gyakoriságát mutatja.

15.21. ábra - Az unigram és bigram számlálóknak egy részleges táblázata a jelen könyv (eredeti angol nyelvű kiadásának) szavai alapján. A „the” a leggyakoribb egyedülálló szó 33 508 előfordulással (513 893 összes szóból). A leggyakoribb bigram az „of the”, 3833 előfordulással. Néhány szám nagyobb, mint várható (például 4-szer fordul elő az „on is”), mivel a bigram számlálás figyelmen kívül hagyja az elválasztást: egy mondat végződhet „on”-ra és a következő kezdődhet „is”-zel.

Szó	Unigram gyakoriság	Előző szó							
		of	in	is	on	to	from	model	agent
the	33508	3833	2479	832	944	1365	597	28	24
on	2573	1	0	33	2	1	0	0	6
of	15474	0	0	29	1	0	0	88	7
to	11527	0	4	450	21	4	16	9	82
is	10566	3	6	1	4	2	1	47	127
model	752	8	1	0	1	14	0	6	4
agent	2100	10	3	3	2	3	0	0	36
idea	241	0	0	0	0	0	0	0	0

A modellt ki lehet bővíteni **trigram**má (**trigram**), amely a $P(w_i|w_{i-1}w_{i-2})$ mennyiségeket használja. Ez egy hatékonyabb nyelvi modell, képes megadni, hogy az „ate a banana (banánt ettem)” valószínűbb, mint az „ate a bandana (kendőt ettem)”. A trigram modellben és kisebb mértékben a bigram és unigram modellekben is

[8] Valóban jobb lenne, ha minden valószínűséget egy adott helyzetre vonatkozó feltétel mellett lehetne figyelembe venni. Kevés beszédfelismerő rendszer él ezzel a lehetőséggel, mert nehéz formálisan meghatározni, hogy mi számít adott helyzetnek.

Created by XMLmind XSL-FO Converter.

problémát jelentenek a nulla értékű számlálók: nem szeretnénk egy szókombinációt lehetetlennek minősíteni pusztán azért, mert nem fordulnak elő a tanító korpuszban. A **simítás (smoothing)** folyamata ilyen kombinációkhoz egy kis, pozitív valószínűséget rendel. Ezt a 1. szakasz - Valószínűségi nyelvi modellek részben tárgyaljuk.

A bigram és trigram modellek nem annyira kifinomultak, mint a 22. és 23. fejezetben szereplő nyelvi modellek némelyike, de jobban tekintetbe veszik a helyi összefüggések hatásait, és bizonyos helyi szintaxist sikeresen kifejeznek. Például az a tény, hogy az „I has” és a „man have” szópárok alacsony pontszámokat kapnak, az alany-állítmány egyeztetést fejezi ki. A probléma az, hogy ezeket a kapcsolatokat csak helyileg lehet észlelni: „the man have” alacsony pontszámot kap, de a „the man with yellow hat have” nincs büntetve.

Most gondoljuk meg a nyelvi modell és a szómodellek egyesítését, hogy minden szósorozatot megfelelően tudjunk kezelni. Az egyszerűség kedvéért egy bigram nyelvi modellt fogunk feltételezni. Egy ilyen modellel az összes szómodell (ami viszont kiejtési és beszédhangmodellekből áll) egyetlen nagy RMM-modellbe állítható össze. Az egy-szavas RMM-modell egy állapota egy keret, felcímkézve az aktuális beszédhanggal és beszédhangállapottal (például $[\mathrm{m}]_{\mathrm{Kezdet}}$); egy folytonos beszédű RMM-ben egy állapot a szóval is fel van címkézve, mint például $[\mathrm{m}]^{\mathrm{tomato}}_{\mathrm{Kezdet}}$. Ha minden szónak átlagban p háromállapotú beszédhang szerepel a kiejtési modelljében, akkor W szó esetén a folytonos beszédű RMM-nek $3pW$ állapota van. Átmenetek bekövetkezhetnek egy adott beszédhangon belül a beszédhang állapotai között, egy adott szó beszédhangjai között, illetve egy szó utolsó állapota és egy másik szó kezdő állapota között. A szavak közti átmenetek a bigram modell szerinti valószínűséggel következnek be.

Ha előállítottuk az egyesített RMM-modellt, akkor a felhasználásával elemezhetjük a folytonos beszédjelet. Különösen a (15.9) egyenletben megadott Viterbi-algoritmusnak vehetjük hasznát a legvalószínűbb állapotsorozat megtalálásában. Ebből az állapotsorozatból már kinyerhető a szósorozat egyszerűen az állapotok szócímkéinek kigyűjtésével. Így a Viterbi-algoritmus a szószegmentálás problémáját úgy oldja meg, hogy dinamikus programozást felhasználva (valójában) egyszerre figyelembe veszi az összes lehetséges szósorozatot és szóhatárt.

Vegyük észre, hogy nem azt mondtuk, hogy „kinyerhető a *legvalószínűbb* szósorozat”. A legvalószínűbb szósorozat nem szükségképpen az, ami a legvalószínűbb állapotsorozatot tartalmazza. Ez amiatt van így, mert egy szósorozat valószínűsége az összes olyan állapotszekvencia valószínűségének összege, amely konzisztens ezzel a szósorozattal. Két szósorozatot összehasonlítva, mondjuk „a back” és „aback” szósorozatokat, előfordulhat, hogy tíz alternatív állapotsorozat van az „a back”-hez, ezek mindegyike 0,03 valószínűséggel, de csak egyetlen állapotsorozat az „aback”-hez, viszont 0,20 valószínűséggel. A Viterbi-algoritmus az „aback”-t választja, pedig az „a back” valójában valószínűbb.

A gyakorlatban ez a nehézség nem életveszélyes, de elég komoly ahhoz, hogy más módszereket is kipróbáljanak. A legáltalánosabb választás az **A* dekódoló** (**A* decoder**), ami leleményesen használja fel az A* keresést (lásd 4. fejezet) a legvalószínűbb szósorozat megtalálásában. Az ötlet az, hogy minden szósorozatot egy olyan gráfon átvezető útnak fogunk fel, aminek a csomópontjai a szavakkal vannak felcímkézve. Egy csomópont gyermekei az összes olyan szó, amelyek utána következhetnek; így az összes n hosszúságú vagy rövidebb mondathoz tartozó gráfnak n szintje van, mindegyik W szélességű, ahol W a lehetséges szavak száma. A bigram modellnél a w_1 és w_2 címkéjű csomópontok közti nyílhoz rendelt $g(w_1,w_2)$ költséget a $-\log P(w_2|w_1)$ definiálja. Ekkor egy sorozat teljes útköltsége

$$\mathit{Költség}(w_1 \ldots w_n) = \sum_{i=1}^{n} -\log P(w_i \,|\, w_{i-1}) = -\log \prod_{i=1}^{n} P(w_i \,|\, w_{i-1})$$

Az útköltség ezen definíciója mellett, a legrövidebb út megtalálása pontosan megegyezik a legvalószínűbb szósorozat megtalálásával. A keresés hatékonyságához egy jó $h(w_i)$ heurisztika is szükséges a szósorozat befejezéséhez szükséges költség megbecslésére. Nyilvánvalóan ennek főként a beszédjelnek ahhoz a részéhez kell kapcsolódnia, amit a jelenlegi úton fekvő szavak még nem fedtek le. Eddig még nem született különösebben érdekes heurisztika erre a problémára.

6.4. Egy beszédfelismerő építése

Egy beszédfelismerő rendszer minősége az összes részének a minőségétől függ – a nyelvi modelltől, a szókiejtési modelltől, a beszédhangmodelltől és a jelfeldolgozó algoritmusoktól, amelyek a spektrális jegyeket

Created by XMLmind XSL-FO Converter.

kinyerik az akusztikus jelből. Megtárgyaltuk, hogyan hozhatunk létre egy nyelvi modellt, és a jelfeldolgozás részletei más könyvekben is megtalálhatók. Így már csak a kiejtési és a beszédhangmodellek maradnak. A kiejtési modellek *struktúrája* – mint a paradicsom modelljei a 15.19. ábrán – általában kézi fejlesztésű. Nagy fonetikai szótárak most már elérhetők angol és más nyelvekhez, bár a pontosságuk nagymértékben változik. A háromállapotú beszédhangmodell struktúrája minden beszédhangra ugyanaz, mint a 15.20. ábrán látható. Így már csak a valószínűségek maradnak. Honnan szerezhetők be ezek, különösen hogy a modellek paraméterek százezreit vagy akár millióit is igényelhetik?

Az egyetlen elfogadható módszer a modellek tanulása valódi beszédadatból, amiből bizonyosan nincsen hiány. A következő kérdés a tanulás miként*je*. A teljes választ a 20. fejezetben adjuk meg, de a fő elképzeléseket itt is bemutathatjuk. Gondoljuk át a bigram nyelvi modellt; ennek megtanulását már elmagyaráztuk, ami a szópárok gyakoriságának a vizsgálatán alapult valódi szövegben. Megtehető ugyanez mondjuk a beszédhangok átmenet-valószínűségeire is a kiejtési modellben? A válasz igen, de csak akkor, ha valaki veszi a fáradságot, és végigjegyzeteli az összes előforduló szót a helyes beszédhangsorozattal. Ez egy bonyolult és számos hibalehetőséget magában rejtő feladat, de néhány standard adathalmazra elvégezték, amelyek több órás beszédet tartalmaznak. Ha ismerjük a beszédhang-szekvenciát, akkor a kiejtési modell átmenet-valószínűségeit megbecsülhetjük a beszédhangpárokból. Hasonlóan, ha minden keretre ismert a beszédhangállapot – egy még gyötrelmesebb kézi felcímkézés eredményeképpen – akkor megbecsülhetjük a beszédhangmodell átmenet-valószínűségeit is. Ha minden keretben ismerjük a beszédhangállapotot és az akusztikus jegyeket, akkor pedig megbecsülhetjük az akusztikai modellt, akár közvetlenül a gyakoriságokból (a VK modellek esetében) vagy statisztikai illesztési módszerekkel (a Gauss-modellek keveréke esetében; lásd 20. fejezet).

Fontos

A kézi címkézésű adatnak a költsége és a ritkasága, illetve az a tény, hogy az elérhető kézi címkézésű adathalmazok nem feltétlenül reprezentálják a beszélők és az akusztikus feltételek azon típusait, amelyek egy új beszédfelismerési feladatban fellépnek, ezt a megközelítést bukásra ítélik. *Szerencsére a* ***várhatóérték-maximalizálás*** *vagy* ***EM algoritmus*** *(****expectation-maximization****,* ***EM****) anélkül is megtanulja az RMM átmenetvalószínűségeit és érzékelő modelljeit, hogy címkézett adatot használna.* Kézi címkézésű adatokból származó becsléseket felhasználhatunk a modell kezdeti beállítására; ezután az EM következik, és betanítja a modellt az éppen aktuális feladatra. Az ötlet egyszerű: egy adott RMM és egy megfigyeléssorozat esetén, felhasználhatjuk a 15.2. és a 15.3. alfejezet simító algoritmusait az egyes állapot-valószínűségek kiszámítására minden időpontban, illetve egy egyszerű kiterjesztéssel az állapot-állapot párok valószínűségét is az egymást követő időpontokban. Ezeket a valószínűségeket felfoghatjuk bizonytalan címkéknek. A bizonytalan címkékből megbecsülhetünk új átmenet- és érzékelési valószínűségeket, majd az EM eljárás megismétlődik. A módszer minden iterációban bizonyítottan növeli a modell adathoz való illeszkedését, és általában a kezdetben beállított kézi címkézésű becslések értékeinél, a paraméterértékek egy sokkal jobb halmazához konvergál.

A legkorszerűbb beszédfelismerő rendszerek óriási adathalmazokat és hatalmas számítási erőforrásokat használnak fel a modelljeik tanításához. Egyedülálló szófelismerésnél, jó akusztikus feltételekkel (háttérzaj és visszhang nélkül), egy néhány ezres szótárnál és egy beszélő esetén, a pontosság 99% feletti lehet. Folytonos beszédfelismerésnél és több különféle beszélő esetén, a 60–80%-os pontosság az általános, még jó akusztikus körülmények esetén is. Háttérzajnál és telefonos átvitelnél a pontosság tovább romlik. Bár az üzembe helyezett rendszerek évtizedek óta javulnak, számos ötlet még kidolgozásra vár.

7. Összefoglalás

Ez a fejezet az ábrázolás és a következtetés általános problémáját tárgyalta időbeli valószínűségi folyamatok esetében. A legfőbb pontok a következők:

- A világ változó állapotát valószínűségi változók halmazának a felhasználásával kezeljük, amelyek az állapotot ábrázolják minden időpontban.
- Kidolgozhatók olyan reprezentációk, amelyek eleget tesznek a **Markov-tulajdonság**nak (**Markov property**), nevezetesen, hogy a jövő független a múlttól a jelen ismeretében. Ez együttesen a folyamat **stacionaritás**ának (**stationary**) feltevésével – azaz azzal, hogy a dinamika az idővel nem változik – nagyban egyszerűsíti a reprenzentációt.

Created by XMLmind XSL-FO Converter.

- Egy időbeli valószínűségi modellt felfoghatunk egy **állapotátmenet-modell** (**transition model**) és egy **érzékelő modell** (**sensor model**) együttesének, ahol az első az állapot alakulását, a második pedig a megfigyelés folyamatát írja le.

- Időbeli modellekben az alapvető következtetési feladatok a **szűrés** (**filtering**), az **előrejelzés** (**prediction**), a **simítás** (**smoothing**) és a **legvalószínűbb magyarázat kiszámítása** (**most likely explanation**). Ezek mindegyike elvégezhető egyszerű, rekurzív algoritmusokkal, melyek futási ideje a szekvencia hosszában lineáris.

- Időbeli modellek három családját tanulmányoztuk részletesebben: a **rejtett Markov-modell**eket (**hidden Markov model**), a **Kalman-szűrő**ket (**Kalman filter**s), és a **dinamikus Bayes-hálók**at (**dynamic Bayesian network**s) (ami magában foglalja a másik kettőt, mint speciális esetet).

- A **beszédfelismerés** (**speech recognition**) és a **követés** (**tracking**) az időbeli valószínűségi modellek két fontos alkalmazási területe.

- Hacsak nincsenek speciális feltételezéseink, mint a Kalman-szűrőknél, egzakt következtetés több állapotváltozó esetén nem kivitelezhető. A gyakorlatban a **részecskeszűrő** (**particle filtering**) algoritmus tűnik egy hatékony közelítő algoritmusnak.

7.1. Irodalmi és történeti megjegyzések

Számos alapötlet dinamikus rendszerek állapotának a becslésére a matematikus C. F. Gausstól származik, aki egy determinisztikus legkisebb-négyzeteken alapuló algoritmust alkotott meg az égitestek pályájának a megbecslésére csillagászati megfigyelésekből (Gauss, 1809). Az orosz matematikus A. A. Markov fejlesztette ki a később róla elnevezett Markov-feltételezést a sztochasztikus folyamatokat vizsgálva; az elsőrendű Markov-láncot maga Markov alkalmazta először a *Jevgenyij Anyegin* szövegének betűszekvencia-elemzésénél (Markov, 1913). Jelentős titkosított munka folyt a szűrés problémáján a második világháború alatt folytonos idejű folyamatok esetében (Wiener, 1942) és diszkrét idejű folyamatoknál is (Kolmogorov, 1941). Bár ez a munka fontos technikai fejlődést eredményezett a következő 20 évben, a felhasznált frekvenciatartománybeli reprezentáció több számítást is elég nehézkessé tett. A sztochasztikus folyamatok közvetlen állapottérbeli modellezése egyszerűbbnek bizonyult, ahogy azt Swerling és Kalman munkái megmutatták (Swerling, 1959; Kalman, 1960). Az utóbbi munka vezette be a most Kalman-szűrésnek nevezett technikát előrefelé következtetés elvégzésére lineáris rendszerekben Gauss-zaj mellett. Rauch és társai fontos eredményeket értek el a simításhoz kapcsolódóan, a megkapóan elnevezett Rauch–Tung–Striebel-simítás még ma is szabványtechnika (Rauch és társai, 1965). Számos korai eredményt gyűjtöttek össze a (Gelb, 1974) kiadványban. Bar-Shalom és Fortmann egy modernebb tárgyalást kínál bayesi vonásokkal, miközben számos hivatkozást is ad a hatalmas mennyiségű szakirodalomra (Bar-Shalom és Fortmann, 1988). Chatfield az idősor analízis „klasszikus” megközelítését fedi le (Chatfield, 1989).

A Kalman-szűrés számos alkalmazásában nem csak a bizonytalan érzékeléssel és a dinamikával kell foglalkozni, hanem bizonytalan azonosítással is; azaz ha több objektumot kell megfigyelni, a rendszernek meg kell határozni, hogy melyik megfigyelést melyik objektum generálta, mielőtt az állapotbecsléseket felfrissíthetné. Ez az **adattársítás** (**data association**) problémája (Bar-Shalom és Fortmann, 1988; Bar-Shalom, 1992). Azon esetben, amikor n megfigyelés és n követés van (ami egy igen szelíd eset), a megfigyelések $n!$ lehetséges módon rendelhetők a követésekhez; a helyes valószínűségi kezelés mindet figyelembe venné, ami megmutathatóan NP-teljes (Cox, 1993; Cox és Hingorani, 1994). Az MCMC-alapú polinom idejű közelítő módszerek úgy tűnik, jól működnek a gyakorlatban (Pasula és társai, 1999). Érdekes, hogy az adattársítás problémája egy elsőrendű nyelvben végzett valószínűségi következtetés egy esete; a legtöbb valószínűségi következtetési problémától eltérően, amelyek tisztán ítéletlogikai szinten vannak, az adattársítás objektumokat és azonossági relációt foglal magában. Ezért közvetlenül kapcsolódik a 14. fejezetben említett elsőrendű valószínűségi nyelvekhez. Új kutatások megmutatták, hogy általában az azonosság feletti érvelés és konkrétan az adattársítás elvégezhető az elsőrendű valószínűségi keretben (Pasula és Russell, 2001).

A rejtett Markov-modellt és a kapcsolódó következtető és tanuló algoritmusokat, beleértve az előre-hátra algoritmust, Baum és Petrie fejlesztették ki (Baum és Petrie, 1966). Ettől függetlenül hasonló ötletek megjelentek a Kalman-szűréssel foglalkozó kutatásnál is (Rauch és társai, 1965). Az előre-hátra algoritmus volt az egyik fő előzménye az EM algoritmus általános megfogalmazásának (Dempster és társai, 1977); lásd még 20. fejezet. Az állandó idejű simítás Bindernek és munkatársainak a munkájában jelenik meg, csakúgy mint az „oszd meg és uralkodj” algoritmus, amit a 15.3. feladatban mutatunk be (Binder és társai, 1997b).

Created by XMLmind XSL-FO Converter.

A dinamikus Bayes-hálókra tekinthetünk úgy, mint a Markov-folyamatok egy ritka kódolására, és az MI-n belül elsőként a következő munkákban használták fel: (Dean és Kanazawa, 1989b; Nicholson, 1992; Kjaerulff, 1992). A legutóbbi munkák egyike a Hugin Bayes-hálós rendszer általános kiterjesztése, ami dinamikus Bayes-hálók generálásához és szerkesztéséhez biztosítja a szükséges eszközöket. A dinamikus Bayes-hálók népszerűvé váltak különböző komplex mozgási folyamatok modellezésére a számítógépes látás területén (Huang és társai, 1994; Intille és Bobick, 1999). A kapcsolatot az RMM-k és a DBH-k között, illetve az előre-hátra algoritmus és a Bayes-hálós terjesztés között Smyth és társai tisztázták (Smyth és társai, 1997). Egy további egyeségesítés a Kalman-szűrőkkel (és más statisztikai modellekkel) Roweis és Ghahramani munkájában jelenik meg (Roweis és Ghahramani, 1999).

A 15.5. fejezetben leírt részecskeszűrő algoritmusnak különösen érdekes története van. Az első szűrésre szolgáló mintavételi algoritmus a szabályozáselmélet területén született (Handschin és Mayne, 1969), és az újramintavételezés ötlete, ami a részecske- szűrés központi eleme egy szabályozástechnikával foglalkozó orosz folyóiratban jelent meg (Zaritskii és társai, 1975). Később többször újra felfedezték: a statisztika területén mint **szekvenciális fontossági mintavételezésű újramintavételezés** (**SUFM**) (**sequential importance-sampling resampling**, **SIR**) (Rubin, 1988; Liu és Chen, 1998), a szabályozáselmélet területén mint részecskeszűrés (Gordon és társai, 1993; Gordon, 1994), a mesterséges intelligencia területén mint **a legjobb túlélése** (**survival of the fittest**) (Kanazawa és társai, 1995) és a számítógépes látásban mint **sűrítés** (**condensation**) (Isard és Blake, 1996). Kanazawa munkája egy javítást is tartalmazott, az úgynevezett **bizonyítékmegfordítás**t (**evidence reversal**), ami által a $t + 1$ időpontban az állapot mintavételezése a t időpontbeli állapot és a $t + 1$ *időpontbeli bizonyíték* feltételek mellett történik (Kanazawa és társai, 1995). Ez lehetővé teszi, hogy a bizonyíték közvetlenül befolyásolja a minták generálását és bizonyítottan csökkenti a közelítési hibát (Doucet, 1997).

A közelítő szűrésre szolgáló módszerek között találjuk a **lecsengő MCMC** (**decayed MCMC**) algoritmust (Marthi és társai, 2002) és a faktorizált közelítés módszerét Boyen és társaitól (Boyen és társai, 1999). Mindkét módszer rendelkezik azzal a fontos tulajdonsággal, hogy a közelítési hiba nem nő az idővel. Variációs technikákat (lásd 14. fejezet) szintén fejlesztettek ki időbeli modellekre. Ghahramani és Jordan egy közelítő algoritmust ismertet faktoriális RMM-kre, így olyan DBH-ra, amiben két vagy több függetlenül fejlődő Markov-lánc egy közös megfigyelési folyammal van összekapcsolva (Ghahramani és Jordan, 1997). Jordan és társai számos más alkalmazást is tárgyalnak (Jordan és társai, 1998). A keverési idők tulajdonságait Pak, valamint Luby és Vigoda tárgyalja (Pak, 2001; Luby és Vigoda, 1999).

A beszédfelismerés őstörténete az 1920-as években kezdődött Radio Rexszel, a hangvezérlésű játék kutyával. Rex ugrált az 500 Hz körüli hangfrekvenciákra válaszul, ami az [eh] magánhangzóhoz tartozik a „Rex!”-ben. Kissé komolyabb munka a második világháború után kezdődött. Az AT&T Bell Labsnál egy rendszert építettek egyedülálló számjegyek felismerésére akusztikus jellemzők egyszerű mintaillesztésével (Davis és társai, 1952). A beszédhang átmenet-valószínűségeket először egy, a londoni University College-ban épített rendszerben használtak (Fry, 1959; Denes, 1959). 1971-ben az Egyesült Államok Védelmi Minisztériumának Kutatási Ügynöksége (Defense Advanced Research Project Agency, DARPA) négy ötéves kutatási tervet kezdett el finanszírozni, hogy nagy teljesítményű beszédfelismerő rendszereket fejlesszenek ki. A győztes, és egyben az egyetlen rendszer, ami az 1000 szavas szótáron a kitűzött 90%-os pontosságot elérte, a HARPY rendszer volt a CMU-ról (Lowerre, 1976; Lowerre és Reddy, 1980).[9] A HARPY végső változatát egy DRAGON nevű rendszerből származtatták, amit egy CMU-s diák, James Baker épített (Baker, 1975). A DRAGON volt az első rendszer, ami RMM-eket használt beszédre. Majdnem egy időben Jelinek az IBM-nél kifejlesztett egy másik RMM-alapú rendszert (Jelinek, 1976). Ettől az időponttól kezdve, a valószínűségi módszerek általában is, de az RMM-ek különösen egyre inkább dominálták a beszédfelismerés kutatását és fejlesztéseit. Az utóbbi éveket a gyarapodó fejlődés, a nagyobb adathalmazok és modellek, és a szigorúbb verseny realisztikusabb beszédhelyzeteken jellemzi. Egyes kutatók megvizsgálták a DBH-k felhasználásának lehetőségét az RMM-ek helyett, azzal a céllal, hogy a DBH-k nagyobb kifejezőerejét kihasználva a beszédképző szervek komplex rejtett állapotából többet tudjanak megragadni (Zweig és Russell, 1998; Richardson és társai, 2000).

A beszédfelismeréshez jó bevezető irodalom (Rabiner és Juang, 1993; Jelinek, 1997; Gold és Morgan, 2000; Huang és társai, 2001). A témához tartozó fontos cikkek gyűjteménye (néhány áttekintő cikket is beleértve) (Waibel és Lee, 1990). Jelen fejezet anyagát Kay, Gawron és Norvig áttekintő munkájára (Kay és társai, 1994), valamint Jurafsky és Martin egyik könyvére alapoztuk (Jurafsky és Martin, 2000). A beszédfelismerésről kutatási eredményeket közölnek a *Computer Speech and Language,* a *Speech Communication* és az *IEEE*

[9] A versenyben másodikként végző rendszernek, a HEARSAY–II-nek az MI más ágaira volt nagy hatása, mert a **táblaarchitektúrá**t (**blackboard architecture**) használta (Erman és társai, 1980). Ez egy szabályalapú szakértői rendszer volt, több többé-kevésbé független moduláris **tudásforrás**sal (**knowledge source**) rendelkezett, amelyek egy közös **tábla** (**blackboard**) révén kommunikáltak egymással, olvashattak a tábláról és írhattak rá. A táblaarchitektúrát használó rendszerek a modern felhasználói felületek architektúráinak az alapjai.

Created by XMLmind XSL-FO Converter.

Transactions on Acoustics, Speech, and Signal Processing folyóiratok, valamint a *DARPA Workshops on Speech and Natural Language Processing* kiadványok, továbbá a Eurospeech, az ICSLP és az ASRU konferenciák kiadványai.

7.2. Feladatok

15.1.

Mutassa meg, hogy bármely másodrendű Markov-folyamatot át lehet írni elsőrendű Markov-folyamattá az állapotváltozóknak egy megnövelt halmazát használva. Megtehető-e ez mindig *takarékosan,* azaz az állapotátmenet-modellt megadó paraméterek számának a növelése nélkül?

15.2.

Ebben a feladatban azt vizsgáljuk, hogy mi történik a valószínűségekkel az esernyős világban, ha az idősorok hossza a végtelenbe tart.

a. Tételezzük fel, hogy a napok olyan végtelen sorát figyeljük meg, amikor az esernyő mindig feltűnik. Mutassa meg, hogy a napok múlásával az eső valószínűsége az aktuális napon monoton növekszik egy határértékhez. Számítsa ki ezt a határértéket.

b. Most gondolja át az *előrejelzés*-t az egyre távolabbi jövőben, csupán a két első esernyő megfigyelés ismeretében. Először számolja ki a $P(R_{2+k} | U_1, U_2)$ valószínűséget $k = 1 \ldots 20$ esetén, és ábrázolja az eredményeket. A valószínűségnek láthatóan konvergálnia kell egy határértékhez. Számítsa ki a határérték pontos értékét.

15.3.

Ez a feladat a 15.4. ábrán leírt előre-hátra algoritmus egy tártakarékos változatát fejleszti ki. A $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ valószínűséget szeretnénk kiszámítani $k = 1, \ldots, t$-re. Ezt egy „oszd meg és uralkodj” séma szerint végezzük el.

a. Tételezzük fel az egyszerűség kedvéért, hogy t páratlan, és legyen a felezőpont $h = (t + 1) /2$. Mutassa meg, hogy $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ kiszámítható $k = 1, \ldots, h$ esetén csupán az $\mathbf{f}_{1:0}$ kezdő előrefelé üzenetet, a $\mathbf{b}_{h+1:t}$ visszafelé üzenetet és az $\mathbf{e}_{1:h}$ bizonyítékokat ismerve.

b. Mutassa meg, hogy a sorozat második felére is fennáll egy hasonló eredmény.

c. Az (a) és (b) eredmények ismeretében egy rekurzív „oszd meg és uralkodj” algoritmust lehet létrehozni, először előrefelé haladva a sorozaton és aztán visszafelé a végétől, csak a szükséges üzeneteket tárolva középen és a végeken. Aztán az algoritmus lefut mindkét félre. Írja le az algoritmust részletesen.

d. Számítsa ki az algoritmus tár- és időkomplexitását a sorozat hosszának, t-nek a függvényében. Hogyan változik ez, ha felosztjuk a bemenetet két részre?

15.4.

A 2.3. szakasz - A legvalószínűbb sorozat megtalálása részben vázoltunk egy hibás eljárást a legvalószínűbb állapotszekvencia megtalálására egy adott megfigyelési szekvencia esetén. Az eljárás azon alapul, hogy megkeresi a legvalószínűbb állapotokat minden egyes időpontra, és visszaad egy ezen állapotokból álló sorozatot. Mutassa meg, hogy bizonyos időbeli valószínűségi modellekre és megfigyelési sorozatokra, ez az eljárás egy lehetetlen állapotsorozatot ad vissza (azaz amely sorozatnak az a posteriori valószínűsége nulla).

15.5.

Gyakran szeretnénk megfigyelni egy olyan folytonos állapotú rendszert, aminek a viselkedése megjósolhatatlanul vált k különböző „mód” között. Például egy repülőgép egy rakéta elkerülése közben különböző manővereket hajthat végre, melyeket a rakéta megpróbálhat követni. Egy ilyen **váltó Kalman-szűrő** Bayes-háló reprezentációja a 15.22. ábrán látható.

a. Tételezzük fel, hogy az S_t diszkrét állapotnak k lehetséges értéke van, illetve, hogy a $\mathbf{P}(\mathbf{X}_0)$ a priori folytonos állapotbecslés egy többváltozós Gauss-eloszlás. Mutassa meg, hogy a $\mathbf{P}(\mathbf{X}_1)$ előrejelzés **Gauss-eloszlások**

Created by XMLmind XSL-FO Converter.

keveréke (**mixture of Gaussians**) – azaz Gauss-eloszlások súlyozott összege, ahol a súlyok 1-re összegződnek.

b. Mutassa meg, hogy ha a jelenlegi folytonos állapot becslése, $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$, m Gauss-eloszlás keveréke, akkor általános esetben a $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1})$ frissített állapotbecslés km Gauss-eloszlás keveréke lesz.

c. Az időbeli folyamatnak milyen jellegét reprezentálják a súlyok a Gauss-keverékben?

Az (a) és (b) eredmények együtt azt mutatják, hogy az a posteriori eloszlás reprezentációja korlátozás nélkül nő, még a váltó Kalman-szűrők esetében is, amelyek a legegyszerűbb hibrid dinamikus modellek.

15.6.

Pótolja a hiányzó lépést a (15.17) egyenlet levezetésében, az első frissítési lépést az egydimenziós Kalman-szűrőnél.

15.7.

Vizsgáljuk meg a szórásnégyzet-frissítés alakulását a (15.18) egyenletben.

a. Ábrázolja a σ_t^2 értékét a t függvényében, adott σ_x^2 és σ_z^2 értékek mellett.

b. Mutassa meg, hogy a frissítésnek létezik egy σ^2 határértéke úgy, hogy $t \rightarrow \infty$ esetén $\sigma_t^2 \rightarrow \sigma^2$, és számítsa ki ezt a σ^2 értéket.

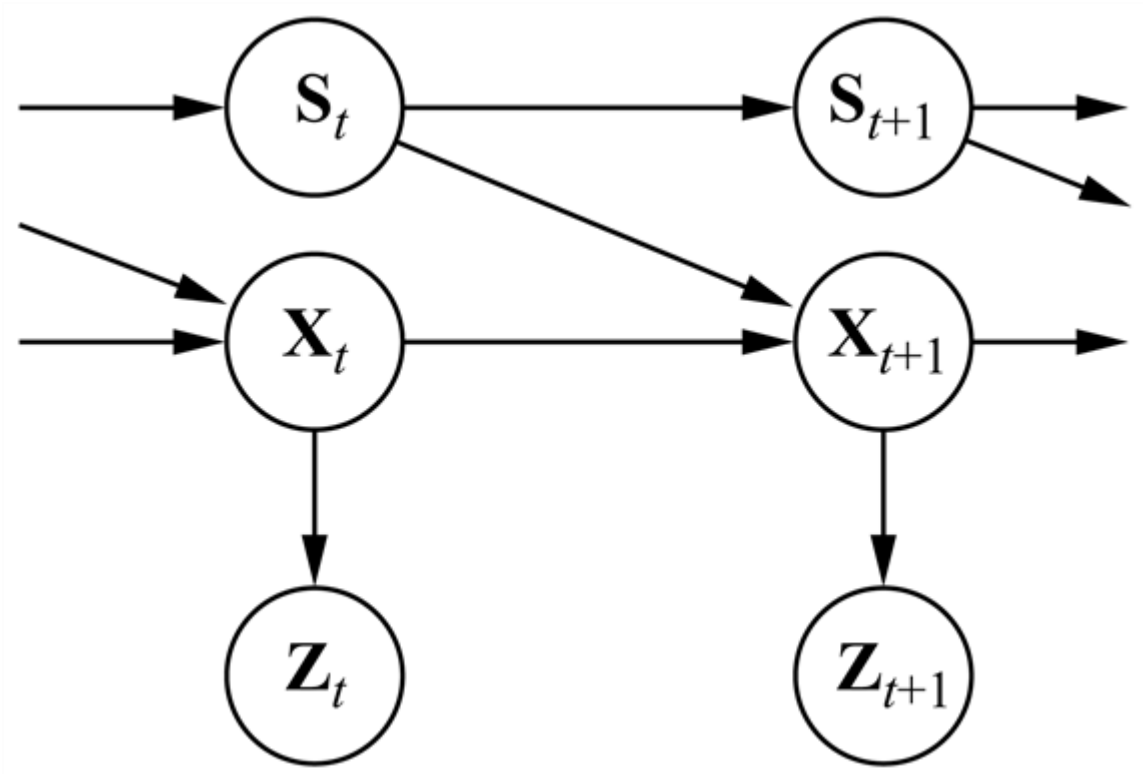
c. Adjon kvalitatív magyarázatot arra, hogy mi történik, ha $\sigma_x^2 \rightarrow 0$ és $\sigma_z^2 \rightarrow 0$.

15.8.

Mutassa meg, hogyan reprezentálna egy RMM-et egy rekurzív relációs valószínűségi modellként, amint azt a 14.6. fejezetben javasoltuk.

15.22. ábra - Egy váltó Kalman-szűrő Bayes-háló reprezentációja. Az S_t váltó változó egy diszkrét állapotváltozó, aminek az értéke meghatározza az $\mathbf{X}_t$ folytonos állapotváltozók átmeneti modelljét. Minden i diszkrét állapotra a $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{X}_t, S_t = i)$ állapotátmenet-modell egy lineáris Gauss-modell, pontosan úgy, mint egy szabályos Kalman-szűrőben. A diszkrét állapotok közti $\mathbf{P}(S_{t+1}|S_t)$ állapotátmenet-modellt egy mátrixnak tekinthetjük, ahogyan egy rejtett Markov-modellben.

Created by XMLmind XSL-FO Converter.



15.9.

Ebben a feladatban részletesebben elemezzük az akkumulátorérzékelő tartóshiba-modellt (lásd 15.13. (a) ábra).

a. A 15.13. (b) ábra véget ér $t = 32$-nél. Írja le kvalitatívan, mi történik $t \rightarrow \infty$ esetén, ha az érzékelő továbbra is 0-t mutat.

b. Tegyük fel, hogy a külső hőmérséklet befolyásolja az akkumulátor érzékelőjét, méghozzá olyan módon, hogy a hőmérséklet emelkedésével az átmeneti hibák valószínűbbekké válnak. Mutassa meg, hogyan egészítené ki a 15.13. (a) ábra DBH-struktúráját, és magyarázza el az esetlegesen szükséges változtatásokat az FVT-kben.

c. Az új hálóstruktúra esetén felhasználhatja-e a robot az akkumulátormméréseket az aktuális hőmérséklet kikövetkeztetésére?

15.10.

Gondoljuk meg a változó eliminálás algoritmusának az alkalmazását az esernyős DBH három szeletre történő kibontása esetén a $\mathbf{P}(R_3|U_1, U_2, U_3)$ kérdés megválaszolására. Mutassa meg, hogy az algoritmus komplexitása – a legnagyobb tényező mérete – ugyanaz, függetlenül attól, hogy az eső változókat előrefelé vagy visszafelé sorrendben eliminálják.

15.11.

A 15.19. ábra „tomato” modellje megengedi a koartikulációt az első magánhangzón két lehetséges beszédhangot kínálva fel. Egy alternatív megközelítés a háromállapotú modell használata, amelyben az [ow(t,m)] beszédhang automatikusan magában foglalja a magánhangzó változását. Rajzoljon egy teljes háromállapotú modellt a „tomato”-ra, ideértve a tájszólásos változatokat.

15.12.

Számolja ki a legvalószínűbb utat a 15.20. ábra RMM-én keresztül a $[C_1, C_2, C_3, C_4, C_5, C_6, C_7]$ kibocsátási sorozat esetén. Adja meg ennek valószínűségét is.

Created by XMLmind XSL-FO Converter.

16. fejezet - Egyszerű döntések meghozatala

Ebben a fejezetben láthatjuk, hogyan kell egy ágensnek döntéseket úgy meghoznia, hogy elérje, amit akar – az esetek nagy részében legalábbis.

Ebben a fejezetben visszatérünk a hasznosságelmélet gondolatához, amit a 13. fejezetben vezettünk be. Megmutatjuk a hasznosságelméletnek a valószínűség-számítással való összekapcsolását egy döntéselméleti ágens létrehozásához, amely a meggyőződéseinek és célkitűzéseinek megfelelő racionális döntéseket képes hozni. Az ilyen ágensek képesek döntéseket hozni olyan esetekben is, amikor a bizonytalanság és az ellentétes célok egy logikai ágens számára nem tennék lehetővé a döntést. Egy célorientált ágens valójában az állapotokat két csoportba, egy jó (cél) és egy rossz (nem cél) csoportba sorolja be, míg egy döntéselméleti ágens az állapotok jóságát egy folytonos mértékkel fejezi ki.

A 16.1. alfejezet a döntéselmélet alapvető elveit vezeti be: a várható hasznosság maximálását. A 16.2. alfejezet megmutatja, hogy bármely racionális ágens viselkedése leírható egy hasznosságfüggvény feltételezésével és egy azon alapuló maximálással. A 16.3. alfejezet részletesebben megvizsgálja a hasznosságfüggvény tulajdonságait, különösen ezek kapcsolatát olyan egyedi mennyiségekhez, mint a pénz. A 16.4. alfejezet ismerteti, hogyan lehet a több mennyiségtől függő hasznosságfüggvényeket kezelni. A 16.5. alfejezet leírja a döntéshozó rendszerek megvalósítását. Nevezetesen, bevezetjük a **döntési hálózat**ok (**decision network**s) formalizmust – más néven **hatásdiagram**ot, (**influence diagram**) –, ami kiterjeszti a valószínűségi hálózatokat, hogy a „cselekvéseket” és a „hasznosságokat” is tudják kezelni. A fejezet többi része azokat a kérdéseket vizsgálja, amelyek a döntéselméletnek a szakértői rendszerekkel kapcsolatos alkalmazásakor lépnek fel.

1. Meggyőződések és kívánságok összekapcsolása bizonytalanság esetén

Arnauld francia filozófus az 1662-ben írt *Port-Royal Logic* c. művében azt írta, hogy

Annak megítéléséhez, hogy valaki elérje a jót, és elkerülje a gonoszt, nemcsak a jót és a gonoszt kell önmagában megfontolni, hanem annak valószínűségét is, hogy ezek megtörténnek-e vagy sem; és azt az arányt kell megnézni, amely ezek együtteséhez tartozik.

A kortárs tudományos szövegek inkább hasznosságról beszélnek, mint jóról és gonoszról, de az elvek ugyanazok. Az ágens preferenciáit a világ állapotai között egy **hasznosságfüggvény** (**utility function**) adja meg, ami az egyes állapotok kívánatosságának kifejezésére minden állapothoz egyetlen számot rendel. A hasznosságokat a cselekedetek következményeinek a valószínűségével kombinálva kapjuk az egyes cselekedetekhez tartozó várható hasznosságot.

Egy *S* állapotnak a döntést meghozó ágens szempontja szerinti hasznosságára az *U*(*S*) jelölést fogjuk használni. Mostani vizsgálódásunknál az állapotokat a világ teljes pillanatfelvételeinek fogjuk tekinteni, hasonlóan a 10. fejezetben szereplő **szituáció**khoz (**situation**s). Bár ez egyszerűsíti a kezdeti fejtegetésünket, a hasznosság definiálása minden egyes állapotra külön-külön elég nehézkessé válhat. A 16.4. alfejezetben látni fogjuk, hogy az állapotok hogyan bonthatók fel bizonyos körülmények között a hasznosság hozzárendelése érdekében.

Egy nemdeterminisztikus *A* cselekvésnek az $Eredmény_i(A)$ állapotok a lehetséges következményei, ahol az *i* index a különböző következményeken fut végig. Az *A* végrehajtása előtt az ágens egy $P(Eredmény_i(A)|Tesz(A), E)$ valószínűséget rendel minden egyes következményhez, ahol az *E* az ágens által a világról elérhető tényeket jelöli, és a *Tesz*(*A*) egy állítás, hogy az *A* cselekvés végrehajtódik a jelenlegi állapotban. Ekkor a következő formulával kiszámíthatjuk a cselekvés *EU*(*A*|*E*) **várható hasznosság**át (**expected utility**) adott tények esetén:

$$EU(A|E) = \sum_i P(Eredmény_i(A)|Tesz(A), E)\, U(Eredmény_i(A)) \qquad (16.1)$$

A **maximális várható hasznosság** (**MVH**) (**maximum expected utility**, **MEU**) elve azt mondja ki, hogy egy racionális ágensnek azt a cselekvést kell választania, ami maximalizálja az ágens várható hasznosságát. Ha cselekvések egy legjobb sorozatát szeretnénk kiválasztani ennek az egyenletnek a felhasználásával, akkor az

Created by XMLmind XSL-FO Converter.

összes lehetséges cselekvéssorozatot számba kellene venni, és a legjobbat kiválasztani, ami hosszú sorozatok esetén nyilvánvalóan nem lehetséges. Ezért ez a fejezet egyszerű döntésekre (általában egyetlen cselekvésre vonatkozó döntésekre) koncentrál, és a következő fejezet mutat be új technikákat cselekvéssorozatok hatékony kezelésére.

Bizonyos értelemben az MVH-elv felfogható a teljes MI meghatározásának. Hiszen egy intelligens ágensnek mindössze annyit kell tennie, hogy kiszámítja a különféle mennyiségeket, maximalizálja a hasznosságot a cselekvései felett, és kész. Ám ez nem jelenti, hogy ezzel a definícióval az MI problémaköre meg lenne *oldva!*

Ámbár az MVH-elv bármely döntési helyzet esetén meghatározza a helyes cselekvést, a szükséges számítások lehetnek kivitelezhetetlenek, és néha maga a probléma megfogalmazása is bonyolult. A világ kezdeti állapotának ismerete érzékelést, tanulást, tudásreprezentációt és következtetést igényel. A $P(Eredmény_i(A)|Tesz(A), E)$ kiszámítása a világ teljes okozati modelljét igényli, és – ahogyan azt a 14. fejezetben láttuk – NP-teljes számítást a valószínűségi hálózatokban. Az egyes állapotok $U(Eredmény_i(A))$ hasznosságának a kiszámítása gyakran keresést vagy tervezést igényel, mivel az ágens nem tudja, hogy egy állapot mennyire jó addig, ameddig nem tudja, hogy hova is kerülhet ebből az állapotból. Így a döntéselmélet nem csodaszer, ami megoldja az MI-problémát. Másrészről azonban, ez egy olyan keretet ad, amelyben áttekinthető egy MI-rendszer összes részének a beilleszkedése.

Fontos

Az MVH-elv egyértelműen kapcsolódik a 2. fejezetben felvetett teljesítménymértékek elképzeléshez. Az alapötlet nagyon egyszerű. Tekintsük azokat a lehetséges környezeteket, amelyek érzékeléssel és memóriával rendelkező ágenseket tesznek lehetővé, és gondoljuk át azt, hogy milyen lehetséges ágenseket tudnánk tervezni. *Ha egy ágens maximalizálja a hasznosságfüggvényét, és az helyesen tükrözi a teljesítmény mértékét, amivel a viselkedését megítélik, akkor ez a lehető legmagasabb teljesítménypontszámot éri el, ha a hasznosságot azon lehetséges környezetek felett átlagoljuk, amelyekbe az ágens kerülhet.* Ez az alapvető igazolása az MVH-elvnek. Bár az állítás tautológiának tűnhet, valójában nagyon fontos átmenetet jelent a racionalitás globális, külső kritériumaitól – a múltbeli környezetek felett vett teljesítménytől – egy lokális, belső kritériumig, ami a következő állapotra vonatkozó hasznosságfüggvény maximalizálását jelenti.

Ebben a fejezetben csak az egyszeri vagy **egylépéses döntés**ekkel (**one-shot decision**s) foglalkozunk, bár a 2. fejezet a teljesítménymértékeket a múltbeli környezetek felett definiálja, amelyek általában számos döntést tartalmaznak. A következő fejezetben, ami a **szekvenciális döntés**ekkel (**sequential decision**s) foglalkozik, megmutatjuk, hogy ez a két nézet hogyan békíthető össze.

2. A hasznosságelmélet alapjai

Ránézésre az MVH-elv ésszerű módszernek látszik döntések meghozatalára, de az nem világos, hogy vajon ez-e az *egyetlen* lehetőség. Végül is, a hasznosság *átlagának* maximálása miért is lenne olyan különleges? Miért nem maximalizáljuk a lehetséges hasznosságok köbeinek összegét, vagy miért nem minimalizáljuk a lehetséges veszteségek maximumát? Hasonlóan, miért nem cselekedhet az ágens racionálisan, ha csupán az állapotok közötti preferenciáit rögzítené, és nem rendelne hozzájuk numerikus értéket? Végül, egyáltalán miért kell léteznie egy megkövetelt tulajdonságokkal rendelkező hasznosságfüggvénynek? Talán egy racionális ágensnek olyan bonyolult a preferenciarendszere, hogy azt nem lehet ilyen egyszerűen az egyes állapotokhoz rendelt valós számokkal kifejezni.

2.1. Megkötések a racionális preferenciákra

Ezek a kérdések megválaszolhatók, ha leírjuk a preferenciákra vonatkozó azon megkötéseket, amelyekkel egy racionális ágensnek rendelkeznie kell, majd ezután megmutatjuk, hogy az MVH-elv ezekből a megkötésekből levezethető. Az ágens preferenciáit a következő jelöléssel írjuk le:

$A \succ B$ *A preferált B-hez képest*

$A \sim B$ *az ágens egyformán preferálja A-t és B-t*

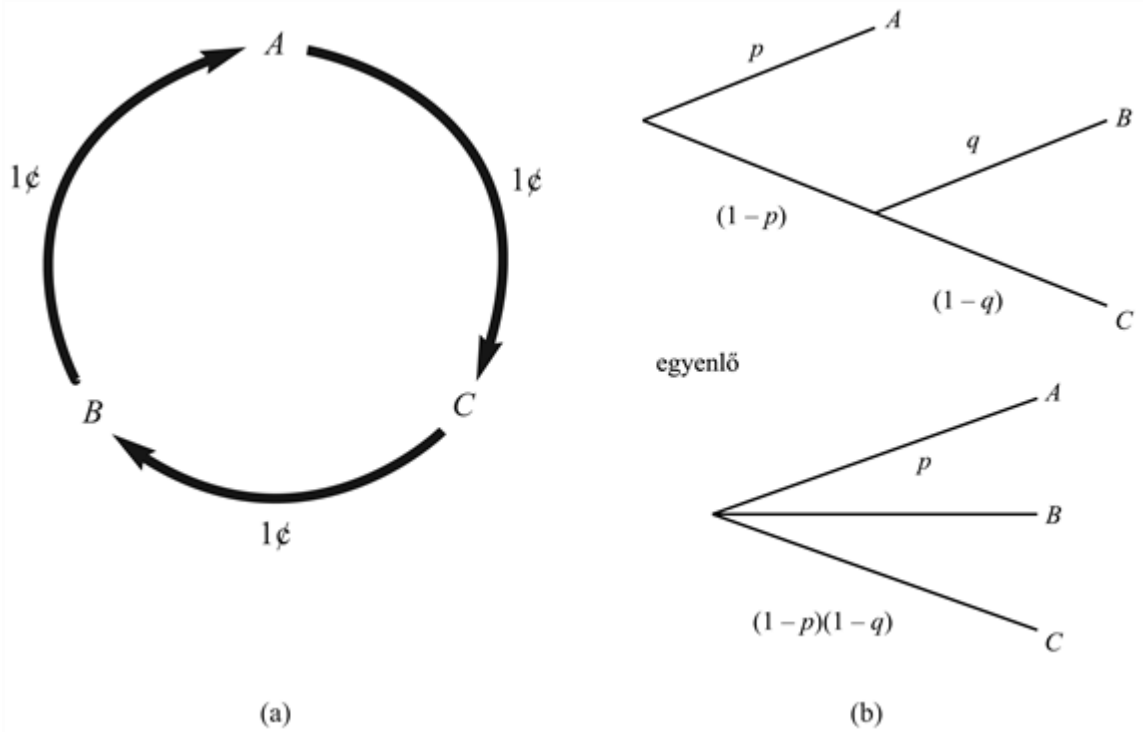
$A \succsim B$ *az ágens A-t preferálja B-hez képest, vagy egyformán preferálja A-t és B-t*

Created by XMLmind XSL-FO Converter.

Természetesen tisztázni kell, hogy mit is jelölhet *A* és *B*. Ha az ágens cselekvései determinisztikusak, akkor *A* és *B* tipikusan a cselekvések konkrét, teljesen specifikált eredményállapotai. Az általánosabb, nemdeterminisztikus esetben *A* és *B* **szerencsejáték**ok (**lotteries**). Egy szerencsejáték alapvetően egy valószínűségi eloszlás az aktuális következmények halmaza felett (melyek a szerencsejáték „díjai"). Az *L* szerencsejátékot a C_1, …, C_n következményekkel, amelyek p_1, …, p_n valószínűségekkel következhetnek be, a következőképpen jelöljük:

$$L = [p_1, C_1; p_2, C_2; \ldots p_n, C_n]$$

16.1. ábra - (a) A cserék egy teljes ciklusa, bemutatva, hogy a nemtranzitív $A \succ B \succ C \succ A$ preferenciák irracionális viselkedést eredményeznek. (b) A felbonthatósági axióma.



(Az egyetlen következménnyel rendelkező szerencsejáték *A*-val vagy [1, *A*]-val jelölhető.) Általában egy szerencsejáték bármely következménye lehet egy atomi állapot vagy egy másik szerencsejáték. Az elsődleges kérdés a hasznosságelmélet számára annak megértése, ahogyan az összetett szerencsejátékok közötti preferenciák viszonyulnak ezen szerencsejátékokat meghatározó állapotok közötti preferenciákhoz.

Ennek eléréséhez ésszerű megkötéseket írunk elő a preferenciarelációkra, hasonlóan ahhoz, ahogyan racionális kényszereket vártunk el a meggyőződés mértékeivel kapcsolatban a valószínűség axiómáinak a származtatásánál a 13. fejezetben. Egy ésszerű megkötés, hogy a preferenciák **tranzitív**ak (**transitive**) legyenek: azaz ha $A \succ B$ és $B \succ C$, akkor elvárhatjuk, hogy $A \succ C$. A tranzitivitás indoklásaként megmutatjuk, hogy egy ágens, amelynek a preferenciái nem tranzitívak, irracionálisan viselkedne. Tételezzük például fel, hogy egy ágens preferenciái nem tranzitívak: $A \succ B \succ C \succ A$, ahol *A*, *B* és *C* áruk, amik szabadon cserélhetők. Ha az ágensnél jelenleg *A* van, akkor felkínálhatjuk *C*-t cserébe *A*-ért és egy kis pénzért. Ha az ágens jobban szereti *C*-t, akkor hajlandó lesz pénzt is áldozni *C* megszerzésére. Ezután felkínálhatjuk *B*-t *C*-ért, még több pénzt begyűjtve, és végül elcseréljük *B*-t *A*-ért. Ezzel visszaértünk a kezdeti állapotba, azt leszámítva, hogy az ágensnek kevesebb a pénze (16.1. (a) ábra). Ezt a ciklust addig folytathatjuk, ameddig az ágens összes pénze el nem fogy. Indokoltnak látszik azt állítani, hogy az ágens ebben a helyzetben nem cselekedett racionálisan.

A következő hat megkötést a hasznosságelmélet axiómáinak szokás tekinteni. Ezek a preferenciákra és a szerencsejátékokra vonatkozó legnyilvánvalóbb szemantikai megkötéseket határozzák meg.

- **Sorrendezhetőség** (**orderability**). Bármely két állapot esetén az ágensnek vagy preferálnia kell az egyiket a másikkal szemben, vagy egyformán preferáltnak kell minősítenie mindkettőt. Azaz az ágens nem kerülheti el a döntést. Ahogyan az 3.1. szakasz - A valószínűségi axiómák használata részben tárgyaltuk, egy fogadás visszautasítása az idő folyásának a felfüggesztéséhez hasonló.

Created by XMLmind XSL-FO Converter.

$(A \succ B) \vee (B \succ A) \vee (A \succ B)$

- **Tranzitivitás** (**transitivity**). Bármely három állapot esetén, ha az ágens preferálja *A*-t *B*-vel szemben, és *B*-t *C*-vel szemben, akkor az ágensnek preferálnia kell *A*-t is *C*-vel szemben.

$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$

- **Folytonosság** (**continuity**). Ha valamely *B* állapot *A* és *C* között helyezkedik el a preferenciák szempontjából, akkor létezik egy *p* valószínűség, amely mellett a racionális ágens számára közömbössé válik, hogy a biztos *B*-t kapja, vagy egy olyan szerencsejátékot, amiben *p* valószínűséggel *A*, 1 – *p* valószínűséggel pedig *C* nyerhető.

$A \succ B \succ C \Rightarrow \exists p\ [p,\ A;\ 1 - p,\ C] \sim B$

- **Helyettesíthetőség** (**substitutability**). Ha egy ágens az *A* és *B* szerencsejátékot egyformán preferálja, akkor az ágens két összetettebb szerencsejátékot is egyformán preferál, amelyek csak abban különböznek, hogy az egyikben *A* *B*-re van cserélve. Ez a szerencsejátékokbeli valószínűségektől és más kimenetektől függetlenül igaz.

$A \sim B \Rightarrow [p,\ A;\ 1 - p,\ C] \sim [p,\ B;\ 1 - p,\ C]$

- **Monotonitás** (**monotonicity**). Tételezzük fel, hogy két szerencsejátéknak ugyanaz a két kimenetele van, *A* és *B*. Ha az ágens *A*-t preferálja *B*-vel szemben, akkor az ágensnek azt a szerencsejátékot kell preferálnia, ami nagyobb valószínűséggel eredményezi *A*-t (és fordítva is).

$A \succ B \Rightarrow (p \geq q \Leftrightarrow [p,\ A;\ 1 - p,\ B] \gtrsim [q,\ A;\ 1 - q,\ B]$

- **Felbonthatóság** (**Decomposability**): Összetett szerencsejátékok egyszerűbbekre bonthatók a valószínűség-számítás szabályai szerint. Ezt „a hazárdjáték unalmas” szabálynak nevezték el, mivel ez azt mondja ki, hogy két egymást követő szerencsejátékot össze lehet olvasztani egyetlen ekvivalens szerencsejátékba, ahogyan azt a 16.1. ábra mutatja.[1]

$[p,\ A;\ 1 - p,\ [q,\ B;\ 1 - q,\ C]] \sim [p,\ A;\ (1 - p)q,\ B;\ (1 - p)(1 - q),\ C]$

2.2. És aztán jött a hasznosság

Vegyük észre, hogy a hasznosságelmélet axiómái nem mondanak semmit a hasznosságról. Kizárólag a preferenciákról szólnak. Azaz a racionális ágens alapvető tulajdonságainak a preferenciákat tételezzük fel. A hasznosságfüggvény létezése a hasznosság- axiómákból *következik:*

1. A **hasznosság elv** (**utility principle**)

Ha az ágens preferenciái eleget tesznek a hasznosság axiómáinak, akkor létezik egy, az állapotokon értelmezett *U* valós értékű függvény, mellyel $U(A) > U(B)$ akkor és csak akkor, ha *A* preferált *B*-vel szemben, és $U(A) = U(B)$ akkor és csak akkor, ha az ágens számára *A* és *B* egyformán preferált.

$U(A) > U(B) \Leftrightarrow A \succ B$

$U(A) = U(B) \Leftrightarrow A \sim B$

2. A **maximális várható hasznosság elve** (**maximum Expected Utility principle**)

Egy szerencsejáték hasznossága az egyes kimenetelek valószínűségeivel szorzott kimenetelhasznosságok összege.

$$U([p_1, S_1; \ldots; p_n, S_n]) = \sum_i p_i U(S_i)$$

Más szóval, amint a lehetséges kimeneteli állapotoknak a valószínűségei és hasznosságai specifikáltak, az ezeket tartalmazó összetett szerencsejáték hasznossága teljesen meghatározott. Mivel egy nemdeterminisztikus cselekmény kimenetele egy szerencsejáték, ez egy MVH-alapú döntési szabályt jelent a (16.1) egyenlet szerint.

[1] A hazárdjáték élvezetét ekkor úgy modellezhetjük, hogy belefoglalhatjuk a hazárdjátékjátszás eseményeit az állapotleírásokba; például a „10 dollár és hazárdjátékjátszás” preferált lehet a „10 dollár hazárdjátékjátszás nélkül”-lel szemben.

Created by XMLmind XSL-FO Converter.

Fontos megjegyezni, hogy a hasznosságfüggvény léte, ami leírja az ágens preferenciáit, nem jelenti azt szükségszerűen, hogy az ágens *explicit módon* maximalizál egy hasznosságfüggvényt a mérlegelései során. Amint azt a 2. fejezetben megmutattuk, racionális viselkedés számos módon előállítható, némelyik sokkal hatékonyabb, mint a hasznosság maximalizálásának explicit elvégzése. Az ágens preferenciáinak megfigyelésével azonban lehetségessé válik a hasznosságfüggvény megkonstruálása, ami az ágens elérni szándékozott céljait reprezentálja.

3. Hasznosságfüggvények

A hasznosság egy függvény, ami valós számokat rendel az állapotokhoz. Vajon ez minden, amit a hasznosságfüggvényről el lehet mondani? Szigorúan nézve, igen, ennyi. A korábban felsorolt megkötéseknek eleget téve az ágensnek tetszőleges preferenciái lehetnek. Például az ágens preferálhatja, hogy a bankbetétjein elhelyezett dollárok száma prímszám legyen; ekkor, ha 16 dollárja volna, 3-at elajándékozna. Lehet, hogy egy 1973-as ütött-kopott Ford Pintót jobban szeret, mint egy csillogó új Mercedest. A preferenciák kapcsolatban állhatnak egymással; például lehet, hogy csak akkor preferál prímszámú dollárt, ha Pintója van, de amikor Mercedese van, akkor a több dollárt jobban szereti, mint a kevesebbet.

Ha az összes hasznosságfüggvény ilyen tetszőleges volna, mint ez, akkor viszont a hasznosságelméletnek nem lenne sok haszna, mivel meg kellene figyelnünk az ágens preferenciáit minden lehetséges körülmény esetén, mielőtt a viselkedésével kapcsolatban bármilyen előrejelzésre is képesek lennénk. Szerencsére az igazi ágensek preferenciái sokkal rendszerezettebbek. Ennek megfelelően szisztematikus módszerek léteznek a hasznosságfüggvények megtervezésére, amelyeket aztán egy mesterséges ágensbe beépítve az ágens a helyes, általunk elvárt viselkedést fogja produkálni.

3.1. A pénz hasznossága

A hasznosságelmélet a közgazdaságtanból származik, a közgazdaságtan pedig egy nyilvánvaló jelöltet kínál a hasznosság mérésére: a pénzt (vagy pontosabban az ágens teljes nettó vagyonát). A pénz majdnem univerzális felcserélhetősége bármely árura és szolgáltatásra azt mutatja, hogy a pénz jelentős szerepet játszik az emberi hasznosságfüggvényekben. (Valójában, a legtöbb ember a közgazdaságtant a pénz tanulmányozásához köti, pedig a *közgazdaság* szó gyökere a gazdálkodásra vonatkozik, míg jelenleg a hangsúly a választások kezelésén van.)

Ha csak azokat a cselekvéseket vesszük figyelembe, amelyek az ágens pénzmennyiségét befolyásolják, akkor általában az lesz a tapasztalat, hogy az ágens a több pénzt előnyben részesíti a kevesebbel szemben, ha egyéb dolgok egyenlők. Azt mondjuk, hogy az ágens **monoton preferenciá**t **(monotonic preference)** mutat egy adott összegű pénz esetén. Azonban ez nem elegendő annak garantálására, hogy a pénzt egy hasznosságfüggvénynek tekinthessük, mivel ez semmit sem mond azon *szerencsejátékok* közötti preferenciákról, amelyek kimenetele pénz.

Tételezzük fel, hogy ön győzedelmeskedett a versenytársak felett egy televíziós játékban. A házigazda most választásra kéri fel: elviheti az 1 000 000 dolláros díjat, vagy felteheti egy pénzfeldobásos hazárdjátékon. Ha fej, nem kap semmit, ha írás, akkor kap 3 000 000 dollárt. Ha hasonló a többi emberhez, akkor vonakodna játszani, és zsebre vágná a milliót. Ez irracionális volna?

Feltéve, hogy hisz az érme szabályos voltában, a játék várható pénzügyi értéke (VPÉ) (expected monetary value) $\frac{1}{2}(0 \text{ dollár}) + \frac{1}{2}(3\,000\,000 \text{ dollár}) = 1\,500\,000 \text{ dollár}$, és az eredeti díj, a VPÉ-je természetesen 1 000 000 dollár, ami kisebb. De ez nem jelenti szükségszerűen, hogy a hazárdjáték elfogadása a jobb döntés. Tételezzük fel, hogy S_n jelöli az n dollárt birtokló állapotot, és a jelenlegi vagyon k dollár. Ekkor a két cselekedetnek, a hazárdjáték elfogadásának vagy visszautasításának a várható hasznossága:

$$EU(Elfogad) = \frac{1}{2}U(S_k) + \frac{1}{2}U(S_{k+3\,000\,000})$$
$$EU(Elutasit) = U(S_{k+1\,000\,000})$$

Annak eldöntéséhez, hogy mit is tegyünk, az egyes kimeneteli állapotokhoz hasznosságot kell rendelnünk. A hasznosság nem közvetlenül arányos a pénzügyi értékkel, mivel a hasznosság – az okozott pozitív változás az életstílusban – az első millió dollár esetén nagyon nagy (vagy legalábbis azt mondják), ezzel szemben a további

Created by XMLmind XSL-FO Converter.

milliók hasznossága sokkal kisebb. Tételezzük fel, hogy 5-ös hasznosságot rendel a jelenlegi (S_k) pénzügyi helyzethez, 10-et az $S_{k+3\,000\,000}$ állapothoz, 8-at az $S_{k+1\,500\,000}$ állapothoz. Ekkor a racionális döntés az elutasítás lenne, mivel az elfogadás várható hasznossága 7,5 (kisebb, mint az elutasításhoz tartozó 8). Most tételezze fel, hogy már van a számláján 500 000 000 dollár (és a játékban való részvétel csak a móka kedvéért történik). Ebben az esetben a hazárdjáték valószínűleg elfogadható, feltéve, hogy a több pénzt preferálja a kevesebbel szemben, mivel az 503-adik millió haszna valószínűleg ugyanakkora, mint az 501-edik millióé.

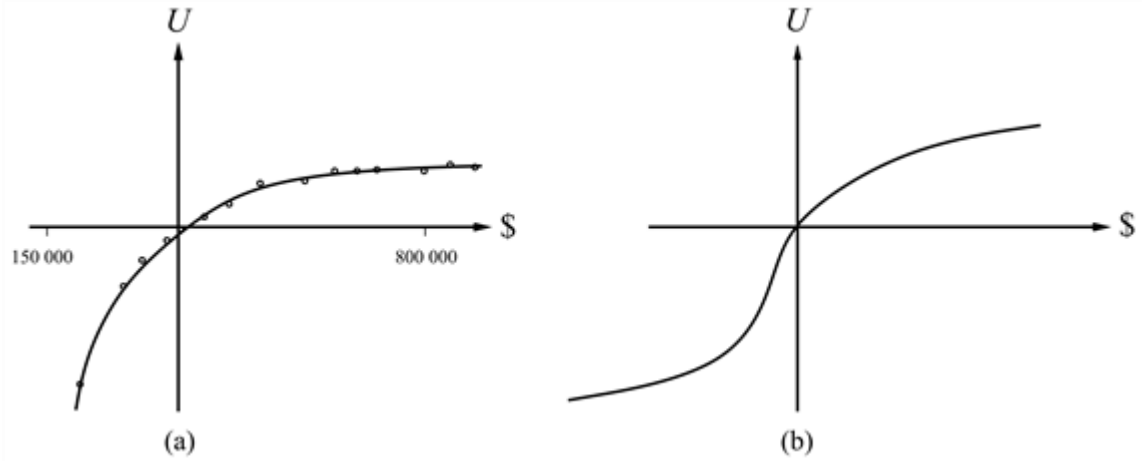
A valóságos hasznossági függvényekről szóló úttörő jelentőségű tanulmányban Grayson azt találta, hogy a pénz hasznossága majdnem teljesen arányos a mennyiségének *logaritmusával* (Grayson, 1960). (Ezt először Bernoulli vetette fel [Bernoulli, 1783]; lásd 16.3. feladat.) Egy ilyen görbe, egy bizonyos Beard úré, a 16.2. (a) ábrán látható. A Beard úrtól kapott preferenciák konzisztensek a következő hasznossági függvénnyel:

$$U(S_{k,n}) = -263{,}31 + 22{,}09 \log(n + 150\ 000)$$

az $n = -150\,000$ dollár és $n = 800\,000$ dollár közötti tartományban.

Azt nem tételezhetjük fel, hogy ez egy univerzális hasznosságfüggvénye a vagyoni értéknek, de valószínű, hogy a legtöbb embernek olyan hasznosságfüggvénye van, ami konkáv a pozitív tartományban. Adósságba keveredni általában vészesnek tartott dolog, de a preferenciák a különböző adósságszintek között a pozitív tartománybeli konkávitás megfordulását is mutathatják. Például ha valakinek már van 10 000 000 dollár adósága, akkor lehet, hogy igencsak részt vesz egy olyan pénzfeldobásos hazárdjátékban, ahol 10 000 000 dollár a nyeremény, ha fej, és 20 000 000 dollár a veszteség, ha írás.[2] Ez egy S alakú görbét eredményez, amit a 16.2. (b) ábra mutat.

16.2. ábra - A pénz hasznossága. (a) Empirikus adat Beard úrtól egy korlátos tartományban. (b) Egy tipikus görbe a teljes tartományban.



Ha csak a görbe pozitív részét vizsgáljuk, ahol a meredekség csökken, akkor itt bármely L szerencsejáték esetén a várható hasznosság kisebb, mint annak a biztos eseménynek a hasznossága, amikor az átadott pénzügyi érték a várható pénzügyi értékkel egyezik meg.

$$U(L) < U(S_{VPÉ(L)})$$

Azaz az ilyen alakú görbével rendelkező ágens **kockázatkerülő** (**risk-averse**): előnyben részesít egy biztos eseményt egy szerencsejátékkal szemben, még akkor is, ha a biztos összeg kisebb, mint a szerencsejáték várható pénzügyi értéke. Másfelől, a 16.2. (b) ábrán a nagy adóságokhoz tartozó „elkeseredett” régióban a viselkedés **kockázatkereső** (**risk-seeking**). Azt az értéket, amit az ágens a szerencsejáték helyett elfogad, a játék **determinisztikus ekvivalens**ének (**certainty equivalent**) nevezzük. A tanulmányok azt mutatják, hogy a legtöbb ember elfogad körülbelül 400 dollárt egy olyan szerencsejátékért cserébe, ami $\frac{1}{2}$ valószínűséggel eredményez 1000 dollárt és 0 dollárt – azaz ennek a szerencsejátéknak a determinisztikus ekvivalense 400 dollár. A szerencsejáték várható pénzügyi értéke és a determinisztikus ekvivalense közötti értéket **biztosítási prémium**nak (**insurance premium**) hívjuk. A kockázatkerülés az alapja a biztosítási üzletágnak, mivel ez azt jelenti, hogy a biztosítási prémium pozitív. Az emberek inkább fizetnek egy alacsonyabb biztosítási díjat,

[2] Az ilyen viselkedést elkeseredettnek nevezhetjük, mindazonáltal teljesen racionális, ha valaki kilátástalan helyzetben van.

Created by XMLmind XSL-FO Converter.

mintsem hogy kockára tegyék a házuk teljes árát egy lehetséges tűzeset miatt. Viszont a biztosítási társaságok szemszögéből a ház teljes ára igen kicsi a cég összes vagyonához képest. Emiatt a biztosító hasznossági görbéje megközelítően lineáris ebben a kicsiny régióban, és a biztosítás a társaságnak alig kerül valamibe.

Vegyük észre, hogy a *kis* vagyoni változások az aktuális vagyoni helyzethez képest közel lineárisak a görbe bármely szakaszán. A lineáris görbével rendelkező ágenst **kockázatsemlegesnek** (**risk-neutral**) nevezzük. Ezért kis összegű hazárdjátékok esetén kockázatsemlegességet várhatunk el. Egy bizonyos értelemben ez igazolja azt az egyszerűsített eljárást, ami kis összegű szerencsejátékokkal segítette valószínűségek megbecsülését, és igazolta a valószínűség-számítás axiómáit a 13. fejezetben.

3.2. Hasznosságskálák és a hasznosság megbecslése

A hasznosságaxiómák, ha az ágens viselkedési preferenciái rögzítettek, nem határoznak meg egy kitüntetett hasznosságfüggvényt. Például az $U(S)$ hasznosságfüggvényt a következő alakra transzformálhatjuk:

$$U'(S) = k_1 + k_2\, U(S)$$

ahol k_1 egy állandó és k_2 egy tetszőleges pozitív állandó. Nyilvánvaló, hogy ez a lineáris transzformáció nem befolyásolja az ágens viselkedését.

Determinisztikus környezetekben, ahol állapotok vannak, és nincsenek szerencsejátékok, a viselkedés nem fog megváltozni semmilyen *monoton* transzformációra. Például vehetjük a köbét az összes hasznosságnak anélkül, hogy ez a cselekmények preferenciarendezését befolyásolná. Egy determinisztikus környezetben lévő ágens esetén azt mondjuk, hogy az ágensnek **értékfüggvénye** (**value function**) vagy **sorrendezett hasznosságfüggvénye** (**ordinal utility function**) van; a függvény valójában csak az állapotok sorrendezését biztosítja, és nem ad értelmezhető numerikus értékeket. Ezt a különbségtételt láttuk a 6. fejezetben a játékok esetén: determinisztikus játékokban, mint a sakk, a kiértékelő függvények értékfüggvények, míg nemdeterminisztikus játékokban, mint az ostábla, ezek valódi hasznosságfüggvények.

Fontos

Emberi ítélethozatal és a hibázás lehetősége

A döntéselmélet egy **normatív elmélet** (**normative theory**): azt írja elő, hogy az ágensnek hogyan *kell* cselekednie. A közgazdaságtan elméletének alkalmazását nagyban javítaná, ha ez egyben egy **leíró** (**descriptive**) elmélet is volna az aktuális emberi döntéshozatalról. Azonban kísérleti bizonyítékok vannak arra, hogy az emberek szisztematikusan megsértik a hasznosságelmélet axiómáit. Egy példát a pszichológus Tversky és Kahneman adnak a közgazdász Allais példája alapján (Tversky és Kahneman, 1982; Allais, 1953). A kísérleti személyeknek kétszer kell választaniuk, az *A* és a *B* szerencsejáték, majd a *C* és a *D* szerencsejáték között.

A: 4000 dollár 80% eséllyel *C*: 4000 dollár 20% eséllyel

B: 3000 dollár 100% eséllyel *D*: 3000 dollár 25% eséllyel

A többség *B*-t választja *A*-val szemben, és *C*-t *D*-vel szemben. De ha elfogadjuk, hogy $U(0 \text{ dollár}) = 0$, akkor az első választás arra utal, hogy $0{,}8U(4000 \text{ dollár}) < U(3000 \text{ dollár})$, míg a második választás pontosan a fordítottjára. Másképpen fogalmazva, nincs olyan hasznosságfüggvény, ami ezekkel a választásokkal konzisztens lenne. Az egyik lehetséges következtetés, hogy az emberek egyszerűen irracionálisak a hasznosságelmélet axiómái alapján. Egy alternatív nézet, hogy az elemzés nem veszi figyelembe a **megbánást** (**regret**) – azt az érzést, amiről az emberek tudják, hogy érezni fognak, ha egy biztos nyeremény (*B*) helyett egy 80%-os valószínűségű nagyobb nyereményt választanak, majd vesztenek. Más szavakkal, ha *A*-t választja, 20% esélye van, hogy nem lesz pénze, és úgy érzi magát, mint egy komplett idióta.

Kahneman és Tversky továbbment, és kifejlesztett egy leíró elméletet, ami megmagyarázza, hogy az emberek miért kerülik a kockázatot nagy valószínűségű eseményeknél, viszont miért hajlandók nagyobb kockázatot vállalni valószínűtlen jutalmakkal. A kapcsolat az MI és ezen tapasztalat között az, hogy az ágensünk döntései csak annyira jók, amennyire jók a preferenciák, amin alapulnak. Ha az emberi informátoraink ellentmondásos preferenciákhoz ragaszkodnak, akkor az ágensünk semmit sem tud tenni, hogy konzisztens legyen velük.

Created by XMLmind XSL-FO Converter.

> Szerencsére, az emberek által hozott preferenciaítéletek gyakran könnyen felülbírálhatók későbbi megfontolások alapján. Keeney és Raiffa egy korai munkájukban a Harvard Business Schoolon a pénz hasznosságának a megbecslése kapcsán a következőt írta (Keeney és Raiffa, 1976, 210. o):
>
> Kísérleti vizsgálatok nagy része mutatta ki, hogy súlyos hiányosságok vannak a becslési protokollban. A kísérleti személyek kicsiben hajlamosak túlságosan kockázatkerülővé válni és ezért… az illesztett hasznosságfüggvények elfogadhatatlanul nagy kockázati prémiumot mutatnak a nagy tartományú szerencsejátékokra. …A kísérleti személyek azonban rendezni tudják az inkonzisztenciájukat, és úgy érzik, hogy valami fontosat tanultak arról, hogyan akarnak viselkedni. Eredményképpen néhányan lemondták a gépjármű baleseti biztosításukat és több időszakos életbiztosítást kötöttek meg.
>
> Az emberi (ir)racionalitás még ma is intenzív vizsgálatok tárgya.

A hasznosságok megbecsülésére egy megoldás, hogy választunk egy skálát a „lehető legnagyobb díj” $U(S) = u_\top$ és a „lehető legnagyobb katasztrófa” $U(S) = u_\perp$ alapján. **Normált hasznosság**ok (**normalized utilities**) esetén a skála $u_\perp = 0$ és $u_\top = 1$. A közbeeső kimenetelek hasznosságának a megbecslése úgy történik, hogy az ágenst megkérik, jelezze a preferenciáját az S kimeneteli állapot és egy $[p, u_\top; (1 - p), u_\perp]$ **standard szerencsejáték** (**standard lottery**) között. A p valószínűséget addig módosítják, amíg az ágens nem preferálja egyformán S-et és a standard szerencsejátékot. Normalizált hasznosságokat feltételezve S hasznossága p.

Fontos

> Orvosi, szállítási és környezetvédelmi problémáknál, egyebek között, emberek élete forog kockán. Ezekben az esetekben $u_\perp$ a közvetlen halálesethez (esetleg halálesetekhez) hozzárendelt érték. *Ámbár senkinek sem könnyű értékeket rendelni emberi életekhez, az tény, hogy szünet nélkül kompromisszumokat kötünk.* A repülőgépeket csak adott időközönként vetik alá ellenőrzéseknek, amit az utazások és megtett kilométerek határoznak meg, nem pedig minden utazás után. A költségcsökkentések miatt a gépkocsik karoszszériája viszonylag vékony fémből készül, a balesetek túlélési arányának csökkenése ellenére. Az ólmozott üzemanyag még mindig igen elterjedt, ámbár jól ismert, hogy az egészségre káros. Így paradox módon az az ellenkezés, hogy „az élet pénzben nem kifejezhető”, azt eredményezi, hogy az élet *alulértékelt*. Ross Shachter egy kormányhivatallal kapcsolatos élményéről számol be, amely hivatal megrendelt egy tanulmányt az azbeszt iskolákból történő eltávolításáról. A tanulmány egy adott pénzösszeget rendelt az iskoláskorú gyerekek életéhez, és úgy érvelt, hogy ezen feltevés mellett a racionális döntés az azbeszt eltávolítása. A kormányhivatal, morálisan felháborodva teljes mértékben elutasította a jelentést. Ezzel egyben az azbeszt eltávolítása ellen is döntöttek.

Történtek próbálkozások arra nézve, hogy kiderítsék, mennyi értéket tulajdonítanak az emberek az életüknek. Az orvosi és biztonsági elemzésekben két elterjedt „pénznemet” használnak, a **mikrohalál**t (**micromort**) (az egymilliomod esélyét az elhalálozásnak) és a **QALY**-t, vagy szabványos minőségű életévet (ami ekvivalens egy évvel, jó vagyoni helyzetben, betegségektől mentesen). Számos, személyek igen széles körét vizsgáló tanulmány azt mutatta, hogy a mikrohalál 20 dollárt ér (1980-ban). Azt már láttuk, hogy a hasznosságfüggvény nemlineáris, ezért ez nem jelenti azt, hogy a döntéshozó megölné magát 20 millió dollárért. Újra hangsúlyozandó, hogy a hasznossággörbe lokális linearitása azt jelenti, hogy a mikrohalál és a QALY értékei kis relatív kockázatok és jutalmak esetén hasznosak.

4. Többváltozós hasznosságfüggvények

A közérdekű ügyeknél történő döntéshozatal mind dollármilliókhoz, mind emberek életéhez és halálához is kapcsolódik. Például annak eldöntésekor, hogy rákkeltő anyagok milyen koncentrációi engedhetők meg a környezetben, a döntéshozóknak mérlegelniük kell a halál megelőzését és azokat a közgazdasági nehézségeket, amelyek bizonyos termékek és folyamatok megszüntetése miatt jelentkezhetnek. Egy új repülőtér elhelyezése esetén meg kell fontolni az építkezés okozta felfordulást; a telkek értékét; a népesebb központoktól való távolságot; a reptér zaját; a biztonsági kérdéseket a helyi földrajzi és az időjárási viszonyok tekintetében és így tovább. Az ehhez hasonló problémákat, ahol a kimenetelt két vagy több attribútum jellemez a **többattribútumú hasznosságelmélet** (**multiattribute utility theory**) kezeli.

Jelölje $\mathbf{X} = X_1, X_2, \ldots, X_n$ az attribútumokat, az attribútumértékek vektorát pedig $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle$. Általában minden attribútumról feltételezzük, hogy diszkrét vagy folytonos értékekkel rendelkezik. Az egyszerűség kedvéért feltételezzük, hogy az attribútumokat úgy határoztuk meg, hogy a nagyobb attribútumértékek nagyobb

Created by XMLmind XSL-FO Converter.

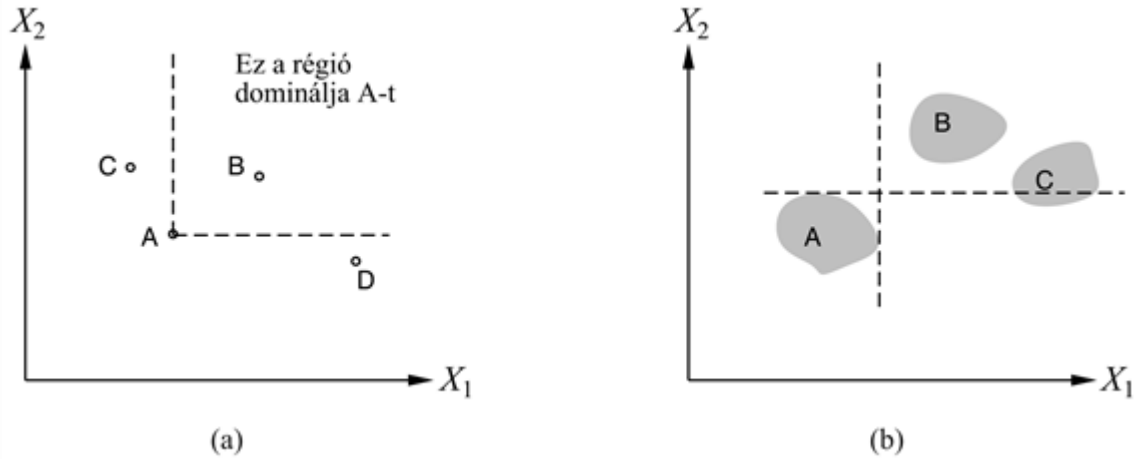
hasznosságértékekhez tartoznak, ha minden más változatlan. Például a repülőtér probléma esetén minél nagyobb a *ZajMentesség* értéke, annál jobb a megoldás. Bizonyos esetekben szükséges lehet az értéktartomány felosztása, ahol a hasznosságértékek az egyes tartományokban monoton módon változnak.

Elsőként azokat az eseteket vizsgáljuk meg, amikor döntéseket anélkül hozhatunk, hogy az attribútumértékeket egyetlen hasznosságértékké egyesítenénk. Majd azokat az eseteket vizsgáljuk, ahol az egyes attribútumkombinációk hasznosságát nagyon tömören lehet megadni.

4.1. Dominancia

Tételezzük fel, hogy az S_1 reptér kevesebbe kerül, kisebb zajjal jár és biztonságosabb is, mint S_2. Ekkor az mondjuk, hogy S_1 **szigorúan dominál**ja (**strict dominance**) S_2-t. Általában, ha egy lehetőség minden attribútumának kisebb az értéke, mint egy másik lehetőségé, akkor nem szükséges tovább vizsgálódni. A szigorú dominancia gyakran nagyon hasznos, mert leszűkíti a választási lehetőségeket a valódi jelöltekre, bár ritkán eredményez egyetlen választási lehetőséget. A 16.3. (a) ábra egy sematikus helyzetet mutat egy kétattribútumos esetben.

16.3. ábra - Szigorú dominancia. (a) Determinisztikus: A-t szigorúan dominálja B, de sem C, sem D nem. (b) Bizonytalan: A-t szigorúan dominálja B, de C nem.



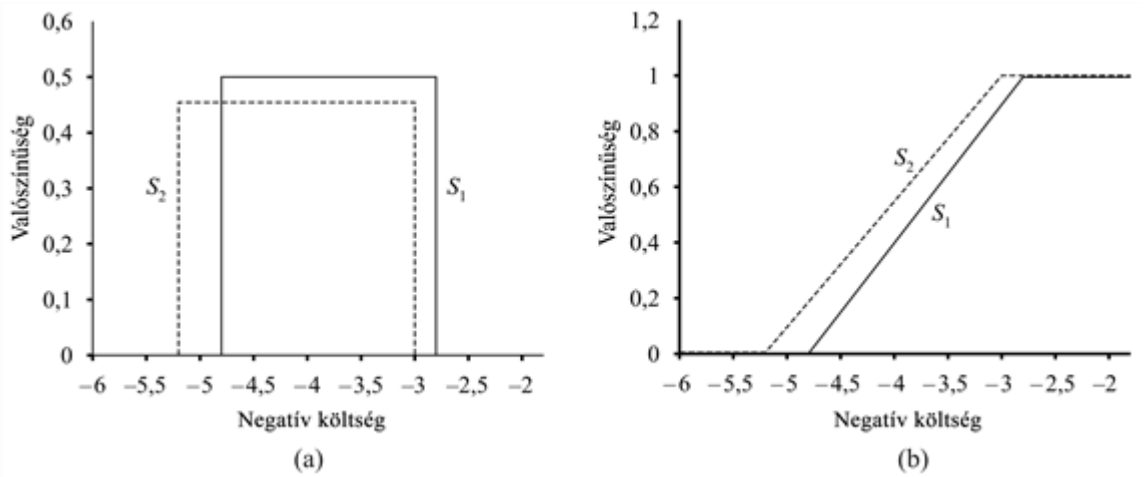
Mindez remekül használható determinisztikus esetekben, ahol a tulajdonságértékek biztosan ismertek. De mi történik abban az általános esetben, ahol a cselekvések kimenetele bizonytalan? A szigorú dominancia közvetlen analógiája könnyen megalkotható, ha a bizonytalanság ellenére, S_1 minden lehetséges kimenetele szigorúan dominálja S_2 minden lehetséges kimenetelét. (Ezt mutatja a 16.3. (b) ábra sematikus ábrázolása.) Természetesen ez valószínűleg sokkal ritkábban fordul elő, mint a determinisztikus esetben.

Szerencsére létezik egy használhatóbb általánosítás, az úgynevezett **sztochasztikus dominancia** (**stochastic dominance**), ami valós problémáknál is gyakran előfordul. A sztochasztikus dominanciát könnyebb megérteni egyetlen változó esetében. Tételezzük fel, hogy a reptér költsége az S_1 helyen egyenletes eloszlású 2,8 és 4,8 milliárd dollár között, és az S_2 helyszínen a költség egyenletes eloszlású 3,0 és 5,2 milliárd dollár között. A 16.4. (a) ábra ezen költségek eloszlásait mutatja, ahol a költségek negatív értékként szerepelnek. Ekkor csupán azt az információt ismerve, hogy a hasznosság csökken a költségekkel, azt mondhatjuk, hogy S_1 sztochasztikusan dominálja S_2-t – azaz S_2 elhagyható. Fontos felismerni, hogy ez *nem* következik a várható költségek összehasonlításából. Például ha tudnánk, hogy S_1 költsége *pontosan* 3,8 milliárd dollár, akkor a pénz hasznosságára vonatkozó további információk nélkül *nem tudnánk* döntést hozni.[3]

16.4. ábra - Sztochasztikus dominancia. (a) S_1 sztochasztikusan dominálja S_2-t a költségek vonatkozásában. (b) S_1 és S_2 negatív költségeinek eloszlásfüggvényei.

[3] Furcsának tűnhet, hogy az S_1 költségéről lévő *több* információ az ágens döntésképességét *csökkenti*. A paradoxonra magyarázatot kapunk, ha figyelembe vesszük, hogy az egzakt költséginformáció hiányában meghozott döntés kisebb valószínűséggel a legnagyobb hasznosságú.

Created by XMLmind XSL-FO Converter.



Az attribútumok eloszlásai közötti pontos kapcsolat, amely a sztochasztikus dominancia megállapításához szükséges, legkönnyebben az *eloszlásfüggvény* megvizsgálásával látható, ahogy azt a 16.3. (b) ábra is illusztrálja. Az eloszlásfüggvény annak a valószínűségét méri, hogy a költség kisebb-e egy adott összegnél, vagy egyenlő vele – azaz az eloszlásfüggvény az eredeti eloszlást integrálja. Ha S_1 eloszlásfüggvénye mindig jobbra esik S_2 eloszlásfüggvényétől, akkor sztochasztikus szemszögből S_1 olcsóbb, mint S_2. Formálisan, ha az A_1 és az A_2 cselekvések a $p_1(x)$ és a $p_2(x)$ valószínűség-eloszlásokra vezetnek az X attribútumon értelmezve, akkor A_1 sztochasztikusan dominálja A_2-t az X-en, ha

$$\forall x \int_{-\infty}^{x} p_1(x')dx' \leq \int_{-\infty}^{x} p_2(x')dx'$$

Fontos

A definíció fontosságát az optimális döntés megválasztásában a következő tulajdonság adja: *ha A_1 sztochasztikusan dominálja A_2-t, akkor bármely $U(x)$ monoton nem csökkenő hasznosságfüggvény esetén, A_1 várható hasznossága legalább olyan nagy, mint A_2 várható hasznossága.* Azaz, ha egy cselekvést egy másik cselekvés minden attribútum esetén sztochasztikusan dominál, akkor az első cselekvés figyelmen kívül hagyható.

Úgy tűnhet, hogy a sztochasztikus dominancia meglehetősen matematikai megközelítést igényel, meghatározása pedig nagy mennyiségű, valószínűséggel kapcsolatos számítást. Valójában azonban számos esetben igen könnyen eldönthető. Tételezzük fel például, hogy az építési költség a népesebb központoktól való távolságtól függ. A költség maga bizonytalan, de a nagyobb távolság nagyobb költséget jelent. Ha S_1 közelebbi, mint S_2, akkor S_1 dominálni fogja S_2-t a költség tekintetében. Bár ismertetésüktől eltekintünk, de léteznek algoritmusok, amelyek képesek kezelni az ilyen kvalitatív információkat is a bizonytalan változók közötti kapcsolatokról a **kvalitatív valószínűségi hálók** (**qualitative probabilistic networks**) felhasználásával, lehetővé téve, hogy egy rendszer a sztochasztikus dominancia alapján hozzon racionális döntéseket anélkül, hogy bármilyen numerikus értéket használna.

4.2. A preferenciák rendszere és a többattribútumos hasznosság

Tételezzük fel, hogy n attribútumunk van, ahol mindegyiknek d különböző értéke lehet. A teljes $U(x_1, \ldots, x_n)$ hasznosságfüggvény megadásához legrosszabb esetben d^n érték szükséges. Ez a legrosszabb eset abban a helyzetben áll fent, amikor az ágens preferenciáiban nincs semmilyen szabályszerűség. A többattribútumos hasznosságelmélet azon a feltételezésen alapul, hogy a legtöbb hasznosságfüggvény sokkal nagyobb strukturáltsággal rendelkezik. Az elfogadott alapvető megközelítés az, hogy a viselkedés preferenciáiban várható szabályszerűségeket azonosítunk, és az úgynevezett **reprezentációs tétel**eket (**representation theorem**s) felhasználva megmutatjuk, hogy az adott preferenciarendszerrel rendelkező ágens leírható a következő hasznosságfüggvénnyel:

$$U(x_1, \ldots, x_n) = f[f_1(x_1), \ldots, f_n(x_n)]$$

Created by XMLmind XSL-FO Converter.

ahol *f* remélhetőleg olyan egyszerű függvény, mint az összeadás. Látható, hogy ez ahhoz hasonló, ahogy a valószínűségi hálókat használtuk az együttes valószínűségeloszlás-függvény felbontására.

Preferenciák bizonytalanság nélkül

Kezdjük a determinisztikus esettel. Emlékezzünk vissza, hogy determinisztikus környezetben az ágensnek egy értékfüggvénye van, : a cél ennek a tömör reprezentálása. A determinisztikus preferenciastruktúrában megtalálható alapvető szabályszerűséget **preferenciafüggetlenség**nek (**preference independence**) nevezzük. Két attribútum, X_1 és X_2 preferenciálisan független X_3-tól, ha a preferencia $\langle x_1, x_2, x_3 \rangle$ és $\langle x_1', x_2', x_3 \rangle$ között nem függ X_3 konkrét értékétől.

Visszatérve a repülőteres problémára, ha egyebek mellett a *Zaj*, a *Költség* és a *Halálesetek* figyelembe veendő attribútumok voltak, felvethető, hogy a *Zaj* és a *Költség* preferenciálisan független a *Halálesetek*-től. Ekkor például, ha előnyben részesítjük azt az állapotot, ahol 20 000 ember lakik a repülési útvonal közelében, és az építési költség 4 milliárd dollár, azzal az állapottal szemben, ahol 70 000 ember lakik a repülési útvonalon, és az építési költség 3,7 milliárd dollár, ha a biztonsági szint mindkét esetben 0,06 haláleset millió utas kilométerenként, akkor a preferenciánk a biztonsági szint 0,13 vagy 0,01 értéke esetében is ugyanaz lenne; és ugyanaz a függetlenség állna fenn bármely más *Zaj*, *Költség* értékpárra. Az is nyilvánvaló, hogy a *Költség* és a *Halálesetek* preferenciálisan függetlenek a *Zaj*-tól, és a *Zaj* és a *Halálesetek* preferenciálisan függetlenek a *Költség*-től. Ekkor azt mondjuk, hogy a {*Zaj*, *Költség*, *Halálesetek*} **kölcsönösen preferenciálisan független**ek (**KPF**) (**mutual preferential independence**, **MPI**). A KPF azt állítja, hogy bár lehet, hogy mindegyik attribútum fontos, ez nem befolyásolja a többi attribútum közötti kompromisszumkötést.

A kölcsönös preferenciális függetlenség olyasvalami, mint egy hosszúra sikeredett szókapcsolat, de a közgazdász Debreu egy figyelemre méltó tételének köszönhetően ez alapján az ágens értékfüggvényét nagyon egyszerű formában felírhatjuk (Debreu, 1960):

Fontos

Ha az $X_1, \ldots, X_n$ tulajdonságok kölcsönösen preferenciálisan függetlenek, akkor az ágens viselkedési preferenciája leírható a következő függvény maximalizálásával:

$$V(x_1, \ldots, x_n) = \sum_i V_i(x_i)$$

ahol minden egyes V_i egy értékfüggvény, amely csak az X_i attribútumnak a függvénye. Például igen valószínű, hogy a repülőtéri döntés meghozható a következő értékfüggvény alapján:

$$V(Zaj, Költség, Halálesetek) = -Zaj \times 10^4 - Költség - Halálesetek \times 10^{12}$$

Az ilyen típusú értékfüggvényt **additív értékfüggvény**nek (**additive value function**) nevezzük. Az additív értékfüggvények egy teljesen illeszkedő módszert adnak az ágens értékfüggvényének a leírására, és számos valósvilág-beli helyzetben érvényesek. Még ha a KPF nem is teljesül teljes mértékben, ahogy ez az attribútumok szélső értékei esetén megtörténhet, egy additív értékfüggvény még mindig jó közelítését adhatja az ágens preferenciáinak. Ez különösen igaz, ha a KPF csupán azon attribútumtartományokban sérül, amelyek a gyakorlatban csak kis valószínűséggel fordulnak elő.

Preferenciák bizonytalansággal

Ha a tárgytartományban bizonytalanság is jelen van, a szerencsejátékok közötti preferenciákat is át kell gondolni, és a hasznosságfüggvény kiadódó tulajdonságait is meg kell érteni, nem csak az értékfüggvényekét. Ennek a problémának a matematikája igen bonyolulttá válik, ezért a főbb eredmények közül csak egyet mutatunk be, hogy érzékeltessük az elérhető eredményeket. Az olvasónak a (Keeney és Raiffa, 1976) irodalmat ajánljuk, ami a terület igen alapos áttekintése.

A **hasznosságfüggetlenség** (**utility independence**) alapvető fogalma a preferenciák függetlenségét terjeszti ki szerencsejátékokra: az attribútumok **X** halmaza hasznosságfüggetlen az attribútumok **Y** halmazától, ha az **X** attribútumokon alapuló szerencsejátékok közötti preferenciák függetlenek az **Y**-beli attribútumokhoz rendelt értékektől. Egy tulajdonsághalmaz **kölcsönösen hasznosságfüggetlen** (**KHF**) (**mutually utility-independent**, **MUI**), ha minden részhalmaz hasznosságfüggetlen a többi attribútumtól. Ismételten értelmesnek tűnik feltételezni, hogy a repülőtéri építkezés attribútumai KHF-ek.

Created by XMLmind XSL-FO Converter.

A KHF-ből következik, hogy az ágens viselkedése leírható egy **multiplikatív hasznosságfüggvénnyel** (**multiplicative utility function**) (Keeney, 1974). A szorzatalakú hasznosságfüggvény általános alakját egy példán mutatjuk be három attribútum esetén. Az egyszerűség kedvéért, az $U_i(X_i)$ jelölésére az U_i-t használjuk:

$$U = k_1U_1 + k_2U_2 + k_3U_3 + k_1k_2U_1U_2 + k_2k_3U_2U_3 + k_3k_1U_3U_1 + k_1k_2k_3U_1U_2U_3$$

Bár ez nem tűnik egyszerűnek, három, egyenként egyetlen attribútumhoz kapcsolódó hasznosságfüggvényt tartalmaz, és csupán három állandót. Általában az n attribútumot tartalmazó KHF-et kielégítő problémát n számú, egyetlen attribútumú hasznossággal és n számú állandóval lehet modellezni. Minden egyes, egyetlen attribútumot leíró hasznosságfüggvényt a többi attribútumtól függetlenül lehet megtervezni, és ez a kombináció garantáltan előállítja a teljes preferenciarendszert. Hogy tisztán additív függvényt kaphassunk, további feltételek szükségesek.

5. Döntési hálók

Ebben az alfejezetben egy általános mechanizmust fogunk megvizsgálni a racionális döntések meghozatalára. A felhasznált eszközt gyakran **hatásdiagram**nak (**influence diagram**) hívják (Howard és Matheson, 1984), de mi a többet sugalló **döntési hálók** (**decision networks**) elnevezést fogjuk használni. A döntési hálók a cselekvésekhez és a hasznosságokhoz szükséges újabb csomóponttípusok hozzáadásával továbbfejlesztett valószínűségi hálók. A repülőtéri elhelyezési problémát fogjuk használni példának.

5.1. Döntési problémák reprezentálása döntési hálókkal

Egy döntési háló a legáltalánosabb formájában az ágens jelenlegi állapotáról tartalmaz információkat, megadja az ágens lehetséges cselekvéseit, azt az állapotot, amit az ágens cselekvése eredményez, és ennek az állapotnak a hasznosságát. Tehát alapot szolgáltat az olyan hasznosságelvű ágensek megépítéséhez, amilyeneket a 2.4. alfejezetben vezettünk be először. A 16.5. ábra egy döntési hálót mutat a repülőtér helyszíne megválasztásának problémájához. Ez bemutatja a három felhasználásra kerülő csomóponttípust:

- A **véletlen csomópont**ok (**chance node**s) (ovális) valószínűségi változókat jelölnek, pont úgy, ahogy a valószínűségi hálókban. Az ágens lehet bizonytalan az építési költségekben, a légi forgalom szintjében és a pereskedés lehetősége miatt, valamint a *Halálesetek*, a *Zaj* és a teljes *Költség* változókban, amelyek mindegyike szintén függ a kiválasztott helyszíntől. Minden egyes véletlen csomópontnak van egy feltételes valószínűségi táblája (FVT), ami a szülőcsomópontok állapotaival indexelt. A döntési hálókban a szülőcsomópontok lehetnek döntési csomópontok és véletlen csomópontok. Kiemelendő, hogy a jelenlegi-állapot véletlen csomópontok bármelyike része lehet egy nagyobb valószínűségi hálónak, ami az építkezés költségét, a légi forgalom nagyságát vagy a perek lehetőségét becsüli meg.

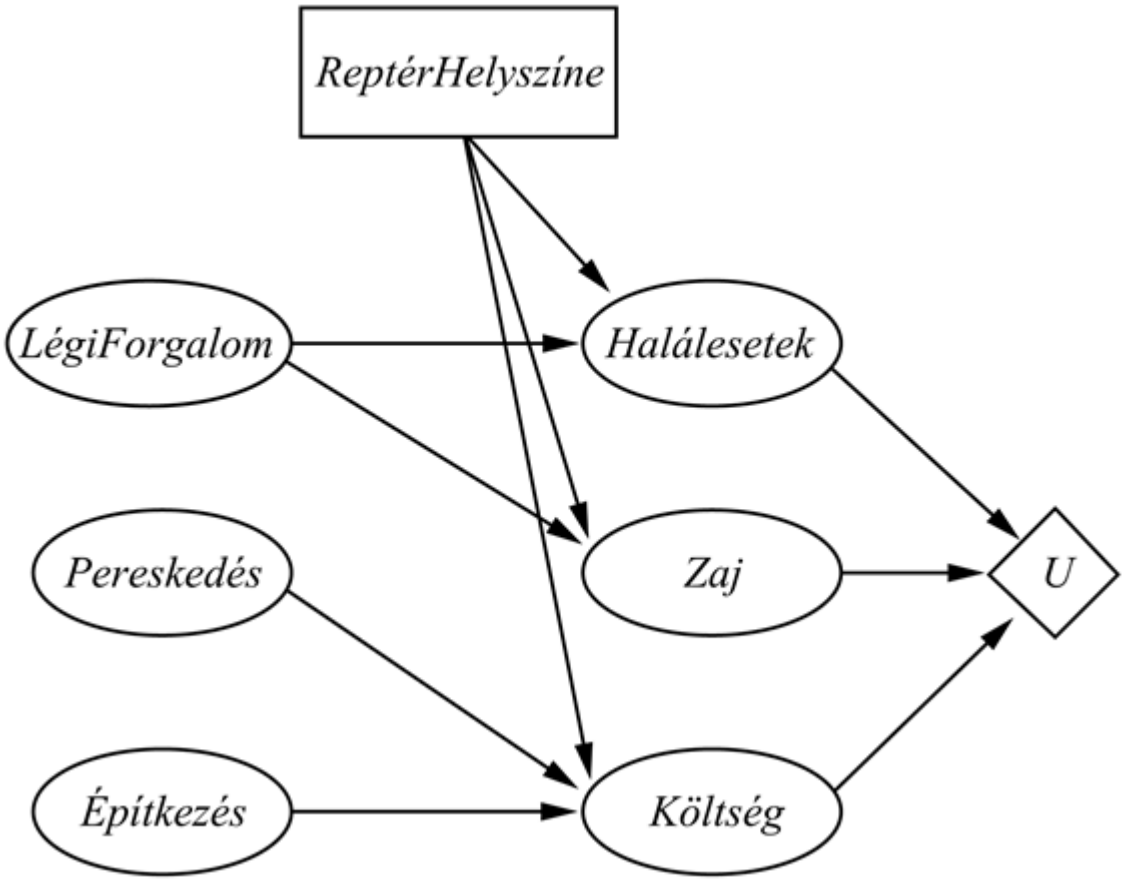
- A **döntési csomópont**ok (**decision node**s) (négyszögek) azokat a beavatkozási pontokat reprezentálják, ahol a döntéshozónak lehetősége van a cselekvésre. Ebben az esetben a *ReptérHelyszíne* különböző értékeket vehet fel a megfontolás alatt álló helyszíneknek megfelelően. A választás befolyásolni fogja a kiadódó költséget, a biztonságot és a repülőtér által keltett zajt. Ebben a fejezetben feltételezzük, hogy egyetlen döntéssel foglalkozunk (azaz egyetlen ilyen csomópont létezik). A 17. fejezet tárgyalja azokat az eseteket, ahol több mint egy döntést kell meghozni.

- A **hasznosságcsomópont**ok (**utility node**s) (rombusz) az ágens hasznosságfüggvényét reprezentálják.[4] A hasznosságcsomópontnak szülője az összes olyan változó, amelyek által leírt kimeneteli állapotok közvetlenül befolyásolják a hasznosságot. A hasznosságcsomóponthoz tartozik egy leírás, ami az ágens hasznosságát adja meg egy szülői attribútumokon definiált függvénnyel. A leírás lehet a függvény egy táblázatos megadása, vagy lehet egy parametrikus additív vagy lineáris függvény.

16.5. ábra - Egy egyszerű döntési hálózat a repülőtér-elhelyezési problémára

[4] Ezeket a csomópontokat gyakran **értékcsomópont**oknak (**value node**s) nevezik az irodalomban. Mi jobbnak láttuk megtartani a különbséget a hasznosság- és az értékfüggvények között, ahogyan ezt korábban indokoltuk, ezért lehet, hogy a kimeneti állapot egy szerencsejátékot reprezentál.

Created by XMLmind XSL-FO Converter.



Sok esetben csupán egy egyszerűsített formát használunk. A jelölés ugyanaz marad, de az a véletlen csomópont, ami a kimeneti állapotot jelöli, elmarad. Ehelyett, a hasznosságcsomópontot közvetlenül a jelenlegi állapot csomópontokhoz és a döntési csomóponthoz kötik. Ebben az esetben a hasznosságcsomópont ahelyett, hogy az állapotokon definiált hasznosságfüggvényt reprezentálná, az egyes cselekvésekre vonatkozó várható hasznosságokat reprezentálja, ahogy azt a (16.1) egyenlet definiálja. Ezért az ilyen táblákat **cselekvéshasznosság tábl**aknak (**action-utility table**s) nevezzük. A 16.6. ábra a repülőtér probléma cselekvéshasznosság-reprezentációját mutatja.

Láthatjuk, hogy mivel a *Zaj*, a *Halálesetek* és a *Költség* véletlen csomópontok a 16.5. ábrán a jövőbeli állapotokra vonatkoznak, ezek soha nem kaphatnak értéket, mint tényváltozók. Így az egyszerűsített verzió, amiből elhagytuk ezeket a csomópontokat, minden olyan esetben használható, amikor az általánosabb forma használható. Bár az egyszerűsített forma kevesebb csomópontot tartalmaz, az elhelyezés kimeneteleihez kapcsolódó közvetlen leírások elhagyása azt eredményezi, hogy kevésbé rugalmas a körülmények esetleges változásaira. Például a 16.5. ábrán a repülőgép zajszintjében bekövetkező változást a *Zaj* csomópont feltételes valószínűségi táblájának megfelelő megváltoztatásával lehet modellezni, ezzel szemben a zajszennyezés fontosságának a megváltozását a hasznosságfüggvényben a hasznosságtábla megváltoztatásával modellezhetjük. Másfelől a cselekvéshasznosság diagramban – 16.6. ábra – létrejövő minden ilyenfajta változtatást a cselekvéshasznosság táblázat megváltoztatásának kell tükröznie. Lényegében a cselekvéshasznosság forma az eredeti forma egy *szerkesztett* verziója.

5.2. Döntési hálók kiértékelése

A cselekvéseket úgy választjuk ki, hogy a döntési hálót kiértékeljük a döntési csomópont minden lehetséges beállítására. Ha egyszer a döntési csomópontot beállítottuk, úgy fog viselkedni, mint egy véletlen csomópont, amit egy tényváltozónak állítottunk be. A döntési háló kiértékelését a következő algoritmus végzi:

1. Állítsa be a tényváltozókat a jelenlegi állapotra.

2. A döntési csomópont minden egyes értékére:

Created by XMLmind XSL-FO Converter.

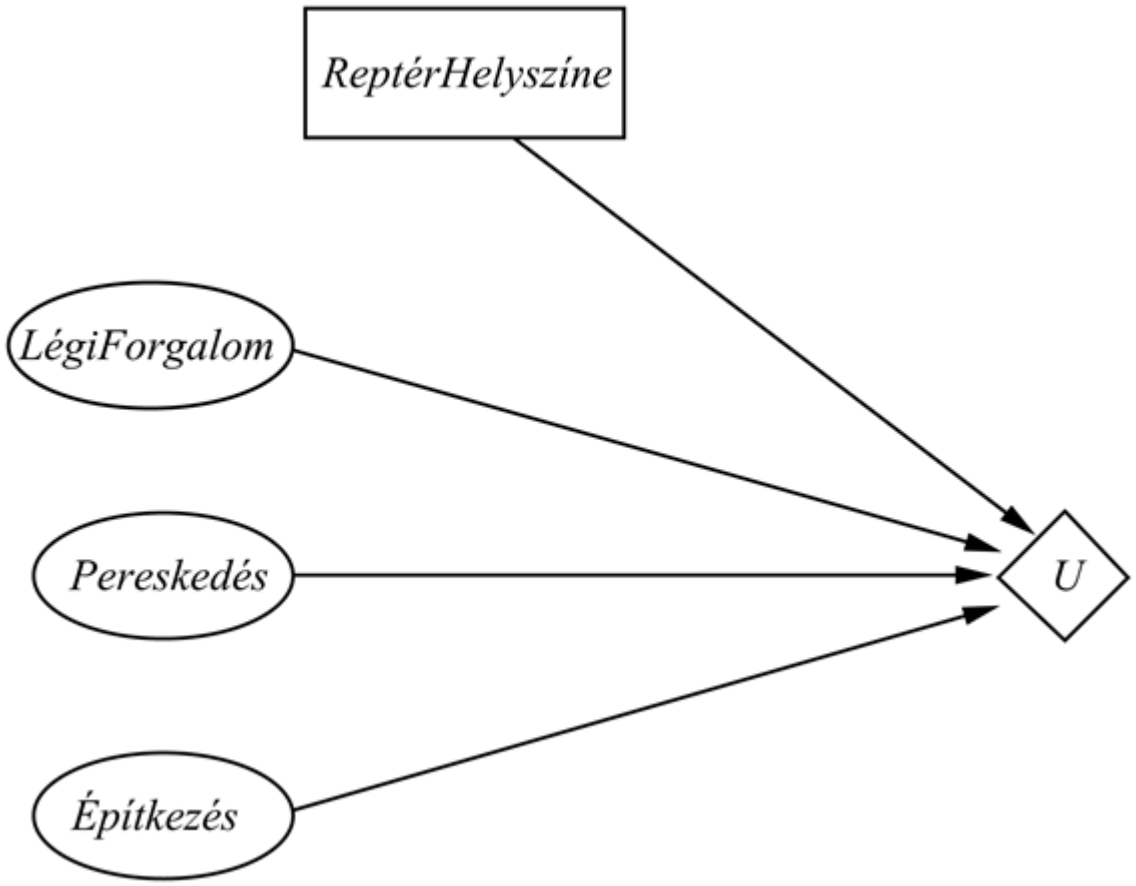
a. Állítsa be a döntési csomópontot erre az értékre.

b. Számítsa ki az a posteriori valószínűségeket a hasznosságcsomópont szüleire, egy szabványos valószínűségi következtető algoritmust használva.

c. Számítsa ki a cselekvés hasznosságát.

3. Térjen vissza a legnagyobb hasznosságértékkel.

16.6. ábra - A repülőtér-elhelyezési probléma egy egyszerűsített reprezentációja. A kimeneti állapotokhoz tartozó véletlen csomópontok hatását kiszámítottuk, majd a csomópontokat elhagytuk.



Ez a valószínűségi hálók következtető algoritmusának egy nyilvánvaló kiterjesztése, és közvetlenül beilleszthető az ágens 13.1. ábrán megadott tervezésébe. A 17. fejezetben látni fogjuk, hogy több egymást követő cselekvés lehetősége a problémát sokkal érdekesebbé teszi.

6. Az információ értéke

Fontos

Az előző elemzésben feltételeztük, hogy minden lényeges információ, de legalábbis minden elérhető információ az ágens rendelkezésére áll. Gyakorlatban ez aligha történhet meg. *A döntéshozatal egyik legfontosabb része, hogy tudjuk, milyen kérdést kell feltenni.* Például az orvos nem várhatja el, hogy rendelkezésére álljon minden lehetséges diagnosztikai teszt és kérdés eredménye, amikor a páciens először belép a rendelőbe.[5] A tesztek gyakran költségesek és néha veszélyesek (mind közvetlenül, mind az általuk okozott késedelmek miatt). A fontosságuk két tényezőn múlik: egyrészt azon, hogy

[5] Az Egyesült Államokban az egyetlen kérdés, amit ez előtt mindig megkérdeznek, hogy a páciensnek van-e biztosítása.

Created by XMLmind XSL-FO Converter.

jelentősen jobb kezelés válik-e lehetővé a teszteredmények alapján, másrészt a különböző teszteredmények valószínűségén.

Ez az alfejezet az **információérték-elmélet**tel (**information value theory**) foglalkozik, ami lehetővé teszi az ágens számára annak kiválasztását, hogy milyen információt szerezzen meg. Az információ megszerzése **érzékelési cselekvés**ekkel (**sensing action**) történik, ahogy a 12. fejezetben leírtuk. Mivel az ágens hasznosságfüggvénye ritkán vonatkozik az ágens belső állapotaira, jóllehet az érzékelési cselekvés célja a belső állapot megváltoztatása, ezért az érzékelési cselekvéseket az ágens ezt követő műveleteire gyakorolt hatása alapján kell kiértékelnünk. Az információérték-elmélet ezért a többlépcsős döntéshozatal egy speciális fajtája.

6.1. Egy egyszerű példa

Tételezzük fel, hogy egy olajtársaság egy tengeri fúróhely létesítésére szeretne jogokat vásárolni, n darab megkülönböztethetetlen terület közül választva. Tételezzük továbbá fel, hogy kizárólag csak az egyik terület tartalmaz olajat C dollár értékben, és hogy a területek ára C/n dollár. Ha a társaság kockázatsemleges, akkor közömbös lesz számára, hogy vásárol-e területet, vagy sem.

Most tételezzük fel, hogy a szeizmológusok felkínálják a 3-as számú területre vonatkozó kutatásaik eredményét a társaságnak, ami világosan megmutatja, hogy a terület tartalmaz-e olajat, vagy sem. Mennyit legyen hajlandó fizetni a társaság ezért az információért? E kérdés megválaszolásának az a módja, hogy megvizsgáljuk, hogy mit tenne a társaság, ha meglenne az információja:

- $1/n$ valószínűséggel a 3-as terület felmérése azt fogja mutatni, hogy a terület tartalmaz olajat. Ebben az esetben a társaság megvásárolja a 3-as területet C/n dollárért, és nyeresége $C - C/n = (n - 1)C/n$ dollár lesz.
- $(n - 1)/n$ valószínűséggel a felmérés azt fogja mutatni, hogy a terület nem tartalmaz olajat, amely esetben a társaság egy másik területet vásárol meg. Ekkor az olajterület eltalálásának a valószínűsége a többi mező között $1/n$-ről $1/(n - 1)$-re változik, így a társaság várható nyeresége $C/(n - 1) - C/n = C/n(n - 1)$ dollár lesz.

Most már kiszámíthatjuk a várható nyereséget, ha ismerjük a kutatás eredményét:

$$\frac{1}{n}\times\frac{(n-1)C}{n}+\frac{n-1}{n}\times\frac{C}{n(n-1)}=C/n$$

Tehát a társaságnak hajlandónak kell lennie, hogy akár C/n dollárt fizessen a szeizmológusnak az információért: az információ annyit ér, mint maga a terület.

Az információ értéke abból a tényből származik, hogy az *információval* valaki az aktuális helyzetben sokkal alkalmasabb cselekedeteket tud megválasztani. Az információval lehetségessé válik a helyzetek megkülönböztetése, ezzel szemben az információ nélkül a lehetséges helyzetek feletti átlagot nézve kell cselekedni. Általánosan fogalmazva, az információ értékét a várható értékek különbsége definiálja az információ megszerzése előtt és után.

6.2. Egy általános képlet

Az információ értékére egyszerű általános formulát adni. Rendszerint feltesszük, hogy pontos bizonyítékaink vannak egyes E_j valószínűségi változók értékeiről, így a **teljes információ értéke** (**value of perfect information**) (**TIÉ**) elnevezést használjuk.[6] Legyen az ágens jelenlegi tudása E. Ekkor az α pillanatnyi legjobb cselekvés értékét a következő definiálja:

$$EU(\alpha|E)=\max_A\sum_i U(Eredmény_i(A))P(Eredmény_i(A)|Tesz(A),E)$$

és a legjobb cselekvés értéke (az E_j új bizonyíték megszerzése után)

$$EU(\alpha_{E_j}|E,E_j)=\max_A\sum_i U(Eredmény_i(A))P(Eredmény_i(A)|Tesz(A),E,E_j)$$

[6] Egy X változóra vonatkozó nem teljes információt modellezhetünk egy Y változóról szóló teljes információval, ahol az Y változó valószínűségi kapcsolatban van X-szel. Erre ad példát a 16.11. feladat.

Created by XMLmind XSL-FO Converter.

De E_j egy valószínűségi változó, aminek az értéke *jelenleg* nem ismert, így átlagolnunk kell az összes lehetséges e_{jk} értékre, amelyeket E_j értékeként az E_j-re vonatkozó *jelenlegi* hiedelmünket felhasználva megkaphatunk. Az E_j megszerzésének értékét a következőképpen definiáljuk:

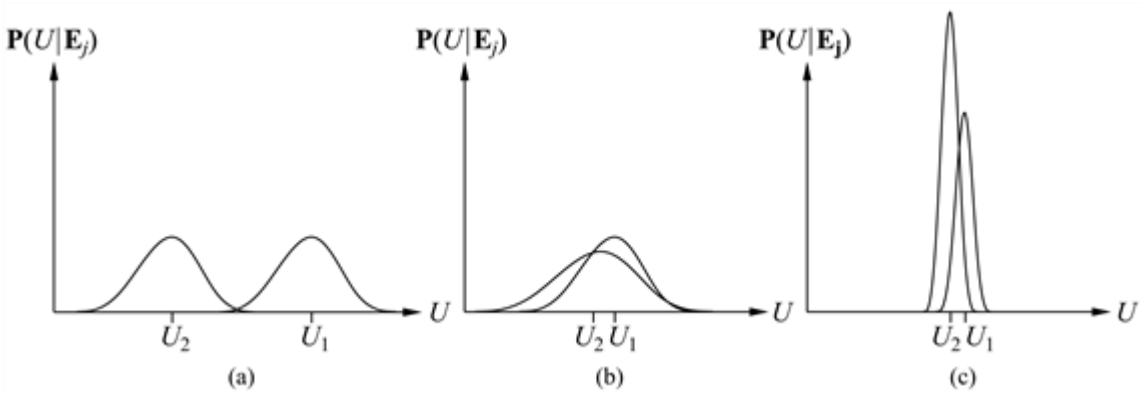
$$TIÉ_E(E_j) = \left(\sum_k P(E_j = e_{jk} | E) EU(\alpha_{e_{jk}} | E, E_j = e_{jk}) \right) - EU(\alpha | E)$$

A formula megértése érdekében gondoljuk át azt az egyszerű esetet, mikor csak két cselekvés, A_1 és A_2 közül lehet választani. A cselekvések jelenleg várható hasznosságai U_1 és U_2. Az E_j információ új várható hasznosságokat, U_1' -et és U_2'-t fog eredményezni a cselekvésekre, de E_j megszerzése előtt már ismerjük a függetlennek feltételezett U_1' és U_2' lehetséges értékeinek valószínűség-eloszlásait.

Tegyük fel, hogy A_1 és A_2 két különböző utat jelent a hegyeken keresztül, téli időben. A_1 egy kényelmes, egyenes autópálya egy alacsony hágón át, míg A_2 egy tekervényes földút a csúcson keresztül. Csak ezt az információt ismerve A_1 egyértelműen előnyben részesíthető, mivel elég valószínű, hogy a második utat lavinák zárják le, miközben az első úton valószínűleg nincsenek torlódások. U_1 ezért egyértelműen nagyobb, mint U_2. Az utak aktuális állapotáról E_j műholdjelentéseket kaphatunk, melyek U_1' és U_2' új várható értékeket adnak az átkelőkre vonatkozóan. A várható hasznosságértékek eloszlását a 16.7. (a) ábra mutatja. Nyilvánvalóan ebben az esetben a költségek miatt nem éri meg a műholdas jelentéseket kikérni, mivel nagyon valószínűtlen, hogy ezek a jelentések a terv megváltozását eredményeznék. A tervek változatlansága mellett azonban az információnak nincs értéke.

Most tételezzük fel, hogy két, kissé eltérő hosszúságú kanyargós földút közül választunk, és súlyosan sérült utasokat viszünk. Ekkor bár lehet, hogy U_1 és U_2 elég közel esnek egymáshoz, az U_1' és U_2' széles tartományon belül vehetnek fel értékeket. Jelentős esélye van annak, hogy a második út tiszta lesz, míg az első le lesz zárva, és ebben az esetben a hasznosságok különbsége nagyon nagy lesz. A *TIÉ* formula azt jelzi, hogy a műholdjelentéseket megéri kikérni. Ezt a helyzetet a 16.7. (b) ábra mutatja.

16.7. ábra - Három általános eset az információ értékére. Az (a) esetben A_1 majdnem teljes bizonyossággal jobb marad A_2-höz képest, így az információ nem szükséges. A (b) esetben a választás nem egyértelmű, és az információ döntő fontosságú. A (c) esetben a választás bizonytalan, de igazából nem számít, így az információ is kevésbé értékes.



Tételezzük most azt fel, hogy nyári időben választunk a két földút közül, amikor a lavinák igen valószínűtlenek. Ebben az esetben a műholdjelentések azt mutathatják, hogy az egyik útvonal látványosabb, mint a másik, a virágzó hegyi rétek miatt, vagy talán csak nedvesebb az arra kanyargó patakok miatt. Ekkor igen valószínű, hogy ha megkapjuk ezt az információt, megváltoztatjuk a tervünket. Azonban ebben az esetben, a különbség a két út között még mindig igen kicsi, így nem foglalkozunk a jelentések megszerzésével. Ezt a helyzetet a 16.7. (c) ábra mutatja.

Fontos

Összegzésül azt mondhatjuk, hogy *az információnak annyiban van értéke, amennyire valószínű, hogy a terv megváltozását okozza, és amennyire az új terv jelentősen jobb lesz, mint a régi.*

Created by XMLmind XSL-FO Converter.

6.3. Az információ értékének tulajdonságai

Érdekes kérdés, hogy lehetséges-e az, hogy az információ káros – lehet-e az információnak negatív várható értéke? Az a megérzésünk, hogy ez lehetetlen. Végül is legrosszabb esetben az információ egyszerűen figyelmen kívül hagyható, és úgy tehetünk, mintha soha nem kaptuk volna meg. Ezt igazolja a következő tétel, ami bármely döntéselméleti ágensre alkalmazható. *Az információ értéke nem negatív:*

$$\forall j,\ E \quad TIÉ_E(E_j) \geq 0$$

A tétel közvetlenül következik a *TIÉ* definíciójából, a bizonyítást gyakorló feladatként az olvasóra bízzuk (16.12. feladat). Fontos emlékezni arra, hogy a *TIÉ* a jelenlegi információs állapottól függ, ezért szerepel ez az alsó indexben. Ez megváltozhat, amint több információt szerzünk. Szélsőséges esetben nulla is lehet, ha a kérdéses változó értéke már ismert. Így a *TIÉ* nem additív. Azaz

$$TIÉ_E(E_j, E_k) \neq TIÉ_E(E_j) + TIÉ_E(E_k) \quad \text{(általánosságban)}$$

Azonban a *TIÉ* sorrendfüggetlen, aminek intuitíve nyilvánvalónak kell lennie. Azaz,

$$TIÉ_E(E_j, E_k) = TIÉ_E(E_j) + TIÉ_{E,E_j}(E_k) = TIÉ_E(E_k) + TIÉ_{E,E_k}(E_j)$$

A sorrendfüggetlenség megkülönbözteti az érzékelési és a hagyományos cselekvéseket, és egyszerűsíti egy érzékelési cselekvéssorozat értékének a kiszámítását.

6.4. Egy információgyűjtő ágens megvalósítása

Egy értelmes ágensnek a felhasználóhoz ésszerű sorrendben kell kérdéseket intéznie, nem szabad lényegtelen kérdéseket feltennie, az információ fontosságát a költségéhez kell viszonyítania, és a megfelelő ponton abba kell hagynia a kérdezősködést. Mindezen képességek elérhetők az információérték használatának a segítségével.

A 16.8. ábra egy olyan ágens teljes vázát mutatja, ami mielőtt cselekedne, képes intelligensen információkat gyűjteni. Időlegesen tételezzük fel, hogy minden megfigyelhető E_j tényváltozóhoz létezik egy hozzárendelt költség, *Költség*(E_j), ami a bizonyíték tesztek, tanácsadók, kérdések és bármi egyéb módszer segítségével történő megszerzésének a költsége. Az ágens a legértékesebb információt kéri el, a költségeket is figyelembe véve. Feltételezzük, hogy a *Kérés*(E_j) cselekvés eredménye az, hogy a következő érzékelés az E_j értéket szolgáltatja. Ha nincs olyan megfigyelés, ami megérné a költségét, az ágens egy nem információgyűjtő cselekvést választ ki.

16.8. ábra - Egy egyszerű információgyűjtő ágens váza. Az ágens folyamatosan a legnagyobb információértékű megfigyelést választja, mindaddig, amíg a következő megfigyelés költsége kisebb, mint a várható haszna.

```
function INFORMÁCIÓ-GYŰJTŐ-ÁGENS(észlelés) returns egy művelet
   static: D, egy döntési hálózat

   olvassuk az észlelést a D döntési hálóba
   j ← TIÉ(E_j)-t maximalizáló érték – Költség(E_j)
   if TIÉ(E_j) > Költség(E_j)
      then return KÉRDEZ(E_j)
else return a D-beli legjobb művelet
```

Az itt megadott ágens az információgyűjtés úgynevezett **rövidlátó** (**myopic**) formáját valósítja meg. Ez azért van, mert az ágens a *TIÉ* formulát rövidlátó módon használja, azaz úgy számítja ki az információ értékét, mintha csak egyetlen tényváltozót szerezne meg. Ha nincsen olyan tényváltozó, ami sokat segítene, lehet, hogy a rövidlátó ágens kapkodó módon belekezd egy cselekvésbe, pedig jobb lett volna előbb lekérdezni két vagy három változót, és csak aztán cselekedni. A rövidlátó szabályozás ugyanazon a heurisztikán alapul, mint a mohó keresés, és a gyakorlatban gyakran jól működik. (Például kimutatták, hogy diagnosztikai tesztek kiválasztásában jobb a teljesítménye, mint az orvosszakértőknek.) Azonban egy teljesen racionális információgyűjtő ágensnek végig kell gondolnia az összes lehetséges információkérés-szekvenciát, ami egy külső akcióban végződik, és

Created by XMLmind XSL-FO Converter.

ezeknek a kéréseknek az összes lehetséges kimenetelét. Mivel a második kérés értéke függ az első kérés kimenetelétől, az ágensnek fel kell derítenie a feltételes tervek terét, amint azt a 12. fejezetben leírtuk.

7. Döntéselméleti szakértő rendszerek

Az 1950-es és az 1960-as években kifejlődött **döntéselemzés** (**decision analysis**) a döntéselmélet alkalmazását tanulmányozza konkrét döntési problémákon. Fő alkalmazási területe a kiemelt fontosságú területeken meghozandó döntések segítése, ahol a tétek igen nagyok, ilyen területek például az üzleti élet, a kormányzati munka, a törvényhozás, a katonai stratégia, az orvosi diagnosztika és a közegészségügy, a mérnöki tervezés és az erőforrás-gazdálkodás. A folyamat magában foglalja a lehetséges cselekvések és kimenetelek gondos tanulmányozását csakúgy, mint a kimenetelek közötti preferenciákét. A döntéselemzésben hagyományosan két szerepet tételeznek fel: a **döntéshozó** (**decision maker**) a kimenetelek közötti preferenciákat határozza meg, a **döntéselemző** (**decision analyst**) pedig a lehetséges cselekvéseket, kimeneteleket veszi számba, és kikérdezi a preferenciákat a döntéshozótól, hogy meghatározza a legjobb cselekvést. Az 1980-as évek elejéig a döntéselemzés célját abban látták, hogy az embereket segítsék olyan döntések meghozatalában, amik a tényleges preferenciáiknak felelnek meg. Manapság egyre több és több döntési folyamat automatizált, és a döntéselemzést annak biztosítására használják, hogy az automatizált folyamatok megfelelően alakuljanak.

Ahogy erről a 14. fejezetben szóltunk, a korai szakértői rendszerekkel kapcsolatos kutatás inkább kérdések megválaszolására irányult, mint döntések meghozatalára. Azok a rendszerek, amelyek cselekvéseket javasoltak a véleménynyilvánítás helyett, valójában ha-akkor szabályokkal érték ezt el, nem pedig a kimenetelek és preferenciák közvetlen reprezentációjával. A valószínűségi hálók megjelenése az 1980-as évek vége körül azonban lehetővé tette olyan nagy rendszerek megépítését, amelyek helyes valószínűségi következtetéseket származtattak a tényekből. A döntési hálók megjelenése azt jelentette, hogy olyan szakértői rendszereket lehet kifejleszteni, amelyek optimális döntéseket javasolnak a felhasználó preferenciáinak és az elérhető tényeknek megfelelően.

Egy rendszer, ami a hasznosságokat közvetlenül tartalmazza, elkerülheti a tanácsadói folyamatok egy gyakori csapdáját: a valószínűség és a fontosság összekeverését. Megszokott módszer volt például a korai orvosszakértői rendszerekben, hogy a lehetséges diagnózisokat a valószínűségük alapján sorba állították, és a legvalószínűbbet jelentették. Sajnos ez végzetes lehet! Általában a páciensek többségénél a két legvalószínűbb diagnózis többnyire az, hogy „semmi baj sincs” és „megfázott”, ha azonban a harmadik legvalószínűbb diagnózis egy páciensnél a tüdőrák, az már aggodalomra ad okot. Nyilvánvaló, hogy a teszteknek és a kezeléseknek mind a valószínűségektől, mind a hasznosságoktól kell függeniük.

A következőkben leírunk egy döntéselméleti szakértői rendszerhez tartozó tudásmérnöki folyamatot. Példaként egy orvosi problémát ismertetünk, amiben kezeléseket kell megválasztani bizonyos veleszületett szívrendellenességek esetén (Lucas, 1996).

A gyermekek 0,8 százaléka szívrendellenességgel születik, ami többnyire **aortaszűkület** (**aortic coarctation**). Ez a rendellenesség sebészi beavatkozással, érműtéttel (kitágítják az aortát egy ballont helyezve az artériába) vagy gyógyszerrel kezelhető. A feladat a kezelés megválasztása és annak eldöntése, hogy mikor végezzék el: minél fiatalabb egy gyermek, annál nagyobb a kockázata bizonyos kezeléseknek, de túl sokáig sem szabad várni. A problémához egy döntéselméleti szakértői rendszer készíthető egy olyan csapattal, amelynek legalább egy orvosszakértő (egy gyermekkardiológus) és egy tudásmérnök tagja van. A folyamat a következő lépésekre bontható (amelyek összehasonlíthatók a logikai rendszerek tervezésének lépéseivel a 8.4. alfejezetben).

Alkosson meg egy oksági modellt. Határozza meg a lehetséges tüneteket, rendellenességeket, kezeléseket és kimeneteleket. Aztán jelezze közöttük haladó nyilakkal, hogy melyik rendellenesség milyen tüneteket okozhat, és melyik kezelés milyen rendellenességekre hatásos. Ezek egy része a tárgyterületi szakértő számára ismert lesz, más részük a szakirodalomból származhatnak. Gyakran a modell jól fog illeszkedni a szakkönyvek leírásaiban szereplő informális diagramokhoz.

Egyszerűsítse a kvalitatív döntési modellt. Mivel a modellt kezelések közötti választásra tervezzük használni, és nem más célokra (mint például bizonyos rendellenességek/tünetek kombinációihoz rendelhető együttes valószínűségek kiszámítására), így gyakran elhagyhatunk változókat, amelyek nem befolyásolják a kezelések megválasztását. Gyakran a változókat fel kell bontani vagy össze kell olvasztani, hogy megfeleljen a szakértő gondolkodásmódjának. Például az eredeti aortaszűkületi modellben szerepelt egy *Kezelés* változó *műtét, érsebészet* vagy *gyógyszeres kezelés* értékekkel, és egy önálló változó a kezelés *Időzítés*-ére. Azonban a

Created by XMLmind XSL-FO Converter.

szakértőnek nehéz volt ezeket külön kezelni, ezért össze kellett olvasztani, így a *Kezelés* olyan értékeket vehet fel, mint *műtét egy hónapon belül*. Ezt a modellt mutatja be a 16.9. ábra.

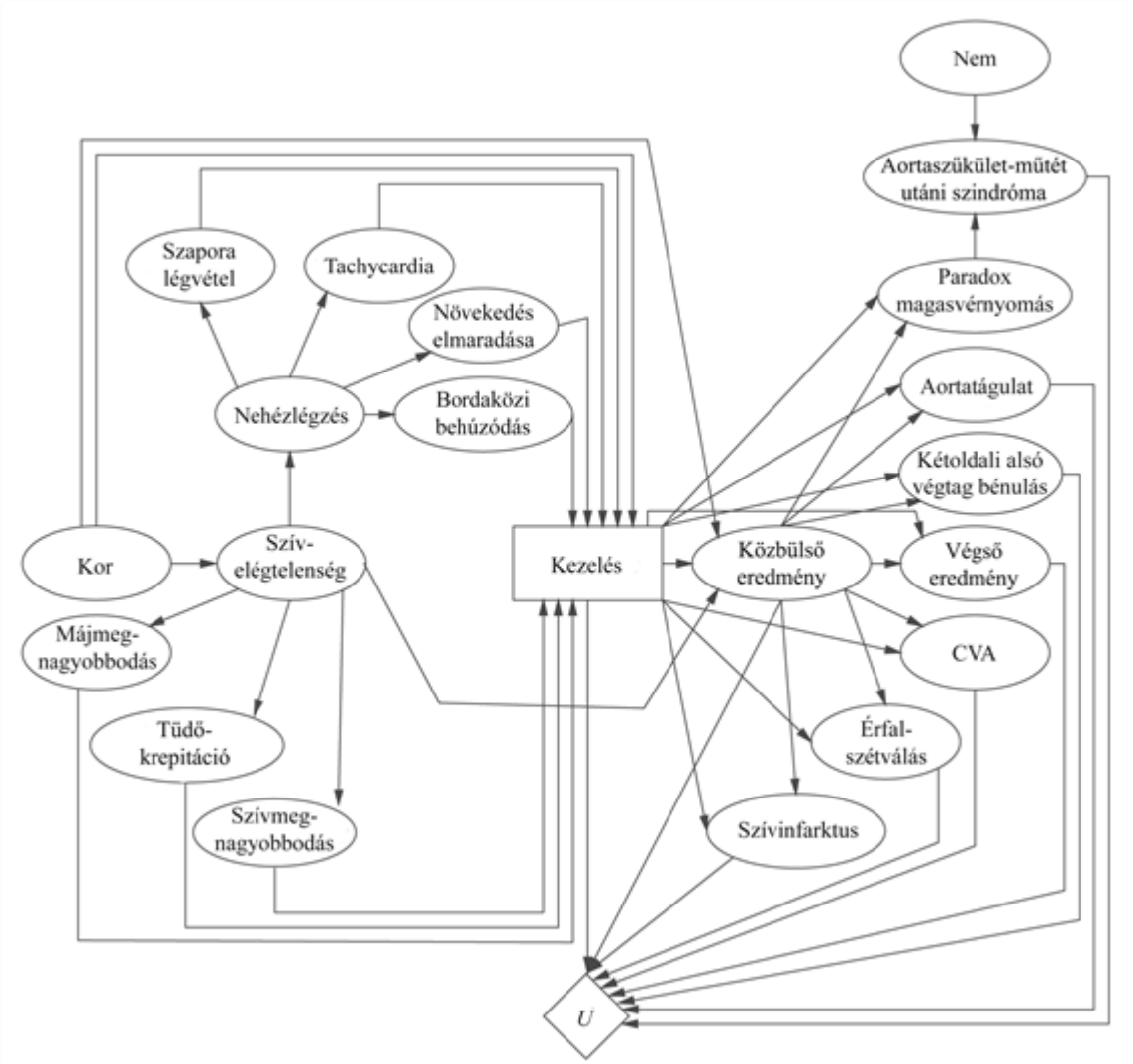
Rögzítse a valószínűségeket. A valószínűségek betegek adatbázisából, szakirodalmi tanulmányokból vagy szakértők szubjektív becsléseiből származhatnak. Azokban az esetekben, amikor nem megfelelő formában érhetők el a szakirodalomban a valószínűségek, akkor Bayes-szabállyal és vetítéssel számíthatjuk ki a kívánt értékeket. Bebizonyosodott, hogy a szakértők leginkább egy ok hatásának a valószínűségét tudják megbecsülni (például $P(nehézlégzés|szívelégtelenség)$) sokkal inkább, mint a másik irányban.

Rögzítse a hasznosságokat. Kevés számú kimenetel esetén ezek felsorolhatók és egyenként kiértékelhetők. Meghatározhatunk egy skálát a legrosszabbtól a legjobb kimenetelig, mindegyikhez egy numerikus értéket rendelve, például –1000-et az elhalálozáshoz és 0-t a teljes felépüléshez. Ekkor a többi kimenetel már ezen a skálán elhelyezhető. Ezt elvégezheti egy szakértő, de jobb, ha a betegek (vagy gyermekek esetén a szüleik) is részt vesznek ebben, mivel a különböző embereknek különböző preferenciáik vannak. Exponenciálisan sok kimenetel esetén szükségünk van olyan módszerekre, hogy többattribútumú hasznosságfüggvényekkel kombinálhassuk őket. Például feltehetjük, hogy a különböző komplikációk negatív hasznossága additív.

Ellenőrizze és finomítsa a modellt. A rendszer kiértékeléséhez szükségünk van helyes (bemenet, kimenet) párokra; az úgynevezett **referenciá**kra (**gold standard**), amihez lehet hasonlítani. Ez orvosszakértő rendszerek esetén azt jelenti, hogy összehívjuk az elérhető legjobb orvosszakértőket, néhány beteg esetét megmutatjuk nekik, diagnózis felállítására és kezelési javaslat meghatározására kérjük őket. Majd megnézzük, hogy a rendszer mennyire illeszkedik a javaslataikhoz. Ha csak kevéssé, akkor megpróbáljuk a rosszul teljesítő részeket azonosítani és megjavítani. Hasznos lehet a rendszert „visszafelé” tesztelni. Ahelyett hogy a rendszerbe a tüneteket vinnénk be, és diagnózist kérnék, vigyük be a diagnózist, mint például szívelégtelenség, és vizsgáljuk meg a tünetek, mint például szapora szívverés, jósolt valószínűségeit, és hasonlítsuk ezt össze szakirodalmi adatokkal.

16.9. ábra - Az aortaszűkület hatásdiagramja (Peter Lucas hozzájárulásával)

Created by XMLmind XSL-FO Converter.



Végezzen érzékenységvizsgálatot. Ez a fontos lépés azt ellenőrzi, hogy a legjobb döntés érzékeny-e a megadott valószínűségek és hasznosságok kis változására, ehhez szisztematikusan változtassuk meg ezeket a paramétereket, és futtassuk le a kiértékelést újra. Ha kis változások jelentősen eltérő döntést eredményeznek, akkor lehet, hogy megéri több energiát rászánni, hogy pontosabb adatokat gyűjtsünk. Ha minden változtatás ugyanahhoz a döntéshez vezet, akkor a felhasználó biztos lehet, hogy ez a helyes döntés. Az érzékenységvizsgálat különösen fontos lehet, mivel az egyik fő kritikája a szakértői rendszerek valószínűségi megközelítésének, hogy nagyon nehéz a szükséges numerikus valószínűségek megbecslése. Az érzékenységvizsgálat gyakran kimutatja, hogy sok numerikus értéket elég közelítőleg meghatározni. Például bizonytalanok lehetünk a $P(tachycardia)$ a priori valószínűségben, de ha kipróbálunk különböző értékeket erre a valószínűségre, és minden egyes esetben a javasolt cselekmény a hatásdiagram alapján ugyanaz, akkor kevésbé kell foglalkoznunk ezzel az ismerethiánnyal.

8. Összefoglalás

Ez a fejezet megmutatta, hogy hogyan kombinálhatjuk össze a hasznosságelméletet a valószínűségekkel, lehetővé téve az ágensnek olyan cselekvések kiválasztását, amelyek maximalizálni fogják a várható teljesítményét.

- A **valószínűség-elmélet** (**probability theory**) előírja, hogy az ágens a tényei alapján milyen hiedelmekkel rendelkezzen, a **hasznosságelmélet** (**utility theory**) azt írja elő, hogy az ágens mit akar, a **döntéselmélet** (**decision theory**) pedig összeilleszti a kettőt, és előírja, hogy az ágens mit tegyen.

Created by XMLmind XSL-FO Converter.

- A döntéselmélet felhasználásával építhetünk egy olyan rendszert, ami a döntések meghozatalakor figyelembe veszi az összes lehetséges cselekvést, és azt választja ki, ami a legjobb várható kimenetelt adja. Egy ilyen rendszert **racionális ágens**nek (**rational agent**) nevezhetünk.

- A hasznosságelmélet megmutatja, hogy az ágens, amelynek szerencsejátékok közötti preferenciái eleget tesznek bizonyos egyszerű axiómáknak, leírható egy hasznosságfüggvénnyel; továbbá az ágens a cselekvéseit úgy választja meg, mintha a várható hasznosságát maximalizálná.

- A **többattribútumú hasznosságelmélet** (**multiattribute utility theory**) olyan hasznosságokkal foglalkozik, amelyek az állapotok több, különböző attribútumától függnek. A **sztochasztikus dominancia** (**stochastic dominance**) egy különösen hasznos technika egyértelmű döntések meghozatalára még az attribútumok pontos hasznosságértékei nélkül is.

- A **döntési hálók** (**decision networks**) egy egyszerű formalizmust biztosítanak a döntési problémák leírására és megoldására. Ezek a valószínűségi hálók magától értetődő kiterjesztései döntési és hasznossági csomópontokkal a véletlen csomópontokon túl.

- Bizonyos esetekben a probléma megoldásához hozzátartozik a többletinformáció megszerzése a döntés meghozatala előtt. Az **információ értéké**t (**value of information**) úgy definiáljuk, mint a várható hasznosság javulását az információ nélküli helyzethez hasonlítva.

- Azok a **szakértő rendszer**ek (**expert system**s), amelyek hasznosságinformációkat is tartalmaznak, többletképességgel rendelkeznek a tisztán következtető rendszerekhez képest. Azon túl, hogy döntéseket tudnak hozni, dönthetnek az információ megszerzéséről annak értéke alapján, és kiszámíthatják döntéseiknek a valószínűségek és hasznosságok kis megváltozására vonatkozó érzékenységét.

8.1. Irodalmi és történeti megjegyzések

A maximális várható hasznosság elvének az egyik legkorábbi, bár végtelen hasznosságokat is tartalmazó szokatlan alkalmazása Pascal fogadása volt, melyet először mint a *Port-Royal Logic* egy részét Arnauld közölt (Arnauld, 1662). Daniel Bernoulli volt az első, a szentpétervári paradoxont vizsgálva, aki felismerte a szerencsejátékokra vonatkozó preferenciák felmérésének a fontosságát, megállapítva, hogy „egy dolog *értékét* nem az *árának* kell meghatároznia, hanem a *hasznosságának*, amit eredményez” (Daniel Bernoulli, 1738, eredeti kiemelés). Jeremy Bentham javasolta a **hedonisztikus kalkulus**t (**hedonistic calculus**) az „örömök” és „bánatok” összehasonlítására, amellett érvelve, hogy minden döntésnek (nem csak a pénzügyieknek) redukálhatónak kell lennie hasznosságok összehasonlítására (Bentham, 1823).

A numerikus hasznosságok származtatása a preferenciákból Ramseynél jelent meg először (Ramsey, 1931); a jelenlegi szövegben a preferenciákra megadott axiómák megfogalmazásukban közelebb állnak azokhoz az axiómákhoz, amelyek egy újrafelfedezés során a *Theory of Games and Economic Behavior*ben (Neumann és Morgenstern, 1944) jelentek meg. Ezeknek az axiómáknak kiváló bemutatását adja Howard a kockázatpreferenciák tárgyalása során (Howard, 1977). Ramsey szubjektív valószínűségeket (nem pontosan hasznosságokat) vezetett le az ágens preferenciáiból; mostanában Savage és Jeffrey vitt végbe hasonló konstrukciót (Savage, 1954; Jeffrey, 1983). Von Winterfeldt és Edwards modernebb szempontból ismerteti a döntéselemzést és annak az emberi preferenciastruktúrákkal való kapcsolatát (Von Winterfeldt és Edwards, 1986). A mikrohalál hasznosságmértéket Howard elemzi (Howard, 1989). Az *Economist* egy 1994-es felmérése egy emberi élet értékét 750 000 és 2,6 millió dollár közzé teszi. Azonban Richard Thaler irracionális variánsokat talált arra nézve, hogy mennyit hajlandó valaki fizetni, hogy elkerüljön egy halálos kockázatot, ahhoz képest, hogy mennyiért hajlandó felvállalni ugyanazt a kockázatot. 1/1000 esély esetén a válaszadó maximum 200 dollárt fizetne, hogy elkerülje a kockázatot, míg 50,000 dollárért sem hajlandó felvállalni ezt a kockázatot (Thaler, 1992).

A QALY-t sokkal szélesebb körben használták orvosi és társadalmi döntések meghozatalánál, mint a mikrohalált, lásd (Russell, 1990), ami jellemző példája azoknak az érveléseknek, hogy a QALY-t használó várható hasznosság növekedésének alapján a közegészségügyben alapos változtatásokra van szükség.

A Decisions with Multiple Objectives: Preferences and Value Tradeoffs (Keeney és Raiffa, 1976) c. könyv részletes bevezetést ad a többattribútumú hasznosságelméletbe. Ez olyan módszerek korai számítógépes megvalósítását írja le, amelyekkel a többattribútumú hasznosságfüggvényhez a szükséges paraméterek kikérdezhetők, és hosszasan tárgyalja az elmélet valós alkalmazásait. Az MI-n belül a legelső hivatkozás az MVH elvre Wellman cikke volt (Wellman, 1985), amely egy URP-nek (Utility Reasoning Package) nevezett

Created by XMLmind XSL-FO Converter.

rendszert tartalmaz, ami képes kezelni a preferenciák függetlenségéről és feltételes függetlenségéről szóló állításokat, hogy elemezze a döntési probléma struktúráját. A sztochasztikus dominanciát a kvalitatív valószínűségi modellel Wellman vizsgálta alaposan (Wellman, 1988; 1990a). Wellman és Doyle egy előzetes vázlatát adja annak, hogy hogyan lehet felhasználni hasznosság-függetlenség relációk komplex halmazát a hasznosságfüggvény strukturált modelljének megalkotásához, hasonlóan ahhoz, ahogyan a valószínűségi háló az együttes valószínűségeloszlás-függvény strukturált modelljét adja (Wellman és Doyle, 1992). Bacchus és Grove, valamint La Mura és Shoham további eredményeket közöl ezzel kapcsolatban (Bacchus és Grove, 1995; 1996; La Mura és Shoham, 1999).

A döntéselmélet már az 1950-es évek óta szabványos eszköz a közgazdaságtanban, a pénzügyben és a menedzsment tudományban. Az 1980-as évekig az egyszerű döntési szituációk reprezentálására a döntési fák voltak a fő eszközök. A döntési hálókat vagy hatásdiagramokat Howard és Matheson vezette be (Howard és Matheson, 1984), bár a szükséges fogalmakat már korábban kifejlesztette egy csoport (benne Howard és Matheson) az SRI-nél (Miller és társai, 1976). Howard és Matheson eljárása azonban egy közbenső lépést, a döntési hálónak egy döntési fára történő átírását tartalmazta, ahol általános esetben a fa exponenciális méretű volt. Shachter kifejlesztett egy eljárást, ami közvetlenül a döntési hálókon alapult anélkül, hogy egy közbeeső döntési fát hozna létre (Shachter, 1986). Ez az algoritmus volt az első, amely teljes következtetést tudott elvégezni többszörösen összekötött Bayes-hálókban. Egy mostani munkában Nilsson és Lauritzen összekapcsolta a döntési hálókhoz tartozó algoritmusokat a Bayes-hálóknál kifejlesztett klaszterezési algoritmusokkal (Nilsson és Lauritzen, 2000). Oliver és Smith gyűjteménye számos hasznos cikket tartalmaz a döntési hálókról, mint ahogy a *Networks* folyóirat 1990-es különkiadása is hasznos cikkekből áll (Oliver és Smith, 1990). Döntési hálókról és hasznosságmodellezésről szóló cikkek a *Management Science*-ben is rendszeresen megjelennek.

Az információérték-elméletet Ron Howard elemezte először (Howard, 1966). Ez a cikke a következő megjegyzéssel ér véget: „Ha az információérték-elmélet és a hozzá kapcsolódó döntéselméleti struktúrák a jövőben nem képezik igen nagy részét a mérnökök képzésének, akkor a mérnöki szakma azt fogja látni, hogy hagyományos szerepét, a tudományos és közgazdasági erőforrásoknak az emberiség előnyére történő kezelését, egy másik szakma elorozta.” A hivatkozott forradalom a vezetési módszerekben a mai napig nem következett be, bár ez változhat amint az információérték-elmélet felhasználása a bayesi szakértő rendszerekben elterjedtebbé válik.

Az orvosi döntéseknél történő korai alkalmazásoktól eltekintve meglepően kevés MI-kutató alkalmazta a 13. fejezetben tárgyalt döntéselméleti eszközöket. A kevés kivételek egyike Jerry Feldman volt, aki a döntéselméletet a látás problémájára (Feldman és Yakimovsky, 1974) és a tervkészítés problémájára (Feldman és Sproull, 1977) alkalmazta. Az 1980-as években aztán hirtelen újra megnőtt az érdeklődés az MI részéről a valószínűségi módszerek iránt, a döntéselméleti szakértő rendszerek széles körben elterjedtekké váltak (Horvitz és társai, 1988) – valójában 1991-től még az *Artificial Intelligence* fedőlapja is egy döntési hálót ábrázolt, bár élve művészi szabadságukkal egyes nyilak irányát megváltoztatták.

8.2. Feladatok

16.1.

(David Heckerman alapján, de a magyar kiadású Guinness Rekordok Könyvére alapozva [Guinness Rekordok Könyve 1991, Solaris Kft., Budapest].) Ez a feladat az **Almanac Game**-hez kapcsolódik, amit döntéselemzők használnak a numerikus becslések kalibrálására. Adjon becslést az alábbi problémákra, azaz azt a számot adja meg, amiről úgy gondolja, hogy ugyanolyan valószínűséggel lehet túl nagy is, mint túl kicsi. Próbálja meg eltalálni a 25 percentilises becslést is, azaz azt a számot, amiről úgy gondolja, hogy 25% esélye van, hogy túl nagy lesz, és 75% esélye, hogy túl kicsi. Ugyanígy tippelje meg a 75 percentilisest is. (Így minden egyes kérdésre három becslést kell megadnia összesen – egy kisebbet, a mediánt és egy nagyobb értéket.)

a. Melyik évben születtek meg az első (névadó) sziámi ikrek?

b. Hány helyi értékig képes valaki emlékezetből visszaidézni a π számot?

c. Hány rák volt az eddig megfigyelt legnagyobb krill rajban?

d. A kőkorszak óta – a háborúktól és a katasztrófáktól eltekintve – az elhalálozások hány százalékáért felelős a maláriaszúnyog?

Created by XMLmind XSL-FO Converter.

e. Az állatkertekben végzett mérések szerint egy kifejlett csimpánz hányszor erősebb egy átlagos normális férfinál?

f. Az eddigi legmagasabb felállított karácsonyfa magassága?

g. Milyen hosszú a Csendes-óceánt átszelő legrövidebb hajóút hossza?

h. A Föld átlagos sűrűsége hányszorosa a vízének?

i. Egy évben átlagosan hány esetben észlelnek szeizmikus tevékenységet, ebből mennyi érezheto, és mennyi okoz kárt?

j. Milyen magas a hőmérséklet a Holdon a legjobban napsütötte helyen (Egyenlítőre merőlegesen)?

A helyes válaszok a fejezet utolsó feladata után találhatók meg. A döntéselemzés szempontjából nem az az érdekes, hogy az ön mediántippjei milyen közel vannak a valódi válaszokhoz, hanem inkább az, hogy milyen gyakran van a valódi érték az ön által tippelt 25%-os és 75%-os határ között. Ha ez az esetek felénél így van, akkor a határai pontosak. De ha ön az emberek többségéhez hasonló, akkor a kelleténél magabiztosabb, és az esetek felénél kevesebbszer lesz az igazi érték az ön által tippelt határokon belül. Ezen gyakorlással „javíthat", és így hasznosabb információt nyújthat a döntéshozáshoz. Próbálja ki ezt a második kérdéssort, hogy lássa, bekövetkezett-e valamilyen javulás:

a. Becslések szerint Picasso hány képet, illetve vázlatot festett az élete során?

b. Hány statiszta szerepel Sir Richard Attenborough *Ghandi* c., 1982-ben készült filmjének temetési jelenetében?

c. Milyen magas a világ legmagasabb építménye (varsói rádiótorony)?

d. Milyen hosszú a transzszibériai vasútpálya?

e. Hány utas fordult meg az O'Hara chicagói repülőtéren 1988-ban?

f. A világ leglassabban mozgó mechanikus szerkezete (óra) hány év alatt tesz meg egy fordulatot?

g. A dobozos sör piacra dobásának éve?

h. A világ legnagyobb „tojásgyárában", a Croton Egg Farmon Amerikában a cég tyúkjai hány tojást tojnak naponta?

i. Milyen volt a népsűrűség 1968-ban Macaó portugál tartományban, a világ legsűrűbben lakott helyén?

j. A Man-szigeten ülésezik a megszakítás nélkül leghosszabb ideje fennálló törvényhozás. Az idén hány éves?

k. Mikor helyezték üzembe az első forgalomirányító jelzőlámpákat (Londonban)?

l. Mikor alakult a Nemzetközi Labdarúgó Szövetség (a FIFA)?

16.2.

Az állami szerencsejáték 1 dollárba kerül. Két lehetséges díj van: 10 dollár 1/50 valószínűséggel és 1 000 000 dollár 1/2 000 000 valószínűséggel. Mennyi a szerencsejátékjegy várható pénzügyi értéke? Mikor racionális (ha van ilyen helyzet egyáltalán) jegyet vásárolni? Pontosabban – mutassa ezt be egy hasznosságokat tartalmazó egyenlettel. Azt feltételezheti, hogy $U(10 \text{ dollár}) = 10 \times U(1 \text{ dollár})$, de $U(1\,000\,000 \text{ dollár})$-ra hasonlót nem lehet feltételezni. Szociológiai tanulmányok mutatják, hogy az alacsonyabb jövedelmű emberek aránytalan számú sorsjegyet vásárolnak. Mit gondol, ez azért van, mert rosszabb döntéshozók, vagy mert eltérő hasznosságfüggvényük van?

16.3.

J. Bernoulli 1738-ban vizsgálta a szentpétervári paradoxont, ami a következő. Lehetősége nyílik részt venni egy szerencsejátékban, amelyben egy szabályos érmét dobnak fel ismételten az első fej előfordulásáig. Ha az első fej az n-edik dobásnál következik be, akkor a nyereménye 2^n dollár.

Created by XMLmind XSL-FO Converter.

a. Mutassa meg, hogy ennek a játéknak a várható pénzügyi értéke végtelen.

b. Mennyit fizetne ön, személy szerint a részvételért?

c. Ezt a nyilvánvaló paradoxont Bernoulli azzal a javaslattal oldotta meg, hogy a pénz hasznossága logaritmikus skálát követ (azaz $U(S_n) = a + \log_2 n + b$, ahol S_n azt az állapotot jelöli, hogy n dollárunk van). Mi a játék várható hasznossága ezzel a feltevéssel?

d. Mi az a maximum összeg, amit racionális lenne kifizetni a részvételért, feltéve, hogy a kezdeti vagyona k dollár?

16.4.

Becsülje meg a saját hasznosságfüggvényét különböző pénznövekményekre. Ezt úgy teheti meg, hogy egy sor preferenciatesztet futtat egy meghatározott M_1 összeg és egy $[p, M_2; (1 - p), 0]$ szerencsejáték között. Válasszon különböző M_1 és M_2 értékeket, és változassa addig p-t, ameddig nem lesz ön számára közömbös, hogy melyiket választja a kettő közül. Ábrázolja a hasznosságfüggvényt.

16.5.

Írjon számítógépes programot, ami automatizálja a 16.4. feladat folyamatát. Próbálja ki a programot különböző hátterű és politikai irányultságú embereken. Elemezze eredményei konzisztenciáját az összes személy esetén és egyetlen személy esetén is.

16.6.

Mennyit ér egy mikrohalál önnek? Fejlesszen ki protokollt ennek meghatározására. Fogalmazzon meg kérdéseket abból a szempontból, hogy mennyit ér a kockázat elkerülése, és abból a szempontból is, hogy mennyiért vállalja a kockázatot.

16.7.

Mutassa meg, hogyha X_1 és X_2 preferenciálisan független X_3-tól, és X_3 és X_2 preferenciálisan független X_1-től, akkor szükségszerűen X_1 és X_3 is preferenciálisan független X_2-től.

16.8.

Ez a feladat a 16.5. ábrán bemutatott repülőtéri elhelyezési probléma elemzését fejezi be.

a. Adjon meg ésszerű, a témakörhöz kapcsolódó változókat, valószínűségeket és hasznosságokat a háló számára, feltételezve, hogy három helyszín van.

b. Oldja meg a döntési problémát.

c. Mi történik, ha a műszaki fejlődés következtében a repülőgépek csak fele akkora zajt okoznak?

d. És ha a zaj elkerülése háromszor fontosabbá válik?

e. Számolja ki a *TIÉ* mennyiséget a *LégiForgalom*, a *Pereskedés* és az *Építkezés* esetén a modellben.

16.9.

Ismételje meg a 16.8. feladatot, felhasználva a 16.6. ábrán látható cselekvéshasznosság-reprezentációt.

16.10.

Melyik feltételes valószínűségi tábla bejegyzésre a legérzékenyebb a hasznosság a 16.8. és a 16.9. feladatban konstruált repülőtéri elhelyezési diagramok esetén, ha az elérhető tényeket ismerjük.

16.11.

(Judea Pearltől átvett feladat [Pearl, 1988].) Egy használt autót vásárló eltérő költségek mellett különböző teszteket végeztethet (például megrugdossa a gumikat, autószerelőhöz viszi a kocsit), majd a tesztek eredménye

Created by XMLmind XSL-FO Converter.

alapján eldönti, hogy melyik autót vegye meg. Feltételezzük, hogy a vásárló akkor veszi meg a c_1 autót, ha elég idő van legalább egy teszt elvégzésére, továbbá, hogy c_1-nek t_1 a tesztje, és ez 50 dollárba kerül.

A kocsi jó állapotban (q^+ minőség) vagy rossz állapotban (q^- minőség) lehet, és a tesztek segíthetnek a kocsi állapotának meghatározásában. A c_1 kocsi 1500 dollárba kerül, piaci értéke azonban 2000 dollár, ha jó állapotban van; ha nem, akkor 700 dollár szükséges a jó állapotba hozásához. A vásárló úgy becsüli, hogy 70% esélye van, hogy a kocsi jó állapotban van.

a. Rajzolja fel a döntési hálót, ami ezt a problémát reprezentálja.

b. Számítsa ki a várható nettó nyereséget c_1 megvásárlása esetén, ha nincsenek tesztek.

c. A tesztek azzal a valószínűséggel jellemezhetők, hogy a kocsi átmegy-e a teszten vagy elbukik, attól függően, hogy jó vagy rossz állapotban van. A következő információkkal rendelkezünk:

$$P(Átmegy(c_1, t_1) | q^+(c_1)) = 0,8$$

$$P(Átmegy(c_1, t_1) | q^-(c_1)) = 0,35$$

Felhasználva a Bayes-tételt, számítsa ki annak valószínűségét, hogy a kocsi átmegy (vagy elbukik) a teszten, és így azt a valószínűséget, hogy jó vagy rossz állapotban van adott tesztkimenetel esetén.

d. Számítsa ki az optimális döntést, ha átmegy, illetve ha elbukik, és számítsa ki a várható hasznosságukat.

e. Számítsa ki a teszt információértékét, és vezesse le a vásárló optimális tervét.

16.12.

Bizonyítsa be, hogy az információ értéke nem negatív és sorrendfüggetlen, amint azt a 16.6. alfejezetben állítottuk. Magyarázza el, hogyan lehetséges, hogy valaki rosszabb döntést hoz az információ megkapása után, mint amit a megkapása előtt hozott volna.

16.13.

Módosítsa és egészítse ki a Bayes-hálós kódot a kódtárban, hogy képes legyen döntési hálókat létrehozni és kiértékelni, illetve kiszámítani az információ értékét.

A 16.1. feladat válaszai: Első sorozat: 1811, 40 ezer, 10 millió, 50%, 5-6, 67,36 m, 17 550 km, 5,515, 500 000, 100 000, 1000, 117,2 °C. Második sorozat: 13 500, 300 000, 646,28 m, 9438 km, 56 281 000, 25 753, 1935, 3,7 millió, 28 343 fő/km^2, 1020, 1868, 1904.

Created by XMLmind XSL-FO Converter.

17. fejezet - Komplex döntések meghozatala

Ebben a fejezetben olyan módszereket vizsgálunk meg, amelyekkel eldönthetjük, hogy mit tegyünk a mai napon, feltételezve, hogy holnap is lehetőségünk lesz a cselekvésre.

Ebben a fejezetben a döntéshozatalhoz kapcsolódó számítási kérdéseket tárgyaljuk meg. Míg a 16. fejezetben egyszeri vagy epizodikus döntési problémákkal foglalkoztunk, ahol az egyes cselekvések kimeneteleinek a hasznossága jól ismert volt, a 17. fejezetben **szekvenciális döntési problémá**kkal (**sequential decision problem**s) fogunk foglalkozni, ahol az ágens hasznossága a döntések sorozatától függ. A szekvenciális döntési problémák, amelyek hasznosságot, bizonytalanságot és érzékelést is magukban foglalnak, a keresés és tervezés problémáját általánosítják, melyeket a II., illetve a IV. részben írtunk le. A 17.1. alfejezet kifejti, hogyan definiálhatunk szekvenciális döntési problémákat, a 17.2. és a 17.3. alfejezet elmagyarázza ezek megoldását, hogy optimális viselkedést eredményezzenek, amely kiegyensúlyozza a bizonytalan környezetben meghozott cselekvések kockázatát és jutalmát. A 17.4. alfejezet kiterjeszti ezeket az ötleteket a részlegesen megfigyelhető környezetekre, és a 17.5. alfejezet egy teljes tervezési módot fejleszt ki részlegesen megfigyelhető környezetben lévő döntéselméleti ágensek tervezésére, összekapcsolva a dinamikus Bayes-hálókat a 15. fejezetből a 16. fejezetbeli döntési hálókkal.

A fejezet második része többágenses környezeteket tárgyal. Ilyen környezetekben az optimális viselkedés fogalma sokkal bonyolultabbá válik az ágensek közötti interakció miatt. A 17.6. alfejezet ismerteti a **játékelmélet** (**game theory**) alapötleteit, ideértve azt az elképzelést is, hogy egy racionális ágensnek lehet, hogy véletlenszerűen kell viselkednie. A 17.7. alfejezet azt vizsgálja meg, hogyan tervezhetők olyan többágenses rendszerek, amelyekben az ágensek egy közös célt tudnak megvalósítani.

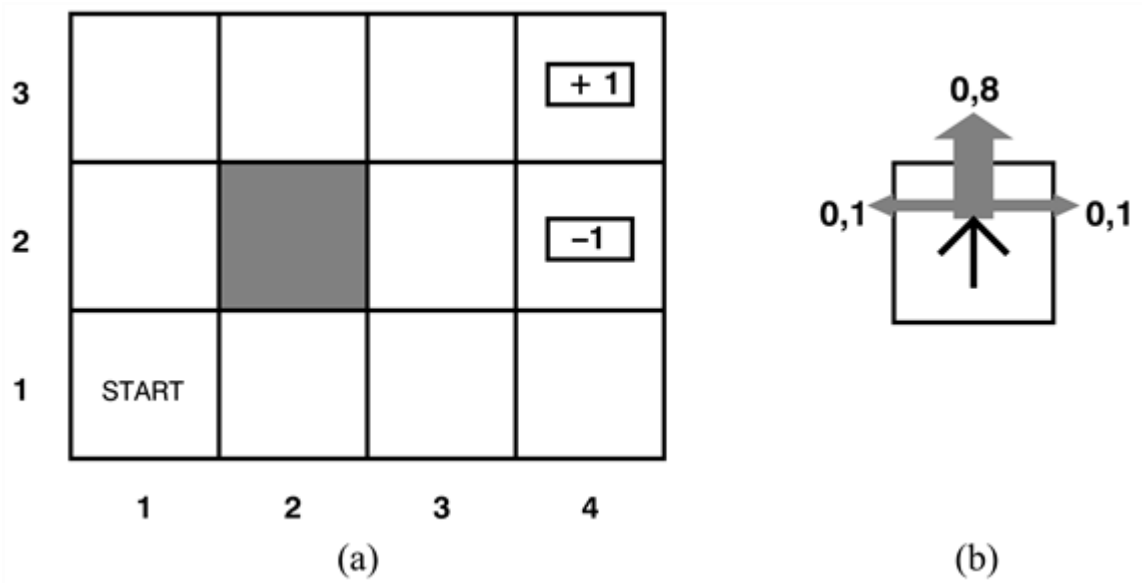
1. Szekvenciális döntési problémák

1.1. Egy példa

Tételezzük fel, hogy egy ágens a 17.1. (a) ábrán látható 4 × 3-as környezetben helyezkedik el. A kezdő állapotból indulva minden időpontban választania kell egy cselekvést. A környezettel való interakció leáll, ha az ágens eléri a +1-gyel vagy a –1-gyel jelölt állapotot. Az egyes helyzetekben az elérhető cselekvések nevei: *Fel*, *Le*, *Balra*, *Jobbra*. Egyelőre feltételezzük, hogy a környezet **teljesen megfigyelhető** (**fully observable**), azaz az ágens mindig ismeri a helyzetét.

17.1. ábra - (a) Egy egyszerű 4 × 3-as környezet, ami az ágens számára szekvenciális döntési probléma. (b) A környezet állapotátmenet-modelljének illusztrálása: a „szándékolt” kimenetel 0,8 valószínűséggel következik be, de 0,2 valószínűséggel az ágens oldalra mozdul a szándékolt irányhoz képest. A fallal való ütközéskor nincsen mozgás. A két végállapot jutalma a jelzett +1 és –1, az összes többi állapot jutalma –0,04.

Created by XMLmind XSL-FO Converter.



Ha a probléma determinisztikus volna, a megoldás könnyű lenne: [*Fel, Fel, Jobbra, Jobbra, Jobbra*]. Sajnos a környezet nem maradna mindig szinkronban ezzel a megoldással, mivel a cselekvések megbízhatatlanok. A véletlenszerű mozgás egy általunk elfogadott modelljét a 17.1. (b) ábra mutatja be. Minden cselekvés 0,8 valószínűséggel éri el a kívánt hatását, de a maradék esetben a cselekvések az ágenst a kívánt iránytól jobbra mozgatják. Továbbá, ha az ágens falba ütközik, akkor ugyanazon a mezőn marad. Például ha az (1, 1) kezdő négyzeten áll, a *Fel* cselekvés az ágenst az (1, 2)-re mozgatja 0,8 valószínűséggel, de 0,1 valószínűséggel a (2, 1)-re kerül, és 0,1 valószínűséggel balra megy, ahol is falnak ütközik és (1, 1)-en marad. Egy ilyen környezetben a [*Fel, Fel, Jobbra, Jobbra, Jobbra*] sorozat $0{,}8^5 = 0{,}32768$ valószínűséggel kerüli meg az akadályokat, és éri el a (4, 3) célállapotot. Igen kis eséllyel az is megtörténhet, hogy a célt a másik úton keresztül éri el, $0{,}1^4 \times 0{,}8$, ami összességében 0,32776 valószínűséget jelent (lásd 17.1. feladat).

Az egyes állapotokban végrehajtott egyes akcióknak a kimeneti valószínűségeit **állapotátmenet-modell**nek (**transition model**) nevezzük, (vagy csak „modell”-nek, ha nem értelemzavaró). A $T(s, a, s')$ jelölést fogjuk használni annak a valószínűségnek a jelölésére, hogy az s állapotban az a cselekvés végrehajtása s' állapotot eredményez. Feltesszük, hogy az átmenetek teljesítik a 15. fejezetben megfogalmazott **Markov-tulajdonság**ot, azaz, hogy s' elérése s-ből csak s-től függ, más korábbi állapotoktól már nem. Egyelőre a $T(s, a, s')$-t tekinthetjük a valószínűségek egy nagy háromdimenziós táblázatának. Később, a 17.5. alfejezetben látni fogjuk, hogy az állapotátmenet-modell reprezentálható **dinamikus Bayes-háló**kkal (**dynamic Bayesian network**), csakúgy mint a 15. fejezetben.

Hogy teljessé tegyük a feladat környezetének a definiálását, meg kell adnunk az ágens hasznosságfüggvényét. Mivel a döntési probléma szekvenciális, a hasznosságfüggvény az állapotok sorozatától – a **környezeti történet**től (**environment history**) – fog függni, nem pedig egyetlen állapottól. A fejezetben később megvizsgáljuk az ilyen hasznosságfüggvények általános megadását; egyelőre egyszerűen kikötjük, hogy az ágens minden s állapotban egy $R(s)$ **jutalmat** (**reward**) kap, ami lehet pozitív vagy negatív, de mindenképpen korlátos. A konkrét példánkban a jutalom minden állapotban –0,04, kivéve a végállapotokat (ahol +1 vagy –1). Egy környezeti történet hasznossága (egyelőre) egyszerűen a kapott jutalmak *összege*. Például ha az ágens a +1 állapotot 10 lépés után éri el, akkor az összhasznossága 0,6 lesz. A –0,04 negatív jutalom arra ösztönzi az ágenst, hogy minél gyorsabban érje el a (4, 3)-at, így ez a probléma a 3. fejezetbeli keresési problémák egy sztochasztikus általánosítása. Ez más megfogalmazásban úgy hangzik, hogy az ágens nem élvezi az életet ebben a környezetben, és olyan gyorsan ki akar kerülni, amilyen gyorsan csak lehet.

Egy teljesen megfigyelhető környezetben megadott szekvenciális döntési problémát Markov-állapotátmenet-modelljével és additív jutalmakkal **Markov döntési folyamat**nak neveznek (**MDF**) (**Markov decision process**). Egy MDF-et a következő három összetevő határoz meg:

Kezdőállapot: S_0

Állapotátmenet-modell: $T(s, a, s')$

Created by XMLmind XSL-FO Converter.

Jutalomfüggvény:[1] $R(s)$

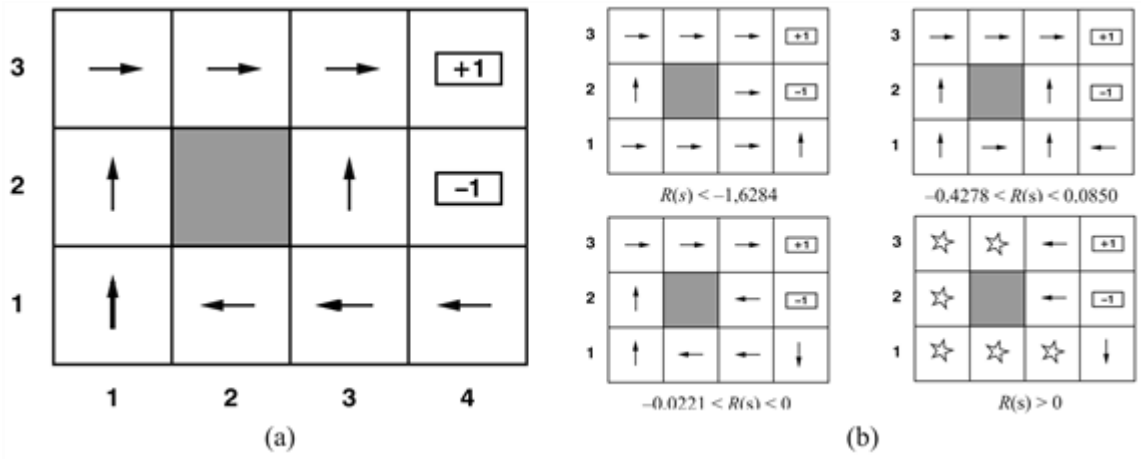
A következő kérdés az, hogyan néz ki egy megoldás a problémára? Azt már láttuk, hogy egy rögzített cselekvéssorozat nem oldja meg a problémát, mivel az ágens végül más állapotba jut a célállapot helyett. Ezért egy megoldásnak minden, az ágens által elérhető állapotra elő kell írnia, hogy az ágens mit tegyen. Egy ilyenfajta megoldás az **eljárásmód** (**policy**). Egy eljárásmódot általában π-vel jelölünk, és a π eljárásmód az s állapotban a $\pi(s)$ cselekvést javasolja. Ha az ágens egy teljes eljárásmóddal rendelkezik, akkor függetlenül attól, hogy egy cselekvésnek mi is a kimenetele, az ágens mindig tudni fogja, mit tegyen a következő alkalommal.

Egy adott eljárásmód kezdőállapotból induló végrehajtása minden alkalommal egy különböző környezeti történetet eredményez a környezet sztochasztikus volta miatt. Egy eljárásmód minőségét ezért az adott eljárásmód generálta lehetséges környezeti történetek várható hasznosságával mérik. Az **optimális eljárásmód** (**optimal policy**) az az eljárásmód, ami a legnagyobb várható hasznosságot eredményezi. Az optimális eljárásmódot π^*-gal jelöljük. A π^* ismeretében az ágens úgy dönti el, hogy mit tegyen, hogy figyelembe veszi az aktuális érzékelését, ami az aktuális s állapotot közli vele, majd végrehajtja a $\pi^*(s)$ cselekvést. Az eljárásmód az ágens függvényét explicit módon reprezentálja, és ezért az voltaképpen egy egyszerű reflexágens leírása, amit egy hasznosságalapú ágens által használt információkból számítunk ki.

A 17.1. ábrán mutatott világ optimális eljárását a 17.2. (a) ábra mutatja. Látható, hogy mivel egy lépés költsége meglehetősen kicsi ahhoz a büntetéshez viszonyítva, hogy véletlenül a (4, 2) pozícióra kerülünk, az optimális eljárásmód a (3, 1) állapot esetén óvatos. Inkább egy hosszú kerülő út megtételét javasolja, mint a rövid átvágást a (4, 2) kockáztatásával.

A kockázat és a jutalom egyensúlyának megváltozása függ az $R(s)$ értékétől a nem végső állapotoknál. A 17.2. (b) ábrán az $R(s)$ értékének négy különböző intervallumához tartozó optimális eljárásmódot találhatunk. Ha $R(s) \leq -1{,}6284$, akkor az élet anynyira elviselhetetlen, hogy az ágens egyenesen a legközelebbi kijárathoz tart, annak ellenére, hogy ez –1 értékű. Amikor $-0{,}4278 \leq R(s) \leq -0{,}0850$, az élet elég kellemetlen; az ágens a +1 állapothoz vezető utat választja, vállalva annak kockázatát, hogy esetleg a –1 állapotba kerül. Nevezetesen az ágens a (3, 1)-ben választja az átvágást. Amikor az élet csak kevéssé bánatos ($-0{,}0221 < R(s) < 0$), az optimális eljárásmód nem vállal *semmilyen kockázatot*. A (4, 1) és (3, 2) állapotokban az ágens teljesen elkerüli a –1 állapotot, így nem tud véletlenül beleesni, még ha ez azt is jelenti, hogy elég sokszor beveri a fejét a falba. Végül, ha $R(s) > 0$, akkor az élet kifejezetten élvezhető, és az ágens *mindkét* kijáratot elkerüli. Mindaddig, amíg a (4, 1), (3, 2) és (3, 3) állapotokban a cselekvések azok, amelyek az ábrán láthatók, minden eljárásmód optimális, és az ágens végtelen teljes jutalmat ér el, mivel soha nem lép be a végállapotba. Meglepő módon megmutatható, hogy hat más optimális eljárásmód is létezik az $R(s)$ különböző intervallumaira. A 17.7. feladat kéri majd ezek megkeresését.

17.2. ábra - (a) Egy optimális eljárásmód a sztochasztikus környezetre $R(s) = -0{,}04$ esetén nem végállapotoknál. (b) Optimális eljárásmódok $R(s)$ négy különböző értéktartománya esetén.



[1] Az MDF egyes definíciói megengedik, hogy a jutalom függjön a cselekvéstől és a kimeneteltől is, így a jutalomfüggvény $R(s, a, s')$ alakú. Ez néhány környezet leírását egyszerűsíti, de a problémát alapvetően nem változtatja meg.

Created by XMLmind XSL-FO Converter.

A kockázat és a jutalom óvatos egyensúlyozása az MDF-ek egy olyan meghatározó tulajdonsága, ami nem jelentkezik determinisztikus keresési problémáknál; másfelől ez meghatározó tulajdonsága számos valósvilág-beli döntési problémának. Emiatt az MDF-eket számos tudományterületen tanulmányozzák, ideértve az MI-t, az operációkutatást, a közgazdaságtant és a szabályozáselméletet. Algoritmusok tucatjait javasolták optimális eljárásmódok kiszámítására. A 17.2. és 17.3. alfejezetben leírunk a legfontosabb algoritmuscsaládok közül kettőt. Először azonban be kell fejeznünk a hasznosságok és eljárásmódok vizsgálatát a szekvenciális döntési problémák esetében.

1.2. Optimalitás szekvenciális döntési problémákban

A 17.1. ábrán látható MDF-példában az ágens teljesítményét a meglátogatott állapotokban kapott jutalmak összege mérte. Ez a teljesítménymérték nem önkényes, de nem is az egyetlen lehetőség. Ez az alfejezet a lehetséges alternatív teljesítménymértékeket vizsgálja – azaz alternatív hasznosságfüggvényeket a környezeti történéseken, amit úgy jelölünk, hogy $U_h([s_0, ..., s_n])$. Az alfejezet a 16. fejezetből származó ötleteken alapul, és a technikai részleteket is bemutatja; a főbb pontokat a végén összegezzük.

Fontos

Az első megválaszolandó kérdés az, hogy **véges horizont** (**finite horizon**) vagy **végtelen horizont** (**infinite horizon**) van a döntéshozatalnál. A véges horizont azt jelenti, hogy létezik egy *rögzített N* idő, ami után semmi nem érdekes – a játéknak vége, mondhatni. Így $U_h([s_0, ..., s_{N+k}]) = U_h([s_0, ..., s_N])$ minden $k > 0$ esetén. Például tegyük fel, hogy az ágens (3, 1)-ből indul a 17.1. ábra 4 × 3-as világában, és tegyük fel, hogy $N = 3$. Ekkor, hogy a +1 állapot elérésének legalább az esélye meglegyen, az ágensnek egyenesen felé kell tartani, és az optimális cselekvés a *Fel*. Ezzel szemben, ha $N = 100$, akkor bőségesen van idő a biztonságos utat követni *Balra* menve. Azaz, *véges horizont esetében az optimális cselekvés egy adott állapotban idővel változhat*. Azt mondjuk, hogy az optimális eljárásmód véges horizont esetében **nem- stacionárius** (**nonstationary**). Rögzített időkorlát hiányában ezzel szemben nincs ok különböző viselkedésre ugyanabban az állapotban más és más időpontokban. Így az optimális cselekvés csak az aktuális állapottól függ, és az optimális eljárásmód **stacionárius** (**stationary**). A végtelen horizontú esethez tartozó eljárásmódok ezért egyszerűbbek, mint a véges horizontú esethez tartozók, és ebben a fejezetben mi főként a végtelen horizontú esettel foglalkozunk.[2] Vegyük észre, hogy a „végtelen horizont" nem jelenti szükségszerűen azt, hogy az összes állapotsorozat végtelen; mindössze annyit jelent, hogy nincs egy rögzített határ. Nevezetesen, egy végtelen horizontú MDF-ben lehetnek végállapotot tartalmazó véges állapotsorozatok.

A következő eldöntendő kérdés az állapotsorozatok hasznosságának a kiszámítása. Tekinthetjük ezt a kérdést a **többattribútumú hasznosságelmélet** (**multiattribute utility theory**) egy kérdésének (lásd 16.4. alfejezet) azzal, hogy minden s_i állapotot az $[s_0, s_1, s_2, ...]$ állapotsorozat egy attribútumának veszünk. Egy egyszerű kifejezés eléréséhez, ami az attribútumokból épül fel, fel kell tételeznünk valamilyen preferenciafüggetlenséget. A legtermészetesebb feltevés az, hogy az ágens preferenciái az állapotsorozatok között **stacionárius**ok (**stationary**). A preferenciákra vonatkozó stacionaritás a következőket jelenti: ha két állapotsorozat, $[s_0, s_1, s_2, \ldots]$ és $[s'_0, s'_1, s'_2, \ldots]$, ugyanazzal az állapottal kezdődik (például $s_0 = s'_0$), akkor a két sorozat a preferencia-sorrendjének ugyanannak kell lennie, mint az $[s_1, s_2, \ldots]$ és az $[s'_1, s'_2, \ldots]$ sorozatoknak. Ez tehát azt jelenti, hogy ha egy holnaptól kezdődő lehetséges jövőbeli eseménysort egy másik eseménysorral szemben preferálunk, akkor ez nem változhat, ha az eseménysor a mai naptól kezdődne. A stacionaritás egy meglehetősen ártalmatlannak tűnő feltevés, nagyon erős következményekkel: bizonyítható, hogy stacionaritás esetén a sorozatokhoz csak kétféle módon rendelhetők hasznosságok:

1. **Additív jutalmak** (**additive reward**s): egy állapotsorozat hasznossága ekkor

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \ldots$$

A 17.1. ábra 4 × 3-as világa additív jutalmakat használ. Vegyük észre, hogy az additivitást hallgatólagosan kihasználtuk az utak költségfüggvényeinek felhasználásakor a heurisztikus keresési algoritmusokban (4. fejezet).

2. **Leszámított jutalmak** (**discounted reward**s): egy állapotsorozat hasznossága ekkor

[2] Ez a teljesen megfigyelhető környezetekre igaz. Látni fogjuk, hogy a részlegesen megfigyelhető környezetekben a végtelen horizont esete nem annyira egyszerű.

Created by XMLmind XSL-FO Converter.

$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$

ahol a γ **leszámítolási tényező** (**discount factor**) egy 0 és 1 közötti szám. A leszámítolási tényező fejezi ki egy ágens preferenciáját a jelenlegi és a jövőbeli jutalmak között. Amikor γ közel 0, akkor a távoli jövőbeli jutalmakat jelentéktelennek tekinti. Amikor γ értéke 1, akkor a leszámított jutalmak pontosan megegyeznek az additív jutalmakkal, így az additív jutalmak a leszámított jutalmak speciális esetei. A leszámítolás jó modellnek tűnik mind az állati, mind az emberi időbeli preferenciákra. Egy γ leszámítolási tényező ekvivalens egy $(1/\gamma) - 1$ kamatlábbal.

A fejezet hátralevő részében leszámított jutalmakat fogunk feltételezni, aminek okai hamarosan világosak lesznek, bár néha megengedjük a $\gamma = 1$ értéket.

A végtelen horizont elfogadása mögött azonban rejlik egy probléma: ha a környezet nem tartalmaz egy végállapotot, vagy ha az ágens soha nem jut végállapotba, akkor az összes környezettörténet végtelen hosszú lesz, és a hasznosságok additív jutalmakkal általában végtelenek lesznek. Abban egyetérthetünk, hogy a $+\infty$ jobb, mint a $-\infty$, de két $+\infty$ hasznosságú állapotsorozat összehasonlítása már bonyolultabb. Három megoldás kínálkozik, amelyek közül kettőt már láttunk:

1. Leszámított jutalmakkal egy végtelen sorozat hasznossága *véges*. Valójában, ha a jutalmakra létezik egy $R_{\max}$ korlát és $\gamma < 1$, akkor azt kapjuk, hogy

$$U_h\left([s_0, s_1, s_2, \ldots]\right) = \sum_{t=0}^{\infty} \gamma^t R_{(s_t)} \le \sum_{t=0}^{\infty} \gamma^t R_{\max} = R_{\max} / (1-\gamma) \qquad (17.1)$$

felhasználva a végtelen mértani sorozat összegképletét.

2. Ha a környezet tartalmaz végállapotokat, *és ha garantált, hogy az ágens végül bekerül az egyikbe,* akkor soha nem lesz szükségünk végtelen sorozatok összehasonlítására. Egy eljárásmódot, ami garantáltan végállapotba juttat, **véges eljárásmód**nak (**proper policy**) nevezünk. Véges eljárásmódoknál használhatjuk a $\gamma = 1$-t (azaz az additív jutalmakat). A 17.2. (b) ábrán látható első három eljárásmód véges, a negyedik azonban nem. Egy nem véges eljárásmód végtelen teljes jutalmat ér el a végállapotoktól való távolmaradással, amikor a nem végállapotok jutalma pozitív. A nem véges eljárásmódokra additív jutalmaknál nem mindig működnek az MDF-eket megoldó alapalgoritmusok, ami jó ok a leszámított jutalmak használatára.

3. Végtelen sorozatok összehasonlítására egy másik lehetőség az időegységenkénti **átlagjutalom** (**average reward**) felhasználása. Tegyük fel, hogy a 4 × 3-as világban az (1, 1) mező jutalma 0,1, míg más nem végállapotoké 0,01. Ekkor egy olyan eljárásmódnak, ami minden tőle telhetőt megtesz, hogy az (1, 1)-ben maradjon, nagyobb lesz az átlagos jutalma, mint annak, amelyik máshol marad. Az átlagos jutalom hasznos kritérium bizonyos problémákra, de az átlagjutalomra tervezett algoritmusok meghaladják e könyv kereteit.

Összegezve, a leszámított jutalmak használata jár a legkevesebb bonyodalommal az állapotsorozatok kiértékelésében. A végső lépés annak megmutatása, hogyan válasszunk az eljárásmódok között, észben tartva azt, hogy egy adott π eljárásmód nem egyetlen állapotsorozatot generál, hanem a legkülönfélébb lehetséges állapotsorozatokat, mindegyiket egy adott valószínűséggel, amit a környezet állapotátmenet-modellje határoz meg. Így egy eljárásmód értéke a (sorozatonként) elért leszámított jutalmak *várható* értéke, ahol a várható érték képzése az összes állapotsorozat felett történik, ami az eljárás végrehajtása esetén előfordulhat. Egy π^* optimális eljárásmód ekkor

$$\pi^* = \arg\max_{\pi} E\left[\sum_{t=0}^{\infty} R(s_t) \middle| \pi\right] \qquad (17.2)$$

A következő két alfejezet algoritmusokat ír le az optimális eljárásmódok megtalálására.

2. Értékiteráció

Ebben az alfejezetben az optimális eljárásmód kiszámítására mutatunk be egy algoritmust, az úgynevezett **értékiteráció**t (**value iteration**). Az alapötlet az, hogy kiszámítjuk minden egyes *állapot* hasznosságát, majd az állapothasznosságokat felhasználjuk az egyes állapotoknál az optimális cselekvés megválasztásához.

Created by XMLmind XSL-FO Converter.

2.1. Az állapotok hasznossága

Az állapotok hasznosságát az állapotsorozatok hasznosságán keresztül definiáljuk. Nagyjából azt mondhatjuk, hogy egy állapot hasznossága a belőle kiinduló állapotsorozatok várható hasznosságával egyenlő. Nyilvánvalóan az állapotsorozatok függnek a végrehajtott eljárásmódtól, így elsőként egy adott π eljárásmódra definiáljuk a hasznosságot, $U^{\pi}(s)$-t. Jelöljük s_t-vel az ágens állapotát a π eljárásmód *t* lépésnyi végrehajtása után, ekkor azt kapjuk, hogy

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \middle| \pi, s_0 = s\right] \qquad (17.3)$$

Felhasználva ezt a definíciót, egy állapot valódi hasznossága az $U^{\pi}(s)$, amit $U(s)$-sel jelölünk. Ez a a leszámított jutalmak várható értéke akkor, amikor az ágens egy optimális eljárásmódot hajt végre. Vegyük észre, hogy az $U(s)$ és az $R(s)$ igen eltérő mennyiségek: az $R(s)$ a „rövid távú" jutalom az *s*-ben tartózkodásért, míg az $U(s)$ a „hosszú távú" összjutalom *s*-től kezdve. A 17.3. ábrán láthatók a hasznosságok a 4 × 3-as világban. Látható, hogy a hasznosságok nagyobbak a +1-es kijárathoz közeli állapotoknál, mivel kevesebb lépés szükséges a kijárat eléréséhez.

Az $U(s)$ hasznosságfüggvény lehetővé teszi az ágensnek, hogy a cselekvéseit a 16. fejezetben szereplő maximális várható hasznosság elve alapján válassza meg; azt a cselekvést választja, amelyik maximalizálja a bekövetkező állapot várható hasznosságát:

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')U(s') \qquad (17.4)$$

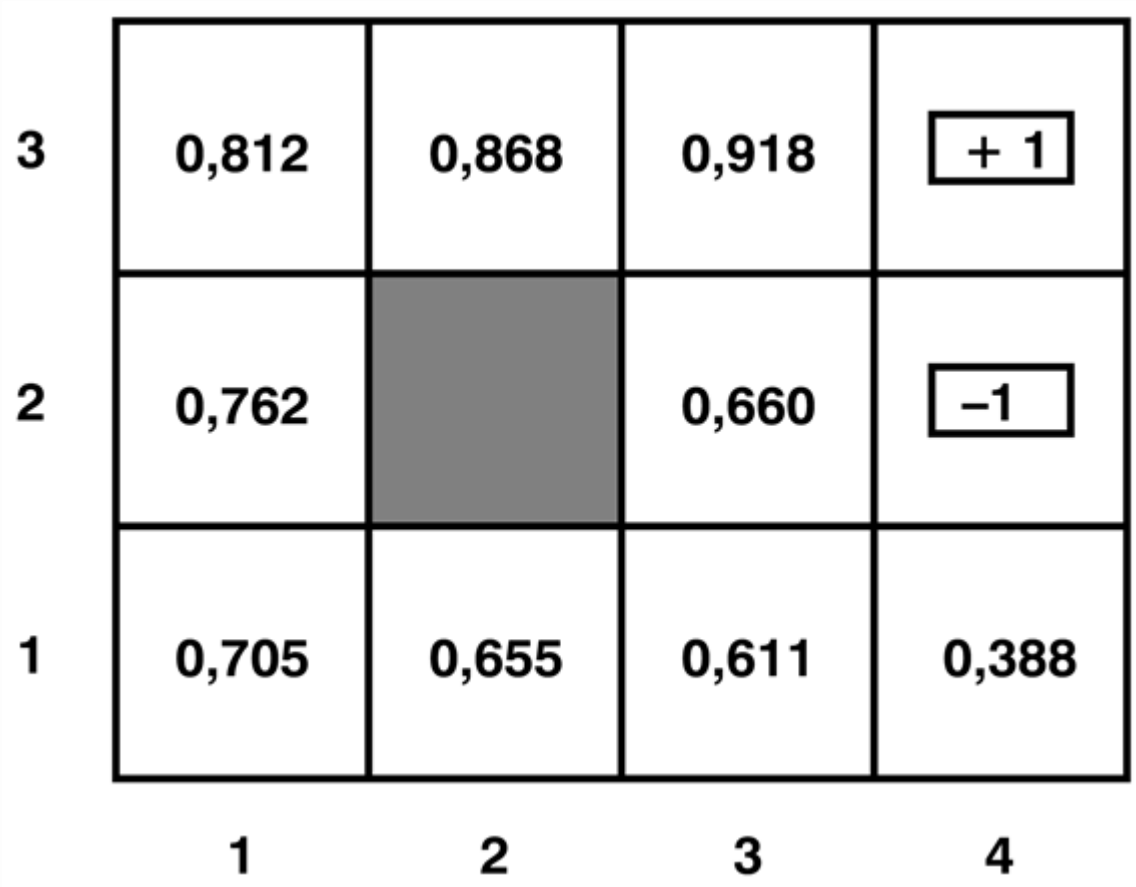
Fontos

Most azonban, ha egy állapot hasznossága a leszámított jutalmak várható értékének összege attól a kiindulóponttól kezdve, akkor közvetlen kapcsolat áll fenn egy állapot hasznossága és a szomszédjainak a hasznossága között: *egy állapot hasznossága az állapotban tartózkodás közvetlen jutalmának és a következő állapot várható leszámított hasznosságának az összege, feltéve, hogy az ágens az optimális cselekvést választja*. Azaz egy állapot hasznossága a következő:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U(s') \qquad (17.5)$$

17.2. ábra - Az állapotok hasznosságai a 4 × 3-as világban, $\gamma = 1$ és a nem végállapotoknál $R(s) = -0{,}04$ esetén

Created by XMLmind XSL-FO Converter.



A (17.5) egyenletet **Bellman-egyenlet**nek (**Bellman equation**) nevezik Richard Bellman tiszteletére (Bellman, 1957). Az állapotok hasznosságai – mint a bekövetkező állapotsorozatok várható hasznossága a (17.3) egyenlet szerint – a Bellman-egyenletek egy rendszerének a megoldásai. Valójában ezek egyértelmű megoldások, ahogyan a következő két alfejezetben megmutatjuk.

Nézzük meg a 4 × 3-as világ Bellman-egyenleteinek egyikét. Az (1, 1) állapothoz tartozó egyenlet:

$$U(1,1) = -0{,}04 + \gamma \max\{0{,}8U(1,2) + 0{,}1U(2,1) + 0{,}1U(1,1) \ (Fel)$$

$$0{,}9U(1,1) + 0{,}1U(1,2) \ (Balra)$$

$$0{,}9U(1,1) + 0{,}1U(2,1) \ (Le)$$

$$0{,}8U(2,1) + 0{,}1U(1,2) + 0{,}1U(1,1)\} \ (Jobbra)$$

A 17.3. ábrán látható számok behelyettesítésével azt láthatjuk, hogy a *Fel* a legjobb cselekvés.

2.2. Az értékiteráció algoritmus

A Bellman-egyenletek képezik az alapját az MDF-ek megoldására szolgáló értékiteráció algoritmusnak. Ha *n* lehetséges állapot van, akkor *n* Bellman-egyenlet létezik, mindegyik állapotra egy. Az *n* egyenlet *n* ismeretlent tartalmaz – az állapotok hasznosságát. Így a hasznosságok megállapításához ezen egyenletek együttesét szeretnénk megoldani. Azonban van egy probléma: az egyenletek nemlineárisak, mivel a „max” operátor nemlineáris operátor. Míg a lineáris egyenletrendszerek hatékonyan megoldhatók lineáris algebrai eszközökkel, a nemlineáris egyenletrendszerek már problematikusabbak. Ekkor iteratív módszerrel próbálkozhatunk. Kezdéskor tetszőleges kezdeti értékeket választunk a hasznosságoknak, kiszámítjuk az egyenletek jobb oldalát, majd behelyettesítjük a bal oldalra – így frissítve az egyes állapotok hasznosságát a szomszédjainak a hasznosságával. Ezt addig ismételjük, ameddig el nem érünk egy egyensúlyi helyzetet. Jelölje $U_i(s)$ az *s* állapot hasznosságértékét az *i*-edik iterációban. A **Bellman-frissítés**nek (**Bellman update**) nevezett iterációs lépés a következőképpen néz ki:

Created by XMLmind XSL-FO Converter.

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s,a,s') U_i(s') \qquad (17.6)$$

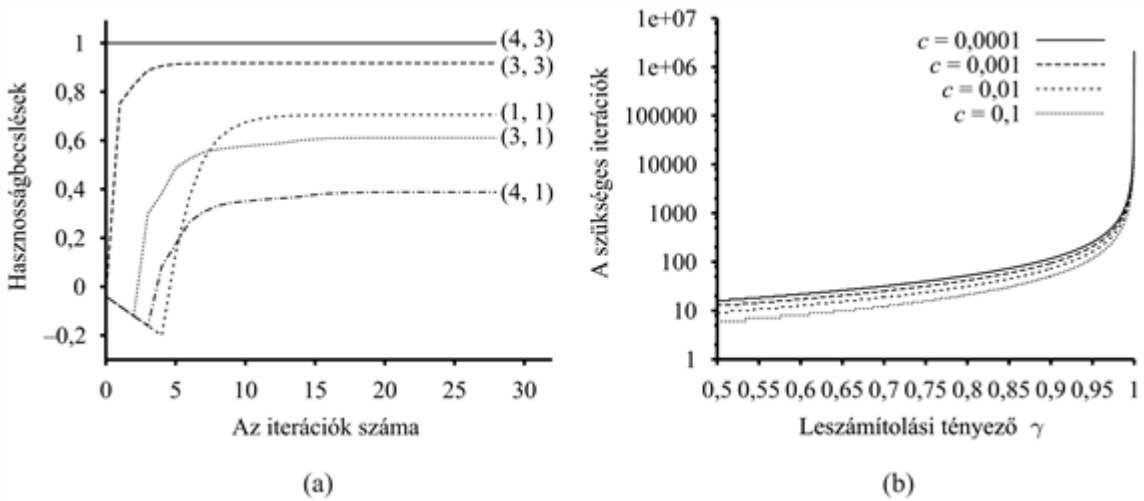
Ha a Bellman-frissítést végtelen sokszor alkalmazzuk, garantált, hogy egyensúlyi helyzetet érünk el (lásd következő alfejezet), és ekkor a végső hasznosságértékeknek a Bellman-egyenletek megoldását kell adniuk. Valójában ezek *egyértelmű* megoldások is, és a hozzájuk tartozó eljárásmód optimális (amit a (17.4) egyenlettel nyerhetünk). Az algoritmus, amit ÉRTÉKITERÁCIÓ-nak nevezünk, a 17.4. ábrán látható.

17.4. ábra - Az értékiteráció algoritmus az állapotok hasznosságának kiszámítására. A leállási feltétel a (17.8) egyenletből származik.

function ÉRTÉKITERÁCIÓ(*mdf*, ε) **returns** egy hasznosságfüggvény
  inputs: *mdf*, egy MDF *S* állapotokkal, *T* állapotátmenet-modellel, *R* jutalomfüggvénnyel,
    γ leszámítolási tényezővel
    ε, maximális megengedett hiba az állapotok hasznosságában
  local variables: *U*, *U'* hasznosságértékek vektorai *S*-beli állapotokhoz, kezdetben nulla
    δ, a maximális változás egy állapot hasznosságában egy iteráción belül

  repeat
    $U \leftarrow U'$; $\delta \leftarrow 0$
    for each *s* állapotra **in** *S* **do**
      $U'[s] \leftarrow R[s] + \gamma \max_a \sum_{s'} T(s, a, s')\, U[s']$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \varepsilon(1-\gamma)/\gamma$
  return *U*

17.5. ábra - (a) A grafikonokon kiválasztott állapotok hasznosságainak a fejlődése látható az értékiteráció felhasználásánál. (b) A szükséges értékiterációk száma (*k*), hogy a hiba garantáltan legfeljebb $\varepsilon = cR_{max}$ legyen, *c* különböző értékeinél, mint a γ leszámítolási tényező függvénye.



Az értékiterációt alkalmazhatjuk a 17.1. (a) ábra 4 × 3-as világára. Nulla kezdeti értékekről indulva, a hasznosságok alakulása a 17.5. (a) ábrán látható. Vegyük észre, ahogy a (4, 3) állapottól különböző távolságra lévő állapotok negatív jutalmakat halmoznak fel egészen addig, amíg egy bizonyos ponton utat találnak a (4, 3)-hoz, amikortól is a hasznosságok elkezdenek növekedni. Az értékiteráció algoritmusára gondolhatunk úgy, mint lokális frissítések általi *információterjesztésre* az állapottérben.

2.3. Az értékiteráció konvergenciája

Created by XMLmind XSL-FO Converter.

Azt állítottuk, hogy az értékiteráció végül a Bellman-egyenletek egy egyértelmű megoldásához konvergál. Ebben az alfejezetben kifejtjük, miért is történik ez. Ennek során bevezetünk néhány hasznos matematikai fogalmat, és olyan módszereket kapunk, amelyek megbecsülik a hasznosságfüggvény hibáját , ha az algoritmust hamar állítottuk le; ennek hasznosságát az adja, hogy nem kell végtelen hosszan futtatnunk az algoritmust. Az alfejezet a technikai részleteket is bemutatja.

Az értékiteráció konvergenciájának megmutatásában használt alapfogalom az **összehúzás** (**contraction**). Az összehúzás nagyjából egy olyan egyváltozós függvény, ami két különböző bemeneti értékre alkalmazva, két olyan kimeneti értéket ad, amelyek „egymáshoz közelebbiek", mint az eredeti értékek, legalább egy bizonyos állandó mennyiséggel. Például a „kettővel való osztás" egy összehúzás, mivel bármely két szám elosztása után a különbségük megfeleződik. Vegyük észre, hogy a „kettővel való osztásnak" van egy fix pontja, nevezetesen a nulla, ami a függvény alkalmazásával nem változik. Ebből a példából az összehúzás két fontos tulajdonságát vehetjük észre:

- Egy összehúzásnak egyetlen fix pontja van: ha két fix pontja lenne, nem kerülnének egymáshoz közel a függvény alkalmazásakor, így nem is volna összehúzás.
- A függvény tetszőleges argumentumra történő alkalmazásakor a függvényértéknek közelebb kell kerülniük a fix ponthoz (mivel a fix pont nem mozdul el), így egy öszszehúzás ismételt alkalmazása a fix pontot adja határértékben.

Most tegyük fel, hogy a Bellman-frissítést (a (17.6) egyenletet) egy B operátornak tekintjük, aminek egyszeri alkalmazása az összes állapot hasznosságát frissíti. Jelölje U_i az összes állapot i-edik iterációbeli hasznosságának a vektorát. Ekkor a Bellman-frissítési egyenletet felírhatjuk úgy, hogy

$$U_{i+1} \leftarrow BU_i$$

Most egy olyan módszerre van szükségünk, amellyel a hasznosságvektorok távolságát mérhetjük. A **maximum normát** (**max norm**) fogjuk használni, ami egy vektor hoszszát a legnagyobb komponensének hosszával méri:

$$\|U\| = \max_s |U(s)|$$

Fontos

Ezzel a definícióval, a két vektor közötti „távolság" $\|U - U'\|$ az összetartozó elemek közötti különbségek maximuma. Az alfejezet fő eredménye a következő: *Legyen U_i és két tetszőleges hasznosságvektor. Ekkor*

$$\|BU_i - BU_i'\| \leq \gamma \|U_i - U_i'\| \quad (17.7)$$

Azaz, a Bellman-frissítés egy összehúzás egy γ tényezővel a hasznosságvektorok terében. Így az értékiteráció mindig a Bellman-egyenletek egy egyértelmű megoldásához konvergál.

Nevezetesen, a (17.7) egyenletben lecserélhetjük U_i' -t a valódi U hasznosságokkal, amire $BU = U$. Ekkor a következő egyenlőtlenséget kapjuk

$$\|BU_i - U\| \leq \gamma \|U_i - U\|$$

Ezért, ha $\|U_i - U\|$-ra mint az U_i becslés *hibájára* tekintünk, látjuk, hogy a hiba minden iterációban legalább egy γ tényezővel csökken. Ez azt jelenti, hogy az értékiteráció exponenciálisan gyorsan konvergál. A szükséges iterációk számát egy adott e hibahatár eléréséhez a következőképpen számíthatjuk ki: először, a (17.1) egyenlet szerint, az összes állapot hasznossága korlátos $\pm R_{max}/(1-\gamma)$ értékkel. Ez azt jelenti, hogy a maximális kezdeti hiba $\|U_0 - U\| \leq 2R_{max}/(1-\gamma)$. Tegyük fel, hogy N iterációt végzünk ahhoz, hogy a hiba legfeljebb ε értékű legyen. Mivel a hiba minden egyes alkalommal legalább γ -szorosára csökken, annak kell teljesülnie, hogy $\gamma^N 2R_{max}/(1-\gamma) \leq \varepsilon$. Ha vesszük a két oldal logaritmusát, az adódik, hogy

$$N = \lceil \log(2R_{max} / \varepsilon(1-\gamma)) / \log(1/\gamma) \rceil$$

számú iteráció elegendő. A 17.5. (b) ábrán látható, hogyan változik γ függvényében az N különböző értékű ε/R_{max} hányadosok esetében. A jó hír az, hogy az exponenciális gyorsaságú konvergencia miatt N nem függ túlságosan az ε/R_{max} hányadostól. A rossz hír az, hogy N gyorsan nő, ahogy γ közel kerül 1-hez. Így gyors

Created by XMLmind XSL-FO Converter.

konvergenciát úgy érhetünk el, ha γ -t kicsivé tesszük, de ez valójában az ágensnek rövid távú horizontot biztosít, és az ágens cselekvéseinek hosszú távú hatásait figyelmen kívül hagyhatja.

Az előző bekezdés hibakorlátja ad bizonyos tájékozódást az algoritmus futási idejét befolyásoló tényezőkről, de gyakran túlságosan óvatos módszert jelent annak eldöntésére, hogy mikor álljon le az iteráció. Az utóbbi célra felhasználhatunk egy korlátot, ami a hibát az egyes iterációkban a Bellman-frissítés nagyságához kapcsolja. Az összehúzási tulajdonságból ((17.7) egyenlet) meg lehet mutatni, hogy ha a frissítés kicsi (azaz egyetlen állapot hasznossága sem változik nagyon), akkor a hiba a valódi hasznosságfüggvényhez képest szintén kicsi. Pontosabban,

ha $||U_{i+1} - U_i|| < \varepsilon(1 - \gamma)/\gamma$, akkor $||U_{i+1} - U|| < \varepsilon$ (17.8)

Ez a leállási feltétel szerepel az ÉRTÉKITERÁCIÓ algoritmusában a 17.4. ábrán.

Fontos

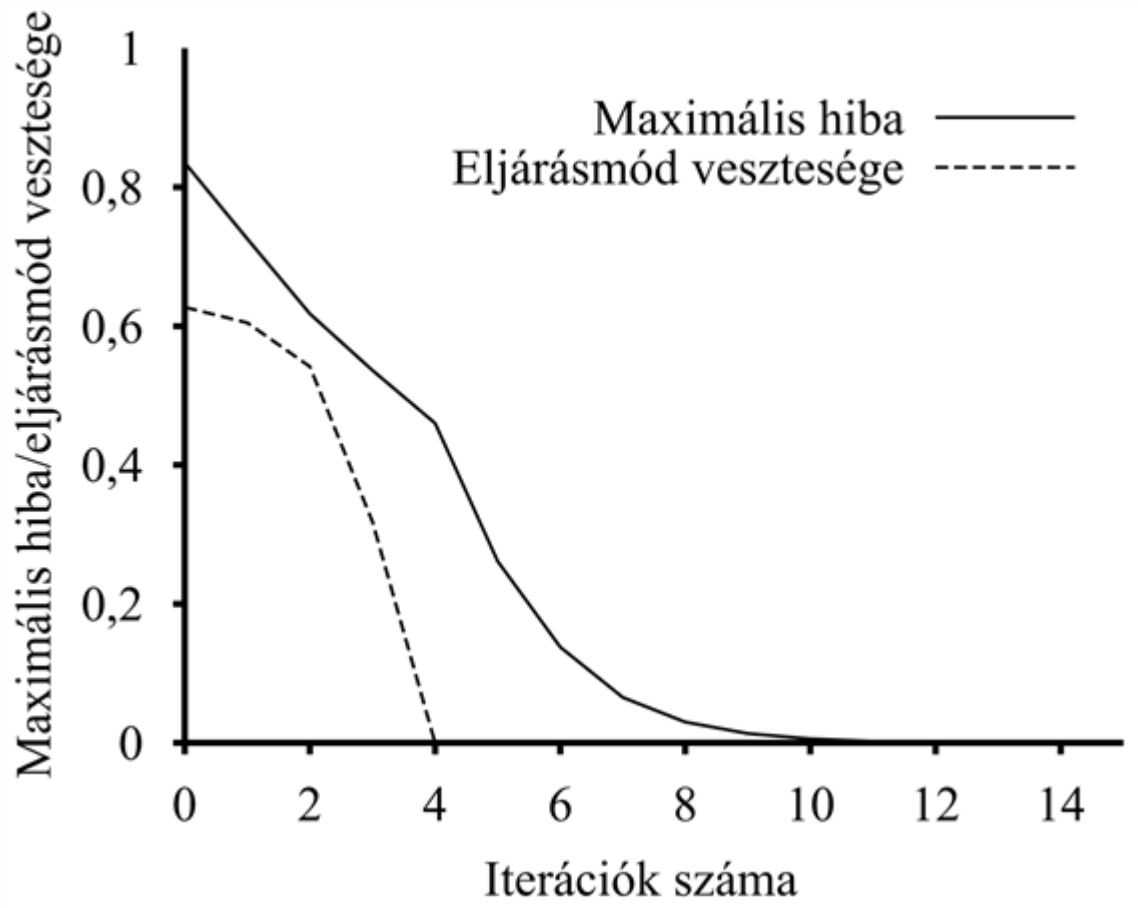
Eddig az értékiteráció algoritmusa által visszaadott hasznosságfüggvény hibáját elemeztük. *Az ágenst azonban igazán az érdekli, hogyan fog boldogulni, ha egy ilyen döntésfüggvény alapján hoz döntéseket.* Tegyük fel, hogy az értékiteráció *i*-edik iterációja után az ágens a valódi U hasznosságra egy U_i becslést kap, és az U_i-n alapuló egylépéses előrenézés felhasználásával (ahogy a (17.4) egyenletben) megkapja a π_i MVH-eljárásmódot. Vajon a kiadódó viselkedés megközelítőleg lesz-e olyan jó, mint az optimális viselkedés? Ez döntő kérdés minden valós ágens számára, és bizonyítható, hogy a válasz igen. Jelölje $U^{\pi_i}(s)$ azt a hasznosságot, ami a π_i végrehajtásakor adódik *s*-ből kiindulva. Ekkor az $\|U^{\pi_i} - U\|$ az **eljárásmód vesztesége** (**policy loss**) annak a maximális értéke, amit az ágens veszíthet π_i-t végrehajtva az optimális π^* eljárásmód helyett. A π_i eljárásmód veszteségét az U_i-beli hibához a következő egyenlőtlenség kapcsolja:

$$\text{ha} \quad \|U_i - U\| < \varepsilon, \quad \text{akkor} \quad \|U^{\pi_i} - U\| < 2\varepsilon\gamma/(1-\gamma) \qquad (17.9)$$

A gyakorlatban sokszor előfordul, hogy π_i jóval hamarabb optimálissá válik, mielőtt U_i konvergált volna. A 17.6. ábrán látható, ahogy a 4 × 3-as környezetben a maximális U_i-beli hiba és az eljárásmód vesztesége az értékiteráció folyamatának előrehaladtával nullához közelít, γ = 0,9 értéknél. A π_i eljárásmód már i = 4-nél optimális, pedig az U_i-beli maximális hiba még 0,46.

17.6. ábra - A hasznosságbecslés maximális hibája $||U_i - U||$ és az eljárásmód vesztesége $\|U^{\pi_i} - U\|$ az optimális eljárásmódhoz viszonyítva az értékiteráció iterációszámának függvényében

Created by XMLmind XSL-FO Converter.



Most már minden rendelkezésünkre áll, hogy az értékiterációt a gyakorlatban is felhasználjuk. Tudjuk, hogy a helyes hasznosságokhoz konvergál, a hasznosság becslésének hibájára korlátokat tudunk adni, ha véges számú iteráció után leállunk, és korlátokat tudunk adni az eljárásmód veszteségére, ami a becsült hasznosságértékekhez tartozó MVH-eljárásmód végrehajtása miatt lép fel. A végső megjegyzés, hogy az összes eredmény feltétele ebben az alfejezetben a $\gamma < 1$-gyel való leszámítolás. Ha $\gamma = 1$ és a környezet tartalmaz végállapotokat, akkor bizonyos technikai feltételek teljesülése esetén konvergenciaeredmények és hibakorlátok hasonló halmaza származtatható.

3. Eljárásmód-iteráció

Az előző alfejezetben megfigyeltük, hogy optimális eljárásmódot akkor is kaphatunk, amikor a hasznosságfüggvény becslése pontatlan. Ha egy cselekvés egyértelműen jobb, mint a többi, akkor a releváns állapotok hasznosságainak a pontos nagyságát nem szükséges precízen tudnunk. Ez a megérzés egy alternatív módot javasol az optimális eljárásmód megkeresésére. Az **eljárásmód-iterációs (policy iteration)** algoritmus egy π_0 kezdeti eljárásmódtól indulva a következő két lépést váltogatja:

- **Eljárásmód-értékelés (policy evaluation)**: Egy adott π_i eljárásmódnál számítsuk ki $U_i = U^{\pi_i}$-t, az egyes állapotok hasznosságát mintha π_i volna végrehajtva.
- **Eljárásmód-javítás (policy improvement)**: Számítsunk ki egy új π_{i+1} MVH-eljárásmódot, felhasználva az U_i-n alapuló egylépéses előrenézést (mint a (17.4) egyenletben).

Az algoritmus leáll, amikor az eljárásmód-javítás lépése nem eredményez változást a hasznosságokban. Ennél a pontnál tudjuk, hogy az U_i hasznosságfüggvény a Bellman-frissítés egy fix pontja, így a Bellman-egyenletek egy megoldása, és π_i-nek egy optimális eljárásmódnak kell lennie. Mivel csak véges sok eljárásmód létezik a véges állapottérben, és megmutatható, hogy minden iteráció jobb eljárásmódot eredményez, az eljárásmód-iterációnak le kell állnia. Az algoritmus a 17.7. ábrán látható.

Created by XMLmind XSL-FO Converter.

Az eljárásmód-javítás lépése nyilvánvalóan egyértelmű, de hogyan valósítsuk meg az eljárásmód-értékelést. Kiderül, hogy ennek elvégzése sokkal egyszerűbb, mint a szabványos Bellman-egyenletek megoldása (amit az értékiteráció végez el), mivel az eljárásmód egy cselekvést minden állapotban rögzít. Az *i*-edik iterációban a π_i eljárásmód a $\pi_i(s)$ cselekvést írja elő. Ez azt jelenti, hogy a (17.5) Bellman-egyenlet egy egyszerűsített változatával állunk szemben, ami az *s* hasznosságát (π_i mellett) a következőképpen kapcsolja a szomszédai hasznosságához:

$$U_i(s) = R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_i(s') \qquad (17.10)$$

Például tegyük fel, hogy π_i a 17.2. (a) ábrán látható eljárásmód. Ekkor azt kapjuk, hogy $\pi_i(1, 1) = Fel$, $\pi_i(1, 2) = Fel$ és így tovább, illetve az egyszerűsített Bellman-egyenletek:

$U_i(1, 1) = -0,04 + 0,8U_i(1, 2) + 0,1U_i(1, 1) + 0,1U_i(2, 1)$

$U_i(1, 2) = -0,04 + 0,8U_i(1, 3) + 0,2U_i(1, 2)$

…

A lényeges az, hogy ezek az egyenletek lineárisak, mivel a „max” operátort eltávolítottuk. Ha az állapotok száma *n*, *n* lineáris egyenlet adódik *n* ismeretlennel, ami egzakt módon megoldható $O(n^3)$ időben standard lineáris algebrai módszerekkel.

17.7. ábra - Az eljárásmód-iterációs algoritmus egy optimális eljárásmód kiszámítására

function ELJÁRÁSMÓD-ITERÁCIÓ(*mdf*) **returns** egy eljárásmód
  inputs: *mdf*, egy MDF *S* állapotokkal, *T* állapotátmenet-modellel
  local variables: *U*, *U'* hasznosságértékek vektorai *S*-beli állapotokhoz, kezdetben zérók
    π, egy eljárásmód vektor állapotokkal indexelve, kezdetben véletlen

  repeat
    U ← ÉRTÉKMEGHATÁROZÁS(π, *U*, *mdf*)
    változatlan? ← igaz
    for each *s* állapotra **in** *S* **do**
      if $\max_a \sum_{s'} T(s,a,s')U[s'] > \sum_{s'} T(s,\pi[s],s')U[s']$ **then**
        $\pi[s] \leftarrow \text{argmax}_a \sum_{s'} T(s,a,s')U[s']$
        változatlan? ← hamis
  until *változatlan*?
  return π

Kis állapotterekre az eljárásmód-kiértékelés egzakt megoldómódszerekkel gyakran a leghatékonyabb megközelítés. Nagy állapotterekre az $O(n^3)$ idő megengedhetetlen lehet. Szerencsére nem szükséges egzakt eljárásmód-kiértékelést végezni. Ehelyett elvégezhetünk bizonyos számú egyszerűsített értékiterációs lépést (egyszerűsített, mivel az eljárásmód rögzített), hogy a hasznosságok elfogadhatóan jó becsléséhez jussunk. Az egyszerűsített Bellman-frissítés ehhez a folyamathoz a következő

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_i(s')$$

amit *k*-szor ismétlünk, hogy a következő hasznosságbecslés előálljon. Az adódó algoritmust **módosított eljárásmód-iteráció**nak (**modified policy iteration**) nevezzük. Ez gyakran sokkal hatékonyabb, mint a szabványos eljárásmód-iteráció vagy értékiteráció.

Created by XMLmind XSL-FO Converter.

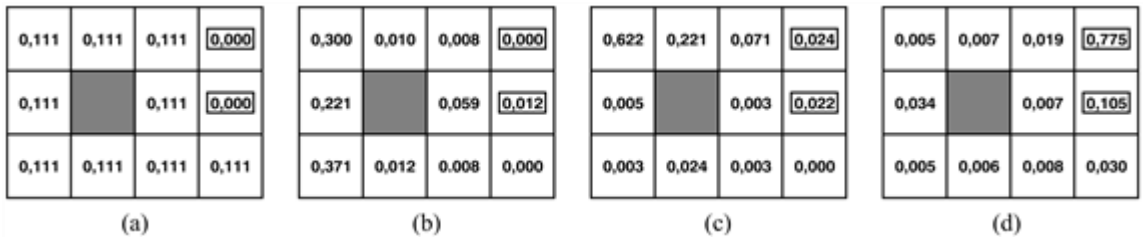
Az eddig leírt algoritmusok megkövetelik a hasznosságok vagy eljárásmódok összes állapotbeli egyszerre történő frissítését. Megmutatható, hogy ez nem szigorúan szükséges. Valójában minden iterációnál kiválaszthatjuk az állapotok egy *tetszőleges* részhalmazát, és alkalmazhatjuk rá *bármelyik* fajta frissítést (eljárásmód-javítást vagy egyszerűsített értékiterációt). Ezt az általánosított algoritmust **aszinkron eljárásmód-iteráció**nak (**asynchronous policy iteration**) nevezzük. A kezdeti eljárásmód és a hasznosságfüggvény adott feltételek melletti megválasztásánál az aszinkron eljárásmód-iteráció garantáltan konvergál az optimális eljárásmódhoz. A feldolgozandó állapotok megválasztásának szabadsága azt jelenti, hogy hatékonyabb heurisztikus algoritmusokat tervezhetünk – például olyan algoritmusokat, amelyek azon állapotok értékének frissítésére koncentrálnak, amelyeket egy jó eljárásmód valószínűleg elér. A valós életben ez nagyon hasznosnak tűnik: ha valakinek nincs szándékában a mélybe vetni magát egy szikláról, akkor nem kell aggódással töltenie az időt a bekövetkező állapotok pontos értékéről.

4. Részlegesen megfigyelhető Markov döntési folyamatok

A Markov döntési folyamatok leírása a 17.1. alfejezetben feltételezte, hogy a környezet **teljesen megfigyelhető** (**fully observable**). E mellett a feltételezés mellett az ágens mindig tudja, hogy melyik állapotban van. Ez együttesen az állapotátmenet-modell Markov-tulajdonságának a feltevésével azt eredményezi, hogy az optimális eljárásmód csak az aktuális állapottól függ. Amikor a környezet csak **részlegesen megfigyelhető** (**partially observable**), a helyzet nem ennyire átlátható. Az ágens nem tudja feltétlenül, hogy melyik állapotban is van, így nem hajthatja végre az adott állapothoz javasolt $\pi(s)$ cselekvést. Továbbá, egy s állapot hasznossága és az s-beli optimális cselekvés nemcsak s-től függ, hanem attól is, hogy az ágens mennyit tud, amikor s-ben van. Ebből kifolyólag a **részlegesen megfigyelhető MDF**-eket (**RMMDF**) (**partially observable MDP**s, **POMDP**s) általában sokkal nehezebb dolgoknak tartják, mint a hagyományos MDF-eket. Az RMMDF-ek azonban nem megkerülhetők, mivel a valódi világ is egy RMMDF.

Példaként gondoljuk át újra a 17.1. ábra 4 × 3-as világát, de most tegyük fel, hogy az ágensnek *egyáltalán nincsenek érzékelői,* és *elképzelése sincs* arról, hogy hol lehet. Pontosabban, tegyük fel, hogy az ágens kezdeti állapota egyenlő valószínűséggel valamelyike a kilenc nem végállapotnak (17.8. (a) ábra). Világos, hogz ha az ágens tudná, hogy (3, 3)-ban van, akkor *Jobbra* mozgást hajtana végre; ha *tudná*, hogy (1, 1)-ben van, akkor *Fel* mozogna; de mivel bárhol lehet, mit is kellene tennie? Egy lehetséges válasz, hogy az ágensnek először olyan cselekvéseket kell végrehajtani, amelyek csökkentik a bizonytalanságát, és csak ezután kell megpróbálni a +1 kijárat felé tartani. Például ha az ágens öt alkalommal *Balra* mozgást hajt végre, akkor igen valószínű, hogy a bal oldali falnál van (17.8. (b) ábra). Ezután öt alkalommal *Fel* mozgást hajt végre, akkor igen valószínű, hogy a felső falnál van, valószínűleg a bal felső sarokban (17.8. (c) ábra). Végül öt *Jobbra* mozgást hajt végre, ekkor jó eséllyel – körülbelül 77,5%-kal – eléri a +1 kijáratot (17.8. (d) ábra). A *Jobbra* mozgás ezt követő folytatása az esélyét 81,8%-ra növeli. Ez az eljárásmód így meglepően biztonságos, de ebben az esetben az ágens igen lassan éri el a kijáratot, és a várható hasznossága csupán 0,08 körül van. Az optimális eljárásmód, amit hamarosan leírunk, sokkal jobban teljesít.

17.8. ábra - (a) Az ágens elhelyezkedésének kezdeti valószínűségi eloszlása. (b) Ötszöri *Balra* mozgás után. (c) Ötszöri *Fel* mozgás után. (d) Ötszöri *Jobbra* mozgás után.



Az RMMDF-ek kezeléséhez először is megfelelően kell őket definiálni. Egy RMMDF-nek az elemei ugyanazok, mint egy MDF-nek – egy $T(s, a, s')$ állapotátmenet-modell és egy $R(s)$ jutalomfüggvény –, de van egy $O(s, o)$ **megfigyelési modell**je (**observational model**) is, ami az s állapotban az o megfigyelés

Created by XMLmind XSL-FO Converter.

érzékelésének a valószínűségét adja meg.[3] Például az érzékelő nélküli ágensünknek csak egyetlen megfigyelése van (az üres megfigyelés), ami minden állapotban 1 valószínűséggel bekövetkezik.

A 3. és 12. fejezetben nemdeterminisztikus és részlegesen megfigyelhető tervkészítési problémákat tanulmányoztunk, és megállapítottuk, hogy a **hiedelmi állapot** (**belief state**) – az aktuális állapotok halmaza, amelyekben az ágens lehet – kulcsfogalom a megoldások leírására és kiszámítására.

Az RMMDF-ekben a fogalmat kissé finomítjuk. Egy b hiedelmi állapot most egy valószínűség-eloszlás lesz az összes lehetséges állapot felett. Például a 17.8. (a) ábra kezdeti hiedelmi állapota úgy írható, hogy $\left\langle \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, 0, 0 \right\rangle$. A b hiedelmi állapot által az aktuális i állapothoz rendelt valószínűséget $b(s)$-sel jelöljük. Az ágens úgy számíthatja ki a jelenlegi hiedelmi állapotát, mint egy *feltételes eloszlást* az aktuális állapotok felett, az eddigi megfigyelések és cselekvések sorozatának ismeretében. Ez lényegében a **szűrés**i (**filtering**) feladat (lásd 15. fejezet). A rekurzív szűrési alapegyenlet (lásd (15.3) egyenlet a 2.1. szakasz - Szűrés és előrejelzés részben) megmutatja, hogy hogyan számolhatjuk ki az új hiedelmi állapotot az előző hiedelmi állapotból és az új megfigyelésből. Az RMMDF-eknél a jelölés eltérő, és meg kell gondolni még a cselekvést is, de az eredmény lényegében ugyanaz. Ha $b(s)$ volt az előző hiedelmi állapot, és az ágens az a cselekvést hajtja végre és az o megfigyelést érzékeli, akkor az új hiedelmi állapot a következő:

$$b'(s') = \alpha O(s', o) \sum_{s} T(s, a, s') b(s) \qquad (17.11)$$

ahol a egy normalizációs konstans, ami a hiedelmi állapot 1-re összegzését biztosítja. Ezt az egyenletet úgy rövidíthetjük, hogy $b' = \text{ELŐRE}(b, a, o)$.

Fontos

A RMMDF-ek megértéséhez szükséges lényegi meglátás pedig ez: *az optimális cselekvés csak az ágens aktuális hiedelmi állapotától függ.* Így az optimális eljárásmód leírható a hiedelmi állapotok cselekvésekre történő $\pi^*(b)$ leképzésével. Ez *nem* függ az ágens *aktuális* állapotától, amiben tartózkodik. Ez hasznos tulajdonság, mivel az ágens nem ismeri az aktuális állapotát; csupán a hiedelmi állapotát. Ekkor egy RMMDF ágens döntési ciklusa a következő:

1. Adott jelenlegi b hiedelmi állapot esetén hajtsuk végre az $a = \pi^*(b)$ cselekvést.

2. Fogadjuk az o megfigyelést.

3. Állítsuk a jelenlegi állapothiedelmet $\text{ELŐRE}(b, a, o)$-ra, és ismételjük meg a ciklust.

Most úgy gondolhatunk az RMMDF-ekre, mint amelyek keresést igényelnek a hiedelmi állapotok terében, pontosan úgy, ahogy az érzékelő nélküli és az eshetőségi problémákra vonatkozó módszerek a 3. fejezetben. A fő különbség az, hogy az RMMDF hiedelmi állapot tere *folytonos,* mivel egy RMMDF hiedelmi állapot egy valószínűségi eloszlás. Például a hiedelmi állapot a 4 × 3-as világ esetében egy pont a 11 dimenziós folytonos térben. Egy cselekvés megváltoztatja a hiedelmi állapotot is, nemcsak a fizikai állapotot, így kiértékelése aszerint az információ szerint történik, amit a cselekvés eredményeként az ágens megszerez. Az RMMDF-ekbe ezért beletartozik az információ értéke is (16.6. alfejezet) mint a döntési probléma egy komponense.

Nézzük meg tüzetesebben a cselekvések kimenetelét. Nevezetesen, számítsuk ki annak valószínűségét, hogy az ágens a b hiedelmi állapotból a b' hiedelmi állapotba jut az a cselekvés végrehajtása után. Ekkor, ha ismernénk a cselekvést és a *bekövetkező megfigyelést,* akkor a (17.11) egyenlet a hiedelmi állapot egy determinisztikus frissítését adná: $b' = \text{ELŐRE}(b, a, o)$. Természetesen a bekövetkező megfigyelés még nem ismert, így az ágens a számos lehetséges hiedelmi állapot egyikébe, b'-be kerülhet az előforduló megfigyelés függvényében. Az o érzékelésének valószínűsége, feltéve hogy a b' hiedelmi állapotban az a cselekvés került végrehajtásra, az összes olyan s' aktuális állapot feletti összegzéssel adódik, amelyeket az ágens elérhet:

[3] A megfigyelési modell alapvetően azonos az időbeli folyamatok **érzékelő modell**jével (**sensor model**) a 15. fejezetből. Ahogyan az MDF-ek jutalomfüggvényénél, a megfigyelési modell szintén függhet a cselekvéstől és a kimeneteli állapottól, de ez most sem egy alapvető változás.

Created by XMLmind XSL-FO Converter.

$$P(o|a,b) = \sum_{s'} P(o|a,s',b)P(s'|a,b)$$
$$= \sum_{s'} O(s',o)P(s'|a,b)$$
$$= \sum_{s'} O(s',o)\sum_{s} T(s,a,s')b(s)$$

Jelölje $\tau(b, a, b')$ annak a valószínűségét, hogy b-ből b'-be jutunk egy a cselekvéssel. Ennek felhasználásával ekkor

$$\tau(b,a,b') = P(b'|a,b) = \sum_{o} P(b'|o,a,b)P(o|a,b)$$
$$= \sum_{o} P(b'|o,a,b)\sum_{s'} O(s',o)\sum_{s} T(s,a,s')b(s) \qquad (17.12)$$

ahol $P(b'|o, a, b)$ 1, ha b' = ELŐRE(b, a, o), egyébként pedig 0.

Fontos

A (17.2.) egyenletet felfoghatjuk egy állapotátmenet-modellnek a hiedelmi állapot térben. Definiálhatunk egy jutalomfüggvényt is a hiedelmi állapotokra (azaz azon aktuális állapotok várható jutalmát, amelyekben az ágens lehet):

$$\rho(b) = \sum_{s} b(s)R(s)$$

Így láthatóan $\tau(b, a, b')$ és $\rho(b)$ együtt egy *megfigyelhető* MDF-et definiál a hiedelmi állapotok terén. Továbbá megmutatható, hogy egy $\pi^*(b)$ optimális eljárásmód erre az MDF-re, szintén optimális eljárásmód az eredeti RMMDF-re. Máshogy fogalmazva, *egy RMMDF megoldása a fizikai állapottérben redukálható egy MDF megoldására a hozzá tartozó hiedelmi állapot térben.* Ez a tény talán kevésbé meglepő, arra gondolva, hogy a hiedelmi állapot definíció szerint az ágens számára mindig megfigyelhető.

Vegyük észre, hogy bár az RMMDF-eket sikeresen redukáltuk az MDF-ekre, a kapott MDF-eknek folytonos (és általában sokdimenziós) állapotterük van. A 17.2. és 17.3. alfejezetekben leírt algoritmusok egyike sem alkalmazható közvetlenül ilyen MDF-ekre. Kiderült, hogy az érték- és eljárásmód-iterációnak kifejleszthetők olyan változatai, amelyek folytonos állapotú MDF-ekre is alkalmazhatók. Az alapötlet az, hogy egy $\pi(b)$ eljárásmód reprezentálható a hiedelmi állapot tér olyan *részeinek* egy halmazaként, amelyek mindegyikéhez egy adott optimális cselekvés tartozik.[4] Az értékfüggvény b különböző lineáris függvényét rendeli az egyes térrészekhez. Minden érték- és eljárásmód-iterációs lépés finomít a térrészek határain, és új térrészeket vezethet be.

Az algoritmus részletei meghaladják a könyv kereteit, de megadjuk a megoldást az érzékelőmentes 4 × 3-as világra. Az optimális eljárásmód a következő:

```
[Balra, Fel, Fel, Jobbra, Fel, Fel, Jobbra, Fel, Fel, Jobbra, Fel, Jobbra, Fel, …]
```

Az eljárásmód egy sorozat, mivel ez a probléma determinisztikus a hiedelmi állapotok terében – nincsenek megfigyelések. Azt a „trükköt” alkalmaztuk, hogy az ágens egyetlen *Balra* mozgással biztosítja, hogy *nincs* (4, 1)-ben, azért, hogy aztán meglehetős biztonsággal mozoghasson *Fel* és *Jobbra* a +1 kijárat eléréséhez. Az ágens a +1 kijáratot 86,6% gyakorisággal éri el, és sokkal gyorsabban, mint az alfejezet elején megadott eljárásmód esetén, így a várható hasznossága 0,38, szemben az előző 0,08-as értékével.

Bonyolultabb RMMDF-eknél, amikor megfigyelések is vannak, nagyon nehéz (valójában PSPACE-beli – azaz valóban nagyon nehéz) közelítőleg optimális eljárásmódokat találni. A problémák néhány tucat változóval gyakran kivitelezhetetlenek. A következő alfejezet egy eltérő közelítő módszert ír le RMMDF-ek megoldására, ami az előrenéző keresésen alapul.

[4] Bizonyos RMMDF-ekre az optimális eljárásmódhoz végtelen számú térrész tartozik, így az egyszerű térrészek listája megközelítés nem működik, és ötletesebb módszerek szükségesek még egy közelítés megtalálásához is.

Created by XMLmind XSL-FO Converter.

5. Döntéselméleti ágensek

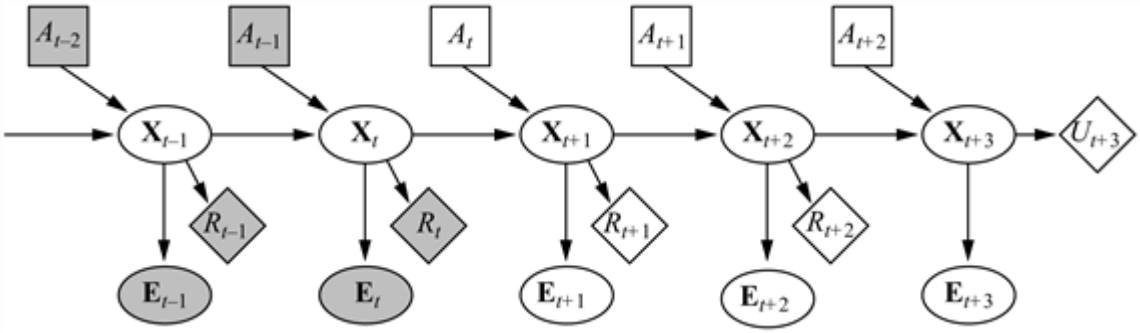
Ebben a fejezetben egy átfogó szemléletmódot vázolunk fel az ágensek megtervezésére részlegesen megfigyelhető, sztochasztikus környezetek esetén. A tervezés alapelemei már ismertek:

- Az állapotátmenet- és megfigyelési modelleket egy **dinamikus Bayes-háló**val (**dynamic Bayesian network**) ábrázoljuk (a 15. fejezetben leírtak szerint).
- A dinamikus Bayes-háló kiegészült döntési és hasznossági csomópontokkal, amelyek megegyeznek a 16. fejezetben szereplő döntési hálókban használatosakkal. A kirajzolódó modellt **dinamikus döntési háló**nak vagy **DDH**-nak (**dynamic decision network**, **DDN**) nevezzük.
- Egy szűrési algoritmust használunk az új érzékelések és cselekvések befogadására és a hiedelmi állapot reprezentációjának a frissítésére.
- Döntéseket hozunk lehetséges cselekvéssorozatok elgondolásával és a legjobb kiválasztásával.

Egy dinamikus Bayes-háló felhasználásának elsődleges haszna az állapotátmenet- és érzékelő modellek ábrázolásánál az, hogy felbontja az állapotleírást valószínűségi változók egy halmazára, hasonlóan ahhoz, ahogy a tervkészítési algoritmusok logikai ábrázolást vesznek igénybe az állapottér felbontására, amit a keresési algoritmusok használnak. Az ágensterv ezért a 2. fejezetben vázolt hasznosságalapú ágens egy gyakorlati megvalósítása.

Mivel dinamikus Bayes-hálókat használunk, visszatérünk a 15. fejezetbeli jelöléshez, ahol $\mathbf{X}_t$ az állapotváltozók egy halmazát jelölte egy t időpillanatban, $\mathbf{E}_t$ pedig a bizonyítékváltozókat. Így ahol ebben a fejezetben eddig s_t-t használtunk (a t-beli állapotra), mostantól $\mathbf{X}_t$-t fogunk használni. A t-beli cselekvést A_t-vel fogjuk jelölni, így a $T(s, a, s')$ állapotátmenet-modell ugyanaz, mint a $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{X}_t, A_t)$, és az $O(s, o)$ megfigyelési modell ugyanaz, mint a $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$. A t időpontban kapott jutalmat jelöljük R_t-vel, és U_t-vel az állapot hasznosságát a t időpontban. E szerint a jelölés szerint egy dinamikus döntési háló a 17.9. ábrán látható módon néz ki.

17.9. ábra - Egy dinamikus döntési háló általános struktúrája. Az ismert értékű változók árnyékoltak. A jelenlegi időpont *t*, és az ágensnek el kell döntenie, hogy mit tegyen – azaz választania kell egy értéket A_t számára. A háló három jövőbeli lépésre van kibontva, és a jövőbeli jutalmakat, valamint az előrelátó horizonton lévő állapot hasznosságát ábrázolja.



A dinamikus döntési hálók tömör leírást biztosítanak nagy RMMDF-ekre, így bármely RMMDF algoritmus bemenetéül szolgálhatnak, ideértve az érték- és eljárásmód-iterációra való módszereket is. Ebben az alfejezetben az előrenéző módszerekre összpontosítunk, amelyek cselekvéssorozatokat képzelnek el a jelenlegi hiedelmi állapottól kiindulva, nagyon hasonlóan a 6. fejezetbeli játékok algoritmusaihoz. A 17.9. ábrán a háló három jövőbeli lépésig elképzelve szerepel; a jelenlegi és a jövőbeli döntések és a jövőbeli megfigyelések és jutalmak mind ismeretlenek. Figyeljük meg, hogy a háló csomópontokat tartalmaz az $\mathbf{X}_{t+1}$ és $\mathbf{X}_{t+2}$-beli *jutalmakra*, de *hasznosságot* az $\mathbf{X}_{t+3}$ esetében. Ez azért van így, mert az ágensnek maximalizálnia kell az összes jövőbeli jutalom (leszámított) összegét, és $U(\mathbf{X}_{t+3})$ reprezentálja az $\mathbf{X}_{t+3}$-hoz tartozó és az összes ezt követő jutalmat. Ahogyan a 6. fejezetben, most is feltesszük, hogy U-nak csak egy közelítő alakja érhető el: ha elérhetők lennének pontos hasznosságértékek, akkor nem lenne szükség 1 lépésnél hosszabb előretekintésre.

Created by XMLmind XSL-FO Converter.

A 17.9. ábrán látható DDH-hoz tartozó háromlépéses előrenéző keresési fa egy részét a 17.10. ábra mutatja. Mindegyik háromszög alakú csomópont egy olyan hiedelmi állapot, amiben az ágens egy A_{t+i} döntést hoz $i = 0, 1, 2, ...$ esetén. A kerek csomópontok a környezet válaszaihoz tartoznak, nevezetesen hogy milyen $\mathbf{E}_{t+i}$ megfigyelés fog felbukkanni. Látható, hogy nincsen véletlen csomópont, ami a cselekvések kimeneteléhez tartozna. Ez azért van így, mert egy cselekvésnél a hiedelmi állapot frissítése a konkrét kimeneteltől függetlenül determinisztikus.

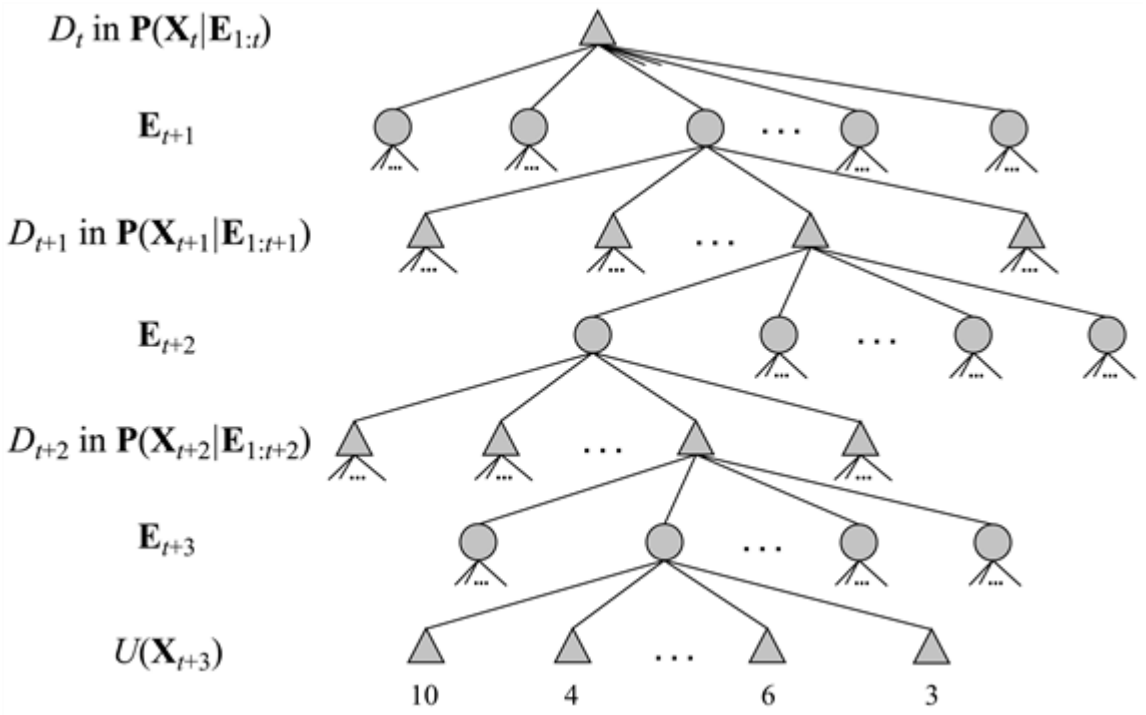
A hiedelmi állapot az egyes háromszög csomópontoknál kiszámítható egy szűrési algoritmusnak a hozzávezető megfigyelések és cselekvések sorozatán való alkalmazásával. Ezzel a módszerrel, az algoritmus figyelembe veszi azt a tényt, hogy egy A_{t+i} döntésnél az ágens számára elérhetők lesznek az $\mathbf{E}_{t+1}, ..., \mathbf{E}_{t+i}$ érzékelések, még ha a t időpontban nem is tudja, hogy ezek az érzékelések mi is lesznek. Ezen a módon a döntéselméleti ágens automatikusan figyelembe veszi az információ értékét és információgyűjtő cselekvéseket hajt végre, ha szükséges.

Egy döntés a keresési fából a levelek hasznosságértékének hátrafelé terjesztésével nyerhető ki, a véletlen csomópontoknál átlagot, a döntési csomópontokban pedig maximumot számítva. Ez hasonló a véletlen csomópontokkal rendelkező játékfákra vonatkozó `VÁRHATÓMINIMAX` algoritmushoz, kivéve, hogy (1) nem csak levélállapotokban lehet jutalom, és (2) a döntési csomópontok hiedelmi állapotokhoz, és nem aktuális állapotokhoz tartoznak. A d mélységű kimerítő keresés időkomplexitása $O(|D|^d \cdot |\mathbf{E}|^d)$, ahol $|D|$ az elérhető cselekvések száma, $|\mathbf{E}|$ pedig a lehetséges megfigyelések száma. Olyan problémáknál, ahol a γ leszámítolási tényező nincs közel 1-hez, gyakran elegendő sekély keresés az optimálishoz közeli döntés megtalálásához. Az is lehetséges, hogy az átlagszámítás lépését a lehetséges megfigyelések mintavételezésével közelítjük, az összes lehetséges megfigyelés feletti összegzés helyett. Számos más módja van jó közelítő megoldások gyors megtalálásának, de ezeket a 21. fejezetre hagyjuk.

A dinamikus döntési háló alapú döntéselméleti ágenseknek számos előnye van más, egyszerűbb, a korábbi fejezetekben bemutatott ágensfelépítésekhez képest. Nevezetesen, részlegesen megfigyelhető és sztochasztikus környezeteket kezelnek, és könnyen felülbírálják a „terveiket” a váratlan események kezeléséhez. Megfelelő érzékelő modellekkel képesek kezelni az érzékelőmeghibásodást, és képesek megtervezni az információ begyűjtését. Az idő szorításában és komplex környezetekben „teljesítményromlásuk fokozatos”, különböző közelítő technikák kihasználásával. Így jogos a kérdés, hogy mi is hiányzik még? A DDH-alapú algoritmusunknak a legkomolyabb hiányossága az előrefelé kereséstől való függés, ahogyan a II. részbeli állapottér-keresési algoritmusoknak is. A IV. részben elmagyaráztuk, hogy részlegesen rendezett, absztrakt tervek mérlegelésének képessége célirányos kereséssel hogyan biztosította a problémamegoldó képesség lényegi növekedését, különösen a tervkönyvtárakkal összeillesztve. Ezeket a módszereket megkísérelték a valószínűségi területre is kiterjeszteni, de ez eddig nem bizonyult hatékonynak. Egy második, kapcsolódó probléma a DDH nyelv alapvető kijelentéslogikai természete. Szeretnénk, ha az elsőrendű valószínűségi nyelvek elképzeléseit (lásd 14.6. alfejezet) ki tudnánk terjeszteni a döntéshozatal problémájára. A jelenlegi kutatások azt mutatják, hogy ez a kiterjesztés lehetséges, és jelentős előnyöket rejt magában, ahogy ezt a fejezet végi megjegyzéseknél megtárgyaljuk.

17.10. ábra - A 17.9. ábrán látható DDH előrenéző megoldásának egy része

Created by XMLmind XSL-FO Converter.



6. Többágenses döntések: a játékelmélet

Ez a fejezet a bizonytalan környezetbeli döntéshozatalra összpontosított. De mi a helyzet akkor, ha a bizonytalanság más ágenseknek és azok döntéseinek köszönhető? És ha ezeknek az ágenseknek a döntéseit a mi döntéseink befolyásolják? Ezzel a kérdéssel egyszer már foglalkoztunk, amikor a 6. fejezetben a játékokat tanulmányoztuk. Ott azonban többfordulós, teljes információs játékokkal foglalkoztunk, amelyekre a minimax kereséssel optimális lépések találhatók. Ebben az alfejezetben a **játékelmélet** (**game theory**) azon vonatkozásait tanulmányozzuk, amelyek felhasználhatók szimultán játékok elemzésére. Egyszerűsítés céljából elsőként olyan játékokat nézünk meg, amelyek csak egy lépés hosszúak. A „játék” szó és az egyetlen lépésre való korlátozás ezt triviálisnak tüntetheti fel, de valójában a játékelméletet nagyon komoly döntéshozatali helyzetekben használják – ideértve a bankcsődeljárásokat, a távközlési frekvenciasávok aukcióját, a termékfejlesztési és ármegállapítási döntéseket, illetve a honvédelmi kérdéseket – olyan helyzetekben, amelyek dollármilliárdok és életek százezreinek a sorsát érintik. A játékelmélet legalább kétféle módon használható fel:

1. **Ágenstervezés** (**agent design**): A játékelmélet képes elemezni az ágensek döntéseit, és kiszámítani az egyes döntések várható hasznosságát (azon feltevés mellett, hogy más ágensek a játékelmélet szerint optimálisan cselekednek). Például a **kétujjas snóblijáték**ban (**two-finger Morra**) két játékos, *O* és *E* egyidejűleg felmutatja egy vagy két ujját. Legyen az ujjak összes száma *f*. Ha *f* páratlan, akkor *O* kap *f* dollárt *E*-től, és ha *f* páros, akkor *E* kap *f* dollárt *O*-tól. A játékelmélet képes meghatározni a legjobb stratégiát egy racionális ellenféllel szemben, és az egyes játékosok várható hasznát.[5]

2. **Működési mód tervezés** (**mechanism design**): Amikor a környezetet számos ágens népesíti be, előfordulhat, hogy a környezet szabályait (azaz a játékot, amit az ágenseknek játszaniuk kell) olyannak határozhatjuk meg, hogy az összes ágens együttes haszna akkor maximális, amikor mindegyik ágens egy saját hasznát maximáló játékelméleti megoldást követ. Például a játékelmélet segíthet olyan protokollok tervezésében az internetes forgalomirányítók számára, hogy minden irányítónak olyan ösztönzői legyenek, hogy a globális átvitel maximális legyen. A működési mód tervezés arra is felhasználható, hogy olyan intelligens **többágenses rendszer**eket (**multiagent system**s) hozzunk létre, amelyek komplex problémákat elosztott módon oldanak meg, anélkül, hogy az egyes ágenseknek ismerniük kellene a megoldandó teljes problémát.

[5] A snóblijáték az **ellenőrzési játék** (**inspection game**) egy szabadidős változata. Ilyen játékokban az ellenőr megválaszt egy napot egy létesítmény (mint például egy vendéglő vagy egy biológiai fegyvergyár) ellenőrzésére, és a létesítmény működtetője szintén megválaszt egy napot, amikor az összes kellemetlen dolgot elrejti. Az ellenőr győz, ha a napok különböznek, és a létesítmény fenntartója, ha ugyanazok.

Created by XMLmind XSL-FO Converter.

Egy játékot a játékelméletben a következő alkotóelemek definiálnak:

- **Játékos**ok (**player**) vagy ágensek, akik döntéseket hoznak. A legnagyobb figyelem a kétszemélyes játékokra irányult, bár az *n* személyes játékok *n* > 2-re szintén gyakoriak. A játékosokra nagybetűs neveket használunk, mint *Aliz* és *Bendegúz* vagy *O* és *E*.

- **Cselekvés**ek (**action**s), amiket a játékosok választhatnak. A cselekvéseknek kisbetűs neveket választunk, mint az *egy* vagy a *tanúskodik*. Lehet, hogy a játékosok a cselekvéseknek ugyanazt a halmazát érik el, de az is lehet, hogy különbözőt.

- **Jutalommátrix** (**payoff matrix**), ami az összes játékos cselekvéseinek mindegyik kombinációjára, mindegyik játékos számára megadja a hasznosságot. A jutalommátrix a kétujjas snóblijátékhoz a következő:

	O: *egy*	*O*: *kettő*
E: *egy*	$E = 2, O = -2$	$E = -3, O = 3$
E: *kettő*	$E = -3, O = 3$	$E = 4, O = -4$

Például a jobb alsó sarokban az látható, amikor *O* a kettő cselekvést és *E* szintén a kettő cselekvést választja, a jutalom *E*-nek 4, *O*-nak pedig –4.

Minden játékosnak a játékban választania kell, és aztán végrehajtani egy **stratégiá**t (**strategy**) (ami a 16. fejezetben szereplő eljárásmód elnevezése a játékelméletben). A **tiszta stratégia** (**pure strategy**) egy determinisztikus eljárásmód, ami mindegyik helyzethez előír egy végrehajtandó konkrét cselekvést; az egylépéses játékokban a tiszta stratégia csupán egyetlen cselekvés. A játékok elemzése vezetett el a **kevert stratégiá**k (**mixed strategy**) gondolatához, amely eljárásmódban egy konkrét cselekvés véletlenszerűen kerül kiválasztásra egy adott eloszlásból a cselekvések felett. Azt a kevert stratégiát, ami *p* valószínűséggel az *a* cselekvést, egyébként pedig a *b*-t választja, úgy jelöljük, hogy [*p*: *a*; (1 – *p*): *b*]. Például a kétujjas snóblijátékban egy lehetséges kevert stratégia a [0,5: *egy*; 0,5: *kettő*]. A **stratégiaprofil** (**strategy profil**) egy hozzárendelés, ami mindegyik játékoshoz egy stratégiát rendel; adott stratégiaprofilnál a játék **kimenetel**e (**outcome**) egy numerikus érték mindegyik játékoshoz.

Egy játék **megoldás**a (**solution**) egy olyan stratégiaprofil, amiben minden játékos egy racionális stratégiát fogad el. Látni fogjuk, hogy a játékelméletben a legfontosabb kérdés a „racionális” jelentésének a definiálása akkor, amikor mindegyik ágens csak egy részét választja meg a kimenetelt meghatározó stratégiaprofilnak. Fontos azt felismerni, hogy a kimenetelek a játék egyes meneteinek a konkrét eredményei, míg a megoldások a játék elemzéséhez használt elméleti konstrukciók. Megmutatjuk, hogy bizonyos játékoknak csak a kevert stratégiák között van megoldása. Ez azonban nem jelenti azt, hogy egy játékosnak szó szerint egy kevert stratégiát kell elfogadnia a racionális viselkedéshez.

Gondoljuk át a következő történetet: két állítólagos betörőt, Alizt és Bendegúzt tetten érik egy betörés helyszínéhez közel, és a rendőrség külön-külön kihallgatja őket. Mindketten tudják, hogy ha mindketten beismerik a bűncselekményt, akkor 5 év börtönt kapnak betörésért, de ha mindketten tagadnak, akkor csak 1 évet kapnak lopott holmik birtoklásának enyhébb vádjával. Azonban a rendőrség külön-külön alkut ajánl mindkettőnek: ha a partneredről tanúsítod, hogy egy betörőbanda vezetője, akkor szabadon engednek, míg a partnered 10 évet kap. Ekkor Aliz és Bendegúz az úgynevezett **fogolydilemmá**val (**prisoner's dilemma**) szembesülnek: tanúskodniuk kell vagy tagadniuk? Racionális ágensként Aliz és Bendegúz is a saját várható hasznosságát akarja maximálni. Tegyük fel, hogy Alizt teljességgel hidegen hagyja partnerének a sorsa, így az ő hasznossága a saját börtönben töltött éveinek számával arányosan csökken, függetlenül attól, hogy mi történik Bendegúzzal. Bendegúz teljesen hasonlóan érez. A racionális döntés meghozatalához mindketten megalkotják a következő jutalommátrixot:

	Aliz:tanúskodik	*Aliz:tagad*
Bendegúz:tanúskodik	$A = -5, B = -5$	$A = -10, B = 0$
Bendegúz:tagad	$A = 0, B = -10$	$A = -1, B = -1$

Created by XMLmind XSL-FO Converter.

Aliz a következőképpen elemzi a jutalommátrixot: Tegyük fel, hogy Bendegúz tanúskodik. Ha tanúskodom, 5 évet kapok, és 10 évet, ha nem, ebben az esetben a tanúskodás jobb. Másrészről, ha Bendegúz tagad, akkor 0 évet kapok, ha tanúskodom, és 1 évet, ha tagadok, így a tanúskodás ebben az esetben is jobb. Így mindkét esetben számomra jobb tanúskodni, így ez az, amit tennem kell.

Aliz felfedezte, hogy a *tanúskodás* a játék **domináns stratégiá**ja (**dominant strategy**). Azt mondjuk, hogy egy *p* játékos egy *s* stratégiája **erősen dominál**ja (**strongly dominate**) az *s′* stratégiát, ha az *s* kimenetele *p* számára jobb, mint az *s′* kimenetele, a többi játékos minden stratégiaválasztása esetén. Egy *s* stratégia **gyengén dominál**ja (**weakly dominates**) *s′*-t, ha *s* jobb, mint *s′* legalább egy stratégiaprofilnál, és nem rosszabb a többiben. Egy domináns stratégia olyan stratégia, ami az összes többit dominálja. Egy erősen dominált stratégiát irracionális követni, és ha létezik, irracionális eltérni a domináns stratégiától. Racionális volta miatt Aliz a domináns stratégiát választja. Már csak egy kevés terminológiára van szükségünk a továbbhaladáshoz: egy kimenetelt **Pareto-optimális**nak (**Pareto optimal**) nevezünk,[6] ha nincs másik kimenetel, amit az összes játékos preferálna. Egy kimenetel **Pareto-dominált** (**Pareto dominated**) egy másik kimenetel által, ha az összes játékos a másik kimenetelt preferálná.

Ha Aliz okos és racionális, folytatni fogja az érvelést a következőképpen: Bendegúz domináns stratégiája szintén a tanúskodás. Így ő is tanúskodni fog, és mindketten 5 évet kapunk. Amikor mindegyik játékosnak van domináns stratégiája, ezek kombinációját **domináns stratégiai egyensúly**nak (**dominant strategy equilibrium**) nevezzük. Általában egy stratégiaprofil akkor van **egyensúlyi helyzet**ben (**equilibrium**), ha egyik játékos sem nyer stratégiája váltásával, az összes többi játékos stratégiájának változatlansága mellett. Egy egyensúlyi helyzet lényegében egy **lokális optimum** (**local optimum**) az eljárásmódok terében: egy csúcs teteje lejtőkkel minden dimenzióban, ahol a dimenziók egy játékos stratégiai választásaihoz tartoznak.

A *dilemma* a fogolydilemmában az, hogy az egyensúlyi helyzet kimenetele mindkét játékosnak rosszabb, mint az a kimenetel, amit akkor érnének el, ha mindketten megtagadnák a tanúskodást. Másképpen fogalmazva, az egyensúlyi megoldás kimenete Pareto-dominált a (*tagad, tagad*)-nak a (–1, –1)-es kimenetelével.

Van-e módja Aliznak és Bendegúznak a (–1, –1)-es kimenetelt elérni? Az bizonyosan *megengedett lehetőség* mindkettőjüknek, hogy megtagadják a tanúskodást, de *valószínűtlen* lehetőség. Akármelyik játékos is kezdene elmélkedni azon, hogy a *tagadást* választja, felismeri, hogy jobban teszi, ha a *tanúskodik*. Az egyensúlyi helyzetnek ez a vonzereje.

Fontos

A matematikus John Nash (1928–) bebizonyította, hogy *minden játéknak van egy itt definiált típusú egyensúlyi helyzete*. Ezt a tiszteletére most már Nash-egyensúlynak nevezik. Nyilvánvaló, hogy egy domináns stratégiai egyensúly **Nash-egyensúly** (**Nash equilibrium**) (lásd 17.9. feladat), de nem minden játéknak vannak domináns stratégiái. Nash tétele azt mondja ki, hogy egyensúlyi stratégiák akkor is léteznek, ha nincs domináns stratégia.

A fogolydilemmánál csak a (*tanúskodik, tanúskodik*) stratégiaprofil Nash-egyensúly. Nehezen látható, hogy racionális játékosok hogyan tudják elkerülni ennek a kimenetelét, mivel bármely javasolt nem egyensúlyi megoldásnál legalább egy játékos kísértésben lenne, hogy változtasson a stratégiáján. Játékelméleti kutatók egyetértenek, hogy a Nash-egyensúly fennállása szükséges feltétel egy megoldás fennállásához – bár abban már nem értenek egyet, hogy ez elégséges feltétel-e.

A (*tanúskodik, tanúskodik*) megoldás könnyűszerrel elkerülhető, ha valamilyen módon megváltoztatjuk a játékot (vagy a játékosokat). Módosíthatjuk például egy olyan iterált játékra, amiben a játékosok tudják, hogy újra találkozni fognak (de az döntő, hogy bizonytalanok legyenek abban, hogy hányszor fognak még újra találkozni). Vagy ha az ágensek morális meggyőződése erősíti az együttműködést és az igazságosságot, megváltoztathatjuk a jutalommátrixot úgy, hogy tükrözze az egyes ágensek többiekkel való együttműködésének a hasznosságát. Később látni fogjuk, hogy az ágens olyan megváltoztatása, hogy a számítási kapacitása korlátos, ahelyett hogy teljesen racionális következtetésre lenne képes, szintén befolyásolja a kimenetelt, mivel azt árulhatja el az egyik ágensnek, hogy a másik racionalitása korlátos.

Most nézzünk meg egy játékot, aminek nincs domináns stratégiája. Az Acme nevű videojáték-hardver gyártójának döntenie kell, hogy a következő játékgép DVD-ket vagy CD-ket használjon. Eközben a videojáték-szoftver gyártójának, Bestnek is döntenie kell, hogy DVD-n vagy CD-n adja ki a következő játékát. A nyereség

[6] A Pareto-optimalitás a közgazdász Vilfredo Paretóról (1848–1923) kapta nevét.

Created by XMLmind XSL-FO Converter.

mindkettőjüknek pozitív, ha egyetértenek, és negatív, ha eltérnek, ahogy a következő jutalommátrixból ez látható:

	Acme:dvd	*Acme:cd*
Best:dvd	$A = 9, B = 9$	$A = -4, B = -1$
Best:cd	$A = -3, B = -1$	$A = 5, B = 5$

Fontos

Erre a játékra nincs domináns stratégiai egyensúly, de létezik két Nash-egyensúly: (*dvd*, *dvd*) és (*cd*, *cd*). Tudjuk, hogy ezek Nash-egyensúlyok, mivel akármelyik játékos is vált egyoldalúan egy különböző stratégiára, az rosszabbul fog állni. Most az ágenseknek a problémája a következő: *több elfogadható megoldás is létezik, de ha az egyes ágensek különböző megoldást választanak, akkor a kiadódó stratégiaprofil egyáltalán nem lesz megoldás, és mindkét ágens veszteséget fog elszenvedni.* Hogyan egyezhetnek meg egy megoldásban? Egy válasz erre, hogy mindkettőnek a Pareto-optimális megoldást kell választania (*dvd*, *dvd*); azaz leszűkíthetjük a „megoldás" definícióját az egyértelmű Pareto-optimális Nash-egyensúlyokra, feltéve, hogy létezik ilyen. Minden játéknak van legalább egy Pareto-optimális megoldása, de egy játéknak lehet több egyensúlyi pontja, de lehet, hogy ezek nem egyensúlyi pontok. Például beállíthatjuk a jutalmakat a (*dvd*, *dvd*)-nél 5-re 9 helyett. Ebben az esetben két egyenlő Pareto-optimális egyensúlyi pont létezik. A közöttük való választáshoz az ágensek vagy találgatnak, vagy *kommunikálnak*, ami vagy egy megállapodás kimondását jelenti, ami sorrendezi a megoldásokat a játék megkezdődése előtt, vagy tárgyalást jelent, hogy egy kölcsönösen előnyös megoldás alakuljon ki a játék folyamán (ami egy többlépéses játék részeként megjelenő kommunikációs cselekvések létét jelentené). A kommunikáció így pontosan ugyanazokból az okokból jelenik meg, mint a többágenses tervezésnél a 12. fejezetben. Az ilyen játékokat, ahol a játékosoknak szükséges kommunikálniuk, **koordinációs játék**oknak (**coordination game**s) nevezzük.

Láttuk, hogy egy játéknak lehet több mint egy Nash-egyensúlya: honnan tudjuk, hogy minden játéknak van legalább egy? Az lehetséges, hogy egy játéknak nincs tiszta stratégiájú Nash-egyensúlya. Gondoljunk át például egy tetszőleges *tiszta stratégiájú* profilt a kétujjas snóblijátékhoz (6. szakasz - Többágenses döntések: a játékelmélet rész). Ha az ujjak összértéke páros, akkor *O* szeretne váltani; ha páratlan, akkor pedig *E* szeretne váltani. Így egy tiszta stratégiájú profil nem lehet egyensúlyi, és kevert stratégiákat kell megnéznünk.

De melyik kevert stratégiát? 1928-ban Neumann kifejlesztett egy módszert az optimális kevert stratégia megkeresésére kétszemélyes zérusösszegű játékokra. A **zérusösszegű játék**[7] (**zero-sum game**) olyan játék, amiben a jutalmak a jutalommátrix elemeiben nullára összegződnek. Világos, hogy a snóblijáték ilyen játék. A kétszemélyes zérusösszegű játékoknál tudjuk, hogy a jutalmak egyenlők és ellentétesek, így csak az egyik játékos jutalmait kell figyelembe vennünk, aki a maximáló lesz (ahogy a 6. fejezetben is). A snóblijátéknál az *E* páros játékost választjuk maximálónak, így a jutalommátrixot az $U_E(e, o)$ értékekkel definiálhatjuk – a jutalom *E*-nek, ha *E* *e*-t hajt végre és *O* *o*-t.

Neumann módszerét **maximin technikának** nevezik, és a következő módon működik:

- Tegyük fel, hogy úgy változtatjuk meg a szabályokat, hogy elsőként *E*-nek kötelező felfedni a stratégiáját, majd *O* következik. Így egy fordulókon alapuló játékot kapunk, amire a standard **minimax algoritmus**t alkalmazhatjuk a 6. fejezetből. Tegyük fel, hogy ez egy $U_{E,O}$ kimenetelt ad. Nyilvánvaló, hogy ez a játék *O*-nak kedvez, így a játék igazi hasznossága (*E* szempontjából) legalább $U_{E,O}$. Például ha csak tiszta stratégiákat nézünk, a minimax játékfának a gyökérbeli értéke –3 (lásd 17.11. (a) ábra), így tudjuk, hogy $U \geq -3$.

- Most tegyük fel, hogy a szabályokat úgy változtatjuk meg, hogy elsőként *O*-nak kötelező felfednie a stratégiáját, majd *E* következik. Ennek a játéknak a minimax értéke $U_{E,O}$, és mivel ez a játék *E*-nek kedvez, tudjuk, hogy *U legfeljebb* $U_{E,O}$. Tiszta stratégiáknál az érték +2 (lásd 17.11. (b) ábra), így tudjuk, hogy $U \leq +2$.

[7] Általánosabb az **állandó összegű játék**ok (**constant-sum game**s) fogalma, amelyekben a játékbeli minden elemnél az összeg egy *c* állandót ad. Egy *n* személyes állandó összegű játék egy zérusösszegű játékká konvertálható *c*/*n* kivonásával minden jutalomból. Így a sakk, a tradicionális jutalmakkal, mint 1 a győzelemért, ½ a döntetlenért és 0 a veszteségért, technikailag egy állandó összegű játék *c* = 1-gyel, de könnyen egy zérus- összegű játékká konvertálható, ha minden jutalomból kivonunk ½-et.

Created by XMLmind XSL-FO Converter.

Ezt a két érvelést összekapcsolva láthatjuk, hogy a megoldás valódi hasznossága (U) eleget fog tenni annak, hogy

$U_{E,O} \leq U \leq U_{O,E}$ vagy ebben az esetben $-3 \leq U \leq 2$

Fontos

Az U értékének pontos megállapításához az elemzésünkben át kell térni a kevert stratégiákra. Elsőként vegyük észre a következőt: *ha az első játékos felfedte a stratégiáját, a második nem veszthet azzal, hogy tiszta stratégiát követ*. Az ok egyszerű: ha a második játékos egy [p: *egy*; (1 – p): *kettő*] kevert stratégiát játszik, akkor a várható hasznossága a tiszta stratégiák hasznosságainak, u_{egy}-nek és $u_{kettő}$-nek egy lineáris kombinációja, $(p\ u_{egy} + (1 - p) \cdot u_{kettő})$. Ez a lineáris kombináció nem haladhatja meg az u_{egy} és az $u_{kettő}$ közül a nagyobbat, így a második játékos is játszhat egy tiszta stratégiát.

Ezt az észrevételt emlékezetünkben tartva, a minimax fáknál képzelhetjük azt, hogy a gyökérnél végtelen sok ág van, amelyek az első játékos által választható végtelen sok kevert stratégiához tartoznak. Ezek mindegyike egy kettős elágazású csomóponthoz vezet, amelyek a második játékos tiszta stratégiáihoz tartoznak. Ezeket a végtelen fákat úgy ábrázoljuk véges módon, hogy a gyökérnél egy „parametrizált” választás van:

- Ha E lép először, a helyzet a 17.11. (c) ábrán látható. E a gyökérnél [p: *egy*; (1 – p): *kettő*] szerint játszik, és O p értéke alapján választ egy lépést. Ha O *egy*-et választ, a várható jutalom (E-nek) $2p - 3(1 - p) = 5p - 3$; ha O *kettő*-t választ, a várható jutalom $-3p + 4(1 - p) = 4 - 7p$. Ezeket a jutalmakat mint egyeneseket tüntethetjük fel a grafikonon, ahol p 0 és 1 között lehet az x tengelyen, ahogyan az a 17.11. (e) ábrán látható. O, a minimalizáló, mindig az alsót fogja választani a két egyenes közül, amit az ábrán a vastag egyenes jelez. Így a legjobb, amit E tehet a gyökérnél, hogy p-t a metszéspontnál választja meg, aminek értéke:

$5p - 3 = 4 - 7p \Rightarrow p = 7/12$

A hasznosság E számára ebben a pontban $U_{E,O} = -1/12$.

- Ha O lép először, a helyzet a 17.11. (d) ábrán látható. O a gyökérnél [q: *egy*; (1 – q): *kettő*] szerint játszik, és E a q értéke alapján választ egy lépést. A jutalmak $2q - 3(1 - q) = 5q - 3$ és $-3q + 4(1 - q) = 4 - 7q$.[8] Ismét látható a 17.11. (f) ábrán, hogy a legjobb, amit O a gyökérnél tehet, hogy a metszéspontot választja:

$5q - 3 = 4 - 7q \Rightarrow q = 7/12$

A hasznosság E számára ebben a pontban $U_{O,E} = -1/12$.

Most már tudjuk, hogy a játék valódi hasznossága –1/12 és –1/12 között helyezkedik el, azaz pontosan –1/12! (A tanulság az, hogy ebben a játékban jobb O-nak lenni, mint E-nek.) Továbbá, a valódi hasznosságot a [7/12: *egy*; 5/12: *kettő*] kevert stratégiával lehet elérni, amit mindkét játékosnak érdemes követnie. Ezt a stratégiát a játék **maximin egyensúly**ának nevezik (**maximin equilibrium**), és ez egy Nash-egyensúly. Vegyük észre, hogy az egyensúlyi kevert stratégia minden komponensének ugyanaz a várható hasznossága. Ebben az esetben mind az *egy*-nek, mind a *kettő*-nek ugyanaz a várható hasznossága, –1/12, mint magának a kevert stratégiának.

Fontos

A snóblijátékra vonatkozó eredményünk Neumann általános eredményének egy példája: *minden kétszemélyes, zérusösszegű játéknak van egy maximin egyensúlya, amikor kevert stratégiák lehetségesek.* Továbbá, egy zérusösszegű játékban minden Nash-egyensúly mindkét játékos számára egy maximin. A maximin egyensúly megtalálásának általános algoritmusa zérusösszegű játékokban kissé bonyolultabb, mint amit a 17.11. (e) és 17.11. (f) ábra sugallhatna. Ha n cselekvés lehetséges, akkor egy kevert stratégia egy pont az n dimenziós térben, és az egyenesek hipersíkok lesznek. Az is lehetséges, hogy a második játékos számára bizonyos tiszta stratégiák másokkal dominálnak, így ezek nem optimálisak az első játékos *egyetlen* stratégiája ellen sem. Az összes ilyen stratégia eltávolítása után (amit lehet, hogy ismétlődően kell elvégezni), az optimális választás a gyökérnél a legmagasabb (vagy a legalacsonyabb) metszéspontja a megmaradt hipersíkoknak. Ennek a választásnak a megtalálása **lineáris programozás**i feladat (**linear programming**): egy célfüggvény maximálása lineáris kényszerek mellett. Ilyen problémák szabványos technikákkal megoldhatók polinomiális

[8] Az egy egybeesés, hogy ezek az egyenletek ugyanazok, mint a p-re vonatkozók; az egybeesés amiatt lép fel, mert U_E(*egy*, *kettő*) = U_E(*kettő*, *egy*) = –3. Ez azt is megmagyarázza, hogy az optimális stratégia miért ugyanaz mindkét játékosnál.

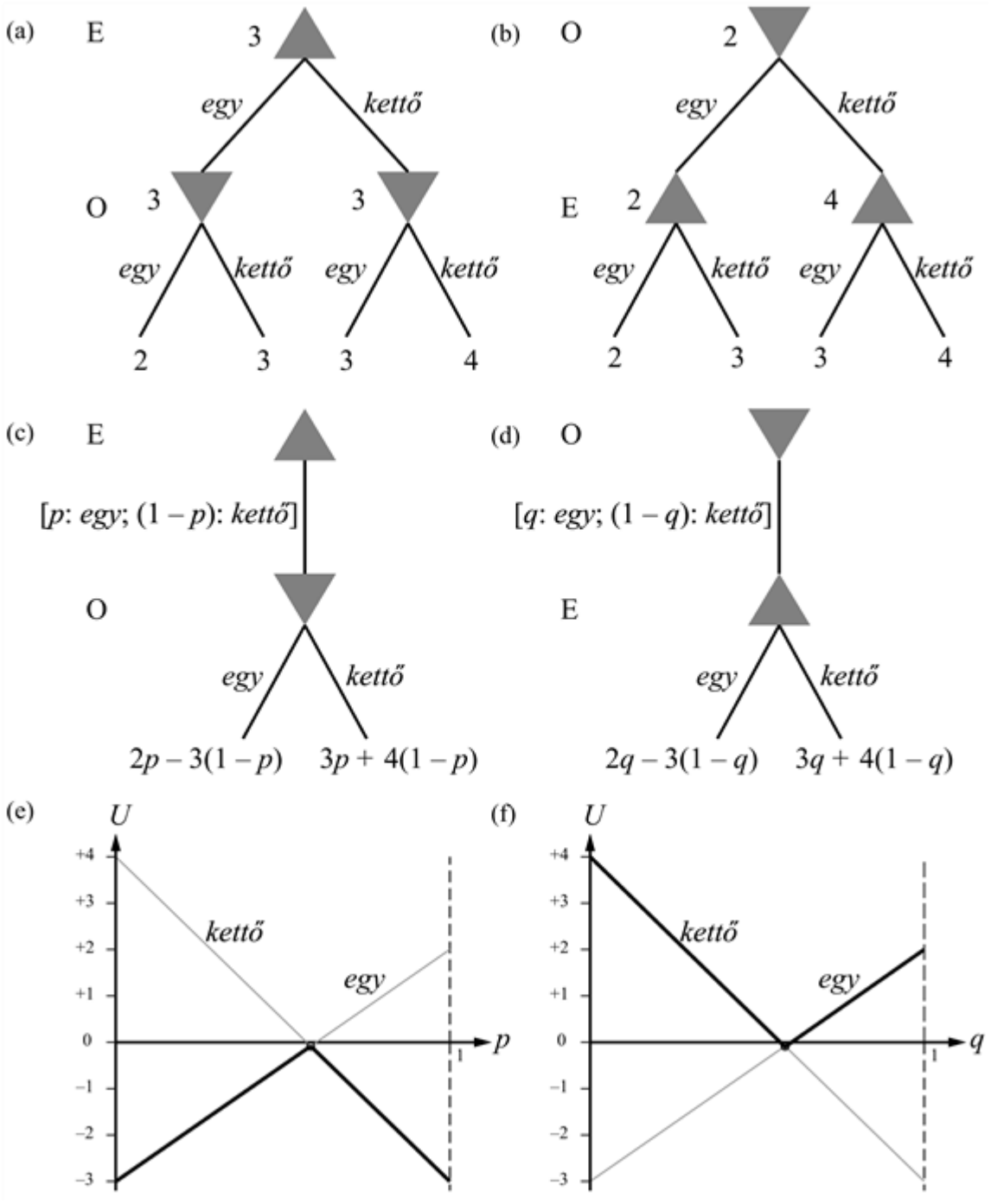
Created by XMLmind XSL-FO Converter.

időkomplexitással a cselekvések számában (és a jutalomfüggvény megadásához használt bitek számában, ha precízek akarunk lenni).

A kérdés továbbra is megválaszolatlan, mit kell egy racionális ágensnek konkrétan *tennie* ha a snóblijáték egy menetét játssza? A racionális ágens le fogja vezetni azt a tényt, hogy a [7/12: *egy*; 5/12: *kettő*] egy maximin egyensúlyi stratégia, azt fogja feltételezni, hogy egy racionális ellenfélnél ez a tudás kölcsönös. Az ágens használhat egy 12 oldalú kockát vagy egy véletlenszám generátort, hogy e szerint a kevert stratégia szerint sorsoljon, amely esetben a várható jutalom E számára –1/12. Vagy az ágens egyszerűen dönthet úgy, hogy mindig *egy*-et vagy *kettő*-t játszik. A várható jutalom bármely esetben –1/12 marad E számára. Érdekes, hogy egy konkrét cselekvés egyoldalú megválasztása nem veszélyezteti a várható jutalmat, de ha a másik ágnes számára ismertté válik egy ilyen egyoldalú döntés, az *már befolyásolja* a várható jutalmat, hiszen az ellenfél ennek megfelelően igazíthatja a stratégiáját.

17.11. ábra - (a) és (b): Minimax játékfák a kétujjas snóblijátékhoz, ha a játékosok a fordulókban tiszta stratégiát játszanak. (c) és (d): Parametrizált játékfák, ahol az első játékos kevert stratégiát játszik. A jutalmak függnek a kevert stratégia (p és q) valószínűségi paramétereitől. (e) és (f): A valószínűségi paraméter bármely konkrét értékénél a második játékos a két cselekvés közül a „jobbikat” fogja választani, így az első játékos kevert stratágiájának az értékét a vastag vonal jelöli. Az első játékos a kevert stratégia valószínűségi paraméterét a metszéspontnál fogja megválasztani.

Created by XMLmind XSL-FO Converter.



A megoldások (azaz Nash-egyensúlyok) megtalálása nem zéró összegű véges játékokra kissé bonyolultabb. Az általános megközelítés kétlépéses: (1) Soroljuk fel a cselekvések összes lehetséges részhalmazát, amelyek kevert stratégiákat alkothatnak. Például elsőként próbálkozzunk az összes olyan stratégiaprofillal, amelyekben mindegyik játékos egyetlen cselekvést használ, aztán azokkal, amelyekben mindegyik játékos egy vagy két cselekvést használ és így tovább. Ez a cselekvések számában exponenciális, és így csak viszonylag kis játékoknál alkalmazható. (2) Minden (1)-ben felsorolt stratégiaprofilnál ellenőrizzük, hogy az egyensúlyi-e. Ez olyan egyenletek és egyenlőtlenségek egy rendszerének a megoldásával érhető el, amelyek hasonlók a zérusösszegű játékban használtakhoz. Két játékosra ezek az egyenletek lineárisak, és alapvető lineáris programozási technikákkal megoldhatók, de három vagy több játékosnál már nem-lineárisak, és esetleg nagyon nehezen oldhatók meg.

Eddig csak egylépéses játékokat vizsgáltunk. A legegyszerűbb többlépéses játék az **ismétlődő játék** (**repeated game**), amelyben a játékosok újra és újra ugyanazzal a választással szembesülnek, de az egyes alkalmakkor már ismert az összes játékos korábbi választásainak a története. Az ismétlődő játéknál egy stratégiaprofil egy

Created by XMLmind XSL-FO Converter.

cselekvésválasztást ad meg mindegyik játékos számára, minden időpontban és a korábbi választások minden lehetséges történetére. Ahogyan az MDF-eknél, a jutalmak az időben additívak.

Gondoljuk át a fogolydilemma ismétlődő változatát. Együtt fog-e működni Aliz és Bendegúz, visszautasítva a tanúskodást, azt tudva, hogy újra fognak találkozni? A válasz a találkozás részleteitől függ. Tegyük fel, hogy Aliz és Bendegúz tudja, hogy pontosan 100 menetben játszanak fogolydilemmásat. Ekkor mindketten tudják, hogy a 100. menet nem lesz ismételt játék – azaz, a kimenetelének nincs hatása jövőbeli menetekre –, és ezért ebben a menetben mindketten a domináns stratégiát, a *tanúskodás*-t választják. De a 100. menet meghatározásával a 99. menetnek nem lehet hatása következő menetekre, így ennek szintén lesz egy domináns stratégiai egyensúlya a (*tanúskodik, tanúskodik*)-nál. Indukcióval mindkét játékos a tanúskodást fogja választani minden menetben, fejenként összesen 500 év börtönt kapva. Eltérő megoldásokat nyerhetünk az interakció szabályainak a megváltoztatásával. Tegyük fel, hogy minden menet után 99% esélye van, hogy a játékosok újra találkoznak.

Ekkor a menetek várható száma még mindig 100, de egyik játékos sem tudja biztosan, hogy melyik menet lesz az utolsó. Ezen feltételek mellett együttműködőbb viselkedés lehetséges. Például a *tagadás* mindegyik játékosnak egyensúlyi stratégia mindaddig, amíg a másik játékos nem választ *tanúskodás*-t. Ezt a stratégiát **örökös büntetés**nek (**perpetual punishment**) nevezik. Tegyük fel, hogy mindkét játékos ezt a stratégiát fogadta el, és ez kölcsönösen ismert. Ekkor mindaddig, amíg egyik játékos sem választotta a *tanúskodás*-t, bármely időpontban a várható jövőbeli teljes jutalom mindegyik játékosnak

$$\sum_{t=0}^{\infty} 0{,}99^t(-1) = -100$$

Az a játékos, aki a *tanúskodás*-t választja, egy 0 pontszámot nyer –1 helyett a soron következő lépésben, de a teljes várható jövőbeli jutalma a következőre változik:

$$0 + \sum_{t=1}^{\infty} 0{,}99^t(-5) = -499{,}5$$

Ezért egyik lépésnél sincs indíték eltérni a (*tagad, tagad*)-tól. Az örökös büntetés a fogolydilemma egy „kölcsönösen biztosított megsemmisítési” stratégiája: alighogy az egyik játékos a tanúskodás mellett dönt, ez biztosítja, hogy mindkét játékos nagy veszteséget fog elszenvedni. De ez csak akkor elrettentő, ha a másik játékos elhiszi, hogy ezt a stratégiát fogadtuk el – vagy legalábbis hogy elfogadhattuk.

Vannak más stratégiák, amelyek megbocsátóbbak. A leghíresebb a „**szemet szemért**” (**tit-for-tat**), ami szerint *tagadás*-sal kell kezdeni, majd megismételni a másik játékos előző lépését az összes elkövetkező lépésben. Így Aliz tagadna, ameddig Bendegúz tagadna, és a Bendegúz tanúskodása utáni lépésben ő is tanúskodna, de visszatérne a tagadáshoz, ha Bendegúz visszatért. Bár nagyon egyszerű, ez a stratégia igen robusztusnak és hatékonynak bizonyult stratégiák széles köre ellen.

Különböző megoldásokat az ágensek megváltoztatásával is kaphatunk, a találkozás szabályainak változtatása helyett. Tegyük fel, hogy az ágensek véges állapotú gépek n állapottal, és egy $m > n$ lépéses játékot játszanak. Az ágensek így nem képesek reprezentálni a hátramaradó lépések számát, így azt ismeretlenként kell kezelni. Így ők nem képesek az indukciós levezetésre, és lehetőségük van a kedvezőbb (*tagad, tagad*) esetet elérni. Ebben az esetben a tudatlanság áldás – vagy inkább az, hogy elhitessük az ellenféllel, hogy tudatlanok vagyunk. Ezekben az ismétlődő játékokban a sikerünk a másik játékos *benyomásán* múlik, hogy nyomulósok vagy mamlaszok vagyunk, és nem a valódi tulajdonságunkon.

Az ismétlődő játékok teljesen általános tárgyalása meghaladja a könyv kereteit, de számos helyzetben megjelennek. Például konstruálhatunk egy szekvenciális játékot, ha a 17.1. ábra 4 × 3-as világába két ágenst behelyezünk. Ha kikötjük, hogy nem következik be mozgás, amikor a két ágens ugyanarra a mezőre próbál lépni (ugyanaz a probléma számos forgalmi útkereszteződésben), akkor bizonyos tiszta stratégiák örökre beragadhatnak. Megoldás, ha mindegyik ágens véletlenszerűen választ az előrefelé lépés és a helyben maradás között; a holtpont gyorsan feloldódik, és mindkét ágens boldog lesz. Pontosan ezt teszik az Ethernet-hálózatokban a csomagütközés feloldására.

Az ismétlődő játékok jelenleg ismert megoldási módszerei a 6. fejezetből ismert többfordulós játékok megoldási módszereihez hasonlítanak abban, hogy egy játékfa építhető a gyökértől lefelé, és megoldható a levelektől felfelé. A fő különbség az, hogy egy gyerekcsomópont értékei feletti egyszerű maximum- vagy minimumképzés

Created by XMLmind XSL-FO Converter.

helyett az algoritmusnak minden szinten meg kell oldania egy kevert stratégiájú játékot, feltételezve, hogy a gyerekcsomópontok már megoldottak, és jól definiált értékük felhasználható.

Az ismétlődő játékokat *részlegesen megfigyelhető* környezetekben **részleges információ**jú (**partial information**) játékoknak nevezik. A példák között találhatunk olyan kártyajátékokat, mint a póker és a bridzs, amelyekben mindegyik játékos a kártyáknak csak egy részét láthatja, de olyan komolyabb „játékokat" is, mint a nukleáris háborús modellek, ahol egyik fél sem ismeri az ellenfél összes fegyverének a helyzetét. Részleges információjú játékokat a hiedelmi állapotok fájának a vizsgálatával oldanak meg úgy, mint az RMMDF-ekben (lásd 17.4. alfejezet). Fontos különbség az, hogy míg a saját hiedelmi állapotok megfigyelhetők, addig az ellenfél hiedelmi állapotai nem. Ilyen játékokra csak mostanában fejlesztettek ki a gyakorlatban használható algoritmusokat. Megoldások születtek a póker bizonyos egyszerűbb változataira, bebizonyítva, hogy a blöffölés valóban racionális választás egy jól kiegyensúlyozott kevert stratégia részeként. Az ilyen tanulmányokból alakult ki az a lényeges meglátás, hogy a kevert stratégiák nemcsak azért hasznosak, mert valakinek a cselekvését megjósolhatatlanná teszik, hanem azért is, mert minimalizálják annak az információnak a mennyiségét, amit az ellenfél a cselekvések megfigyelésével tanulni képes. Érdekes, hogy annak ellenére, hogy a bridzs játszásához való programok tervezői nagyon is tisztában vannak az információ gyűjtésének és elrejtésének a fontosságával, senki sem javasolta véletlenszerű stratégiák használatát.

A játékelmélet felhasználásának az ágenstervezés szélesebb területén eddig számos akadálya volt. Elsőként vegyük észre, hogy egy Nash-egyensúlyban egy játékos teljes mértékben feltételezi azt, hogy az ellenfél egyensúlyi stratégiát játszik. Ez azt jelenti, hogy a játékos semmilyen elvárását sem képes felhasználni a többi játékos valószínű cselekvésével kapcsolatban, és így lehet, hogy olyan fenyegetések ellen való védekezésre vesztegeti el az értékeit, amelyek soha nem is lépnek fel. A **Bayes–Nash-egyensúly** (**Bayes–Nash equilibrium**) fogalma részben ezt a hiányosságot pótolja: ez egy olyan egyensúly, ami megfelel egy játékos a priori valószínűség-eloszlásának a többi játékos stratégiái felett – másképpen fogalmazva, kifejezi egy játékos elvárásait a többi játékos valószínű stratégiájáról. Másodszor, jelenleg nincs jó módszer a játékelméleti és az RMMDF irányítási stratégiák összekapcsolására. Ezek és más problémák miatt a játékelméletet elsődlegesen inkább olyan környezetek *elemzésére* használják, amelyek egyensúlyban vannak, a környezetben lévő ágensek *irányítása* helyett. Hamarosan látni fogjuk, hogy a játékelmélet hogyan segítheti a környezetek tervezését.

7. Működési mód tervezés

Az előző alfejezetben azt a kérdést próbáltuk megválaszolni, hogy „Ha adott egy játék, mi a racionális stratégia?" Ebben az alfejezetben azt kérdezzük, hogy „Ha az ágensek racionálisak, milyen játékot tervezzünk?" Pontosabban, egy olyan játékot szeretnénk tervezni, aminek a megoldása az egyes ágensek által követett saját racionális stratégiáik együttese, és ez egy globális hasznosságfüggvény maximálását eredményezi. Ezt a problémát **működési mód tervezés**nek (**mechanism design**) nevezik, vagy néha **inverz játékelmélet**nek (**inverse game theory**). A működési mód tervezés a közgazdaság-tudományok és a politikai tudományok a lényege. Ágensek együttesénél ez annak lehetőségét hordozza magában, hogy játékelméleti működési módokat használva ügyes rendszereket hozunk létre korlátoltabb rendszerek együtteséből – még nem együttműködő rendszerekből is –, nagyon hasonlóan ahhoz, ahogy emberek csoportjai olyan célokat tudnak elérni, amik messze túl vannak az egyéni lehetőségeken.

A működési mód tervezés példái között olcsó repülőjegyek elárverezése, TCP-csomagok számítógépek közötti továbbításához megfelelő útvonal keresése, szigorló orvosok kórházakhoz rendelése, illetve annak eldöntése szerepel, hogy robotfocisták hogyan működjenek együtt a csapattársaikkal. A működési mód tervezés az 1990-es évektől lépett ki az egyetemek falai közül, amikor számos ország, a műsorszóró frekvenciasávok elárverezésének a problémájával szembesülve, dollárszázmilliókat vesztett a lehetséges bevételeiből a gyenge működési mód tervezés eredményeképpen. Formálisan a **működési mód** (**mechanism**) tartalmaz (1) egy nyelvet az ágensek által választható megengedett stratégiák (potenciálisan végtelen) halmazának a leírására és (2) egy G kimeneteli szabályt, ami a jutalmakat határozza meg az ágenseknek megengedett stratégiák egy adott stratégiaprofilja esetén.

Első ránézésre a működési mód tervezés problémája triviálisnak tűnik. Tegyük fel, hogy az U globális hasznosságfüggvény dekomponálódik egyéni U_i ágens hasznosságfüggvények valamely halmazára úgy, hogy $U = \Sigma_i U_i$. Ekkor mondhatnánk azt, hogy ha mindegyik ágens maximálja a saját hasznosságát, ez biztosan a globális hasznosság automatikusan maximálásához fog vezetni. (Például a kapitalizmus egyszeregye szerint, ha egy társadalomban mindenki megpróbál gazdagodni, a társadalom összgazdagsága növekedni fog.) Sajnos ez nem működik. Az egyes ágensek cselekvései befolyásolhatják a többi ágens jólétét oly módon, hogy a globális hasznosság csökken. Erre példa a **közlegelő tragédiája** (**tragedy of commons**), amely helyzetben az egyéni

Created by XMLmind XSL-FO Converter.

gazdálkodók az összes lábasjószágukat ingyen legeltetik a város közlegelőjén, ezért a közlegelő tönkremegy, és az összes gazdálkodó nagy veszteséget szenved el. Mindegyik gazdálkodó egyénileg racionálisan cselekedett, azzal érvelve, hogy a közlegelő ingyenes, és azzal, hogy bár a közlegelő használata a tönkremeneteléhez vezet, a használatának a mellőzése ezt nem befolyásolja (hiszen mások úgyis használnák). Hasonló érvelések alkalmazhatók a szennyező anyagok kibocsátása esetén a légkör és az óceánok használatára is.

Az ilyen problémák szabványos megközelítése a működési mód tervezésben az, hogy a köztulajdon használatáért mindegyik ágensnél költséget számítunk fel. Általánosabban, azt kell biztosítanunk, hogy minden **külsőség** (**externalities**) – olyan globális hasznosságot befolyásoló tényező, amely az egyéni ágensek tranzakcióiban nincsenek elismerve – különállóan nevesítve megjelenjen. Ebben a nehéz a helyes árak megállapítása. Ennek a megközelítésnek a végletes formája egy olyan működési mód megalkotását jelenti, amelyben mindegyik ágenstől valójában a globális hasznosság maximálását követeljük meg. Ez megoldhatatlanul nehéz feladat az ágens számára, aki sem megbecsülni nem tudja a világ jelenlegi állapotát, sem megfigyelni a cselekvéseinek az összes többi ágensre gyakorolt hatását. A működési mód tervezés ezért olyan működésmódok megtalálására összpontosít, amelyeknél az egyes ágensek döntési problémája könnyen megoldható.

Gondoljuk át először az árveréseket. Az árverés a legáltalánosabb formájában egy működésmód bizonyos áruknak egy ajánlattevő csoport tagjai számára történő eladására. Az árajánlatok, a stratégiák és a kimenetel meghatározza, hogy ki kapja az árukat és mennyit fizet. A mesterséges intelligenciában például az árverések akkor jelennek meg, amikor ágensek egy csoportja eldönti, hogy együttműködjenek-e egy közös terven. Hunsberger és Grosz megmutatta, hogy ez hatékonyan elérhető árveréssel, amiben az ágensek ajánlatokat tesznek a közös tervbeli szerepükre (Hunsberger és Grosz, 2000). Egyelőre olyan árveréseket tekintünk át, amelyekben (1) egyetlen áru van, (2) mindegyik ajánlattevőnek van egy v_i hasznosságértéke az árura, és (3) ezek az értékek csak az ajánlattevő számára ismertek. Az ajánlattevők megteszik a b_i ajánlataikat, és a legmagasabb ajánlat nyeri el az árukat, de a működésmód határozza meg, hogyan tehetők meg az ajánlatok, és mi a győztes által fizetendő ár (ami nem szükségszerűen b_i). Az árverések legismertebb típusa az **angol árverés** (**English auction**), amelyben az árverező mindaddig növeli az áruk árát, amíg csak egyetlen ajánlattevő marad, ellenőrizve közben, hogy vajon az ajánlattevők érdekeltek-e még. Ennek a működésmódnak az a tulajdonsága, hogy a legnagyobb v_i értékkel bíró ajánlattevő nyeri el az árukat $b_m + d$ áron, ahol b_m a legmagasabb ajánlat az összes többi játékos között, és d az árverező növekménye az ajánlatok között.[9] Az angol árverésnél az ajánlattevőknek egyszerű domináns stratégiájuk van: addig tegyünk árajánlatokat, ameddig a jelenlegi költség a személyes érték alatt van. Emlékezzünk arra, hogy a „domináns” azt jelenti, hogy a stratégia minden más stratégia ellen működik. Ez viszont azt jelenti, hogy egy játékos a többi stratégiától függetlenül választhatja ezt. Ezért a játékosoknak nem kell időt és energiát vesztegetniük a többi játékos stratégiáján való elmélkedéssel. Egy működésmódot **stratégiamentes**nek (**strategy-proof**) nevezünk, ha a játékosoknak van domináns stratégiája, ami magában foglalja a valódi indítékok felfedését is.

Az angol árverés hátrányos tulajdonsága a nagy kommunikációs költség, így vagy egy szobában kell az árverésnek lezajlania, vagy az összes ajánlattevőnek nagysebességű, biztonságos kommunikációs csatornával kell rendelkeznie. Egy kevesebb kommunikációt igénylő alternatív működésmód a **zárt ajánlatú árverés** (**sealed bid auction**). Ebben minden ajánlattevő egyetlen ajánlatot tesz, és ezt közli az árverezővel, és a legnagyobb ajánlat győz. Ennek a működésmódnak az esetében az a stratégia, aminek az árajánlata a valódi érték, már nem domináns. Ha valakinek az értéke v_i, és az elvárása szerint az összes többi játékos maximális ajánlata b_m lesz, akkor az árajánlatának a v_i és $b_m + \varepsilon$ közül az alacsonyabbnak kell lennie. A zárt ajánlatú árverés két hátránya, hogy előfordulhat, hogy a legnagyobb v_i értékkel bíró játékos nem kapja meg az árukat, illetve az, hogy a játékosoknak fáradozniuk kell a többi játékos stratégiáján való elmélkedéssel.

A zárt ajánlatú árverés szabályainak kis változtatásával adódik a **zárt ajánlatú második áras árverés** (**sealed bid second-price auction**), ami **Vickrey-árverés** (**Vickrey auction**) néven is ismert.[10] Az ilyen árveréseknél a győztes a *második* legmagasabb ajánlati árat fizeti a saját árajánlatának a kifizetése helyett. Ez az egyszerű módosítás teljesen kiküszöböli a szabványos (avagy **első áras**, **first-price**) zárt ajánlatú árverésnél szükséges összetett mérlegeléseket, mivel ekkor a domináns stratégia saját értékű árajánlat. Ennek belátásához vegyük észre, hogy bármely játékos tekintheti az árverést egy kétszemélyes játéknak, figyelmen kívül hagyva az összes játékost, kivéve önmagát és a többi játékos közül a legmagasabb árajánlatot tevőt. Az *i.* játékos hasznossága a saját b_i ajánlata, a v_i értéke és a többi játékos legjobb b_m ajánlata szempontjából

[9] Valójában van egy kis esélye, hogy a legnagyobb v_i értékkel bíró játékosnak nem sikerül megszereznie az árukat abban az esetben, amikor $b_m < v_i < b_m + d$. Annak az esélyét, hogy ez megtörténjen, tetszőlegesen kicsivé lehet tenni a d növekmény csökkentésével.
[10] William Vickreyről elnevezve, aki az 1996-os közgazdasági Nobel-díj kitüntetettje.

Created by XMLmind XSL-FO Converter.

$$u_i(b_i, b_m) = \begin{cases} (v_i - b_m) & \text{ha } b_i > b_m \\ 0 & \text{egyébként} \end{cases}$$

Annak belátásához, hogy a $b_i = v_i$ domináns stratégia, vegyük észre, hogy amikor $(v_i - b_m)$ pozitív, akkor bármelyik ajánlat optimális, ami megnyeri az árverést, és v_i megtétele nevezetesen megnyeri az árverést. Másrészt, amikor $(v_i - b_m)$ negatív, akkor bármelyik ajánlat optimális, ami elveszti az árverést, és v_i megtétele nevezetesen elveszti az árverést. Így a v_i árajánlat optimális az összes lehetséges b_m értékre, és valójában ez csak a v_i árajánlatra teljesül. Az egyszerűsége miatt, továbbá mind az árverező, mind az ajánlattevő számára jelentkező minimális számítási igénye miatt a Vickrey-árverést széles körben használják elosztott MI-rendszerek létrehozásában.

Most gondoljuk át az internet forgalomirányítási problémáját. A játékosok a kapcsolódási háló gráfjában az élekhez tartoznak. Minden játékos ismeri egy üzenet küldésének a költségét a saját élén át; ha nincs küldendő üzenet, akkor a költség 0. A cél az, hogy egy üzenet számára megtaláljuk a legolcsóbb utat a kiindulástól a célig, ahol a teljes út költsége az egyes élek költségének az összege. A 4. fejezet számos algoritmust ad az élköltségek ismeretében a legrövidebb út kiszámítására, így minden, amit tennünk kell, az, hogy az egyes ágenseket rávegyük, hogy jelentsék a valódi c_i költségüket. Sajnos, ha csupán megkérdezzük az ágenseket, akkor magas költséget fog jelenteni, hogy arra bátorítson, küldjük át az üzenetet máshol. Ki kell fejlesztenünk egy stratégiamentes működési módot. Egy ilyen működési mód, ha minden játékosnak fizetünk egy p_i jutalmat, ami a legrövidebb, az i-edik élet nem tartalmazó út hossza mínusz a legrövidebb út (keresési algoritmus által kiszámolt) hossza, ahol az i-edik él költségét 0-nak tételezzük fel.

p_i = Hossz(út c_i = ∞-nél) – Hossz(út c_i = 0-nál)

Megmutatható, hogy e mellett a működési mód mellett a domináns stratégia mindegyik játékosnak a c_i őszinte jelentése, illetve, hogy ennek megtétele a legolcsóbb utat fogja eredményezni. E kívánatos tulajdonság ellenére az itt vázolt működésmódot a gyakorlatban nem használják a nagy kommunikációs és központi számítási költség miatt. A működési mód tervezőnek kommunikálnia kell az összes n játékossal, és aztán egy optimalizációs problémát kell megoldania. Ez megérheti, ha a költségek eloszlanának sok üzenet felett, de valós hálózatokban a c_i költségek folyamatosan változnának a forgalom torlódása, a gépek üzemképtelenné válása és belépése miatt. Eddig teljesen kielégítő megoldást még nem fejlesztettek ki.

8. Összefoglalás

A fejezet megmutatta, hogyan használjuk a világról meglévő tudásunkat döntések meghozatalára, még akkor is, amikor egy cselekvés kimenetelei bizonytalanok, és a cselekvésért járó jutalmat nem arathatjuk le csak több cselekvés után. A legfőbb pontok:

- A szekvenciális döntési problémákat bizonytalan környezet esetén, más néven **Markov döntési folyamat**okat vagy **MDF**-ket (**Markov decision process**, **MDP**), teljesen definiálja egy cselekvések valószínűségi kimenetelét meghatározó **állapotátmenet-modell** (**transition model**) és az egyes állapotokbeli jutalmakat meghatározó **jutalomfüggvény** (**reward function**).

- Egy állapotsorozat hasznossága a sorozathoz tartozó összes jutalom összege, ami lehet az idővel leszámított. Egy MDF megoldása egy **eljárás** (**policy**), ami mindegyik ágens által elérhető állapothoz hozzárendel egy döntést. Egy optimális eljárás maximálja a végrehajtása során fellépő állapotsorozatok hasznosságát.

- Egy állapot hasznossága az azon állapotsorozatok várható hasznossága, amelyek ebből az állapotból kezdődnek, és egy optimális eljárás végrehajtása esetén lépnek fel. Az MDF-k megoldására szolgáló **értékiteráció** (**value iteration**) algoritmusa a működése során az egyenleteket iteratívan oldja meg, az egyes állapotok hasznosságait a szomszédai hasznosságához kapcsolva.

- Az **eljárásmód-iteráció** (**policy iteration**) felváltva kiszámolja az állapotoknak az aktuális eljáráshoz tartozó hasznosságát, majd az aktuális hasznosságokhoz képest javít az aktuális eljáráson.

- A részlegesen megfigyelhető MDF-ek vagy RMMDF-ek sokkal nehezebben oldhatók meg, mint az MDF-ek. Megoldásuk során konvertáljuk őket egy olyan MDF-be, amit hiedelmi állapotok folytonos terében definiálunk. Az optimális viselkedés az RMMDF-ekben magában foglalja az információgyűjtést a bizonytalanság csökkentése céljából, és így a jövőbeli jobb döntések meghozatala miatt is.

Created by XMLmind XSL-FO Converter.

- Az RMMDF-környezetekre egy döntéselméleti ágens hozható létre. Az ágens egy **dinamikus döntési hálót** (**dynamic decision network**) használ az állapotátmenet-modell és megfigyelési modell ábrázolására, a hiedelmi állapotának frissítésére, illetve lehetséges cselekvési sorozatok előre elgondolásához.

- A **játékelmélet** (**game theory**) olyan szituációkban írja le az ágensek racionális viselkedését, amelyekben több ágens egyidejű interakciója van jelen. A játékok megoldásai Nash-egyensúlyok – olyan stratégiaprofilok, amelyekben egyik ágensnek sincs indítéka a megadott stratégiától eltérni.

- A **működési mód tervezés** (**mechanism design**) használható fel az ágensek interakcióját megadó olyan szabályok megállapítására, amelyek egy globális hasznosságot az egyéni racionális ágensek tevékenységén keresztül maximálnak. Néha léteznek olyan működésmódok, amelyek ezt a célt úgy érik el, hogy az egyes ágenseknek nem kell meggondolniuk a többi ágens által hozott döntéseket.

Az MDF-ek és az RMMDF-ek világára vissza fogunk térni a 21. fejezetben, amikor a megerősítéses tanulás módszerét tanulmányozzuk, ami lehetővé teszi az ágens számára, hogy a tapasztalatai alapján javítsa a viselkedését szekvenciális, bizonytalan környezetekben.

8.1. Irodalmi és történeti megjegyzések

Richard Bellman kezdeményezte a szekvenciális döntési problémák modern megközelítési módját, és javasolta a dinamikus programozás módszerét általában és külön az értékiterációval kapcsolatban is (Bellman, 1957). Egy PhD-disszertációban Ron Howard vezette be az eljárásmód-iterációt és az átlagjutalom fogalmát végtelen horizontú problémák megoldására (Howard, 1960). Számos további eredményt Bellman és Dreyfus vezettek be (Bellman és Dreyfus, 1962). A módosított eljárásmód-iteráció van Nunentől, illetve Putermantól és Shintől származik (van Nunen, 1976; Puterman és Shin, 1978). Az aszinkron eljárásmód-iterációt Williams és Baird elemezték, tőlük származik a (17.9) egyenletbeli korlát is az eljárásmód-veszteségre (Williams és Baird, 1993). A leszámítolás technikáját stacionárius preferenciák segítségével megfogalmazva Koopmans dolgozta ki (Koopmans, 1972). A (Bertsekas, 1987; Puterman, 1994; Bertsekas és Tsitsiklis, 1996) könyvek precíz bevezetést adnak a szekvenciális döntési problémákhoz. Papadimitriou és Tsitsiklis az MDF-k számítási komplexitásáról közölnek eredményeket (Papadimitriou és Tsitsiklis, 1987).

Sutton és Watkins nagy hatású munkái, amelyek az MDF-ek megoldására szolgáló megerősítéses tanulási módszerekre irányultak, jelentős szerepet játszottak az MDF-ek megismertetésében az MI-kutatás területén (Sutton és Watkins, 1988), hasonlóan Barto és társainak későbbi áttekintéséhez (Barto és társai, 1995). (Werbos [1977] korábbi munkája számos hasonló ötletet tartalmazott, de nem vált ilyen széles körben elfogadottá.) Az első kapcsolatot az MDF-ek és a mesterséges intelligencia tervezési problémái között Sven Koenig teremtette meg (Koenig, 1991), aki megmutatta hogyan nyújtanak a valószínűségi Strips operátorok egy tömör ábrázolást az állapotátmenet-modellek számára. (Lásd még Wellman, 1990b.) Dean és munkatársai, illetve Tash és Russell munkáikban a nagy állapotterek miatti kombinatorikus problémákon kíséreltek meg úrrá lenni korlátozott keresési horizont és absztrakt állapotok felhasználásával (Dean és társai, 1993); Tash és Russell, 1994). Az információ értékén alapuló heurisztikák felhasználhatók olyan állapottérrészek kiválasztására, ahol a horizont lokális kiterjesztése a döntés minőségének jelentős javulását eredményezi. Ezt a megközelítést használva az ágensek az időkényszerekhez igazíthatják az erőfeszítéseiket, és érdekes viselkedést alakíthatnak ki, mint például a jól ismert „kitaposott utak” használatát az állapottérben való gyors útkeresésnél, elkerülve az optimális döntések újbóli kiszámítását az egyes pontokban. Boutilier és munkatársai mostani munkái propozicionális és elsőrendű kifejezéseken alapuló *szimbolikus* reprezentációk dinamikus programozásbeli felhasználására összpontosultak, mind az állapotátmenet-modellek, mind az értékfüggvények esetében (Boutilier és társai, 2000; 2001).

Azt a megfigyelést, hogy a részlegesen megfigyelhető Markov döntési folyamatok átalakíthatók hagyományos Markov döntési folyamatokká hiedelmi állapotok felhasználásával, Astrom közli (Astrom, 1965). Az első teljes algoritmust a részlegesen megfigyelhető Markov döntési folyamatok egzakt megoldására Edward Sondik javasolta a PhD-disszertációjában (Sondik, 1971). (Egy későbbi folyóiratcikk [Smallwood és Sondik, 1973] tartalmaz néhány hibát, de jobban elérhető.) Az RMMDF-el kapcsolatos aktuális eredmények áttekintését adja Lovejoy (Lovejoy, 1991). Az első jelentős eredmény az MI területén a Witness-algoritmus volt (Cassandra és társai, 1994; Kaelbling és társai, 1998), ami egy javított változata az RMMDF értékiterációjának. Hamarosan következtek más algoritmusok, köztük egy Hansentól származó megközelítés, ami egy eljárásmódot fokozatosan hoz létre egy végesállapotú automata formájában (Hansen, 1998). Az eljárásmód ezen ábrázolásában a hiedelmi állapotok közvetlenül az automata konkrét állapotaihoz tartoznak. Közelítőleg optimális eljárásmódok az RMMDF-ekhez olyan eljárással hozhatók létre, ahol az előrefelé keresést a

Created by XMLmind XSL-FO Converter.

lehetséges megfigyelések és cselekvések kimeneteleinek mintavételezésével kombináljuk (Kearns és társai, 2000; Ng és Jordan, 2000). További, az RMMDF-ekre vonatkozó munkákat a 21. fejezetben tárgyalunk.

A dinamikus döntési hálókat használó ágensarchitektúra alapötleteit Dean és Kanazawa javasolták (Dean és Kanazawa, 1989a). Dean és Wellman könyve, a *Planning and Control,* sokkal részletesebb kapcsolatot teremt a DBH/DDH-modellek és a szűrés klasszikus szabályozáselméletbeli szakirodalma között (Dean és Wellman, 1991). Tatman és Shachter mutatta meg, hogy hogyan alkalmazhatók dinamikus programozási algoritmusok DDH-modellekre (Tatman és Shachter, 1990). Russell különböző módszereket közöl ilyen ágensek felskálázására, és számos nyitott kutatási kérdést fogalmaz meg (Russell, 1998).

A játékelmélet kezdetei a 17. századra nyúlnak vissza, Christiaan Huygens és Gottfried Leibniz javaslataihoz, a versengő és együttműködő emberi interakciók tudományos és matematikai vizsgálatára. A 19. század folyamán számos vezető közgazdász alkotott egyszerű matematikai példákat, hogy a versengési helyzetek konkrét eseteit elemezze. Az első formális eredmények a játékelméletben Zermelótól származnak (Zermelo, 1913) (aki egy évvel korábban a minimax keresés egy – bár hibás – változatát javasolta játékokra). Emil Borel vezette be a kevert stratégia fogalmát (Borel, 1921). Neumann János bizonyította be, hogy minden kétszemélyes, zérusösszegű játéknak van egy maximin egyensúlya a kevert stratégiák körében és egy jól definiált értéke (Neumann, 1928). Neumann együttműködése a közgazdász Oskar Morgensternnel vezetett a *Theory of Games and Economic Behavior* c. könyv megjelenéséhez 1944-ben, ami a játékelmélet meghatározó könyve. A könyv megjelenése a háború miatti papírhiány miatt késett, míg végül a Rockefeller család egy tagja személyesen támogatta a könyv kiadását.

John Nash, 1950-ben, 21 évesen jelentette meg elgondolásait az általános játékbeli egyensúlyi helyzetekkel kapcsolatban. Az egyensúlyi megoldás általa megadott definíciója, bár Cournot munkáján alapult (Cournot, 1838), úgy vált ismertté, mint Nash-egyensúly. Az 1959-től fellépő skizofréniája miatti hosszú szünet után Nash 1994-ben közgazdasági Nobel-díjat kapott (megosztva Reinhart Seltennel és Harsányi Jánossal).

A Bayes–Nash-egyensúlyt Harsányi közli (Harsányi, 1967) és Kadane és Larkey elemzi (Kadane és Larkey, 1982). A játékelmélet ágensek vezérlésében történő felhasználásának bizonyos kérdéseit Binmore tárgyalja (Binmore, 1982).

A fogolydilemmát mint tantermi példát Albert W. Tucker alkotta meg és Axelrod tárgyalja részletesen (Axelrod, 1985). Az ismétlődő játékokat Luce és Raiffa vezette be (Luce és Raiffa, 1957) mint részleges információjú játékokat (Kuhn, 1953).

Az első gyakorlati algoritmust részleges információjú játékokra az MI területén Koller és munkatársai fejlesztették ki (Koller és társai, 1996); Koller és Pfeffer cikke olvasmányos bevezetést ad a terület általános részeihez és egy működő rendszert ír le szekvenciális játékok ábrázolására és megoldására (Koller és Pfeffer, 1997). A játékelméletet és az MDF-eket a Markov-játékok elméletében egységesíti Littman (Littman, 1994). Shapley valójában már Bellman előtt közölte az értékiteráció algoritmusát (Shapley, 1953), de az eredményei nem váltak ismertté, talán amiatt, hogy nem a Markov-játékok keretei között voltak megfogalmazva. A játékelmélet általános tárgyalását tartalmazzák a (Myerson, 1991; Fudenberg és Tirole, 1991; Osborne és Rubinstein, 1994) művek.

A köztulajdon tragédiáját, ami működési mód tervezés egy motiváló problémája, Hardin mutatta be (Hardin, 1968). Hurwicz alkotta meg a működési mód tervezés matematikai alapjait (Hurwicz, 1973). Milgrom egy általa tervezett, akár több milliárd dolláros tételekre is szánt árverés működésmódot közöl (Milgrom, 1997). Az árverések a tervezésben (Hunsberger és Grosz, 2000) és az ütemezésben is felhasználhatók (Rassenti és társai, 1982). Varian rövid áttekintést ad a számításelméleti szakirodalomhoz kapcsolódó hivatkozásokkal együtt (Varian, 1995), a Rosenschein és Zlotkin szerzőpárostól pedig egy könyv terjedelmű tárgyalás jelent meg elosztott MI-alkalmazásokról (Rosenschein és Zlotkin, 1994). Az elosztott MI-ben más nevek alatt folynak kapcsolódó munkák, ideértve az együttes intelligenciát (Collective Intelligence), (Tumer és Wolpert, 2000) és a piacalapú szabályozást (market-based control) (Clearwater, 1996). Az árverések számítási vonatkozásairól gyakran az ACM Conferences on Electronic Commerce-en jelennek meg cikkek.

8.2. Feladatok

17.1.

Created by XMLmind XSL-FO Converter.

A 17.1. ábra által mutatott 4 × 3-as világ esetén számítsa ki mely négyzetek és milyen valószínűséggel érhetők el az [1, 1]-ből a [*Fel*, *Fel*, *Jobbra*, *Jobbra*, *Jobbra*] cselekvéssorozattal. Magyarázza el, hogyan kapcsolódik ez a számítás egy rejtett Markov-modell előre elgondolásához („projecting”).

17.2.

Tegyük fel, hogy egy állapotsorozat hasznosságát a sorozat állapotaiban kapott jutalmak maximumaként definiáljuk. Mutassuk meg, hogy ez a hasznosságfüggvény nem eredményez stacionárius preferenciákat az állapotszekvenciák között. Lehetséges-e mégis olyan hasznosságfüggvényt definiálni az állapotokon, hogy a MVH-döntéshozatal optimális viselkedést adjon?

17.3.

Pontosan konvertálható-e egy véges keresési probléma Markov döntési problémává úgy, hogy az utóbbi optimális megoldása az előzőnek is optimális megoldása legyen? Ha igen, magyarázza el pontosan, hogyan konvertálja a problémát, és hogyan konvertálja a megoldást visszafelé; ha nem, akkor magyarázza el, hogy miért nem (azaz adjon ellenpéldát).

17.4.

Képzeljünk el egy leszámítolatlan MDF-et három állapottal, (1, 2, 3) és sorrendben a következő jutalmakkal –1, –2, 0. A 3-as állapot egy végállapot. Az 1-es és 2-es állapotokban két lehetséges cselekvés van: *a* és *b*. Az állapotátmenet-modell a következő:

- Az 1-es állapotban az *a* cselekvés az ágenst a 2-es állapotba mozgatja 0,8 valószínűséggel, míg 0,2 valószínűséggel helyben hagyja.
- Az 2-es állapotban az *a* cselekvés az ágenst az 1-es állapotba mozgatja 0,8 valószínűséggel, és 0,2 valószínűséggel helyben hagyja.
- Mind az 1-es, mind a 2-es állapotban a *b* cselekvés az ágenst a 3-as állapotba mozgatja 0,1 valószínűséggel, és 0,9 valószínűséggel helyben hagyja.

Válaszolja meg a következő kérdéseket:

a. Mi határozható meg *kvalitatívan* az optimális eljárásmódról az 1-es és a 2-es állapotokban?

b. Alkalmazzon eljárásmód-iterációt az optimális eljárásmód, illetve az 1-es és a 2-es állapotok értékének a meghatározására. Minden lépést teljesen illusztráljon. Tegyük fel, hogy a kezdeti eljárásmód mindkét állapotban a *b* cselekvés.

c. Mi történik az eljárásmód-iterációval, ha a kezdeti eljárásmód mindkét állapotban az *a* cselekvés? Segít a leszámítolás? Függ-e az optimális eljárásmód a leszámítolási tényezőtől?

17.5.

Alkalmanként az MDF-ek megadása tartalmaz egy $R(s, a)$ jutalomfüggvényt, ami a véghezvitt cselekvéstől függ, vagy egy $R(s, a, s')$ jutalomfüggvényt, ami még a kimeneteli állapottól is függ.

a. Írja fel a Bellman-egyenleteket ezt a formalizmust használva.

b. Mutassa meg, hogyan lehet egy $R(s, a, s')$ jutalomfüggvényt tartalmazó MDF-et átalakítani egy másik, $R(s, a)$ jutalomfüggvényt tartalmazó MDF-be úgy, hogy az optimális eljárásmódok az új MDF-ben pontosan megfeleljenek az eredeti MDF-beli optimális eljárásmódoknak.

c. Most ugyanígy konvertálja az $R(s, a)$-t tartalmazó MDF-eket $R(s)$-t tartalmazó MDF-ekbe.

17.6.

Tekintsük a 17.1. ábrán látható 4 × 3-as világot.

Created by XMLmind XSL-FO Converter.

a. Valósítson meg egy olyan környezetszimulátort erre a környezetre, amelyben a környezet földrajzi sajátosságai könnyen megváltoztathatók. Ehhez egy bizonyos forráskód már elérhető a könyv forráskódtárában a világhálón.

b. Hozzon létre egy eljárásmód-iterációt használó ágenst, és mérje meg a teljesítményét a környezetszimulátorban különböző kiindulási állapotokból. Végezzen nagyszámú kísérletet mindegyik kiindulási állapotból, és hasonlítsa össze a futásonként kapott összjutalmak átlagát az állapot hasznosságával (amit a létrehozott algoritmus határozott meg).

c. Kísérletezzen a környezet méretének a megnövelésével. Hogyan változik az eljárásmód-iteráció futási ideje a környezet méretének a változásával?

17.7.

A 17.1. ábrán mutatott környezetre keresse meg az $R(s)$ függvényre az összes küszöbértéket, azaz azokat az értékeket, amelyek átlépésekor az optimális eljárásmód megváltozik. Ehhez egy módszer szükséges az optimális eljárásmód meghatározásához és egy rögzített $R(s)$ esetén az értékének a kiszámításához. (*Segítség*: bizonyítsa be, hogy az érték minden rögzített eljárásmódnál $R(s)$-sel lineárisan változik.)

17.8.

Ebben a feladatban kétszereplős MDF-ekkel foglalkozunk, amelyek a zérusösszegű, többfordulós játékokhoz tartoznak, mint amilyenek a 6. fejezetben szerepelnek. Legyen a két játékos A és B, és legyen $R(s)$ az A játékos jutalma az s állapotban. (B jutalma mindig egyenlő és ellentétes előjelű.)

a. Legyen $U_A(s)$ az s állapot hasznossága, amikor A lépése következik s-ben, és legyen $U_B(s)$ az s állapot hasznossága, amikor B következik s-ben. Az összes jutalom és hasznosság A szempontjából van megfogalmazva (pontosan úgy, ahogyan a minimax játékfában). Írja fel az $U_A(s)$-t és az $U_B(s)$-t definiáló Bellman-egyenleteket.

b. Magyarázza el, hogyan végezhet kétszemélyes értékiterációt ezekkel az egyenletekkel, és adjon meg egy alkalmas leállási kritériumot.

c. Tekintsük a 6.14. ábrán látható játékot a 8.2. szakasz - Feladatok részben. Rajzolja fel az állapotteret (a játékfa helyett), folytonos vonallal mutatva A mozgásait és szaggatott vonallal B mozgásait. Jegyezze fel minden állapothoz az $R(s)$ értékét. Hasznos lehet az (s_A, s_B) állapotok elrendezése egy kétdimenziós rácson, s_A-t és s_B-t „koordinátaként” használva.

d. Most alkalmazza a kétszemélyes értékiterációt a játék megoldására, és származtassa az optimális eljárásmódot.

17.9.

Mutassuk meg, hogy egy domináns stratégiai egyensúly egy Nash-egyensúly, de ez visszafelé nem igaz.

17.10.

A kő-papír-olló gyerekjátékban mindegyik játékos egyszerre felmutatja a választását, ami kő, papír vagy olló. A papír becsomagolja a követ, a kő kicsorbítja az ollót, az olló elvágja a papírt. A kiterjesztett kő-papír-olló-tűz-víz változatban a tűz üti a követ, papírt és ollót, a kő, papír és olló üti a vizet, és a víz üti a tüzet. Írja fel a jutalmazási mátrixot, és keressen egy kevert stratégiájú megoldást a játékhoz.

17.11.

Oldja meg a háromujjas snóblijátékot.

17.12.

A Nemzeti Jégkorong Szövetségben (National Hockey League) 1999 előtt a csapatok 2 pontot kaptak a győzelemért, 1-et a döntetlenért és 0-t, ha veszítettek. Ez zérusösszegű játék? 1999-ben a szabályokat úgy módosították, hogy egy csapat 1 pontot kap, ha hosszabbításban kap ki. A győztes csapat továbbra is 2 pontot kap. Hogyan változtatja meg ez a módosítás a fenti kérdésekre a válaszokat? Ha törvényes lenne, mikor volna racionális a két csapatnak titokban megegyezni, hogy a rendes játékidőben döntetlent érnek el, és majd a

Created by XMLmind XSL-FO Converter.

hosszabbításban döntik el a küzdelmet? Tegyük fel, hogy mindegyik csapat számára a kapott pontszámok jelentik a hasznosságot, és hogy létezik egy kölcsönösen ismert p a priori valószínűség, hogy az első csapat hosszabbításban fog nyerni. Milyen p értékekre értene egyet mindkét csapat ezzel az egyezséggel?

17.13.

A következő jutalommátrix Blindertől származik, amit Bernstein közöl (Blinder, 1983; Bernstein, 1996), a politikusok és a jegybank (Federal Reserve) közötti játékot ábrázolja.

	Bank: szigorít	Bank: semmi	Bank: lazít
Pol: szigorít	$B = 7, P = 1$	$B = 9, P = 4$	$B = 6, P = 6$
Pol: semmi	$B = 8, P = 2$	$B = 5, P = 5$	$B = 4, P = 9$
Pol: lazít	$B = 3, P = 3$	$B = 2, P = 7$	$B = 1, P = 8$

A politikusok lazíthatják vagy szigoríthatják a pénzügyi politikát, míg a jegybank lazíthatja vagy szigoríthatja a monetáris politikát. (És természetesen mindkét oldal választhatja azt, hogy nem tesz semmit.) Mindegyik oldalnak vannak preferenciái, hogy kinek mit kell tennie – egyik oldal sem szeretne rossznak látszani. A mutatott jutalmak egyszerűen sorrendi rangok: 9 az első választásért és végül 1 az utolsóért. Keresse meg a játék Nash-egyensúlyát a tiszta stratégiák között. Ez Pareto-optimális megoldás? Az olvasó esetleg érdemesnek tartja elemezni a jelenlegi kormány politikáját ennek fényében.

Created by XMLmind XSL-FO Converter.

18. fejezet - Megfigyelések alapján történő tanulás

Ebben a fejezetben olyan ágensekkel foglalkozunk, amelyek saját tapasztalataik szorgos tanulmányozásával javítják teljesítményüket.

A tanulás alapgondolata az, hogy a megfigyeléseket ne csak az ágens jelenlegi cselekvéseinek kialakítására használjuk, hanem arra is, hogy javítsuk a cselekvésre való jövőbeli képességeit. A tanulás aközben zajlik, míg az ágens megfigyeli a környezettel való kölcsönhatásait és saját döntéshozó folyamatait. A tanulás széles skálán értelmezhető: egyik végén a tapasztalatok egyszerű memorizálása található, amit a 10. fejezetben megismert wumpus ágens példáz, a skála másik végén pedig a tapasztalatok alapján kialakított komplett tudományos elmélet áll, mint például Albert Einstein elmélete. Ebben a fejezetben a megfigyeléseken alapuló **induktív tanulás**sal (**inductive learning**) foglalkozunk. Speciálisan azzal, hogy hogyan tanulhatók meg a propozíciós logikában megfogalmazott egyszerű elméletek. Annak elemzésére, elméleti magyarázatára is sor kerül, hogy miért működik az induktív tanulás.

1. Tanulási formák

A második fejezetben láttuk, hogy a tanuló ágens felfogható úgy, mint aminek van egy **cselekvő komponens**e (**performance element**), amellyel eldönti, hogy milyen cselekvést válasszon, és egy **tanuló komponens**e (**learning element**), amellyel módosítja ezt a cselekvő elemet annak érdekében, hogy a későbbiekben jobb döntéseket hozzon (lásd 2.15. ábra). A gépi tanulással foglalkozó kutatók a tanuló komponensek széles választékával álltak elő. Ezek megértését segíti, ha megnézzük, hogy a leendő működési területük hogyan befolyásolja tervezésüket. A tanuló komponens tervezését három dolog befolyásolja alapvetően. Ezek:

- A cselekvő elem mely *komponenseit* akarjuk tanítani.
- Milyen *visszacsatolás* áll rendelkezésre ezen komponensek tanítására.
- Hogyan *reprezentáljuk* a komponenseket.

Ezeket a szempontokat rendre elemezni fogjuk. Láttuk, hogy számos lehetőség van a cselekvő elemek építésére, a 2. fejezetben számos ágenskialakítást megismertünk (lásd 2.9., 2.11., 2.13. és 2.14. ábra). Ezeknek az ágenseknek a komponensei a következőket tartalmazzák:

1. Az aktuális állapot feltételeinek közvetlen leképezése cselekvésekre.
2. Olyan lehetőség, amely egy megfigyelési szekvenciából a világ releváns tulajdonságaira képes következtetni.
3. A világ alakulására vonatkozó, valamint az ágens lehetséges cselekvéseinek következményeit leíró információ.
4. A világ lehetséges állapotainak számunkra kívánatos voltát megadó hasznosság- információ.
5. Cselekvés-hasznosság információ, amely az egyes cselekvések kívánatosságát jelzi számunkra.
6. Célok: ezek olyan állapotosztályokat adnak meg, amelyek elérése az ágens hasznosságát maximálja.

Megfelelő visszacsatolás alkalmazása esetén ezen komponensek mindegyike tanulható. Vegyünk például egy ágenst, amely taxivezetést tanul. Minden egyes alkalommal, amikor a felügyelő elkiáltja magát: „Fék!”, az ágens megtanulhat egy feltétel-cselekvés fékezési szabályt (1. komponens). Amennyiben számos olyan képet mutatunk az ágensnek, amelyről közöljük, hogy busz van rajta, akkor megtanulhatja felismerni azokat (2). Ha egy cselekvést próbálgat, és megfigyeli az eredményt – például nedves úton erősen fékez –, megtanulhatja cselekvéseinek hatását (3). Ha nem kap borravalót utasától, akit jól összerázott az út, akkor értékes elemekkel gazdagíthatja az általános hasznosság- függvényét (4).

Az ágens előtt álló tanulási folyamat meghatározásában rendszerint a *visszacsatolás jellege* a legfontosabb faktor. A gépi tanulás területén általában három esetet különböztetnek meg: **ellenőrzött** (**supervised**), **nem ellenőrzött** (**unsupervised**) és **megerősítéses** (**reinforcement**) tanulást.

Created by XMLmind XSL-FO Converter.

Az **ellenőrzött tanulás** (**supervised learning**) egy *leképezésnek* a bemeneti és kimeneti minták alapján történő megtanulását jelenti. Az (1), (2) és (3) esetek mindegyike az ellenőrzött tanulás példája. Az (1) esetben az ágens a fékezés feltétel-cselekvés szabályát tanulja – ez az állapottérről egy Boole-típusú kimenetre való leképezés (fékezzen vagy ne fékezzen). A (2) esetben az ágens képekről egy Boole-típusú kimenetre való leképezést tanul meg (tartalmaz-e a kép buszt, vagy sem). A (3) esetben a tanult fékezési elmélet egy, az állapotok és a fékezési cselekvések teréről a – mondjuk méterben mért – megállási útra történő leképezés. Vegyük észre, hogy az (1) és (2) esetekben a tanító szolgáltatta a mintákhoz tartozó helyes választ; a harmadikban a kimeneti értékeket közvetlenül az ágens észlelései szolgáltatták. Teljesen megfigyelhető környezet esetén mindig fennáll a lehetőség, hogy az ágens megfigyeli a cselekvésének következményeit, így használhat ellenőrzött tanulási módszereket annak érdekében, hogy megjósolja azokat. Részlegesen megfigyelhető környezetben nehezebb a probléma, mert a közvetlen hatások láthatatlanok maradhatnak.

Nem ellenőrzött tanulási (**unsupervised learning**) probléma esetén bemeneti minták tanulása történik, de a kimeneti kívánt minták nem biztosítottak. Például egy taxivezető ágens apránként kialakíthatja a „jó közlekedési napok” és a „rossz közlekedési napok” koncepcióját, anélkül hogy bármikor címkézett példákat kapott volna bármelyikről. Egy tisztán nem ellenőrzött tanulást végző ágens nem képes megtanulni, hogy mit cselekedjék, mivel nincs olyan információja, amely egy cselekvést helyesnek vagy egy állapotot kívánatosnak minősítene. A nem ellenőrzött tanulást elsősorban a valószínűségi következtető rendszerek kapcsán fogjuk tárgyalni (lásd 20. fejezet).

A **megerősítéses tanulás** (**reinforcement learning**) problémája, amelyet a 21. fejezetben tárgyalunk, a legáltalánosabb a három közül. Ahelyett hogy egy tanító útmutatását követhetné, egy megerősítéses tanulást végző ágensnek **megerősítés**i információ alapján kell tanulnia.[1] Például a borravaló hiánya az út végén (vagy egy hatalmas számla, ha beleszáll az előtte haladó kocsiba) némi információt nyújt az ágensnek arról, hogy viselkedése nem volt megfelelő. A megerősítéses tanulás tipikusan magában foglalja azt a részproblémát, hogy az ágensnek meg kell tanulnia azt is, hogyan működik a világ.

A *megtanult információ reprezentációja* szintén nagyon fontos szerepet játszik abban, hogy meghatározhassuk azt, hogyan is kell működnie a tanulási algoritmusnak. Egy ágens bármelyik komponense reprezentálható az ebben a könyvben található sémák bármelyikével. Számos példát láttunk: súlyozott polinomokat, amelyeket a játékot játszó ágensek használtak hasznosságfüggvényként; ítéletkalkulus-beli és elsőrendű logikai állításokat, amelyeket logikai ágensek használtak; valamint valószínűségi leírásokat, mint például a Bayes-hálókat, amelyeket döntéselméleti ágensek használtak következtető komponensként. Ezek mindegyikére alkottak hatékony tanulási eljárásokat. Ebben a fejezetben a propozíciós logikát alkalmazó módszereket tárgyaljuk. A 19. fejezet az elsőrendű logikát alkalmazó módszereket mutatja be, míg a 20. fejezet a Bayes-hálókat és a neurális hálókat tárgyalja (amelyek speciális esetként tartalmazzák a lineáris polinomokat).

A tanuló rendszerek tervezésének utolsó fontos tényezője az *a priori információ rendelkezésre állásának* kérdése. Az MI, a számítástudomány és a pszichológia területén a tanulással foglalkozó kutatás nagy része azzal az esettel foglalkozott, amikor kezdetben az ágensnek semmi információja nincs arról, amit megpróbál megtanulni. Bár ez egy fontos speciális eset, de egyáltalán nem ez az általános. A legtöbb emberi tanulás egy jó adag kiinduló háttértudásra épül. Néhány pszichológus és nyelvész szerint még az újszülötteknek is van ismeretük a világról. Bármi is legyen az igazság ezt illetően, az kétségtelen, hogy a kiinduló tudás rengeteget segíthet a tanulásban. Buborék-kamra fényképek alapján egy fizikus egy új – meghatározott tömegű és töltésű – részecske létezésére mutató elméletre következtethet. Ugyanakkor egy kritikus ugyanazon képhalmaz vizsgálata alapján csupán arra következtethet, hogy a „művész” bizonyára valamilyen absztrakt vagy expresszionista irányzat követője. A 19. fejezet számos módszert mutat be arra, hogy a tanulást hogyan segíti a meglévő tudás; azt is megmutatjuk ebben a fejezetben, hogy a tudást hogyan lehet úgy *összeállítani,* hogy felgyorsítsuk a döntéshozatalt. A 20. fejezet azt mutatja be, hogy a kiinduló tudás hogyan segít a valószínűségi elméletek tanulásában.

2. Induktív tanulás

Egy determinisztikus, felügyelt tanulást végző algoritmus bemenetként megkapja az ismeretlen függvény bizonyos bemeneti értékekhez tartozó válaszait, feladata az ismeretlen függvény – vagy ahhoz nagyon közel álló leképezés – előállítása. Formálisan azt mondjuk, hogy egy **minta** (**example**) nem más, mint egy (x, $f(x)$) értékpár, ahol x a bemeneti érték, míg $f(x)$ az x-re alkalmazott függvény kimeneti értéke. A **tisztán induktív következtetés** (**pure inductive inference**) (vagy **indukció** [**induction**]) feladata a következő:

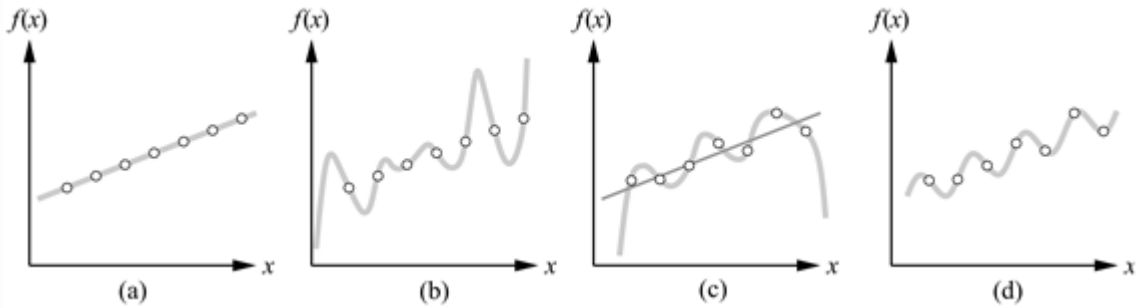
[1] A 17. fejezetben használt **jutalom** (**reward**) kifejezés a **megerősítés** (**reinforcement**) szinonimája.

Created by XMLmind XSL-FO Converter.

Adott az *f*-re vonatkozó minták egy halmaza, ennek alapján határozzunk meg egy *h* függvényt, amely közelíti *f*-et.

A *h* függvényt **hipotézis**nek (**hypothesis**) nevezzük. A tanulás nehézsége elvi szempontból abban áll, hogy nem könnyű egy adott *h* függvényről megmondani, hogy jól approximálja-e *f*-et. Egy jó hipotézis jól **általánosít** (**generalize**) – azaz jó becslést kapunk a még nem látott mintákra is. Ez az **induktív következtetés alapproblémája** (**problem of induction**). A problémát már évszázadok óta kutatják, a 18.5. alfejezet egy részmegoldást szolgáltat.

18.1. ábra - (a) Példa (*x*, *f*(*x*)) értékpárok és egy velük konzisztens lineáris hipotézis. (b) Egy – ugyanazokkal az adatokkal konzisztens – hetedfokú polinom. (c) Egy másik adathalmaz egy pontosan illeszkedő hatodfokú polinommal, illetve egy közelítő lineáris illesztéssel. (d) Egy egyszerű szinuszos illesztés ugyanazokra az adatokra.



Fontos

A 18.1. ábra ismert példát mutat be: egy adathalmazra egyváltozós függvényt illesztünk. A minták (*x*, *f*(*x*)) értékpárok, ahol mind *x*, mind *f*(*x*) valós értékek. Legyen a **H hipotézistér** (**hypothesis space H**) – azon hipotézisek halmaza, amelyeket meg fogunk vizsgálni – a legfeljebb *k*-ad fokú polinomok tere. (Például $3x^2 + 2$, $x^{17} - 4x^3$ stb.) A 18.1. (a) ábra valamilyen adatokat és egy az adatokra pontosan illeszkedő egyenes vonalat (elsőfokú polinom) mutat. Ez a vonal itt egy **konzisztens** (**consistent**) hipotézis, mivel valamennyi adatra illeszkedik. A 18.1. (b) ábra egy nagy fokszámú polinomot mutat, amely szintén konzisztens ugyanazokra az adatokra nézve. Ez a két példa illusztrálja az induktív tanulás első komoly megoldandó kérdését: *hogyan válasszunk több konzisztens hipotézis közül?* Egy lehetséges válasz az ún. **Ockham borotvája**[2] (**Ockham's razor**) elv: részesítsük előnyben a *legegyszerűbb* olyan hipotézist, amely konzisztens az adatokkal. Ez józan intuíciónak tűnik, mivel azok a hipotézisek, amelyek nem egyszerűbbek, mint maguk az adatok, nem nyernek ki semmilyen *mintázatot* az adatokból. Az egyszerűség definíciója nem könnyű, de értelmesnek tűnik azt mondani, hogy egy elsőfokú polinom egyszerűbb, mint egy 12-ed fokú.

Fontos

A 18.1. (c) ábra egy másik adathalmazt mutat. Ezekre az adatokra nem lehet konzisztens módon egyenest illeszteni, valójában egy hatodfokú polinomra (amely 7 szabad paraméterrel rendelkezik) van szükség a pontos illesztéshez. A halmazban csak 7 adat van, így a polinomnak épp annyi paramétere van, ahány adatpont. Ennek megfelelően úgy tűnik, hogy nem sikerül valamilyen mintázatot találnunk az adathalmazban, így nem várunk jó általánosítóképességet. Jobban járhatunk, ha inkább egy egyszerű egyenest illesztünk, amely ugyan nem konzisztens, de ésszerű becsléseket adhat. Ez azt valószínűsíti, hogy a keresett függvény nem determinisztikus (vagy ami ezzel közelítőleg ekvivalens: a valódi bemeneti minták nem figyelhetők meg teljes pontossággal). *Nemdeterminisztikus függvények esetén elkerülhetetlen, hogy kompromisszumot kössünk a hipotézis komplexitása és az adatokhoz való illeszkedés pontossága között.* A 20. fejezetben bemutatjuk, hogy ezt a kompromisszumot hogyan lehet a valószínűség-számítás elméletének felhasználásával kialakítani.

[2] A 14. századi angol filozófus, William of Ockham után. Ockham nevét gyakran hibásan Occamnak írják, ami talán a francia „Guillaume d'Occam" fordításából ered.

Created by XMLmind XSL-FO Converter.

Észben kell tartanunk, hogy egy egyszerű, konzisztens hipotézis megtalálásának esélye erősen függ a választott hipotézistértől. A 18.1. (d) ábra bemutatja, hogy a (c) alatti adatokra pontosan illeszthető egy egyszerű $ax + b + c \sin(x)$ alakú függvény. Ez a példa jól mutatja a hipotézistér jó megválasztásának fontosságát. Ha olyan hipotézisteret választunk, amely véges fokszámú polinomokat tartalmaz, akkor szinuszos függvényeket nem tudunk pontosan reprezentálni. Így egy olyan tanuló, amelyik ezt a hipotézisteret használja, nem fog tudni szinuszos adatokból tanulni. Egy tanulási problémát **realizálható**nak (**realizable**) nevezünk, ha a hipotézistérnek eleme az igazi függvény, különben a problémát **nem realizálható**nak (**unrealizable**) nevezzük. Sajnos nem mindig tudjuk megmondani, hogy egy tanulási probléma realizálható-e, mivel nem ismerjük a valódi függvényt. Egy lehetőség e korlát túllépésére az a priori tudás felhasználása. Ennek segítségével olyan hipotézisteret alkotunk, amelyről tudjuk, hogy tartalmazza az igazi függvényt. Ezt a témát a 19. fejezetben tárgyaljuk.

Fontos

Egy másik lehetséges megközelítés, hogy a lehető legnagyobb hipotézisteret használjuk. Például miért ne legyen a **H** hipotézisterünk az összes Turing-gépek osztálya? Végül is minden kiszámítható függvény reprezentálható valamilyen Turing-géppel, tehát ez a legjobb, amit tehetünk. Ennek a megközelítésnek az a problémája, hogy nem veszi figyelembe a tanulás *számítási komplexitását*. *Kompromisszumot kell kötnünk a hipotézistér kifejezőképessége és egyszerű konzisztens hipotézisek megtalálásának komplexitása között.* Például: egyenesek illesztése az adatokra könnyen megoldható, nagy fokszámú polinomok illesztése nehezebb, Turing-gépek illesztése pedig nagyon nehéz. Ugyanis annak eldöntése, hogy egy adott Turing-gép konzisztens-e az adatokkal, általánosságban még csak nem is lehetséges. Egy másik ok, ami miatt előnyben részesítjük az egyszerű hipotézistereket, hogy az eredményképp kapott hipotézisek használata is egyszerűbb lehet – könnyebb kiszámítani $h(x)$-et, ha h egy lineáris függvény, mint ha egy tetszőleges Turing-gép programja lenne.

Ezen okokból a legtöbb tanulással foglalkozó eddigi kutatás a viszonylag egyszerű hipotézisreprezentációkra helyezte a fő hangsúlyt. Ebben a fejezetben az ítéletlogikára és az ehhez kapcsolódó nyelvekre koncentrálunk. Az elsőrendű logikai reprezentációt használó tanulás elméletével a 19. fejezet foglalkozik. Látni fogjuk, hogy a kifejezőképesség-komplexitás kompromisszum nem is olyan egyszerű, mint először gondolnánk. Gyakran az a helyzet – mint azt a 8. fejezetben is láttuk –, hogy egy kifejezőnyelv lehetővé teszi az adatokra illesztett *egyszerű* elméletet, míg a nyelv kifejezőképességének korlátozása azt is jelenti, hogy bármely konzisztens elmélet szükségszerűen bonyolult lesz. A sakk szabályait például egy-két oldalon le tudjuk írni elsőrendű logika segítségével, míg az ítéletlogikával történő leírás több ezer oldalt igényelne. Ilyen esetekben kell arra lehetőségnek lennie, hogy a kifejezőbb nyelven sokkal gyorsabb legyen a tanulás.

3. Döntési fák megalkotása tanulással

A döntési fa tanulás egyike a legegyszerűbb, mégis az eddigiekben az egyik legsikeresebbnek bizonyult tanulási algoritmusnak. Jó bevezetésként szolgál az induktív tanulás területén, és ráadásul könnyen implementálható. Először bemutatjuk a cselekvő alrendszert, majd megmutatjuk, hogy miképpen lehet tanítani. Mindeközben olyan elveket mutatunk be, amelyek az induktív tanulás minden területére jellemzők.

3.1. Cselekvő komponensként használt döntési fák

Egy **döntési fa** (**decision tree**) bemenetként egy **attribútum**okkal (**attribute**s) leírt objektumot vagy szituációt kap, és egy „döntést” ad vissza eredményként – a bemenetre adott válasz jósolt értékét. A bemeneti attribútumok lehetnek diszkrétek vagy folytonosak. Jelen tárgyalásban diszkrét bemeneteket tételezünk fel. A kimeneti érték szintén lehet diszkrét vagy folytonos; egy diszkrét értékkészletű függvény tanulását **osztályozás** (**classification**) tanulásnak, míg a folytonos függvény tanulását **regresszió**nak (**regression**) nevezzük. *Bináris (Boolean)* osztályozásra fogunk koncentrálni, ahol minden példát vagy **igaznak** (**pozitív**), vagy **hamisnak** (**negatív**) sorolunk be.

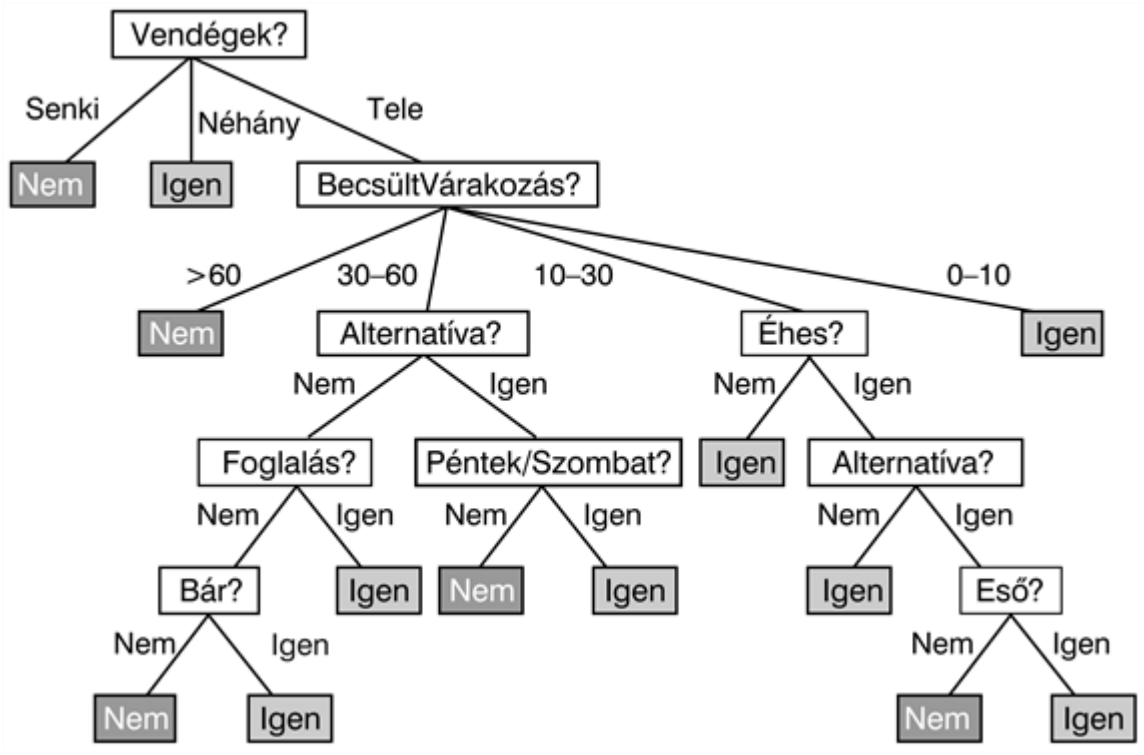
A döntési fa egy tesztsorozat elvégzése során jut el a döntéshez. A fa minden egyes belső csomópontja valamely tulajdonság értékére vonatkozó tesztnek felel meg, a csomópontból kilépő ágakat pedig a teszt lehetséges kimeneteivel címkézzük. Minden egyes levélcsomópont megadja azt az értéket, amelyet vissza kell adnunk, ha ezt a levelet elértük. Úgy tűnik, hogy a döntési fa reprezentáció az emberek számára rendkívül természetes; valójában számos „Hogyan csináljuk?” kézikönyvet (például az autójavítási kézikönyveket) írtak meg egyetlen hatalmas, több száz oldalra kiterjedő döntési faként.

Created by XMLmind XSL-FO Converter.

Valamelyest egyszerűbb példa lehet az a probléma, hogy várjunk-e egy étteremben egy asztal felszabadulására. Az a célunk, hogy tanulással kialakítsuk a *VárjunkE* **célpredikátum** (**goal predicate**) definícióját. Ahhoz, hogy ezt tanulási feladatként kezelhessük, először meg kell határozzuk, hogy milyen attribútumok állnak rendelkezésre ahhoz, hogy ezen a problématerületen leírjuk a példákat. A 19. fejezetben megmutatjuk, hogyan lehet automatizálni ezt a feladatot; most egyszerűen tegyük fel, hogy a következő attribútumlista mellett döntöttünk:

1. *Alternatíva*: van-e a közelben megfelelő alternatívaként kínálkozó étterem.
2. *Bár*: van-e az étteremnek kényelmes bár része, ahol várakozhatunk.
3. *Péntek/Szombat*: igaz értéket vesz fel pénteken és szombaton.
4. *Éhes*: éhesek vagyunk-e.
5. *Vendégek*: hány ember van az étteremben (értékkészlet *Senki*, *Néhány* és *Tele*).
6. *Drága*: az étterem menyire drága ($, $$, $$$).
7. *Eső*: esik-e odakint az eső.
8. *Foglalás*: foglaltunk-e asztalt.
9. *Konyha*: az étterem típusa (francia, olasz, thai vagy burger).
10. *BecsültVárakozás*: a pincér becsülte várakozási idő (0–10, 10–30, 30–60, >60 perc)

18.2. ábra - Egy döntési fa annak eldöntésére, hogy várjunk-e asztalra



A szerzők egyike (SR) által erre a problémára rendszerint használt döntési fa a 18.2. ábrán látható. Vegyük észre, hogy a fa nem használja a *Drága* és a *Konyha* attribútumokat, valójában irrelevánsnak tekinti azokat. A példákat a döntési fa a gyökérnél kezdi feldolgozni, követi a megfelelő ágakat, amíg egy levélhez el nem ér. Például egy *Vendégek* = *Tele* és *BecsültVárakozás* = 0–10 attribútumokkal jellemezhető példa pozitív kimenetet fog eredményezni (azaz várni fogunk egy asztalra).

3.2. A döntési fák kifejezőképessége

Created by XMLmind XSL-FO Converter.

Logikai felírást használva minden egyes döntési fa, amely a *VárjunkE* célpredikátum egy hipotézise, a következő formában felírt állításnak felel meg:

$$\forall s \; VárjunkE(s) \Leftrightarrow (P_1(s) \vee P_2(s) \vee \ldots \vee P_n(s))$$

ahol mindegyik $P_i(s)$ feltétel azon tesztek konjunkciójának felel meg, amelyeket a gyökértől egy pozitív kimenetet jelentő levélig megtett út során végeztünk. Bár ez egy elsőrendű logikában felírt mondatnak tűnik, valójában bizonyos értelemben ítéletlogikai kifejezés, mivel csak egyetlen változót tartalmaz és az összes predikátum unáris. Valójában a döntési fa a *VárjunkE* és az attribútumértékek bizonyos logikai kapcsolatát írja le. A döntési fa nem használható olyan tesztek reprezentálására, amelyek kettő vagy több különböző objektumra vonatkoznak – például:

$$\exists r_2 \; Közel(r_2, r) \wedge Drága(r, p) \wedge Drága(r_2, p_2) \wedge Olcsóbb(p_2, p)$$

(van a közelben egy olcsóbb étterem?). Nyilvánvalóan felvehetnénk egy *OlcsóbbÉtteremKözel* nevű logikai attribútumot, de az *összes* ilyen attribútum hozzáadása kezelhetetlenné teszi a problémát. A 19. fejezet mélyebbre hatol az elsőrendű logikában történő tanulás területén.

Az ítéletlogikai nyelvek területén a döntési fák teljes kifejezőképességgel bírnak, ami azt jelenti, hogy tetszőleges logikai (Boole) függvény felírható döntési faként. Ezt triviálisan megvalósíthatjuk, ha a függvény igazságtáblájának minden sorát megfeleltetjük a döntési fa egy útjának. Ez exponenciálisan növekvő döntési fára vezet, mivel az igazságtábla exponenciálisan növekvő számú sort tartalmaz. Nyilvánvalóan a döntési fák sok függvényt jóval kisebb fával képesek reprezentálni.

Ugyanakkor némely függvényfajtánál ez valóban problémát jelent. Például ha a függvényünk a **paritásfüggvény** (**parity function**), amely akkor és csak akkor ad 1-et, ha páros számú bemenet 1 értékű, akkor egy exponenciálisan nagy döntési fára lesz szükség. Hasonlóan nehéz a **többségfüggvény** (**majority function**) reprezentálása döntési fával, amely függvény akkor ad 1-et, ha bemeneteinek több mint fele 1 értékű.

Más szavakkal, a döntési fák bizonyos függvények esetén jók, mások esetén roszszak. Van *bármilyen* olyan reprezentáció, amely *minden*fajta függvény esetén hatékony? A válasz sajnos az, hogy nincs. Ezt általánosan is meg tudjuk mutatni. Vizsgáljuk az összes *n* bemeneti attribútummal rendelkező logikai (Boole) függvényt. Hány különböző függvény van ebben a halmazban? Ez éppen a lehetséges felírható igazságtáblák számával egyezik meg, hiszen egy függvényt az igazságtáblája ad meg. Az igazságtáblának 2^n sora van, mivel minden bemeneti esetet *n* attribútummal adunk meg. Úgy tekinthetjük a tábla „válasz” oszlopát, mint egy 2^n bites számot, amely definiálja a függvényt. Mindegy, hogy a függvények milyen reprezentációját választjuk, néhánynak (valójában szinte mindegyiknek) a reprezentálásához tényleg szükség lesz ennyi bitre.

Ha 2^n bitre van szükség a függvény megadásához, akkor *n* attribútum esetén 2^{2^n} a lehetséges függvények száma. Ez megrázóan nagy szám. Például csupán hat Boole-változó esetén is $2^{2^6} = 18\,446\,744\,073\,709\,551\,616$ különböző logikai (Boole) függvény állítható elő. Szükségünk lesz néhány szellemes algoritmusra, hogy egy ilyen hatalmas térben konzisztens hipotézist tudjunk találni.

3.3. A döntési fák példák alapján történő felépítése

Egy logikai döntési fa által kezelhető példa a bemeneti attribútumok *X* vektorából és egyetlen logikai kimeneti értékből, *y*-ból áll. A 18.3. ábra egy (X_1, y_1)...(X_{12}, y_{12}) példahalmazt mutat. Azokat nevezzük pozitív példáknak (X_1, X_3, ...), amelyekben a *VárjunkE* értéke igaz, és azok a negatív példák, amelyekben az értéke hamis (X_2, X_5, …). A példák teljes halmazát **tanító halmaz**nak (**training set**) nevezzük.

Az a probléma, hogy a tanító halmaznak megfelelő döntési fát találjunk, bonyolultnak tűnik ugyan, de valójában van egy triviális megoldása. Egyszerűen egy olyan fát konstruálhatunk, amelyben minden példához egy külön saját utat hozunk létre egy – a példához tartozó – levélcsomóponthoz. A levélhez vezető út mentén sorra teszteljük az attribútumokat, és a példához tartozó tesztértéket követjük, a levél pedig a példa besorolását adja. Ha ismét ugyanazt a példát[3] vesszük, akkor a döntési fa a helyes osztályozási eredményt adja. Szerencsétlen módon nemigen ad információt egyetlen más esetről sem!

[3] Nagyon fontos, hogy megkülönböztessük az azonos példát egy *azonos leírással rendelkező példától*. Erre a fontos különbségre még visszatérünk a 19. fejezetben.

Created by XMLmind XSL-FO Converter.

18.3. ábra - Példák az étterem problématerületre

Példa	Attribútumok										Cél
	Alter-nativa	*Bár*	*Péntek/ Szombat*	*Éhes*	*Vendégek*	*Drága*	*Eső*	*Foglalás*	*Konyha*	*Becsült Várakozás*	*Vár-junkE*
X_1	*Igen*	*Nem*	*Nem*	*Igen*	*Néhány*	*$$$*	*Nem*	*Igen*	*Francia*	*0–10*	*Igen*
X_2	*Igen*	*Nem*	*Nem*	*Igen*	*Tele*	*$*	*Nem*	*Nem*	*Thai*	*30–60*	*Nem*
X_3	*Nem*	*Igen*	*Nem*	*Nem*	*Néhány*	*$*	*Nem*	*Nem*	*Burger*	*0–10*	*Igen*
X_4	*Igen*	*Nem*	*Igen*	*Igen*	*Tele*	*$*	*Igen*	*Nem*	*Thai*	*10–30*	*Igen*
X_5	*Igen*	*Nem*	*Igen*	*Nem*	*Tele*	*$$$*	*Nem*	*Igen*	*Francia*	*>60*	*Nem*
X_6	*Nem*	*Igen*	*Nem*	*Igen*	*Néhány*	*$$*	*Igen*	*Igen*	*Olasz*	*0–10*	*Igen*
X_7	*Nem*	*Igen*	*Nem*	*Nem*	*Senki*	*$*	*Igen*	*Nem*	*Burger*	*0–10*	*Nem*
X_8	*Nem*	*Nem*	*Nem*	*Igen*	*Néhány*	*$$*	*Igen*	*Igen*	*Thai*	*0–10*	*Igen*
X_9	*Nem*	*Igen*	*Igen*	*Nem*	*Tele*	*$*	*Igen*	*Nem*	*Burger*	*>60*	*Nem*
X_{10}	*Igen*	*Igen*	*Igen*	*Igen*	*Tele*	*$$$*	*Nem*	*Igen*	*Olasz*	*10–30*	*Nem*
X_{11}	*Nem*	*Nem*	*Nem*	*Nem*	*Senki*	*$*	*Nem*	*Nem*	*Thai*	*0–10*	*Nem*
X_{12}	*Igen*	*Igen*	*Igen*	*Igen*	*Tele*	*$*	*Nem*	*Nem*	*Burger*	*30–60*	*Igen*

Ezzel a triviális megoldással az a baj, hogy egyszerűen memorizálja a példákat. Mivel nem nyer ki semmilyen mintázatot a példákból, ezért nem várhatjuk, hogy extrapolálni tudjon azokra a mintákra, amelyeket még sohasem látott. Az Ockham borotvája elvét alkalmazva nekünk azt a *legkisebb* döntési fát kell megtalálnunk, amely konzisztens a példákkal. Sajnálatos módon a „legkisebb” összes értelmes definíciója esetén a legkisebb döntési fa megtalálása kezelhetetlen problémát jelent. Viszont némi egyszerű heurisztika bevetésével jó eredményt érhetünk el egy „kicsike” fa megtalálásában. A `DÖNTÉSI-FA-TANULÁS` algoritmus alapötlete az, hogy teszteljük először a legfontosabb attribútumot. A „legfontosabb” alatt azt az attribútumot értjük, amelyik a legnagyobb változást okozza a példák besorolásában. Ezen a módon – reményünk szerint – kisszámú teszttel helyes osztályozáshoz jutunk, ami azt jelenti, hogy a fában minden út rövid lesz, tehát az egész fa kicsi lesz.

A 18.4. ábrán látható, hogy hogyan indul az algoritmus. 12 példánk van, amelyeket pozitív és negatív példahalmazokba sorolunk. Ezek után eldöntjük, hogy melyik attribútumot teszteljük először a fában. A 18.4. (a) ábrán megmutattuk, hogy a *Konyha* attribútum rossz választás lenne, mert tesztjének 4 kimenetele van, és mindegyik esetén ugyannyi pozitív és negatív példánk lesz az eredményül kapott halmazokban. Másrészt viszont a 18.4. (b) ábrán látható, hogy a *Vendégek* egy meglehetősen fontos attribútum, hiszen ha értéke *Senki* vagy *Néhány*, akkor határozott választ tudunk adni (az első esetben *Nem*, a másodikban *Igen* a válaszunk). Ha az attribútum értéke *Tele*, akkor a példák vegyes halmazát kapjuk. Általánosságban megállapítható, hogy miután az első attribútum tesztje csoportokra bontotta a példákat, mindegyik teszteredmény egy újabb döntési fa tanulási problémát eredményez, kevesebb példával és egygyel kevesebb attribútummal. Négy esetet kell áttekintenünk ezekben a rekurzív problémákban:

1. Ha van néhány pozitív és néhány negatív példánk, akkor válasszuk a legjobb attribútumot a szétosztásukra. A 18.4. (b) ábrán bemutattuk, hogy az *Éhes* attribútum alkalmas a megmaradó példák osztályozására.

2. Ha valamennyi megmaradt példánk pozitív (vagy mind negatív), akkor készen vagyunk: válaszolhatunk *Igen*-t vagy *Nem*-et. A 18.4. (b) ábra bemutatja ezt a *Senki*, illetve a *Néhány* esetekben.

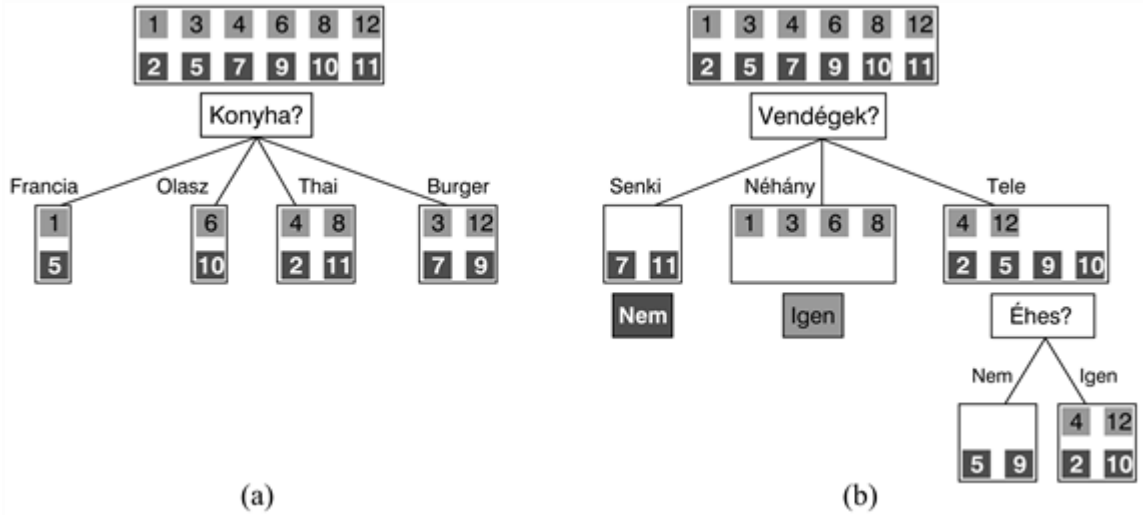
3. Ha nem marad példa a teszt egyik kimenetele esetén, akkor ez azt jelenti, hogy nem figyeltünk meg ilyen esetet, és a szülőcsomópontban többségben levő választ adjuk.

4. Ha nem maradt attribútumunk, amelyet tesztelhetnénk, de mind pozitív, mind negatív példáink maradtak, akkor bajban vagyunk. Ez azt jelenti, hogy ezeknek a példáknak pontosan azonos jellemzőik vannak, de különböző osztályokba tartoznak. Ez egyrészt akkor fordulhat elő, ha néhány adat nem megfelelő, azt mondjuk, hogy **zajos**ak (**noise**) az adatok. Másrészt akkor is előállhat ez a helyzet, ha az attribútumok nem adnak elég információt a szituáció teljes leírására, vagy a problématér valójában nemdeterminisztikus. Egyszerű megoldása lehet ennek a problémának a többségi szavazás használata.

18.4. ábra - A példahalmaz attribútumteszteléssel történő szétosztása. (a) A *Konyha* alapján történő szétosztás nem visz közelebb ahhoz, hogy a pozitív és negatív példákat

Created by XMLmind XSL-FO Converter.

megkülönböztessük. (b) A *Vendégek* attribútum tesztje jól szeparálja a pozitív és negatív példákat. Miután a *Vendégek* tesztjét elvégeztük, az *Éhes* egy eléggé jó második tesztlehetőséget ad.



A DÖNTÉSI-FA-TANULÁS algoritmust a 18.5. ábrán mutatjuk be. Az ATTRIBÚTUM-VÁLASZTÁS módszer bemutatása a következő alfejezetben található.

Az algoritmus által a 12 mintából álló adathalmaz alapján létrehozott fa a 18.6. ábrán látható. Az eredményként kapott fa nyilvánvalóan különbözik a 18.2. ábrán látható eredetitől, annak ellenére, hogy az adatokat egy, az eredeti fát használó ágens generálta. Arra gondolhatnánk, hogy az algoritmus nem ért el különösebben jó eredményt a helyes függvény megtanulásában. Mindamellett ez téves következtetés lenne. A tanuló algoritmus a *példákat* látja, nem az eredeti függvényt. Valójában a létrehozott hipotézis (lásd 18.6. ábra) nem csupán megfelel az összes példának, de lényegesen egyszerűbb is, mint az eredeti fa. A tanuló algoritmusnak semmi oka sincs az *Eső* és *Foglalás* attribútumokra vonatkozó teszteket felvenni, mivel ezek nélkül is tudja osztályozni az összes példát. Mellesleg, felfedett egy érdekes és váratlan viselkedést: az első szerző hétvégeken várni fog thai ételekre.

18.5. ábra - A döntési fa tanulásra szolgáló algoritmus

function DÖNTÉSI-FA-TANULÁS(*példák*, *attribútumok*, *alapérték*) **returns** egy döntési fa
  inputs: *példák*, a példák halmaza
    attribútumok, az attribútumok halmaza
    alapérték, a célpredikátum alapértéke

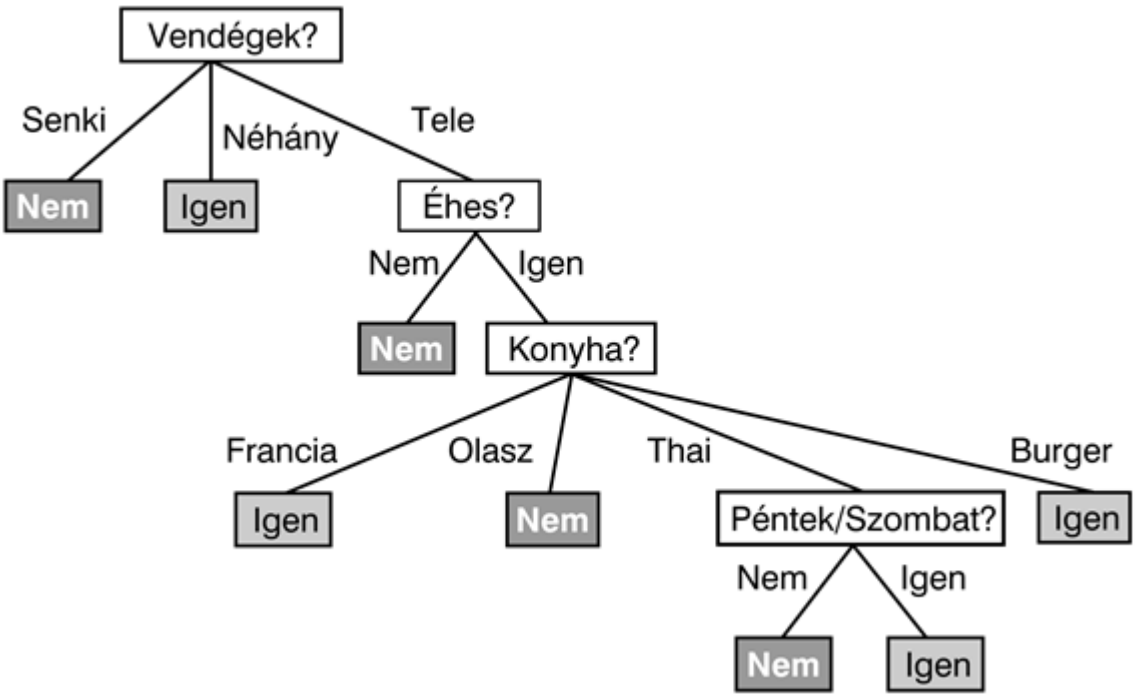
  if *példák* üres **then return** *alapérték*
  else if *példák* minden elemének azonos a besorolása **then return** a besorolás
  else if *attribútumok* üres halmaz **then return** TÖBBSÉGI-ÉRTÉK(*példák*)
  else
    legjobb ← ATTRIBÚTUM-VÁLASZTÁS(*attribútumok*, *példák*)
    fa ← egy új döntési fa, a gyökér a *legjobb* attribútum tesztje
    m ← TÖBBSÉGI-ÉRTÉK($példák_i$)
    for each *legjobb* minden v_i értékére **do**
      $példák_i$ ← {a *példák* azon elemei, amelyekre $legjobb = v_i$}
      részfa ← DÖNTÉSI-FA-TANULÁS($példák_i$, *attribútumok-legjobb*, *m*)
      a *fa* döntési fához adjunk egy v_i címkéjű ágat és a *részfa* részfát
    return *fa*

Természetesen ha több mintát gyűjtöttünk volna, akkor egy, az eredetihez hasonlóbb fát hozhattunk volna létre az induktív tanulással. A 18.6. ábrán látható fa elkerülhetetlenül hibákat is el fog követni; például sohasem látott

Created by XMLmind XSL-FO Converter.

olyan helyzetet, amelyben a várakozási idő 0–10 perc, de az étterem tele van. Egy olyan esetben, amikor az *Éhes* attribútum hamis értékű, a fa azt választja, hogy ne várjunk, pedig én (S. Russell) bizonyára várnék. Felmerül tehát az a kézenfekvő kérdés, hogy ha az algoritmus egy konzisztens, de inkorrekt fát indukál, akkor mennyire lesz inkorrekt ez a fa? Ezt azután fogjuk kísérletileg elemezni, hogy először elmagyaráztuk az attribútumteszt-választási lépés részleteit.

18.6. ábra - A 12 példás tanító minta halmazból származtatott döntési fa



3.4. Attribútumteszt-választás

A döntési fa tanulás során az attribútumok kiválasztására szolgáló eljárás arra irányul, hogy minimalizáljuk az eredményül kapott fa mélységét. Az alapötlet az, hogy azt az attribútumot válasszuk, amellyel a lehető legmesszebbre jutunk a példák pontos osztályozásában. Egy tökéletes attribútum a példákat egy csupa pozitív és egy csupa negatív példát tartalmazó halmazra osztja. A *Vendégek* nem tökéletes attribútum, de meglehetősen jó. Egy valójában haszontalan attribútum, mint például a *Konyha*, tesztjének eredményeként a kapott halmazokban nagyjából ugyanolyan arányban lesz pozitív és negatív példa, mint az eredeti halmazban.

Mindössze arra van szükségünk, hogy formális mértéket találjunk arra, hogy mit jelent a „meglehetősen jó” és a „valójában haszontalan”, ezek után implementálni tudjuk a 18.5. ábrán látható `ATTRIBÚTUM-VÁLASZTÁS` függvényt. A mérték akkor érje el maximumát, amikor az attribútum tökéletes, és akkor legyen minimális, amikor az attribútumnak egyáltalán nincs semmi haszna. Egy megfelelő mérték az attribútum által szolgáltatott **információ** (**information**) várható értéke, ahol az információt abban a matematikai értelmezésben használjuk, ahogy először Shannon és Weaver definiálta (Shannon és Weaver, 1949). Az információ definíciójának megértéséhez gondoljunk például annak a kérdésnek a megválaszolására, hogy egy feldobott pénzérme fej oldala lesz-e felül. Az, hogy a válaszban mennyi információ rejlik, az az előzetes a priori tudástól függ. Minél kevesebbet tudunk, annál több a szolgáltatott információ. Az információelmélet **bit**ekben méri az információtartalmat. Egy bit információ ahhoz elég, hogy egy olyan kérdésre, amelyről semmilyen előzetes elképzelésünk sem volt (például egy szabályos érme feldobásának eredménye) igen/nem választ megadjunk. Általánosságban, ha a lehetséges v_i válaszok valószínűsége $P(v_i)$, akkor a válasz I információtartalmát a következő összefüggés adja meg:

$$I\left(P(v_1),\ldots,P(v_n)\right) = \sum_{i=1}^{n} -P(v_i)\log_2 P(v_i)$$

A pénzfeldobás esetre ellenőrizve az összefüggést, a következőt kapjuk:

Created by XMLmind XSL-FO Converter.

$$I\left(\frac{1}{2},\frac{1}{2}\right)=-\frac{1}{2}\log_2\frac{1}{2}-\frac{1}{2}\log_2\frac{1}{2}=1\text{ bit}$$

Ha az érme hamisított oly módon, hogy a dobás 99%-ban fejre jön ki, akkor $I(1/100, 99/100) = 0{,}08$ bitre van szükség, és ha a fej kimenetel valószínűsége 1-hez tart, akkor az aktuális kimenetel megjóslásához szükséges információ 0-hoz tart.

Döntési fa tanulás esetén a megválaszolásra váró kérdés az, hogy egy adott példának mi a helyes besorolása. Egy jó döntési fa választ ad erre a kérdésre. Még mielőtt egyetlen attribútumot teszteltünk volna, a válaszok valószínűségét becsülhetjük a tanító halmazban található pozitív és negatív minták arányával. Tegyük fel, hogy a tanító halmazban p pozitív és n negatív példa található. Ez esetben a helyes válasz információtartalmának becslése:

$$I\left(\frac{p}{p+n},\frac{n}{p+n}\right)=-\frac{p}{p+n}\log_2\frac{p}{p+n}-\frac{n}{p+n}\log_2\frac{n}{p+n}$$

A 18.3. ábrán bemutatott étterem tanító halmaz $p = n = 6$ mintát tartalmaz, tehát 1 bit információra van szükségünk.

Egyetlen A attribútum tesztje nem fogja megadni mindezt az információt, de valamennyit megad belőle. Pontosan mérni tudjuk, hogy mennyit, ha megnézzük, hogy mennyi információra van még szükségünk az attribútumteszt *után*. Bármely A attribútum, amely v különböző értéket vehet fel, E_1, ..., E_v részhalmazokra bontja az E tanító halmazt, az A lehetséges értékei szerint. Mindegyik E_i részhalmaz p_i pozitív és n_i negatív példát tartalmaz, így ezen az ágon továbbhaladva $I(p_i/(p_i + n_i), n_i/(p_i + n_i))$ bit információra van szükségünk a válasz megadásához. A tanító halmazból véletlen mintavétellel nyert minta esetén annak valószínűsége, hogy az A attribútum ezen a mintán az i-edik értéket veszi fel: $(p_i + n_i)/(p + n)$. Ennek megfelelően az A attribútum tesztje után az átlagos információszükséglet, amely a példa osztályozásához kell:

$$MaradékInfoSzükséglet(A)=\sum_{i=1}^{v}\frac{p_i+n_i}{p+n}I\left(\frac{p_i}{p_i+n_i},\frac{n_i}{p_i+n_i}\right)$$

Az attribútum tesztjéből származó **információnyereség** (**information gain**) az eredeti (A tesztje előtti) információszükséglet és a teszt utáni új információszükséglet különbségeként kapható meg:

$$InfoNyereség(A)=I\left(\frac{p}{p+n},\frac{n}{p+n}\right)-MaradékInfoSzükséglet(A)$$

Az `ATTRIBÚTUM-VÁLASZTÁS` függvényben használt heurisztika csupán azt takarja, hogy válasszuk a legnagyobb nyereséget biztosító attribútumot. Visszatérve a 18.4. ábrán vizsgált attribútumokhoz:

$$InfoNyereség(Vendégek)=1-\left[\frac{2}{12}I(0,1)+\frac{4}{12}I(1,0)+\frac{6}{12}I\left(\frac{2}{6},\frac{4}{6}\right)\right]\approx 0{,}541\text{ bit}$$

$$InfoNyereség(Konyha)=1-\left[\frac{2}{12}I\left(\frac{1}{2},\frac{1}{2}\right)+\frac{2}{12}I\left(\frac{1}{2},\frac{1}{2}\right)+\frac{4}{12}I\left(\frac{2}{4},\frac{2}{4}\right)+\frac{4}{12}I\left(\frac{2}{4},\frac{2}{4}\right)\right]=0$$

Ez megerősíti azt az intuíciónkat, hogy a *Vendégek* jobb attribútum a tanítóhalmaz részhalmazokra szabdalására, mint a *Konyha*. Az a helyzet, hogy a *Vendégek* attribútumnak van a legnagyobb információnyeresége, ezért a döntési fa tanuló algoritmus ennek tesztjét választaná a fa gyökércsomópontjába.

3.5. A tanuló algoritmus teljesítményének becslése

Egy tanuló algoritmus akkor jó, ha olyan hipotéziseket hoz létre, amelyek jól jósolják meg az általuk előzetesen nem látott példák osztályba sorolását. A 18.5. fejezetben azt mutatjuk be, hogy mi módon lehet előre megbecsülni a jóslás minőséget. Ebben a fejezetben egy olyan módszertant vizsgálunk meg, amellyel az osztályba soroló képességet méréssel lehet becsülni.

Created by XMLmind XSL-FO Converter.

Nyilvánvalóan akkor jó egy jóslás, ha igaznak bizonyul, így a hipotézis minőségét megbecsülhetjük az ismertté vált tényleges osztálybasorolások alapján. Ezt egy **teszthalmaz**nak (**test set**) nevezett mintahalmaz segítségével végezhetjük el. Ha az összes rendelkezésünkre álló példát tanításra használjuk, akkor továbbiakat kell gyűjtenünk a teszteléshez. Ezért gyakran kényelmesebb a következő módszert alkalmazni:

1. Gyűjtsünk egy nagy példahalmazt.
2. Osszuk két diszjunkt részre: a **tanító halmaz**ra (**training set**) és a **teszthalmaz**ra (**test set**).
3. Alkalmazzuk a tanító algoritmust a tanító halmazon, és így generáljunk egy *h* hipotézist.
4. Mérjük meg a teszthalmazon, hogy a *h* hipotézis a halmaz hány százalékára ad helyes osztálybasorolást.
5. Ismételjük meg az 1–4 lépéseket különböző tanító halmaz méretekre, és mindegyik mérethez különböző véletlenszerűen kiválasztott tanító halmazokra.

Ennek az eljárásnak az eredményeként egy adathalmazt kapunk, amelynek feldolgozásával megkaphatjuk az átlagos jóslási képességet a tanító halmaz méretének függvényében. Ezt a függvényt ábrázolva kapjuk az adott algoritmusnak egy adott tématerületre vonatkozó **tanulási görbé**jét (**learning curve**). A `DÖNTÉSI-FA-TANULÁS` algoritmus étterem példáinkkal felvett tanulási görbéje a 18.7. ábrán látható. Vegyük észre, hogy a tanító halmaz méretével javul a predikció minősége! (Ez okból az ilyen görbéket ún. **boldog görbé**knek (**happy graph**s) is nevezik.) Ez annak a jele ugyanis, hogy valóban van valami mintázat az adatokban, és az algoritmus felfedezi ezt a mintázatot.

18.7. ábra - Tanulási görbe az étterem problématérre véletlenszerűen generált 100 példa esetén. A görbe 20 kísérlet eredményét összegzi.



A tanuló algoritmusnak nyilvánvalóan nem szabad az előtt „látnia” a tesztadatokat, mielőtt a megtanított hipotézist teszteltük velük. Szerencsétlen módon nagyon könnyű beleesni abba a hibába, hogy tanítás közben a tesztadatokra **kukucskál**unk (**peeking**). Ez tipikusan a következő módon zajlik: egy tanuló algoritmus általában számos hangolási lehetőséggel rendelkezik, amellyel az algoritmus viselkedését változtatni tudjuk. Például számos különböző kritérium lehet, amelyek alapján a következő attribútumot kiválasztjuk a döntési fa tanulásnál. Különböző beállítások mentén számos különböző hipotézist generálunk, majd mindegyiket

Created by XMLmind XSL-FO Converter.

leteszteljük a teszthalmazon, és a legjobb hipotézis predikciós eredményét tekintjük eredménynek. Fájdalmas bár, de kukucskálás történt! Ennek oka, hogy a hipotézist *a teszthalmazon elért eredménye alapján választottuk ki,* így a teszthalmaz által hordozott információ beszivárgott a tanuló algoritmusba. Ennek a mesének az a tanulsága, hogy bármely eljárásnak, amelynek része a hipotéziseknek a teszthalmazon nyújtott teljesítményük alapján való összehasonlítása, egy *új* teszthalmazt kell használnia, amelyen a végül kiválasztott hipotézis teljesítményét méri. A gyakorlatban ez túl körülményes, így az emberek továbbra is ily módon szenynyezett adathalmazaikon folytatják kísérleteiket.

3.6. Zaj és túlilleszkedés

Korábban már láttuk, hogy ha van kettő vagy több olyan példa, amelyeknek azonos a leírásuk (az attribútumokra nézve), de eltérő az osztálybasorolásuk, akkor a `DÖNTÉSI-FA-TANULÁS` algoritmus nem lehet képes olyan döntési fát találni, amely minden példával konzisztens. Az általunk korábbiakban említett megoldás a következő. Ha determinisztikus osztályzásra van igény, akkor minden levélcsomópont adja vissza a hozzá tartozó halmaz többségi osztályát. Más esetekben pedig adja vissza a relatív gyakoriságok alapján becsült osztályba tartozási valószínűségeket. Sajnos, ez még távol van a történet végétől. Könnyen elképzelhető, sőt valójában elég valószínű, hogy a tanuló algoritmus akkor is talál egy olyan döntési fát, amely az összes példával konzisztens, ha nagyon fontos információ hiányzik. Ennek oka, hogy az algoritmus felhasználhat *irreleváns* attribútumokat, ha vannak ilyenek, hogy hamis megkülönböztetést tegyen a példák közt.

Vizsgáljuk meg azt a problémát, amikor egy kockadobás eredményét akarjuk megjósolni. Tegyük fel, hogy több napon át kísérleteket végzünk számos kockával. Mindegyik tanító példát a következő attribútumokkal írunk le:

1. *Nap:* az a nap, amelyen a kockadobást végeztük (hétfő, kedd, szerda, csütörtök).

2. *Hónap:* a hónap, amelyben a kockadobást végeztük (január vagy február).

3. *Szín:* a kocka színe (vörös vagy kék).

Mindaddig, amíg két dobásnak nem lesz pontosan azonos a leírása, a `DÖNTÉSI-FA-TANULÁS` képes lesz egzakt hipotézist találni. Minél több attribútum van, annál valószínűbb, hogy egzakt hipotézist kapunk. Viszont az összes ilyen hipotézis teljesen hamis lesz! Amit szeretnénk, az az, hogy a `DÖNTÉSI-FA-TANULÁS` – miután elég példát látott már – egyetlen levélcsomópontot adjon vissza, minden egyes dobási eredményhez közelítőleg 1/6 valószínűséggel.

Amikor a hipotézisek nagy halmaza lehetséges, akkor óvatosnak kell lennünk, nehogy arra használjuk az ebből eredő nagy szabadságot, hogy értelmetlen „szabályosságot” találjunk az adatokban. Ezt a problémát **túlilleszkedés**nek (**overfitting**) nevezzük. Ez rendkívül általános jelenség, akkor is jelentkezhet, amikor a keresett függvénynek egyáltalán nincs valószínűségi jellege. Mindamellett ez a probléma az összes tanulási algoritmust sújtja, nem csak a döntési fákat.

A túlilleszkedés kezelésének kimerítő matematikai tárgyalása meghaladja e könyv kereteit. A probléma megoldására itt csupán egy egyszerű – **döntési fa metszés** (**decision tree pruning**) nevű – módszert mutatunk be. A metszés (vagy nyesés) azon az alapon működik, hogy megakadályozzuk a nem nyilvánvalóan releváns attribútumok tesztje mentén a mintahalmaz ismételt (rekurzív) felosztását, még akkor is, ha az adatok osztálybasorolása az adott csomópontban nem egyforma. Az a kérdés, hogyan vesszük észre egy attribútumról, hogy irreleváns?

Tegyük fel, hogy egy irreleváns attribútumra alapozva osztottuk ketté a mintahalmazunkat. Általánosságban szólva azt várjuk, hogy ez esetben az eredményül kapott részhalmazokban nagyjából ugyanabban az arányban fognak szerepelni az egyes osztályokba tartozó minták, mint az eredeti halmazban. Ekkor az információnyereség közel nulla.[4] Tehát az információnyereség (annak hiánya) jó jelzés lehet az irrelevanciára. Most az a kérdés merül fel, hogy mekkorának kell lennie az információnyereségnek ahhoz, hogy ezen attribútum mentén szétosszuk a mintahalmazt?

Ezt a kérdést a statisztikai **szignifikanciatesztet** (**significance test**) használva válaszolhatjuk meg. Ez a teszt azzal a feltételezéssel indul, hogy a példákban egyáltalán nincs közös mintázat (ez az úgynevezett **nullhipotézis**). Ezek után az aktuális adathalmazt vizsgáljuk: meg akarjuk határozni annak mértékét, hogy az adathalmaz mennyire tér el a tökéletesen mintázat nélküli helyzettől. Ha az eltérés mértéke statisztikailag már

[4] Valójában a nyereség pozitív lesz, kivéve azt az esetet, amikor az arányok mind pontosan egyformák (lásd 18.10. feladat).

Created by XMLmind XSL-FO Converter.

valószínűtlen (rendszerint 5% vagy ennél kisebb valószínűséget értünk ezalatt), akkor ezt annak bizonyítékaként vesszük, hogy az adatokban jelen van egy alapvető mintázat. A valószínűséget a véletlen mintavételezés esetén várható eltérések standard eloszlását feltételezve számítjuk ki.

Ebben az esetben a nullhipotézis az, hogy az attribútum irreleváns, ennek megfelelően egy végtelen nagy mintahalmazra vett információnyereség nulla lenne. Azt kell kiszámítanunk, hogy a nullhipotézist feltéve egy ν méretű mintahalmazban a várt pozitív és negatív eseteloszlástól a megfigyelt eloszlás eltérése milyen valószínűséggel léphet fel. Az eltérés mértékét megadhatjuk a részhalmazok tényleges pozitív és negatív esetszámának (p_i, n_i) a nullhipotézis fennállása esetén várt esetszámokkal $(\hat{p}_i, \hat{n}_i)$ való összehasonlításával:

$$\hat{p}_i = p \times \frac{p_i + n_i}{p + n} \qquad \hat{n}_i = n \times \frac{p_i + n_i}{p + n}$$

A teljes eltérés kényelmesen számítható mértékét a következő összefüggés adja:

$$D = \sum_{i=1}^{\nu} \frac{(p_i - \hat{p}_i)^2}{\hat{p}_i} + \frac{(n_i - \hat{n}_i)^2}{n_i}$$

A nullhipotézist feltételezve a D eltérés mérték ν– 1 szabadságfokú χ^2 (khí-négyzet) eloszlást követ. Annak valószínűségét, hogy az attribútum valóban irreleváns, standard χ^2 táblázatok vagy statisztikai szoftver segítségével számíthatjuk ki. A 18.11. feladatban tűztük ki célul, hogy a `DÖNTÉSI-FA-TANULÁS` algoritmusban elvégezzük azokat a változtatásokat, amelyek a metszés ezen – χ^2 **metszés** (χ^2 **pruning**) néven ismert – formájához vezetnek.

A zaj metszés segítségével kezelhető: az osztályozási hibák lineáris növekedést okoznak a predikciós hibában, míg az esetleírásban fellépő hibák aszimptotikus hatást gyakorolnak, amely egyre rosszabb, ahogy a döntési fa egyre kisebb halmazok kezelésére zsugorodik. Ha az adatok nagy zajjal terheltek, akkor a metszéssel készült döntési fák lényegesen jobb eredményt adnak, mint a metszés nélkül készültek. Ráadásul a metszéssel készült fák gyakran jóval kisebbek, ezért könnyebben érthetők.

Egy másik túlilleszkedést csökkentő technika a **keresztvalidáció** (**cross-validation**). Bármely tanulási eljárásnál alkalmazható, nem csupán a döntési fa tanulásnál. Az alapvető ötlet annak megbecsülése, hogy az egyes hipotézisek mennyire jól fogják megjósolni a még nem látott esetekre adandó válaszokat. Ezt úgy becsülhetjük, hogy az ismert adatok egy részét félretesszük, és ezekkel teszteljük a megmaradt adatok alapján tanulással létrehozott fa predikciós képességét. *K*-szoros keresztvalidációnak nevezzük, ha *k* kísérletet végzünk, és minden esetben az adatok más és más 1/*k*-ad részét tesszük félre validációs tesztcélra, majd a végén átlagoljuk az eredményeket. Az 5 és a 10 elterjedten használt *k* értékek. Szélső esetként használják a $k = n$ választást, amelyet hagyj-ki-egyet keresztvalidációs módszerként ismerünk. A keresztvalidációt tetszőleges döntési fa tanulási módszerrel együtt alkalmazhatjuk (beleértve a metszést is), célja mindig az, hogy olyan döntési fa kiválasztását segítsük elő, amely jó predikciós képességgel rendelkezik. A kukucskálási jelenség elkerülése végett ezek után ezt a predikciós képességet egy új teszthalmazon kell mérnünk.

3.7. A döntési fák alkalmazhatóságának kiterjesztése

Egy sor problémát meg kell oldanunk ahhoz, hogy a döntési fa indukciót a problémák szélesebb körére kiterjeszthessük. Nagyjából felvázoljuk mindegyiket, azt javasolva, hogy a teljesebb megértés érdekében a kapcsolódó feladatokat oldja meg az olvasó.

- **Hiányzó adatok** (**missing data**): Számos területen nem ismerhető meg minden példa összes attribútuma. Lehet, hogy az értékek nem kerültek tárolásra, vagy túl drága lenne a mérésük. Ez két problémát vet fel: először, ha van is egy teljes döntési fánk, akkor hogyan tudunk egy olyan mintát osztályozni, amelynek egy tesztelendő attribútuma hiányzik? Másodszor, hogyan módosítsuk az információnyereségre vonatkozó formulát, ha néhány példa ezen attribútumának értéke nem ismert? Ezeket a kérdéseket a 18.2. feladatban vizsgáljuk.

- **Sokértékű attribútum**ok (**multivalued attribute**s): Ha egy attribútum nagyszámú értéket vehet fel, akkor az információnyereség nagysága nem megfelelő mértéke az attribútum hasznosságának. Szélsőséges esetben egy olyan attribútumot használnánk, mint például az *ÉtteremNév* attribútum, amely minden példára más és más értéket ad. Ez esetben példák minden részhalmaza egyelemű lesz, egyedi osztálybasorolással, így az

Created by XMLmind XSL-FO Converter.

információnyereség erre az attribútumra veszi fel maximális értékét. Mindamellett ez az attribútum irreleváns, értéktelen. Egy lehetséges megoldás a **nyereségarány** (**gain ratio**) használata (lásd 18.13. feladat).

- **Folytonos és egész értékű bemeneti attribútumok** (**continuous and integer-valued input attributes**): A folytonos vagy az egész értékű bemeneti attribútumok – mint például a *Magasság* és a *Súly* – által felvehető értékek halmaza végtelen. A döntési fa tanuló algoritmusok az ilyen esetekre nem generálnak végtelen elágazású csomópontokat, inkább megkeresik azt a **küszöbpont**ot (**split point**), amely a legnagyobb információnyereséget eredményezi. Például egy adott csomópontnál az lehet a helyzet, hogy a *Súly* > 160 teszt adja a legnagyobb információnyereséget. Léteznek hatékony dinamikus programozási módszerek jó küszöbpontok megtalálására, de még mindig messze ez a legtöbb erőforrást igénylő része a döntési fa tanulási eljárások valós problémákra való alkalmazásának.

- **Folytonos értékkészletű kimeneti attribútumok** (**continuous-valued output attributes**): Ha egy numerikus értéket akarunk megjósolni, például egy műtárgy árát, akkor nem osztályozásra van szükségünk, hanem egy **regressziós fá**ra (**regression tree**). Egy ilyen fa nem egyetlen értéket (osztályt) ad vissza az egyes levél csomópontjaiban, hanem egy attribútumhalmaz lineáris függvényét. Például a műtárgyak regreszsziós fájának kézzel festett metszetekre vonatkozó ága a területnek, a kornak és a színek számának lineáris függvényéhez vezethet. A tanuló algoritmusnak kell azt megoldania, hogy mikor álljon le a minták csomópontokban történő szétosztásával, áttérve a maradék attribútumokat (vagy azok bizonyos részhalmazát) felhasználó lineáris regresszió megalkotására.

Egy döntési fa tanulásra szolgáló, valós problémák megoldását célzó rendszernek képesnek kell lennie mindezen problémák kezelésére. A folytonos értékű változók kezelése különösen fontos, mivel mind a fizikai, mind a gazdasági folyamatok numerikus értékekkel jellemezhetők. Számos, üzleti forgalomban kapható programcsomag készült, amelyek megfelelnek mindezen a kritériumoknak, és segítségükkel több száz – valamilyen konkrét területen – alkalmazott rendszert fejlesztettek ki. Az ipar és kereskedelem sok területén, ha minták alapján kialakított osztályozásra van szükség, akkor elsősorban döntési fákkal próbálkoznak. A döntési fák egyik fontos tulajdonsága, hogy az ember számára jól érthető a tanuló algoritmus által előállított eredmény. (Valójában olyan gazdasági döntések esetén, amelyeknek megkülönböztetés elleni jogszabályoknak kell megfelelniük, ez *jogi követelmény*.) A neurális hálózatoknál hiányzik ez a fontos tulajdonság (lásd 20. fejezet).

4. Hipotézishalmaz együttes tanulása

Az eddigiekben olyan tanulási algoritmusokat vizsgáltunk, amelyek eredményeként a hipotézistérből kiválasztott egyetlen hipotézist használunk arra, hogy predikciót végezzünk. Az **együttes tanulás** (**ensemble learning**) alapötlete, hogy válasszunk ki a hipotézistérből egy teljes hipotéziskollekciót vagy hipotézis**együttes**t (**ensemble**), és kombináljuk az általuk adott predikciókat. Generálhatunk például ugyanazon tanító halmaz alapján száz különböző döntési fát, és egy új példa esetén szavazással alakíthatjuk ki a legjobb predikciós osztályozási eredményt.

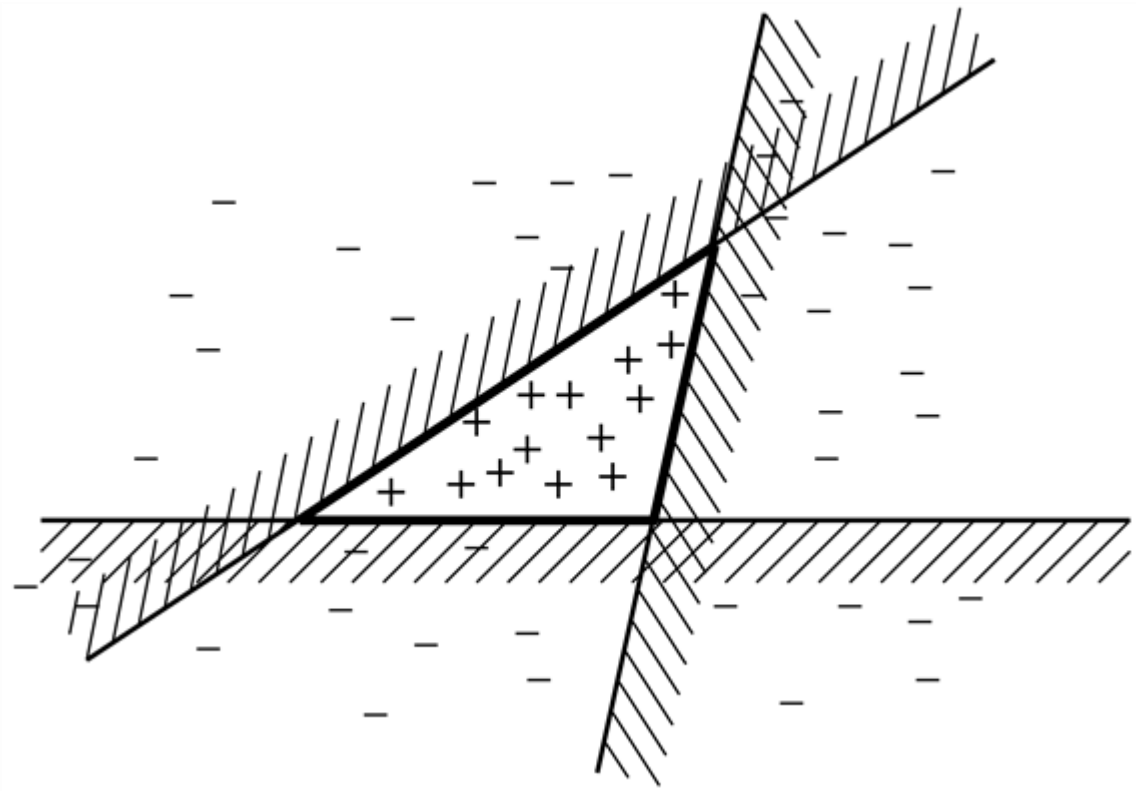
Az együttes tanulás motivációja kézenfekvő. Vizsgáljunk egy $M = 5$ hipotézisből álló együttest, és tegyük fel, hogy a predikciós eredményeket egyszerű többségi szavazással kombináljuk össze. Ahhoz, hogy az együttes rosszul osztályozzon egy új példát, *legalább háromnak az öt hipotézisből rossz osztályozási eredményt kell adnia!* Reményünk szerint ez sokkal kevésbé valószínű, mint az, hogy egyetlen hipotézis rossz osztályozásra jusson. Tegyük fel, hogy az együttes minden h_i hipotézisének p a hibavalószínűsége, azaz p annak valószínűsége, hogy h_i rosszul osztályoz egy véletlen módon kiválasztott példát. Továbbá tegyük fel, hogy a hipotézisek által elkövetett hibák *függetlenek*. Ez esetben – ha p kicsi – rendkívül kicsi a valószínűsége, hogy nagy számban történjék téves osztályozás. Egyszerű számítással (lásd 18.14. feladat) megmutatható például, hogy öt hipotézisből álló együttest használva a hibaarány 0,1-ről kevesebb mint 0,01-ra csökken. Valójában a függetlenség feltételezése nem racionális, mert a hipotéziseket a tanító halmaz félrevezető tulajdonságai hasonló módon vezetik félre. Mindamellett, ha a hipotézisek legalább kismértékben eltérőek, lecsökkentve a hibáik közti korrelációt, akkor az együttes tanulás nagyon hasznos lehet.

Az együttes tanulás tárgyalható más aspektusból is: gondolhatunk rá úgy, mint a hipotézistér egy általános kiterjesztésére. Tekintsünk úgy magára az együttesre, mint egy hipotézisre, és az új hipotézistér legyen mindazon lehetséges együttesek halmaza, amelyek az eredeti hipotézistérből konstruálhatók. A 18.8. ábra mutatja, hogyan eredményezhet ez egy sokkal kifejezőbb hipotézisteret. Ha az eredeti hipotézistér egyszerű és hatékony tanulási eljárást tett lehetővé, akkor az együttes tanulás módszere egy sokkal kifejezőbb hipotézisosztály tanulását biztosítja. Ennek általában nem túl nagy számítási és algoritmikus komplexitás növekedés az ára.

Created by XMLmind XSL-FO Converter.

A legelterjedtebben használt együttes tanulási módszert **turbózás**nak (**boosting**) nevezzük. Ahhoz, hogy működését megértsük, be kell vezetnünk a **súlyozott tanító halmaz** (**weighted training set**) fogalmát. Egy ilyen tanító halmazban minden mintához hozzárendelünk egy $w_j \geq 0$ súlyt. Minél magasabb ez a súly, annál nagyobb jelentőséget tulajdonítunk az adott mintának a hipotézis tanulása során. Az eddigiekben áttekintett tanuló algoritmusaink kézenfekvő módon módosíthatók úgy, hogy súlyozott tanító halmazokkal is tudjanak működni.[5]

18.8. ábra - A együttes tanulás eredményeképpen nyert megnövekedett kifejezőerőt illusztráló példa. Három lineáris küszöb jellegű hipotézist vizsgálunk, mindegyik a nem vonalkázott félsíkon ad pozitív besorolást. Az együttesben csak arra adunk pozitív osztályozást, amelyre mindhárom pozitívat ad. Az eredményül kapott háromszögterület egy olyan hipotézisnek felel meg, amelyet az eredeti hipotézistér egyetlen eleme sem képes kifejezni.



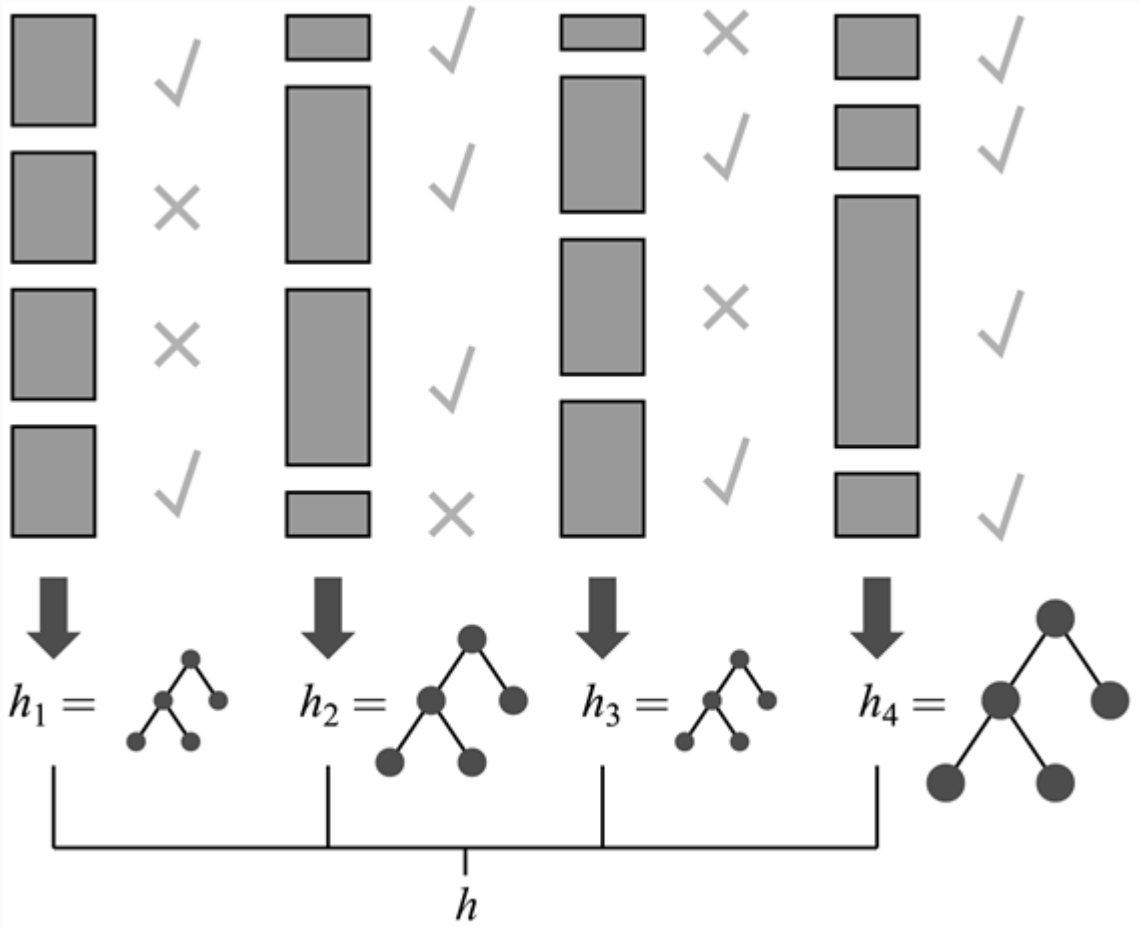
A turbózás az összes mintára $w_j = 1$ értékkel (azaz egy normál tanító halmazzal) indul. Ebből a halmazból generálja az első hipotézist, h_1-et. Ez a hipotézis egyes tanítómintákat jól osztályoz, másokat rosszul. Azt szeretnénk, ha a következő hipotézis jobb eredményt érne el a rosszul osztályozott mintákon, ezért megnöveljük a rosszul osztályozottak súlyait, míg a jól osztályozottakét csökkentjük. Ebből az új, súlyozott mintahalmazból generáljuk a h_2 hipotézist. Az eljárást addig folytatjuk, míg M hipotézist nem generáltunk, ahol M a turbó tanulási algoritmus bemenő paramétere. Az algoritmus eredményeként kapott hipotézis együttes nem más, mint az M hipotézis súlyozott többségi szavazással kombinált eredője. A súlyokat aszerint határozzuk meg, hogy az egyes hipotézisek mennyire jól teljesítettek a tanító halmazon. A 18.9. ábrán bemutatjuk az algoritmus koncepcióját. Sok variánsa ismert a turbó tanulásnak, eltérő súlymódosítással, illetve a hipotézisek különböző kombinálási lehetőségeivel. Egy kiválasztott – `ADABOOST` nevű – algoritmust mutatunk be a 18.10. ábrán. Az alkalmazott súlymódosítási eljárás jelen esetben nem túl lényeges, viszont az `ADABOOST` algoritmusnak van egy nagyon lényeges jellemzője. Tegyük fel, hogy a használt L tanulási algoritmus egy **gyenge tanulás**i (**weak learning**) algoritmus, ami azt jelenti, hogy L mindig visszaad egy olyan hipotézist, amelyik legalább egy csöppet jobban teljesít a tanító halmaz súlyozott hibájára nézve, mint a véletlen találgatás (azaz eredménye jobb mint 50% bináris osztályozás esetén). Ez esetben, kellően nagy M-re az `ADABOOST` mindig előállít egy olyan

[5] Azon tanulási eljárásokra, ahol ez nem lehetséges, **ismétléses tanító halmaz**t (**replicated training set**) hozhatunk létre, ahol a j-edik minta a w_j súllyal arányos példányszámban szerepel. Az esetleges nem egész súlyokat úgy kezelhetjük, hogy a mintaszámot valószínűségi változóként fogjuk fel.

Created by XMLmind XSL-FO Converter.

hipotézist, amely *a tanító mintákat tökéletesen osztályozza*! Tehát az algoritmus *fokozza* az eredeti tanuló eljárásnak a tanító mintán mért pontosságát. Ez mindig fennáll, attól függetlenül, hogy esetleg mennyire kevéssé kifejező az eredeti hipotézistér, és mennyire bonyolult a keresett függvény.

18.9. ábra - A turbó algoritmus működése. Mindegyik árnyékolt téglalap egy példának felel meg, a téglalap magassága mutatja a példa súlyát. A pipák és keresztek mutatják, hogy a példát jól vagy rosszul osztályoztuk a pillanatnyi hipotézis alapján. A döntési fa mérete mutatja, hogy ennek a hipotézisnek mekkora a súlya a végső együttesben.



Vizsgáljuk meg, hogy mennyire jól működik a turbózás az éttermi adatainkon. Kiinduló hipotézistérként az úgynevezett **döntési tönk**ök (**decision stump**s) terét választjuk, amelyek olyan döntési fák, amelyek csupán egy – a gyökércsomópontban elhelyezkedő – tesztből állnak. A 18.11. (a) ábra alsó görbéje mutatja, hogy turbózás nélkül ezek a döntési tönkök nem adnak valami jó eredményt, csupán 81%-os a 100 mintán mért predikciós teljesítményük. Amikor turbózást alkalmazunk (M = 5), akkor a teljesítmény javul, eléri a 93%-ot 100 példa esetén.

18.10. ábra - Az együttes tanulás turbó módszereinek ADABOOST nevű variánsa. Az algoritmus hipotézisek kombinációját hozza létre. A SÚLYOZOTT-TÖBBSÉG egy hipotézist generál. Ez az új hipotézis azt a kimeneti értéket adja vissza, amely a h-ban található hipotézisek közül, a z-ben számon tartott súlyozással a legtöbb szavazatot kapta.

Created by XMLmind XSL-FO Converter.

```
function AdaBoost(példák, L, M) returns egy súlyozott többségi hipotézis
  inputs: példák, minták N elemű címkézett halmaza (x1, y1), ..., (xN, yN)
          L, egy tanuló algoritmus
          M, az együttesben szereplő hipotézisek száma
  local variables: w, az N mintához rendelt súlyok vektora, indulásnál 1/N
                   h, az M darab hipotézisből képzett vektor
                   z, az M hipotézishez rendelt súlyok vektora
  for m = 1 to M do
    h[m] ← L(példák, w)
    error ← 0
    for j = 1 to N do
    if h[m](xj) ≠ yj then error ← error + w[j]
    for j = 1 to N do
    if h[m](xj) ≠ yj then w[j] ← w[j] · error/(1 − error)
    w ← Normalizál(w)
    z[m] ← log(1 − error)/error
  return Súlyozott-Többség(h, z)
```

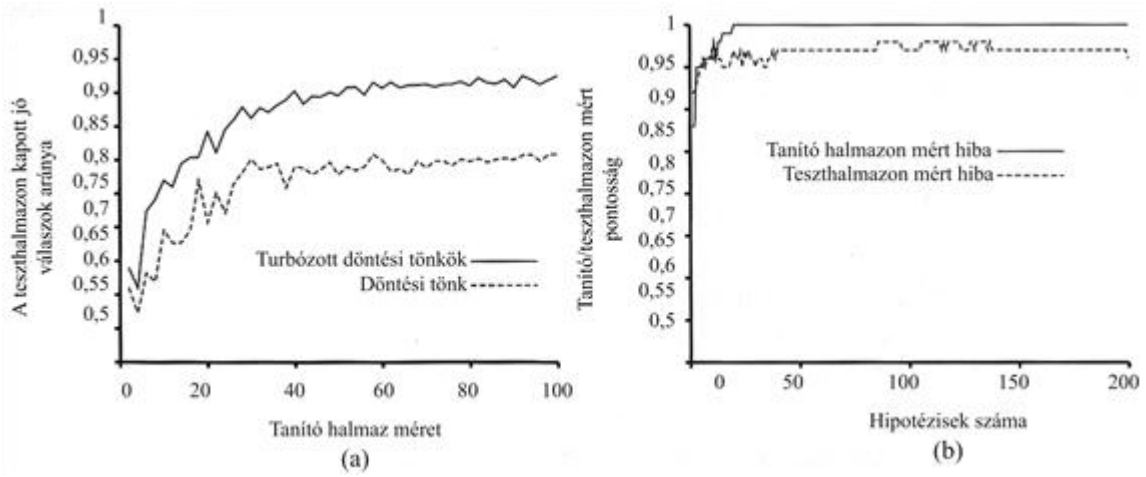
Fontos

Érdekes dolog történik, amikor az együttes M mérete növekszik. A 18.11. (b) ábra mutatja a tanító halmazon mért teljesítményt M függvényében. Figyeljük meg, hogy M = 20-nál a hiba eléri a nullát (ahogy a turbózással foglalkozó elmélet megjósolta), azaz 20 döntési tönk súlyozott többségi szavazással összefogott kombinációja elégséges a 100 mintára való pontos illeszkedéshez. Ha további tönköket adunk az együtteshez, akkor a hiba nulla marad. A görbe mutatja ugyanakkor, hogy *a teszthalmazon mért teljesítmény még hosszasan nő azután is, hogy a tanító halmazon mért hiba elérte a nullát*. M = 20 esetén a teszthalmazon mért teljesítmény 0,95 (a hiba 0,05), M = 137-ig növelve az együttes résztvevőinek számát a teljesítmény 0,98-ra nő, ezután leesik 0,95-re.

Ez a megfigyelés, amely különböző mintahalmazokra és hipotézisterekre nézve meglehetősen robusztusnak bizonyult, nagy meglepetést keltett a felfedezésekor. Az Ockham borotvája elv azt sugallja, hogy ne tegyük a hipotéziseket komplexebbé, mint szükséges, de a görbe azt mutatja, hogy a predikciók *javulnak,* ahogy az együttes hipotézis összetettebbé válik! Számos magyarázatot javasoltak erre a jelenségre. Az egyik megközelítés szerint a turbózás nem más, mint a **Bayes-tanulás** közelítése (lásd 20. fejezet), amely egy bizonyíthatóan optimális tanulási eljárás. Ez a közelítés javul, ahogy több és több hipotézist adunk az együtteshez. Másik lehetséges magyarázat, hogy további hipotéziseknek az együtteshez adása lehetővé teszi, hogy *határozottabb* legyen a megkülönböztetés a pozitív és negatív példák között, ami segít az új minták osztályozásában.

18.11. ábra - (a) A döntési tönkök és a turbózott döntési tönkök teljesítményének az étterem példán történő összehasonlítása M = 5 esetén. (b) A tanító halmazon és a teszthalmazon kapott jó válaszok aránya M-nek – az együttesben található hipotézisek számának – a függvényében. Figyeljük meg, hogy a teszthalmazon mért pontosság még azután is javul egy kicsit, amikor a tanító halmazon elérte az 1-et, azaz miután az együttes pontosan illeszkedett az adatokra.

Created by XMLmind XSL-FO Converter.



5. Miért működik a tanulás: a tanulás számítási elmélete

A legfontosabb megválaszolatlan kérdés, amelyet a 18.2. alfejezetben tettünk fel, a következő: hogyan bizonyosodhat meg valaki arról, hogy a tanulási algoritmusa által létrehozott elmélet helyesen fogja megjósolni a jövőt? Formálisan: honnan tudjuk, hogy a *h* hipotézis jól közelíti az *f* keresett függvényt, ha nem ismerjük *f*-et? Évszázadok óta foglalkoznak ezekkel a kérdésekkel. Amíg nem találjuk meg a válaszokat, addig a gépi tanulást – a legjobb esetben is – csak zavarba hozza saját sikere.

Fontos

Az ebben az alfejezetben tárgyalt megközelítés a **tanulás számítási elméletén** (**computational learning theory**) alapul, amely az MI, a statisztika és az elméleti számítástudomány közös határterülete. Ennek az elméletnek az alapvető elve a következő: *bármely súlyosan hibás elmélet szinte bizonyosan nagy valószínűséggel felismerhető kisszámú példa vizsgálata alapján, mivel helytelen predikciót fog adni. Tehát minden hipotézis, amely egy kielégítően nagy tanító példahalmazzal konzisztens választ ad, nem valószínű, hogy súlyosan hibás lenne: azaz* ***valószínűleg közelítőleg helyes****nek* ***(probably approximately correct)*** *kell lennie.* Minden olyan tanuló algoritmust, amely valószínűleg közelítőleg helyes hipotéziseket ad, **VKH-tanuló** (**PAC-learning**) algoritmusnak nevezünk.

Van néhány bonyolultabb rész az előbbi gondolatmenetben. A fő kérdés a tanító és a tesztpéldák viszonya: végül is mi azt szeretnénk, ha a hipotézis nem csupán a tanító halmazon lenne közelítőleg helyes, hanem elsősorban a teszthalmazon. Az alapvető feltevés az, hogy a tanító és a teszthalmazt *ugyanolyan valószínűség-eloszlást* használva, véletlenszerűen és függetlenül választottuk ugyanabból a példapopulációból. Ezt **stacionaritási** (**stationarity**) feltevésnek nevezzük. A stacionaritási feltevés nélkül az elmélet semmilyen, a jövőre vonatkozó igénnyel nem léphet fel, mivel nem lesz szükségszerű kapcsolat a múlt és a jövő között. A stacionaritási feltevés támasztja alá azt a gondolatot, hogy a példákat kiválasztó eljárás nem rosszindulatú. Persze ha a tanító halmaz kizárólag furcsa példákat tartalmaz – kétfejű kutyákat például –, akkor a tanuló algoritmus nyilvánvalóan nem tehet mást, mint sikertelen általánosításra jut a felől, hogy mi módon lehet a kutyákat felismerni.

5.1. Hány példára van szükség?

Néhány definícióra szükségünk lesz ahhoz, hogy ezeket a felismeréseket a gyakorlatba át tudjuk ültetni:

- Jelöljük **X**-szel az összes lehetséges példák halmazát.
- Jelölje *D* azt a valószínűségi eloszlást, amely alapján a példákat választjuk.
- Legyen **H** a lehetséges hipotézisek halmaza.
- Legyen *N* a tanító halmaz elemeinek a száma.

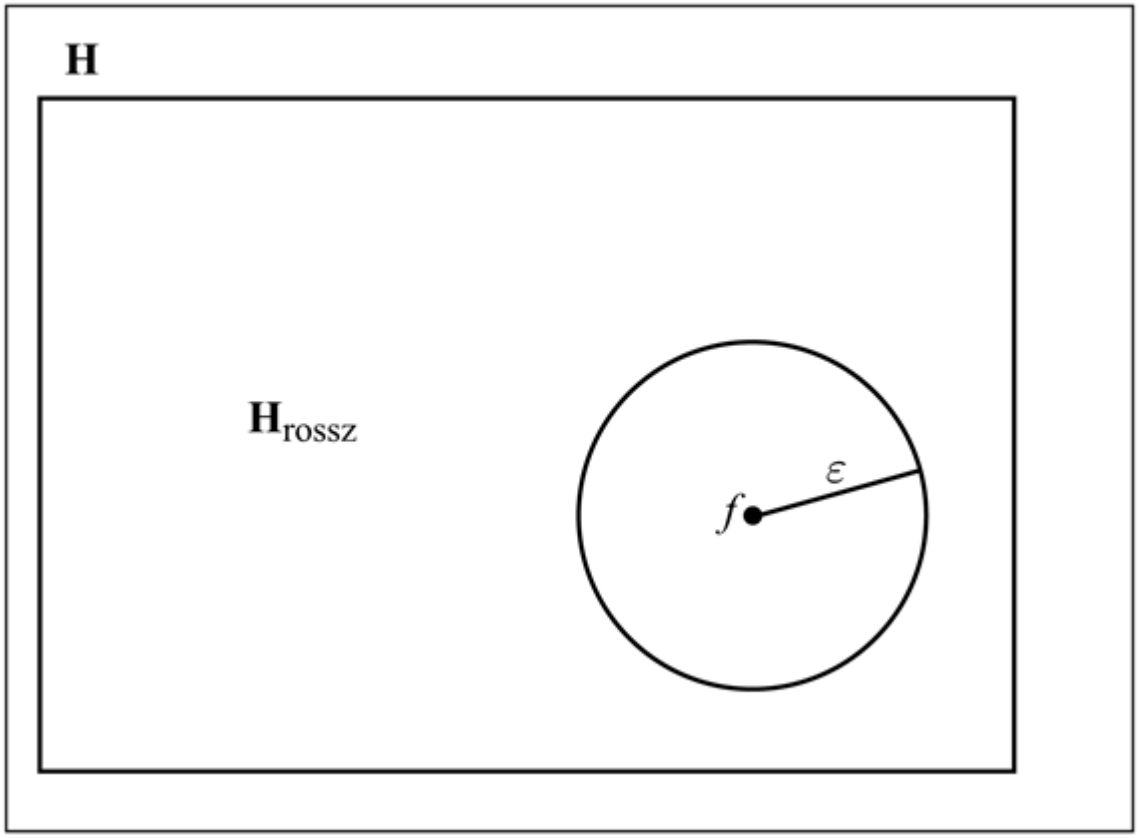
Created by XMLmind XSL-FO Converter.

Előzetesen feltesszük, hogy a keresett *f* függvény eleme **H**-nak. Most már definiálhatjuk azt a **hibá**t (**error**), ami egy *h* hipotézisnek a keresett *f* függvénytől való eltérését jellemzi a mintákra vonatkozó adott eloszlás mellett, mint annak valószínűségét, hogy *h* az *f*-től eltérő választ ad egy példára:

$$error(h) = P(h(x) \neq f(x) | x \text{ a } D \text{ alapján választva})$$

Ez ugyanaz a mennyiség, mint amelyet az előzőkben bemutatott tanulási görbékkel mértünk a gyakorlatban.

18.12. ábra - A hipotézistér sematikus ábrázolása a keresett *f* függvény körül felvett „ *ε* -gömb” ábrázolásával



Egy *h* hipotézist **közelítőleg helyes**nek (**approximately correct**) nevezünk, ha error(*h*) ≤ *ε*, ahol *ε* egy kis konstans. A bizonyítás megközelítése az, hogy megmutatjuk: *N* példa vizsgálata után az összes konzisztens hipotézis nagy valószínűséggel közelítőleg helyes lesz. Úgy gondolhatunk egy közelítőleg helyes hipotézisre, mint amely közel van a keresett függvényhez a hipotézistérben: egy – a keresett *f* körül felvett – **ε-gömb**ön (**ε-ball**) belül van. A 18.12. ábrán bemutatjuk az összes hipotézis **H** terét, amelyet két részre osztottunk, egyik az *ε*-gömb *f* körül, a másik a $\mathbf{H}_{rossz}$-szal jelölt maradék.

A következők alapján kiszámíthatjuk annak valószínűségét, hogy egy „súlyosan hibás” $h_r \in \mathbf{H}_{rossz}$ hipotézis konzisztens lesz az első *N* példával. Tudjuk, hogy error(h_r) > *ε*. Tehát annak valószínűsége, hogy egy adott példára helyes választ ad, legfeljebb 1 – *ε*. Ezek után az *N* példára vonatkozó összefüggés:

$$P(h_r \text{ konzisztens } N \text{ példával}) \leq (1 - \varepsilon)^N$$

Annak valószínűségét, hogy $\mathbf{H}_{rossz}$ legalább egy konzisztens hipotézist tartalmaz, a lehetséges egyedi hipotézisek száma határozza meg:

$$P(H_{rossz} \text{ tartalmaz egy konzisztens hipotézist}) \leq |H_{rossz}|(1 - \varepsilon)^N \leq |H|(1 - \varepsilon)^N$$

Kihasználtuk, hogy $|\mathbf{H}_{rossz}| \leq |\mathbf{H}|$. Ezt a valószínűséget valamely kellően kis *δ* érték alá akarjuk csökkenteni:

$$|H|(1- \varepsilon)^N \leq \delta$$

Felhasználva, hogy $(1 - \varepsilon) \leq e^{-\varepsilon}$, a célunk elérhető, ha lehetővé tesszük, hogy az algoritmus

Created by XMLmind XSL-FO Converter.

$$N \geq \frac{1}{\varepsilon}\left(\ln\frac{1}{\delta} + \ln|\mathbf{H}|\right) \qquad (18.1)$$

példát megvizsgálhasson. Ennek értelmében, ha egy algoritmus ilyen mennyiségű példával konzisztens hipotézist ad vissza, akkor legalább $1 - \delta$ valószínűséggel a hibája legfeljebb ε. Más szavakkal közelítőleg helyesnek nevezzük. A szükséges példaszámot, ami ε és δ függvényeként adható meg, a hipotézistér **minta komplexitás**ának (**sample complexity**) nevezzük.

Kiderült, hogy kulcskérdés a hipotézistér mérete. Mint korábban láttuk, ha **H** az *n* attribútumon felvehető Boole-függvények halmaza, akkor $|\mathbf{H}| = 2^{2^n}$. Tehát a tér minta komplexitása 2^n szerint nő. Mivel a lehetséges példák száma szintén 2^n, ebből az következik, hogy a Boole-függvények terében egyetlen tanuló algoritmus sem tud jobb eredményt elérni, mint egy táblázat, ha csupán arra szorítkozik, hogy egy olyan hipotézist adjon vissza, amely az összes ismert példával konzisztens. Ennek megmutatására egy másik lehetőség az, hogy megfigyeljük: egy tetszőleges ismeretlen példára a hipotézistér ugyannyi pozitív kimenetelt jósoló konzisztens hipotézist tartalmaz, mint amennyi negatív kimenetelt jósol.

A következő dilemmával kerültünk szembe: ha nem korlátozzuk a szóba jövő függvények terét, akkor az algoritmus nem lesz képes tanulni, viszont ha korlátozzuk a függvények terét, akkor fennáll a veszély, hogy kihagyjuk a keresett függvényt. Két lehetőségünk van, hogy „kimeneküljünk” ebből a dilemmából. Az első, hogy nem csupán ahhoz ragaszkodunk, hogy az algoritmus egy tetszőleges konzisztens hipotézist adjon vissza, hanem ahhoz is, hogy részesítse előnyben az egyszerűeket (mint ahogy a döntési fa tanulásnál tettük). Ezen algoritmusok elméleti tárgyalása túlmegy ennek a könyvnek a keretein, de megemlítjük, hogy azon esetekben, amikor az egyszerű konzisztens hipotézisek keresése kezelhető probléma, a mintakomplexitás-eredmények jobbak, mint a csupán konzisztencián alapuló vizsgálatoké. A másik menekülési lehetőség, amelyet követni is fogunk, hogy a Boole-függvények teljes halmazából a megtanulható függvények részhalmazára fogunk koncentrálni. Az ötlet abban áll, hogy legtöbb esetben nincs szükségünk a Boole-függvények teljes kifejezőerejére, korlátozottabb nyelvek is kielégítők számunkra. A következőkben egy ilyen korlátozottabb nyelvet vizsgálunk meg részletesebben.

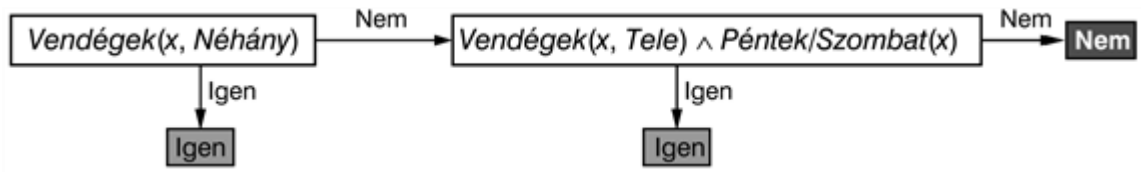
5.2. Döntési listák tanulása

A **döntési lista** (**decision list**) egy kötött formájú logikai kifejezés. Egy tesztsorozatból áll, amely tesztek mindegyike literálisok konjunkciója. Ha egy teszt valamelyik példa leírására alkalmazva pozitív eredményt ad, akkor a döntési lista határozza meg a példára adandó választ. Ha a teszt eredménye negatív, akkor a lista következő tesztjével folytatódik a feldolgozás.[6] A döntési listák emlékeztetnek a döntési fákra, de a globális struktúrájuk egyszerűbb, ezzel szemben az egyes tesztek bonyolultabbak, mint a döntési fáknál. A 18.3. ábra egy döntési listát mutat be, amely a következő hipotézist reprezentálja:

$$\forall x\ \mathit{VárjunkE}(x) \Leftrightarrow \mathit{Vendégek}(x, \mathit{Néhány}) \vee (\mathit{Vendégek}(x, \mathit{Tele}) \wedge \mathit{Péntek/Szombat}(x))$$

Ha megengedünk tetszőleges méretű teszteket, akkor a döntési listák bármely Boole-függvény reprezentálására képesek (lásd 18.15. feladat). Másrészről, ha az egyes tesztek méretét *k* literálisra korlátozzuk, akkor lehetővé válik, hogy a tanuló algoritmus kisszámú példa alapján sikeresen általánosítson. Ezt **k-DL** nyelvnek nevezzük. Könnyen megmutatható, hogy a *k*-DL nyelvek részhalmazként tartalmazzák a **k-DF** nyelveket, amelyek a legfeljebb *k* mélységű döntési fák halmazát alkotják. Fontos, hogy egy adott nyelv, amelyre *k*-DL jelöléssel hivatkozunk, függ azoktól az attribútumoktól, amelyekkel a példákat leírjuk. A továbbiakban *k*-DL(*n*) lesz a jelölésünk azokra a *k*-DL nyelvekre, amelyek *n* logikai attribútumot használnak.

18.13. ábra - Az éttermi problémára alkotott döntési lista



[6] Egy döntési lista tehát szerkezetében megegyezik a Lisp nyelv cond állításával.

Created by XMLmind XSL-FO Converter.

Első feladatunk annak megmutatása, hogy a *k*-DL nyelvek megtanulhatók, azaz bármely *k*-DL függvény pontosan közelíthető, ha kellő számú tanító mintán tanultunk. Ehhez ki kell számítanunk a nyelvben felvehető hipotézisek számát. Legyen a tesztek nyelve – *n* attribútumot használva, legfeljebb *k* literális konjunkciója – *Conj*(*n*, *k*). A döntési listákat tesztekből építjük fel, és minden teszthez egy *Igen* vagy *Nem* választ rendelhetünk, illetve harmadik lehetőségként a teszt hiányozhat a listáról, ezért legfeljebb $3^{|Conj(n,k)|}$ különböző teszthalmaz létezhet. Ezek a tesztek tetszőleges sorrendben alkalmazhatók, így

$$|k\text{-DL}(n)| \leq 3^{|Conj(n,k)|}|Conj(n,k)|!$$

Az *n* attribútumot használó *k* literális lehetséges konjunkcióinak száma:

$$\left|Conj(n,k)\right| = \sum_{i=0}^{k}\binom{2n}{i} = O\left(n^k\right)$$

Így néhány lépés után:

$$\left|k-\text{DL}(n)\right| = 2^{O\left(n^k \log_2\left(n^k\right)\right)}$$

Ezt a (18.1) egyenletbe helyettesítve megmutathatjuk, hogy a *k*-DL függvények VKH-tanulásához szükséges minták száma polinomiálisan nő *n*-nel:

$$N \geq \frac{1}{\varepsilon}\left(\ln\frac{1}{\delta} + O\left(n^k \log_2\left(n^k\right)\right)\right)$$

Ennek értelmében bármely algoritmus, amely konzisztens döntési listát ad vissza, kis *k* értékek esetén elfogadható példaszám alapján VKH-tanulással képes előállítani egy *k*-DL függvényt.

A következő feladatunk, hogy hatékony algoritmust találjunk, amely konzisztens döntési listát eredményez. A DÖNTÉSI-LISTA-TANULÁS nevű mohó algoritmust fogjuk használni. Ez ismételten talál egy olyan tesztet, amely pontosan a tanító halmaz valamely részhalmazának felel meg. Amikor talál egy ilyen tesztet, akkor azt hozzáadja a döntési listához, egyben eltávolítva a tesztnek megfelelő példákat a tanító halmazból. Ezek után létrehozza a döntési lista hátralevő részét pusztán a maradék példákra alapozva. Ezt addig ismétli, amíg nem marad több példa. Az algoritmust a 18.14. ábra mutatja.

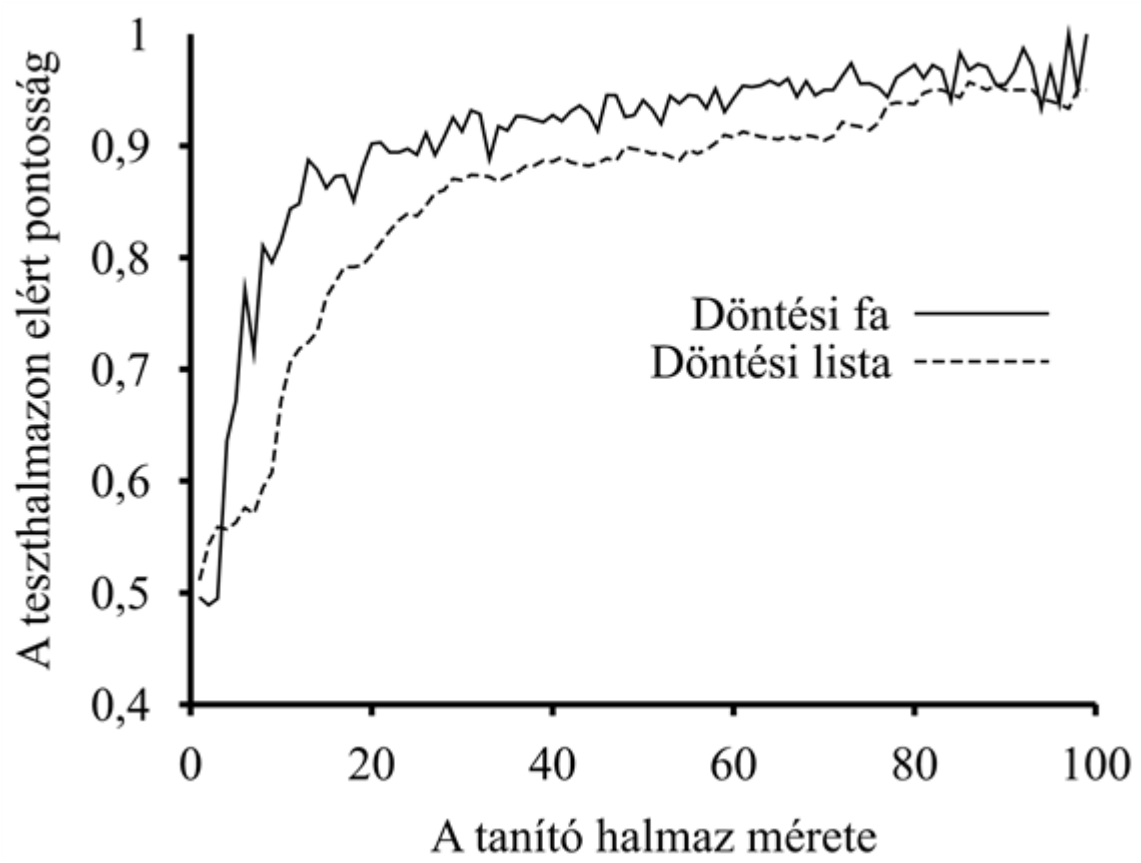
Ez az algoritmus nem specifikálja, hogy milyen módszert használjunk a következő – a döntési listához adni kívánt – teszt kiválasztásához. Bár a bemutatott formálisan elért eredmények nem függnek a kiválasztás módszerétől, mégis az látszik józannak, ha előnyben részesítjük azokat az egyszerű teszteket, amelyek az osztályozott példák nagy részhalmazainak felelnek meg. Ennek eredményeként a döntési lista a lehető legtömörebb lesz. A legegyszerűbb stratégia, ha azt a legrövidebb *t* tesztet keressük meg, amely egy tetszőleges, egyforma osztálybasorolással rendelkező részhalmaznak megfelel, figyelmen kívül hagyva a részhalmaz méretét. Még ez a megközelítés is egész jól működik, mint az a 18.15. ábrán látható.

18.14. ábra - Egy döntési lista tanulásra használható algoritmus

function DÖNTÉSI-LISTA-TANULÁS(*példák*) **returns** egy döntési lista vagy *kudarc*
  if *példák* üres **then return** a triviális döntési lista, *Nem*
  $t \leftarrow$ egy teszt, amely pontosan megfelel a *példák* egy $példák_t$ nemüres részhalmazának, azaz a $példák_t$ minden elemére a teszt vagy pozitív, vagy negatív értéket ad
  if nincs ilyen *t* **then return** *kudarc*
  if $példák_t$ elemeire a teszt pozitív **then** $o \leftarrow$ *Igen* **else** $o \leftarrow$ *Nem*
  return egy döntési lista, amelynek első eleme *t*, a válasz *o*, és a maradék teszteket a DÖNTÉSI-LISTA-TANULÁS(*példák* – $példák_t$) adja

18.15. ábra - A DÖNTÉSI-LISTA-TANULÁS algoritmusnak az éttermi adatokon felvett tanulási görbéje. Az összehasonlítás kedvéért feltüntettük a DÖNTÉSI-FA-TANULÁS algoritmus görbéjét is.

Created by XMLmind XSL-FO Converter.



5.3. Elemzés

A tanulás számítási elmélete új szempontból világította meg a tanulás problémáját. Az 1960-as évek elején a tanulás elmélet az **identifikáció határátmenet**ben (**identification in the limit**) problémájára koncentrált. E szerint egy identifikációs algoritmusnak mindenképpen vissza kell adnia egy olyan hipotézist, amely pontosan megegyezik a keresett függvénnyel. Erre a következőkben ismertetett eljárás ad egy lehetséges módszert. Először is rendezzük a **H** hipotézistér minden elemét valamilyen egyszerűség mérték szerint. Ezek után válasszuk azt a legegyszerűbb hipotézist, amely konzisztens az összes már rendelkezésre álló példával. Ahogy újabb példák érkeznek, a módszer fel fogja adni az egyszerűbb hipotézist, ha az új példák érvénytelenítették, és helyette egy bonyolultabbat választ. Amikor eléri a keresett függvényt, akkor azt már soha nem fogja feladni. Sajnálatos, hogy számos hipotézistérben óriási nagy lehet a keresett függvény eléréséhez szükséges idő, valamint a szükséges példák száma. Ezért a tanulás számítási elmélete nem ragaszkodik ahhoz, hogy a tanuló ágens megtalálja a környezetét vezérlő „egyedüli helyes törvényt". Ehelyett inkább egy olyan hipotézist keres, amely egy bizonyos prediktív pontosságot elér. A tanulás számítási elmélete ezen felül erősen hangsúlyozza a hipotézis nyelv kifejezőképessége és a tanulás komplexitása közötti kompromisszum fontosságát. Ez az elmélet vezetett egy fontos tanuló algoritmus osztályhoz, a szupport vektorgépekhez.

Az általunk bemutatott, VKH-tanulásra vonatkozó eredmények a legrosszabb esetre érvényesek, nem feltétlenül mutatva azt az átlagos esetre vonatkozó mintakomplexitást, amelyeket viszont a bemutatott tanulási görbék tükröznek. Egy, az átlagos esetre vonatkozó analízisnek további feltételezésekkel kell élnie a minták eloszlásáról, valamint a keresett függvények eloszlásáról. Miközben ezeket a kérdéseket egyre jobban megértjük, a tanulás számítási elmélete folyamatosan a gépi tanulással foglalkozó kutatók hasznos útmutatójának bizonyul. Ezek a kutatók algoritmusaik módosításában vagy algoritmusaik tanulási képességének meghatározásában érdekeltek. A döntési listák mellett a Boole-függvények szinte minden ismert alosztályára, a neurális hálókra (lásd 20. fejezet), valamint az elsőrendű logikai állítások halmazaira (lásd 19. fejezet) születtek eredmények. Az elért eredmények azt mutatják, hogy a tisztán indukciós tanulás rendkívül nehéz. Tisztán indukciós tanuláson azt a helyzetet értjük, amelyben az ágens nem rendelkezik semmilyen, a keresett függvényre vonatkozó, előzetes ismerettel. Mint a 19. fejezetben megmutatjuk, az előzetes tudásnak az induktív tanulás vezérlésére való felhasználása lehetővé teszi, hogy meglehetősen nagy állításhalmazokat elfogadható mintaszám alapján megtanuljunk, még egy olyan nyelven is, amelynek kifejezőereje megegyezik az elsőrendű logikáéval.

Created by XMLmind XSL-FO Converter.

6. Összefoglalás

Ebben a fejezetben determinisztikus függvények példák alapján való induktív tanulására koncentráltunk. A főbb pontok a következők voltak:

- A tanulásnak számos formája van, amely a cselekvő alrendszer jellegétől, a javítani kívánt komponenstől és a rendelkezésre álló visszacsatolástól függ.
- Ha akár egy tanár, akár a környezet lehetővé teszi a minták helyes értékének visszacsatolását, akkor **ellenőrzött tanulás**i (**supervised learning**) problémáról beszélünk. A feladat, amelyet **induktív tanulás**nak (**inductive learning**) is nevezhetünk, abban áll, hogy egy függvényt kell megtanulnunk annak bemeneti és kimeneti mintái alapján. Diszkrét értékkészletű függvény tanulását **osztályozás**nak (**classification**), folytonos értékű függvény tanulását pedig **regresszió**nak (**regression**) nevezzük.
- Az induktív tanulás magában foglalja, hogy egy, a példákkal **konzisztens** hipotézist kell találnunk. Az **Ockham borotvája** (**Ockham's razor**) elv értelmében célszerű a legegyszerűbb konzisztens hipotézis választása. Ennek a feladatnak a nehézségét a választott reprezentáció döntően befolyásolja.
- A **döntési fá**k (**decision tree**s) alkalmasak tetszőleges Boole-függvény reprezentálására. Az **információnyereség**re (**information gain**) alapozó heurisztika hatékony módszert biztosít egyszerű, konzisztens döntési fák konstruálására.
- A tanulási algoritmus teljesítményét a **tanulási görbé**n (**learning curve**) mérhetjük le, amely a **teszthalmaz**on (**test set**) mért predikciós pontosságot mutatja a **tanító halmaz** (**training set**) méretének függvényében.
- Az együttes tanulási módszerek, például a **turbózás** (**boosting**), gyakran jobb eredményt érnek el, mint az egyedi módszerek.
- A **tanulás számítási elmélete** (**computational learning theory**) az induktív tanulás minta komplexitásának és számítási komplexitásának analízisére koncentrál. Kompromisszumot kell kötnünk a hipotézis nyelv kifejezőképessége és a tanulás nehézsége között.

6.1. Irodalmi és történeti megjegyzések

Az 1. fejezet felvázolta az induktív tanulás filozófiai vizsgálatának történetét. William of Ockham (1280–1349) volt korának legnagyobb hatású filozófusa és a középkori ismeretelmélet, a logika és a metafizika egyik nagy alakja. Neki tulajdonítják azt a mondást, ami „Ockham borotvája" néven rögzült a köztudatban. Latinul: *Entia non sunt multiplicanda praeter necessitatem,* ami azt jelenti magyarul, hogy „A dolgokat nem kell a szükségesnél jobban bonyolítani". Sajnos ez a dicséretes mondás sehol sem található meg a műveiben pontosan így megfogalmazva.

Az EPAM (Elementary Perceiver And Memorizer, Elemi észlelő és megjegyző) volt az első rendszerek egyike, amely döntési fákat (**diszkriminációs hálók**at, **discrimination net**s) használt (Feigenbaum, 1961). Az EPAM létrehozásának célja az emberi fogalom-tanulás kognitív-szimulációs modelljének megalkotása volt. A CLS-rendszer (Hunt és társai, 1979) heurisztikus előretekintő eljárást használt a döntési fák konstrukciójára. Az ID3 (Quinlan, 1979) hozta azt a kulcsfontosságú ötletet, hogy a heurisztikus függvény használja az információtartalmat. Az információelméletet Claude Shannon fejlesztette ki a kommunikáció tanulmányozására (Shannon és Weaver, 1949). (Shannon hozzájárult a gépi tanulás területéhez is, az egyik legkorábbi példát szolgáltatva mechanikus egerével, amely egy útvesztőben navigált a kísérlet-kudarc elvet használva.) A döntési fák χ^2 metszését Quinlan írta le először (Quinlan, 1986). Az ipari felhasználásra alkalmas C4.5 döntési fa programcsomag is Quinlan munkáiban található meg (Quinlan, 1993). A döntési fa tanulásnak egy, az előzőktől független hagyománya alakult ki a statisztikusok körében. A *Classification and Regression Trees* (Breiman és társai, 1984), amelyet röviden „CART könyv"-nek neveznek, az alapvető irodalom e tekintetben.

A tanulás számos másik algoritmikus megközelítését is vizsgálták. A **pillanatnyilag-legjobb-hipotézis** (**current-best-hypothesis**) megközelítés egyetlen hipotézissel foglalkozik, specializálva, ha túl tágnak bizonyul, általánosítva, ha túl szűk. Ez a filozófiában már régóta ismert elv (Mill, 1843). A kognitív pszichológia korai munkájában szintén ezt javasolták, mint az emberi fogalom tanulás természetes formáját (Bruner és társai, 1957). Az MI-területen ez a megközelítés elsősorban Patrick Winston munkájához köthető, aki PhD-disszertációjában (Winston, 1970) a komplex objektumok leírásának tanulásával foglalkozott. A **verziótér**

Created by XMLmind XSL-FO Converter.

(**version space**) módszer (Mitchell, 1977; 1982) eltérő megközelítésen alapul, az összes konzisztens hipotézissel egyszerre foglalkozik, elhagyva azokat, amelyek az új példákkal inkonzisztenciát mutatnak. Ezt a megközelítést a META-DENDRAL kémiai szakértő rendszerben használták (Buchanan és Mitchell, 1978), majd később Mitchell LEX rendszerében (Mitchell, 1983), ami matematikai analízis problémák tanulással történő megoldását célozta. A harmadik nagyhatású irányzat kialakítása Michalski és társai munkájához köthető, akik az AQ algoritmusok sorozatával foglalkoztak, amelyek logikai szabályok halmazát tanulták (Michalski, 1969; Michalski és társai, 1986b).

Az együttes tanulás a tanuló algoritmusok teljesítménynövelésének egyre népszerűbb módszere. A **zsákolás** (**bagging**)[7] (Breiman, 1966) az első hatékony módszer, amely többféle **indító** (**bootstrap**) adathalmaz alapján megtanult hipotézisek kombinációját hozza létre. Ezeket az adathalmazokat az eredeti alulmintavételezésével állítják elő. A fejezetben ismertetett **turbózás** (**boosting**) módszere eredetileg Schapire elméleti munkájából (Schapire, 1990) származik. Az AdaBoost algoritmust Freund és Schapire dolgozta ki (Freund és Schapire, 1996), és Schapire adta meg elméleti analízisét (Schapire, 1999). Friedman és társai statisztikai alapon tárgyalják a turbózás módszerét (Friedman és társai, 2000).

A tanulási algoritmusok elméleti vizsgálata Gold munkájával kezdődött (Gold, 1967) az **identifikáció határátmenet**ben (**identification in the limit**) probléma tanulmányozásával. Ez a megközelítés ugyan részben tudományfilozófiai felfedezésből eredt (Popper, 1962), de alkalmazásának legfőbb területe nyelvtanok példamondatok alapján történő tanulása volt (Osherson és társai, 1986).

Míg az identifikáció határátmenetben megközelítés a végső konvergenciára koncentrál, addig a **Kolmogorov-komplexitás** (**Kolmogorov complexity**) vagy más néven **algoritmikus komplexitás** (**algorithmic complexity**), amelyet egymástól függetlenül Solomonoff (1964) és Kolmogorov (1965) fejlesztett ki, formális definíciót próbál adni az Ockham borotvája elvben használt egyszerűségnek. Hogy valahogy kimeneküljenek abból a csapdából, hogy az egyszerűség függ az információ reprezentációjának módjától, azt javasolták, hogy az egyszerűséget annak a legrövidebb programnak a hosszával mérjék, amely egy általános Turing-gépen helyesen adja vissza a megfigyelt adatokat. Bár sok különböző általános Turing-gépet lehet létrehozni, és így több különböző „legrövidebb" program lehetséges, ezek hossza legfeljebb csak egy konstansban tér el, amely független az adatok mennyiségétől. Ezt a gyönyörű eredményt, amely betekintést nyújt abba, hogy *bármely* előzetes reprezentációs eltérést végül maguk az adatok győznek le, csupán az teszi tönkre, hogy a legrövidebb program hosszának kiszámítása eldönthetetlen problémára vezet. Közelítő mértékeket lehet alkalmazni, például a **legrövidebb leíró hossz**t (**minimum description length**) vagy LLH-t (Rissanen, 1984), amivel kimagasló gyakorlati eredményeket értek el. Li és Vitányi írása a Kolmogorov komplexitás legjobb összefoglalása (Li és Vitányi, 1993).

A tanulás számítási elméletét – azaz a VKH-tanulás elméletét – Lesli Valiant vezette be (Valiant, 1984). Valiant munkája a számítási komplexitásra és a mintakomplexitásra helyezte a fő hangsúlyt. Michael Kearnsszel együtt Valiant megmutatta, hogy számos olyan fogalomosztály van, amely VKH-tanulással nem kezelhető probléma, még akkor sem, ha a példák elegendő információt nyújtanak. Bizonyos problémaosztályokban, például döntési listáknál, elértek néhány pozitív eredményt (Rivest, 1987).

A mintakomplexitás vizsgálatának ettől független tradícióját találhatjuk a statisztikusok körében, amely az **egyenletes konvergencia elmélet**ének (**uniform convergence theory**) vizsgálatával kezdődött (Vapnik és Cservonenkis, 1971). Az úgynevezett **VC-dimenzió** nagyjából a VKH tanulásból származó mértékhez hasonló – bár annál általánosabb – mértéket ad. A VC-dimenzió, ellentétben a VKH-analízissel, folytonos függvényosztályokra is alkalmazható. A VKH-analízis és a VC elmélet között először a „négy német", Blumer, Ehrenfeucht, Haussler és Warmuth (1989) teremtett kapcsolatot (akik közül valójában egyik sem német). A VC elmélet további fejlődése vezetett a **szupport vektorgép** vagy **SVM**[8] (**support vector machine**) kidolgozásához (Boser és társai, 1992; Vapnik, 1998). A szupport vektorgéppel a 20. fejezetben foglalkozunk.

Nagyon sok, a gépi tanulással foglalkozó fontos publikációt gyűjtöttek egybe a *Readings in Machine Learning* c. kötetben (Shavlik és Dietterich, 1990). A kétkötetes *Machine Learning 1* (Michalski és társai, 1983), illetve *Machine Learning 2* (Michalski és társai, 1986a) is sok fontos közlést tartalmaz, továbbá hatalmas bibliográfiát. Weiss és Kulikowski a függvény tanulás áttekinthető bevezetését adja gépi tanulási, statisztikai és neurális alapokon (Weiss és Kulikowski, 1991). A STATLOG projektben (Michie és társai, 1994) végezték a tanuló algoritmusok teljesítményének messze legkimerítőbb összehasonlítását. A jelenleg is folyó, a gépi tanulással

[7] A **bagging** név eredetileg a „bootstrap aggregating" rövidítéseként jelent meg, de az angol „bag" szó jelentését nagyrészt átvette, ezért fordítottuk „zsákolásnak". (*A ford.*)
[8] Magyarul is az SVM rövidítés terjedt el. (*A ford.*)

Created by XMLmind XSL-FO Converter.

foglalkozó fontos kutatási eredmények az International Conference on Machine Learning éves kiadványaiban, a Neural Information Processing Systems konferencián, a *Machine Learning* és a *Journal of Machine Learning Research* folyóiratokban, továbbá a fontosabb MI-újságokban jelennek meg. A tanulás számítási elméletének eredményei megjelennek az éves ACM Workshop on Computational Learning Theory (COLT) kiadványokban, valamint Kearns és Vazirani (1994), továbbá Anthony és Bartlett (1999) publikációiban.

6.2. Feladatok

18.1.

Vizsgálja meg a beszélni, illetve egy nyelvet megérteni tanuló gyerek problémáját! Magyarázza meg, hogy ez a folyamat hogyan illeszkedik az általános tanulási modellhez, és megfelelően azonosítsa a modell egyes komponenseit!

18.2.

Ismételje meg a 18.1. feladatot a tenisz (vagy valamilyen ön által ismert sport) tanulásának esetére! Ez felügyelt vagy megerősítéses tanulás?

18.3.

Rajzoljon fel egy döntési fát arra a problémára, hogy az útkereszteződésben elinduljunk-e előre, ha a lámpa éppen most váltott zöldre!

18.4.

Soha nem teszteljük kétszer ugyanazt az attribútumot a döntési fa egy útvonala mentén. Miért?

18.5.

Tegyük fel, hogy egy döntési fa segítségével generálunk egy tanító mintahalmazt, majd döntési fa tanulást alkalmazunk erre a halmazra. A tanulási algoritmus végül is a helyes döntési fát fogja visszaadni, ha a tanító halmaz mérete a végtelenhez tart? Miért, vagy miért nem?

18.6.

Egy jó „butácska” tanulási algoritmus a következő: készítsünk egy táblázatot az összes tanítópéldából. Vizsgáljuk meg, hogy milyen kimenet szerepel a legtöbbször a tanítópéldákban: jelöljük ezt *d*-vel. Ezek után, ha olyan minta jelenik meg a bemeneten, ami nincs a táblázatban, akkor adjunk vissza *d*-t. Olyan bemenetekre, amelyek megtalálhatók a táblában, adjuk vissza a táblázatban a bemenethez asszociált kimeneti értéket (vagy ha egynél több kimeneti érték van azonos bemenethez, akkor a leggyakrabban előfordulót). Implementálja ezt az algoritmust, és vizsgálja meg egy példának választott területen (például az étterem problémán) azt, hogy mennyire működik jól! Ennek alapján fogalmat alkothatunk a tanulás területének alapszintjéről – a minimális elvárható teljesítményről, amelyet minden tanuló algoritmusnak el kell érnie.

18.7.

Tegyük fel, hogy egy új algoritmussal tanulási kísérletet folytat. Egy adathalmazt használ, amiben 25-25 példa van az előforduló két osztályra. Azt tervezi, hogy a hagyj-ki-egyet keresztvalidációs módszert fogja használni. Összehasonlítási alapként egy egyszerű többségi szavazót használ. (A többségi szavazónak megadva egy tanító halmazt, a bemenetektől függetlenül mindig azt a kimenetet adja, amely kimeneti érték a halmazban legtöbbször előfordul.) Azt várta, hogy a többségi szavazó, a hagyj-ki-egyet keresztvalidációban nagyjából 50%-ot fog elérni, de meglepetésére 0% az eredmény. Meg tudja magyarázni, hogy miért?

18.8.

A döntési fák rekurzív előállítása során néha előfordul, hogy pozitív és negatív példák vegyes halmaza marad egy levélcsomópontban, még akkor is, ha az összes attribútumot használtuk már. Tegyük fel, hogy *p* pozitív és *n* negatív példánk van.

a. Mutassa meg, hogy a `DÖNTÉSI-FA-TANULÁS` algoritmusban használt megoldás, amely a többségi osztályozást választja, minimalizálja a példahalmazra vonatkozó abszolút hibát a levélnél!

Created by XMLmind XSL-FO Converter.

b. Mutassa meg, hogy a $p/(p + n)$ **osztály-valószínűség** (**class probability**) eredményként való visszaadása minimalizálja négyzetes hiba összegét!

18.9.

Tegyük fel, hogy egy tanulási algoritmus konzisztens hipotézist keres, és az osztályozandó példákat véletlen módon generáljuk. A példákat egyenletes eloszlással választjuk az n Boole-attribútum alapján kialakított $2n$ példa lehetőségből. Számítsa ki, hány példa kell ahhoz, hogy egy ellentmondás felmerülésének a valószínűsége 0,5 legyen!

18.10.

Tegyük fel, hogy egy attribútum a példák E halmazát E_i részhalmazokra bontja, és mindegyik részhalmazban p_i pozitív és n_i negatív példa van. Mutassa meg, hogy ha a $p_i/(p_i + n_i)$ arány nem egyforma minden i-re, akkor az attribútum információnyeresége biztosan pozitív!

18.11.

Módosítsa a DÖNTÉSI-FA-TANULÁS algoritmust úgy, hogy tartalmazza a χ^2 metszést! A részleteket megtalálhatja Quinlan írásában (Quinlan, 1986).

18.12.

A fejezetben leírt standard DÖNTÉSI-FA-TANULÁS algoritmus nem képes azon esetek kezelésére, amelyekben néhány példa esetén hiányoznak attribútumértékek.

a. Először is valamilyen módszerre van szükségünk, amellyel egy adott döntési fa alapján be tudunk sorolni példákat akkor is, ha a fában olyan attribútumokra vonatkozó tesztek szerepelnek, amelyek értéke hiányozhat egyes példáknál. Tegyük fel, hogy az X példa A attribútumra vonatkozó értéke hiányzik, és a döntési fa teszteli A-t egy olyan csomópontban, amit X elér. Ennek az esetnek egyik kezelési lehetősége, hogy úgy teszünk, mintha a példa az összes lehetséges attribútumértékkel rendelkezne, de súlyozzuk az egyes értékeket azzal a gyakorisággal, amelyet az összes olyan példán mérünk, amely eléri a döntési fa ezen csomópontját. A besorolási algoritmusnak minden olyan csomópont összes ágát követnie kell, amelyben egy érték hiányzik, és mindegyik út mentén össze kell szoroznia a súlyokat. Írja meg azt a módosított döntési fa algoritmust, amely rendelkezik ezzel a tulajdonsággal!

b. Módosítsa az információnyereség számítást a következő módon: a fa építésének fázisában egy adott csomópontba jutó tetszőleges C tanító példahalmaznál azok a példák, amelyeknek hiányzik bármely hátralevő (még nem vizsgált) attribútumhoz tartozó értéke, a C halmazban ezen attribútumra előforduló értékgyakoriság szerint kapjanak „mint-ha” értéket!

18.13.

A 18. fejezetben megmutattuk, hogy azok az attribútumok, amelyeknek nagyon sokféle lehetséges értékük van, problémát jelenthetnek az információnyereség számításánál. Ezek az attribútumok jó eséllyel nagyon sok kis osztályra, akár egyelemű osztályokra bontják a halmazt, ezért úgy tűnik a nyereség számításánál, hogy nagyon fontosak az adott csomópontban. A **nyereségarány** (**gain ratio**) kritérium az attribútumokat az általuk hozott információnyereség és a saját belső információtartalmuk arányával méri. A belső információtartalom arra a kérdésre adott válasz, hogy „Mi az értéke ennek az attribútumnak?”. A nyereségarány kritérium ennek megfelelően azt próbálja mérni, hogy mennyire hatékony információt biztosít a példa helyes osztályozásához ez az attribútum. Alkossa meg az attribútum információtartalmának matematikai kifejezését, és módosítsa a DÖNTÉSI-FA-TANULÁS algoritmust nyereségarány-alapúra!

18.14.

Egy együttes tanulási algoritmust vizsgál, amely M megtanult hipotézis eredményének egyszerű többségi szavazását használja. Tegyük fel, hogy az összes hipotézisnek a hibája, és az egyes hipotézisek által elkövetett hibák függetlenek a többiek hibájától. Adja meg M és ε függvényében azt az összefüggést, amelylyel az együttes tanulással előállított eszköz hibája számítható, és számítsa ki $M = 5$, 10 és 20, illetve $\varepsilon = 0{,}1$, 0,2 és 0,4 esetén! Ha feladjuk a függetlenségre vonatkozó feltételezést, akkor előfordulhat-e, hogy az együttes hibája *rosszabb* lesz, mint ε?

Created by XMLmind XSL-FO Converter.

18.15.

Ez a feladat a döntési listák kifejező erejével foglalkozik (lásd 18.5. alfejezet).

a. Mutassa meg, hogy egy döntési lista képes bármely Boole-függvény reprezentálására, ha nem korlátozzuk a tesztek méretét!

b. Mutassa meg, hogy ha egy teszt legfeljebb *k* literálist tartalmazhat, akkor a döntési lista minden olyan függvény reprezentálására képes, amely egy *k* mélységű döntési fával reprezentálható!

Created by XMLmind XSL-FO Converter.

19. fejezet - A tudás szerepe a tanulásban

Ebben a fejezetben a tanulással foglalkozunk, de csak akkor, amikor már tudunk is valamit.

Az előbbi három fejezetben bemutatott tanulási módszerek mindegyikében az volt az alapötlet, hogy egy olyan függvényt konstruálunk, amelynek bemeneti/kimeneti viselkedése megegyezik azzal, amit az adatoknál megfigyeltünk. Bármilyen esetet is nézünk, a tanulási módszer tulajdonképpen nem más, mint egy megfelelő függvénynek a hipotézistérben történő megkeresése, a függvény formájára vonatkozó igen elemi feltételezésekből kiindulva. Ilyen feltételezés lehet például, hogy a függvény „egy másodrendű polinom" vagy „egy döntési fa", vagy az az elfogultság, hogy „az egyszerűbb a jobb". Ennek megvalósítása felért azzal, mintha azt mondanánk, hogy mielőtt valami újat tanulnánk, el kellene felejteni (majdnem) mindent, amit tudunk. Ebben a fejezetben olyan tanulási módszereket tanulmányozunk, amelyek **a priori tudás**ból (**prior knowledge**) merítenek előnyöket. Az esetek többségében az a priori tudást elsőrendű elméletek segítségével reprezentáljuk. Így most először fogjuk összekapcsolni a tudásreprezentációra és a tanulásra vonatkozó eredményeket.

1. A tanulás logikai megfogalmazása

A 18. fejezetben a tisztán induktív tanulást olyan folyamatnak tekintettük, amely egy, az adatokkal konzisztens hipotézist keresett. Most ezt a definíciót olyan esetre pontosítjuk, amikor a hipotézist logikai állítások halmazával reprezentáljuk. A példaleírások és besorolások szintén logikai állítások, egy új példát pedig úgy tudunk osztályozni, hogy egy osztályozó állítást a hipotézisből és a példa leírásából kikövetkeztetünk. Ezzel a megközelítéssel a hipotéziseket inkrementálisan, állításonként építhetjük. A megközelítés lehetővé teszi az a priori tudás használatát is, mivel a már megismert állításokat bevethetjük új példák osztályozásába. A tanulás logikai megfogalmazásában először látszólag sok a munka, ez azonban a tanulás számos aspektusát segíti tisztázni. Ez lehetővé teszi, hogy a 18. fejezetben megismert egyszerű tanulási módszereket jócskán túlszárnyaljuk azáltal, hogy a logikai következtetés teljes erejét tanulási célokra használjuk.

1.1. Példák és hipotézisek

Emlékezzünk a 18. fejezetben a tanulási problémára bemutatott étterem példára: egy olyan döntési szabály megtanulására, hogy vajon érdemes-e egy asztalra várni. A példákat olyan **attribútum**okkal (**attribute**s) írtuk le, mint az *Alternatíva*, *Bár*, *Péntek/Szombat* stb. Logikai megközelítésben egy példa egy logikai állítással leírt objektum; az attribútumok pedig unáris predikátumok. Nevezzük általánosságban az i-edik példát X_i-nek. A 18.3. ábra első példáját az alábbi állítások írják le:

$$Alternatíva(X_1) \wedge \neg Bár(X_1) \wedge \neg PéntekSzombat(X_1) \wedge Éhes(X_1) \wedge \ldots$$

A $D_i(X_i)$ jelöléssel az X_i leírására fogunk hivatkozni, ahol D_i tetszőleges, egyargumentumú logikai kifejezés lehet. Az objektumok osztályozását a

$$VárjunkE(X_1)$$

állítás végzi. Pozitív példák esetén a $Q(X_i)$, negatív példák esetén a $\neg Q(X_i)$ általános jelölést fogjuk használni. A teljes tanító halmaz az összes leíró és besoroló állítás konjunkciója.

Logikai megközelítésben az induktív tanulás célja a Q célpredikátummal ekvivalens logikai kifejezés megtalálása, amivel korrektül tudjuk a példákat osztályozni. Mindegyik hipotézis egy-egy javaslat a keresett logikai kifejezésre, amit mi a célpredikátum **definíciójelölt**jének (**candidate definition**) nevezünk. Jelöljük C_i-vel a definíciójelöltet, ekkor mindegyik H_i hipotézis egy $\forall x\ Q(x) \Leftrightarrow C_i(x)$ formájú kifejezés. Például a döntési fa azt fejezi ki, hogy a célpredikátum egy objektum esetében csak akkor igaz, ha az *igaz*-hoz vezető egyik ága teljesül. Így a 18.6. ábrán látható döntési fa a következő logikai definíciót fejezi ki (amelyre a jövőben H_r-rel fogunk hivatkozni):

$$\forall r\ VárjunkE(r) \Leftrightarrow Vendégek(r, Néhány)$$

$$\vee\ Vendégek(r, Tele) \wedge Éhes(r) \wedge Konyha(r, Francia)$$

Created by XMLmind XSL-FO Converter.

$$\vee \textit{Vendégek}(r, \textit{Tele}) \wedge \textit{Éhes}(r) \wedge \textit{Konyha}(r, \textit{Thai})$$

$$\wedge \textit{Péntek/Szombat}(r)$$

$$\vee \textit{Vendégek}(r, \textit{Tele}) \wedge \textit{Éhes}(r) \wedge \textit{Konyha}(r, \textit{Burger}) \quad (19.1)$$

Mindegyik hipotézis azt jósolja meg, hogy példák egy bizonyos halmaza, nevezetesen azok, amelyek kielégítik a hipotézisdefiníció jelöltjét, a célpredikátumnak megfelelő példák lesznek. Ezt a halmazt a predikátum **kiterjedés**ének (**extension**) nevezzük. Két különböző kiterjedéssel rendelkező hipotézis így logikailag ellentmondásban van egymással, hiszen legalább egy példa esetén eltérő eredményt jósolnak. Ha viszont azonos a kiterjedésük, akkor logikailag ekvivalensek.

A **H** hipotézistér ezek után az összes olyan hipotézis $\{H_1, \ldots, H_n\}$ halmaza, amelyek kezelésére a tanuló algoritmust tervezték. Például `DÖNTÉSI-FA-TANULÁS` algoritmus az adott attribútumokkal kialakítható összes döntési fa hipotézissel képes foglalkozni; így hipotézistere az összes döntési fát tartalmazza. Feltehetően a tanuló algoritmus hisz abban, hogy az egyik hipotézis korrekt, azaz hisz a

$$H_1 \vee H_2 \vee H_3 \vee \ldots \vee H_n \quad (19.2)$$

logikai kifejezésben. Ahogy a példák érkeznek, a példákkal nem **konzisztens** (**consistent**) hipotéziseket ki lehet zárni. Vizsgáljuk meg alaposabban a konzisztencia ezen fogalmát. Nyilvánvaló, hogy ha a H_i hipotézis a teljes tanító halmazzal konzisztens, akkor mindegyik példával konzisztensnek kell lennie. Mit jelent viszont az, hogy valamelyik példával ellentmondásban van? Ez kétféle módon történhet meg:

- A hipotézis szempontjából egy példa **hamis negatív** (**false negative**), ha a valóságban pozitív, de a hipotézis szerint negatív. Például a következő leírással rendelkező X_{13} új példa:

$$\textit{Vendégek}(X_{13}, \textit{Tele}) \wedge \textit{BecsültVárakozásE}(X_{13}, \textit{0-10}) \wedge \neg \textit{Éhes}(X_{13}) \wedge \ldots \wedge \wedge \textit{VárjunkE}(X_{13})$$

a korábban definiált H_r hipotézis számára hamis negatív példa lesz. H_r-ből, illetve a példa leírásából kikövetkeztethető mind a $\textit{VárjunkE}(X_{13})$ – ezt állítja a példa, mind a $\neg\textit{VárjunkE}(X_{13})$ – ezt állítja a hipotézis. Tehát a hipotézis és a példa logikailag ellentmondásban van.

- A hipotézis szempontjából egy példa **hamis pozitív** (**false positive**), ha a valóságban negatív, de a hipotézis szerint pozitív.[1]

Ha egy példa hamis negatív vagy hamis pozitív egy hipotézis szempontjából, akkor a példa és a hipotézis logikailag ellentmondásban van egymással. Feltéve, hogy a példa a tények korrekt megfigyelésén alapul, a hipotézist kizárhatjuk. Logikailag ez a következtetés a rezolúciós szabállyal pontos analógiában van (lásd 9. fejezet). Hipotézisek diszjunkciója felel meg egy klóznak, a példa pedig megfelel annak a literálisnak, amely a klózban levő egyik literálissal rezolvál. Egy szokásos logikai következtető rendszer tehát elméletileg tud példákból tanulni oly módon, hogy egy vagy több hipotézist kizár. Tegyük fel például, hogy a példát az I_1 állítással jelöljük, és a hipotézistér $H_1 \vee H_2 \vee H_3 \vee H_4$. Ha I_1 ellentmondásban van H_2-vel és H_3-mal, akkor a logikai következtető rendszer származtatni tudja az új $H_1 \vee H_4$ hipotézisteret.

Tehát az induktív tanulást logikai leírással olyan folyamatként jellemezhetjük, mint amelyik fokozatosan kizárja a példáknak ellentmondó hipotéziseket, leszűkítve a lehetőségek körét. Mivel a hipotézistér rendszerint nagy (vagy elsőrendű logika esetén egyenesen végtelen), nem javasoljuk egy rezolúcióalapú tételbizonyító rendszer építését és a hipotézistér teljes számbavételét. Ehelyett két megközelítést tárgyalunk, amelyek sokkal kisebb erőfeszítéssel logikailag ellentmondásmentes hipotéziseket képesek találni.

1.2. A pillanatnyilag legjobb hipotézis keresése

A **pillanatnyilag legjobb hipotézis** (**current-best-hypothesis**) keresési eljárás alapgondolata az, hogy egyetlen hipotézist vegyünk figyelembe, és ha új példa érkezik, akkor ennek figyelembevételével alakítsuk át a hipotézist annak érdekében, hogy az ellentmondás-mentességet fenntartsuk. Az algoritmus alapját John Stuart Mill írta le (Mill, 1843), de könnyen lehet, hogy már korábban megalkották.

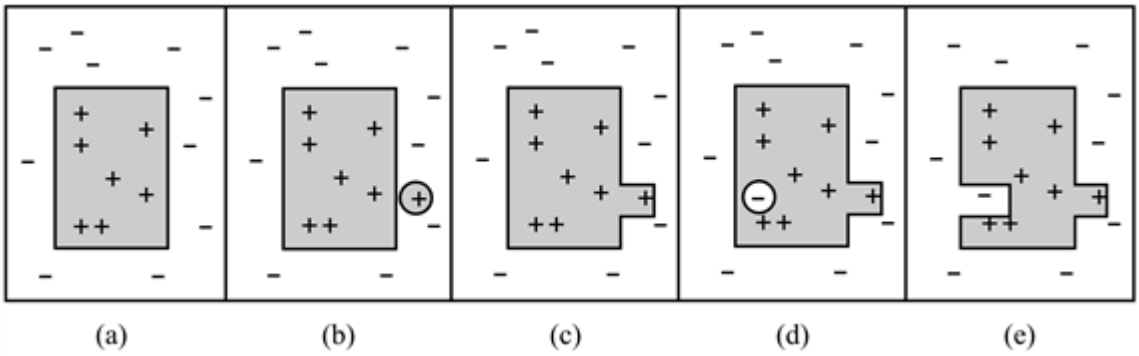
Tegyük fel, hogy van valamilyen hipotézisünk, például H_r, amit már meglehetősen megkedveltünk. Amíg egy új példával sincs ellentmondásban, semmit sem kell tennünk. Aztán egyszer csak egy hamis negatív példa (X_{13})

[1] „Hamis pozitív” és „hamis negatív” kifejezéseket először a gyógyításban használták arra, hogy a téves laboratóriumi eredményeket jellemezzék. Egy eredmény akkor hamis pozitív, ha azt jelzi, hogy a páciens az adott betegségben szenved, míg valójában nem ez a helyzet.

Created by XMLmind XSL-FO Converter.

érkezik. Mit csináljunk? A 19.1. (a) ábra sematikusan úgy mutatja be H_r-t, mint egy területet, a négyszögön belül minden H_r kiterjedésébe tartozik. Az eddig látott példákat „+” vagy „–” jelöli, és látható, hogy H_r jól sorolja be az összes példát, mint *VárjunkE* pozitív vagy negatív példáit. A 19.1. (b) ábrán egy új – hamis negatív – példa jelenik meg (bekarikázva): a hipotézis azt állítja, hogy negatív, pedig valójában pozitív példa. A hipotézis kiterjedését növelni kell ahhoz, hogy tartalmazza ezt a példát is. Ezt a lépést **általánosítás**nak (**generalization**) nevezzük: egy lehetséges általánosítás a 19.1. (c) ábrán látható. Ezután a 19.1. (d) ábrán egy hamis pozitív példát látunk: a hipotézis azt állítja, hogy pozitív, pedig valójában negatív. A hipotézis kiterjedését csökkenteni kell ahhoz, hogy kizárjuk ezt a példát. Ezt **szűkítés**nek (**specialization**) nevezzük; a 19.1. (e) ábrán a hipotézis egy lehetséges szűkítését látjuk.

19.1. ábra - (a) Egy konzisztens hipotézis. (b) Egy hamis negatív példa. (c) A hipotézist általánosítottuk. (d) Egy hamis pozitív példa. (e) A hipotézist szűkítettük.



Most már specifikálni tudjuk a 19.2. ábrán látható `PILLANATNYILAG-LEGJOBB-TANULÁS` algoritmusát. Vegyük észre, hogy valahányszor általánosítjuk vagy szűkítjük a hipotézist, ellenőriznünk kell az összes többi példára, hiszen nincs garancia arra, hogy a kiterjedés önkényes növelése/csökkentése elkerüli bármelyik másik negatív/pozitív példa befoglalását/kirekesztését.

19.2. ábra - A pillanatnyilag-legjobb-hipotézis tanuló algoritmus. Konzisztens hipotézist keres és viszszalép, ha nem található konzisztens szűkítés/általánosítás.

```
function PILLANATNYILAG-LEGJOBB-TANULÁS(példák) returns egy hipotézis

  H ← egy olyan hipotézis, amelyik a példák halmaz első példájával konzisztens
  for each a példák további példáira do
    if a H alapján e hamis pozitív then
        H ← choose H olyan szűkítését, amely konzisztens a példák példáival
    else if a H alapján e hamis negatív then
        H ← choose H olyan általánosítását, amely konzisztens a példák példáival
    if ha nem található konzisztens szűkítés/általánosítás then fail
  return H
```

Az általánosítást és a szűkítést úgy definiáltuk, mint olyan műveleteket, amelyek a hipotézis *kiterjedését* változtatják meg. Most pontosan meg kell határoznunk azt, hogy ezek a műveletek hogyan implementálhatók olyan szintaktikus műveletekként, amelyek a hipotézishez rendelt definíciójelöltet úgy változtatják meg, hogy azt egy program végre tudja hajtani. Ehhez először is vegyük észre, hogy az általánosítás és a szűkítés egyben hipotézisek közötti *logikai* relációk. Ha a C_1 definíciónak megfelelő H_1 hipotézis a C_2 definíciónak megfelelő H_2 hipotézis általánosítása, akkor fenn kell állnia annak, hogy:

$$\forall x \;\; C_2(x) \;\; \Rightarrow C_1(x)$$

Tehát ahhoz, hogy a H_2 hipotézis általánosítását létrehozzuk, egyszerűen egy olyan C_1 definíciót kell találnunk, amelyik logikailag következik C_2-ből. Ezt könnyen megtehetjük. Ha például $C_2(x)$ *Alternatíva*$(x) \wedge$ *Vendégek*$(x,$ *Néhány*$)$, akkor egy általánosítási lehetőség a $C_1(x) \equiv$ *Vendégek*$(x,$ *Néhány*$)$. Ezt **feltételek törlésé**nek (**dropping conditions**) nevezzük. Felfoghatjuk úgy, hogy egy gyengébb definíció jön létre, így a pozitív példák nagyobb halmazát teszi lehetővé. Számos egyéb általánosítási eljárás is létezik, azon nyelvtől függően, amelyen a

Created by XMLmind XSL-FO Converter.

műveleteket végezzük. Hasonlóképpen, egy hipotézist szűkíteni tudunk, ha további feltételeket adunk a hozzá tartozó definíciójelölthöz, illetve ha eltávolítunk tagokat egy diszjunktív definícióból. Lássuk, hogyan működik ez az éttermi feladatban, 18.3. ábra adatait használva:

- Az első példa – X_1 – pozitív. *Alternatíva*(X_1) igaz, tehát vegyük fel a következő kiinduló hipotézist:

H_1: $\forall x$ *VárjunkE*(x) $\Leftrightarrow$ *Alternatíva*(x)

- A második példa – X_2 – negatív. H_1 alapján pozitív lenne, így hamis pozitív. Ezért szűkítenünk kell H_1-et. Ezt megtehetjük például úgy, hogy egy olyan további feltételt adunk hozzá, amely feltétel kizárja X_2-t. Egy lehetőség:

H_2: $\forall x$ *VárjunkE*(x) $\Leftrightarrow$ *Alternatíva*(x) $\wedge$ *Vendégek*(x, *Néhány*)

- A harmadik példa – X_3 – pozitív. H_2 alapján negatív lenne, így hamis negatív. Tehát általánosítanunk kell H_2-t. Ezt elérhetjük, ha töröljük *Alternatíva* feltételt, amely a következő hipotézisre vezet:

H_3: $\forall x$ *VárjunkE*(x) $\Leftrightarrow$ *Vendégek*(x, *Néhány*)

- A negyedik példa – X_4 – pozitív. H_3 alapján negatív lenne, így hamis negatív. Így általánosítanunk kell H_3-at. Nem törölhetjük *Vendégek* feltételt, mert ez egy mindent tartalmazó hipotézishez vezetne, ami ellentmondásban van X_2-vel. Egyik lehetőség a következő diszjunkció bevezetése:

H_4: $\forall x$ *VárjunkE*(x) $\Leftrightarrow$ *Vendégek*(x, *Néhány*)

$\vee$ (*Vendégek*(x, *Tele*) $\wedge$ *Péntek/Szombat*(x))

A hipotézis ezek után már kezd ésszerűnek tűnni. Nyilvánvaló, hogy más lehetőségek is vannak, amelyek konzisztensek az első négy példával, kettő ezek közül:

$$H_4' : \forall x\ VárjunkE(x) \Leftrightarrow BecsültVárakozás(x, 30-60)$$
$$H_4'' : \forall x\ VárjunkE(x) \Leftrightarrow Vendégek(x, Néhány) \vee (Vendégek(x, Tele) \wedge BecsültVárakozás(x, 10-30))$$

A PILLANATNYILAG-LEGJOBB-TANULÁS algoritmusát nem determinisztikusan írjuk le, mivel bármely ponton számos olyan szűkítési vagy általánosítási lehetőség lehet, melyek bármelyikét alkalmazhatjuk. A már meghozott döntések nem szükségszerűen vezetnek a legegyszerűbb hipotézishez, sőt olyan megoldhatatlan szituációhoz is vezethetnek, amelyben nincs a hipotézisnek olyan egyszerű módosítása, amely minden adattal konzisztens hipotézist állít elő. Ilyen esetekben a programnak egy előző választási ponthoz kell visszalépnie.

A PILLANATNYILAG-LEGJOBB-TANULÁS algoritmusát – és különböző változatait – számos gépi tanulást felhasználó rendszerben alkalmazták, először Patrick Winston (Winston, 1970) „ív-tanuló” programjában. Azonban ha a tér nagy, és nagyszámú példával dolgozunk, akkor bizonyos nehézségek merülnek fel:

1. Az összes előző példa újraellenőrzése minden egyes módosításnál nagyon munkaigényes.

2. Nehéz jó keresési heurisztikát találni, és a visszalépések örökké tarthatnak. Mint a 18. fejezetben már láttuk, a hipotézistér kétszeresen exponenciálisan nagy tér lehet.

1.3. Legkisebb megkötés elvű keresés

A visszalépésre azért van szükség, mert a „pillanatnyilag-legjobb-hipotézis” eljárásban *választani* kell egy legjobb becslést, egy partikuláris hipotézist, akkor is, ha még nincs elég adatunk ahhoz, hogy biztosak legyünk a választásunk helyességében. Ehelyett azt tehetjük, hogy az összes olyan és csak olyan hipotézist megtartjuk, amelyek az eddigi adatainkkal konzisztensek. Ezek után mindegyik új eset vagy nincs hatással, vagy kizár néhányat a hipotéziseink közül. Emlékezzünk arra, hogy a hipotézistér egy diszjunktív állításnak tekinthető:

$H_1 \vee H_2 \vee H_3 \vee \ldots \vee H_n$

Ahogy egyre több hipotézis bizonyul a példákkal inkonzisztensnek, ez a diszjunktív állítás zsugorodik, csak azokat a hipotéziseket tartva meg, amelyeket nem szűrtünk ki. Feltéve, hogy az eredeti hipotézistér tartalmazza a helyes választ, a redukált diszjunktív állításnak is szükségszerűen tartalmaznia kell azt, hiszen csak inkorrekt

Created by XMLmind XSL-FO Converter.

hipotéziseket szűrtünk ki. A megmaradt hipotézisek halmazát nevezzük **verziótér**nek **(version space).** Az ilyen elven működő tanulási algoritmust (amelyet a 19.3. ábrán vázoltunk fel) verziótér tanulási algoritmusnak (más néven **jelölteltávolítás**i – **candidate elimination** – algoritmusnak) nevezzük.

19.3. ábra - Verziótér tanuló algoritmus. *V*-nek olyan részhalmazát tanulja meg, amelynek elemei a *példák*kal konzisztensek.

```
function VERZIÓ-TÉR-TANULÁS(példák) returns egy verziótér
    local variables: V, a verziótér: az összes hipotézis tere

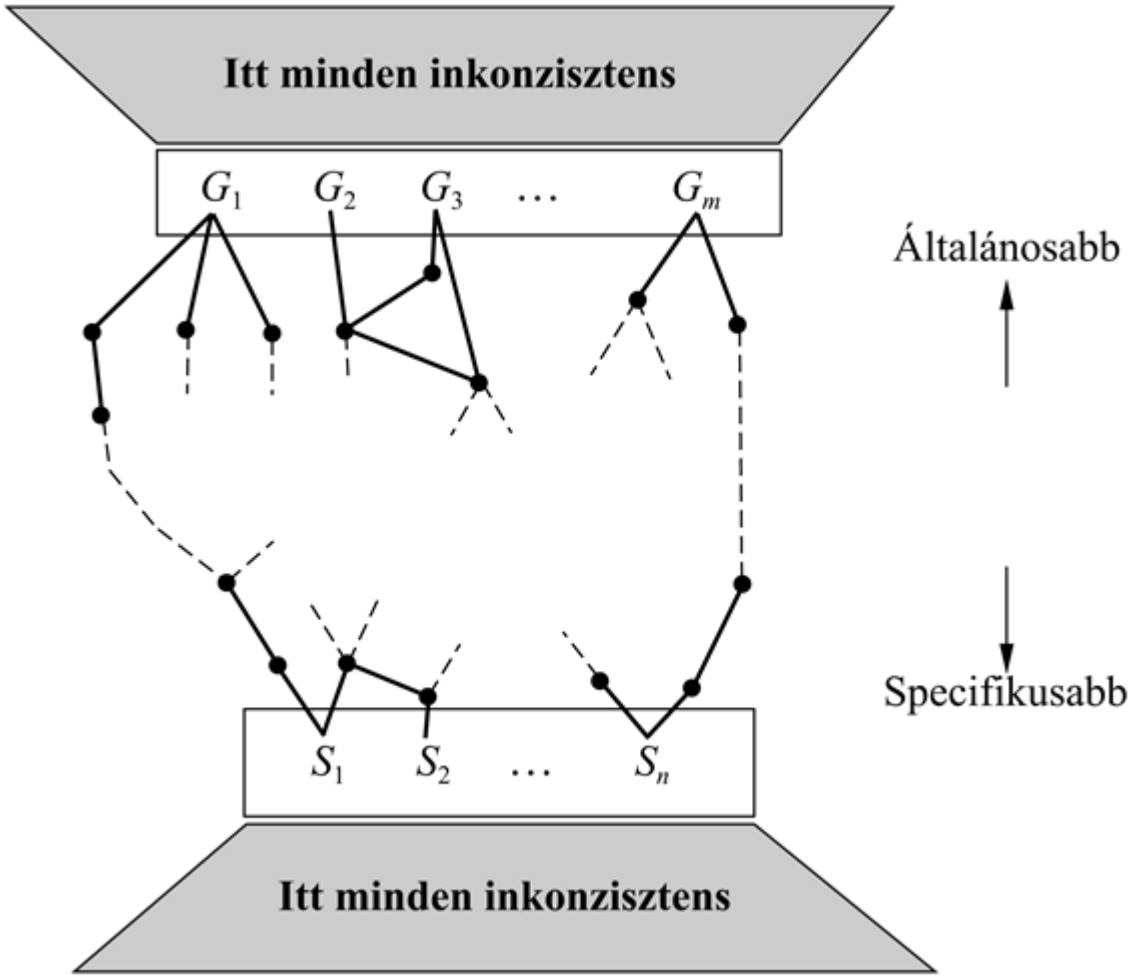
    V ← az összes hipotézis halmaza
    for each a példák összes e példájára do
        if V nem üres then VERZIÓ-TÉR-MÓDOSÍTÁS(V, e)
    return V

function VERZIÓ-TÉR-MÓDOSÍTÁS(V, e) returns egy módosított verziótér
    V ← {h ∈ V : h konzisztens e-vel}
```

Fontos tulajdonsága ennek a megközelítésnek, hogy *inkrementális:* soha nem kell visszalépnünk és újravizsgálnunk a régi példákat. Az összes megmaradt hipotézis garantáltan konzisztens az összes példával. Másrészt **legkisebb megkötés elv**ű (**least commitment**) algoritmusnak is nevezhetjük, ugyanis nem tesz önkényes választásokat (vö. a 11. fejezetben található részben rendezett tervkészítő algoritmussal). Van viszont egy nyilvánvaló probléma. Már leszögeztük, hogy a hipotézistér óriási, hogyan tudjuk akkor leírni ezt az óriási diszjunktív állítást?

19.4. ábra - A verziótér a példákkal konzisztens összes hipotézist tartalmazza

Created by XMLmind XSL-FO Converter.



A következő egyszerű analógia sokat segít a megértésben. Hogyan ábrázoljuk az összes 1 és 2 közé eső valós számot? Végül is végtelen sok ilyen van! A megoldás az intervallumábrázolás, amelyik csak a halmaz határait specifikálja: [1, 2]. Ez azért működik, mert a valós számok felett értelmezett egy *rendezés*.

A hipotézistér felett is rendelkezünk rendezéssel, nevezetesen az általánosítás/szűkítés alapján. Ez egy részleges rendezés, ami azt jelenti, hogy a határ nem egy pont, hanem a hipotézisek egy halmaza: a **határhalmaz** (**boundary set**). A nagyszerű a dologban az, hogy a teljes verziótér reprezentálható csupán két határhalmaz: egy legáltalánosabb határhalmaz (a **G halmaz**) és egy legszűkebb határhalmaz (az **S halmaz**) segítségével. *E kettő közé eső bármely hipotézis garantáltan konzisztens az összes példával*. Mielőtt ezt bizonyítjuk, röviden foglaljuk össze az eddigieket:

- A jelenlegi verziótér azon hipotézisek halmaza, amelyek az összes eddigi példával konzisztensek. Ezt az S halmaz és a G halmaz segítségével ábrázoljuk, mindkettő a hipotézisek valamilyen halmaza.
- Az S halmaz összes eleme konzisztens az összes eddigi megfigyeléssel, és nincs ennél szűkebb konzisztens hipotézis.
- A G halmaz összes eleme konzisztens az összes eddigi megfigyeléssel, és nincs ennél általánosabb konzisztens hipotézis.

Azt kívánjuk meg, hogy a kezdeti tér (mielőtt bármelyik példát megvizsgáltuk volna), az összes elképzelhető hipotézist tartalmazza. Ezt úgy biztosítjuk, hogy a G halmazt egyszerűen az *Igaz* értékre állítjuk (az a hipotézis, amely mindent tartalmaz), az S halmazt pedig egyszerűen a *Hamis* értékre állítjuk (az a hipotézis, amelynek kiterjedése üres).

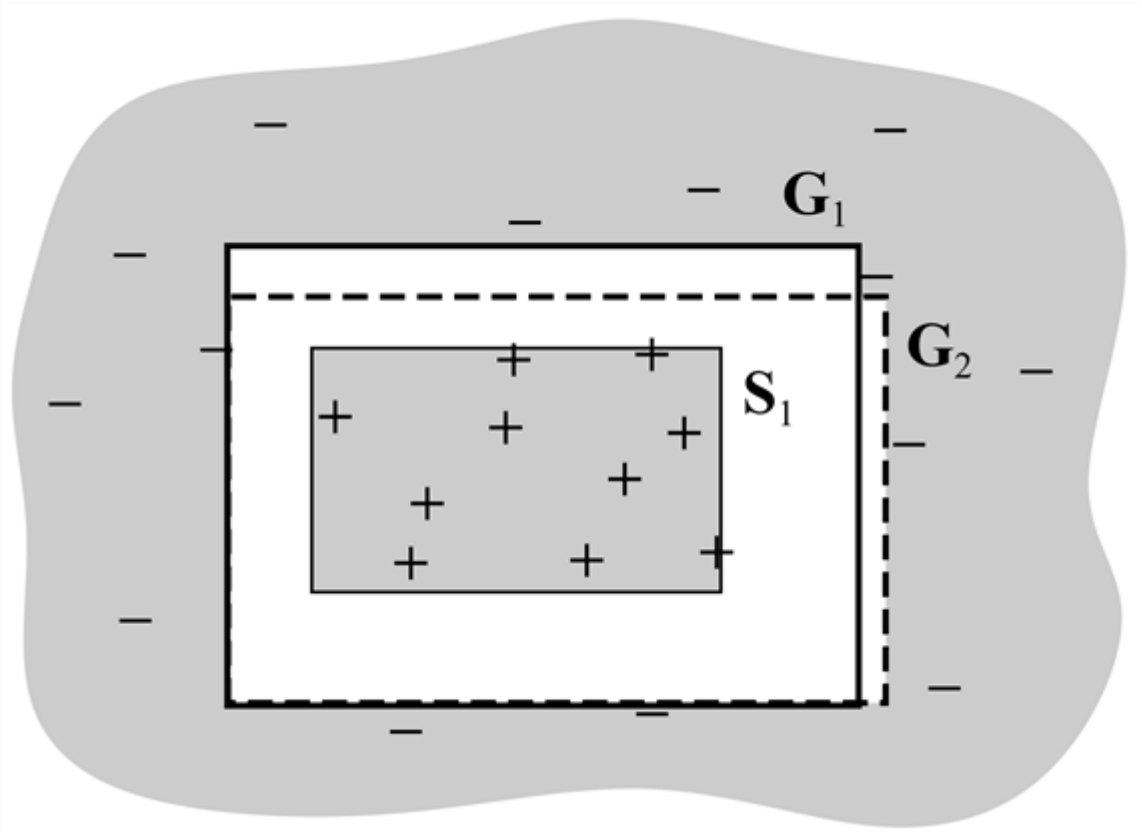
A 19.4. ábra a verziótér határhalmazokkal való ábrázolásának általános struktúráját mutatja. Ahhoz, hogy megmutassuk, hogy ez az ábrázolás elégséges, két tulajdonságra van szükségünk:

Created by XMLmind XSL-FO Converter.

1. Mindegyik konzisztens hipotézis (amelyik nem valamelyik határhalmaz eleme) szűkebb, mint a G halmaz valamelyik eleme, és általánosabb, mint az S halmaz valamelyik eleme. (Azaz nincsenek kívül „kóbor” elemek.) Ez közvetlenül következik az *S* és *G* definíciójából. Ha lenne egy kóbor *h*, akkor az vagy nem lehetne szűkebb *G* valamely eleménél, tehát *G*-hez kellene tartoznia, vagy nem lehetne általánosabb *S* valamely eleménél, ebben az esetben viszont *S*-hez tartozna.

2. Mindegyik hipotézis, amelyik szűkebb a G halmaz valamely eleménél, és általánosabb az S halmaz valamely eleménél, konzisztens hipotézis (azaz a határok között nincsenek „lyukak”). Bármely – az *S* és *G* közé eső – *h*-nak az összes olyan negatív példát vissza kell utasítania, amelyet *G* minden egyes tagja visszautasít (mivel szűkebb), ugyanakkor el kell fogadnia az összes pozitív példát, amelyet az *S* akármelyik tagja elfogad (mivel általánosabb). Így *h* az összes példát jól kezeli, tehát nem lehet inkonzisztens. A 19.5. ábra mutatja be a helyzetet: nincs ismert példa, amely *S*-en kívül, de *G*-n belül helyezkedik el, így a kettő közt elhelyezkedő összes hipotézisnek konzisztensnek kell lennie.

19.5. ábra - A *G* és az *S* elemeinek kiterjesztései. A kettő között nem található ismert példa.



Bemutattuk tehát, hogy *ha* *S*-t és *G*-t definíciójuknak megfelelően kezeljük, akkor a verziótér kielégítő leírását adják. Az egyetlen hátralévő probléma, hogy hogyan *módosítsuk* *S*-t és *G*-t egy új példa esetén (a `VERZIÓ-TÉR-MÓDOSÍTÁS` függvény feladata). Elsőre ez meglehetősen bonyolultnak tűnik, de a definíciók és a 19.4. ábra alapján nem túl nehéz az algoritmus előállítása.

Az S és G halmazok S_i és G_i elemeivel kell foglalkoznunk. Az új példa bármelyik esetén lehet hamis pozitív vagy hamis negatív.

1. S_i-re hamis pozitív. Azt jelenti, hogy S_i túl általános, de mivel S_i-nek (a definíciója értelmében) nincs konzisztens szűkítése, így S_i-t eltávolítjuk az S halmazból.

2. S_i-re hamis negatív. Azt jelenti, hogy S_i túl szűk, így S_i-t az összes közvetlen általánosításával helyettesítjük, feltéve, hogy ezek szűkebbek, mint *G* egyes elemei.

3. G_i-re hamis pozitív. Azt jelenti, hogy G_i túl általános, így az összes közvetlen szűkítésével helyettesítjük, feltéve, hogy ezek általánosabbak, mint *S* egyes elemei.

Created by XMLmind XSL-FO Converter.

4. G_i-re hamis negatív. Azt jelenti, hogy G_i túl szűk, de mivel nincs konzisztens általánosítása (definíciója értelmében), így G_i-t eltávolítjuk a G halmazból.

Ezeket a műveleteket minden új példára elvégezzük addig, amíg az alábbi három eset valamelyike fel nem lép:

1. Pontosan egy hipotézis marad a verziótérben, ebben az esetben ezt adjuk vissza mint az egyetlen hipotézist.

2. A verziótér *összeomlik* – vagy S, vagy G üressé válik, azt jelezve, hogy nincs a tanító halmazra nézve konzisztens hipotézis. Ez megegyezik az egyszerű döntési fa algoritmus sikertelen kimenetelével.

3. Kifogyunk a példákból, miközben számos hipotézis maradt a verziótérben. Ez azt jelenti, hogy a verziótér a hipotézisek diszjunkcióját reprezentálja. Ha egy új példa esetén a diszjunkció összes tagjában azonos eredményt kapunk, akkor ezt adhatjuk vissza mint a példa besorolását. Ha az eredmények nem egyeznek meg, akkor egyik lehetőségként alkalmazhatjuk a többségi szavazást.

Gyakorlás céljára hagyjuk a VERZIÓ-TÉR-TANULÁS algoritmusnak az éttermi adatokra való alkalmazását.

A verziótér megközelítésnek két alapvető hátránya van:

- Ha a problématerület adatai zajt tartalmaznak, vagy ha az attribútumok a pontos besoroláshoz nem elégségesek, akkor a verziótér mindig összeomlik.

- Ha korlátlan méretű diszjunkciót megengedünk a hipotézistérben, akkor az S halmaz mindig tartalmazni fog egy egyedi legszűkebb hipotézist, nevezetesen az összes eddig látott pozitív példa diszjunkcióját. Hasonlóképpen a G halmaz tartalmazni fogja a negatív példák diszjunkciójának negáltját.

- Egyes hipotézistereknél az S- és a G-beli elemek száma az attribútumok számában exponenciálisan növekedhet annak ellenére, hogy az ilyen hipotézisterekhez ismertek hatékony tanuló algoritmusok.

A zaj problémájára napjainkig nem ismert teljes, sikeres megoldás. A diszjunktív kapcsolatok problémája kezelhető oly módon, hogy csak korlátozott méretű diszjunktív formákat engedünk meg, vagy az általánosabb predikátumokra egy **általánosítási hierarchiát** (**generalization hierarchy**) vezetünk be. Például a *BecsültVárakozás*(*x*, *30-60*) ∨ *BecsültVárakozás*(*x*, >60) helyett használhatjuk a *HosszúVárakozás*(*x*) literált. Az általánosítási és a szűkítési műveletek egyszerűen kiterjeszthetők ennek kezelésére.

A verziótér algoritmus tiszta változatát először a Meta-DENDRAL rendszerben alkalmazták. Ezt a rendszert olyan szabályok megtanulására tervezték, amelyek azt írják le, hogy egy tömegspektrométerben a molekulák hogyan esnek szét darabokra (Buchanan és Mitchell, 1978). A Meta-DENDRAL képes volt olyan szabályok generálására, amelyek elég újak voltak egy analitikus kémiával foglalkozó újságban való publikálhatósághoz. Ezek voltak az első – egy számítógépes program által felfedezett – valós tudományos eredmények. Az algoritmust az elegáns LEX rendszerben (Mitchell és társai, 1983) szintén használták, ez a rendszer saját sikereinek és kudarcainak tanulmányozása alapján szimbolikus integrálási problémák megoldásának megtanulására volt képes. Bár a verziótér-módszerek valószínűleg a legtöbb valós probléma tanulása esetén – főleg a zaj miatt – nem praktikusak, jó betekintést adnak a hipotézistér logikai szerkezetébe.

2. A tudás szerepe a tanulásban

Az előbbi részben az induktív tanulás egyszerű sémájával foglalkoztunk. Ahhoz, hogy a háttértudás szerepét megérthessük, beszélnünk kell a hipotézisek, a példaleírások és a besorolások közötti kapcsolatról. Jelölje a *Leírások* az összes példaleírás konjunkcióját és a *Besorolások* a példabesorolások konjunkcióját. A „megfigyelést megmagyarázó” *Hipotézis*-nek akkor a következő tulajdonsággal kell rendelkeznie (emlékezzünk, hogy a ⊨ jelentése a „logikailag maga után vonz”):

$$Hipotézis \wedge Leírások \models Besorolások \quad (19.3)$$

Az ilyen kapcsolatot, ahol a *Hipotézis* az „ismeretlen”, **vonzatkényszer**nek (**entailment constraint**) nevezzük. Ennek a kényszernek a megoldása a tiszta induktív tanulás, amikor a *Hipotézis*-t valamilyen előre definiált hipotézistérből vesszük. Ha például a döntési fát logikai formulának tekintjük (lásd 1.1. szakasz - Példák és hipotézisek részben (19.1) képlet), akkor egy döntési fa, amely az összes példával konzisztens, kielégíti a (19.3) képletet. Természetesen abban az esetben, ha a hipotézisek logikai formáját semmiben *nem* korlátozzuk, a *Hipotézis* = *Besorolások* szintén teljesíti a kényszert. Általában az Ockham borotvája elve azt sugallja, hogy *kis*

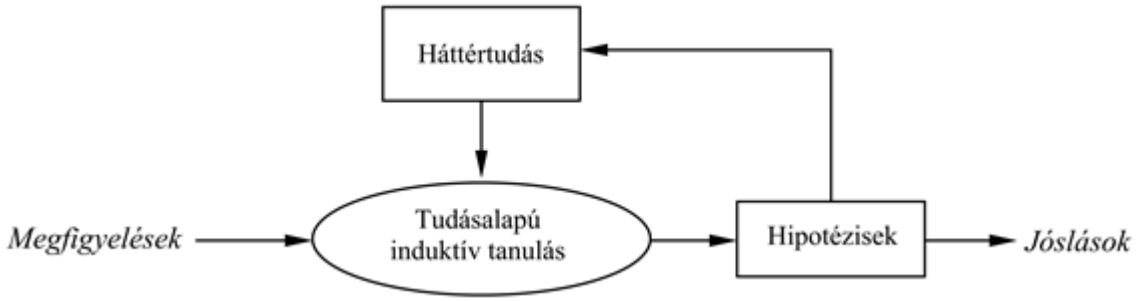
Created by XMLmind XSL-FO Converter.

konzisztens hipotéziseket kellene előnyben részesíteni, valami jobbat kell tehát kitalálnunk annál, minthogy csupán a példákat memorizáljuk.

Fontos

Az induktív tanulás ezen egyszerű képe az 1980-as évek elejéig uralkodott. A korszerű megközelítés az, hogy olyan ágenseket tervezünk, amelyek *már tudnak valamit,* és amelyek igyekeznek, hogy még többet tudjanak. Lehet, hogy ez nem tűnik félelmetesen mély gondolatnak, azonban igen nagy különbséget jelent, ha az ágensek tervezését tekintjük. Sőt az is lehet, hogy ez a megközelítés valamilyen mértékben hozzájárul annak megértéséhez, hogy a tudomány maga hogyan működik. Az általános séma a 19.6. ábrán látható.

19.6. ábra - A kumulatív tanulási folyamat felhasználja és idővel folyamatosan bővíti a háttértudás készletét



Amennyiben egy háttértudást felhasználó autonóm tanuló ágenst szeretnénk felépíteni, az ágensnek legelőször valamilyen módszerrel meg kell szereznie a háttértudást, hogy utána azt az új tanulási epizódokban felhasználhassa. E módszernek önmagában is egy tanulási folyamatnak kell lennie. Az ágens életútját így *kumulatív* vagy *inkrementális* fejlődés jellemzi. Az ágens feltételezhetően a semmiből kezdhet, egy jó kis induktív program mintájára. Ha egyszer azonban evett már a Tudás Fájából, többé ilyen naiv spekulációra nem adhatja a fejét, és a háttértudást a minél hatásosabb tanulásra kell fordítania. A kérdés most az, hogyan tudjuk ezt megvalósítani.

2.1. Néhány egyszerű példa

Nézzünk néhány „józan ész” példát a háttértudás felhasználásával történő tanulásra. Sok, a következtetésen alapuló, látszólag racionális viselkedésről ki fog derülni, hogy ha a megfigyeléseket is figyelembe kell venni, nyilvánvaló lesz, hogy az ágens nem a tiszta indukció egyszerű elvét követi.

- Néha az általános konklúziót már egyetlenegy megfigyelés alapján megfogalmazhatjuk. Gary Larson egy karikatúrájában Zog, a szemüveges ősember egy fanyárs végére tűzött gyíkot süt a tűz felett. Kevésbé intelligens barlangtársainak bámulatba ejtett csoportja figyeli: ők az ennivalót a tűz felett csupasz kezükben tartják. Ez a megvilágosító tapasztalat elegendő, hogy a csoport tagjai a fájdalommentes főzés elvét elsajátítsák.

- Vagy vegyük például a Brazíliába érkező utazó esetét, amikor is az első brazil bennszülöttel találkozik. Hallván, hogy az portugálul beszél, utazónk egyből megállapítja, hogy Brazíliában mindenki beszél portugálul, felfedezvén azonban, hogy a bennszülöttet Fernandónak hívják, ezt a felfedezését az összes brazil férfira egyáltalán nem terjeszti ki. Hasonló esetek a tudományra is jellemzők. Amikor például egy kezdő fizikushallgató egy adott hőmérsékletű rézmintának a sűrűségét és az elektromos vezetőképességét méri, a mérési eredményeket magabiztosan általánosítja a réz minden lehetséges darabkájára. Ha azonban a minta tömegét méri, meg sem fordul a fejében, hogy más rézdarabkáknak is ez lenne a tömege. Ez az általánosító feltevés viszont egészen értelmes lenne, ha például az amerikai 1 centes érmékről lenne szó.

- Végül vegyük egy gyógykezelésben járatlan, diagnosztikai szempontból azonban képzett orvostanhallgató esetét, aki egy páciens és egy szakértő belgyógyász konzultációját figyeli. Egy sor kérdés és felelet után a szakértő egy bizonyos antibiotikumos gyógykezelést ír elő a páciens részére. A hallgatónk azt az általános szabályt fogalmazza meg, hogy ez a konkrét antibiotikum hatásos erre a konkrét típusú fertőzésre.

Created by XMLmind XSL-FO Converter.

Fontos

Ezek mind olyan esetek, amikor *a háttértudás felhasználása sokkal gyorsabb tanulást tesz lehetővé ahhoz képest, mint amit a tisztán induktív programtól el lehetne várni*.

2.2. Néhány általános séma

Az előző példákban azért fordulunk a háttértudáshoz, hogy a megválasztott általánosítást megpróbáljuk igazolni. Most azt nézzük meg, mik azok a vonzatkényszerek, amelyek ott rejtőznek. A kényszerek a *Hipotézis*-en, a megfigyelt *Leírások*-on és a *Besorolások*-on kívül a *Háttértudás*-ra is kiterjednek.

A gyíksütés esetén az ősemberek úgy általánosítottak, hogy *megmagyarázták* a nyárs sikerét: a nyárs megtartja a gyíkot a tűz felett, miközben a kéz sértetlen marad. Ebből a magyarázatból azt az általános szabályt vonhatják le, hogy egy hosszú, merev, hegyes objektum alkalmas a kis, puha testű, ehető dolgok sütéséhez. Az ilyen általánosító folyamatot **magyarázatalapú tanulás**nak vagy **MAT**-nak (**explanation-based learning**, **EBL**) nevezzük. Figyeljük meg, hogy az általános szabály az ősemberek birtokában lévő háttértudásból *logikai módon következik*. A MAT által teljesített vonzatkényszerek tehát a következők:

$$Hipotézis \wedge Leírások \models Besorolások$$

$$Háttértudás \models Hipotézis$$

Fontos

Az első kényszer a (19.3) képlettel megegyezik, azért a MAT-ot eredetileg a példákból való tanulás jobb módszerének tekintették. Azonban amiatt, hogy a háttértudásnak elegendőnek kell lennie a *Hipotézis* megmagyarázásához is, ami viszont a megfigyelést magyarázza meg, *a megfigyelt esetből az ágens igazából semmi tényszerűen újat nem tanul*. Az ágens a példát akár ki is *következtethette volna* az ismert tudásanyag alapján, bár elképzelhető, hogy ez elfogadhatatlan mennyiségű számítással járt volna együtt. Újabban a MAT-ot úgy tekintik, mint az egyik olyan módszert, amely elsődleges elméleteknek speciális célú tudásba való konvertálását végzi. A MAT algoritmussal a 19.3. alfejezetben foglalkozunk.

A Brazíliába utazónak egészen más a helyzete. Az utas szükségszerűen nem is képes megmagyarázni, hogy Fernando miért beszél úgy, ahogy beszél, amíg az idevágó pápai bullákat nem ismeri. Az általánosításra azonban képes lesz a történelemben teljesen járatlan utas is. Ebben az esetben a releváns háttértudás az, hogy minden egyes országban az emberek többsége ugyanazt a nyelvet beszéli. Azt azonban nem tételezzük fel, hogy minden brazil embert Fernandónak hívnak, mert az effajta szabályosság a nevekre nem vonatkozik. Hasonlóképpen, az elsőéves fizikushallgató is nehezen tud magyarázatot adni a réznél felfedezett vezetési és sűrűségi értékekre. Azt azonban tudja, hogy a vezetést az objektum anyaga és annak hőmérséklete határozza meg együttesen. Mindegyik esetben az *a priori Háttértudás* szerepe az, hogy a célpredikátum szempontjából **releváns** tulajdonsághalmazok körülhatárolhatók legyenek. Ez a tudás *megfigyelésekkel együtt* lehetővé teszi, hogy az ágens új, általános szabályt hozzon létre megfigyeléseinek megmagyarázására:

$$Hipotézis \wedge Leírások \models Besorolások$$

$$Háttertudás \wedge Leírások \wedge Besorolások \models Hipotézis \quad (19.4)$$

Ezt a fajta általánosítást **relevanciaalapú tanulás**nak vagy **RAT**-nak (**relevance-based learning**, **RBL**) nevezzük (bár ez az elnevezés még nem mondható teljesen elfogadottnak). Vegyük észre, hogy annak ellenére, hogy a RAT a megfigyelések tartalmát felhasználja, a megfigyelések és a háttértudás logikai tartalmán túlmutató hipotéziseket nem eredményez. Ez a tanulás egy *deduktív* formája, és egymagában nem lehet felelős új tudásanyagnak az alapokból való létrehozásáért.

A szakértőt figyelő orvostanhallgató esetében feltételezzük, hogy a hallgató előzetes tudása elegendő ahhoz, hogy a páciens *D* betegségét a tünetekből megállapítsa. Ez azonban nem elegendő annak indoklására, hogy az orvos miért éppen az *M* orvosságot írta fel. A hallgatónak egy másik szabályt kell javasolnia, azaz, hogy az *M* orvosság a *D*-vel szemben hatásos. Ezzel a szabállyal és a hallgató előzetes tudásával most már képes magyarázatot adni, hogy az orvos ebben a konkrét esetben miért az *M* gyógyszert írja fel. A példát általánosíthatjuk egy vonzatkényszer megadásával:

$$Háttértudás \wedge Hipotézis \wedge Leírások \models Besorolások \quad (19.5)$$

Created by XMLmind XSL-FO Converter.

Fontos

Példák magyarázatát tehát a háttértudás és az új hipotézis együttesen adja meg. A tisztán induktív tanuláshoz hasonlóan a tanuló algoritmusnak a lehető legegyszerűbb és a kényszerekkel konzisztens hipotéziseket kellene előállítania. A (19.5) kényszert teljesítő algoritmusokat **tudásalapú induktív tanulás**nak vagy **TIT**-nek (**knowledgebased inductive learning**, **KBIL**) nevezzük.

A 19.5. alfejezetben részletesen leírt TIT algoritmusokat főleg az **induktív logikai programozás**, **ILP** (**inductive logical programming**) területén tanulmányozták. Az előzetes tudásnak egy ILP-rendszerben két kulcsszerepe van a tanulás komplexitásának csökkentésében:

1. Tekintettel arra, hogy a megfogalmazott hipotézisek mindegyikének konzisztensnek kell lennie a megfigyelésekkel és az előzetes tudással is, ténylegesen egy redukált hipotézistérrel van dolgunk, hiszen ez a hipotézistér csakis azokat az elméleteket tartalmazza, amelyek az eddig ismert dolgokkal konzisztensek.

2. Bármilyen adott megfigyeléshalmaz esetén a megfigyelések magyarázatát megadó hipotézis lényegesen redukált méretű lehet, hiszen a megfigyeléseket magyarázó új szabályok meghatározásához az előzetes tudás is rendelkezésünkre áll. Viszont minél kisebb a hipotézis, annál könnyebb azt megtalálni.

Az a priori tudás befogadásán túl, az ILP-rendszerek képesek arra, hogy hipotéziseket általános elsőrendű logikában és nem a 18. fejezetben használt, korlátozott attribútumalapú nyelven fogalmazzák meg. Ez azt is jelenti, hogy az ILP-rendszerek képesek olyan környezetekben tanulni, amelyek érthetetlenek az egyszerűbb rendszerek számára.

3. Magyarázatalapú tanulás

Ahogy a fejezet bevezetőjében elmagyaráztuk, a magyarázatalapú tanulás olyan módszer, ami megfigyelésekből általános szabályokat nyer ki. Tekintsük például az algebrai kifejezések egyszerűsítését és differenciálását (lásd 9.15. feladat). X^2-et X szerint differenciálva $2X$-et kapunk (vegyük észre, hogy az aritmetikai ismeretlent nagy X betűvel jelöljük az x logikai változótól megkülönböztetve). Egy logikai következtető rendszerben a célt így a `KÉRDEZ`(*Derivált*(X^2, X) = d, *TB*) kifejezéssel fejezhetjük ki, ahol a megoldás a $d = 2X$.

Bárki, aki a differenciálszámítást ismeri, a megoldást „ránézésre” tudja. Az ilyen problémákkal első alkalommal találkozó hallgatónak vagy netán a tapasztalatot nélkülöző programnak sokkal nehezebb lesz a dolga. A differenciálás közismert szabályai segítségével a kifejezést előbb-utóbb az $1 \times (2 \times (X^{(2-1)}))$ alakra lehet hozni, és ez előbb-utóbb elvezet minket a $2X$-hez. A szerzők logikai programjának ez 136 bizonyítási lépésbe került, amiből 99 lépés a bizonyítás zsákutca elágazásaira esett. Ezt tapasztalva azt szeretnénk, ha a program ugyanennek a problémának a megoldását a következő alkalommal lényegesen gyorsabban adná meg.

A **memoizálás** (**memoization**) módszerét a számítógép-tudományban régóta alkalmazzák, miszerint a számítások meggyorsíthatók az eredmények eltárolása révén. A memofüggvény alapgondolata, hogy bemenet/kimenet párokat gyűjtünk ki egy külön adatbázisba. Meghíváskor a függvény először megvizsgálja az adatbázist, hátha a probléma teljes újbóli megoldását így meg tudja kerülni. A magyarázatalapú tanulás ezt az ötletet továbbfejleszti úgy, hogy egy *általános* szabályt fogalmaz meg, amely az esetek egész osztályát képes lefedni. A differenciálás esetében a memoizálás emlékezne ugyan arra, hogy az X^2-nek X szerinti deriváltja $2X$, a Z^2 Z szerinti deriváltjának kiszámítását azonban teljes egészében az ágensre hagyná. Célszerű lenne, ha egy olyan általános szabályt[2] tudnánk megfogalmazni, amely minden lehetséges u aritmetikai ismeretlen esetén megadná, hogy az u^2-nek u szerint deriváltja $2u$. A logikában ezt az alábbi szabállyal fejezhetjük ki:

$$AritmetikaiIsmeretlen(u) \Rightarrow Derivált(u^2, u) = 2u$$

Ha a tudásbázis rendelkezik egy ilyen szabállyal, akkor minden új eset, ami a szabály egy példánya, azonnal megoldható.

Fontos

Természetesen ez csupán egy triviális példa egy igen általános jelenségre. Ha egyszer megértettünk valamit, azt általánosítva más körülmények között is felhasználhatjuk. Egy „nyilvánvaló” megoldási

[2] Természetesen az u^n általános szabályát is meg tudnánk valósítani, a mondanivalót azonban a közölt példa is jól illusztrálja.

Created by XMLmind XSL-FO Converter.

lépést kapunk, amely építőkockaként felhasználható bonyolultabb problémák megoldásában. Alfred North Whitehead, aki Bertrand Russell-lel együtt a *Principia Mathematica* társszerzője – talán a Zog felfedezése jellegű események megértéséhez saját magára alkalmazva a MAT elvét – azt írta, hogy *„A civilizáció azáltal halad előre, hogy szaporítja azoknak a fontos cselekvéseknek a számát, amelyeket gondolkodás nélkül véghezvihetünk”* (Whitehead, 1911). Ha ön, kedves olvasó, a deriválási példa lényegét megértette, akkor az agya már buzgón azon dolgozik, hogy megkísérelje a magyarázatalapú tanulás általános elvét kinyerni ebből a példából. Figyelje meg, hogy hacsak nem lényegesen okosabb a szerzőknél, a magyarázatalapú példa bemutatása előtt *még nem* ismerte fel a MAT-ot. A Zogot figyelő ősemberekhez hasonlóan önnek (és nekünk is) egy példát kellett látnunk, mielőtt az alapelvet meg tudtuk fogalmazni. Ez azért van így, mert *megmagyarázni,* hogy *miért* jó egy ötlet, sokkal könnyebb, mint magát az ötletet megfogalmazni.

3.1. Általános szabályok kinyerése példákból

A MAT alapötlete az, hogy először az előzetes tudásra alapozva megkonstruáljuk a megfigyelés magyarázatát, majd meghatározzuk annak az esetosztálynak a definícióját, amelyre a megkonstruált magyarázat alkalmazható. A definíció alapul szolgál az osztály eseteit lefedő szabály számára. A „magyarázat” lehet egy logikai bizonyítás, de általánosságban lehet akármilyen következtetési vagy problémamegoldó folyamat, feltéve, hogy a lépései jól definiáltak. Kulcsfontosságú azoknak a szükséges feltételeknek az azonosítása, amelyek révén az egyes lépések más esetre is alkalmazhatók lesznek.

Következtető rendszerként a 9. fejezetben leírt egyszerű, hátrahaladó tételbizonyító rendszert fogjuk használni. A *Derivált*(X^2, X) = 2X bizonyítási fája túlságosan nagy ahhoz, hogy példaként szerepeljen, ezért az általánosítás módszertanát egy valamivel egyszerűbb példával fogjuk illusztrálni. Tegyük fel, hogy az $1 \times (0 + X)$ kifejezést akarjuk egyszerűsíteni. A tudásbázis az alábbi szabályokat tartalmazza:

Átírás(*u*, *v*) ∧ *Egyszerűsítés*(*v*, *w*) ⇒ *Egyszerűsítés*(*u*, *w*)

Primitív(*u*) ⇒ *Egyszerűsítés*(*u*, *u*)

AritmetikaiIsmeretlen(*u*) ⇒ *Primitív*(*u*)

Szám(*u*) ⇒ *Primitív*(*u*)

Átírás(1 × *u*, *u*)

Átírás(0 + *u*, *u*)

…

Annak bizonyítása, hogy a válasz *X*, a 19.7. ábra felső részében látható. A MAT-módszer egyidőben két bizonyítási fát konstruál. A második bizonyítási fa a *szabad változókkal rendelkező célt* használja, ahol az eredeti cél konstansait változókkal helyettesítettük. Ahogy az eredeti bizonyítás halad előre, úgy halad a szabad változós bizonyítás is, *pontosan ugyanazokat a szabályokat alkalmazva*. Lehetséges, hogy néhány változót közben le kell kötni. Ahhoz, hogy az *Átírás*(1 × *u*, *u*) szabályt használhassuk, az *Átírás*(*x* × (*y* + *z*), *v*) részcél *x* változóját 1-re kell lekötni. Hasonlóképpen az *Átírás*(*y* + *z*, *v*') részcél *y* változóját 0-ra kell lekötni, ha az *Átírás*(0 + *u*, *u*) szabályt szeretnénk alkalmazni. Az általánosított bizonyítási fa birtokában a levelekből (a szükséges kötéseket figyelembe véve) a célpredikátum általános szabályát képezzük:

Átírás(1 × (0 + *z*),0 + *z*) ∧ *Átírás*(0 + *z*, *z*) ∧ *AritmetikaiIsmeretlen*(*z*)

⇒ *Egyszerűsítés*(1 × (0 + *z*), *z*)

Vegyük észre, hogy a bal oldal első két feltétele igaz *függetlenül attól, hogy z-nek mi az értéke*. Így ezeket kihagyhatjuk a szabályból, és eredményül azt kapjuk, hogy:

AritmetikaiIsmeretlen(*z*) ⇒ *Egyszerűsítés*(1 × (0 + *z*), *z*)

Általánosságban elmondható, hogy a végleges szabályból azok a feltételek hagyhatók ki, amelyek nem jelentenek kényszert a szabály jobb oldalán lévő változókra vonatkozólag. Az eredményül kapott szabály továbbra is igaz, sőt még hatékonyabb is lesz. Vegyük észre azt is, hogy az *AritmetikaiIsmeretlen*(*z*) feltétel nem hagyható ki, hiszen a *z*-nek nem minden lehetséges értéke aritmetikai ismeretlen. A *z* más jellegű értékei esetén

Created by XMLmind XSL-FO Converter.

feltehetően más egyszerűsítési szabályokat kellene alkalmazni – például ha *z* netán 2 × 3 lenne, akkor az 1 × (0 + (2 × 3)) helyes egyszerűsítése 6, és nem 2 × 3 lenne.

Összegezve, az alap MAT-módszer a következőképpen működik:

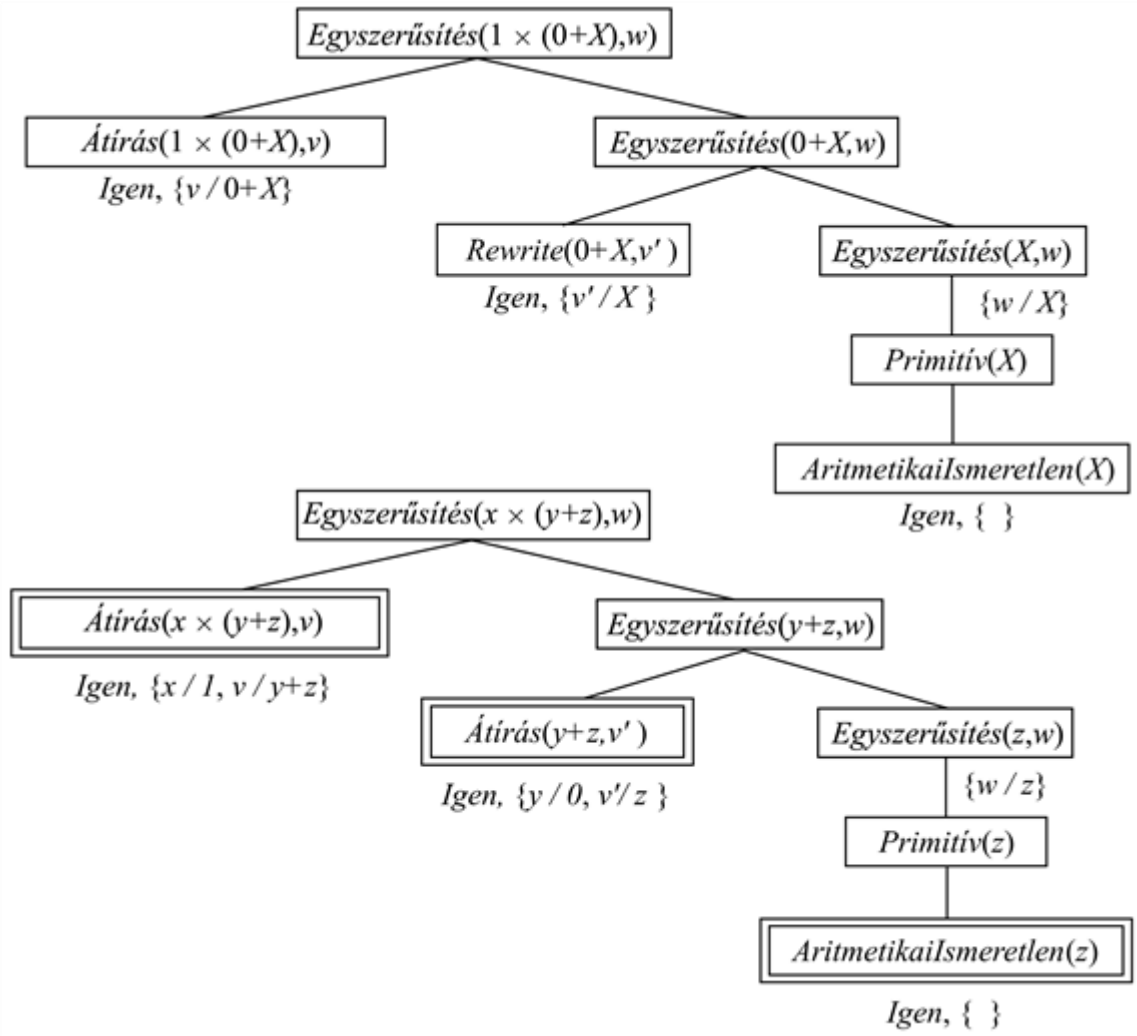
1. Ha adott egy példa, a rendelkezésre álló háttértudást felhasználva konstruáljunk egy bizonyítást arra, hogy a célpredikátum érvényes a példára.

2. Az előbbivel egy időben az eredeti bizonyítás következtetési lépéseivel azonos lépéseket alkalmazva konstruáljuk meg a szabad változós cél általánosított bizonyítási fáját.

3. Konstruáljuk meg az új szabályt úgy, hogy a szabály bal oldala a fa leveleiből áll, a jobb oldala viszont a szabad változós cél (az általánosított bizonyítás változókötéseit figyelembe véve).

4. Hagyjuk ki azokat a feltételeket, amelyek a célban foglalt változók értékeitől nem függnek.

19.7. ábra - Az egyszerűsítési probléma bizonyítási fái. Az első fa az eredeti problémapéldány bizonyítása, amiből az *AritmetikaiIsmeretlen*(*z*) ⇒ *Egyszerűsítés*(1 × (0 + *z*), *z*) vezethető le. A második fa az a bizonyítás, amikor az eredeti problémapéldányban szereplő összes konstanst változókkal helyettesítettük, amiből viszont további szabályok sokaságát vezethetjük le.



3.2. A hatékonyság javítása

Created by XMLmind XSL-FO Converter.

A 19.7. ábrán látható általánosított bizonyítási fából több általánosított szabályt is képesek vagyunk kinyerni. Ha a fa jobb oldali ágának növekedését leállítjuk, vagy az ágat **lemetsszük** (**prune**), amikor *Primitív* lépéshez jutottunk, az alábbi szabályt kapjuk:

```
Primitív(z) ⇒ Egyszerűsítés(1 × (0 + z), z)
```

Ez a szabály annyira érvényes, mint az *AritmetikaiIsmeretlen*-t felhasználó szabály, bár annál általánosabb, mivel olyan esetekre is vonatkozik, amikor *z* numerikus értékű. Még általánosabb szabályokat is kinyerhetünk, ha az *Egyszerűsítés*(*y* + *z*, *w*) után metszünk. Ekkor a szabály:

```
Egyszerűsítés(y + z, w) ⇒ Egyszerűsítés(1 × (y + z), w)
```

Általánosságban elmondható, hogy szabályokat az általánosított bizonyítási fa *bármely részfájából* nyerhetünk. Szembe kell néznünk azonban azzal a problémával, hogy ezek után melyik szabályt válasszuk.

Az a döntés, hogy melyik szabályt érdemes létrehozni, végső soron a hatékonyságon múlik. A MAT által biztosított hatékonyságnövekedésnek három tényezője van:

1. A tudásbázishoz nagyszámú szabály hozzáadása a következtetési folyamatot lelassíthatja, mivel a következtetési mechanizmusnak ezeket a szabályokat akkor is meg kell vizsgálnia, amikor ezek nem járulnak hozzá a megoldáshoz. Másképpen fogalmazva, a keresési tér **elágazási tényező**je (**branching factor**) ilyenkor növekszik.

2. Ahhoz, hogy ezt a jelenséget kompenzáljuk, a létrehozott szabályoknak lényeges sebességnövekedést kell garantálniuk az általuk lefedett problémapéldányok esetében. Az ilyen sebességnövekedés főleg abból származik, hogy a származtatott szabályok révén elkerülhetjük azokat a holtágakat, amelyekbe különben a bizonyítás során belemennénk, illetve abból, hogy a bizonyítások rövidebbek lesznek.

3. A származtatott szabályoknak a lehető legáltalánosabbaknak kell lenniük, hogy a lehető legnagyobb esethalmazt fedjék le.

A származtatott szabályok hatékonyságát megszokott módon úgy biztosíthatjuk, hogy megköveteljük a szabály minden részcéljától, hogy **hatásos** (**operational**) legyen. Durván fogalmazva, egy cél hatásos, ha „könnyű” megoldani. A *Primitív*(*z*) részcélt például könnyű megoldani, a megoldásához két lépés elegendő. Ezzel szemben az *Egyszerűsítés*(*y* + *z*, *w*) részcél tetszőleges számú következtetéshez vezethet az *y* és a *z* értékeinek függvényében. Ha az általánosított bizonyítás megkonstruálásakor a hatásossági tesztet minden lépésnél elvégezzük, az ágat azonnal lemetszhetjük, amint egy hatásos részcélra rátaláltunk. A hatásos részcélt ilyenkor az új szabály egy konjunktív elemeként megtartjuk.

Sajnos a hatásosság és az általánosság között általában kompromisszumot kell kötni. A konkrétabb részcélokat könnyebb megoldani, viszont azok kevesebb esetet fednek le. A hatásosság tovább fokozható; egy vagy két lépés nyilvánvalóan hatásos, de mi a helyzet 10-zel vagy 100-zal? Egy adott részcél megoldásának a költsége végül attól függ, hogy a tudásbázis milyen egyéb szabályokat tartalmaz. A költség növekedhet vagy csökkenthet, ahogy a tudásbázishoz új szabályokat adunk. Az adott kezdeti tudásbázis hatékonyságának maximalizálásánál a MAT-rendszerek valóban igen komplex optimizációs problémával kerülnek szembe. Néha megalkotható annak matematikai modellje, hogy egy adott szabály hozzáadása általában milyen hatással van a hatékonyságra, és ennek a modellnek a használatával a hozzáadandó legjobb szabály megválasztható. Az elemzés azonban igen komplikált lehet, különösképpen ha rekurzív szabályokkal van dolgunk. Ígéretes megközelítés a hatékonyság empirikus vizsgálata, amikor néhány szabály hozzáadásával meggyőződünk arról, hogy a szabályok közül melyik az, amelyik ténylegesen hasznos, és amelyik az eljárásokat felgyorsítja.

Fontos

A hatékonyság empirikus elemzésének gondolata a MAT lényegét érinti. Amit eddig informálisan „az adott tudásbázis hatékonyságának” neveztünk, az nem más, mint egy átlagos esetkomplexitás a megoldandó problémák egy eloszlásán mérve. A *régi példák általánosításával a MAT növeli a tudásbázis hatékonyságát a jövőben ésszerűen várható problémák szempontjából*. Az ötlet addig jó, amíg a régi példák eloszlása durván megegyezik a jövőbeli példák eloszlásával. Ez a feltételezés azonos azzal, amit a 18.5. alfejezetben a VKH-tanulás kapcsán megtettünk. Ha a MAT-rendszert gondosan kivitelezték, a későbbi problémáknál lényeges javulás érhető el. Egy nagyon nagy méretű, svéd és angol nyelv közötti beszédfordításra kifejlesztett, Prolog-alapú, természetes nyelvi rendszerben

Created by XMLmind XSL-FO Converter.

például, a valós idejű fordítási képességet csak az elemző folyamatra alkalmazott MAT révén sikerült elérni (Samuelsson és Rayner, 1991).

4. Tanulás releváns információ alapján

Úgy tűnik, hogy a Brazíliába utazónk képes a Brazíliában beszélt nyelvre vonatkozó megbízható általánosításra. A következtetésre az a háttértudása jogosítja, miszerint egy adott ország lakói (általában) ugyanazt a nyelvet beszélik. Ezt a tudást az elsőrendű logikában az alábbi módon fejezhetjük ki:[3]

$$Nemzetisége(x, n) \land Nemzetisége(y, n) \land Nyelve(x, l) \Rightarrow Nyelve(y, l) \quad (19.6)$$

(szó szerint: „Ha x és y azonos n állampolgárságú, és x az l nyelvet beszéli, akkor y szintén az l nyelvet beszéli.”). Nem nehéz kimutatni, hogy a fenti kijelentésből és a

$$Nemzetisége(Fernando, Brazil) \land Nyelve(Fernando, Portugál)$$

megfigyelésből logikailag következik (lásd 19.1. feladat), hogy:

$$Nemzetisége(x, Brazil) \Rightarrow Nyelve(x, Portugál)$$

A (19.6) típusú mondatok szigorú relevanciát fejeznek ki: a nemzetiség teljesen meghatározza a nyelvet. Másképpen fogalmazva: a nyelv a nemzetiség függvénye. Az ilyen mondatokat **funkcionális függőségek**nek (**functional dependencies**) vagy **meghatározás**oknak (**determination**s) nevezzük. Bizonyosfajta alkalmazásoknál (például az adatbázisrendszer-tervek specifikálásánál) ezek annyira általánosan fordulnak elő, hogy a felírásukhoz speciális szintaxist használnak. Davies (Davies, 1985) jelölésével élve:

$$Nemzetisége(x, n) \succ Nyelve(x, l)$$

Ez persze csak egy egyszerű szintaktikai nyalánkság, azonban világosan mutatja, hogy egy meghatározás valójában predikátumok közötti reláció: a nemzetiség meghatározza a nyelvet. Az anyag vezetőképességét és sűrűségét meghatározó releváns tulajdonságokat hasonlóképpen kifejezhetjük:

$$Anyaga(x, m) \land Hőmérséklete(x, t) \succ Vezetőképessége(x, \rho)$$

$$Anyaga(x, m) \land Hőmérséklete(x, t) \succ Sűrűsége(x, d)$$

A hozzá tartozó általánosítások a meghatározásokból és a megfigyelésekből származnak.

4.1. A hipotézistér meghatározása

Fontos

Annak ellenére, hogy meghatározások igazolják az összes brazilra vagy az adott hőmérsékletű réz minden darabjára vonatkozó általános következtetések levonását, ahhoz természetesen nem elegendők, hogy egyetlenegy példából következtessünk ki egy általános prediktív tulajdonságú elméletet, amely az *összes* nemzetiségre, az *összes* hőmérsékletre és anyagra vonatkozik. Hatásuk úgy képzelhető el legjobban, hogy leszűkítik a tanuló ágens által figyelembe veendő hipotézisek terét. A vezetőképesség előrejelzésénél például csupán az anyagra és a hőmérsékletre kell koncentrálnunk, figyelmen kívül hagyva a tömeget, a tulajdonviszonyokat, a hét napját, a jelenlegi államelnököt stb. A hipotézisek természetesen tartalmazhatnak olyan komponenseket, mint például a molekuláris struktúra, a termikus energia, a szabad elekronok sűrűsége, amit az anyag és a hőmérséklet befolyásol. *A meghatározások egy elégséges alapszótárt adnak meg ahhoz, hogy a célpredikátumra vonatkozó hipotéziseket megkonstruáljuk*. Ezt a kijelentést úgy láthatjuk be, hogy megmutatjuk: egy adott meghatározás logikailag ekvivalens azzal az állítással, miszerint a célpredikátum helyes definíciója a meghatározás bal oldalán lévő predikátumok által kifejezhető definíciók egyike.

Intuitív módon érthető, hogy a hipotézistér méretének számítási redukálásának könnyebbé kell tennie a célpredikátum megtanulását. A számítási tanulás elmélet alaperedményeit (lásd 18.5. alfejezet) felhasználva, a

[3] Az egyszerűség kedvéért feltételezzük, hogy egy személy csak egy nyelven beszél. Az olyan országok esetén, mint Svájc vagy India, a szabályt természetesen bővíteni kell.

Created by XMLmind XSL-FO Converter.

lehetséges nyereség mennyiségileg is kifejezhető. Emlékezzünk arra, hogy egy Boole-függvény esetén log(|**H**|) számú példa (ahol |**H**| a hipotézistér mérete) szükséges ahhoz, hogy egy elfogadható hipotézishez konvergáljunk. Ha a tanuló n Boole-tulajdonsággal rendelkezik a hipotézisek megkonstruálásához, akkor egyéb korlátozások hiányában $|\mathbf{H}| = O\left(2^{2^n}\right)$, így a példák száma $O(2^n)$. Ha a meghatározás bal oldalán d számú predikátum áll, a tanulónak $O(2^d)$ példára van szüksége, ami $O(2^{n-d})$ méretű redukciót jelent. Elfogult hipotézistér, például konjuktív elfogultságok esetén, a redukció még mindig számottevő lesz, bár kevésbé lesz látványos.

19.8. ábra - Algoritmus egy minimális konzisztens meghatározás megkeresésére

```
function MINIMÁLIS-KONZISZTENS-MEGH(E, A) returns az attribútumok egy halmaza
  inputs: E, a példák egy halmaza
          A, az n nagyságú attribútumok egy halmaza

  for i ← 0, …, n do
    for each A_i, az A i nagyságú részhalmaza do
      if KONZISZTENS-MEGH?(A_i, E) then return A_i

function KONZISZTENS-MEGH?(A, E) returns egy igazságérték
  inputs: A, az attribútumok egy halmaza
          E, a példák egy halmaza
  local variables: H, egy hash-tábla

  for each e példa in E do
    if valamelyik példa a H-ban az A attribútumokra ugyanolyan értékű, mint e,
      de eltérő besorolással rendelkezik then return hamis
    tároljuk e besorolását H-ban, az e példa A attribútumainak értékeivel indexelve
  return igaz
```

4.2. Tanulás releváns információ felhasználásával

Ahogy ezt a fejezet bevezetőjében már megemlítettük, az előzetes tudás hasznos a tanulás szempontjából, azonban valahogy ezt is meg kell tanulni. Hogy a relevanciaalapú tanulást teljes mértékben megfogalmazhassuk a meghatározások tanulási mechanizmusát meg kell még adnunk. A most bemutatásra kerülő algoritmus alapja a megfigyelésekkel konzisztens legegyszerűbb meghatározás megtalálása. Egy $P \succ Q$ meghatározás azt jelenti, hogy ha egy példa illeszkedik P-re, akkor Q-ra is kell illeszkednie. Egy meghatározás konzisztens a példák egy halmazával, ha minden példapár, amely a bal oldali predikátumokon illeszkedik egymásra, a célpredikátumon is illeszkedik, magyarán, ha a besorolása azonos. Tegyük fel, hogy bizonyos anyagi mintákat megmérve a vezetőképesség alábbi méréseivel rendelkezünk:

Minta	Tömeg	Hőmérséklet	Anyag	Méret	Vezetőképesség
S1	12	26	réz	3	0,59
S1	12	100	réz	3	0,57
S2	24	26	réz	6	0,59
S3	12	26	ólom	2	0,05
S3	12	100	ólom	2	0,04
S4	24	26	ólom	4	0,05

A minimális konzisztens meghatározás: *Anyaga* ∧ *Hőmérséklete* ≻ *Vezetőképessége*. Létezik konzisztens, de nem minimális meghatározás is: *Tömege* ∧ *Nagysága* ∧ *Hőmérséklete* ≻ *Vezetőképessége*. Ez azért konzisztens a példákkal, mert a tömeg és a méret meghatározza a sűrűséget, és az adatbázisunkban nincs két különböző anyag, amelynek sűrűsége azonos lenne. Mint mindig, most is nagyobb példahalmazra lenne szükségünk, hogy a közel helyes hipotézist kiszűrjük.

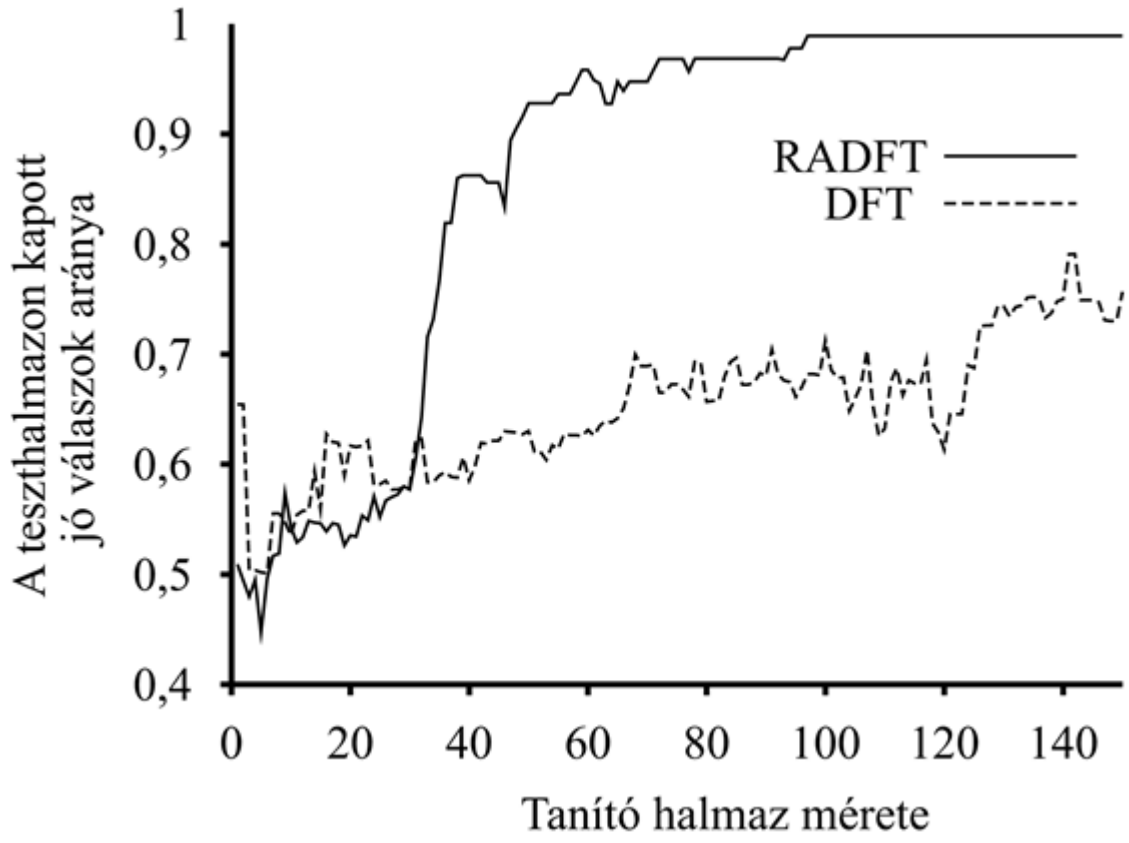
Created by XMLmind XSL-FO Converter.

A minimális konzisztens meghatározás megkeresésének több lehetséges algoritmusa létezik. Kézenfekvő megközelítés a meghatározások terében keresni, először az egypredikátumos, a kétpredikátumos stb. meghatározásokat ellenőrizve, amíg rá nem lelünk egy konzisztens meghatározásra. Egyszerű attribútumalapú ábrázolást tételezünk fel, hasonlóan ahhoz, mint amit a döntési fák tanulásánál a 18. fejezetben használtunk. Tekintettel arra, hogy a célpredikátumot rögzítettnek vesszük, a *d* meghatározást a bal oldal attribútumhalmaza fogja képviselni. Az algoritmust a 19.8. ábra mutatja.

Ennek az algoritmusnak az időigénye a legkisebb konzisztens meghatározás nagyságától függ. Tegyük fel, hogy az ilyen meghatározásnak *p* attribútuma van a lehetséges *n*-ből. Az algoritmus addig nem lesz képes a meghatározást megtalálni, amíg az *A*-nak *p* nagyságú részhalmazait nem kezdi végigvizsgálni. Miután $\binom{n}{p} = O(n^p)$ ilyen részhalmaz létezik, következésképpen az algoritmus a minimális meghatározás méretét tekintve exponenciális lesz. Az is kiderült, hogy a probléma NP-teljes, így általános esetben sem várhatunk jobbat. A legtöbb tárgytartomány azonban elegendő lokális strukturáltsággal (a lokálisan strukturált tárgytartomány definíciójára lásd 14. fejezet) rendelkezik, így *p* minden bizonnyal kicsi lesz.

Ha a meghatározások tanuló algoritmusa már adott, a tanuló ágensnek módja van egy olyan minimális hipotézist megkonstruálni, amelyen belül kell a célpredikátumot megtanulnia. Összemásolhatjuk a `MINIMÁLIS-KONZISZTENS-MEGH` és a `DÖNTÉSI-FA-TANULÁS` algoritmusait, és így az RADFT-t, a relevanciaalapú döntési fa tanuló algoritmust kapjuk, amely először a releváns attribútumok minimális halmazát azonosítja, majd ezen halmazt tanulás céljából döntési fa algoritmusának adja tovább. A `DÖNTÉSI-FA-TANULÁS`-sal ellentétben az RADFT egyidejűleg tanulja és használja a releváns információt ahhoz, hogy a hipotézisteret minimalizálja. Elvárjuk, hogy az RADFT tanulási görbéje a `DÖNTÉSI-FA-TANULÁS` tanulási görbéjénél jobb legyen, és tényleg ez a helyzet. A 19.9. ábrán a két algoritmus tanulási görbéje látható, véletlen módon generált adatokon és egy olyan tanulandó függvény esetén, amely a lehetséges 16 attribútumból csupán 5-öt használ fel. Természetesen azokban az esetekben, amikor az összes attribútum releváns, az RADFT nem mutat előnyöket.

19.9. ábra - Az RADFT és a `DÖNTÉSI-FA-TANULÁS` hatékonyságának összehasonlítása véletlen módon generált adatok és egy olyan célfüggvény esetén, amely a lehetséges 16 attribútumból csak 5-öt használ



Created by XMLmind XSL-FO Converter.

Ez a rész csak érintette a **deklaratív elfogultság** (**declarative bias**) területét, melynek az a célja, hogy megértsük, hogy az előzetes tudást hogyan kell felhasználni a megfelelő hipotézistér azonosításához, amelyben a korrekt céldefiníciót keressük. Sok kérdés megválaszolatlan maradt:

- Hogyan kell az algoritmust kiterjeszteni a zaj kezelésére is?
- Tudunk-e folytonos értékeket felvevő változókat kezelni?
- Hogyan lehetne felhasználni az előzetes tudás más típusát is, a meghatározásokon kívül?
- Hogyan lehet az algoritmusokat általánosítani, hogy egy elsőrendű elméletre, és nem csupán az attribútumalapú reprezentációra vonatkozzanak ?

Az egyes kérdésekre a következő alfejezetben keresünk választ.

5. Induktív logikai programozás

Az induktív logikai programozás (ILP) az induktív módszereket az elsőrendű reprezentációk erejével kombinálva, az elméletek logikai programok formájában történő kifejezésére helyezi a hangsúlyt.[4] Az ILP három oknál fogva terjedt el. Először is az ILP az általános tudásalapú induktív tanulási probléma precíz, szigorú megközelítését adja. Másodszor, teljes algoritmusokat szolgáltat az általános elsőrendű elméletek indukciós úton, példák alapján történő előállítására. Következésképpen olyan területeken is képes sikeresen tanulni, ahol az attribútumalapú algoritmusokat nehéz alkalmazni. Ennek egy példája a fehérjemolekulák összehajtogatásának a tanulása (lásd 19.10. ábra). A fehérjemolekula háromdimenziós konfigurációját egy attribútumhalmazzal ésszerűen kifejezni nem lehet, mert a konfiguráció lényege az objektumok közötti *relációkra* vonatkozik, és nem az egyes objektumok attribútumaira. A relációk leírására megfelelő apparátus az elsőrendű logika. Harmadszor, induktív logikai programozás által létrehozott hipotézisek emberek számára (viszonylag) könnyen olvashatók. A 19.10. ábrabeli természetes nyelvű fordítást a gyakorló biológusok végigelemezhetik és kritizálhatják. Ez azt jelenti, hogy az induktív logikai programrendszerek részt vehetnek a kísérletezés, a hipotézisgenerálás, a megvitatás és a cáfolat tudományos ciklusában. Az ilyen részvétel a „fekete doboz” osztályozókat gyártó módszerek számára, mint amilyenek például a neurális hálók, lehetetlen lenne.

5.1. Egy példa

Emlékezzünk a (19.5) egyenlet alapján, hogy az általános tudásalapú indukciós probléma az alábbi vonzatkényszer:

$$Háttértudás \wedge Hipotézis \wedge Leírások \models Besorolások$$

„megoldása” az ismeretlen *Hipotézis*-re nézve, feltéve, hogy adottak a *Háttértudás*, valamint a *Leírások* és a *Besorolások* által leírt példák. Hogy ezt megvilágítsuk, tekintsük a rokonsági relációk példák alapján történő tanulását. A megfigyeléseket egy kiterjedt családfa képezi, az *Anyja*, *Apja*, *Házas* relációkkal, valamint a *Férfi* és *Nő* tulajdonságokkal leírva. A 8.11. feladat családfáját fogjuk használni, amit a 19.11. ábra mutat. A példához tartozó leírások az alábbiak:

```
Apja(Fülöp, Károly) Apja(Fülöp, Anna) ...

Anyja(Mami, Margit) Anyja(Mami, Erzsébet) ...

Házas(Diana, Károly) Házas(Erzsébet, Fülöp) ...

Férfi(Fülöp) Férfi(Károly) ...

Nő(Beatrix) Nő(Margit) ...
```

A *Besorolások* állításai azon múlnak, hogy milyen célfogalmat szeretnénk megtanulni. A célpredikátumok olyan fogalmak lehetnének, mint például a *Nagyszülője*, a *Sógora*, illetve az *őse*. A *Nagyszülője* esetén a *Besorolások* teljes halmaza $20 \times 20 = 400$

```
Nagyszülője(Mami, Károly) Nagyszülője(Erzsébet, Beatrix) …
```

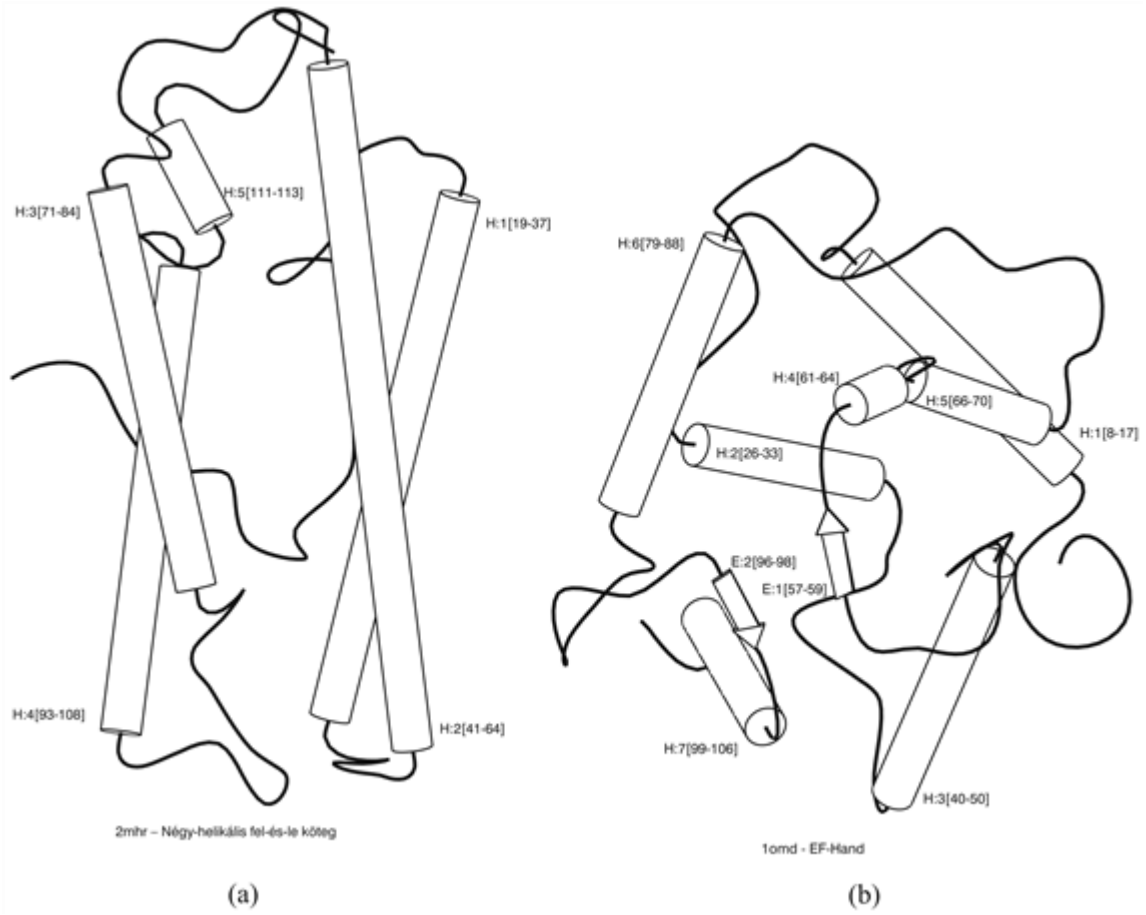
[4] Azt javasoljuk, hogy ezen a ponton az olvasó ismételten fusson végig a 9. fejezetben bemutatott néhány fogalmon, beleértve a Horn-klózokat, a konjunktív normál formát, az egyesítést és a rezolúciót.

Created by XMLmind XSL-FO Converter.

¬*Nagyszülője*(*Mami*, *Harry*) ¬*Nagyszülője*(*Spencer*, *Péter*) …

alakú konjunktból áll. Természetesen e teljes halmaz részhalmazából is tudnánk tanulni.

19.10. ábra - Az (a) és (b) ábra a fehérjemolekula összehajtogatásának tárgyterületén a „négy-helikális fel-és-le köteg” fogalom pozitív és negatív példáját mutatja. Mindkét példa struktúráját egy olyan kb. 100 konjunktív tagot tartalmazó logikai kifejezésbe kódolták, mint amilyen például a *TeljesHossz*(*D2mhr*, 118) ∧ *HelikálisSzám*(*D2mhr*, 6) ∧ … kifejezés.



Az ilyen leírásokból és az *Összehajtogatás*(NÉGY-HELIKÁLIS-FEL-ÉS-LE-KÖTEG, *D2mhr*) jellegű besorolásokból a PROGOL induktív logikai programrendszer (Muggleton, 1995) az alábbi szabályt tanulta meg:

Összehajtogatás(Négy-Helikális-Fel-és-Le-Köteg, *f*) ⇐

Helikális(*f*, cs_1) ∧ *Hossz*(cs_1, Magas) ∧ *Pozíció*(*f*, cs_1, *n*)

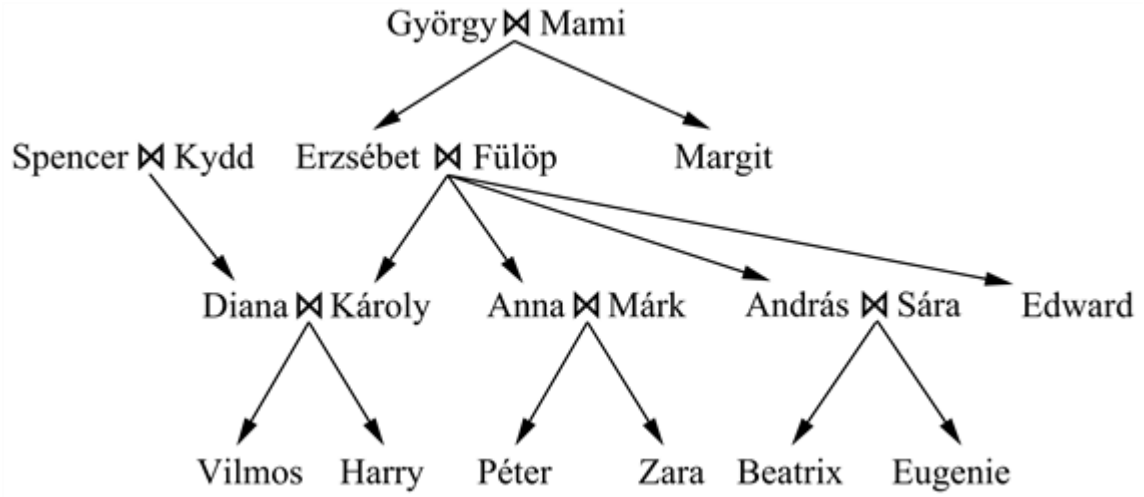
∧ (1 ≤ *n* ≤ 3) ∧ *Szomszédos*(*f*, cs_2, cs_2) ∧ *Helikális*(*f*, cs_2)

Ilyen típusú szabályt az előbbi fejezetekben látott, attribútumalapú mechanizmusokkal megtanulni, sőt kifejezni sem lehet. A szabály a természetes nyelvben kifejezve:

Az *F* fehérje a „Négy-helikális fel-és-le köteg” összehajtogatási osztályhoz tartozik, ha egy hosszú cs_1 helikálist tartalmaz a másodlagos struktúra 1. és 3. közötti pozícióban, valamint cs_1 a második helikálissal szomszédos.

19.11. ábra - Egy tipikus családfa

Created by XMLmind XSL-FO Converter.



Az induktív tanuló program tárgya egy olyan állításhalmaz *Hipotézis*-ként való előállítása, amely a vonzatkényszert kielégíti. Átmenetileg tegyük fel, hogy az ágens nem rendelkezik háttértudással, a *Háttértudás* üres. Akkor egy lehetséges megoldás a *Hipotézis*-re a következő:

$$Nagyszülője(x, y) \Leftrightarrow [\exists z \ \ Anyja(x, z) \wedge Anyja(z, y)]$$

$$[\exists z \ \ Anyja(x, z) \wedge Apja(z, y)]$$

$$[\exists z \ \ Apja(x, z) \wedge Anyja(z, y)]$$

$$[\exists z \ Apja(x, z) \wedge Apja(z, y)]$$

Vegyük észre, hogy egy attribútumalapú tanuló algoritmus, mint például a DÖNTÉSI-FA-TANULÁS sehogy sem fog ezzel a problémával boldogulni. Ahhoz, hogy a *Nagyszülője*-t attribútumként (azaz egy unáris predikátumként) fejezzük ki, az embereket *párosával* objektumként kell tudnunk kezelni:

$$Nagyszülője(\langle Mami, Károly \rangle)\ldots$$

Ezek után ott akadunk el, amikor megkíséreljük egy példa leírását formálisan reprezentálni. Csupán olyan elrettentő attribútumokat tudnánk használni, mint az:

$$ErzsébetAnyjaAzElsőElem(\langle Mami, Károly \rangle)$$

Fontos

A *Nagyszülője*-nek az ilyen attribútumokkal kifejezett definíciója nem lesz más, mint egy nagy diszjunkció, ami az egyes konkrét esetekből áll, és amit lehetetlen az új esetekre általánosítani. Az *attribútumalapú tanuló algoritmusok képtelenek relációs predikátumokat megtanulni*. Az ILP egyik legfontosabb előnye tehát az, hogy a problémák sokkal szélesebb választékában alkalmazhatók, beleértve a relációs problémákat is.

Az olvasó természetesen észreveszi, hogy egy kis háttértudás segítene a *Nagyszülője* definíciójának reprezentációjánál. Így például, ha a *Háttértudás* tartalmazná az alábbi állítást:

$$Szülője(x, y) \Leftrightarrow [Anyja(x, y) \vee Apja(x, y)]$$

akkor a *Nagyszülője* definícióját a:

$$Nagyszülője(x, y) \Leftrightarrow [\exists z \ Szülője(x, z) \wedge Szülője(z, y)]$$

állításra lehetne redukálni. Ez megmutatja, hogy mennyire tud a háttértudás hozzájárulni a példák megmagyarázásához szükséges hipotézis méretének drasztikus csökkentéséhez.

Az ILP-algoritmus új predikátumokat is *létesíthet,* hogy a magyarázó jellegű hipotézis kifejezését egyszerűsítse. Az előbbi példát tekintve, teljesen ésszerű egy pótpredikátumot javasolni – amit *„Szülője”*-nek tudnánk nevezni –, hogy a célpredikátum definícióját egyszerűsítsük. Azok az algoritmusok, amelyek új predikátumokat tudnak

Created by XMLmind XSL-FO Converter.

létesíteni, az ún. **konstruktív indukciós** (**constructive induction**) algoritmusok. Világos, hogy a konstruktív indukció a bevezetőben vázolt kumulatív tanulás egy szükséges komponense. A kumulatív tanulás a gépi tanulás egyik legnehezebb problémája, azonban egyes ILP-technikák kellően hatékony mechanizmusnak bizonyultak a megoldására.

A fejezet hátralévő részében az ILP két alapvető megközelítését fogjuk tanulmányozni. Az első a döntési fák általánosításán, a másik a rezolúciós bizonyítás invertálásán alapul.

5.2. Felülről lefelé tanulási módszerek

Az ILP első megközelítésében egy igen általános szabállyal kezdünk és azt fokozatosan leszűkítjük, hogy az adatokra illeszkedjen. Ez az, ami lényegében a döntési fa tanulásánál történik, ahol a döntési fa fokozatosan növekszik, amíg a megfigyelésekkel konzisztens nem lesz. Hogy az ILP-t magvalósítsuk, elsőrendű literálokat használunk attribútumok helyett, a hipotézis pedig egy klózhalmaz, a döntési fa helyett. Ebben a részben az egyik legelső ILP-programmal, a FOIL-lal foglalkozunk (Quinlan, 1990).

Tegyük fel, hogy továbbra is a *Nagyszülője*(*x, y*) predikátum definícióját szeretnénk megtanulni az előbb közölt családi példák alapján. A döntési fa tanulásához hasonlóan a példákat pozitív és negatív példákra bontjuk. A pozitív példák:

⟨ *György, Anna*⟩ , ⟨ *Fülöp, Péter*⟩ , ⟨ *Spencer, Harry*⟩ , …

míg a negatív példák:

⟨ *György, Erzsébet*⟩ , ⟨ *Harry, Zara*⟩ , ⟨ *Károly, Fülöp*⟩ , …

Figyeljük meg, hogy mindegyik példa egy objektum*páros*, hiszen a *Nagyszülője* egy bináris predikátum. Egészét tekintve a családfában 12 pozitív és 388 negatív (a személyek minden más párosítása) példa található.

A FOIL program klózok egy halmazát alakítja ki, mindegyik *Nagyszülője*(*x, y*)-nal mint fejjel. A klózoknak a 12 pozitív példát a *Nagyszülője*(*x, y*) reláció példányosításának kell osztályozniuk, a többi 388-at viszont kizárni. A klózok Horn-klózok, melyeket negált literálokkal bővítünk ki, ahol a negálást mint kudarcot használjuk, hasonlóan a Prologhoz. Egy üres testű klózzal kezdünk:

⇒ *Nagyszülője*(*x, y*)

Mivel ez a választás minden példát pozitívnak sorol be, leszűkítésre szorul. Ez úgy tehető meg, hogy egy lépésben egy literált adunk a bal oldalhoz. A három lehetséges eset:

Apja(*x, y*) ⇒ *Nagyszülője*(*x, y*)

Szülője(*x, z*) ⇒ *Nagyszülője*(*x, y*)

Apja(*x, z*) ⇒ *Nagyszülője*(*x, y*)

(Vegyük észre, hogy a *Szülője*-t definiáló klózzal a háttértudás már rendelkezik.) E három eset közül az első az összes 12 pozitív példát helytelenül negatívnak sorolja be, így ezt az esetet kizárjuk. A második és a harmadik az összes pozitív példával összhangban van. A második eset azonban a negatív példák nagyobb halmazán – pontosabban kétszer annyi példán – helytelen, mert az anyákat és az apákat is megengedi. Így a választásunk a harmadik esetre esik.

Most az esetet tovább kell szűkíteni, kizárva azokat az eseteket, amikor *x* valamilyen *z* szülője, de *z* mégsem *y* szülője. Egy egyedi *Szülője*(*z, y*) literál hozzáadása a következő klózt eredményezi:

Apja(*x, z*) ∧ *Szülője*(*z, y*) ⇒ *Nagyszülője*(*x, y*)

amely az összes példát helyesen sorolja be. A FOIL képes lesz ezt a literált megtalálni és kiválasztani, és ezzel a tanulási feladatot megoldja. Általánosságban, a helyes megoldás megtalálása előtt, a FOIL-nak számos sikertelen klóz között kell keresnie. Ez a példa a FOIL működésének egyszerű illusztrációja. A teljes algoritmus váza a 19.12. ábrán látható. Az algoritmus lényegében literálról literálra egy klózt konstruál, amíg az meg nem egyezik a pozitív példák valamilyen részhalmazával, és egyik negatív példával sem egyezik. A klóz által lefedett pozitív példákat ekkor a tanító halmazból elhagyjuk, és az eljárást folytatjuk, amíg egyetlen pozitív példa sem marad.

Created by XMLmind XSL-FO Converter.

Az algoritmus két fő komponense szorul magyarázatra – az ÚJ-LITERÁLOK, amely a klózhoz hozzáadandó összes új literált konstruálja, és a LITERÁL-MEGVÁLASZTÁSA, amely kiválasztja a hozzáadandó literált.

19.12. ábra - A FOIL algoritmus váza, amely példákból elsőrendű Horn-klózokat tanul meg. Az ÚJ-LITERÁLOK és a LITERÁL-MEGVÁLASZTÁSA eljárásokat a szövegben magyarázzuk meg.

```
function FOIL(példák, cél) returns a Horn-klózok egy halmaza
  inputs: példák, a példák halmaza
          cél, egy literál a célpredikátum részére
  local variables: klózok, a klózok halmaza, kezdetben üres

  while példák tartalmaznak még pozitív példákat do
    klóz ← ÚJ-KLÓZ(példák, cél)
    a példák-ból hagyjuk el a klóz lefedte példákat
    a klóz-t adjuk hozzá a klózok-hoz
  return klózok

function ÚJ-KLÓZ(példák, cél) returns egy Horn-klóz
  local variables: klóz, a cél-re mutató, üres testű klóz
                   l, a klózhoz hozzáadandó literál
                   kiterjedt-példák, a példák halmaza, az új változóknak adott értékekkel együtt

  kiterjedt_példák ← példák
  while a kiterjedt_példák tartalmaznak még negatív példákat do
    l ← LITERÁL-MEGVÁLASZTÁSA(ÚJ-LITERÁLOK(klóz), kiterjedt_példák)
    l hozzáfűzése a klóz testéhez
    kiterjedt_példák ← a PÉLDA-KITERJEDÉS-t a kiterjedt_példák minden
                       példájára történő alkalmazásával kapott példahalmaz
  return klóz

function PÉLDA-KITERJEDÉS(példa, literál) returns
  if példa kielégíti a literál-t
     then return a példa kiterjesztésével kapott példahalmaz úgy, hogy a literál
                 minden új változóját minden lehetséges konstans értékre lekötöttük
  else return üres halmaz
```

Az ÚJ-LITERÁLOK egy klózt kap bementként, és minden lehetséges „hasznos” literált konstruál, amit a klózhoz hozzá lehetne adni. Vegyük például az alábbi klózt:

$$Apja(x, z) \Rightarrow Nagyszülője(x, y)$$

Háromfajta literált adhatunk hozzá a klózhoz:

1. *Predikátumokat használó literálok:* a literál lehet negált vagy ponált, minden létező predikátum (a célpredikátumot is beleértve) használható, és minden argumentum csakis változó lehet. A predikátumok argumentumai tetszőleges változók lehetnek, egy kikötéssel: minden literálnak tartalmaznia kell *legalább* egy változót egy korábbi literálból vagy a klóz fejéből. Olyan literálok, mint: *Anyja*(*z*, *u*), *Házas*(*z*, *z*), ¬*Férfi*(*y*) és *Nagyszülője*(*v*, *x*) megengedettek, ám *Házas*(*u*, *v*) nem. Megjegyzendő, hogy a klóz fejéből származó predikátumok használata lehetővé teszi, hogy a FOIL a *rekurzív* definíciót is megtanulja.

2. *Egyenlőségi és egyenlőtlenségi literálok:* ezek a klózban már szereplő változókat kötik össze. Így például a klózhoz hozzáadhatjuk, hogy: $z \neq x$. Ezekben a literálokban a felhasználó által specifikált konstansok is felhasználhatók. Az aritmetikát tanulva célszerű kitüntetni a 0-t és az 1-et, a listafüggvényeknél pedig az üres [] listát.

Created by XMLmind XSL-FO Converter.

3. *Aritmetikai összehasonlítások:* folytonos változójú függvények tanulásánál célszerű lehet az olyan literálok hozzáadása, mint az $x > y$ és az $y \leq z$. A döntési fa tanulásához hasonlóan egy konstans küszöbértéket is lehet használni, ami a teszt diszkriminációs erejét maximalizálja.

Mindez a keresési térben nagyon nagy elágazási tényezőhöz vezet (lásd 19.6. feladat), azonban a Foil implementációiban a típusinformációt is fel lehet használni a hipotézistér leszűkítésére. Az embereket és a számokat tartalmazó tárgytartományban például a típusleszűkítés nem engedné, hogy az ÚJ-LITERÁLOK olyan literálokat generáljon, mint a *Szülője*(*x*, *n*), ahol *x* egy ember, *n* viszont egy szám.

A LITERÁL-MEGVÁLASZTÁSA az információnyereségre (lásd 3.4. szakasz - Attribútumteszt-választás rész) valamilyen mértékben hasonló heurisztikát használ, hogy eldöntse, melyik literált adja hozzá a klózhoz. A részletek itt nem különösen fontosak, ráadásul számos kipróbált változat is létezik. A Foil egyik érdekes járulékos tulajdonsága az Ockham borotvája elvének az alkalmazása bizonyos hipotézisek eliminálására. Ha (egy bizonyos mérték szerint) egy klóz hosszabbnak mutatkozik, mint az általa megmagyarázott pozitív példák összhossza, akkor ezt a klózt a továbbiakban nem tekintjük lehetséges hipotézisnek. Ezzel a technikával elkerülhetjük az adatokban található zajra illeszkedő, túlságosan bonyolult klózok származtatását. A zaj és a klózhossz kapcsolatának magyarázatát lásd a 1. szakasz - Statisztikai tanulás részben.

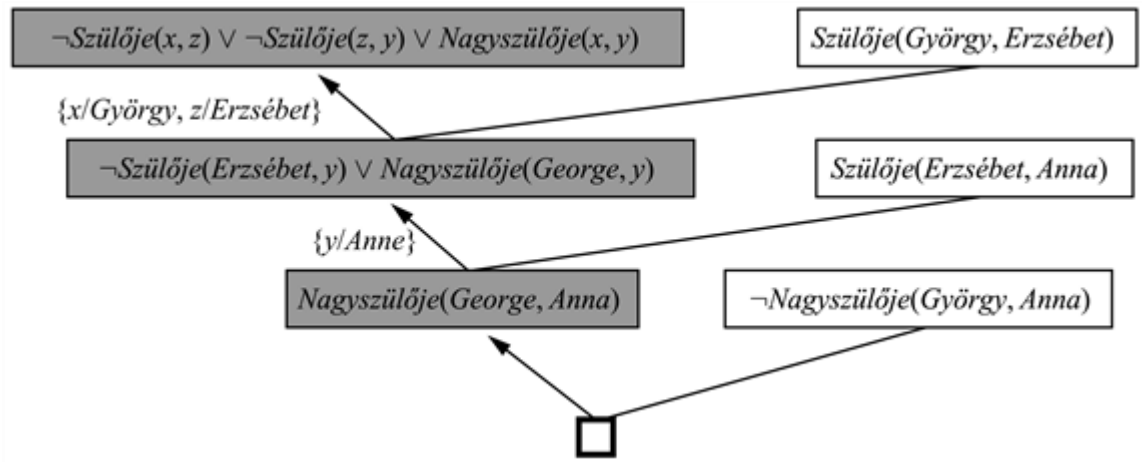
A FOIL-t és társait számos definíció megtanulására vetették be. Az egyik leghatásosabb demonstráció (Quinlan és Cameron-Jones, 1993) a listakezelő függvényekre vonatkozó feladatok hosszú sorának a megoldása volt Bratko Prolog-tankönyvéből (Bratko, 1986). A program mindegyik esetben képes volt a függvény helyes definícióját kisszámú példa alapján megtanulni, miközben háttértudásként használta a korábban megoldott függvényeket.

5.3. Induktív tanulás inverz rezolúcióval

Az ILP-nek másik fő megközelítése a megszokott deduktív bizonyítási folyamat invertálása. Az **inverz rezolúció** (**inverse resolution**) azon az észrevételen alapul, hogy ha a példa *Besorolások* a *Háttértudás* ∧ *Hipotézis* ∧ *Leírások*-ból következnek, akkor ezt a tényt rezolúcióval be kell tudnunk bizonyítani (hiszen a rezolúció teljes). Ha képesek volnánk a bizonyítást „visszafelé pörgetni”, akkor tudnánk egy olyan *Hipotézis*-t találni, amire a bizonyítás sikeres lesz. A kulcsprobléma tehát, hogy hogyan tudnánk a rezolúciós eljárást invertálni, hogy a bizonyítás visszafelé fusson le.

Az inverz rezolúció visszafelé haladó bizonyítási eljárását mutatjuk be, amely egyedi visszafelé mutató lépésekből áll. Egy közönséges rezolúciós lépés a C_1 és C_2 klózból indul ki, és azokat rezolválja egy C **rezolvens**t létrehozva eredményül. Az inverz rezolúciós lépés a C rezolvens alapján két klózt, C_1-et és C_2-t hoz létre úgy, hogy C a C_1 és C_2 rezolválásának az eredménye, vagy pedig C és C_1 alapján állít elő egy lehetséges C_2-t.

19.13. ábra - Az inverz rezolúciós eljárás kezdő lépései. Az árnyékolt klózokat az inverz rezolúciós eljárás generálja a jobb oldalon és az alatta lévő klózokból. A nem árnyékolt klózok a *Leírások*-ból és a *Besorolások*-ból származnak.



Created by XMLmind XSL-FO Converter.

A 19.13. ábra inverz rezolúciós eljárás kezdő lépéseit mutatja, ahol a figyelem középpontjában a *Nagyszülője*(*György*, *Anna*) pozitív példa áll. Az eljárás a bizonyítás végén kezd (amit az ábra alján ábrázolunk). A C rezolvensnek üres klózt (azaz egy ellentmondást) választunk, C_2-nek pedig a ¬*Nagyszülője*(*György*, *Anna*)-t, amely a célpélda negálása. Az első inverz lépés C-ből és C_2-ből C_1-nek a *Nagyszülője*(*György*, *Anna*)-t generálja. A következő lépés e klózt C-nek veszi, a *Szülője*(*Erzsébet*, *Anna*) klózt pedig C_2-nek, és az alábbi C_1 klózt generálja:

¬*Szülője*(*Erzsébet*, *y*) ∨ *Nagyszülője*(*György*, *y*)

Az utolsó lépés ezt a klózt rezolvensként kezeli. A *Szülője*(*György*, *Erzsébet*) klózt C_2-nek véve, egy lehetséges C_1 klóz az alábbi hipotézis lehet:

Szülője(*x*, *z*) ∧ *Szülője*(*z*, *y*) ⇒ *Nagyszülője*(*x*, *y*)

Egy rezolúciós bizonyításunk van tehát arra, hogy a hipotézis, a leírások és a háttértudás maga után vonzza a *Nagyszülője*(*György*, *Anna*) osztályozást.

Világos, hogy az inverz rezolúció keresést tartalmaz. Minden egyes inverz rezolúciós lépés nemdeterminisztikus, hiszen az adott C és C_1-hez több, sőt végtelen sok olyan C_2 klóz lehet, amely kielégíti azt a feltételt, miszerint C_1-gyel rezolválva C-t ad eredményül. Így a 19.13. ábra utolsó lépésében a ¬*Szülője*(*Erzsébet*, *y*) ∨ *Nagyszülője*(*György*, *y*) megválasztása helyett az inverz rezolúciós lépés a következő állításokat is tudná generálni:

¬*Szülője*(*Erzsébet*, *Anna*) ∨ *Nagyszülője*(*György*, *Anna*)

¬*Szülője*(*z*, *Anna*) ∨ *Nagyszülője*(*György*, *Anna*)

¬*Szülője*(*z*, *y*) ∨ *Nagyszülője*(*György*, *y*)

(Lásd a 19.4. és a 19.5. feladatot.) Az egyes lépésekben részt vevő klózokat meg lehet választani a *Háttértudás*-ból, a példa *Leírások*-ból, a negált *Besorolások*-ból vagy az inverz rezolúciós fában eddig már legenerált hipotézisklózokból. A nagyszámú lehetőség, egyéb kontroll nélkül, nagy elágazási tényezőt (és így kevéssé hatékony keresést) jelent. A keresés kézben tartására több megközelítést probáltak ki az implementált ILP-rendszerekben:

1. A redundáns választások eliminálhatók – például a legspecifikusabb lehetséges hipotézisek generálásával, valamint megkövetelve, hogy a hipotézisklózok mind konzisztensek legyenek egymással és a megfigyelésekkel. Az utolsó kritérium kizárná az előbbiekben megadott ¬*Szülője*(*z*, *y*) ∨ *Nagyszülője*(*György*, *y*) klózt.

2. A bizonyítási stratégia szintén megszorítható. A 9. fejezetben láttuk például, hogy a **lineáris rezolúció** (**linear resolution**) egy teljes, megszorított stratégia, amely lehetővé teszi, hogy a bizonyítási fáknak csak lineáris elágazási struktúrája lehessen (mint a 19.13. ábrán).

3. A reprezentációs nyelv is megszorítható, a függvényszimbólumok eliminálásával és csak Horn-klózokat megengedve. A PROGOL például Horn-klózokkal dolgozik, **inverz vonzatreláció**t (**inverse entailment**) alkalmazva. Az ötlet a

Háttértudás ∧ *Hipotézis* ∧ *Leírások* ⊨ *Besorolások*

vonzatkényszer megváltoztatása a

Háttértudás ∧ *Leírások* ∧ ¬*Besorolások* ⊨ ¬*Hipotézisek*

logikailag ekvivalens formára.

Ebből a *Hipotézisek* származtatására már a Prolog-beli normál Horn-klóz dedukcióra hasonlító módszert használhatjuk, negálással mint kudarccal. Mivel ez Horn-klózokra korlátozott, így nem teljes módszer, azonban a teljes rezolúciónál hatékonyabb lehet. Inverz vonzatrelációval lehetséges a teljes következtetés alkalmazása is (Inoue, 2001).

4. A következtetést modellellenőrzéssel is megvalósíthatjuk a tételbizonyítás helyett. A PROGOL rendszer (Muggleton, 1995) a modellellenőrzés egy formáját használja kereséskorlátozás céljára. Azaz, hasonlóan a válaszhalmaz-programozáshoz, a logikai változók lehetséges értékeit generálja, és a konzisztenciát ellenőrzi.

Created by XMLmind XSL-FO Converter.

5. A következtetést az alap ítéletklózokkal is megtehetjük az elsőrendű logika helyett. A LINUS-rendszerben (Lavrac és Dzeroski, 1994) az elsőrendű elméleteket ítéletlogikára fordítják le, ítéletlogikai tanuló rendszerrel oldják meg azokat, majd visszafordítják. Egyes problémák esetén ítéletlogikai állításokkal sokkal hatékonyabban lehet dolgozni, ahogy ezt a 11. fejezetben a SATPLAN esetében már láttuk.

5.4. Felfedezés induktív logikai programozással

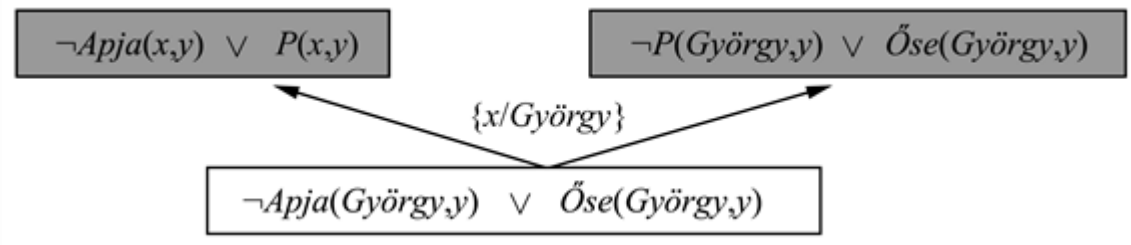
Egy teljes rezolúciós stratégiát invertáló inverz rezolúciós eljárás elvben az elsőrendű elméletek tanulásának egy teljes algoritmusa. Ez azt jelenti, hogy ha valamely ismeretlen *Hipotézis* egy sor példát generál, akkor az inverz rezolúciós eljárás a példákból tudja generálni a *Hipotézis*-t. Ez a megfigyelés egy érdekes lehetőséget is sugall. Tegyük fel, hogy a rendelkezésre álló példák szabadon eső testek által leírt trajektóriák egy halmazát tartalmazzák. Lehetséges, hogy egy inverz rezolúciós program elvben képes a gravitáció törvényét megtanulni? A válasz természetesen igen, mert a gravitáció törvénye, megfelelő matematikai háttér mellett, lehetővé teszi a példák magyarázatát. Hasonlóképpen el lehet képzelni, hogy az elektromágneses hullámok elmélete, a kvantummechanika és a relativitáselmélet, mind beletartoznak az ILP-programok hatáskörébe. Persze a gépíró majom hatáskörébe is beletartoznak; jobb heurisztikákra és a keresési teret jobban strukturáló technikákra még mindig szükség van.

Egy dolgot azért az inverz rezolúciós rendszer meg *fog* tenni a részünkre: új predikátumokat fog kitalálni. Ez a képessége kissé mágikusnak tűnik, hiszen a számítógépre szokás úgy tekinteni, mint ami „csak azzal dolgozik, amit bemenetként kapott”. Tény, hogy az új predikátumok egyenesen az inverz rezolúciós lépésből pottyannak ki. A legegyszerűbb az az eset, amikor egy adott C klóz esetén két új klóz, C_1 és C_2 létezését kellene felvetni. A C_1 és C_2 rezolúciója a két klóz által közösen tartalmazott literált eliminálta, lehetséges hát, hogy az eliminált literálban olyan predikátum is szerepelt, amely C-ben már nem lép fel. Visszafelé haladva tehát lehetőségként adódik egy új predikátum létrehozása, amiből viszont a hiányzó literál már rekonstruálható.

A 19.14. ábra arra mutat példát, ahogy az *Őse* definíciójának tanulása közben egy új P predikátumot generálunk. Ha már előállítottuk a P-t, akkor ez használható az inverz rezolúció későbbi lépéseiben is. Egy későbbi lépés például feltételezheti, hogy az $Anyja(x, y) \Rightarrow P(x, y)$. Az új predikátum jelentését így a predikátumot felhasználó hipotézis generálása korlátozza. Egy másik példa elvezethet az $Apja(x, y) \Rightarrow P(x, y)$-ig. Más szóval a P predikátum olyan valami, ami minket általában a *Szülője* relációra emlékeztet. Korábban említettük, hogy új predikátumok bevezetése a célpredikátum definíciójának a méretét lényegesen csökkentheti. Az új predikátumok kialakításának beépített képességével, az inverz rezolúciós rendszer sokszor olyan tanulási problémákkal is megbirkózik, amelyekre más módszerek alkalmatlanok.

A tudomány legmélyebbre ható forradalmai új predikátumok és függvények bevezetésével függnek össze, ilyen például a Galilei felfedezte gyorsulás vagy Joule termikus energiája. Ha ezek a fogalmak már adottak, az új törvények felfedezése (viszonylag) egyszerű. A fő nehézség arra rájönni, hogy egy új entitás bevezetése, amely a létező entitásokkal meghatározott konkrét kapcsolatban van, lehetővé teszi a megfigyelések egész halmazának a korábban lehetségesnél sokkal egyszerűbb és elegánsabb elmélet keretein belül történő magyarázatát.

19.14. ábra - Egy új *P* predikátumot generáló inverz rezolúciós lépés



Az ILP-rendszerek egyelőre Galilei- vagy Joule-szintű felfedezéseket nem tettek, a felfedezéseit azonban tudományos irodalomban való publikálásra érdemesnek találták. Így például *Journal of Molecular Biology*-ban Turcotte írja le a fehérje összehajtogatási szabályok automatikus felfedezését a PROGOL ILP programmal (Turcotte és társai, 2001). A Progol által felfedezett szabályok közül sokat az ismert elvekből ugyan le lehetett volna következtetni, a többségét azonban a standard biológiai adatbázisok részeként korábban mégsem publikálták (lásd 19.10. ábra példája). Egy ehhez kapcsolódó kutatás részeként Srinivasan (Srinivasan és társai, 1994) molekula-struktúraalapú szabályok felfedezésével foglalkozott nitroaromatikus komponensek mutagenicitására. Ilyen komponensek a gépkocsik által kibocsátott kipufogógázban találhatók. A standard

Created by XMLmind XSL-FO Converter.

adatbázisokban lévő komponensek 80%-ában lehetséges a négy fontos jellemzőt azonosítani, és az erre épülő lineáris regresszió jobb az ILP-nél. A maradék 20% esetén a jellemzők önmagukban nem elegendők az előrejelzéshez, az ILP relációkat azonosít, amelyekkel a lineáris regresszió, a neurális hálók és a döntési fák képességein túltesz. King (King és társai, 1992) azt mutatta meg, hogy a különböző gyógyszerek gyógykezelési hatását hogyan lehetne előre jelezni. Ezen esetek mindegyikében úgy tűnik, hogy az ILP nagy hatékonyságához a relációk reprezentálási képessége és a háttértudás használata járulnak hozzá. Az a tény, hogy az ILP által megtalált szabályokat az emberek is képesek interpretálni, inkább segíti ezeknek a technológiáknak az elfogadását a biológiai folyóiratokban, mint a számítógépes tudománnyal foglalkozó folyóiratokban.

Az ILP a biológián túl más tudományokhoz is hozzájárult. A legfontosabbak egyike a természetes nyelvfeldolgozás, ahol az ILP-t komplex relációs információ szövegekből való kinyerésére alkalmazták. Ezeket az eredményeket a 23. fejezet foglalja össze.

6. Összefoglalás

Ebben a fejezetben különböző módszereket vizsgáltunk meg arra vonatkozóan, hogy az előzetes tudás hogyan segítheti az ágenst abban, hogy új tapasztalatokból tanuljon. Mivel az előzetes tudás zöme inkább a relációs modellekkel, és nem attribútumalapú modellekkel van megadva, a relációs modellekből tanuló rendszerekkel is foglalkoztunk. Az alábbiakban összefoglaljuk a fontos tudnivalókat.

- Az előzetes tudás felhasználása a tanulásnál a **kumulatív tanulás**hoz (**cumulative learning**) vezet, ahol az ágens az új tudás beszerzésével javítja a tanulási képességét.
- Az előzetes tudás segít a tanulásban, mert a különben konzisztens hipotéziseket eliminálja, és a példák magyarázatának „kitöltésével” rövidebb hipotézisekhez vezet. Ezek a tulajdonságok javítják a tanulás minta- és számítási komplexitását.
- Az előzetes tudás által betöltött különböző – **vonzatkényszer**ek (**entailment constraint**s) formájában kifejezett – logikai szerepeknek a megértése segít a különféle tanulási módszerek definiálásában.
- A **magyarázatalapú tanulás** (**MAT**) (**explanation-based learning**, **EBL**) az egyedi példákból úgy emeli ki az általános szabályokat, hogy a példákat először megmagyarázza, majd a magyarázatot általánosítja. Ezzel a deduktív módszerrel az elsődleges tudást hasznos, hatékony, speciális rendeltetésű szaktudássá változtatjuk.
- A **relevanciaalapú tanulás** (**RAT**) (**relevance-based learning**, **RBL**) az előzetes tudást meghatározások formájában használja a releváns attribútumok azonosítására. Ily módon egy redukált hipotézisteret generál, és a tanulási folyamatot meggyorsítja. A RAT lehetővé teszi az egyedi példák deduktív általánosítását is.
- A **tudásalapú induktív tanulás** (**TIT**) (**knowledge-based inductive learning**, **KBIL**) induktív hipotéziseket keres, amelyek a háttértudás segítségével megmagyarázzák a megfigyelések halmazát.
- Az **induktív logikai programozás**i (**ILP**) (**inductive logic programming**) technikák a TIT-hez folyamodnak, az elsőrendű logikában kifejezett háttértudást felhasználva. Az ILP-módszerek olyan relációs tudás megtanulására is alkalmasak, amelyet az attribútumalapú rendszerekben nem lehet kifejezni.
- Az ILP felülről lefelé megközelítéssel, a nagyon általános szabályok finomításával, vagy pedig lentről felfelé megközelítéssel, a deduktív folyamat invertálásával valósítható meg.
- Az ILP-módszerek természetes módon generálnak új predikátumokat is, amelyekkel új elméletek tömören kifejezhetők, és általános célú tudományos elméletformáló rendszerekként is ígéretesnek mutatkoznak.

6.1. Irodalmi és történeti megjegyzések

Bár az előzetes tudás felhasználása a tanulásban természetes témának tűnhet a tudományfilozófusok számára, mostanáig meglepően kevés formális eredmény született. Nelson Goodman filozófus a *Fact, Fiction, and Forecast* c. művében megcáfolta azt a korábbi feltételezést, miszerint az indukció nem más, mint egy univerzálisan kvantifikált kijelentés megfelelően sok példájának a megszemlélése és hipotézisként való elfogadása (Goodman, 1954).

Vegyük például azt a hipotézist, hogy „Minden smaragd zölék”, ahol a **zölék** azt jelenti, hogy „zöld, ha *t* idő előtt figyeljük meg, utána azonban kék”. A *t* idő előtt bármikor példák millióit láthatnánk, amelyek mind

Created by XMLmind XSL-FO Converter.

alátámasztják, hogy a smaragdok zölkék, negatív példák nélkül, mégis hezitálnánk a szabályt elfogadni. A jelenség csakis az indukciós folyamat szempontjából releváns a priori tudás szerepével magyarázható. Goodman a hasznosítható a priori tudás teljes választékát javasolja, a meghatározások **túlzott hipotézis**nek (**overhypothesis**) nevezett változatát is beleértve. Sajnos a gépi tanulással foglalkozó korai kutatásnál Goodman munkájára nem figyeltek fel.

A MAT gyökerei a tervkészítésnél használt STRIPS módszereihez nyúlnak vissza (Fikes és társai, 1972). Egy terv elkészítésével a terv egy általánosított változatát a tervek könyvtárában tárolták, és később **makróoperátor**ként (**macro-operator**) a tervkészítésnél felhasználták. Hasonló ötletek Anderson ACT* architektúrájában is megjelentek, **tudáskompilálás** (**knowledge compilation**) címen (Anderson, 1983) és SOAR architektúrájában **tudásdarabolás** (**chunking**) címen (Laird és társai, 1986). A **sémabeszerzés** (**schema acquisition**) (DeJong, 1981), az **analitikus általánosítás** (**analytical generalization**) (Mitchell, 1982) és a **kényszeralapú általánosítás** (**constraint-based generalization**) (Minton, 1984) a MAT tanulás irányában jelentkező és a (Mitchell és társai, 1986; DeJong és Mooney, 1986) publikációk stimulálta gyorsan növekvő érdeklődésnek közvetlen előfutárai voltak. A szövegben bemutatott MAT algoritmust Hirsh (Hirsh, 1987) vezette be, aki azt is megmutatta, hogy az algoritmust hogyan kell egy logikai programozási rendszerbe ágyazni. Van Harmelen és Bundy a MAT-ot a programelemző rendszerekben (Jones és társai, 1993) használt **részleges kiértékelő** (**partial evaluation**) módszer variánsaként magyarázzák (Van Harmelen és Bundy, 1988).

Az utóbbi időben a szigorú elemzés és a kísérleti kutatás a MAT jobb megértéséhez vezetett a problémamegoldás sebességében mért lehetséges költségeinek és előnyeinek terén. Minton azt mutatta meg, hogy hacsak tekintélyes munkát nem fektetünk bele, a MAT a programot lényegesen lelassíthatja (Minton, 1988). Tambe hasonló problémákkal küszködött a tudásdarabolásnál, és a szabálynyelv kifejezőerejének a mérséklését javasolta, hogy a szabályok munkamemóriához való illesztésének jelentős költségét minimálizáljuk (Tambe és társai, 1990). Erős a párhuzam ezen kutatás és az elsőrendű logika leszűkített változatának következtetési komplexitására vonatkozó jelenlegi eredményei között (lásd 9. fejezet). A MAT várható nyereségének formális valószínűségi elemzése a (Greiner, 1989; Subramanian és Feldman, 1990) munkákban található. Egy kiváló áttekintés Dietterichtől származik (Dietterich, 1990).

Ahelyett hogy a példákat mint az általánosítás fókuszpontjait tekintenénk, új problémák megoldására közvetlenül felhasználhatók az **analógiás következtetés** (**analogical reasoning**) elnevezésű megközelítésben. Az ilyen következtetésnek több változata is létezik. Idetartozik a hasonlóság mértékén alapuló plauzíbilis következtetés (Gentner, 1983) és a meghatározásokon alapuló deduktív következtetés (Davies és Russell, 1987), amely azonban megkívánja a példa jelenlétét ahhoz, hogy a belőle kialakított „lusta” MAT a régi példa általánosítását az új probléma szükségletei szerint irányítsa. Az analógiás következtetésnek ezt az utóbbi formáját általában meg lehet találni az **esetalapú következtetés**nél (**case-based reasoning**) (Kolodner, 1993) és a **származtatási analógiá**nál (**derivational analogy**) (Veloso és Carbonell, 1993).

A releváns információval funkcionális függőségek formájában először az adatbázis-kutatásnál foglalkoztak, ahol ez a nagy attribútumhalmazok kezelhető részhalmazokra való strukturálásának eszköze volt. Funkcionális függőségeket analógiás következtetésre Carbonell és Collins, majd logikai színezettel Bobrow és Raphael használtak (Carbonell és Collins, 1973; Bobrow és Raphael, 1974). Davies és Russell (Davies, 1985; Davies és Russell, 1987) a függőségeket másoktól függetlenül újra felfedezték, és az analógiás következtetés szempontjából teljes körű logikai elemzésnek vetették alá. A függőségeket deklaratív elfogultság céljából Russell és Grosof használták (Russell és Grosof, 1987). A meghatározások és a leszűkített szótárral rendelkező hipotézistér közötti ekvivalenciát Russell (Russell, 1988) bizonyította be. A meghatározásokat tanuló algoritmust és az RADFT megnövelt hatékonyságát először a FOCUS algoritmusban mutatták be (Almuallim és Dietterich, 1991). Tadepalli a meghatározások tanulására egy ügyes algoritmust közölt, amely a tanulási sebességben igen lényeges javulást mutat (Tadepalli, 1993).

Az a gondolat, hogy az induktív tanulás kivitelezhető inverz dedukcióval W. S. Jevonsig vezethető vissza (Jevons, 1874), aki azt írta, hogy: „Mind a formális logika, mind a valószínűség-elmélet tanulmányozása arra az álláspontra juttatott engem, hogy egy olyan dolog, mint egy külön indukciós módszer, a dedukcióval szembeállítva nincs, és az indukció egyszerűen a dedukció egy fordított alkalmazása.” A számítástudományi elemzések Gordon Plotkin figyelemre méltó PhD-disszertációjával kezdődtek Edinburghban (Plotkin, 1971). Bár Plotkin számos, a mai ILP területén használt tételt és módszert dolgozott ki, az indukció egyes részproblémáira vonatkozó bizonyos eldönthetetlenségi eredmények elvették a kedvét. A MIS (Shapiro, 1981) újból bevezette a logikai programok tanulási problémáját, azonban ezt főleg az automatikus hibakeresés elméletéhez való hozzájárulásának tekintették. A szabályindukció kutatása, vagyis az olyan rendszerek, mint az ID3 (Quinlan, 1986) és CN2 (Clark és Niblett, 1989) a FOIL-hoz vezettek (Quinlan, 1990), amely első ízben tette lehetővé relációs szabályok gyakorlati indukcióját. A területet Muggleton és Buntine pezsdítették fel

Created by XMLmind XSL-FO Converter.

(Muggleton és Buntine, 1988). CIGOL programjuk az inverz rezolúció nem egészen teljes változatát tartalmazta, és új predikátumok generálására is alkalmas volt.[5] A predikátum felfedezésével Wirth és O'Rorke is foglalkoztak (Wirth és O'Rorke, 1991). A következő nagyobb rendszer a GOLEM volt (Muggleton és Feng, 1990), amely a Plotkin-féle relatíve legkevésbé általános általánosítás ötletén alapuló lefedő algoritmust használja. A FOIL fentről lefelé módon, míg a CIGOL és a GOLEM lentről felfelé módon dolgoztak. E korszak más rendszerei az ITOU (Rouveirol és Puget, 1989) és a CLINT (De Raedt, 1992) rendszerek voltak. Újabban a PROGOL (Muggleton, 1995) az inverz vonzatreláció egy hibrid (fentről lefelé és lentről felfelé) megközelítését valósította meg, és a rendszert számos gyakorlati problémára alkalmazták, különösképpen a biológiában és a természetes nyelvfeldolgozásban. Muggleton a PROGOL egy kiterjesztését írja le (Muggleton, 2000), sztochasztikus logikai programok alakjában reprezentált bizonytalanság kezelésére.

Az ILP-módszerek formális elemzését (Muggleton, 1991) tartalmazza, míg a (Muggleton, 1992) egy nagy cikkgyűjtemény. A módszerek és az alkalmazások nagy gyűjteménye a (Lavrac és Dzeroski, 1994) könyv. A terület történetéről és a jövő kihívásairól újabb áttekintést ad (Page és Srinivasan, 2002). Haussler korai komplexitási eredményei azt sugallták, hogy az elsőrendű állítások tanulása reménytelenül bonyolult (Haussler, 1989). A klózokra vonatkozó különböző szintaktikai korlátozások jobb megértésével azonban pozitív eredmények is megjelentek, még a rekurziót tartalmazó klózokra is (Dzeroski és társai, 1992). Az ILP komplexitási eredményeiről Kietz és Dzeroski, valamint Cohen és Page írtak átfogó cikkeket (Kietz és Dzeroski, 1994; Cohen és Page, 1995).

Bár az ILP jelenleg a konstruktív indukció problémájának domináló megközelítése, nem csak ezzel próbálkoztak. Az ún. **felfedező rendszer**ek (**discovery system**s) célja az új koncepciók tudományos felfedezésének általában a koncepció definícióterében történő közvetlen keresés útján történő modellezése. Doug Lenat AM (Automated Mathematician) programja (Davis és Lenat, 1982) szakértő szabályok formájában megfogalmazott felfedező heurisztikákat használt arra, hogy elemi számelméleti koncepciók és sejtések keresését irányítsa. Éles ellentétben a matematikai következtetésre kifejlesztett rendszerekkel, az AM nélkülözte a bizonyítás fogalmát, és csak sejtés szinten tudott működni. A program újra felfedezte a Goldbach-sejtést és az egyértelmű prímtényezőkre bontást. Az AM architektúráját később az EURISKO rendszerben (Lenat, 1983) általánosították, a rendszert a saját felfedező heurisztikákat átírni képes szabályokkal bővítve. Az EURISKO-t más, a matematikai felfedezéstől eltérő tartományokban is alkalmazták, de az AM-nél kisebb sikerrel. Az AM és az EURISKO módszertana vitatott, lásd (Ritchie és Hanna, 1984; Lenat és Brown, 1984).

A felfedező rendszerek egy másik csoportja a valós fizikai adatokkal való munkát célozza és új törvények felfedezését kísérli meg. A DALTON, a GLAUBER és a STAHL (Langley és társai, 1987) olyan szabályalapú rendszerek, amelyek fizikai rendszerekből származó adatokban kvantitatív összefüggéseket keresnek. Mindegyik rendszer képes volt egy tudomány történetéből jól ismert felfedezés újbóli meghozatalára. A probabilisztikus technikákon – különösképpen az új kategóriákat felfedező klaszterező algoritmusokon – alapuló felfedező rendszerekkel a 20. fejezetben foglalkozunk.

6.2. Feladatok

19.1.

Mutassa ki a konjunktív normál formába való átalakításával és a rezolúció alkalmazásával, hogy a 4. szakasz - Tanulás releváns információ alapján részben található és a brazilokra vonatkozó konklúzió helyes.

19.2.

Az alábbi meghatározás mindegyikére írja fel a meghatározás logikai reprezentációját, és magyarázza meg, hogy a meghatározás miért igaz (ha igaz egyáltalán).

a. A postai kód meghatározza a megyét.

b. A külső kialakítás és a névleges érték meghatározzák az érme tömegét.

c. Adott program esetén a bemenetek meghatározzák a kimeneteket.

d. A klíma, a táplálkozás, a fizikai gyakorlat és a metabolizmus meghatározzák, hogy valaki fogyni vagy súlyban gyarapodni fog.

[5] Az inverz rezolúciós módszer (Russell, 1986)-ben is megjelenik, a teljes algoritmust egy lábjegyzetben közlik.

Created by XMLmind XSL-FO Converter.

e. A kopaszságot (vagy annak hiányát) az anyai ágon az egyik nagyszülőnek a kopaszsága határozza meg.

19.3.

Hasznos lenne a meghatározásoknak egy valószínűségi verziója? Javasoljon egy definíciót.

19.4.

Pótolja a C_1 és/vagy C_2 klózok értékét az alábbi klózhalmazban, figyelembe véve, hogy C a C_1 és a C_2 rezolvense:

(a) $C = Igaz \Rightarrow P(A, B)$, $C_1 = P(x, y) \Rightarrow Q(x, y)$, $C_2 = ??$

(b) $C = Igaz \Rightarrow P(A, B)$, $C_1 = ??$, $C_2 = ??$

(c) $C = P(x, y) \Rightarrow P(x, f(y))$, $C_1 = ??$, $C_2 = ??$

Ha több megoldás is lehetséges, a különböző eseteket külön példákkal illusztrálja.

19.5.

Tegyük fel, hogy egy rezolúciós lépést végrehajtó logikai programot írunk. Legyen tehát a *Rezolválás*(c_1, c_2, c) sikeres, ha a c a c_1 és a c_2 rezolválásának az eredménye. Rendes körülmények között a *Rezolválás*-t a tételbizonyítás részeként fogjuk használni úgy, hogy meghívjuk a c rezolvens generálására, miközben a c_1 és a c_2 változóba konkrét klózértékeket helyettesítünk be. Tegyük most fel, hogy az eljárást a behelyettesített c-vel és a szabad c_1-gyel, valamint c_2-vel hívjuk meg. Megszületik-e az inverz rezolúciós lépés helyes eredménye? Szükséges-e a logikai program valamilyen speciális átalakítása, hogy az inverz lépés helyesen működjön?

19.6.

Tegyük fel, hogy a `FOIL` program egy klózhoz egy literált úgy add hozzá, hogy egy bináris P predikátumot használ, és hogy az előbbi literálok (a klóz fejrészét beleértve) öt különböző változót tartalmaznak.

a. Hány funkcionálisan különböző literált lehet generálni? A két literál funkcionálisan egybeesik, ha csupán az általuk tartalmazott új változók neveiben különböznek.

b. Meg tudna adni általános képletet a különböző, r argumentummal rendelkező predikátumot használó literálok számának meghatározására, ha előzőleg n változót használtunk?

c. A `FOIL` miért nem enged olyan literálokat használni, amelyek az előzőleg használt változókat nem tartalmazzák?

19.7.

A 19.11. ábrán látható családfa adatainak vagy annak részhalmazának felhasználásával alkalmazza az *Őse* predikátum megtanulására a `FOIL` algoritmust.

Created by XMLmind XSL-FO Converter.

20. fejezet - Statisztikai tanulási módszerek

Ebben a fejezetben megfigyelések alapján történő bizonytalan következtetésnek tekintjük a tanulást.

Az V. részben rámutattunk, hogy a valós életre jellemző környezetben gyakran előfordul, hogy a tudás bizonytalan. Az ágensek a bizonytalanságot valószínűség- és döntéselméleti módszerekkel tudják kezelni, de ehhez először tapasztalataik alapján fel kell állítaniuk a világra vonatkozó valószínűségi modelljüket. Ez a fejezet bemutatja, hogy milyen módon tudják ezt megtenni. Látni fogjuk, hogyan kell úgy formalizálni a tanulási feladatot, mint egy valószínűségi következtetési folyamatot (lásd 20.1. alfejezet). Bemutatjuk, hogy a tanulás bayesi megközelítése rendkívül hatékony: általános megoldást ad a zaj, a túlilleszkedés és az optimális predikció problémáira. Továbbá figyelembe veszi azt a tényt, hogy a nem-egészen-mindentudó ágens soha nem tudhatja biztosan, hogy a világról alkotott melyik elmélet helyes, mégis döntéseket kell hoznia.

A 20.2. és 20.3. alfejezetben valószínűségi modellek – elsősorban a Bayes-hálók – tanulási módszereit ismertetjük. A 20.4. alfejezet olyan módszereket tárgyal, amelyek speciális minták tárolását és előhívását végzik. A 20.5. alfejezet a **neurális háló** (**neural network**) tanítással foglalkozik, míg a 20.6. alfejezet a **kernelgépek**kel (**kernel machine**). Ennek a fejezetnek egy része erősebben matematikai megközelítésű (feltételezi a többváltozós analízis alapjainak ismeretét), bár az általános tanulságok megérthetők anélkül is, hogy elmerülnénk a matematikai részletekben. Hasznos lehet, ha az olvasó ezen a ponton először átnézi a 13. és a 14. fejezeteket, valamint az A) függeléket.

1. Statisztikai tanulás

Az ebben a fejezetben használt alapvető koncepciók ismét az **adat** (**data**) és a **hipotézis** (**hypothesis**), **éppúgy**, mint ahogy a 18. fejezetben volt. Itt az adatok a **tény**ek (**evidence**) – ezek a területet leíró valószínűségi változók egy részének vagy mindegyikének egy konkrét megvalósulását jelentik. A hipotézis valamilyen valószínűségi elmélet arról, hogy a világ adott területe hogyan is működik, ennek speciális esetei a logikai elmélettel leírt területek.

Vegyünk egy *nagyon* egyszerű példát. Kedvenc „meglepetés” cukorkánk kétféle, meggy- (nyam-nyam) és citrom- (brrrr) ízben kapható. A cukorkagyártónak sajátos humora van, és mindegyik cukorkát – ízétől függetlenül – ugyanolyan átlátszatlan papírba csomagolja. Az édességet nagyon nagy zsákokban árulják, amelyekből ötféle van – kívülről megint csak megkülönböztethetetlenek egymástól:

h_1: 100% meggy

h_2: 75% meggy + 25% citrom

h_3: 50% meggy + 50% citrom

h_4: 25% meggy + 75% citrom

h_5: 100% citrom

Egy adott új zsák cukorka esetén a H (*hipotézis*t jelölő) véletlen változó a zsák típusát jelenti, lehetséges értékei h_1-től h_5-ig terjednek. Természetesen H közvetlenül nem figyelhető meg. Ahogy a cukorkákat felbontjuk és megvizsgáljuk, adatokat gyűjtünk – amelyeket D_1, D_2, …, D_N jelöl. Mindegyik D_i egy olyan véletlen változó, amelynek lehetséges értékei a *meggy*, illetve a *citrom*. Az ágens feladata, hogy jóslást adjon a következő cukorka ízére.[1] Ez a nyilvánvalóan triviális feladat érdekes módon mégis jó betekintést nyújt a legfontosabb problémák közül többre is. Az ágensnek valóban elméletet kell alkotnia a világról, bár csak nagyon egyszerűt.

A **Bayes-tanulás** (**Bayesian learning**) során egyszerűen kiszámítjuk minden egyes hipotézis valószínűségét az adatokra támaszkodva, majd ennek alapján adunk predikciót. Azaz nem egyetlen „legjobb” hipotézist használunk a predikcióhoz, hanem az összes hipotézist használjuk, valószínűségükkel súlyozva őket.

[1] A statisztikában jártasabb olvasó felismeri, hogy ez a példa valójában az **urna és golyó** (**urn and ball**) feladat változata. Úgy találtuk, hogy az urna és a golyó kevesebb kihívást jelent, mint a cukorka, továbbá a cukorka példa elvezet egy másik feladathoz – elcseréljük-e a zacskót egy barátunkkal, vagy sem (lásd 20.3. feladat).

Created by XMLmind XSL-FO Converter.

Reprezentálja **D** az összes adatot, legyen **d** a megfigyelt értékek vektora, ekkor az egyes hipotézisek valószínűségét a Bayes-szabállyal adhatjuk meg:

$$P(h_i|d) = \alpha P(d|h_i)P(h_i) \quad (20.1)$$

Tegyük fel, hogy egy ismeretlen X mennyiségre vonatkozó predikció a célunk. Ebben az esetben:

$$\mathbf{P}(X|\mathbf{d}) = \sum_i \mathbf{P}(X|\mathbf{d},h_i)\mathbf{P}(h_i|\mathbf{d}) = \sum_i \mathbf{P}(X|h_i)\mathbf{P}(h_i|\mathbf{d}) \quad (20.2)$$

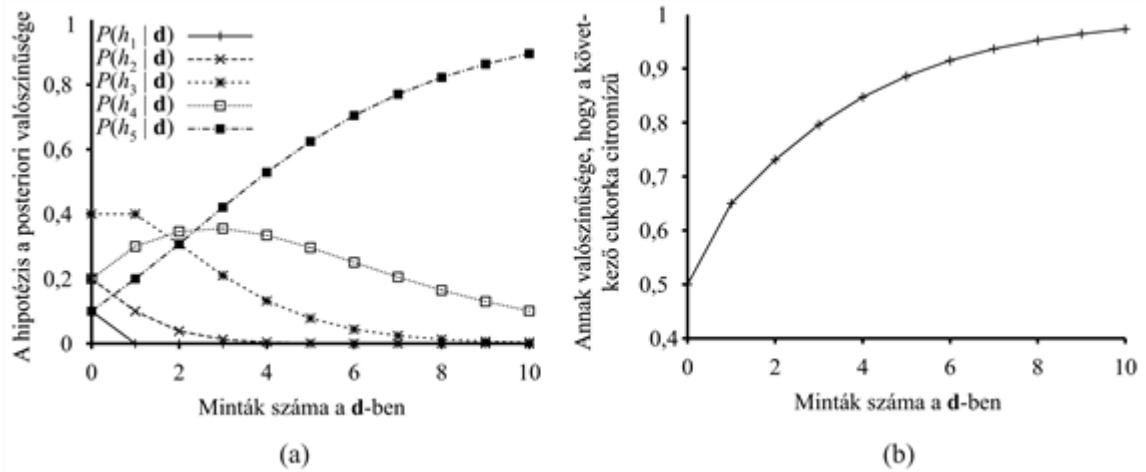
ahol azt feltételeztük, hogy az összes hipotézis meghatároz X-re valamilyen eloszlást. Ez az összefüggés azt mutatja, hogy a predikció az egyes hipotézisekből adódó predikciók súlyozott összege. Maguk a hipotézisek tulajdonképpen a nyers adatok és a predikciók közti „közvetítők". A Bayes-megközelítésben a $P(h_i)$ **prior hipotézis**ek (**hypothesis prior**), illetve a hipotézisek mellett fellépő $P(\mathbf{d}|h_i)$ adat**valószínűség**ek (**likelihood**) a kulcsmennyiségek.

A cukorka példánkban azt feltételezzük, hogy a $h_1,\ldots, h_5$ a priori valószínűségei megfelelnek a gyártó reklámjában közölt ⟨ 0,1, 0,2, 0,4, 0,2, 0,1⟩ értékeknek. Az adatok valószínűségét **e.f.e** (angol rövidítése: **i.i.d.**) feltételezéssel számítjuk – azaz egyforma és független eloszlást (independently and identically distributed) teszünk fel, így:

$$P(\mathbf{d}|h_i) = \prod_j P(d_j|h_i) \quad (20.3)$$

Tegyük fel például, hogy a cukorkás zsák valójában csupa citromtípusú (h_5), és az első 10 kibontott cukorka mind citromízű, ekkor $P(\mathbf{d}|h_3) = 0{,}5^{10}$, mivel a h_3 típusú zsákokban a cukorkák fele citrom.[2] A 20.1. (a) ábra mutatja, hogyan változik az öt hipotézis a posteriori valószínűsége, ahogy sorban észleljük a 10 citromízű cukorkát. Vegyük észre, hogy a valószínűségek az a priori értékekről indulnak, ennek megfelelően kezdetben h_3 a legvalószínűbb lehetőség, és ez így is marad még az első cukor felbontása után is. A második citromízű cukorka felbontása után h_4 a legvalószínűbb, 3 és több esetén h_5 (a rettegett csupa citrom zsák). 10 citromízű cukorka után már meglehetősen biztosak vagyunk végzetünket illetően. A 20.1. (b) ábra mutatja annak a (20.2) egyenlet alapján jósolt valószínűségét, hogy a következő cukorka citromízű. Várakozásunknak megfelelően monoton növekszik az 1 felé.

20.1. ábra - (a) A (20.1) egyenletből számított $P(h_i|d_1,\ldots,d_N)$ a posteriori valószínűségek. A megfigyelések száma 1-től 10-ig terjed, és mindegyik megfigyelés citromízű cukorka. (b) A (20.2) egyenlet alapján számított $P(d_{N+1} = citrom|d_1,\ldots,d_N)$ Bayes-predikció.



Fontos

[2] Korábbiakban leszögeztük, hogy a cukorkás zsákok nagyon nagyok, másképp az e.f.e. feltétel nem áll fenn. Korrektebb lenne (de kevésbé higiénikus) feltételezni, hogy minden vizsgálat, kóstolás után visszacsomagoljuk a cukorkát, és visszatesszük a zsákba.

Created by XMLmind XSL-FO Converter.

Példánk azt mutatja, hogy *a Bayes-predikcióban az igaz hipotézis végülis dominánssá válik*. Ez jellemző a Bayes-tanulásra. Bármilyen rögzített priorra, amely nem zárja ki a helyes megoldást, a hamis hipotézisek a posteriori valószínűsége végülis nullához tart. Ennek egyszerűen az az oka, hogy elhanyagolhatóan kicsi annak valószínűsége, hogy végtelen ideig „nem jellemző" adatokat generáljunk. (Ezen a ponton érvelésünk hasonló a 18. fejezetben, a VKH-tanulásnál alkalmazottal.) Ennél is fontosabb, hogy a Bayes-predikció *optimális,* akár kicsi, akár nagy adathalmazunk van. Adott a priori hipotéziseloszlás mellett bármely más predikció ritkábban lesz helyes, mint a Bayes-predikció.

A Bayes-tanulás optimalitásának természetesen ára van. Mint a 18. fejezetben láttuk, a valós tanulási problémáknál a hipotézistér rendszerint nagyon nagy vagy végtelen. Néha a (20.2) egyenletben az összegzés (vagy folytonos esetben az integrálás) pontos elvégzése kezelhető problémára vezet, de a legtöbb esetben közelítő vagy egyszerűsített megoldásokra kell szorítkoznunk.

Nagyon elterjedt approximációs módszer – a tudományos feladatokban rendszerint ezt alkalmazzuk –, hogy egyetlen, a *legvalószínűbb* hipotézis alapján végezzük a predikciót, azaz olyan h_i alapján, amely maximálja a $P(h_i|\mathbf{d})$-t. Ezt **maximum a posteriori** vagy **MAP** hipotézisnek nevezzük. A MAP hipotézis alapján végzett predikciók közelítőleg Bayes-predikciók. Ez a közelítés annyira jó, amennyire jó a $\mathbf{P}(X|\mathbf{d}) \approx \mathbf{P}(\mathbf{X}|h_{\text{MAP}}$ közelítés. A cukorka példánkban három egymás utáni citromízű cukor észlelése után $h_{\text{MAP}} = h_5$, így a MAP-tanulás alapján a negyedik cukorra 1,0 valószínűséggel citromízűt jósolunk. Ez nyilván sokkal veszélyesebb jóslat, mint a Bayes-predikció, amely 0,8valószínűségű, ahogy a 20.1. ábrán is láthatjuk. Ahogy egyre több adatunk van, a MAP és Bayes-predikciók egyre inkább konvergálnak egymáshoz, mivel a MAP hipotézis alternatívái egyre kevésbé valószínűvé válnak. Bár példánk ezt nem mutatja, de a MAP hipotézis előállítása sokszor lényegesen egyszerűbb, mint a Bayes-tanulásé. Ennek oka, hogy csupán egy optimalizálási probléma megoldását igényli, szemben egy nagyon nagy összegzési (vagy integrálási) problémával. A fejezet későbbi részében majd látunk példákat erre.

Mind a Bayes-tanulásban, mind a MAP-tanulásban a $P(h_i)$ a priori hipotézis valószínűségek nagy szerepet játszanak. Láttuk a 18. fejezetben, hogy **túlilleszkedés** (**overfitting**) léphet fel, ha túlzottan nagy a hipotézistér kifejezőképessége, vagyis túl sok olyan hipotézist tartalmaz, amely jól illeszkedik az adatokra. A Bayes- és a MAP-tanulás nem alkalmaz valamilyen önkényes korlátot a figyelembe vett hipotézisekre, inkább az a priori valószínűségeket használják fel arra, hogy *büntessék a hipotézisek komplexitását*. A komplex hipotéziseknek tipikusan kisebb az a priori valószínűsége – részben azért, mert sokkal több komplex hipotézis van, mint egyszerű. Másrészről a komplex hipotéziseknek nagyobb kapacitása van az adatokra való illeszkedéshez. (Extrém esetet véve egy táblázat 1,0 valószínűséggel és tökéletes pontossággal reprodukálni tudja az adatokat.) Ennek megfelelően az a priori valószínűségben testesül meg a hipotéziskomplexitás és az adatokra való illeszkedési képesség közötti kompromisszum.

Fontos

Ezt a kompromisszumot a logikai esetben figyelhetjük meg legjobban, amikor is H csak *determinisztikus* hipotéziseket tartalmaz. Ebben az esetben $\mathbf{P}(\mathbf{d}|h_i)$ értéke akkor 1, ha h_i konzisztens az adatokkal, különben 0. A (20.1) egyenlet alapján azt látjuk, hogy h_{MAP} a *legegyszerűbb logikai hipotézis* lesz, *amely konzisztens az adatokkal*. Tehát a maximum a posteriori tanulás Ockham borotvájának egy természetes megvalósulása.

A hipotéziskomplexitás és az adatokra való illeszkedési képesség közötti kompromisszumot új módon világítja meg, ha a (20.1) egyenlet logaritmusát képezzük. A h_{MAP} hipotézis olyan kiválasztása, amely maximálja a $\mathbf{P}(\mathbf{d}|h_i)P(h_i)$-t ugyanaz, mint amikor minimalizáljuk a következő kifejezést:

$$-\log_2 P(\mathbf{d}|h_i) - \log_2 P(h_i)$$

Használjuk fel az információkódolás és a valószínűség között a 18. fejezetben bevezetett kapcsolatot. Azt látjuk, hogy a $-\log_2 P(h_i)$ tag nem más, mint a h_i hipotézis specifikálásához szükséges bitek száma. A $-\log_2 P(\mathbf{d}|h_i)$-tag viszont azoknak a további biteknek a száma, amelyek ahhoz szükségesek, hogy az adott hipotézis feltételezésével specifikáljuk az adatokat. (Ennek demonstrálására mutatjuk be azt az esetet, amikor a hipotézis pontosan megjósolja az adatokat, ilyenkor nincs szükség egyetlen bitre sem az adatok specifikálásához. Ilyen például a h_5 hipotézis esete, amikor sorban érkeznek a citromízű cukorkák – és valóban $\log_2 1 = 0$.) Ebben az értelemben a MAP-tanulás jellemzője, hogy maximálisan *tömöríti* az adatokat. Ezt a feladatot sokkal közvetlenebbül célozza a **minimális hosszúságú leírás** (**MHL**) (**minimum description length**, **MDL**) tanulási

Created by XMLmind XSL-FO Converter.

módszer, amely a valószínűségekkel való foglalkozás helyett a hipotézis méretének és az adat kódolásának minimalizálására törekszik.

Az utolsó egyszerűsítést az adja, ha a hipotézistérben **egyenletes** (**uniform**) priort feltételezünk. Ebben az esetben a MAP-tanulás egy olyan h_i választására redukálódik, amely maximálja $P(\mathbf{d}|h_i)$ -t. Ezt **maximum-likelihood** (**ML**) hipotézisnek nevezzük, és h_{ML}-lel jelöljük. A maximum-likelihood tanulás nagyon elterjedt a statisztikában. Ez egy olyan tudomány, amelynek sok kutatója nem bízik az a priori hipotézisek szubjektív természetében. Ez józan megközelítés akkor, amikor nincs semmi okunk, hogy a priori kitüntessük az egyik hipotézist egy másikkal szemben, például amikor az összes hipotézis egyformán komplex. Az ML-tanulás jó közelítését adja a Bayes- és MAP-tanulásnak olyankor, amikor az adathalmaz nagy, hiszen az adatok végül is felülírják a hipotézisek a priori eloszlását, de kis adathalmazok esetén problémák merülnek fel az alkalmazásánál (mint látni fogjuk).

2. Teljes adattal történő tanulás

A statisztikai tanulási módszerek tárgyalását a legegyszerűbb feladattal kezdjük: **paramétertanulás** (**parameter learning**) **teljes adat** (**complete data**) alapján. A paramétertanulás egy rögzített struktúrájú valószínűségi modell paramétereinek megtalálását foglalja magában. Például meg akarjuk tanulni egy adott struktúrájú Bayes-háló feltételes valószínűségeit. Az adatokat akkor nevezzük teljesnek, ha mindegyik adatpont értékeket hordoz a megtanulandó valószínűségi modell mindegyik paraméterére. A teljes adatok nagyban egyszerűsítik a komplex modellek paramétereinek tanulását. Nagy vonalakban áttekintjük majd a struktúratanulás problematikáját is.

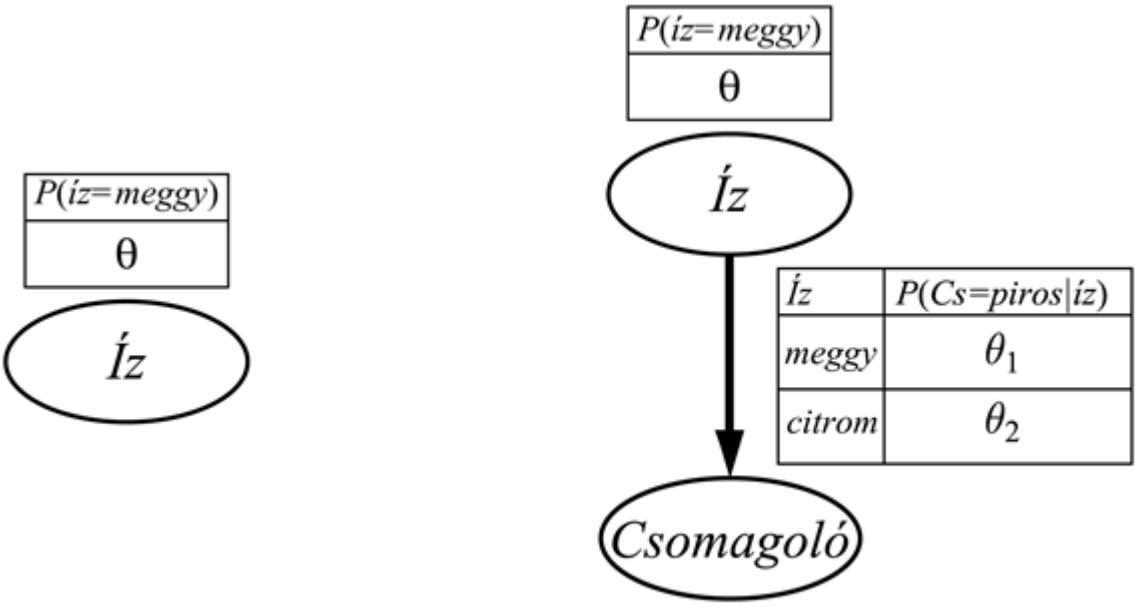
2.1. Maximum-likelihood paramétertanulás: diszkrét modellek

Tegyük fel, hogy új gyártótól vásárolunk egy zsák citrom- és meggycukorkát, a meggyarány teljesen ismeretlen, bárhol lehet 0 és 1 között. Ez esetben kontinuum számosságú hipotézisünk van. A **paraméter** (**parameter**), amelyet θ-val jelölünk, most a meggycukorkák aránya, a hipotézis pedig h_θ. (A citromízűek aránya egyszerűen $1 - \theta$.) Ha feltételezzük, hogy a priori minden arány egyformán valószínű, akkor a maximum-likelihood megközelítés az ésszerű. Ha Bayes-hálóval modellezzük a helyzetet, akkor csupán egyetlen véletlen változóra van szükségünk. Legyen ez az *Íz* nevű változó (a zacskóból véletlenszerűen választott cukorka íze). Lehetséges értékei a *meggy* és a *citrom*, ahol a *meggy* valószínűsége θ (lásd 20.2. (a) ábra). Tegyük fel, hogy kibontunk N cukorkát, amelyek közül c meggyízű és $\ell = N - c$ citromízű. A (20.3) egyenlet alapján ennek a speciális adathalmaznak a valószínűsége:

$$P(\mathbf{d}|h_\theta) = \prod_{j=1}^{N} P(d_j|h_\theta) = \theta^c \cdot (1-\theta)^\ell$$

20.2. ábra - (a) Bayes-háló modell az ismeretlen arányban citrom-, illetve meggyízű cukorkák esetére. (b) Annak a modellje, amikor a csomagolópapír színe függ (valószínűségi alapon) a cukorka ízétől.

Created by XMLmind XSL-FO Converter.



A maximum-likelihood hipotézist az a θ érték adja, amely maximálja ezt a kifejezést. Ugyanezt az értéket kapjuk, ha a **log likelihood** függvényt maximáljuk.

$$L(\mathbf{d}|h_\theta) = \log P(\mathbf{d}|h_\theta) = \sum_{j=1}^{N} \log P(d_j|h_\theta) = c\log\theta + \ell\log(1-\theta)$$

(A kifejezés logaritmusát képezve a szorzatot szummává redukáltuk, amit rendszerint egyszerűbb maximálni.) A maximum-likelihood θ érték megtalálása érdekében differenciáljuk L-et θ szerint, a kapott kifejezést pedig tegyük egyenlővé nullával:

$$\frac{dL(\mathbf{d}|h_\theta)}{d\theta} = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 \quad \Rightarrow \quad \theta = \frac{c}{c+\ell} = \frac{c}{N}$$

Magyarán a h_{ML} maximum-likelihood hipotézis azt állítja, hogy a zacskóban a meggyízű cukorkák valós aránya megegyezik az eddig kibontott cukorkáknál megfigyelt aránnyal!

Úgy tűnik, rengeteget dolgoztunk, hogy felfedezzünk egy nyilvánvaló eredményt. Valójában lefektettünk egy standard módszert a maximum-likelihood paramétertanulásra:

1. Írjunk fel egy – a paraméter(ek)től függő – kifejezést az adatok együttes valószínűségére (írjuk fel a likelihood függvényt).

2. Írjuk fel minden egyes paraméter szerint a log likelihood függvény deriváltját.

3. Keressük meg azokat a paraméterértékeket, amelyek mellett a deriváltak nulla értéket vesznek fel.

Fontos

A legtrükkösebb lépés általában az utolsó. Az előző példánkban triviálisan megoldható volt, de látni fogjuk, hogy sokszor iteratív megoldásokhoz vagy más numerikus optimalizálási technikákhoz kell folyamodnunk, mint ahogy a 4. fejezetben tárgyaltuk. A példa a maximum-likelihood tanulás egy általános problémáját is illusztrálja: *ha az adathalmaz elég kicsi ahhoz, hogy néhány eseményt még nem figyeltünk meg – például nem találtunk még meggyízű cukorkát –, akkor a maximum-likelihood hipotézis nulla valószínűséget rendel ezekhez az eseményekhez.* Számos trükköt használnak, hogy elkerüljék ezt a problémát, mint például minden esemény kezdeti valószínűségét 1-re állítják nulla helyett.

Created by XMLmind XSL-FO Converter.

Nézzünk egy másik példát. Tegyük fel, hogy a cukorka gyártója némi információt akar adni a fogyasztónak, ezért piros és zöld csomagolópapírt használ. A *Csomagoló* mindegyik cukorkához véletlenszerűen kerül kiválasztásra, valamilyen ismeretlen – az íztől függő – feltételes valószínűség-eloszlás szerint. Az ehhez tartozó valószínűségi modellt a 20.2. (b) ábra mutatja. Vegyük észre, hogy három paramétere van: θ, θ_1 és θ_2. Ezekkel a parméterekkel a Bayes-hálók standard szemantikáját használva megadható annak a valószínűsége, hogy – mondjuk – egy meggyízű cukorkát találunk egy zöld csomagolóban (2.1. szakasz - Az együttes valószínűség-eloszlás függvény leírása rész):

$$P(\acute{I}z = meggy,\ Csomagoló = zöld \mid h_{\theta,\theta_1,\theta_2})$$
$$= P(\acute{I}z = meggy \mid h_{\theta,\theta_1,\theta_2}) P(Csomagoló = zöld \mid \acute{I}z = meggy, h_{\theta,\theta_1,\theta_2})$$
$$= \theta(1-\theta_1)$$

Most kibontunk N cukorkát, amelyek közül c meggyízű és ℓ citromízű. A csomagolásfajták számai a következők: r_c meggyízű volt pirosba csomagolva és g_c zöldbe, míg r_ℓ citromízű volt pirosba csomagolva, míg g_ℓ zöldbe. Ezen adatok együttes valószínűsége:

$$P\left(\mathbf{d} \middle| h_{\theta,\theta_1,\theta_2}\right) = \theta^c (1-\theta)^\ell \theta_1^{r_c} (1-\theta_1)^{g_c} \theta_2^{r_\ell} (1-\theta_2)^{g_\ell}$$

Ez meglehetősen ijesztőnek tűnik, de segít, ha a logaritmusát vesszük:

$$L = [c\log\theta + \ell\log(1-\theta)] + [r_c \log\theta_1 + g_c \log(1-\theta_1)] + [r_\ell \log\theta_2 + g_\ell \log(1-\theta_2)]$$

A logaritmusképzés előnye nyilvánvaló: a log likelihood függvény három tag összege, ahol mindegyik tag csupán egyetlen paramétert tartalmaz. Amikor sorban mindegyik paraméter szerint vesszük az összefüggés deriváltját, majd nullává tesszük a deriváltakat, három független egyenlethez jutunk, és mindegyik csupán egyetlen paramétert tartalmaz:

$$\frac{\partial L}{\partial \theta} = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 \quad \Rightarrow \quad \theta = \frac{c}{c+\ell}$$
$$\frac{\partial L}{\partial \theta_1} = \frac{r_c}{\theta_1} - \frac{g_c}{1-\theta_1} = 0 \quad \Rightarrow \quad \theta_1 = \frac{r_c}{r_c + g_c}$$
$$\frac{\partial L}{\partial \theta_2} = \frac{r_\ell}{\theta_2} - \frac{g_\ell}{1-\theta_2} = 0 \quad \Rightarrow \quad \theta_2 = \frac{r_\ell}{r_\ell + g_\ell}$$

A θ-ra kapott megoldás ugyanaz, mint az előbb. A θ_1-re kapott megoldás, tehát annak valószínűsége, hogy egy meggyízű cukorka piros papírba csomagolt, nem más, mint a megfigyelt meggyízű cukorka – piros papír arány, hasonló a helyzet θ_2-vel.

Fontos

Ezek az eredmények nagyon kényelmesek, és könnyen belátható, hogy kiterjeszthetők bármely Bayes-hálóra, amelynek feltételes valószínűségeit táblázatokkal adjuk meg. A legfontosabb eredmény a következő: *teljes adatok esetén a Bayes-háló paramétertanulási problémája elkülönülő tanulási problémákra dekomponálható, egy-egy probléma egy-egy paraméterre.*[3] A második eredmény, hogy az egyes paraméterek adott szülő melletti értékei éppen a szülőértékek mellett megfigyelt gyakoriságokkal egyeznek meg. Éppúgy, mint az előző helyzetben, itt is óvatosnak kell lennünk, hogy kis adathalmazok esetén el tudjuk kerülni a nulla értékeket.

2.2. Naiv Bayes-modellek

Valószínűleg a gépi tanulás területén használt legelterjedtebb Bayes-háló modell a **naiv Bayes-modell** (**naive Bayes**). Ebben a modellben a C „osztályváltozó” (amelyet meg akarunk jósolni) a gyökér, míg az X_i attribútumváltozók a levelek. A modell azért „naiv”, mert feltételezi, hogy adott osztály mellett az attribútumok feltételesen függetlenek egymástól. (A 20.2. (b) ábra modellje egy egyváltozós naiv Bayes-modell.) Logikai változókat feltételezve a paraméterek:

[3] Nem táblázatos formára lásd a 20.7. feladatot, amelyben mindegyik paraméter hatással van számos feltételes valószínűségre.

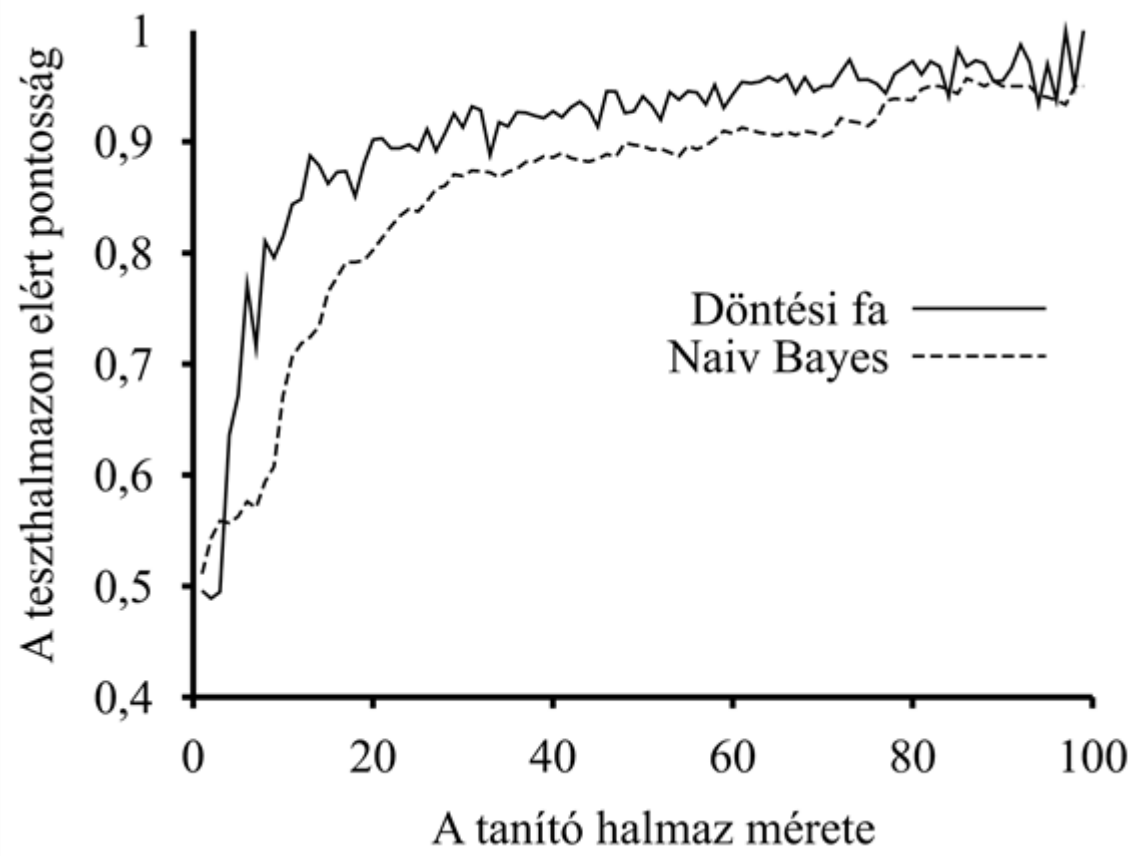
Created by XMLmind XSL-FO Converter.

$\theta = P(C = igaz)$, $\theta_{i_1} = P(X_i = igaz | C = igaz)$, $\theta_{i_2} = P(X_i = igaz | C = hamis)$

A maximum-likelihood paramétereket pontosan úgy kapjuk meg, mint a 20.2. (b) ábra esetén. Ha a modellt ezen az úton megtanítottuk, akkor felhasználható arra, hogy olyan új példákat osztályozzon, amelyekre a C osztályváltozó nem ismert. A megfigyelt $x_1, ..., x_n$ attribútumértékek mellett az egyes osztályok valószínűségét a következő összefüggés adja:

$$\mathbf{P}(C|x_1,\ldots,x_n) = \alpha \mathbf{P}(C) \prod_i \mathbf{P}(x_i|C)$$

20.3. ábra - A 18. fejezet étterem problémájára alkalmazott naiv Bayes-tanulás tanulási görbéje. A döntési fa tanulás tanulási görbéjét összehasonlítás céljából ábrázoltuk.



Fontos

Ha a legvalószínűbb osztályt választjuk, determinisztikus predikció adható. A 20.3. ábra mutatja a módszer tanulási görbéjét, ha a 18. fejezet étterem problémájára alkalmazzuk. A módszer elég jól tanul, de nem olyan jól, mint egy döntési fa tanulás. Ennek oka valószínűleg az, hogy a helyes hipotézis – ami egy döntési fa – nem reprezentálható pontosan naiv Bayes-modellel. A naiv Bayes-tanulás sok alkalmazási területen meglepően jól teljesít, a fokozott teljesítményű, turbó változata *(boosted version)* egyike a leghatékonyabb általános célú tanuló algoritmusoknak (lásd 20.5. feladat). A módszer nagyon nagy méretű problémákhoz is jól alkalmazható, n logikai változó esetén is csak $2n + 1$ paramétere lesz, és *ahhoz, hogy a naiv Bayes-tanulás meghatározza h_{ML}-t, nincs szükség keresésre*. Végül a naiv Bayes-tanulásnak nem jelentenek gondot a zajos adatok, továbbá ha szükséges, akkor képes valószínűségi predikciókat is adni.

2.3. Maximum-likelihood paramétertanulás: folytonos eset

A 14.3. alfejezetben vezettük be a folytonos valószínűségi modelleket, mint például a **lineáris Gauss-** (**linear-Gaussian**) modelleket. Mivel a valós alkalmazásokban mindenütt folytonos változókkal találkozunk, fontos

Created by XMLmind XSL-FO Converter.

ismernünk a folytonos modellek adatokból történő megtanulásának módszereit. A maximum-likelihood tanulás elvei azonosak a diszkrét esetre vonatkozó tanulás elveivel.

Kezdjük egy rendkívül egyszerű esettel: egyetlen skalár változó Gauss-sűrűségfüggvényének a paramétereit tanuljuk. Azaz az adatokat a következő összefüggéssel generáljuk:

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

A modell paraméterei a μ átlag és a σ szórás. (Vegyük észre, hogy a normalizáló konstans is függ σ -tól, ezért nem hanyagolhatjuk el.) Legyenek a megfigyelt értékek $x_1,...,x_N$. Ekkor a log likelihood:

$$L = \sum_{j=1}^{N} \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_j-\mu)^2}{2\sigma^2}} = N\left(-\log\sqrt{2\pi} - \log\sigma\right) - \sum_{j=1}^{N} \frac{(x_j-\mu)^2}{2\sigma^2}$$

A deriváltakat szokásos módon nullává téve a következőket kapjuk:

$$\frac{\partial L}{\partial \mu} = -\frac{1}{\sigma^2}\sum_{j=1}^{N}(x_j-\mu) = 0 \quad \Rightarrow \quad \mu = \frac{\sum_j x_j}{N}$$

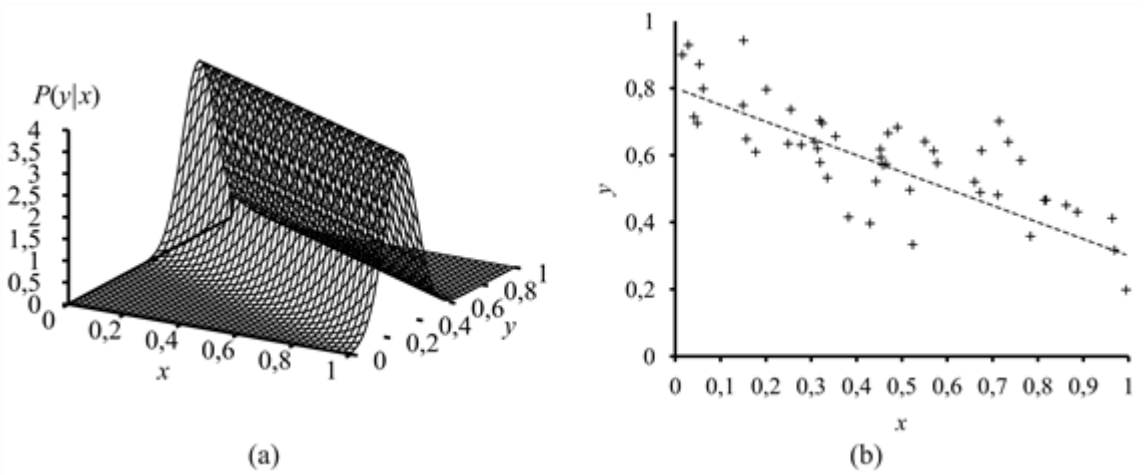
$$\frac{\partial L}{\partial \sigma} = -\frac{N}{\sigma} + \frac{1}{\sigma^3}\sum_{j=1}^{N}(x_j-\mu)^2 = 0 \Rightarrow \quad \sigma = \sqrt{\frac{\sum_j (x_j-\mu)^2}{N}} \qquad (20.4)$$

Tehát az átlag maximum-likelihood becslése a mintaátlag, a szórás maximum-likelihood becslése pedig a minta átlagos szórásnégyzetének négyzetgyöke. Ezek ismét kedvező eredmények, mivel megerősítik a „józan ésszel” követett gyakorlatot.

Vizsgáljunk most egy lineáris Gauss-modellt, amelyben egy X folytonos szülő és Y folytonos gyermek van. Mint az 3.1. szakasz - Bayes-hálók folytonos változókkal részben megmutattuk, Y Gauss-eloszlású, átlaga lineárisan függ X-től, míg varianciája rögzített. A $P(X|Y)$ feltételes eloszlás tanulásához maximalizálhatjuk a feltételes likelihood függvényt:

$$P(y|x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y-(\theta_1 x+\theta_2)^2}{2\sigma^2}} \qquad (20.5)$$

20.4. ábra - (a) Egy $y = (\theta_1 + \theta_j + \theta_2)$ egyenlettel leírható lineáris Gauss-modell additív, rögzített varianciájú Gauss-zajjal. (b) E modell alapján generált 50 adatpontból álló halmaz.



Created by XMLmind XSL-FO Converter.

Itt a paraméterek θ_1, θ_2 és σ. Mint a 20.4. ábrán szemléltettük, az (x_j, y_j) párok gyűjteménye adja az adatokat. A szokásos módszereket használva (lásd 20.6. feladat) megkaphatjuk a paraméterek maximum-likelihood értékeit. Itt most egy másik dolgot akarunk megmutatni. Vegyük csupán a θ_1 és θ_2 paramétereket, amelyek az x és y közti lineáris összefüggést definiálják! Nyilvánvaló, hogy a log likelihood ezen paraméterekkel történő maximalizálása azonos azzal, mintha a (20.5) kifejezésben a kitevő számlálóját *minimalizálnánk:*

$$E = \sum_{j=1}^{N} \left(y_j - \left(\theta_1 x_j + \theta_2 \right) \right)^2$$

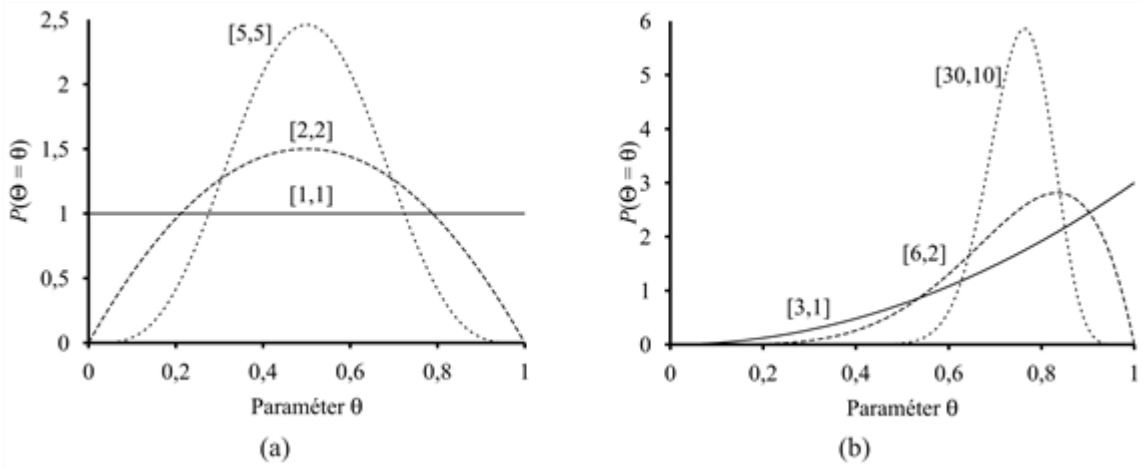
Az $(y_j - (\theta_1 x_j + \theta_2))$ mennyiség valójában az (x_j, y_j) **hibá**ja (**error**) – azaz a tényleges y_j érték és a becsült érték különbsége. Így E nem más, mint a jól ismert **hibanégyzetek összege** (**sum of squared errors**). Ezt a standard **lineáris regresszió** (**linear regression**) minimalizálja. Most megérthetjük, hogy miért: a hibanégyzetek összegének minimalizálása nem más, mint a maximum-likelihood lineáris (egyenessel ábrázolható) modell megadása, *feltéve, hogy az adatokat rögzített varianciájú Gauss-zaj mellett generáltuk*.

2.4. Bayes-paramétertanulás

A maximum-likelihood tanulás alkalmat ad néhány nagyon egyszerű eljárás létrehozására, de kis adathalmazok esetén súlyos hiányosságokat mutat. Például egyetlen meggyízű cukorka észlelése után az a maximum-likelihood hipotézis, hogy a csomag 100%-a meggytípusú (azaz $\theta = 1{,}0$). Ha nincs olyan hipotézis prior, hogy a csomagok mind vagy csupa meggy-, vagy csupa citromtípusúak, akkor ez nem józan következtetés. A Bayes-megközelítésű paramétertanulás egy hipotézis priort állít fel a lehetséges paraméterértékekre, és ahogy az adatok érkeznek, úgy frissíti az eloszlást.

A 20.2. (a) cukorka példának egyetlen θ paramétere van; annak valószínűsége, hogy egy véletlenszerűen kiválasztott cukorka meggyízű. Bayes-megközelítésben θ a Θ valószínűségi változó (ismeretlen) értéke, a hipotézis prior pedig nem más, mint a $\mathbf{P}(\Theta)$ a priori eloszlás. Így $P(\Theta = \theta)$ annak a priori valószínűsége, hogy a csomag θ arányban tartalmaz meggyízű cukrokat.

20.5. ábra - Példák a béta[*a,b*] eloszlásra különböző [*a,b*] értékek esetén



Ha a θ tetszőleges értéket felvehet 0 és 1 között, akkor a $\mathbf{P}(\Theta)$-nek egy folytonos eloszlásnak kell lennie, amely csak 0 és 1 között nem nulla értékű, és integrálja 1. Egy lehetséges jelölt az egyenletes eloszlás $P(\theta) = U[0, 1](\theta)$. (Lásd 13. fejezet.) Az egyenletes eloszlás a **béta-eloszlás**ok (**beta distribution**s) családjának tagja. Minden egyes béta-eloszlás két **hiperparaméter**rel[4] (**hyperparameter**) – a-val és b-vel – definiálható a következő egyenlet szerint:

$$\text{béta}[a,b](\theta) = \alpha \theta^{a-1} (1-\theta)^{b-1} \quad (20.6)$$

A (20.6) megadja θ-ra a [0, 1] tartományban a sűrűségfüggvény értékét. Az α normáló konstans a-tól és b-től függ. (Lásd 20.8. feladat.) A 20.5. ábrán bemutatjuk, hogy hogyan néz ki az eloszlás különböző a-k és b-k esetén. Az eloszlás átlaga $a/(a + b)$, tehát nagyobb a értékek arra utalnak, hogy Θ-t 1-hez közelebb hisszük,

[4] Azért hívjuk hiperparamétereknek, mert θ eloszlásának paramétereiről van szó, ahol θ maga is egy paraméter.

Created by XMLmind XSL-FO Converter.

mint 0-hoz. Az $a + b$ nagyobb értékei az eloszlást csúcsosabbá teszik, ami a Θ értéke felőli nagyobb bizonyosságunkat jelenti. Látható, hogy a béta-család a hipotézis prior lehetőségek hasznos választékát nyújtja.

Rugalmasságán túl a béta-családnak van még egy csodálatos tulajdonsága: ha a Θ priorja béta[a, b], akkor egy adatpont megfigyelése után Θ a posteriori eloszlása is béta-eloszlás. A béta-családot a logikai változók eloszlása **konjugált prior**jának (**conjugate prior**) nevezzük.[5] Lássuk, hogyan is működik ez. Tegyük fel, hogy megfigyeltünk egy meggyízű cukrot, ekkor:

$$P(\theta | D_1 = meggy) = \alpha P(D_1 = meggy | \theta) P(\theta)$$

$$= \alpha' \theta \cdot \text{beta}[a,b](\theta) = \alpha' \theta \cdot \theta^{a-1}(1-\theta)^{b-1}$$

$$= \alpha' \theta^{a}(1-\theta)^{b-1} = \text{beta}[a+1,b](\theta)$$

Tehát egy meggyízű cukrot észlelve egyszerűen inkrementáljuk az a paramétert, hasonlóképpen, ha egy citromízűt észlelünk, akkor inkrementáljuk a b paramétert – ezzel megkapjuk az a posteriori eloszlást. Ezek szerint úgy tekinthetünk a-ra és b-re, mint **virtuális számláló**kra (**virtual count**s), abban az értelemben, hogy a béta[a, b] prior pontosan úgy viselkedik, mintha egyenletes eloszlású priorral indultunk volna, és a – 1 meggycukorkát és b – 1 citromízű cukorkát láttunk volna.

Tanulmányozva a béta-eloszlások sorozatát növekvő – de állandó arányú – a és b mentén, jól láthatjuk, hogyan változik a Θ paraméter a posteriori eloszlása az adatok beérkezése során. Tegyük fel például, hogy a vizsgált zacskó 75%-a meggytípusú. A 20.5. (b) ábra mutatja a béta[3, 1], béta[6, 2], béta[30, 10] eloszlássorozatot. Nyilvánvaló, hogy az eloszlás egy – a valós Θ körül elhelyezkedő – keskeny csúcs felé tart. Nagy adathalmazok esetén a Bayes-tanulás (legalábbis ebben az esetben) ugyanahhoz az eredményhez konvergál, mint amit a maximum-likelihood tanulás adott.

A 20.2. (b) ábrán a hálónak három paramétere volt: θ, θ_1 és θ_2, ahol θ_1 volt annak valószínűsége, hogy piros csomagolás van egy meggycukorkán, míg θ_2 az, hogy egy citromízű cukron van piros csomagolás. A Bayes hipotézis priornak mindhárom paramétert le kell fednie – azaz $P(\Theta, \Theta_1, \Theta_2)$-t kell specifikálnunk. Rendszerint **paraméterfüggetlenség**et (**parameter independence**) tételezünk fel:

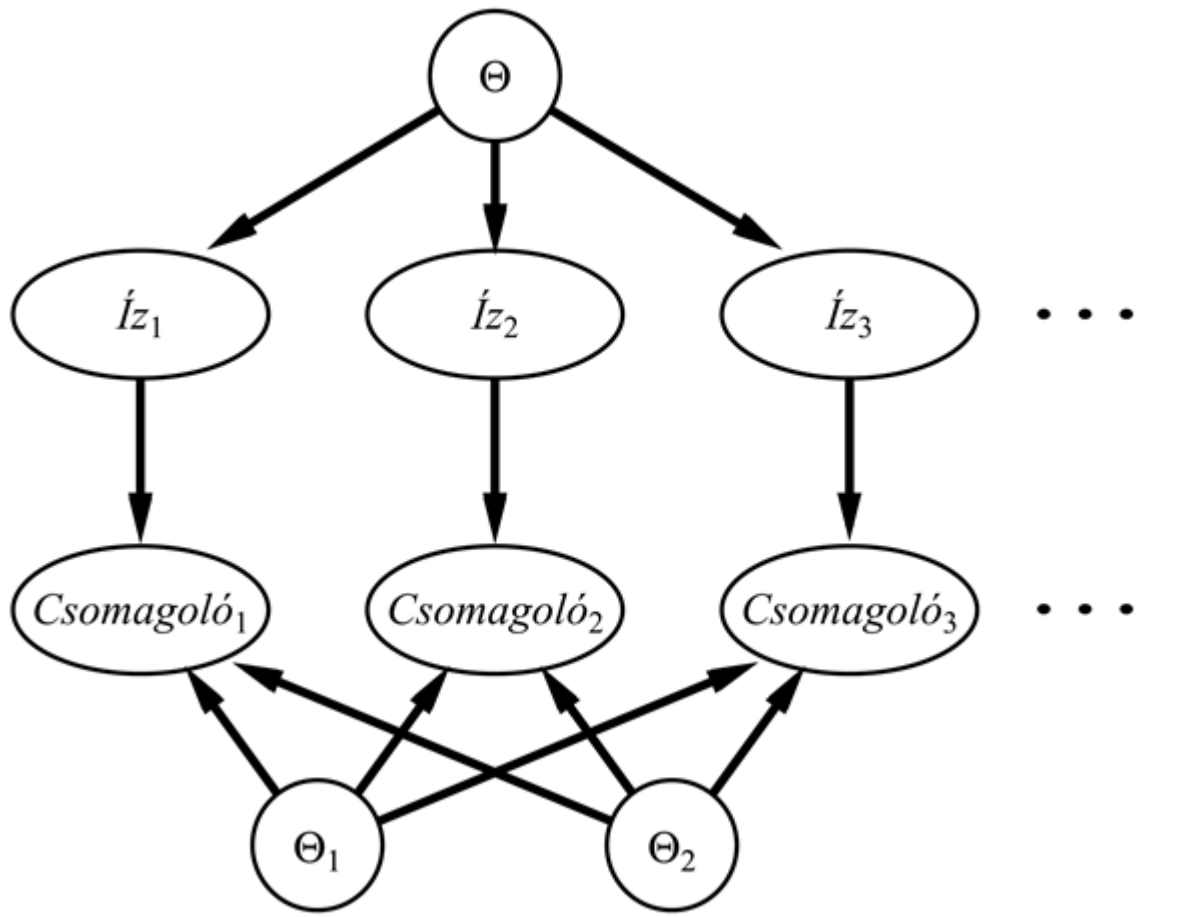
$$P(\Theta, \Theta_1, \Theta_2) = P(\Theta) P(\Theta_1) P(\Theta_2)$$

Ezen feltételezés esetén minden egyes paraméternek saját béta-eloszlása lehet, amelyet külön-külön frissíthetünk az adatok érkeztekor.

20.6. ábra - Egy Bayes-tanulásnak megfelelő Bayes-háló. A Θ, Θ_1, Θ_2 változók a posteriori eloszlásai kikövetkeztethetők az a priori eloszlásokból és az *Ízi*, *Csomagoló*$_i$ változókra vonatkozó tényekből.

[5] További konjugált priorok: a diszkrét többváltozós eloszlások paramétereire a **Dirichlet** család, a Gauss-eloszlások paramétereire a **Normal–Wishart** család. Lásd Bernardo és Smith (1994).

Created by XMLmind XSL-FO Converter.



Ha már az volt az ötletünk, hogy az ismeretlen paramétereket valószínűségi változókkal reprezentáljuk – amilyen például Θ –, akkor természetes módon adódik, hogy azokat beépítsük magába a Bayes-hálóba. Ahhoz, hogy ezt megtehessük, minden egyes példa leírásához másolatot kell készítenünk a változókról. Ha például három cukorkát figyeltünk meg, akkor szükségünk van a következő változókra: $Íz_1$, $Íz_2$, $Íz_3$ és $Csomagoló_1$, $Csomagoló_2$, $Csomagoló_3$. A Θ paraméterváltozó határozza meg minden egyes $Íz_i$ változó valószínűségét:

$$P(Íz_i = meggy \mid \Theta = \theta) = \theta$$

Hasonlóképpen a csomagoló valószínűsége Θ_1-től és Θ_2-től függ. Például:

$$P(Csomagoló_i = piros \mid Íz_i = meggy, \Theta_1 = \theta_1) = \theta_1$$

Fontos

Ezek után az egész Bayes-tanulási folyamat formalizálható egy megfelelően konstruált Bayes-háló következtetési problémájaként, amint a 20.6. ábrán látható. Egy új példány predikciója egyszerűen azt jelenti, hogy új példányváltozókat adunk a hálóhoz, amelyből egyesekre rákérdezünk. A tanulás és a predikció ezen formalizmusa nyilvánvalóvá teszi, hogy a Bayes-tanuláshoz nem kell semmilyen extra „tanulási elv”. Megállapíthatjuk továbbá, hogy *lényegében csak egyetlen tanulási algoritmus van*, ami a Bayes-háló következtetési algoritmusa.

2.5. Bayes-hálóstruktúrák tanulása

Az eddigiekben azt feltételeztük, hogy a Bayes-háló struktúrája ismert, és csak a paramétereket próbáljuk megtanulni. A háló struktúrája a terület alapvető oksági viszonyaira vonatkozó tudást reprezentálja, amit sok esetben egy szakember, de még egy naiv felhasználó is, nagyon egyszerűen meg tud adni. Néhány esetben azonban az oksági összefüggések nem állnak rendelkezésre vagy vitatottak – például bizonyos nagyvállalatok régóta állítják, hogy a dohányzás nem okoz rákot. Ilyenkor fontos megérteni, hogy a Bayes-háló struktúrája mi módon tanulható meg az adatokból. Jelenleg a struktúratanulási algoritmusok gyerekcipőben járnak, ezért csak egy elnagyolt vázlatot adunk a legfontosabb elvekről.

Created by XMLmind XSL-FO Converter.

A legkézenfekvőbb megközelítés, ha egy jó modell érdekében *keresést* folytatunk. Elindulhatunk egy kapcsolatokat nem tartalmazó modellel, majd elkezdünk szülőcsomópontot adni minden csomóponthoz, az előbbiekben bemutatott módszerekkel illesztve a paramétereket, és mérjük a modell pontosságát. Másik lehetőség, hogy egy becsült struktúrával indulunk, és hegymászó vagy szimulált lehűtést alkalmazó algoritmusokat használunk a módosításokhoz, minden egyes struktúraváltoztatás után újrahangolva a paramétereket. A módosítások közé tartozik az élek megfordítása, hozzáadása, törlése. Nem szabad ciklusokat létrehoznunk a folyamat során, ezért sok algoritmus azt feltételezi, hogy adott a változók egy rendezése, és egy csomópont szülője csak azon csomópontok közül kerülhet ki, amelyek előbb jönnek a rendezésben (éppúgy, mint a 14. fejezet konstrukciós eljárásában). A teljes általánosság kedvéért a lehetséges rendezések között is keresnünk kell.

Két alternatív módszer van arra, hogy észrevegyük, amikor egy jó megoldást találtunk. Az első annak tesztelése, hogy az aktuális adatok kielégítik-e azokat a feltételes függetlenségre vonatkozó állításokat, amelyek a struktúrában implicit módon benne vannak. Például az étterem probléma egy naiv Bayes-modellje feltételezi, hogy:

$$P(Péntek/Szombat, Bár|VárjunkE)\ P(Péntek/Szombat|VárjunkE)P(Bár|VárjunkE)$$

és leellenőrizhetjük az adatokon, hogy ugyanez az egyenlet fennáll-e a megfelelő feltételes gyakoriságok közt is. Viszont még akkor is, ha a struktúra a terület valós oksági természetét írja le, az adathalmaz – a statisztikai ingadozások miatt – az egyenletet soha nem elégíti ki *pontosan*. Ezért megfelelő statisztikai próbákat kell elvégeznünk, hogy eldöntsük: elég bizonyítékunk van-e arra, hogy a függetlenségi hipotézisek sérülnek. Az eredményül kapott háló bonyolultsága az ezen tesztekben alkalmazott küszöbtől függ majd – minél szigorúbb a függetlenségi teszt, annál több kapcsolatot adunk majd a struktúrához, és annál nagyobb lesz a túlilleszkedés veszélye.

A jelen fejezetben bemutatott gondolatoknak jobban megfelelő megközelítés annak értékelése, hogy a javasolt modell mennyire magyarázza meg az adatokat (valószínűségi értelemben). Mindamellett óvatosnak kell lennünk ennek mérésénél. Ha egyszerűen a maximum-likelihood hipotézist akarjuk megtalálni, akkor egy teljesen összekötött hálónál fogunk kikötni, mivel további szülőcsomópontok hozzáadása egy csomóponthoz nem csökkentheti a valószínűséget (lásd 20.9. feladat). Valamilyen módon büntetnünk kell a modell bonyolultságát. A MAP (vagy MLH) megközelítés egyszerűen levon egy büntetőtagot az egyes struktúrák valószínűségéből (miután hangolta paramétereiket), ezek után hasonlítja össze a különböző struktúrákat. A Bayes-megközelítés a struktúrák és paraméterek együttes priorját használja. Rendszerint túl sok – a változók számán szuperexponenciális – struktúra van ahhoz, hogy mind felett összegezzünk, így a gyakorlatban legtöbben az MCMC (Markov lánc Monte Carlo) módszert használják, hogy mintát vegyenek a struktúrákból.

A bonyolultság büntetése (akár MAP, akár Bayes-megközelítésben) fontos kapcsolatot hoz be az optimális struktúra és a feltételes valószínűségek hálóbeli reprezentációs módja között. Táblázatosan ábrázolt eloszlás esetén a bonyolultság büntetése a szülőcsomópontok számával exponenciálisan nő, míg, mondjuk, zajos-vagy eloszlások esetén csak lineárisan. Ez azt jelenti, hogy a zajos-vagy (illetve más tömören paraméterezett) modellek tanulása több szülőcsomópontot eredményez, mint a táblázatos eloszlás tanulása.

3. Rejtett változókkal történő tanulás: az EM algoritmus

Fontos

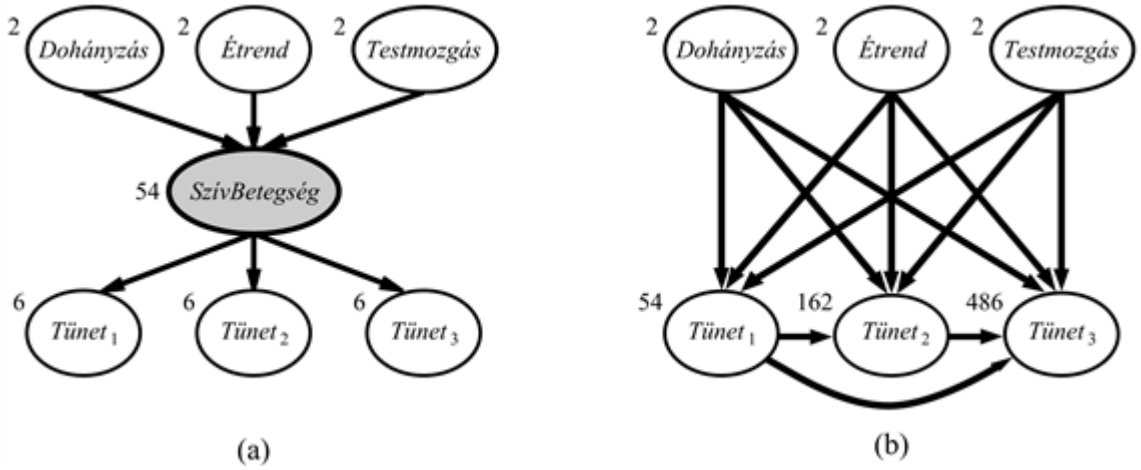
Az előző rész a teljesen megfigyelhető esettel foglalkozott. Számos valós problémában vannak **rejtett változó**k (**hidden variable**s), amelyeket néha **latens változó**knak (**latent variable**s) is hívnak. Ezek nem figyelhetők meg a tanulás céljára rendelkezésre álló adatokban. Például az orvosi feljegyzések gyakran tartalmazzák a megfigyelt tüneteket, az alkalmazott kezelést, esetleg e kezelés eredményét, de ritkán tartalmazzák magának a betegségnek a közvetlen megfigyelését.[6] Megkérdezhetnénk, „ha a betegséget magát nem figyelhetjük meg, akkor miért nem állítunk fel egy modellt nélküle? A választ a 20.7. ábrán tüntettük fel, amely szívbetegségek egy kis, fiktív modelljét mutatja. Három hajlamosító tényező és három megfigyelhető tünet van (amelyek túl nyomasztók ahhoz, hogy megnevezzük őket). Tegyük fel, hogy mindegyik háromféle értéket vehet fel (pl. *nincs*, *közepes*, *súlyos*). A rejtett változó

[6] Néhány feljegyzés tartalmazza az orvos által felállított diagnózist, de ez a szimptómák oksági következménye, amit viszont a betegség okozott.

Created by XMLmind XSL-FO Converter.

eltávolítása az (a) hálóból a (b) hálóhoz vezet; a paraméterek száma 78-ról 708-ra nő. Tehát *a rejtett változók drámaian csökkenthetik a Bayes-háló megadásához szükséges paraméterek számát*. Ez viszont drámaian csökkentheti a paraméterek megtanulásához szükséges adatok mennyiségét.

20.7. ábra - (a) A rejtett változónak feltételezett szívbetegség egyszerű diagnosztikai hálója. Minden egyes változónak három lehetséges értéke van, a változókat a feltételes eloszlásukban szereplő független változók számával címkéztük, a végső számuk 78. (b) A *SzívBetegség* eltávolításával készült ekvivalens háló. Vegyük észre, hogy a tünetváltozók most már nem feltételesen függetlenek, ha a szülőket megadjuk. Ez a háló 708 paramétert igényel.



A rejtett változók fontosak, de bonyolítják a tanulási problémát. A 20.7. (a) ábrán például nem világos, hogy adott szülőcsomópontok esetén hogyan tanulható meg a *SzívBetegség* feltételes eloszlása, mivel nem ismert az értéke az egyes esetekben. Ugyanez a probléma fellép a tünetek eloszlásának tanulásánál is. Ebben a részben egy olyan – **expectation–maximization**,[7] **EM** nevű – algoritmust ismertetünk, amely nagyon általános megoldást ad erre a problémára. Három példát mutatunk be, majd általános leírást adunk. Ez az algoritmus először holmi mágiának tűnik, de amint a mögöttes intuíciót megértettük, a tanulási problémák hatalmas területén egy sor helyen alkalmazhatjuk.

3.1. Nem ellenőrzött osztályozás: Gauss-eloszlások keverékének tanulása

A **nem ellenőrzött osztályozás** vagy **klaszterezés** (**unsupervised clustering**) az a probléma, amikor objektumok valamilyen gyűjteményében több kategóriát különböztetünk meg. A probléma nem ellenőrzött, mivel a kategóriacímkék nincsenek megadva. Tegyük fel például, hogy felvesszük százezer csillag spektrumát. Az a kérdés, hogy a spektrumok alapján vannak-e különböző *típusú* csillagok, és ha igen, akkor hányféle, és mik a jellemzőik? Mindannyian jól ismerünk olyan neveket, mint „vörös óriás” és „fehér törpe”, de a csillagoknak nincs ilyen címke a kalapjukon – a csillagászoknak nem ellenőrzött osztályozást kell végezniük ahhoz, hogy ezeket a kategóriákat felállítsák. Más példák lehetnek: a fajok, nemek, rendek stb. azonosítása a Linné-féle taxonómiában vagy hétköznapi tárgyak természetes kategóriáinak megalkotása (lásd 10. fejezet).

A nem ellenőrzött osztályozás az adatokkal kezdődik. A 20.8. (a) ábra 500 adatpontot ábrázol, mindegyik két folytonos attribútum értékét adja meg. Az adatpontok megfelelhetnek pl. csillagoknak, a két attribútum pedig pl. a spektrumvonalak erőssége két adott frekvencián. A következő lépés, hogy meg kell értenünk, milyen valószínűségi eloszlás generálhatta az adatokat. Az osztályozás azt feltételezi, hogy az adatok a *P* **kevert eloszlás**ból (**mixture distribution**) származnak. Egy ilyen eloszlás *k* **komponens**ből (**component**) áll, amelyek mindegyike önmagában egy-egy eloszlás. Az adatpont úgy jön létre, hogy először kiválasztjuk az egyik komponenst, majd ebből a komponensből generálunk egy mintát. Jelöljük a *C* véletlen változóval a komponenseket, amely változó az 1, …, *k* értékeket veheti fel. A kevert eloszlást ekkor a

[7] Az eljárást magyarul **várhatóérték-képzés–maximalizás** eljárásnak nevezhetnénk, de a magyar szakirodalomban is az angol elnevezés terjedt el. (*A szerk.*)

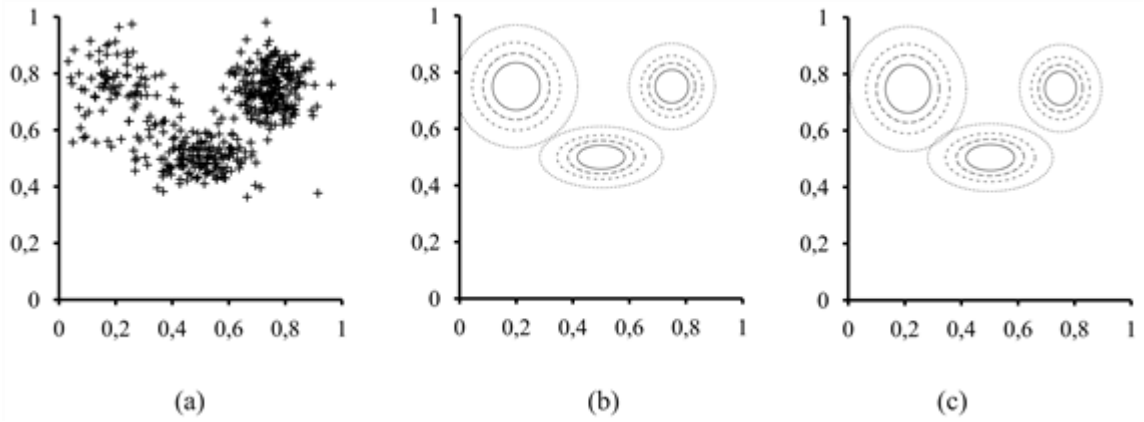
Created by XMLmind XSL-FO Converter.

$$P(\mathbf{x}) = \sum_{i=1}^{k} P(C = i) P(\mathbf{x} | C = i)$$

összefüggés írja le, ahol **x** jelenti az adatpont attribútumait. Folytonos adatok esetén a komponensekre természetesen adódik a többváltozós Gauss-eloszlás, ami az úgynevezett **kevert Gauss-** (**mixture of Gaussians**) eloszlás családot adja. A kevert Gauss-eloszlás paraméterei: $w_i = P(C = i)$ (az egyes komponensek súlya), $\boldsymbol{\mu}_i$ (az egyes komponensek várható értéke) $\boldsymbol{\Sigma}_i$ (az egyes komponensek kovarianciája). A 20.8. (b) ábra három Gauss-eloszlás keverékét mutatja, valójában ez volt az (a) ábrán látható adathalmaz forrása.

Ez esetben a nem ellenőrzött osztályozási probléma abban áll, hogy a 20.8. (a) ábrán látható nyers adatokból kell visszaállítani a 20.8. (b) ábrán láthatóhoz hasonló kevert modellt. Nyilvánvaló, hogy ha *tudnánk,* melyik adatot melyik komponens generálta, akkor könnyű lenne visszaállítanunk az egyes Gauss-komponenseket. Egyszerűen kiválaszthatnánk az egyes komponensekhez tartozó összes adatpontot, majd alkalmazhatnánk a (20.4) egyenletet (tulajdonképpen annak többváltozós verzióját) a Gauss-paramétereknek az adatokhoz való illesztésére. Másrészről, ha *tudnánk* az egyes komponensek paramétereit, akkor – legalábbis valószínűségi értelemben – meg tudnánk adni, hogy melyik adatpont melyik eloszláshoz tartozik. A probléma az, hogy sem az adatpontok komponensekhez rendelését, sem a komponensek paramétereit nem ismerjük.

20.8. ábra - (a) 500 kétdimenziós adatpont, amely három osztály jelenlétét sugallja. (b) Egy három-komponensű kevert Gauss-modell, a súlyok (balról jobbra) 0,2, 0,3 és 0,5. Az (a) ábrán látható adatokat ebből a modellből generáltuk. (c) Az adatokból az EM algoritmussal rekonstruált modell



Az EM algoritmus alapötlete ebben az összefüggésben az, hogy *úgy teszünk, mintha* ismernénk a modell paramétereit, majd kiszámítjuk minden egyes pont minden egyes komponenshez való tartozásának valószínűségét. Ezek után újraillesztjük a komponenseket az adatokhoz: mindegyik komponenst a teljes adathalmazhoz illesztjük, az egyes adatpontokat az adott komponenshez tartozás valószínűségével súlyozva. Az eljárást a konvergencia elértéig folytatjuk. Lényegében „teljessé tesszük" az adatokat azzal, hogy a rejtett változók – melyik adat melyik komponenshez tartozik – valószínűségi eloszlását kikövetkeztetjük a pillanatnyilag használt modell segítségével. A kevert Gauss-eloszlás esetén a paramétereknek tetszőleges módon kezdeti értéket adunk, majd a következő két lépést ismételjük:

1. E-lépés: Számítsuk ki a $p_{ij} = P(C = i|\mathbf{x}_j$ valószínűségeket, amelyek azt fejezik ki, hogy milyen valószínűséggel generálta az i-edik komponens $\mathbf{x}_j$-t. A Bayes-szabály alapján $p_{ij} = \alpha P(\mathbf{x}_j|C = i)\, P(C = i)$. A $P(\mathbf{x}_j|C = i)$ tényező az $\mathbf{x}_j$ valószínűsége az i-edik Gauss-eloszlásban, míg $P(C = i)$ az i-edik Gauss-eloszlás súlya. Használjuk a $p_i = \sum_j p_{ij}$ definíciót.

2. M-lépés: Számítsuk ki az új átlag, szórás és komponens súly paramétereket a következők szerint:

Created by XMLmind XSL-FO Converter.

$$\boldsymbol{\mu}_i \leftarrow \sum_j p_{ij}\mathbf{x}_j / p_i$$

$$\boldsymbol{\Sigma}_i \leftarrow \sum_j p_{ij}\left(\mathbf{x}_j\mathbf{x}_j^\top / p_i\right)$$

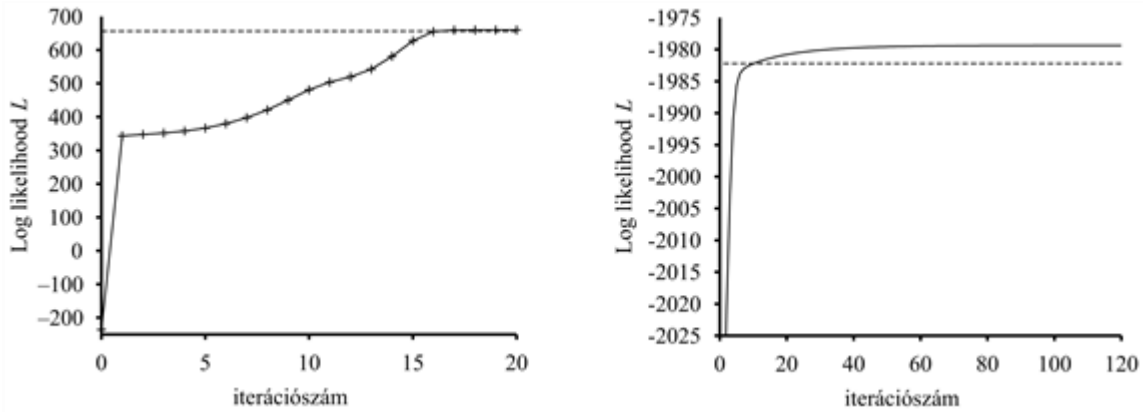
$$w_i \leftarrow p_i$$

Az E-lépés vagy *expectation* (várhatóérték-képzés) lépés úgy fogható fel, mintha a rejtett **indikátor változó** (**indicator variable**) Z_{ij} valószínűségének p_{ij} várható értékét (*expected value*) számítanánk ki. Ha az *i*-edik komponens generálta $\mathbf{x}_j$-t, akkor Z_{ij} értéke 1, különben pedig 0. Az M-lépés vagy *maximization* (maximumkeresés) lépés a rejtett változók adott várható értéke mellett megkeresi azokat a paramétereket, amelyek maximálják az adatokra felállított log likelihood függvényt.

Fontos

A végső modellt – amelyet az EM segítségével a 20.8. (a) ábra adatain tanultunk – a 20.8. (c) ábra mutatja. Látszólag megkülönböztethetetlen attól az eredeti modelltől, amellyel az adatokat generáltuk. A 20.9. (a) ábra az adatoknak a pillanatnyi modellből számított log likelihood értékét mutatja az EM algoritmus futása során. Két pontra hívjuk fel a figyelmet. Először is, a végső modellre a log likelihood némileg meghaladja az arra a valós modellre számított értéket, amelyekkel az adatokat generáltuk. Ez meglepőnek tűnhet, de egyszerűen azt mutatja, hogy az adatokat véletlenszerűen generáltuk, és nem pontosan tükrözik a mögöttes modellt. A második, hogy *az EM minden iterációs lépésben növeli az adatok log likelihood értékét*. Ez általánosságban is bizonyítható. Továbbá bizonyos feltételek mellett belátható, hogy az EM biztosan eléri a likelihood egy lokális maximumát. (Ritka esetekben elérhet egy nyeregpontba, vagy akár egy lokális minimumba is.) Ebben az értelemben az EM egy gradiensalapú hegymászó algoritmusra emlékeztet, de vegyük észre, hogy nincs „lépésméret” paramétere!

20.9. ábra - Az *L* log likelihood értéknek – az EM iterációszámának függvényében való – alakulását bemutató görbék. A vízszintes vonal a helyes modellnek megfelelő log likelihood értéket mutatja. (a) A 20.8. ábrán látható kevert Gauss-modellre vonatkozó görbe. (b) A 20.10. (a) ábrán látható Bayes-hálóra vonatkozó görbe.



A dolgok nem mennek mindig olyan jól, mint ahogy a 20.9. (a) ábra esetleg sugallja. Előfordulhat, hogy az egyik Gauss-komponens úgy összezsugorodik, hogy csak egyetlen adatpontot fed le. Ebben az esetben a varianciája nullához tart, a belőle számított likelihood pedig végtelenhez! Másik probléma lehet, hogy két komponens „összeolvad”, azonos átlagot és varianciát vesz fel, az adatpontokon pedig osztoznak. Az ilyen típusú degenerált lokális maximumok súlyos problémát jelentenek, elsősorban sokdimenziós terekben. Ennek a problémának egyik megoldása lehet megfelelő priorok állítása a modell paramétereire és az EM algoritmus MAP-változatának alkalmazása. Másik lehetőség a komponens újraindítása új paraméterekkel, ha túl kicsivé válik vagy túl közel kerül más komponensekhez. A paraméterek ésszerű kezdeti értékei szintén segítenek a probléma megoldásában.

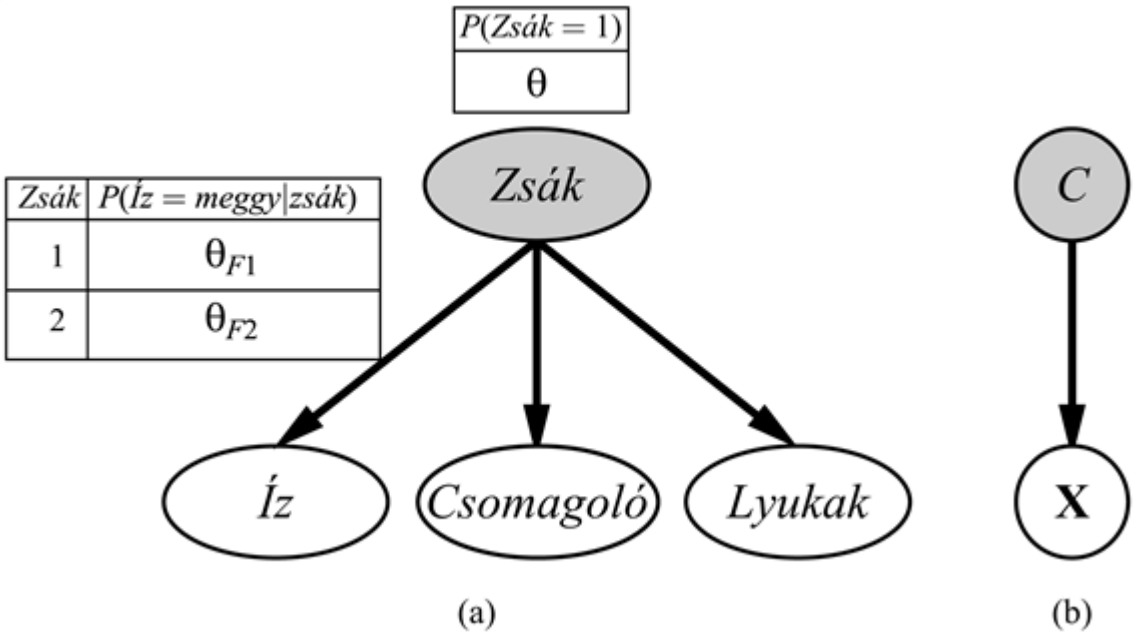
3.2. Rejtett változókkal felépített Bayes-hálók tanulása

Created by XMLmind XSL-FO Converter.

A rejtett változókkal felépített Bayes-hálók tanulásánál ugyanazokat a megközelítéseket alkalmazzuk, amelyek jól működtek a kevert Gauss-eloszlásokra. A 20.10. ábra azt a helyzetet mutatja, amikor két zsák cukorkát összekevertek. A cukrokat három tulajdonság írja le, az *Íz* és a *Csomagolás* mellett néhánynál megjelenik egy *Lyuk* a cukorka közepén, másoknál pedig nincs lyuk.

A cukorkák eloszlását minden egyes zsákban egy **naiv Bayes**-modell írja le; a tulajdonságok adott zsák esetén függetlenek, de mindegyik tulajdonság feltételes valószínűség eloszlása függ a zsáktól. A következő paramétereink vannak: θ annak az a priori valószínűsége, hogy a cukorka az 1. zsákból származik; θ_{I_1}, illetve θ_{I_2} annak valószínűsége, hogy a cukorka meggyízű, feltéve, hogy az 1., illetve a 2. zsákból származik; θ_{Cs_1}, illetve θ_{Cs_2} azokat a valószínűségeket jelölik, hogy a csomagolás piros; θ_{Ly_1}, illetve θ_{Ly_2} pedig azokat a valószínűségeket, hogy a cukorka lyukas. Vegyük észre, hogy az egész modell egy kevert modell. (Valójában a kevert Gauss-eloszlásokat is modellezhetjük Bayes-hálóval, ahogy azt a 20.10. (b) ábra mutatja.) Az ábrán a zsák egy rejtett változó, mivel attól a pillanattól, hogy összekevertük a zsákok tartalmát, többet nem tudjuk megmondani egy cukorkáról, hogy melyik zsákból származott. Vissza tudjuk ebben az esetben nyerni a zsákok leírását, ha csupán az összekevert cukorkákat tudjuk megfigyelni?

20.10. ábra - (a) A cukorkák kevert modellje. A különböző ízű cukorkák, a csomagolások, valamint a lyukas cukrok aránya a zsáktól függ, amelyet nem figyelünk meg. (b) Egy kevert Gaussra vonatkozó Bayes-háló. Az X megfigyelhető változó átlaga és kovarianciája a *C* komponenstől függ.



Vigyük végig az EM-iterációt erre a problémára. Először nézzük az adatokat. 1000 mintát generáltunk a következő paraméterekkel rendelkező (valódi) modell alapján:

$$\theta = 0{,}5,\ \theta_{I_1} = \theta_{Cs_1} = \theta_{Ly_1} = 0{,}8,\ \theta_{I_2} = \theta_{Cs_2} = \theta_{Ly_2} = 0{,}3 \quad (20.7)$$

Azaz a cukorkák egyforma valószínűséggel származhatnak mindkét zsákból, az első zsák nagyobbrészt meggyízű cukrokat tartalmaz piros csomagolásban, és sok köztük a lyukas, a második inkább nem lyukas citromízűeket, zöld csomagolásban. A lehetséges nyolcféle cukor előfordulási számai a következők:

Created by XMLmind XSL-FO Converter.

	$Cs = piros$		$Cs = zöld$	
	$Ly = 1$	$Ly = 0$	$Ly = 1$	$Ly = 0$
$I = meggy$	273	93	104	90
$I = citrom$	79	100	94	167

A paraméterek alapértékre állításával kezdünk. A könnyű számíthatóság kedvéért a következő értékeket választjuk:[8]

$$\theta^{(0)} = 0,6,\ \theta_{I1}^{(0)} = \theta_{Cs1}^{(0)} = \theta_{Ly1}^{(0)} = 0,6,\ \theta_{I2}^{(0)} = \theta_{Cs1}^{(0)} = \theta_{Ly2}^{(0)} = 0,4 \qquad (20.8)$$

Először nézzük a θ paramétert. A teljesen megfigyelhető esetben ezt közvetlenül az 1. zsákból, illetve 2. zsákból származó cukorkák *megfigyelt* számaiból becsülnénk. Mivel a zsák rejtett változó, ehelyett a *várható* számértékekkel számolunk. A várható cukorkaszám $\hat{N}(Zsák = 1)$ annak valószínűsége, az összes cukorkára összegezve, hogy a cukorka az 1. zsákból származik:

$$\theta^{(1)} = \hat{N}(Zsák = 1)/N = \sum_{j=1}^{N} P\left(Zsák = 1 \middle| íz_j, csomagoló_j, lyukak_j\right)/N$$

Ezek a valószínűségek a Bayes-hálókra kifejlesztett tetszőleges következtetési algoritmussal számíthatók. Naiv Bayes-hálókra, mint amilyen példánkban is szerepel, a következtetés „kézzel” elvégezhető, a Bayes-szabály és a feltételes függetlenség felhasználásával:

$$\theta^{(1)} = \frac{1}{N}\sum_{j=1}^{N} \frac{P\left(íz_j \middle| Zsák = 1\right)P\left(csomagoló_j \middle| Zsák = 1\right)P\left(lyukak_j \middle| Zsák = 1\right)P(Zsák=1)}{\Sigma_i P\left(íz_j \middle| Zsák = i\right)P\left(csomagoló_j \middle| Zsák = i\right)P\left(lyukak \middle| Zsák = i\right)P(Zsák = i)}$$

(Figyeljük meg, hogy a normalizáló konstans szintén a paraméterektől függ.) Alkalmazva ezt a képletet, mondjuk, a 273 pirosan csomagolt meggyízű cukorkára, amelyekben lyuk van, a következőt kapjuk:

$$\frac{273}{1000} \cdot \frac{\theta_{I1}^{(0)}\theta_{Cs1}^{(0)}\theta_{Ly1}^{(0)}\theta^{(0)}}{\theta_{I1}^{(0)}\theta_{Cs1}^{(0)}\theta_{Ly1}^{(0)}\theta^{(0)} + \theta_{I2}^{(0)}\theta_{Cs2}^{(0)}\theta_{Ly2}^{(0)}\left(1-\theta^{(0)}\right)} \approx 0,22797$$

A számok táblázatában található másik hét cukorkafajtával folytatva, $\theta^{(1)} = 0,6124$-et kapunk.

Nézzük a további paramétereket, mint pl. θ_{I1}-et. A teljesen megfigyelhető esetben ezt közvetlenül a *megfigyelt,* 1. zsákból származó meggy- és citromízű cukorkák számából becsülnénk. Az 1. zsákból származó meggycukorkák *várható* száma:

$$\sum_{j:íz_j = meggy} P\left(Zsák = 1 \middle| Íz_j = meggy, csomagoló_j, lyukak_j\right)$$

Ezek a valószínűségek ismét bármely Bayes-háló algoritmussal kiszámíthatók. Befejezve ezt az eljárást, megkapjuk az összes paraméter új értékét:

[8] A gyakorlatban jobb véletlen értékeket választani, hogy elkerüljük a szimmetria által okozott lokális maximumokat.

Created by XMLmind XSL-FO Converter.

$$\theta^{(1)} = 0,6124,\ \theta_{J1}^{(1)} = 0,6684,\ \theta_{Cs1}^{(1)} = 0,6483,\ \theta_{Ly1}^{(1)} = 0,6558$$
$$\theta_{J2}^{(1)} = 0,3887,\ \theta_{Cs2}^{(1)} = 0,3817,\ \theta_{Ly2}^{(1)} = 0,3827 \qquad (20.9)$$

Az adatok log likelihood értéke a kezdeti kb. –2044-ről az első iteráció során kb. –2021-re nő, ahogy a 20.9. (b) ábrán látható. Eszerint a frissítés magát a likelihoodot kb. $e^{23} \approx 10^{10}$ szorzófaktorral növelte. A tizedik iterációra a megtanult modell jobban illeszkedik, mint a valódi (L = –1982,214). Ezek után a javulás nagyon lassúvá válik. Ez nem szokatlan az EM algoritmusnál, ezért sok gyakorlatban használt rendszer a tanulás végső fázisában gradiensalapú algoritmusokkal – mint pl. a Newton–Raphson-módszer (lásd 4. fejezet) – kombinálja az EM algoritmust.

Fontos

Az ebből a példából levonható általános tanulság az, hogy *a rejtett változókkal történő Bayes-háló tanulásparamétereinek frissítése közvetlenül rendelkezésünkre áll az egyes példákon történő következtetés eredményeképpen. Ráadásul csak* lokális *a posteriori valószínűségekre van szükségünk az egyes paraméterekhez.* Az általános esetben, amikor az egyes X_i változók feltételes valószínűségi paramétereit – azaz a $\theta_{ijk} = P(X_i = X_{ij}|Pa_i = pa_{ik})$ értékeket – tanuljuk, adott szülőcsomópontok esetén, a frissítést a normalizált várható számértékek adják:

$$\theta_{ijk} \leftarrow \hat{N}\left(X_i = x_{ij}, Pa_i = pa_{ik}\right) / \hat{N}\left(Pa_i = pa_{ik}\right)$$

A várható számértékek a példák összegzésével nyerhetők, kiszámítva a $P(X_i = x_{ij}, Pa_i = pa_{ik})$ valószínűségeket egy tetszőleges Bayes-háló következtetési algoritmussal. A pontos algoritmusok esetén – beleértve a változó eliminációt is – ezek a valószínűségek mind a következtetés melléktermékeként közvetlenül nyerhetők, a tanuláshoz nincs szükség semmilyen speciális extra számításra. Sőt minden egyes paraméter tanulásához szükséges információ *lokálisan* rendelkezésre áll.

3.3. Rejtett Markov-modellek tanulása

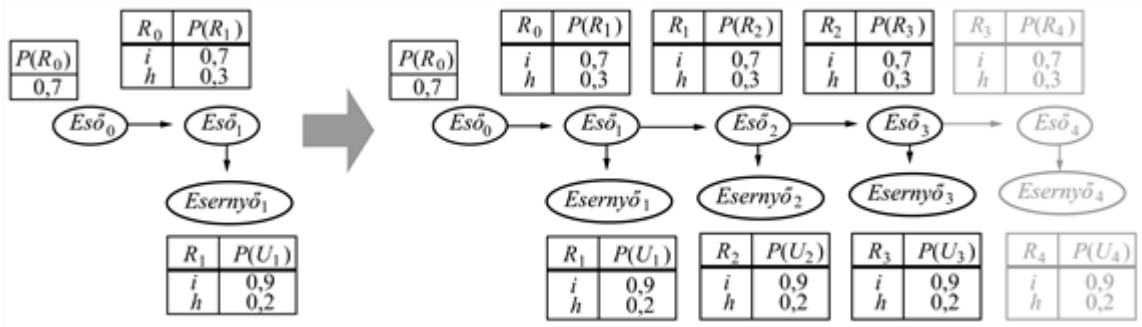
Az EM algoritmus utolsó, itt bemutatott alkalmazása a rejtett Markov-modellek (RMM) (hidden Markov model, HMM) állapotátmenet-valószínűségeinek tanulása. Idézzük fel a 15. fejezetből, hogy a rejtett Markov-modellek olyan dinamikus Bayes-hálóval reprezentálhatók, amelyeknek egyetlen, diszkrét állapotváltozója van, amint ez a 20.11. ábrán is látható. Minden egyes adatpont egy véges hosszúságú megfigyeléssorozatnak felel meg, tehát a megoldandó probléma az, hogy megfigyelési sorozatok halmaza (vagy esetleg egyetlen hosszú sorozat) alapján állapotátmenet-valószínűségeket kell megtanulnunk.

Már kimunkáltuk, hogy mi módon tudunk Bayes-hálókat megtanulni, de van egy nehézség: a Bayes-hálóknál minden egyes paraméter elkülönül, a rejtett Markov-modellben viszont időben *ismétlődnek* az egyes $\theta_{ijt} = P(X_{t+1} = j|X_t = i)$ állapotátmenet-valószínűségek (annak valószínűsége, hogy a t időpillanatban az i-edik állapotból a j-edikbe jutunk), azaz $\theta_{ijt} = \theta_{ij}$ minden t-re. Ahhoz, hogy megbecsüljük az i-edik állapotból a j-edik állapotba való átmenet valószínűségét, egyszerűen kiszámítjuk azon esetek várható arányát, amikor a rendszer az i-edik állapotban van és a j-edik állapotba megy:

$$\theta_{ij} \leftarrow \sum_t \hat{N}\left(X_{t+1} = j, X_{t=1}\right) / \sum_t \hat{N}\left(X_{t=i}\right)$$

20.11. ábra - Rejtett Markov-modellt reprezentáló kiterített dinamikus Bayes-háló (a 15.14. ábra megismétlése)

Created by XMLmind XSL-FO Converter.



Ismét tetszőleges RMM következtetési algoritmust használhatunk a várható számértékek kiszámítására. A 15.4. ábrán bemutatott **előre-hátra** (**forward-backward**) algoritmus könnyen módosítható úgy, hogy kiszámítsa a szükséges valószínűségeket. Egy fontos szempont, hogy azokra a valószínűségekre van szükségünk, amelyeket inkább **simítás**sal (**smoothing**) nyerünk, nem pedig **szűrés**sel (**filtering**): azaz nekünk utólagosan kell értékelnünk a bizonyítékokat az adott állapotátmenet bekövetkezési valószínűségének becslésekor. Mint a 15. fejezetben mondtuk, a gyilkosság bizonyítékát általában a gyilkosság bekövetkezte (azaz az i állapotból a j állapotba való átmenetet) *után* nyerjük.

3.4. Az EM algoritmus általános alakja

Számos példát láttunk az EM algoritmusra. Mindegyik tartalmazta azt a lépést, hogy kiszámítottuk a rejtett változók várható értékét minden példára, majd ezeket a várható értékeket úgy használtuk, mintha megfigyelt értékek lettek volna, és segítségükkel újraszámoltuk a paramétereket. Legyen $\mathbf{x}$ az összes példában az összes megfigyelt érték, jelölje $\mathbf{Z}$ az összes példában az összes rejtett változót, és legyen $\boldsymbol{\theta}$ a valószínűségi modell összes paramétere. Ekkor az EM algoritmus:

$$\boldsymbol{\theta}^{(i+1)} = \arg\max_{\boldsymbol{\theta}} \sum_{\mathbf{z}} P\left(\mathbf{Z} = \mathbf{z} \middle| \mathbf{x}, \boldsymbol{\theta}^{(i)}\right) L\left(\mathbf{x}, \mathbf{Z} = \mathbf{z} \middle| \boldsymbol{\theta}\right)$$

Az EM algoritmus dióhéjban ez az egyenlet. Az E-lépés az összegzés kiszámítása, ami a „teljessé tett” adatokra, a log likelihoodnak a $P(\mathbf{Z} = \mathbf{z}|\mathbf{x}, \boldsymbol{\theta}^{(i)})$ eloszlás szerint számított várható értéke. Ez a valószínűség a rejtett változóknak az adatok felhasználásával nyert a posteriori valószínűsége. Az M-lépés viszont nem más, mint ennek a log likelihoodnak a maximalizálása a paraméterek szerint. Kevert Gauss-jelekre a rejtett változók a Z_{ij}-k, ahol $Z_{ij} = 1$, ha a j-edik mintát az i-edik komponens generálta. Bayes-hálóknál az egyes példák nem megfigyelt változó értékei a rejtett változók. RMM esetben az $i \rightarrow j$ átmenetek a rejtett változók. Ha azonosítottuk a megfelelő rejtett változókat az adott alkalmazásban, akkor az általános formából kiindulva levezethető az EM algoritmus.

Amint megértettük az általános elgondolást, amin az EM alapul, az összes variáns és javítás könnyen levezethető. Például: sok esetben az E-lépés – a rejtett változók posteriorjainak kiszámítása – kezelhetetlen problémát okoz, mint például a nagy Bayes-hálókban. Kiderült, hogy alkalmazhatunk egy *közelítő* E-lépést, és még ezzel is hatékony algoritmust kapunk. Egy mintavételi algoritmussal – mint amilyen például az MCMC (lásd 14.5. alfejezet) – a tanulási folyamat nagyon szemléletes: minden egyes, az MCMC által bejárt állapotot (a rejtett és megfigyelt változók konfigurációját) úgy kezelünk, mintha teljes megfigyelés volna. Ennek megfelelően a paraméterek minden egyes MCMC-átmenet után frissíthetők. A közelítő következtetés más formái – mint például a variációs és hurkos módszerek – szintén hatékonynak bizonyultak nagyon nagy hálók tanulásánál.

3.5. Bayes-hálóstruktúra tanulása rejtett változók esetén

A 20.2. alfejezetben a Bayes-hálóstruktúra tanulásának problémáját tárgyaltuk teljes adatok esetén. Amikor rejtett változókat is figyelembe veszünk, a dolgok nehezebbé válnak. A legegyszerűbb esetben a rejtett változók a megfigyeltekkel együtt fel vannak sorolva, bár az értékük nem ismert, de a tanuló algoritmus tud a létezésükről, és helyet kell nekik találjon a hálóstruktúrában. Például egy algoritmus tanulhatja úgy a 20.7. (a) ábrán látható struktúrát, hogy megadjuk neki, hogy egy *SzívBetegség* nevű (háromértékű) változót be kell építsen a struktúrába. Ha a tanuló algoritmus nem kapja meg ezt az információt, akkor két lehetőség van: vagy úgy viselkedik, mintha az adatok teljesek volnának (ami arra kényszeríti, hogy a 20.7. (b) sokparaméteres modellt tanulja meg), vagy *kitalál* új rejtett változókat a modell egyszerűsítése érdekében. Ez utóbbi

Created by XMLmind XSL-FO Converter.

megközelítés megvalósítható úgy, hogy új módosítási lehetőségeket vezetünk be a struktúrakeresésbe: a kapcsolatok módosításán túl az algoritmus hozzáadhat vagy törölhet egy rejtett változót, illetve megváltoztathatja argumentumszámát. Természetesen az algoritmus nem fogja tudni, hogy a bevezetett új változó neve *SzívBetegség*; hasonlóképpen nem tud értelmes nevet adni az értékeinek. Szerencsére általában a bevezetett új rejtett változó előzetesen már létező változókkal van kapcsolatban, így egy szakember rendszerint meg tudja vizsgálni az új változót is tartalmazó lokális feltételes eloszlásokat, és meg tudja állapítani a jelentését.

Éppúgy, mint a teljes adatokra vonatkozó esetben, egy tisztán maximum-likelihood struktúratanulás teljesen összekötött hálót eredményez (ráadásul rejtett változók nélkülit), így használnunk kell a komplexitásbüntetés valamilyen formáját. Használhatjuk az MCMC-t is a Bayes-tanulás approximációjára. Például: ismeretlen komponensszámú kevert Gauss-jelek tanulásakor túlmintavételezhetjük a számot; ekkor a közelítő Gauss-jelek számának közelítő a posteriori eloszlását az MCMC-folyamat mintavételi frekvenciája adja.

Az eddigiekben az általunk vizsgált eljárásnak volt egy külső hurka, amely a struktúrakeresés folyamata, és egy belső hurka, ami egy paraméteres optimalizálási eljárás. A teljes adatokra vonatkozó esetben a belső hurok nagyon gyors – csupán ki kell nyerni az adathalmazból a feltételes gyakoriságokat. Ha rejtett változók is vannak, akkor a belső hurok az EM algoritmussal történő számos iterációt vagy egy gradiensalapú algoritmust is tartalmazhat. Minden iterációs ciklusban ki kell számítani a Bayes-háló a posteriori eloszlásait, ami önmagában NP-teljes probléma. Ez a megközelítés bonyolult modellek tanulása esetén napjainkig nem bizonyult a gyakorlatban is alkalmazhatónak. Egy lehetséges előrelépés a **strukturális EM** (**structural EM**), ami nagyon hasonlóan működik, mint a normál (parametrikus) EM, leszámítva azt, hogy nemcsak a paramétereket, hanem a struktúrát is frissíti. Éppúgy, mint ahogy a normál EM algoritmus arra használja az aktuális paramétereket, hogy a várható számértékeket meghatározza az E-lépés során, majd alkalmazza ezeket a számértékeket az M-lépésben új paraméterek választására, a strukturális EM az aktuális struktúrát használja a várható számértékek meghatározására, majd arra alkalmazza ezeket a számértékeket az M-lépésben, hogy a potenciális új struktúrák likelihoodját meghatározza. (Ellentétben a külső hurok/belső hurok módszerrel, amelynél minden potenciális struktúrára kiszámítjuk a várható számértékeket.) Ezen az úton a strukturális EM számos strukturális változtatást tud végrehajtani a hálón anélkül, hogy egyszer is újraszámolná a várható számértékeket. Így a módszer képessé válik nemtriviális Bayes-hálóstruktúrák tanulására. Mindamellett sok munkát kell még befektetni, míg azt mondhatjuk majd, hogy a struktúratanulás problémája megoldott.

4. Példányalapú tanulás

Eddigiekben a statisztikai tanulás tárgyalásában arra koncentráltunk, hogy valószínűségi modellek egy *behatárolt* családjának paramétereit egy *struktúrájában nem kötött* adathalmazra illesszük. Például a kevert Gaussokat használó, nem ellenőrzött tanulás feltételezi, hogy az adatok megmagyarázhatók *rögzített* számú *Gauss-eloszlás összegeként*. Az ilyen módszereket **paraméteres tanulás**nak (**parametric learning**) nevezzük. A paraméteres tanuló eljárások gyakran nagyon egyszerűek és hatékonyak, de egy meghatározott behatárolt modellcsalád feltételezése sokszor túlzott egyszerűsítése annak, ami az adatokat szolgáltató valós világban történik. Nos, az természetesen igaz, hogy ha nagyon kevés adatunk van, akkor nem reménykedhetünk egy bonyolult és részletes modell megtanulásában, de butaságnak tűnik a hipotézis komplexitását akkor is rögzítve tartani, amikor az adathalmaz nagyon nagyra nő.

A paraméteres módszerekkel ellentétben a **nemparaméteres tanulás**i (**nonparametric learning**) módszerek lehetővé teszik, hogy a hipotézis bonyolultsága együtt növekedjék az adatokkal. Minél több adatunk van, annál körmönfontabb lehet a hipotézis. Két, nagyon egyszerű nemparaméteres **példányalapú tanulás**t használó módszercsaládot fogunk vizsgálni, amelyeket azért nevezünk így, mert közvetlenül a tanító példányokból hoznak létre hipotéziseket.

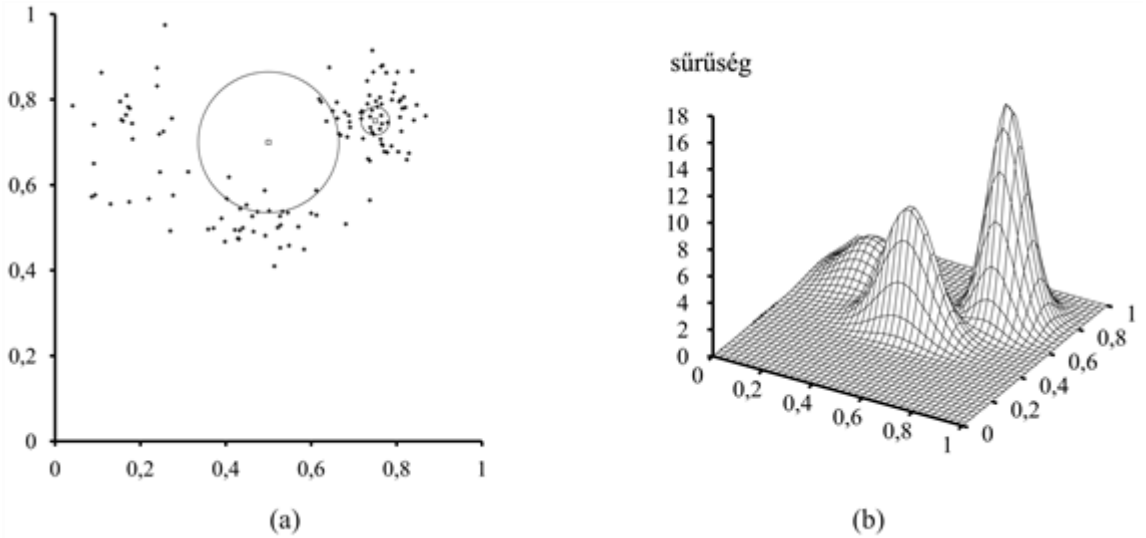
4.1. Legközelebbi-szomszéd modellek

A **legközelebbi-szomszéd** (**nearest-neighbor**) modellek alapötlete az, hogy bármely egyedi **x** bemeneti pont tulajdonságai valószínűleg hasonlók az **x** pont környezetébe eső pontok tulajdonságaihoz. Például ha **sűrűségbecslés**t (**density estimation)** kívánunk végezni – azaz meg akarjuk becsülni egy ismeretlen valószínűség sűrűségfüggvény-értékét **x**-ben –, akkor egyszerűen mérhetjük azt, hogy **x** környezete milyen sűrűn van beszórva pontokkal. Ez nagyon egyszerűen hangzik, amíg rá nem jövünk, hogy meg kell határoznunk, mit is értünk „szomszédság” alatt. Ha a szomszédság túl kicsi, akkor egyáltalán nem fog pontokat tartalmazni. Ha túl nagy, akkor esetleg az *összes* adatpontot tartalmazni fogja, és olyan sűrűséget eredményez, ami mindenütt ugyanakkora. Egy megoldási lehetőség, ha a szomszédságot úgy definiáljuk, hogy legyen éppen elég tág ahhoz,

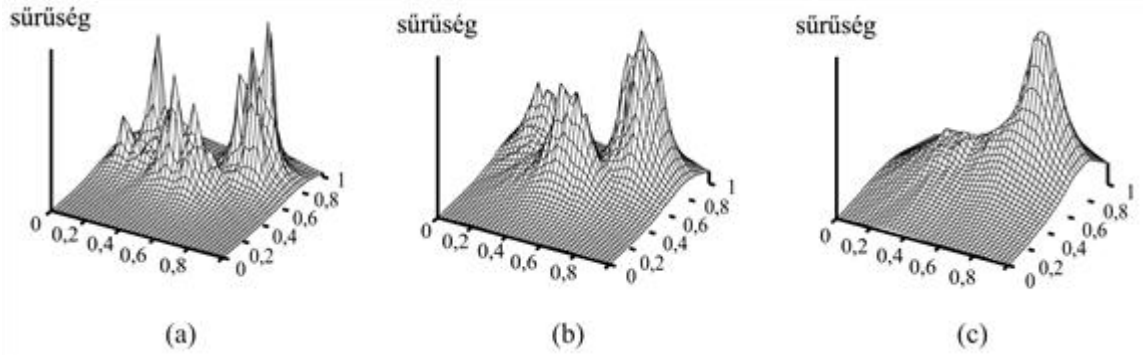
Created by XMLmind XSL-FO Converter.

hogy k pontot tartalmazzon, ahol k elég nagy ahhoz, hogy ésszerű becslést végezhessünk. Rögzített k esetén a szomszédság mérete változó, ahol ritkán vannak az adatok, ott a szomszédság tág térrész, ahol az adatok sűrűn helyezkednek el, ott a szomszédság szűk. A 20.12. (a) ábra egy példát mutat: két dimenzióban szétszórt adatpontokat. A 20.13. ábra mutatja az ezen adatokra elvégzett k-legközelebbi-szomszéd sűrűségbecslést k = 3, 10 és 40 esetére. k = 3 esetében a sűrűségbecslést minden egyes pontban csak 3 szomszédos pontra alapozzuk, és ezért erősen változó eredményt kapunk. k = 10 esetén a 20.12. (b) ábrán látható valódi sűrűség jó becslését kapjuk. k = 40 esetén a szomszédság túl tággá válik, és az adatok struktúráját mindenestől elvesztjük. Gyakorlatban – kis dimenziószám esetén – egy 5 és 10 közé eső k a legtöbb esetben jó eredményt ad. Keresztvalidáció alkalmazásával is jó értéket kaphatunk a k-ra.

20.12. ábra - (a) A 20.8. (a) ábra adatainak 128 pontos részhalmaza, két kiválasztott ponttal, és 10 legközelebbi szomszédjukkal. (b) Az adatok előállításánál használt kevert Gauss-eloszlás 3D ábrája.



20.13. ábra - A 20.12. (a) ábrán látható adatokra végzett sűrűségbecslés, k = 3 (a); k = 10 (b) és k = 40 (c) esetén



Ahhoz, hogy egy kérdéses pontban a legközelebbi szomszédokat meg tudjuk határozni, egy távolságmetrikára – $D(\mathbf{x}_1, \mathbf{x}_2)$-re – van szükségünk. A 20.12. ábra kétdimenziós példájában euklideszi távolságot használtunk. Ez nem felel meg, ha a tér minden dimenziójában valami mást mérünk – például magasságot és súlyt –, mivel egy dimenzió skálájának megváltoztatása megváltoztatja a legközelebbi szomszédok halmazát. Az egyik lehetőség, hogy minden egyes dimenzió skáláját normalizáljuk. Ehhez először megmérjük az összes tulajdonság varianciáját a teljes adathalmazon, majd a tulajdonságértéket úgy fejezzük ki, hogy megadjuk: a tulajdonságérték az adott tulajdonság varianciájának hányszorosa. (Ez a **Mahalanobis-távolság** [**Mahalanobis distance**] – amely a tulajdonságok kovarianciamátrixát veszi figyelembe – speciális esete.) Végül diszkrét esetben alkalmazhatjuk a **Hamming-távolság**ot (**Hamming distance**), amely úgy definiálja $D(\mathbf{x}_1, \mathbf{x}_2)$-t, mint azon tulajdonságok számát, amelyben $\mathbf{x}_1$ és $\mathbf{x}_2$ eltér.

Created by XMLmind XSL-FO Converter.

A 20.13. ábrán láthatóhoz hasonló sűrűségbecslések a bemeneti tér feletti együttes eloszlásokat határoznak meg. A Bayes-hálóktól eltérően a példányalapú reprezentációk nem tartalmaznak rejtett változókat, ami azt jelenti, hogy nem alkalmazhatunk nem ellenőrzött osztályozást, mint ahogy a kevert Gauss-modellnél tettük. Továbbra is alkalmazhatjuk a kívánt y értéknek az $\mathbf{x}$ bemeneti tulajdonságok alapján történő jóslására a sűrűségbecslést, ha kiszámítjuk $P(y|\mathbf{x}) = P(y, \mathbf{x})/P(\mathbf{x})$-et, feltéve, hogy a tanító adatok a kívánt értékre nézve is tartalmaznak értékeket.

A legközelebbi-szomszéd tanulási algoritmus használható közvetlen ellenőrzött tanulásra is. Ha adott egy tesztpélda $\mathbf{x}$ bemeneti értékekkel, akkor az $y = h(\mathbf{x})$ az $\mathbf{x}$ **k**-legközelebbi-szomszédjának y értékeiből nyerhető. Diszkrét esetben többségi szavazással kaphatunk egyetlen jósolt értéket. Folytonos esetben átlagolhatjuk a k darab értéket, vagy lokális lineáris regressziót végezhetünk a k pontra illesztve, ez utóbbi esetben a jóslást a kialakuló hipersík $\mathbf{x}$-beli értéke adja.

A k-legközelebbi-szomszéd tanulási algoritmus nagyon egyszerűen megvalósítható, kevés hangolást igényel, és gyakran elég jól működik. Jó dolog, ha először egy új tanulási problémán próbáljuk ki. Mindazonáltal nagy adathalmazokban hatékony mechanizmust kell találnunk arra, hogy a kérdéses $\mathbf{x}$ pont legközelebbi szomszédjait megtaláljuk, hiszen túl hosszú lenne minden pontra kiszámítani a távolságot. Egy sor szellemes, a tanító adatok előfeldolgozásán alapuló módszert javasoltak ennek a lépésnek a hatékonyná tételére. Szerencsétlen módon a legtöbb ilyen módszer nem viselkedik jól a tér dimenziójának növekedésével (azaz a tulajdonságok számának növekedésével).

A sokdimenziós terek még egy további problémát jelentenek, nevezetesen azt, hogy egy ilyen térben a legközelebbi szomszédok rendszerint nagyon messze vannak. Vegyünk egy N elemű, d dimenziós egységkockában elhelyezkedő adathalmazt, és tegyük fel, hogy a szomszédságok b oldalú hiperkockák, amelyek térfogata b^d. (Ugyanez a megfontolás működne hipergömbök feltételezése esetén is, de a hipergömb térfogatképlete bonyolultabb.) Ahhoz, hogy k pontot tartalmazzon, az átlagos szomszédság a teljes térfogat – amit egységnyinek tekintünk – k/N-ed részét kell elfoglalnia. Ennek megfelelően $b^d = k/N$, vagyis $b = (k/N)^{1/d}$. Eddig jó. Legyen most $d = 100$, $k = 10$ és $N = 1\,000\,000$ értékű. Akkor $b \approx 0{,}89$ – azaz a szomszédság majdnem az egész bemeneti teret elfoglalja! Ez arra utal, hogy sokdimenziós esetben a legközelebbi-szomszéd alapú módszerek nem megbízhatók. Alacsony dimenziószám esetén nincs probléma, $d = 2$ esetén $b = 0{,}003$.

4.2. Kernelmódszerek

A **kernelmodell**ben (**kernel model)** úgy tekintünk minden tanító példányra, mintha egy kis saját sűrűségfüggvényt – **kernelfüggvény**t (**kernel function)** – generálna. Az eredő sűrűségfüggvény becslés nem más, mint a kis kernelfüggvények súlyozott összege. Egy $\mathbf{x}_i$ tanító példány egy $K(\mathbf{x}, \mathbf{x}_i)$ kernelfüggvényt generál, amely a tér minden $\mathbf{x}$ pontjához egy valószínűséget rendel. Így a sűrűségbecslés:

$$P(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} K(\mathbf{x}, \mathbf{x_i})$$

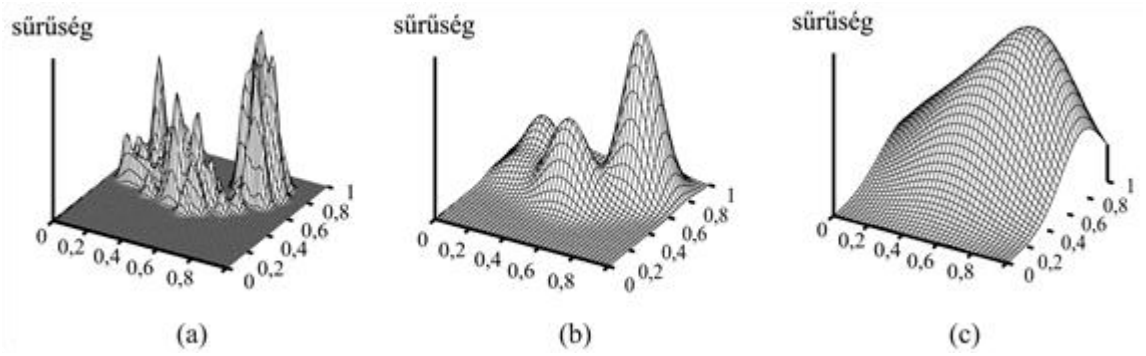
Normális esetben a kernelfüggvény csak az $\mathbf{x}$ és $\mathbf{x}_i$ közötti $D(\mathbf{x}, \mathbf{x}_i)$ távolságtól függ. A legnépszerűbb kernelfüggvény (természetesen) a Gauss-függvény. Az egyszerűség kedvéért egy gömbszimmetrikus Gauss-függvényt feltételezünk, amelynek minden tengely mentén w a szórása, azaz:

$$K(\mathbf{x}, \mathbf{x}_i) = \frac{1}{\left(w^2\sqrt{2\pi}\right)^d} e^{-\frac{D(\mathbf{x}, \mathbf{x}_i)^2}{2w^2}}$$

ahol d az $\mathbf{x}$ dimenziószáma. Hátra van még a megfelelő w érték választásának problémája. Éppúgy, mint az előzőkben egy túl kis szomszédság nagyon hepehupássá teszi a becslést – mint a 20.14. (a) ábrán látható. A (b) ábrán látható, hogy egy közepes w érték nagyon jó becslést ad. A (c) ábrán bemutattuk, hogy a túl nagy szomszédság eredményeképp teljesen elveszítjük a struktúrát. Jó w értéket választhatunk keresztvalidációval.

20.14. ábra - A 20.12. (a) ábrán látható adatok sűrűségbecslése Gauss-kernelekkel, w = 0,02 (a); w = 0,07 (b) és w = 0,20 (c) esetén.

Created by XMLmind XSL-FO Converter.



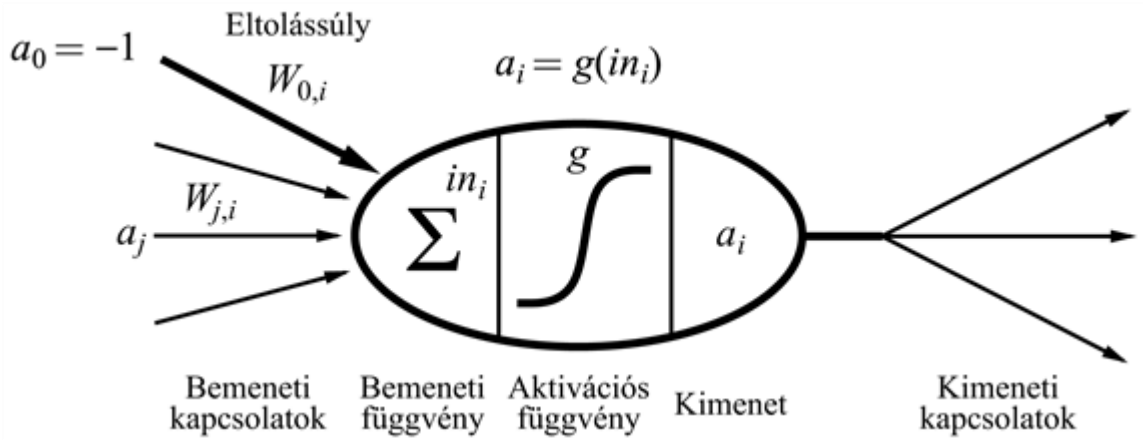
Ellenőrzött tanulást kernelekkel úgy végezhetünk, hogy a tanító példányokból származtatott *összes* jóslás *súlyozott* kombinációját vesszük. (Összehasonlítva a *k*-legközelebbi-szomszéd módszerrel, ott a *k*-legközelebbi-példány súlyozatlan kombinációját használjuk.) Az *i*-edik példánynak a kérdéses **x** pontra vett súlyát a $K(\mathbf{x}, \mathbf{x}_i)$ kernelfüggvényérték adja. Diszkrét jóslás esetén súlyozott szavazást vehetünk, folytonos esetben súlyozott átlagot vagy súlyozott lineáris regressziót. Figyeljük meg, hogy a kernelekkel végzett jóslás azt igényli, hogy az *összes* tanító példányt figyelembe kell vennünk. A kernelek összekombinálhatók a legközelebbi-szomszéd indexelési sémáival azért, hogy csupán a szomszédos példányok alapján előállított súlyozott jóslást alkalmazzuk.

5. Neurális hálók

A **neuron** egy olyan agysejt, amelynek alapfeladata elektromos jelek összegyűjtése, feldolgozása és szétterjesztése. A 2.4. szakasz - Neurális tudományok (1861-től napjainkig) részben az 1.2. ábra egy tipikus neuron sematikus rajzát mutatja. Azt gondoljuk, hogy az agy információfeldolgozó kapacitása elsősorban ilyen neuronok *hálózatából* alakul ki. Ezért a korai MI néhány kutatása mesterséges **neurális hálók** (**neural networks**) létrehozására irányult. (A terület más, szintén használt elnevezései: **konnekcionizmus** [**connectionism**], **párhuzamos elosztott feldolgozás** [**parallel distributed processing**] és **neurális számítástechnika** [**neural computation**].) A 20.15. ábra a neuron egyszerű matematikai modelljét mutatja, ahogy McCulloch és Pitts (McCulloch és Pitts, 1943) megalkották. Elnagyolva az mondható, hogy a neuron akkor „tüzel”, amikor a bemeneti értékek súlyozott összege meghalad egy küszöböt. 1943 óta sokkal részletesebb és valósághűbb modelleket alkottak mind a neuronra, mind az agy nagyobb rendszereire, ez vezetett a **számítógépes idegháló-modellezés** (**computational neuroscience**) modern tudományterületének megjelenéséhez. Másrészt az MI és a statisztika kutatóinak érdeklődését felkeltették a neurális hálózatok absztraktabb tulajdonságai, mint például az elosztott számítás elvégzésére, a bemeneti zajjal szembeni érzéketlenségre és a tanulásra való képesség. Bár ma már tudjuk, hogy más rendszerek, például a Bayes-hálók is rendelkeznek ezekkel a tulajdonságokkal, de a neurális háló maradt a tanuló rendszerek egyik leghatékonyabb és legnépszerűbb formája, ezért megéri külön tárgyalni.

20.15. ábra - A neuron egyszerű matematikai modellje. Az egység kimeneti aktivációja $a_i = g\left(\sum_{j=0}^{n} W_{j,i} a_j\right)$ **ahol** a_j **a** *j*-**edik egység kimeneti aktivációja és** $W_{j,i}$ **a** *j*-**től** *i*-**ig vezető összeköttetés súlya.**

Created by XMLmind XSL-FO Converter.



5.1. A neurális háló egységei

A neurális hálók irányított **kapcsolat**okkal (**link**) összekötött csomópontokból vagy **egység**ekből (**unit**) állnak. A *j*-edik egységtől az *i*-edik felé vezető kapcsolat hivatott az a_j aktivációt *j*-től az *i*-ig terjeszteni. Minden egyes kapcsolat rendelkezik egy hozzá aszszociált $W_{j,i}$ numerikus **súllyal (weight)**, ami meghatározza a kapcsolat erősségét és előjelét. Minden egyes *i* egység először a bemeneteinek egy súlyozott összegét számítja ki:

$$in_i = \sum_{j=0}^{n} W_{j,i} a_j$$

A kimenetét úgy kapja, hogy ezek után egy *g* **aktivációs függvény**t (**activation function**) alkalmaz a kapott összegre:

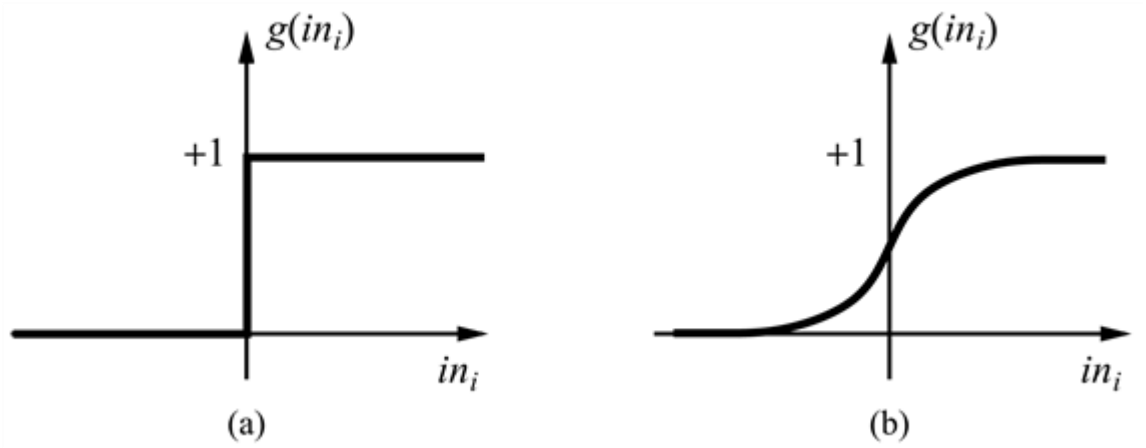
$$a_i = g(in_i) = g\left(\sum_{j=0}^{n} W_{j,i} a_j\right) \qquad (20.10)$$

Figyeljük meg, hogy használtunk egy **eltolássúly**t **(bias weight)** $W_{0,i}$-t, amelyet egy rögzített értékű $a_0 = -1$ bemenetre kapcsolunk. Rövidesen megmagyarázzuk, hogy mi a jelentősége.

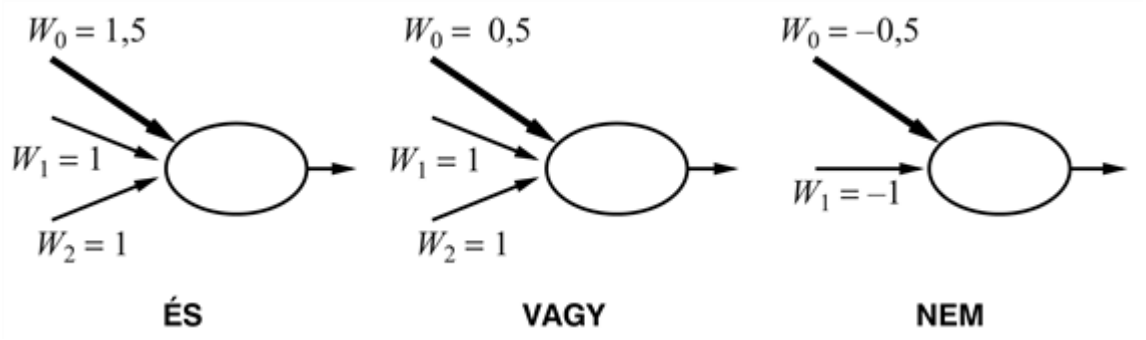
Az aktivációs függvénnyel szemben két elvárásunk van. Először az, hogy az egység legyen „aktív” (+1 körüli kimenet), ha a „helyes” bemeneteket kapja, és „inaktív” (0 körüli kimenet), ha „rossz” bemeneteket kap. Másodszor az, hogy az aktiváció legyen *nemlineáris*, különben az egész neurális háló egy egyszerű lineáris függvénnyé fajul (lásd 20.17. feladat). A 20.16. ábra kétféle aktivációs függvényt mutat be: a **küszöbfüggvény**t (**threshold function**), illetve a **szigmoid függvény**t (**sigmoid function**) (mely utóbbi **logisztikus függvény**ként [**logistic function**] is ismert). A szigmoid függvény előnye, hogy differenciálható, ami – mint később látni fogjuk – fontos a súlytanulási algoritmus szempontjából. Vegyük észre, hogy mindkét függvénynek van egy küszöbpontja (akár kemény, akár lágy) a nullánál; az eltolássúly állítja be az egység *aktuális* küszöbpontját. Ez azt jelenti, hogy az egység akkor aktiválódik, ha a „valódi” bemenetek súlyozott összege $\sum_{j=0}^{n} W_{j,i} a_j$ meghaladja $W_{0,i}$-t.

20.16. ábra - (a) A küszöb aktivációs függvény, amely 1-et ad a kimeneten, ha a bemenete pozitív, különben pedig 0-t. (Néha az előjelfüggvényt is használják ehelyett, amely ±1-et ad a bemenet előjelétől függően.) (b) A szigmoid függvény $1/(1+e^{-x})$.

Created by XMLmind XSL-FO Converter.



20.17. ábra - Megfelelő bemeneti és eltolássúlyokkal rendelkező, küszöbaktivációjú egységek képesek logikai kapuként működni



Némi fogalmunk alakulhat ki az egyes egységek működéséről, ha összehasonlítjuk őket a logikai kapukkal. Az egyes egységek tervezésének eredeti motivációi között (McCulloch és Pitts, 1943) szerepelt az, hogy képesek az alapvető logikai függvények reprezentálására. A 20.17. ábra bemutatja, hogy az és, vagy és nem logikai függvények hogyan reprezentálhatók egy megfelelő súlyokkal rendelkező küszöbegység segítségével. Ez azért fontos, mert azt jelenti, hogy ezen egységek felhasználásával tetszőleges logikai függvény kiszámítására tudunk hálózatot építeni.

5.2. Hálóstruktúrák

A neurális hálóstruktúrák két fő csoportja: a hurokmentes vagy **előrecsatolt hálók** (**feed-forward network**) és a visszacsatolt vagy **rekurrens hálók** (**recurrent network**). Az előrecsatolt háló a pillanatnyi bemenet függvényét reprezentálja, azaz nincs semmilyen más belső állapota, csak maguk a súlyok. A rekurrens háló viszont a kimeneteit visszacsatolja a bemeneteire. Ez azt jelenti, hogy a háló aktivációs szintjei dinamikus rendszert alkotnak, elérhetnek stabil állapotot, de mutathatnak oszcillációt, sőt kaotikus viselkedést is. Ezenfelül a háló egy adott bemenetre adott válasza a kezdeti állapotától függ, amely a korábbi bemenetektől függhet. Ennélfogva a rekurrens hálók (ellentétben az előrecsatolt hálókkal) rövid távú memóriát is biztosíthatnak. Ezáltal érdekesebbé válnak mint agymodellek, de egyben nehezebben is érthetők. Ez a rész az előrecsatolt hálókra koncentrál, a fejezet végén néhány hivatkozást adunk, segítve a rekurrens hálók további tanulmányozását.

Nézzük meg közelebbről azt az állítást, hogy az előrecsatolt háló a bemeneteinek függvényét reprezentálja. Vizsgáljuk a 20.18. ábra egyszerű hálózatát, amelynek két bemeneti egysége, két **rejtett egysége** (**hidden unit**) és egy kimeneti egysége van. (Az egyszerűség kedvéért ebben a példában elhagytuk az eltolásegységeket.) Adott $\mathbf{x} = (x_1, x_2)$ bemeneti vektor esetén a bemeneti egységek aktivációja $(a_1, a_2) = (x_1, x_2)$, és a hálózat a következő számítást végzi:

$$a_5 = g(W_{3,5}a_3 + W_{4,5}a_4)$$

$$= g(W_{3,5}g(W_{1,3}a_1 + W_{2,3}a_2) + W_{4,5}g(W_{1,4}a_1 + W_{2,4}a_2)) \quad (20.11)$$

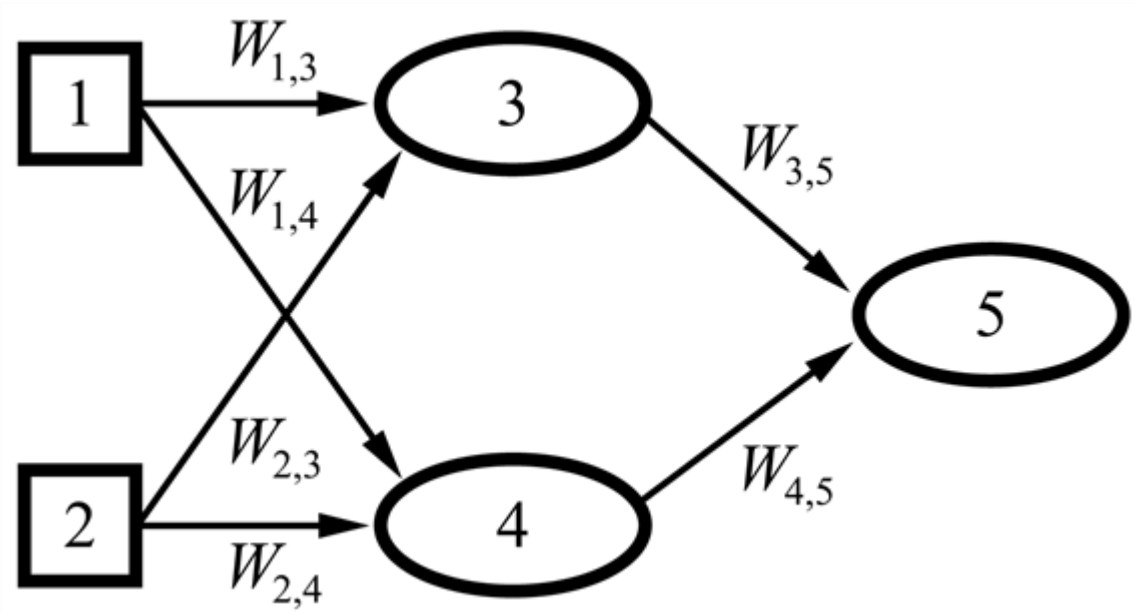
Created by XMLmind XSL-FO Converter.

Azaz kifejezve a rejtett egységek kimenetét, mint a *saját* bemeneteik függvényét, megmutattuk, hogy az egész háló a_5 végső kimenete a háló bemeneteinek függvénye. Továbbá azt látjuk, hogy a háló súlyai ennek a függvénynek a *paramétereiként* szolgálnak; ha **W**-vel jelöljük a paramétereket, akkor a háló a $h_W(\mathbf{x})$ függvényt számítja ki. Ha változtatjuk a súlyokat, akkor változik a háló által reprezentált függvény. Ez a módja a neurális hálók tanulásának.

A neurális hálót osztályozásra vagy regresszióra használhatjuk. Ha folytonos kimenete van a hálónak (pl. szigmoid egységekkel), akkor logikai osztályozás esetén hagyományosan egy kimeneti egységet használunk, és ha ennek aktivációs értéke 0,5 feletti, azt az egyik, ha 0,5 alatti, akkor a másik osztályba tartozásként interpretáljuk. Egy k osztályos osztályozási feladatnál feloszthatjuk az egyetlen kimeneti egység értéktartományát k részre, de megszokottabb, hogy ehelyett k elkülönült kimeneti egységet használjunk, ahol mindegyiknek az aktivációs értéke a bemenet adott osztályba tartozásának valószínűségét reprezentálja.

Az előrecsatolt hálókat rendszerint **réteg**ekbe (**layer**) szervezzük oly módon, hogy minden egyes egység csak a közvetlenül megelőző réteg egységeitől kap bemeneti jelet. A következő két alfejezetben egyrészt az egyrétegű hálózatokkal foglalkozunk, amelyeknek nincsenek rejtett egységei, másrészt a többrétegű hálókkal, amelyek egy vagy több rejtett réteggel rendelkeznek.

20.18. ábra - Egy nagyon egyszerű, két bemeneti egységgel, egyetlen – két egységből álló – rejtett réteggel és egy kimeneti egységgel rendelkező háló



5.3. Egyrétegű előrecsatolt neurális hálók (perceptronok)

Azt a hálót, amelyben az összes bemenet közvetlenül a kimenetekre kapcsolódik **egyrétegű neurális hálón**ak (**single layer neural network**) vagy **perceptron** (**perceptron**) hálónak nevezzük. Mivel mindegyik kimeneti egység független a többitől – mindegyik súly csak egyetlen kimenetre van hatással – vizsgálatainkat korlátozhatjuk az egykimenetű perceptronra, mint azt a 20.19. (a) ábra magyarázza.

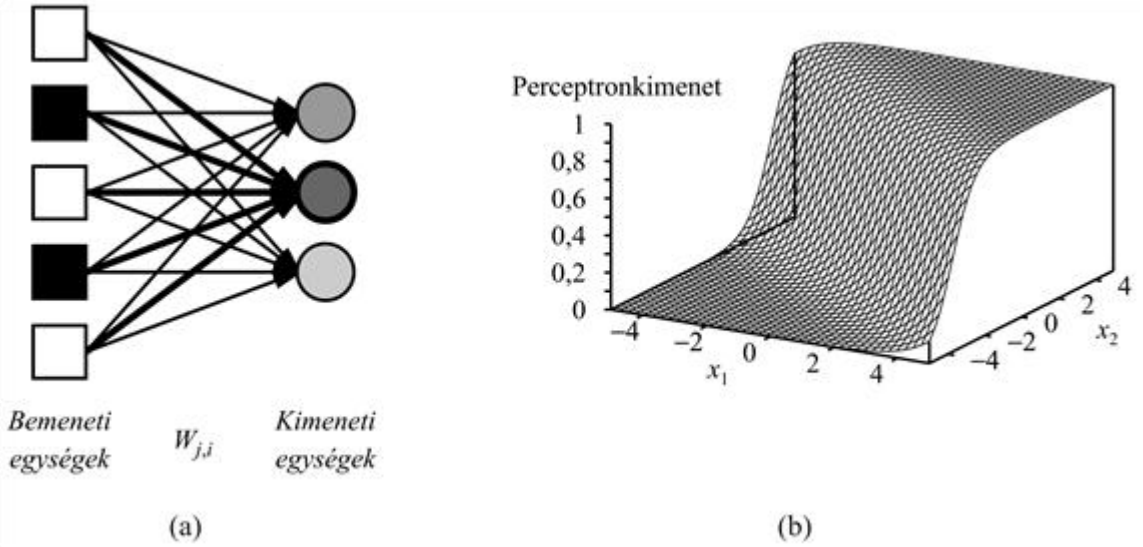
Kezdjük annak a hipotézistérnek a tanulmányozásával, amelyet egyetlen perceptron reprezentálhat. Küszöbaktivációs függvény esetén úgy tekinthetjük a perceptront, mint ami logikai függvényt reprezentál. Az elemi logikai függvényeken (és, vagy és nem, lásd 20.17. ábra) túl a perceptron képes néhány egészen „bonyolult” logikai függvényt is nagyon tömören reprezentálni. Például a **többségfüggvény**t (**majority function**), amely csak akkor ad ki 1-et, ha n bemeneteinek több mint fele 1, egy olyan perceptron reprezentál, amelynek minden súlya $W_j = 1$ és a küszöb $W_0 = n/2$. Egy döntési fának $O(2^n)$ csomópontra lenne szüksége ennek a függvénynek a reprezentálásához.

Sajnos számos olyan logikai függvény van, amelyet a küszöbfüggvényt használó perceptron nem tud reprezentálni. A (20.10) egyenletet vizsgálva látjuk, hogy a küszöbérték-perceptron akkor és csak akkor ad 1-et, ha bemeneteinek (beleértve az eltolás- bemenetet is) súlyozott összege pozitív:

Created by XMLmind XSL-FO Converter.

$$\sum_{j=0}^{n} W_j x_j > 0 \quad vagy \quad \mathbf{W} \cdot \mathbf{x} > 0$$

20.19. ábra - (a) Egy perceptronhálózat három kimeneti egységből, amelyeknek öt közös bemenete van. Ha kiválasztunk egy kimeneti egységet (mondjuk a másodikat, vastag vonallal kiemelve), akkor azt látjuk, hogy bemeneti összeköttetései nincsenek semmilyen hatással a többi kimeneti egységre. (b) Egy kétbemenetű szigmoid aktivációs függvényű perceptronegység kimenetének ábrázolása.

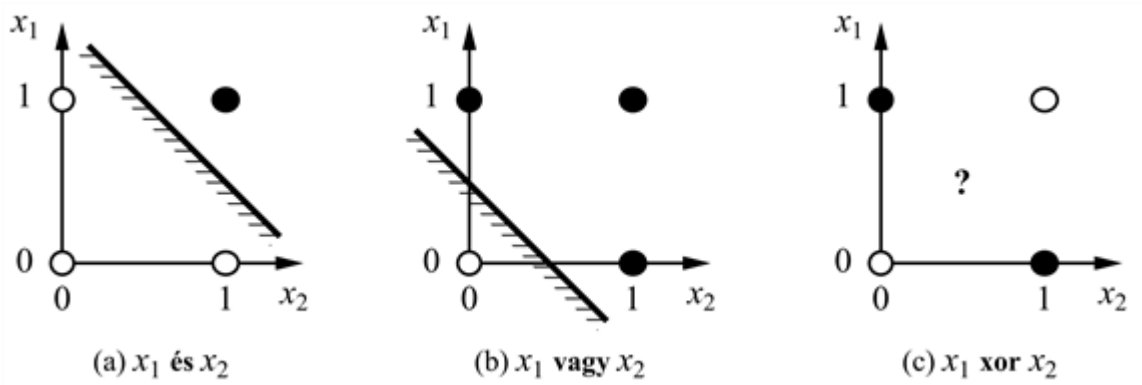


Fontos

A $\mathbf{W} \cdot \mathbf{x} = 0$ egyenlet egy hipersíkot határoz meg a bemeneti térben, tehát a perceptron akkor és csak akkor ad 1-et, ha a bemenet ennek a hipersíknak az egyik oldalán van. Ezért a küszöbperceptront **lineáris szeparátor**nak (**linear separator**) is nevezik. A 20.20. (a) és (b) ábra mutatja a szeparáló hipersíkot (ami két dimenzióban egy egyenes) a kétbemenetű és, illetve vagy függvények perceptronreprezentációja esetén. Fekete pötty jelzi a bemeneti tér olyan pontjait, amelyekre a függvény értéke 1, fehér pötty pedig az olyan pontokat, amelyre 0 ez az érték. A perceptron azért képes reprezentálni ezt a függvényt, mert létezik olyan egyenes, ami az összes fehér pontot az összes feketétől elválasztja. Az ilyen függvényeket **lineárisan szeparálható**nak (**linearly separable**) nevezzük. A 20.20. (c) ábra egy olyan függvényre mutat példát, amely *nem* szeparálható lineárisan – ez az xor függvény. Nyilvánvalóan nincs lehetőség arra, hogy egy küszöbperceptron megtanulja ezt a függvényt. Általánosságban elmondható, hogy *egy küszöbperceptron csak lineárisan szeparálható függvények reprezentációjára képes*. Ez a függvényeknek csak kis töredékét jelenti; a 20.14. feladat azt kéri, hogy fejezze ki számszerűen, mekkora is ez a töredék. A szigmoiddal felépített perceptronok hasonlóképpen korlátozott képességűek abban az értelemben, hogy csupán „lágy” lineáris szeparátorokat reprezentálnak. (Lásd 20.19. (b) ábra.)

20.20. ábra - Lineáris szeparálhatóság küszöbperceptronok esetén. Fekete pötty jelzi a bemeneti tér olyan pontjait, amelyekre a függvény értéke 1, fehér pötty pedig az olyan pontokat, amelyre 0 ez az érték. A perceptronkimenet az egyenes nem árnyékolt oldalán 1. A (c) esetben nincs olyan egyenes, amely jól osztályozná a bemeneteket.

Created by XMLmind XSL-FO Converter.



(a) x_1 **és** x_2 (b) x_1 **vagy** x_2 (c) x_1 **xor** x_2

Fontos

Korlátozott kifejezőképessége ellenére a küszöbperceptronnak vannak előnyei is. Különösen is fontos, hogy *létezik olyan egyszerű tanuló algoritmus, amely a küszöbperceptront tetszőleges lineárisan szeparálható adathalmazra képes illeszteni*. Mégsem mutatjuk most be ezt, inkább *levezetünk* egy ehhez közel álló tanuló algoritmust a szigmoid perceptronokra.

Ennek az algoritmusnak az az alapgondolata (valójában a neuronháló tanuló algoritmusok legtöbbjének is ez), hogy a tanító mintahalmazon mért hiba valamilyen mértékének minimalizálása érdekében módosítjuk a neuronháló súlyait. Tehát a tanulást formálisan a **súlytér**ben (**weight space**) végzett optimalizálási kereséskénti fogalmazzuk meg.[9] A hiba „klasszikus” mértéke a **négyzetes hibaösszeg**, amit a lineáris regressziónál a 2.3. szakasz - Maximum-likelihood paramétertanulás: folytonos eset részben is használtunk. Egyetlen, **x** bemenettel és *y* kívánt kimenettel megadott példa négyzetes hibája a következőképpen írható fel:

$$E = \frac{1}{2}Err^2 \equiv \frac{1}{2}\left(y - h_{\mathbf{W}}(\mathbf{x})\right)^2$$

ahol $h_{\mathbf{W}}(\mathbf{x})$ a perceptron kimeneti értéke.

A négyzetes hiba csökkentésére gradiensalapú optimalizálási eljárást használhatunk, ehhez az *E* minden egyes súlyra vonatkozó parciális deriváltját meg kell határoznunk. Tehát:

$$\begin{aligned}\frac{\partial E}{\partial W_j} &= Err \times \frac{\partial Err}{\partial W_j} \\ &= Err \times \frac{\partial}{\partial W_j}\left(y - g\left(\sum_{j=0}^{n} W_j x_j\right)\right) \\ &= -Err \times g'(in) \times x_j\end{aligned}$$

ahol *g*' az aktivációs függvény deriváltja.[10] Amikor a gradiensalapú algoritmusban *csökkenteni* akarjuk *E*-t, a súlyfrissítés összefüggése a következő:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j \quad (20.12)$$

ahol α a **bátorsági faktor** (vagy **tanulási faktor, learning rate**). Intuitíve ez nagyon ésszerű. Ha a hiba $Err = y - h_{\mathbf{W}}(\mathbf{x})$ pozitív, akkor a háló kimenete túl kicsi, tehát a pozitív bemenetekhez tartozó súlyokat *növeljük,* a negatívakhoz tartozókat pedig *csökkentjük*. Ha a hiba negatív, akkor éppen ennek ellenkezője történik.[11]

A teljes algoritmus a 20.21. ábrán látható. Egyesével sorban végigfuttatja a mintákat a hálón, és minden egyes példa után a hiba csökkentése érdekében kissé módosítja a súlyokat. A mintahalmaz egyszeri végigfuttatását

[9] A folytonos terekben használható általános optimalizálási technikákat lásd a 4.4. alfejezetben.
[10] Szigmoid függvény esetén ez a derivált $g' = g(1 - g)$.
[11] Küszöbperceptronokra, ahol $g'(in)$ nem definiált, a Rosenblatt (Rosenblatt, 1957) által kidolgozott eredeti **perceptron tanulási szabály** (**perceptron learning rule**) megegyezik a (20.12) egyenlettel, leszámítva, hogy a $g'(in)$ kimarad. Mivel $g'(in)$ minden súlyra azonos, elhagyása az egyes mintáknál összességében csak a súlyfrissítés nagyságát változtatja meg, az irányát nem.

Created by XMLmind XSL-FO Converter.

epochnak (**epoch**) nevezzük. Az epochokat addig ismételjük, amíg valamilyen leállási feltétel nem teljesül – tipikus, hogy akkor állunk le, amikor a súlyváltozások már nagyon kicsivé válnak. Más módszerek esetén eredő gradienst számítunk az egész tanító halmazra, egyszerűen összeadva az egyes példáknál a (20.12) származó gradienseket, és az eredő gradiens alapján frissítjük a súlyokat. A **sztochasztikus gradiens** (**stochastic gradient**) módszer nem ciklikusan veszi a tanító halmaz mintáit, hanem inkább véletlenszerűen választ mintákat a tanító halmazból.

A 20.22. ábra két különböző problémára bemutatja a perceptron tanulási görbéjét. A bal oldali ábrán a 11 logikai bemenetre vonatkozó többségfüggvény (tehát a kimenet akkor 1, ha 6 vagy több bemenet 1) tanulási görbéje látható. Várakozásunknak megfelelően a perceptron meglehetősen gyorsan tanulja a függvényt, mivel a többségfüggvény lineárisan szeparálható. Másrészről viszont a döntési fa tanuló nem nagyon halad a tanulással, mert a többségfüggvényt nagyon nehéz (bár nem lehetetlen) döntési fában reprezentálni. A jobb oldali ábrán az étterem példa látható. A probléma megoldása könnyen ábrázolható döntési fával, de lineárisan nem szeparálható. Az adatokra felvett legjobb szeparáló sík csak 65%-ot osztályoz helyesen.

20.21. ábra - A gradiensalapú perceptron tanulási algoritmus, feltételezve, hogy a *g* aktivációs függvény differenciálható. Küszöbperceptronokra elhagyjuk a *g*'(*in*) faktort a súlyfrissítésből. A `NEURÁLIS-HÁLÓ-HIPOTÉZIS` függvény egy olyan hipotézist ad vissza, amely bármely bemenetre kiszámítja a háló válaszát.

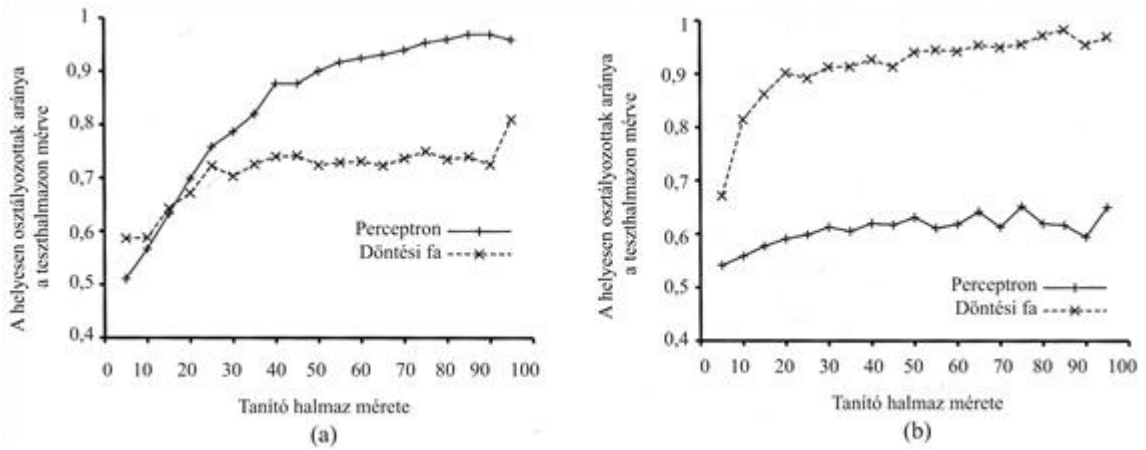
function PERCEPTRON-TANULÁS(*példák*, *háló*) **returns** egy perceptronhipotézis
  inputs: *példák*, egy mintahalmaz, mindegyikhez $\mathbf{x} = x_1, ..., x_n$ bemenet és y kimenet tartozik
    háló, egy $W_j, j = 0...n$ súlyokkal és g aktivációs függvénnyel rendelkező perceptron

  repeat
    for each p **in** *példák* **do**
      $in \leftarrow \Sigma_{j=0}^{n} W_j x_j[p]$
      $Err \leftarrow y[p] - g(in)$
      $W_j \leftarrow W_j + \alpha\, Err \times g'(in) \times x_j[p]$
  until valamilyen megállási feltétel teljesül
  return NEURÁLIS-HÁLÓ-HIPOTÉZIS(*háló*)

Az eddigiekben úgy tekintettük a perceptronokat, mint olyan determinisztikus függvényeket, amelyeknek valószínűleg hibával terhelt a kimenete. Lehetőségünk van, hogy a szigmoid perceptron kimenetét valószínűségként interpretáljuk – annak valószínűségeként, hogy adott bemenetek esetén a valós kimenet 1. Ezzel az interpretációval úgy használhatjuk a szigmoidot, mint a Bayes-hálók feltételes eloszlásainak kanonikus reprezentációját (lásd 14.3. alfejezet). Levezethetünk egy tanuló algoritmust is, a standard módszernek megfelelően maximalizálva az adatok (feltételes) log likelihood értékét, amint ezt a fejezet korábbi részében ismertettük. Lássuk, hogyan is működik ez.

20.22. ábra - A perceptronok és döntési fák teljesítményének összehasonlítása. (a) A perceptronok jobbak a 11 bemenetű többségfüggvény tanulásában. (b) A döntési fák jobbak az étterem példában a *VárjunkE* predikátum tanulásában.

Created by XMLmind XSL-FO Converter.



Vegyünk egyetlen tanító példát, amelynek kívánt kimeneti értéke T, és legyen erre a példára a perceptron válasza p. Ha $T = 1$, akkor az adat feltételes valószínűsége p, ha $T = 0$, akkor a feltételes valószínűség $(1 - p)$. Egy egyszerű trükk alkalmazásával a log likelihoodot differenciálható formában írhatjuk fel. A trükk az, hogy ha a 0/1 változót egy kifejezés kitevőjébe írjuk, akkor **indikátorváltozó**ként (**indicator variable**) viselkedik: p^Takkor p, ha $T = 1$, egyébként 1; hasonlóképpen $(1 - p)^{(1-T)}$ akkor $(1 - p)$, ha $T = 0$, különben 1. Ezek szerint az adat log likelihood értékét felírhatjuk, mint:

$$L = \log p^T(1 - p)^{(1 - T)} = T \log p + (1 - T) \log(1 - p) \quad (20.13)$$

A szigmoid függvény tulajdonságainak hála, a gradiens rendkívül egyszerű formára hozható (20.16. feladat):

$$\frac{\partial L}{\partial W_j} = Err \cdot x_j$$

Fontos

Vegyük észre, hogy *szigmoid perceptronok esetén a maximum-likelihood tanulás súlyfrissítési vektorát megadó egyenlete alapvetően azonos a négyzetes hiba minimalizálásán alapuló frissítés vektorával*. Tehát azt mondhatjuk, hogy a perceptronnak még akkor is van valószínűségi interpretációja, ha a tanulási szabályt determinisztikus megközelítéssel származtattuk.

5.4. Többrétegű előrecsatolt neurális hálók

Vizsgáljuk most a rejtett neuronokkal rendelkező hálókat. A legelterjedtebb esetben egy rejtett réteget[12] szoktak használni, ezt mutatja 20.24. ábra. A rejtett réteg hozzáadásának az az előnye, hogy kiterjeszti a háló által reprezentálható hipotézisek terét. Gondoljunk minden egyes rejtett neuronra úgy, mint ami egy lágy küszöbfüggvényt reprezentál a bemeneti térben – lásd a 20.19. (b) ábrát. Ezek után gondoljunk úgy egy kimeneti neuronra, mint ami számos ilyen függvény lineáris kombinációjának lágy küszöbfüggvénye. Például összeadva két – egymással szemben álló – lágy küszöbfüggvényt, és küszöbözve az eredményt, a 20.23. (a) ábrán látható „hegygerinc” függvényt kapjuk. Egymásra derékszögben álló két ilyen hegygerinc kombinálásával (tehát 4 rejtett neuron kimenetének kombinálásával) a 20.23. (b) ábrán látható „dudort” kapjuk.

Több rejtett neuronnal különböző helyeken több, eltérő méretű dudort tudunk létrehozni. Valójában egyetlen, megfelelően nagy rejtett réteggel a bemenetek tetszőleges folytonos függvénye tetszőleges pontossággal reprezentálható, sőt két réteggel még nemfolytonos függvények is reprezentálhatók.[13] Sajnos egy *egyedi* neurális struktúra esetén nemigen lehet megmondani, hogy milyen függvények reprezentálhatók, és milyenek nem.

Tegyük fel, hogy az étterem problémára akarunk konstruálni egy egy-rejtett-réteggel rendelkező hálót. Minden példát 10 attribútum ír le, tehát egy 10 bemenetű hálóra lesz szükség. Hány rejtett neuron kell? A 20.24. ábrán bemutatunk egy négy rejtett neuronnal felépített hálót. Az derült ki, hogy ez nagyjából megfelelő ehhez a

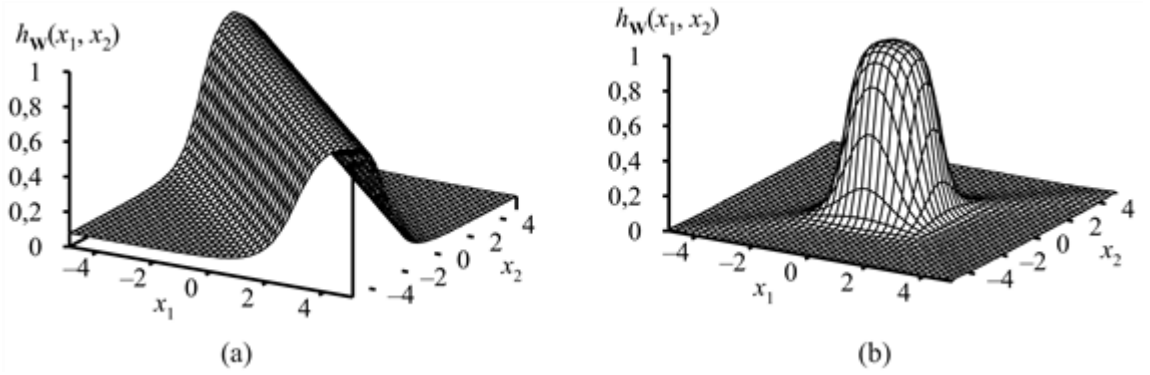
[12] Egyesek ezt háromrétegű hálózatnak nevezik, mások kétrétegűnek (mivel a bemenetek nem „igazi” neuronok). A zűrzavar elkerülése érdekében mi „egy-rejtett-rétegű háló”-nak nevezzük.

[13] A bizonyítás bonyolult, de legfőbb pontja az, hogy a bemenetek számával exponenciálisan nő a szükséges rejtett neuronok száma. Például n bemenetű logikai függvények kódolására $2^n/n$ rejtett neuron kell.

Created by XMLmind XSL-FO Converter.

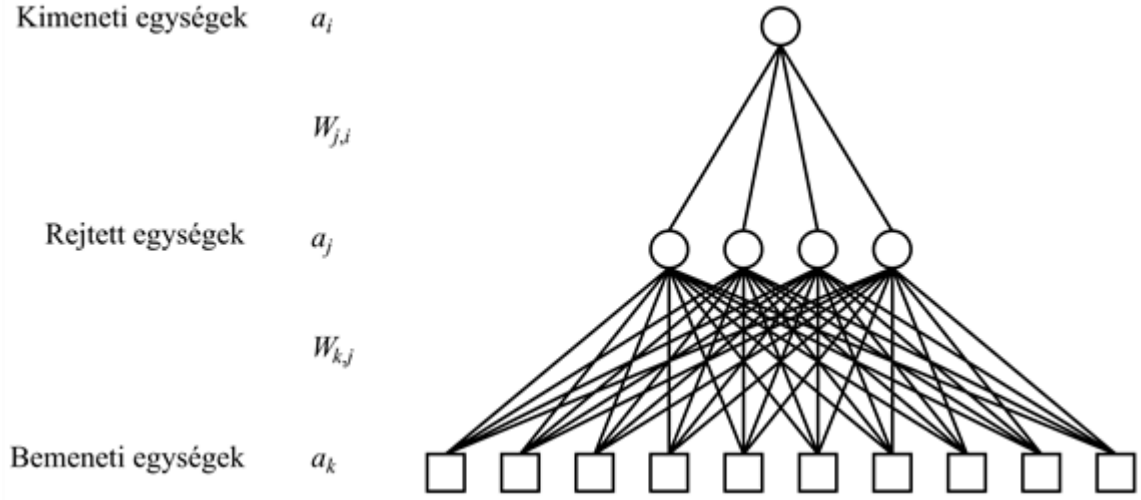
problémához. A rejtett neuronok számának előzetes meghatározása még napjainkban sem jól megoldott probléma. (Lásd 5.5. szakasz - Neurális hálóstruktúrák tanulása részben.)

20.23. ábra - (a) Két – egymással szemben álló – lágy küszöbfüggvény kombinációjának eredménye: egy hegygerinc. (b) Két hegygerinc kombinációjának eredménye a dudor.



A többrétegű háló tanuló algoritmusa hasonló a 20.21. ábrán bemutatott perceptrontanulási algoritmushoz. Egy kisebb különbség, hogy több kimenet is lehet, így nem egy skalár kimeneti értékünk, hanem egy $\mathbf{h}_W(\mathbf{x})$ kimeneti vektorunk van, és minden egyes példához is egy $\mathbf{y}$ kívánt kimeneti vektor tartozik. Fontosabb különbség, hogy míg a kimeneti rétegben a hiba nyilvánvalóan $\mathbf{y} - \mathbf{h}_W$, addig a rejtett rétegben a hiba misztikusnak tűnik, hiszen a tanító minták nem mutatják, hogy minek kellene lennie a rejtett csomópontok értékének. Kiderül, hogy a hibát **viszszaterjeszt**hetjük (**back-propagate**) a kimeneti rétegről a rejtett rétegekre. A hiba-visszaterjesztési eljárás (back-propagation) közvetlenül kiadódik a teljes hibagradiens levezetéséből. Először egy intuitív bizonyítással mutatjuk be az eljárást, majd megmutatjuk a levezetést is.

20.24. ábra - Többrétegű neurális háló egy rejtett réteggel és 10 bemeneti egységgel, alkalmas az étterem problémához



A kimeneti rétegre a súlyfrissítési szabály azonos a (20.12) egyenlettel. Több kimenetünk van, legyen Err_i az $\mathbf{y} - \mathbf{h}_W$ hibavektor i-edik komponense. Hasznos lesz, ha bevezetjük a módosított hibát, $\Delta_i = Err_i \times g'(in_i)$-t, így a súlyfrissítési szabály:

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \cdot \Delta_i \quad (20.14)$$

Ahhoz, hogy a bemeneti neuronok és a rejtett neuronok közti összeköttetések frissítését megoldjuk, a kimeneti csomópontok hibájához hasonló mennyiséget kell definiálnunk. Ez az a pont, ahol a hiba-visszaterjesztést végezzük. A gondolat az, hogy a j rejtett csomópont valamilyen arányban „felelős” minden egyes – vele összeköttetésben lévő – kimeneti csomópont Δ_i hibájáért. Így a Δ_i értékeket a rejtett csomópont és a kimeneti csomópont közötti összeköttetés erőssége alapján osztjuk és visszaterjesztjük, hogy megkapjuk a rejtett réteg Δ_j értékeit. A Δ értékek terjesztési szabálya a következő:

Created by XMLmind XSL-FO Converter.

$$\Delta_j = g'(in_j)\sum_i W_{j,i}\Delta_i \qquad (20.15)$$

Ezek után a bemenetek és a rejtett réteg közötti súlyok frissítési szabálya szinte azonos a kimeneti réteg frissítési szabályával:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$$

A hiba-visszaterjesztési algoritmus a következő módon foglalható össze:

- Számítsuk ki a kimeneti neuronokra a Δ értékeket a megfigyelt hiba alapján.
- A kimeneti réteggel kezdve ismételjük a következő lépéseket minden rétegre, amíg a legelső rejtett réteget el nem érjük:
  - Terjesszük vissza a Δ értékeket a megelőző rétegre.
  - Frissítsük a két réteg közötti súlyokat.

Az algoritmust a 20.25. ábra mutatja be részleteiben.

20.25. ábra - A többrétegű hálókra kidolgozott hiba-visszaterjesztéses tanulási algoritmus

function HIBA-VISSZATERJESZTÉS-TANULÁS(*példák*, *háló*) **returns** egy neurális háló
  inputs: *példák*, minták halmaza, mindegyikhez **x** bemeneti és **y** kimeneti vektor
    háló, többrétegű háló, L réteggel, $W_{j,i}$ súlyokkal és g aktivációs függvénnyel

  repeat
    for each p **in** *példák* **do**
      for each j csomópontra a bemeneti rétegben **do** $a_j \leftarrow x_j[p]$
      for $\ell = 2$ **to** L **do**
        $in_i \leftarrow \sum_j W_{j,i}\, a_j$
        $a_i \leftarrow g(in_i)$
      for each i csomópontra a kimeneti rétegben **do**
        $\Delta_i \leftarrow g'(in_i) \times (y_i[p] - a_i)$
      for $\ell = L - 1$ **to** 1 **do**
        for each j csomópontra az ℓ-edik rétegben **do**
          $\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i}\, \Delta_i$
          for each i csomópontra az $(\ell + 1)$-edik rétegben **do**
            $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$
  until valamilyen megállási feltétel teljesül
  return NEURÁLIS-HÁLÓ-HIPOTÉZIS(*háló*)

A matematika iránt vonzalmat érzők kedvéért most az alapegyenletekből levezetjük a hiba-visszaterjesztési algoritmust. Egyetlen mintára a négyzetes hiba definíciója:

$$E = \frac{1}{2}\sum_i (y_i - a_i)^2$$

ahol az összegzés a kimeneti réteg csomópontjaira vonatkozik. Egy bizonyos $W_{j,i}$ súlyra vett gradiens kiszámításához csak az a_i aktivációt kell deriválnunk, mivel az összeg összes többi tagja független $W_{j,i}$-től:

Created by XMLmind XSL-FO Converter.

$$\frac{\partial E}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial g(in_i)}{\partial W_{j,i}}$$
$$= -(y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{j,i}} = -(y_i - a_i)g'(in_i)\frac{\partial}{\partial W_{j,i}}\left(\sum_j W_{j,i}a_j\right)$$
$$= -(y_i - a_i)g'(in_i)a_j = -a_j\Delta_i$$

ahol Δ_i-t az előzőkkel megegyezően definiáltuk. Ahhoz, hogy a bemeneti réteget és a rejtett réteget összekötő W_{kj} súlyokra vonatkozó gradienst megkapjuk, meg kell tartanunk az i feletti teljes összegzést, hiszen az összes kimeneti a_i értékre hatással lehetnek $W_{k,j}$ változásai. Az a_j aktivációkat szintén mind ki kell fejtenünk. Aprólékosan bemutatjuk a levezetést, mivel érdekes megfigyelni, ahogy a differenciáló operátor visszaterjed hálón keresztül:

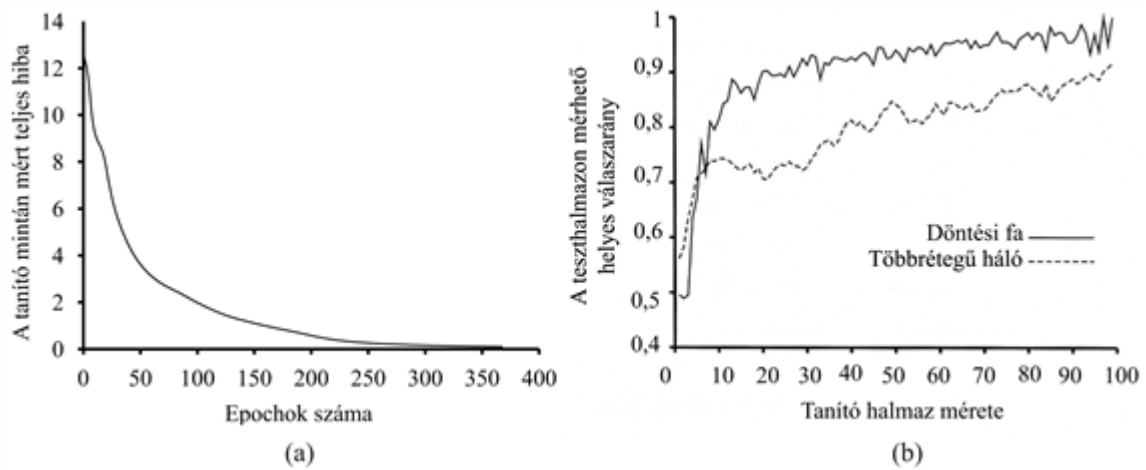
$$\frac{\partial E}{\partial W_{k,j}} = \sum_i (y_i - a_i)\frac{\partial a_i}{\partial W_{k,j}} = -\sum_i (y_i - a_i)\frac{\partial g(in_i)}{\partial W_{k,j}}$$
$$= -\sum_i (y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{k,j}} = \sum_i \Delta_i \frac{\partial}{\partial W_{k,j}}\left(\sum_j W_{j,i}a_j\right)$$
$$= -\sum_i \Delta_i W_{j,i}\frac{\partial a_j}{\partial W_{k,j}} = -\sum_i \Delta_i W_{j,i}\frac{\partial g(in_j)}{\partial W_{k,j}}$$
$$= -\sum_i \Delta_i W_{j,i}g'(in_j)\frac{\partial in_j}{\partial W_{k,j}}$$
$$= -\sum_i \Delta_i W_{j,i}g'(in_j)\frac{\partial}{\partial W_{k,j}}\left(\sum_k W_{k,j}a_k\right)$$
$$= -\sum_i \Delta_i W_{j,i}g'(in_j)a_k = -a_k\Delta_j$$

ahol a Δ_j-t az előzőkkel megegyezően definiáltuk. Tehát megkaptuk azt a frissítési szabályt, amelyet már korábban megkaptunk intuitív úton is. Az is nyilvánvaló, hogy az eljárás folytatható több mint egy rejtett rétegű hálókra is, ami igazolja a 20.25. ábrán bemutatott általános algoritmust.

Keresztülverekedvén magunkat (vagy átugorva) a matematikai levezetésen lássuk most, hogy milyen teljesítményt nyújt egy egy-rejtett-rétegű háló az étterem problémán. A 20.26. ábrán két görbét mutatunk be. Az első a **tanítási görbe** (**training curve**), amely a súlyfrissítés során az átlagos négyzetes hiba alakulását mutatja egy adott 100 elemű étterem példahalmazon mérve. Ez jól demonstrálja, hogy a háló valóban konvergál a tanító mintákra való tökéletes illeszkedéshez. A második görbe az étteremadatok standard tanulási görbéje. A neurális háló jól tanul, bár nem annyira gyorsan, mint a döntési fa tanulás. Ez talán nem meglepő, hiszen az adatokat egy egyszerű döntési fával generáltuk.

20.26. ábra - (a) Az étterem probléma egy adott példahalmazán felvett tanulási görbe, ami a súlyok számos epoch során történő módosításával elért fokozatos hibacsökkenést mutatja. (b) Összehasonlító tanulási görbék, amelyek azt mutatják, hogy a döntési fa tanulás valamivel jobb teljesítményt ad, mint a többrétegű háló hiba-visszaterjesztéses tanulása.

Created by XMLmind XSL-FO Converter.



A neurális hálók természetesen messze bonyolultabb tanulási feladatokra képesek, bár meg kell jegyeznünk, hogy szükség van némi babrálásra ahhoz, hogy megfelelő hálóstruktúrát kapjunk, és a súlytérben valahol a globális optimum közelébe konvergáljunk. A szó szoros értelmében tízezerszám vannak publikált neurális háló alkalmazások. A 20.7. alfejezet alaposabban bemutat egyet.

5.5. Neurális hálóstruktúrák tanulása

Az eddigiekben adott hálóstruktúra mellett történő súlytanulással foglalkoztunk. A Bayes-hálókhoz hasonlóan azt is meg kell értenünk, hogy hogyan találhatjuk meg a legjobb hálóstruktúrát. Ha túl nagy hálót használunk, akkor az egy nagy táblázatot kialakítva képes lesz memorizálni az összes példát, de nem feltétlenül lesz képes olyan bemenetekre jól általánosítani, amelyeket nem látott korábban.[14] Más szavakkal a neurális háló – éppúgy, mint az összes statisztikus modell – hajlamos a **túlilleszkedés**re (**overfitting**), ha túl sok modellparaméter van. Ezt bemutattuk a 18.1. ábrán (2. szakasz - Induktív tanulás rész), ahol a (b) és (c) sokparaméteres modell jól illeszkedett az adatokra, de nem volt képes olyan jó általánosításra, mint a kevés paraméterrel rendelkező (a) és (d) modellek.

Ha ragaszkodunk a teljesen összekötött hálókhoz, akkor egyedül a rejtett rétegek számára és méretére korlátozódnak a választási lehetőségeink. A szokásos megközelítés, hogy sok struktúrát kipróbálunk, és megtartjuk a legjobbat. A 18. fejezetben ismertetett **keresztvalidáció**s (**cross-validation**) technikára van szükségünk, ha el akarjuk kerülni a teszthalmazra való **kukucskálás**t (**peeking**). Azaz azt a hálóarchitektúrát választjuk, amely a validációs halmazon a legnagyobb jóslási pontosságot adja.

Ha figyelembe akarunk venni nem teljesen összekötött hálókat is, akkor szükségünk van egy hatékony módszerre, amely a lehetséges összeköttetési topológiák nagyon nagy terében keres. Az **optimális agykárosodás** (**optimal brain damage**) módszere egy teljesen összekötött hálóból indul ki, és összeköttetéseket távolít el belőle. Miután a hálót első menetben tanítottuk, egy információelméleti megközelítés segítségével meghatározzuk az eltávolítható összeköttetések optimális készletét. A hálót ezek után újratanítjuk, és ha teljesítménye nem csökkent, az eljárást megismételjük. Az összeköttetések eltávolításán felül olyan neuronokat is eltávolíthatunk, amelyek nem sokkal járulnak hozzá a megoldáshoz.

Rengeteg algoritmust javasoltak arra, hogy egy kisebb hálóból nagyobbat növesszenek. Egyikük, a **csempézés** (**tiling**), a döntési fa tanulásra emlékeztet. Az ötlet az, hogy kezdjünk egyetlen neuronnal, amely a legjobbját nyújtja, hogy annyi tanító mintára adjon helyes választ, ahányra csak lehet. További neuronokat adunk hozzá, hogy megoldjuk azokat a példákat, amelyekre az első neuron rossz választ adott. Az algoritmus csak annyi neuront ad hozzá, amennyi az összes minta megoldásához szükséges.

6. Kernelgépek

[14] Azt figyelték meg, hogy a nagyon nagy hálók *mindaddig* jól általánosítanak, *amíg a súlyaikat kis értéken tartjuk*. Ez a megszorítás az aktivációs értékeket a $g(x)$ szigmoid függvény *lineáris* tartományában tartja, ahol x közel nulla. Ez viszont azt jelenti, hogy a háló úgy viselkedik, mint egy sokkal kevesebb paraméterrel rendelkező lineáris függvény (lásd 20.17. feladat).

Created by XMLmind XSL-FO Converter.

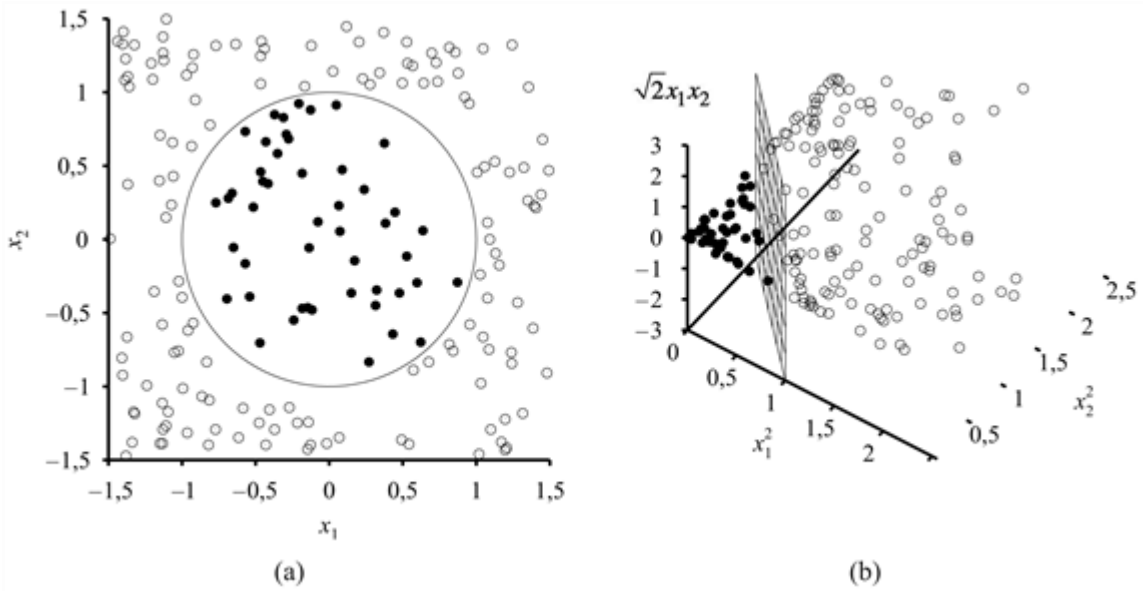
A neurális hálók tárgyalása után maradt egy dilemmánk. Az egyrétegű hálóknak nagyon egyszerű és hatékony tanulási algoritmusuk van, de nagyon korlátozott a kifejezőképességük, csupán lineáris döntési határokat képesek megtanulni a bemeneti térben. Másrészt a többrétegű hálók sokkal kifejezőbbek – általános nemlineáris függvényeket képesek reprezentálni –, de a rengeteg lokális minimum jelenléte, illetve a sokdimenziós súlytér miatt nagyon nehéz a tanításuk. Ebben az alfejezetben egy relatíve új tanulómódszer családot fedezünk fel, az úgynevezett **szupport vektor gépek**et (**support vector machines**, **SVM**), vagy általánosabban a **kernelgép**eket **(kernel machine**s). Bizonyos fokig a kernelgépek a két oldal legjobb tulajdonságait egyesítik. Azaz ezek a módszerek hatékony tanítási algoritmusokat alkalmaznak, *ugyanakkor* képesek bonyolult, nemlineáris függvények reprezentálására.

A kernelgépek teljes mélységű tárgyalása meghaladja ennek a könyvnek a kereteit, de a fő gondolatot egy példán keresztül illusztráljuk. A 20.27. (a) ábra egy kétdimenziós bemeneti teret mutat, amelyet az $\mathbf{x} = (x_1, x_2)$ attribútumok írnak le. A pozitív példák ($y = +1$) egy kör alakú rész belsejében, a negatív példák ($y = -1$) azon kívül helyezkednek el. Nyilvánvaló, hogy a probléma megoldására nem létezik lineáris szeparátor. Tegyük fel, hogy valamilyen számított tulajdonságok segítségével új formára hozzuk a példákat – azaz az összes bemeneti **x** vektort leképezzük a tulajdonságértékekből formált új, $F(\mathbf{x})$ vektorra. A példában használjuk a következő három tulajdonságot:

$$f_1 = x_1^2 \qquad f_2 = x_2^2 \qquad f_3 = \sqrt{2}x_1x_2 \tag{20.16}$$

Rövidesen látni fogjuk, hogy honnan vesszük ezeket a tulajdonságokat, de most csak nézzük meg azt, hogy mi is történt. A 20.27. (b) ábra mutatja az adatokat az új, a tulajdonságok által definiált háromdimenziós térben: ebben a térben *lineárisan szeparálhatók*! Ez a jelenség eléggé általános: ha az adatokat megfelelően sokdimenziós térbe képezzük le, akkor mindig lineárisan szeparálhatók lesznek. Itt mi csak három dimenziót használtunk,[15] de ha N pontunk van – akkor speciális esetek kivételével –, egy $N - 1$ vagy ennél magasabb dimenziós térben a pontok mindig lineárisan szeparálhatók lesznek (lásd 20.21. feladat).

20.27. ábra - (a) Egy kétdimenziós tanító halmaz, amelyben a pozitív példákat fekete, a negatívakat fehér körök jelölik. Az $x_1^2 + x_2^2 \leq 1$ valódi elválasztó határt is bejelöltük. (b) Ugyanazok az adatok az $x_1^2, x_2^2, \sqrt{2}x_1x_2$ háromdimenziós térbe való leképezés után. Az (a) ábrán látható kör alakú döntési határ a háromdimenziós térben lineáris döntési felületbe ment át.



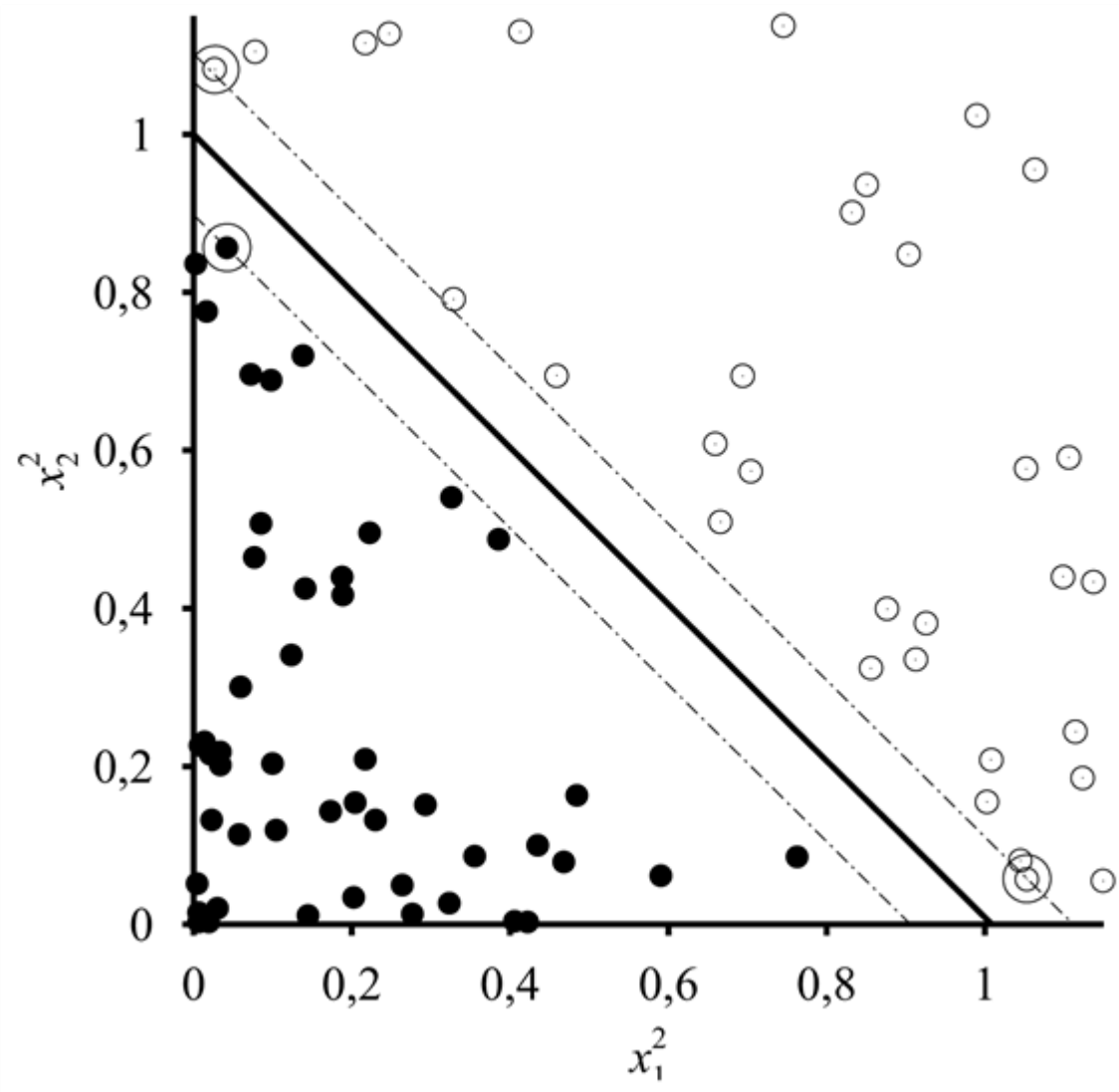
Ennyi az egész? Egyszerűen létrehozunk egy nagy halom számított tulajdonságot, és a megfelelő sokdimenziós térben megkeressük a lineáris szeparátort? Sajnálatos módon nem ilyen egyszerű. Emlékezzünk, hogy a d dimenziós térben a lineáris szeparátort egy d paraméteres egyenlet határozza meg, így aztán az a veszély

[15] Az olvasó felfedezheti, hogy elég lett volna csupán f_1 és f_2 használata, de a 3D leképezés jobban illusztrálja az ötletet.

Created by XMLmind XSL-FO Converter.

fenyeget, hogy ha $d \approx N$ (ahol N az adatpontok száma), akkor könnyen túlilleszkedhetünk az adatokra. (Ez ahhoz hasonló, mint amikor egy magas fokszámú polinommal túlillesztünk adatokat, ahogy ezt a 18. fejezetben tárgyaltuk.) Ezen okból a kernelgépek rendszerint az *optimális* lineáris szeparátort találják meg. Azt nevezzük optimálisnak, amelynek legnagyobb a **tartaléka** (**margin**): a lineáris szeparátor és a pozitív példák között az egyik oldalon, illetve a lineáris szeparátor és a negatív példák között a másikon. (Lásd 20.28. ábra.) A számítógépes tanulás elmélet módszereit (lásd 18.5. alfejezet) használva megmutatható, hogy ez a szeparátor az új példák robusztus általánosítására nézve nagyon jó tulajdonságokkal rendelkezik.

20.28. ábra - Az első két dimenzióra vetített közelkép: a 20.27. (b) ábra optimális szeparátora. A szeparátort vastag vonallal jelöltük, a hozzá legközelebbi pontokat – a szupport vektorokat – bekarikáztuk. A tartalék nem más, mint a pozitív és negatív példák közti elválasztó sáv szélessége.



Hogyan találjuk meg ezt a szeparátort? Kiderül, hogy ez egy **kvadratikus programozás**sal (**quadratic programming**) megoldható optimalizálási feladat. Tegyük fel, hogy $\mathbf{x}_i$ példáink vannak, az osztálybasorolásuk $y_i = \pm 1$, és a bemeneti térben optimális szeparátort akarunk találni. Ekkor a megoldandó kvadratikus programozási feladat azon paraméterértékek megtalálása, amelyek az $\alpha_i \geq 0$ és $\Sigma_i \alpha_i y_i = 0$ korlátozó feltételek mellett maximálják a következő kifejezést:

$$\sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i \cdot \mathbf{x}_j\right) \qquad (20.17)$$

Created by XMLmind XSL-FO Converter.

Fontos

Bár ennek a kifejezésnek a deriválása nem nagyon fontos pontja a történetnek, de azért van két lényeges tulajdonsága. Először is a kifejezésnek egyetlen, globális maximuma van, ami hatékonyan megtalálható. Másodszor *az adatok kizárólag pontpárok skalárszorzataként jelennek meg a kifejezésben*. Ez a második tulajdonság magára a szeparátorra is igaz, ha az optimális α_i-ket kiszámítottuk, akkor:

$$h(\mathbf{x}) = \text{sign}\left(\sum_i \alpha_i y_i (\mathbf{x} \cdot \mathbf{x}_i)\right) \qquad (20.18)$$

Az ezen egyenlettel definiált optimális szeparátor utolsó fontos tulajdonsága az, hogy az egyes adatpontokkal asszociált α_i súlyok mind *nullák*, kivéve a szeparátorhoz legközelebb eső pontokat – ezeket nevezzük **szupport vektor**oknak (**support vector**). (Azért nevezzük így őket, mert ők „tartják" a szeparáló síkot.) Mivel rendszerint jóval kevesebb szupport vektor van, mint adatpont, ezért az optimális szeparátort meghatározó tényleges paraméterszám rendszerint jóval kisebb N-nél.

Rendszerint nem várhatjuk el, hogy lineáris szeparátort találjunk az $\mathbf{x}$ bemeneti térben, de könnyen belátható, hogy a sokdimenziós $F(\mathbf{x})$ tulajdonságtérben találhatunk lineáris szeparátorokat. Ennek érdekében a (20.17) egyenletben $\mathbf{x}_i \cdot \mathbf{x}_j$-t egyszerűen kicseréljük $F(\mathbf{x}_i) \cdot F(\mathbf{x}_j)$-re. Ez önmagában nem túlzottan figyelemre méltó – $\mathbf{x}$ kicserélése $F(\mathbf{x})$-re *bármely* tanuló algoritmusban elérné a kívánt hatást –, de a skalárszorzatnak van néhány érdekes tulajdonsága. Az $F(\mathbf{x}_i) \cdot F(\mathbf{x}_j)$ gyakran kiszámítható anélkül, hogy először kiszámítanánk minden pontra F-et. A (20.16) egyenlettel definiált háromdimenziós tulajdonságtér példánkban némi algebrai átalakításokkal megmutatható, hogy:

$$F(x_1) \cdot F(x_2) = (x_1 \cdot x_2)^2$$

Az $(\mathbf{x}_i \cdot \mathbf{x}_j)^2$ kifejezést **kernelfüggvény**nek (**kernel function**) nevezzük, és $K(\mathbf{x}_i, \mathbf{x}_j)$-vel jelöljük. A kernelgépek szempontjából ez egy olyan függvény, amely pontpárokra alkalmazható avégett, hogy valamilyen tulajdonságtérben kiszámítsuk a skalárszorzatukat. Ennek megfelelően újrafogalmazhatjuk állításunkat: a (20.17) egyenletben $\mathbf{x}_i \cdot \mathbf{x}_j$-t egyszerűen kicserélve a $K(\mathbf{x}_i, \mathbf{x}_j)$ kernelfüggvényre, a sokdimenziós $F(\mathbf{x})$ tulajdonságtérben találhatunk lineáris szeparátorokat. Így a tanulást a sokdimenziós térben végezhetjük, de csupán kernelfüggvények értékét kell kiszámítanunk, nem kell az összes pontra a tulajdonságok teljes készletét kiszámítani.

Fontos

A következő lépés – aminek most már kézenfekvőnek kell lennie – az, hogy meglássuk, a $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$ kernelben nincs semmi különleges. Ez egy bizonyos sokdimenziós tulajdonságtérnek felel meg, de más kernelfüggvények más tulajdonságterekkel vannak kapcsolatban. A **Mercer-tétel** (1909) azt mondja ki, hogy bármely „ésszerű"[16] kernelfüggény megfelel *valamilyen* tulajdonságtérnek. Ez a tulajdonságtér még egész ártatlannak látszó kernelek esetén is nagyon nagy lehet. Például a $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i \cdot \mathbf{x}_j)^d$ **polinomiális kernel** (**polynomial kernel**) egy olyan tulajdonságtérnek felel meg, amelynek dimenziója d-ben exponenciális. Ha a (20.17) egyenletben ilyen kerneleket használunk, akkor *hatékonyan kereshetünk lineáris szeparátorokat sok milliárd (vagy egyes esetekben végtelen) dimenziós terekben*. Az eredményként kapott lineáris szeparátorokat visszavetítve az eredeti bemeneti térbe, a pozitív és negatív példákat elválasztó tetszőlegesen tekervényes, nemlineáris határfelületeket kaphatunk.

Említettük az előző részben, hogy a kernelgépek kiemelkedően teljesítenek a kézírásos számjegyek felismerésében, de gyorsan felhasználják őket más alkalmazásokhoz is, különösen olyanokhoz, ahol sok bemeneti tulajdonság van. Ennek a folyamatnak részeként számos új kernelt dolgoztak ki, amelyek karakterfüzérekre, fákra és más nemnumerikus adattípusokra alkalmazhatók. Az is megfigyelhető, hogy a kernelmódszer nemcsak optimális szeparátorok keresésére alkalmas, hanem bármely olyan algoritmusra, amely úgy átalakítható, hogy csak adatpontpárok skalárszorzatát használja, mint a (20.17), illetve (20.18) egyenlet. Amint ezt sikerült megtennünk, a skalárszorzat kicserélhető egy kernelfüggvényre, és elkészítettük az algoritmus **kernelesített** (**kernelized**) változatát. Ez az átalakítás többek közt a k-legközelebbi-szomszéd algoritmusra és a perceptron tanulásra is könnyen elvégezhető.

[16] Itt ésszerű alatt azt értjük, hogy a $\mathbf{k}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ mátrix pozitív definit; lásd A) függelék.

Created by XMLmind XSL-FO Converter.

7. Esettanulmány: kézzel írott számjegyek felismerése

A kézzel írott számjegyek felismerése számos alkalmazásban felmerülő fontos probléma. Néhány ezek közül: postai levelek irányítószám szerinti automatikus osztályozása, csekkek és adó-visszatérítési számlák automatikus leolvasása, kézi számítógépek adatbevitele. Ez olyan terület, ahol gyors volt a fejlődés, részben a jobb tanuló algoritmusoknak, részben a jobb tanító adatbázisoknak köszönhetően. Az Egyesült Államok Mérés- és Szabványügyi Hivatala (National Institute of Standards and Technology, NIST) egy 60 000 megcímkézett számjegyből álló archívumot hozott létre, amelyben minden számjegy egy 20 × 20 = 400 pixeles, 8 bites szürke árnyalatú képen jelenik meg. Ez az új tanuló algoritmusok összehasonlításának egyik standard mércéjévé (benchmark) vált. A 20.29. ábra bemutat néhány számjegyet.

Számos különböző tanuló algoritmust kipróbáltak. Az elsők egyike, és egyben valószínűleg a legegyszerűbb, a **3-legközelebbi-szomszéd** (**3-nearest-neighbor**) osztályozó, amelynek további nagy előnye, hogy nem igényel tanítási időt. Ugyanakkor memóriaalapú algoritmusként mind a 60 000 képet tárolnia kell, és a futási idejű teljesítmény lassú. A teszthalmazon elért hibaaránya 2,4%.

Kifejlesztettek egy **egy-rejtett-rétegű neurális hálót** (**single-hidden-layer neural network**) a feladatra, 400 bemenettel (egy bemenet pixelenként) és 10 kimenettel (osztályonként egy). Keresztvalidációt használva azt találták, hogy durván 300 rejtett neuron adta a legjobb teljesítményt. A rétegek között teljes összeköttetést valósítottak meg, így összesen 123 300 súlyt használtak. Ez a háló 1,6% hibaarányt ért el.

LeNet néven egy sor **speciális neurális hálót** (**specialized neural network**) szerkesztettek egybe azért, hogy a problémában rejlő struktúrát kihasználják. Az egyik strukturális adottság, hogy a bemeneti pixelek kétdimenziós tömböt alkotnak, a másik, hogy egy ábra kismértékű pozíció- vagy dőlésiirány-változása lényegtelen. Minden egyes háló 32 × 32 egységből álló bemeneti réteget tartalmaz, amelyre a 20 × 20 pixelt úgy kapcsolták, hogy minden egyes bemeneti egység egy lokális szomszédságot reprezentált. Ezt három rejtett réteg követte. Mindegyik réteg számos $n \times n$-es mezőből áll, ahol n kisebb, mint a megelőző réteg, tehát a háló a bemenetek alul-mintavételezését valósítja meg. Egy mezőn belül mindegyik egységre azonos súlyokat kényszerítünk, így a mező tulajdonságdetektorként működik: olyan tulajdonságokat érzékel, mint egy hosszú függőleges vonal vagy egy rövid félkörív jelenléte. A kimeneti réteg 10 neuronból áll. Ennek az architektúrának számos változatát kipróbálták: egyik tipikus képviselőjük a rejtett rétegeiben rendre 768, 192, illetve 30 neuront tartalmazó háló volt. A tanító halmazt kiegészítették további mintákkal, ezeket a valósakból kismértékű affin transzformációkkal nyerték, amelyek a következők voltak: eltolás, kismértékű elforgatás, nagyítás/kicsinyítés. (Természetesen a transzformációknak kismértékűeknek kell lenniük, különben a 6-ot 9-cé transzformáljuk!) A LeNet által elért legkisebb hibaarány 0,9% volt.

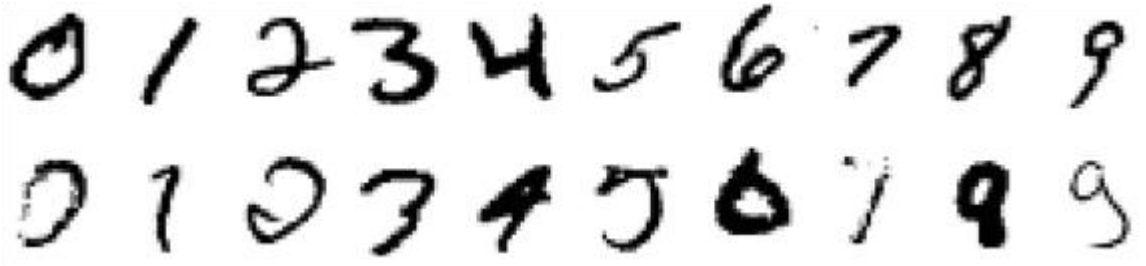
Egy **turbózott neurális háló** (**boosted neural network**) három LeNet-architektúrát kombinált: a másodikat egy olyan mintakeverékkel tanították, amelyen az első 50%-os hibát produkált, a harmadikat pedig olyan mintákkal, amelyekre az első kettő különböző eredményt adott. A tesztelés során a három háló a tíz lehetséges számjegy mindegyikére egy-egy súllyal szavazott, ezeket összeadva határozták meg a győztest. A teszthibaarány 0,7% volt.

Egy 25 000 szupport vektort használó **szupport vektor gép** (**support vector machine**) (lásd 20.6. alfejezet) 1,1% hibaarányt ért el. Ez azért figyelemre méltó, mert az SVM-technika – éppúgy mint az egyszerű legközelebbi-szomszéd megközelítés – szinte nem igényelt gondolkodást vagy iteratív kísérletezést a fejlesztőtől, mégis az évekig fejlesztett LeNet-et megközelítő eredményt ért el. A szupport vektor gépek valójában nem használják ki a probléma struktúráját, tehát ugyanolyan jól teljesítenének, ha a pixeleket valamilyen permutált sorrendben használnánk.

A **virtuális szupport vektor gépet** (**virtual support vector machine**) egy standard SVM-ként indítjuk, majd egy olyan technikával javítjuk, amelyet a probléma struktúrájának kihasználására hoztak létre. Ahelyett hogy az összes lehetséges pixelpárból képzett kernelt megengednénk, ez olyan kernelekre koncentrál, amelyek szomszédos pixelpárokból formálódtak. A LeNethez hasonlóan itt is a példák transzformáltjaival egészítjük ki a tanító halmazt. Egy virtuális SVM érte el az eddigi legjobb hibaarányt, 0,56%-ot.

20.29. ábra - Néhány példa az NIST kézzel írt számjegyekre létrehozott adatbázisából. A felső sorban a 0–9 karakterek könnyebben felismerhető példáit mutatjuk be, az alsó sorban pedig ugyanezen számjegyekre nehezebben felismerhető példákat látunk.

Created by XMLmind XSL-FO Converter.



Az **alakillesztés** (**shape matching**) a számítógépes látás egyik eljárása, amelyet arra hoztak létre, hogy tárgyak két különböző képének megfelelő részeit összerendeljék. (Lásd 24. fejezet.) Az alapötlet, hogy vegyünk egy-egy ponthalmazt mindkét képről, és számítsuk ki az első kép minden egyes pontjára, hogy a második kép mely pontja felel meg neki. Ebből a megfeleltetésből aztán számítsuk ki a két képet összekapcsoló transzformációt. Ez a transzformáció a képek közt egy távolságmértéket határoz meg. Ez a távolságmérték indokoltabb, mint egyszerűen az egymástól különböző pixelek megszámlálása. Kiderült, hogy az erre a távolságmértékre alapozott 3-legközelebbi-szomszéd algoritmus nagyon jól működik. Csupán 20 000-et használva a 60 000 számjegyből, és egy Canny éldetektor képének csupán 100 pixelét felhasználva az alakillesztéses eljárás 0,63% teszthibaarányt ért el.

Becslések szerint az **ember**ek (**human**) kb. 0,2% hibaarányt érnek el ennél a problémánál. Ez a szám némiképpen gyanús, mert az embereket nem tesztelték olyan alaposan, mint a gépi tanulás algoritmusait. Az Egyesült Államok Postájának (United States Postal Service) egy hasonló – számjegyekből képzett – adathalmazán az emberek által elkövetett hibaarány 2,5% volt.

A következő ábra összegzi a tárgyalt hét eljárás esetén a hibaarányt, a futási időt, a memóriaigényt és a szükséges tanítási időt. Még egy mértéket ad ezekhez, a számjegyek azon arányát, amelyet vissza kell utasítanunk, hogy a 0,5% hibaarányt elérjük. Ha például egy SVM-nek megengedjük, hogy bemeneti képeinek 1,8%-át visszautasítsa – azaz átpasszolhatja valakinek, hogy az hozza meg a végső döntést –, akkor a képek többi 98,2%-án a hibaarány 1,1%-ról 0,5%-ra csökken.

A következő táblázat a hét tárgyalt technikára összegzi a hibaarányt és néhány további jellemzőt.

	3 L-Sz	300 rejtett	LeNet	Turbózott LeNet	SVM	Virtuális SVM	Alak-illesztés
Hibaarány (%)	2,4	1,6	0,9	0,7	1,1	0,56	0,63
Futási idő (millisec/számjegy)	1000	10	30	50	2000	200	
Memóriaigény (MB)	12	0,49	0,012	0,21	11		
Tanítás ideje (nap)	0	7	14	30	10		
A 0,5% hiba eléréséhez a visszautasított %	8,1	3,2	1,8	0,5	1,8		

8. Összefoglalás

A statisztikai tanulás módszerei széles skálán helyezkednek el: az egyszerű átlagszámításoktól a bonyolult modellek – mint például a Bayes-hálók vagy neurális hálók – konstruálásáig. Alkalmazási területük a számítógép-tudományra, a mérnöki alkalmazásokra, a neurobiológiára, a pszichológiára és a fizikára is kiterjed. Ebben a fejezetben bemutattunk néhány alapelvet, és ízelítőt adtunk a matematikai tárgyalásból. A következő fő pontok szerepeltek:

- A **Bayes-tanulás**i (**Bayesian learning**) módszerek a tanulást valószínűségi következtetésként fogalmazzák meg, a megfigyelések alapján frissítve a hipotézisek a priori eloszlásait. Ez a megközelítés jó eszköz az Ockham borotvája elv megvalósítására, de bonyolult hipotézisterek esetén hamar kezelhetetlenné válik.

- A **maximum a posteriori** (**MAP**) tanulás az adatok alapján választ ki egyetlen, a legvalószínűbb hipotézist. A hipotézis priort itt is használjuk, ez a módszer gyakran jobban kezelhető, mint a tiszta Bayes-tanulás.

Created by XMLmind XSL-FO Converter.

- A **maximum-likelihood** tanulás egyszerűen azt a hipotézist választja, amely maximálja az adatok likelihood értékét. Megfelel egy egyenletes prior mellett végrehajtott MAP-tanulásnak. Egyszerű esetekben, mint a lineáris regresszió és a teljesen megfigyelhető Bayes-hálók, a maximum-likelihood megoldás könnyen előállítható zárt alakban. A **naiv Bayes-** (**naive Bayes**) tanulás különösen hatékony technika, amely különböző bonyolultságú feladatokra is jól illeszthető.

- Amikor a változók közt van néhány rejtett (nem megfigyelhető), akkor az **EM** algoritmussal lokális maximum-likelihood megoldásokat találhatunk. Az alkalmazások kiterjednek a kevert Gauss-jelek osztályozására, a Bayes-hálók tanulására és a rejtett Markov-modellek tanulására.

- A **modellválasztás**ra (**model selection**) ad példát a Bayes-hálók struktúrájának tanulása. Ez rendszerint egy, a struktúrák terében végzett diszkrét keresést foglal magában. Szükség van valamilyen módszerre a modell bonyolultsága és az illeszkedés mértéke közti kompromisszum létrehozásához.

- A **példányalapú modell**ek (**instance-based model**) a tanító példányok gyűjteményének eloszlását reprezentálják. Így a paraméterek száma a tanító halmaz méretével nő. A **legközelebbi-szomszéd** (**nearest neighbor**) módszerek a kérdéses mintapont közelében lévő példányokat nézik, míg a **kernel**módszerek az összes példány távolsággal súlyozott kombinációját.

- A **neurális hálók** (**neural networks**) nem mások, mint sok paraméterrel rendelkező, komplex nemlineáris függvények. Paramétereiket zajos adatok alapján tanulhatják meg. Több ezer alkalmazásban használták már őket.

- A **perceptron** egy előrecsatolt neurális háló, amelynek nincs rejtett rétege, és csak **lineárisan szeparálható** (**linearly separable**) függvények reprezentálására alkalmas. Ha az adatok lineárisan szeparálhatók, akkor egy egyszerű súlyfrissítési szabály alkalmazásával az adatokra való pontos illeszkedést tudunk elérni.

- A **többrétegű előrecsatolt** (**multilayer feed-forward**) neurális hálók – ha kellő számú neuronjuk van – tetszőleges függvények reprezentálására képesek. A **hiba-visszaterjesztés**i (**back-propagation**) algoritmus a kimeneti hiba minimalizálása érdekében gradiensalapú csökkentést valósít meg a paramétertérben.

A statisztikai tanulás továbbra is igen aktív kutatási terület. Mind az elmélet, mind a gyakorlat hatalmas lépésekkel haladt, míg elértünk addig a pontig, hogy szinte bármely modell megtanulható, ha megvalósítható rá egzakt vagy közelítő következtetés.

8.1. Irodalmi és történeti megjegyzések

Az MI korai éveiben a statisztikai tanuláselmélet a kutatás aktívan művelt területe volt (Duda és Hart, 1973), de elkülönült az MI fősodrától, ahogy ez utóbbi egyre inkább a szimbolikus módszerekre koncentrált. Különböző formákban folytatódott – egyesek explicit módon valószínűségiek, mások nem – olyan területeken, mint az **alakzatfelismerés** (**pattern recognition**) (Devroye és társai, 1996) és az **információkeresés** (**information retrieval**) (Salton és McGill, 1983). Nem sokkal a Bayes-háló modellek bevezetése után, az 1980-as évek végén az érdeklődés ismét erősen ráirányult, nagyjából ugyanebben az időben jelent meg a neurális hálók statisztikai megközelítése. Az 1990-es évek végén a gépi tanulás, a statisztika és a neuronhálók területén is az érdeklődés középpontjába kerültek az adatok alapján nagy valószínűségi modelleket létrehozó módszerek.

A naiv Bayes-modell egyike a legrégebbi és legegyszerűbb Bayes-hálóknak, megjelenése egészen az 1950-es évekig követhető vissza. Eredetüket megemlítettük a 13. fejezet záró megjegyzéseiben. Részleges magyarázat található Domingos és Pazzani publikációjában (Domingos és Pazzani, 1997). A naiv Bayes-tanulás turbózott változata nyerte az első KDD Cup adatbányászati versenyt (Elkan, 1997). Heckerman kitűnő bevezetését adja a Bayes-háló tanulás általános problematikájának (Heckerman, 1998). Spiegelharter és társai a Bayes-hálók Bayesi paramétertanulását tárgyalták Dirichlet-priorok esetére (Spiegelharter és társai, 1993). A `BUGS` szoftvercsomag (Gilks és társai, 1994) számos gondolatot megtestesített ezek közül, nagyon hatékony eszközt biztosított az összetett valószínűségi modellek felállítására és tanulására. A Bayes-hálóstruktúra tanulásának első algoritmusai feltételes függetlenségi teszteket használtak (Pearl, 1988; Pearl és Verma, 1991). Spirtes és társai hasonló elvek alapján dolgozták ki átfogó megközelítésüket, valamint a Tetrad csomagot Bayes-hálóstruktúra tanulás céljaira (Spirtes és társai, 1993). Az azóta végrehajtott algoritmikus javítások a 2001-es KDD Cup adatbányászati versenyen egy Bayes-háló tanulási algoritmus (Cheng és társai, 2002) meggyőző győzelméhez vezettek. (Itt a speciális megoldandó feladat egy 139 351 tulajdonsággal leírt bioinformatikai probléma volt!) Cooper és Herskovits egy likelihood maximalizáláson alapuló struktúratanulási megközelítést fejlesztett ki (Cooper és Herskovits, 1992), ezt Heckerman és társai fejlesztették tovább (Heckerman és társai,

Created by XMLmind XSL-FO Converter.

1994). Friedman és Goldszmidt mutatták ki a lokális feltételes eloszlások reprezentációjának a megtanult struktúrára gyakorolt hatását (Friedman és Goldszmidt, 1996).

A rejtett változókkal és a hiányzó adatokkal való valószínűségi modell tanulás általános problematikáját az EM algoritmussal kísérelték meg kezelni (Dempster és társai, 1977). Ezt számos meglévő módszerből absztrahálták, amelyek közt található a rejtett Markov-modell (HMM) tanulásra szolgáló Baum–Welch-algoritmus is (Baum és Petrie, 1966). (Maga Dempster az EM algoritmust inkább sémának tekinti, nem algoritmusnak, mivel jó adag elméleti matematikai munkára lehet szükség mielőtt egy új eloszláscsaládra alkalmazható lenne.) Manapság az EM egyike a tudományos kutatásban legelterjedtebben használt algoritmusoknak, McLachlan és Krishnan egy teljes könyvet szenteltek neki és tulajdonságainak (McLachlan és Krishnan, 1997). A kevert modellek – beleértve a kevert Gauss-modellek – tanulásának speciális problémáit Titterington és társai tárgyalták (Titterington és társai, 1985). Az AUTOCLASS volt az első sikeres rendszer az MI-n belül, amely az EM-et alkalmazta kevert modellezésre (Cheeseman és társai, 1988; Cheeseman és Stutz, 1996). Az AUTOCLASS-t egy sor valós tudományos osztályozási feladatra alkalmazták; ezek közül kettő: spektrális tulajdonságok alapján új csillagtípusok felfedezése (Goebel és társai, 1989); új fehérje- és intronosztályok felfedezése DNS/fehérjeszekvencia adatbázisokban (Hunter és States, 1992).

A rejtett változókkal rendelkező Bayes-hálók tanulására kifejlesztett EM algoritmus Lauritzen munkája (Lauritzen, 1995). Mind a Bayes-hálók, mind a dinamikus Bayes-hálók esetén a gradiensalapú eljárások is hatékonynak bizonyultak (Russell és társai, 1995; Binder és társai, 1997a). A strukturális EM algoritmus kifejlesztése Friedman nevéhez fűződik (Friedman, 1998). A Bayes-hálók struktúrájának megtanulhatósága szoros kapcsolatban van a *kauzális* kapcsolatok adatokból történő visszanyerésének kérdésével. Azaz lehetséges-e Bayes-hálókat úgy megtanulni, hogy az előállított hálóstruktúra valós kauzális hatásokat jelezzen? A statisztikusok hosszú évek óta elkerülték ezt a kérdést, azt hitték, hogy a megfigyelt adatok (ellentétben a kísérletek során előálltakkal) csak korrelációs információt hordoznak. Végül is bármely két változóra, amelyek egymással kapcsolatban állónak tűnnek, lehet, hogy valójában inkább egy harmadik – mindkettőre kauzális hatást gyakorló – ismeretlen változó hatása alatt állnak, nem pedig egymásra gyakorolnak közvetlen hatást. Ennek ellenkezőjére Pearl adott meggyőző érveket (Pearl, 2000). Megmutatta, hogy valójában számos eset van, amikor a kauzalitás kideríthető, és **kauzális háló** (**causal network**) formalizmus alakítható ki az oksági kapcsolatok, a beavatkozás hatásának, valamint a szokásos feltételes valószínűségek beépítésére.

A legközelebbi-szomszéd modellek legalább Fix és Hodges (Fix és Hodges, 1951) munkájáig nyúlnak vissza, és azóta a statisztika és alakfelismerés standard eszközei. Az MI-n belül Stanfill és Waltz népszerűsítették ezeket a modelleket (Stanfill és Waltz, 1986), ők a távolságmetrika adatokhoz történő adaptálási módszereivel foglalkoztak. Hastie és Tibshirani kifejlesztettek egy módszert, amellyel a tér egyes pontjaihoz kötötték az ezen pont körüli adateloszlástól függő metrikát (Hastie és Tibshirani, 1996). A legközelebbi-szomszédok hatékony indexelési sémával történő megtalálásával az algoritmusokat kutató közösség foglalkozott (pl. Indyk, 2000). A kernelsűrűség-becslést, amelyet **Parzen ablak** (**Parzen window**) sűrűségbecslésnek is neveznek, kezdetben Rosenblatt és Parzen tanulmányozta (Rosenblatt, 1956; Parzen, 1962). Azóta óriási az irodalma a különböző becslők tulajdonságai vizsgálatának. Devroye alapos bevezetést nyújt ehhez a témához (Devroye, 1987).

A neurális hálók irodalma túl nagy ahhoz (napjainkig kb. 100 000 publikáció), hogy részletesen bemutathassuk. A korai fejleményekről Cowan és Sharp készített összefoglalót (Cowan és Sharp, 1988b; 1988a), McCulloch és Pitts munkásságával kezdve az áttekintést (McCulloch és Pitts, 1943). Norbert Wiener – a kibernetika és a szabályozáselmélet egyik úttörője (Wiener, 1948) – együtt dolgozott McCullochkal és Pitts-szel, és nagy hatást gyakorolt egy sor fiatal kutatóra, például Marvin Minskyre, aki valószínűleg elsőként fejlesztett ki működő neurálisháló-hardvert 1951-ben (Minsky és Papert, 1988, pp. ix–x.) Ezalatt Nagy-Britanniában W. Ross Ashby (szintén a kibernetika egyik úttörője; Ashby, 1940), Alan Turing, Grey Walter és mások megalakították a Ráció Klubot (Ratio Club) azok számára, akik „rájöttek Wiener gondolataira, még mielőtt Wiener könyve megjelent”. Ashby *Az agy felépítése* (*Design for a Brain*, 1948, 1952) című könyvében vetette fel, hogy stabil adaptív viselkedést létrehozó alkalmas visszacsatoló hurkokkal rendelkező **homeostatikus**[17] (**homeostatic**) eszközök segítségével intelligenciát lehetne létrehozni. Turing egy kutatási jelentést írt *Intelligens Gépek* (*Intelligent Machinery*) címen (Turing, 1948), amely a következő mondattal kezdődik: „Javaslom megvizsgálni azt a kérdést, hogy vajon lehetséges-e az, hogy a gépek intelligens viselkedést mutassanak”, majd leírja a rekurrens neurális hálózatokat, amelyeket „B típusú nem szervezett gépek” néven vezet be, és megadja tanításuknak egy lehetséges megközelítését. Sajnos ezt a jelentést 1969-ig nem is publikálták, és napjainkig lényegében figyelmen kívül hagyták.

[17] *Homeosztázis*: egy szervezet különböző, de egymással kölcsönhatásban álló elemei vagy elemek csoportjai között fennálló stabil egyensúly vagy egyensúly felé való törekvés. Gyakran biológiai összefüggésben használt kifejezés. (*A ford.*)

Created by XMLmind XSL-FO Converter.

Frank Rosenblatt (Rosenblatt, 1957) nevéhez fűződik a modern „perceptron” felfedezése, és ő bizonyította be a perceptronkonvergencia tételt (Rosenblatt, 1960), bár ezt már a neurális hálók területén kívül eső, tisztán matematikai munkák is előrevetítették (Agmon, 1954; Motzkin és Schoenberg, 1954). Volt némi, a többrétegű hálózatokra irányuló korai kutatás is, amelynek eredményei például a **Gamba-perceptron**ok (Gamba és társai, 1961) és a **madaline**-ok (Widrow, 1962). A *Learning Machines* (Nilsson, 1965) c. könyv áttekintést ad a korai kutatás legnagyobb részéről. A korai perceptronkutatások halálát siettette – a szerzők későbbi állítása szerint csak magyarázta – a *Perceptrons* c. könyv (Minsky és Papert, 1969), amelyben a terület matematikai precizitásának hiányát panaszolták fel. A könyv rámutatott, hogy egyrétegű perceptronokkal csak lineárisan szeparálható helyzetek reprezentálhatók, és felhívta a figyelmet a többrétegű hálók hatékony tanuló algoritmusainak hiányára.

A San Diegóban, 1979-ben tartott konferencia publikációira alapozott kiadvány (Hinton és Anderson, 1981) tekinthető a konnekcionizmus újjáéledése jelének. Nagy figyelmet keltett a kétrészes „PDP” (Párhuzamos elosztott feldolgozás – Parallel Distributed Processing) antológia (Rumelhart és társai, 1986a), illetve a *Nature*-ben megjelent rövid cikk (Rumelhart és társai, 1986b). A neurális hálókkal foglalkozó cikkek száma az 1980–1984 közötti publikációk számáról 200-szorosára nőtt 1990–1994-re. A mágneses spin üvegek fizikai elméletének felhasználásával elvégzett neurális háló analízis (Amit és társai, 1985) nem csupán szorosabb kapcsolatot hozott a statisztikus mechanika és neurális hálók elmélete közt, hanem *tekintélyt* is adott a területnek. A hiba-visszaterjesztés (back-propagation) technikáját viszonylag hamar kitalálták (Bryson és Ho, 1969), de több alkalommal újra felfedezték (Werbos, 1974; Parker, 1985).

Az 1990-es években megjelent szupport vektor gépeknek (Cortes és Vapnik, 1995) napjainkban gyorsan növekvő az irodalmuk, amely olyan tankönyveket is magában foglal, mint Cristianini és Shawe-Taylor könyve (Cristianini és Shawe-Taylor, 2000). Nagyon népszerűnek és bizonyos feladatokra nagyon hatékonynak bizonyultak, ilyenek például a szövegkategorizálás (Joachims, 2001), a bioinformatikai kutatás (Brown és társai, 2000), a természetes nyelvű szöveg feldolgozása, mint a kézzel írt számjegyek DeCoste és Schölkopf által megvalósított felismerése (DeCoste és Schölkopf, 2002). A szavazó perceptron szintén egy olyan technika, amely a kerneltrükköt alkalmazza az exponenciális tulajdonságtér implicit reprezentációjára (Collins és Duffy, 2002).

Számos forrás adható meg a neurális hálók valószínűségi interpretációjára, például (Baum és Wilczek, 1988), valamint (Bridle, 1990). A szigmoid függvény szerepét Jordan tárgyalja (Jordan, 1995). MacKay javasolta a neurális hálók Bayes-i paraméterbecslését (MacKay, 1992), amelyet Neal fejlesztett tovább (Neal, 1996). A neurális hálók függvényreprezentációs képességeit Cybenko vizsgálta (Cybenko, 1988; 1989), aki megmutatta, hogy két rejtett réteg elegendő tetszőleges függvény reprezentációjához, tetszőleges *folytonos* függvény reprezentációjához pedig elég egy réteg. A haszontalan összeköttetések eltávolítását célzó „optimális agykárosodás” módszer LeCun és társai eredménye (LeCun és társai, 1989), míg Sietsma és Dow mutatták meg, hogyan kell a felesleges neuronokat eltávolítani (Sietsma és Dow, 1988). A nagyobb struktúrák növesztéssel való előállítására szolgáló csempézési algoritmus Mézard és Nadal munkája (Mézard és Nadal, 1989). A kézzel írt számjegyek felismerésével foglalkozó algoritmusokról LeCun és társai írtak áttekintő publikációt (LeCun és társai, 1995). Azóta jobb hibaarányt értek el Belongie és társai (2002) az alakillesztési eljárás (Belongie és társai, 2002), valamint DeCoste és Schölkopf a virtuális szupport gép alkalmazásával (DeCoste és Schölkopf, 2002).

A számítógépes tanulás elmélete területén tevékenykedő kutatók foglalkoztak a neurális háló tanulás komplexitásával. Az első számítási eredményeket Judd kapta (Judd, 1990), aki megmutatta, hogy egy példahalmazzal konzisztens súlyhalmaz megtalálásának általános problémája – még nagyon erősen korlátozó feltételek esetén is – NP-teljes. A mintakomplexitásra vonatkozó eredmények közül néhány Baum és Haussler munkájához fűződik, akik megmutatták, hogy W súly esetén a hatékony tanításhoz szükséges mintaszám $W \log W$ arányában nő (Baum és Haussler, 1989).[18] Azóta Anthony és Bartlett egy sokkal fejlettebb elméletet fejlesztettek ki (Anthony és Bartlett, 1999), amely magában foglalja azt a fontos eredményt, hogy a háló reprezentációs képessége a súlyok *nagyságrendjétől* és számától egyaránt függ.

Az általunk nem tárgyalt legnépszerűbb neurális háló a **radiális bázisfüggvény** (**RBF**) (**radial basis function**) háló. A radiális bázisfüggvény kernelek súlyozott kombinációját (természetesen rendszerint Gauss-kernelekét) alkalmazza függvényapproximációra. Az RBF-hálókat két fázisban taníthatjuk: először egy nem ellenőrzött osztályozással tanítjuk a Gauss-függvények paramétereit (az átlagokat és varianciákat), mint a 20.3. alfejezetben láttuk. A második fázisban a Gauss-függvények relatív súlyát határozzuk meg. Ez egy lineáris egyenletrendszer

[18] Ez nagyjából igazolta „Bernie bácsi” törvényét: a szabályt Bernie Widrowról nevezték el, aki azt javasolta, hogy nagyjából tízszer annyi mintát használjanak, mint a súlyok száma.

Created by XMLmind XSL-FO Converter.

megoldása, amiről tudjuk, hogy közvetlenül hogyan oldható meg. Így az RBF-tanítás mindkét fázisának vonzó tulajdonságai vannak: az első fázis nem ellenőrzött, tehát nincs szükségünk hozzá címkézett mintákra, a második ugyan felügyelt, de hatékonyan elvégezhető. A részleteket lásd Bishop publikációjában (Bishop, 1995).

A **rekurrens hálók**at (**recurrent network**), amelyekben a neuronok hurkokba vannak kapcsolva, említettük ugyan a fejezetben, de nem részleteztük. A legjobban megértett rekurrens háló osztályt valószínűleg a **Hopfield-hálók** (**Hopfield networks**) (Hopfield, 1982) alkotják. A Hopfield-háló kétirányú kapcsolatokat használ *szimmetrikus* súlyokkal (azaz $W_{i,j} = W_{j,i}$), minden neuron bemeneti és egyben kimeneti egység is, a g aktivációs függvény az előjelfüggvény, az aktivációs szint pedig csak a ±1 lehet. A Hopfield-háló **asszociatív memória**ként működik: miután egy mintahalmazon tanítottuk, egy új bemeneti stimulus hatására egy olyan aktivációs mintára áll be, amely a tanító példák közül a *leginkább emlékeztet* az új bemenetre. Ha például a tanító halmaz egy halom fénykép, az új bemenet pedig az egyik fénykép egy kis darabja, akkor aktivációs szintjeivel a háló vissza fogja állítani az adott darabból a fényképet. Figyeljük meg, hogy a fényképek nincsenek elkülönülten tárolva a hálóban: mindegyik súly az összes fénykép egy részleges kódja. Érdekes elméleti eredmény, hogy az N neuronból álló Hopfield-háló legfeljebb $0{,}138N$ tanító mintát képes megbízhatóan tárolni.

A **Boltzmann-gép**ek is szimmetrikus súlyokat használnak, de tartalmaznak rejtett egységeket is (Hinton és Sejnowski, 1983; 1986). Ráadásul aktivációs függvényük *sztochasztikus:* a kimenet 1 értékének valószínűsége a teljes súlyozott bemenetnek valamilyen függvénye. A Boltzmann-gépek ennek megfelelően a szimulált lehűtés keresési eljárásra (lásd 4. fejezet) emlékeztető állapotátmenetek során jutnak el a tanító halmazt a legjobban közelítő konfigurációhoz. Kimutatható, hogy a Boltzmann-gépek nagyon közeli rokonságban vannak bizonyos speciális Bayes-hálókkal, amelyeket sztochasztikus szimulációs algoritmussal értékelünk ki. (Lásd 14.5. alfejezet.)

A kernelgépek alapötletének első alkalmazása Aizerman és társai munkájához fűződik (Aizerman és társai, 1964, de a szupport vektor gépek felé mutató, teljes elméleti kidolgozás Vlagyimir Vapniknak és kollégáinak köszönhető (Boser és társai, 1992; Vapnik, 1998). A precíz elméleti tárgyalás megtalálható Cristianini és Shawe-Taylor, illetve Schölkopf és Smola könyvében (Cristianini és Shawe-Taylor, 2000; Schölkopf és Smola, 2002), valamint egy barátságosabb kifejtés található Cristianini és Schölkopf *AI Magazine*-ban megjelent cikkében.

Ennek a fejezetnek az anyaga a statisztika, alakfelismerés és neurális hálók kutatásában elért eredményeket integrálja, tehát a történetet sokszor, sokféle módon elmondták már. A Bayes-statisztikáról jó publikációkat találunk, ilyenek például a (DeGroot, 1970), a (Berger, 1985), továbbá a (Gelman és társai, 1995). A statisztikus tanulás elméletének kiváló tárgyalása található a (Hastie és társai, 2001) munkában. Az alakosztályozás témakörében Duda és Hart könyve számít klasszikusnak (Duda és Hart, 1973), melynek nemrég új kiadása is megjelent (Duda és társai, 2001). A neurális hálókról a (Bishop, 1995) és a (Ripley, 1996) a legfontosabb könyvek. A számítógépes neurális tudományt a (Dayan és Abbott, 2001) könyv tárgyalja. A neurális hálók és hozzájuk kapcsolódó témák legfontosabb konferenciái az évente megrendezett NIPS- (Neural Information Processing Conference) konferenciák, melyek kiadványait az *Advances in Neural Information Processing Systems* sorozatban jelentetik meg. A Bayes-hálók tanulásáról szóló írások az *Uncertainty in AI* és a *Machine Learning* konferenciákon jelennek meg, továbbá számos statisztikával foglalkozó konferencián. A neurális hálókkal foglalkozó újságok között érdemes megemlíteni a *Neural Computation*t, a *Neural Networks*öt és az *IEEE Transaction on Neural Networks*öt.

8.2. Feladatok

20.1.

A 20.1. ábrán használt adatokat úgy tekinthetjük, mintha h_5 generálta volna őket. A másik négy hipotézis mindegyikére generáljon egy-egy 100 hosszúságú adathalmazt, rajzolja fel a megfelelő $P(h_i|d_1,\ldots, d_m)$ és $P(D_{m+1} = citrom|d_1,\ldots, d_m)$ görbéket! Értékelje az eredményeket.

20.2.

Ismételje meg a 20.1. feladatot úgy, hogy a $P(D_{m+1} = citrom|h_{map},\ldots, d_m)$, illetve $P(D_{m+1} = citrom|h_{ml})$ görbéket rajzolja fel.

20.3.

Created by XMLmind XSL-FO Converter.

Tegyük fel, hogy Anna számára a meggycukorkák hasznossága c_A, a citromcukorkáké ℓ_A, míg Béla számára a hasznosságok rendre c_B és ℓ_B. (De ha Anna kibontott egy cukorkát, akkor azt Béla már nem fogja megvásárolni tőle.) Ha Béla sokkal jobban szereti a citromízű cukorkát, mint Anna, akkor Anna feltehetőleg bölcsen teszi, ha eladja a cukros zacskóját, amikor már kellően biztos a citromízűek arányában. Másrészről viszont, ha Anna túl sok cukrot kibontott már az arány eldöntése során, akkor a zacskó értéktelenné válik. Vizsgálja meg a cukroszacskó-eladás optimális pillanatának meghatározását. Határozza meg az optimális eljárás várható hasznosságát, a 20.1. alfejezet a priori eloszlásainak feltételezésével.

20.4.

Két statisztikus elmegy az orvoshoz, és mindkettőről ugyanazt a diagnózist állítja fel az orvos: 40% eséllyel a halálos *A* betegségben, 60% eséllyel a szintén végzetes *B* betegségben szenvednek. Szerencsére van az *A* és *B* betegségnek is olcsó, 100%-ban hatásos, mellékhatás nélküli gyógyszere. A statisztikusoknak lehetőségük van az *A* elleni, a *B* elleni vagy mindkét gyógyszert szedni, de dönthetnek úgy is, hogy egyiket sem szedik. Mit fog az első statisztikus – aki elszánt Bayes-hívő – választani? Mit tesz a második, aki mindig a maximum-likelihood hipotézist választja?

Az orvos némi kutatás után felfedezi, hogy a *B* betegségnek két változata van, a dextro-*B* és a levo-*B*, amelyek egyforma valószínűséggel lépnek fel, és egyformán jól kezelhetők az anti-*B* gyógyszerrel. Mit fog csinálni a két statisztikus most, hogy három hipotézis van?

20.5.

Magyarázza meg, hogyan alkalmazható a 18. fejezetben tárgyalt turbózás módszere a naiv Bayes-tanulásra. Tesztelje a kapott algoritmust az étterem tanulási problémán.

20.6.

Vegyen m darab (x_j, y_j) adatpontot, az x_j-ből a (20.5) egyenlet alapján generálja y_j-t. Keresse meg azokat a θ_1, θ_2 és σ értékeket, amelyek maximálják az adatok feltételes log likelihood értékét.

20.7.

Vizsgálja a láz 14.3. alfejezetben bemutatott zajos-VAGY modelljét. Magyarázza meg, hogyan alkalmazható a maximum-likelihood tanulás arra, hogy egy teljes adathalmazra illesszük egy ilyen modell paramétereit. (*Segítség:* a parciális deriváltakra használja a láncszabályt.)

20.8.

Ebben a feladatban a (20.6) egyenlettel definiált béta-eloszlások tulajdonságait vizsgáljuk.

a. A [0, 1] tartomány feletti integrálás alapján mutassa meg, hogy a béta$[a,b]$ eloszlás normalizáló faktorát $\alpha = \Gamma(a + b)/ \Gamma(a) \Gamma(b)$ adja, ahol $\Gamma(x)$ az úgynevezett **gamma-függvény**, amelynek definíciója: $\Gamma(x + 1) = x \times \Gamma(x)$ és $\Gamma(1) = 1$. (Egész x-ekre $\Gamma(x + 1) = x!$)

b. Mutassa meg, hogy az átlagérték $a/(a + b)$.

c. Állítsa elő a modus(oka)t (a legvalószínűbb θ értéke(ke)t).

d. Jellemezze a béta$[\varepsilon, \varepsilon]$ eloszlást nagyon kis ε esetén. Mi történik, ha egy ilyen eloszlást frissítünk?

20.9.

Vegyen egy tetszőleges Bayes-hálót, egy ehhez a hálóhoz tartozó teljes adathalmazt és az adathalmaznak a háló által megadott likelihood-értékét. Adjon egyszerű bizonyítást arra, hogy az adathalmaz likelihood-értéke nem csökkenhet, ha a hálóhoz hozzáadunk egy új kapcsolatot, majd újraszámoljuk a maximum-likelihood paramétereket.

20.10.

Vizsgálja meg az EM alkalmazását arra a problémára, amikor a 20.10. (a) ábrán látható háló paramétereit akarjuk tanulni, miközben az igazi paraméterértékeketa (20.7) egyenlet adja.

Created by XMLmind XSL-FO Converter.

a. Magyarázza meg, hogy miért nem működne az EM algoritmus, ha csak két attribútum volna, és nem három.

b. Végezze el az első iterációra vonatkozó számítást a (20.8) egyenletből kiindulva.

c. Mi történik, ha induláskor az összes paramétert ugyanarra a p értékre állítjuk? (*Segítség:* hasznos lehet, ha empirikusan megvizsgálja, mielőtt az általános eredményt megpróbálná levezetni.)

d. Írja fel a 3.2. szakasz - Rejtett változókkal felépített Bayes-hálók tanulása részben található adattáblázatra a log likelihoodnak a paraméterekkel kifejezett összefüggését. Számítsa ki minden egyes paraméterre a parciális derivált értékét! Vizsgálja meg a (c) részben rögzített pont jellegét.

20.11.

Hozzon létre manuálisan egy olyan neurális hálót, amely két bemenetének az xor függvényét számítja ki. Bizonyosodjon meg arról, helyesen specifikálta-e azt, hogy milyen egységeket kell használni.

20.12.

Hozzon létre egy olyan szupport vektor gépet, amely az xor függvényt számítja ki. Kényelmesebb lesz, ha mind a bemeneteknél, mind a kimenetnél –1-et és +1-et használ 0 és +1 helyett. Ennek megfelelően egy minta például ([–1, 1], 1) vagy ([–1, –1], –1) lesz. Tipikus, hogy az **x** bemenetet egy ötdimenziós térre képezzük le, ahol kettő az eredeti x_1 és x_2 dimenzió, a másik három pedig x_1^2, x_2^2, illetve x_1 x_2 kombinációi. Ebben a feladatban viszont csak két dimenziót használunk: x_1-et és x_1 x_2-t. Rajzolja fel ebben a térben a négy bemeneti pontot és a maximális tartalékkal rendelkező osztályozót. Mekkora a tartalék? Rajzolja meg a határoló vonalat az eredeti, euklideszi térben.

20.13.

Egy egyszerű perceptron nem képes az xor függvény (vagy általánosabban a paritásfüggvény) ábrázolására. Mutassa be, hogy hogyan alakulnak egy négybemenetű, ugrásfüggvényt használó perceptron súlyai, amikor sorban a paritásfüggvényből származó minták érkeznek (kezdetben minden súly 0,1 értékű volt).

20.14.

Idézzük fel a 18. fejezetből, hogy n bemenet esetén 2^2 különböző logikai függvény létezik. Hányat tud reprezentálni ezek közül az ugrásfüggvényt használó perceptron?

20.15.

Vegyük a következő – hat bemeneti értéket (I_1–I_6) és egy, ezekhez tartozó kívánt kimeneti értéket (T) megadó – mintahalmazt.

I_1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
I_2	0	0	0	1	1	0	0	1	1	0	1	0	1	1
I_3	1	1	1	0	1	0	0	1	1	0	0	0	1	1
I_4	0	1	0	0	1	0	0	1	0	1	1	1	0	1
I_5	0	0	1	1	0	1	1	0	1	1	0	0	1	0
I_6	0	0	0	1	0	1	0	1	1	0	1	0	1	0
T	1	1	1	1	1	1	0	1	0	0	0	0	0	0

a. Futassa a perceptron tanulási szabályt erre az adathalmazra, és adja meg a végső súlyokat.

b. Futassa a döntési fa tanulási algoritmust, és adja meg az eredményül kapott döntési fát.

c. Értékelje az eredményeket.

20.16.

Created by XMLmind XSL-FO Converter.

A (20.13) egyenletből kiindulva mutassa meg, hogy $\partial L/\partial W_j = Err \cdot x_j$.

20.17.

Tegyük fel, hogy egy lineáris aktivációs függvényeket használó neurális hálónk van. Azaz minden neuron kimenete a bemenetek súlyozott összegének c-szerese, ahol c egy konstans.

a. Tegyük fel, hogy a hálónak egy rejtett rétege van. Írja fel a súlyok egy adott **W** értékkészletére azokat az egyenleteket, amelyek megadják a kimeneti réteg neuronjainak kimeneti értékeit **W** és a bemeneti réteg **I** értékeinek függvényében – anélkül hogy explicit módon megjelennének az egyenletekben a rejtett neuronok kimeneti értékei. Mutassa meg, hogy létezik olyan háló, amelynek nincs rejtett rétege, de ugyanezt a függvényt valósítja meg.

b. Ismételje meg az (a) rész alatti feladatot, de tetszőleges számú rejtett réteg esetére. Milyen következtetést vonhat le a lineáris aktivációs függvényekre?

20.18.

Hozzon létre egy adatstruktúrát rétegekbe szervezett, előrecsatolt neurális hálók leírására, amely tartalmazza az előreterjesztéshez és a hiba-visszaterjesztéshez szükséges információt is. Ezt használva írjon egy `NEURÁLIS-HÁLÓ-KIMENET` nevű programot, amely egy mintát és egy neurális hálót kap bemenetként, és kiszámítja a neurális hálónak a mintára adott kimeneti válaszát.

20.19.

Tegyük fel, hogy egy tanító halmaz csupán egyetlen példát tartalmaz, de azt 100-szor. A 100 esetből 80-ban az egyetlen kimeneti érték 1; a másik 20-ban 0. Mit ad erre a példára egy hiba-visszaterjesztéssel tanított háló, ha tanítottuk, és elérte a globális optimumot? (*Segítség:* a globális optimum megtalálásához differenciálja a hibafüggvényt, és keresse a nullahelyét.)

20.20.

A 20.24. ábrán látható hálónak négy rejtett neuronja van. Ezt a neuronszámot némiképpen ötletszerűen választottuk. Végezzen szisztematikus kísérleteket, hogy különböző számú rejtett neuronnal rendelkező hálókra lemérje a tanulási görbét. Mi az optimális neuronszám? Lehetséges lenne keresztvalidációs módszerrel megtalálni a legjobb hálót még a kísérletek előtt?

20.21.

Vegyük azt a problémát, amikor N adatpontot akarunk lineáris osztályozóval szétválasztani pozitív és negatív példákra. Nyilvánvaló, hogy $N = 2$ pont esetén ez egy $d = 1$ dimenziós egyenesen mindig megtehető, függetlenül attól, hogy a pontok hol helyezkednek el, és hogy vannak címkézve (kivéve, ha ugyanazon a helyen vannak).

a. Mutassa meg, hogy a szeparálás mindig elvégezhető $N = 3$ pontra egy $d = 2$ dimenziós síkon, ha a pontok nem egy egyenesre esnek.

b. Mutassa meg, hogy nem lehet mindig elvégezni a szeparálást $N = 4$ pontra egy $d = 2$ dimenziós síkon.

c. Mutassa meg, hogy a szeparálás mindig elvégezhető $N = 4$ pontra egy $d = 3$ dimenziós térben, ha a pontok nem esnek egy síkra.

d. Mutassa meg, hogy nem lehet mindig elvégezni a szeparálást $N = 5$ pontra egy $d = 3$ dimenziós térben.

e. Egy törekvő diák kitűzheti maga elé, hogy bebizonyítja: $N - 1$ dimenziós térben általános helyzetben található N pont lineárisan szeparálható (de $N + 1$ nem). Ebből következik, hogy az $N - 1$ dimenziós lineáris félterek **VC-dimenzió**ja (lásd 18. fejezet) N.

Created by XMLmind XSL-FO Converter.

21. fejezet - Megerősítéses tanulás

Ebben a fejezetben azt vizsgáljuk, hogy egy ágens hogyan képes tanulni a sikereiből és a kudarcaiból, illetve a jutalomból és a büntetésből.

1. Bevezetés

A 18. és 20. fejezetben olyan tanuló módszereket mutattunk be, amelyek példák alapján függvényeket vagy valószínűségi modelleket sajátítanak el. Ebben a fejezetben azzal foglalkozunk, hogy az ágens mi módon tanulhatja meg azt, hogy *mit tegyen,* különösen akkor, amikor nincs tanár, aki minden előforduló körülmény esetére elárulná a helyes cselekvést.

Fontos

Vegyük példaként a sakkot. Tudjuk, hogy az ágens képes megtanulni sakkozni felügyelt tanulással, vagyis ha példaként sakkállásokat mutatunk neki, megnevezve az adott állásban a legjobb lépést. De mit tud csinálni az ágens akkor, ha nincs kéznél egy barátságos tanító, aki példákat mutatna? Véletlen lépések kipróbálásával végül is az ágens felépítheti környezetének prediktív modelljét: milyen lesz az állás, ha meglép egy adott lépést, esetleg még azt is modellezheti, hogy az ellenfél várhatóan mit fog válaszul lépni. A következő azonban a probléma: *visszacsatolás nélkül – tehát anélkül, hogy tudná, mi a jó és mi a rossz – az ágensnek semmilyen alapja sem lesz eldönteni azt, hogy melyik lépést húzza meg*. Az ágensnek tudnia kell, hogy valami jó történt, amikor nyert, illetve valami rossz történt, amikor vesztett. Az ilyen típusú visszacsatolást nevezzük **jutalom**nak (**reward**) vagy **megerősítés**nek (**reinforcement**). A sakkhoz hasonló játékokban a megerősítés csak a játék végén jelenik meg. Más környezetben a jutalom gyakrabban jön, például a pingpongban mindegyik pont jutalomnak tekinthető éppúgy, mint a mászás tanulásakor bármely előrefelé haladás. Az általunk használt tárgyalásmódban az ágens a jutalmat is az észlelés *részeként* kezeli, de valahogy bele kell lennie „huzalozva”, hogy ez a rész jutalom, és nem csupán egy észlelésfajta. Úgy tűnik, hogy az állatokba be van építve, hogy a fájdalom és az éhség negatív „jutalom”, míg a gyönyör és az élelem pozitív jutalom. A megerősítéses tanulást az állatok pszichológiáját kutatók 60 éve intenzíven tanulmányozzák.

A jutalom fogalmát a 17. fejezetben vezettük be, ahol a **Markov döntési folyamat**ok (**Markov decision process**, **MDP**) optimalitásának definiálására szolgált. Optimális stratégiának neveztük azt a stratégiát, amely maximálja a várható összjutalmat. A **megerősítéses tanulás** (**reinforcement learning**) feladata az, hogy optimális (vagy közel optimális) stratégiát tanuljunk az adott környezethez. Míg a 17. fejezetben az ágensnek a teljes környezeti modell és a jutalomfüggvény a rendelkezésére állt, addig a jelen fejezetben egyikről sem feltételezünk semmilyen előzetes tudást. Képzelje el, hogy egy új, ismeretlen játékot játszik, amelynek nem ismeri a szabályait. Nagyjából száz lépés után az ellenfele bejelenti: ön vesztett. Dióhéjban ez jellemzi a megerősítéses tanulást.

Számos bonyolult problématerületen a megerősítéses tanulás az egyetlen lehetséges út arra, hogy egy programnak magas szintű működést tanítsunk. Például a játékok terén nagyon nehéz lenne a szakember dolga: a lehetséges nagyszámú állás pontos és következetes értékelését kellene elvégeznie annak érdekében, hogy a példák alapján közvetlenül megtanítson egy kiértékelő függvényt. Ehelyett a programnak megadható az az információ, hogy nyert vagy vesztett, és ezt felhasználhatja egy olyan kiértékelő függvény tanulására, amely elfogadható pontossággal képes megbecsülni bármely adott állásban a nyerés valószínűségét. Hasonlóképpen rendkívül nehéz megtanítani egy ágenst helikopter vezetésére, de ha megfelelő negatív jutalmat kap minden balesetnél, bukdácsoló mozgásnál vagy a beállított útvonaltól való eltérésnél, akkor az ágens önállóan megtanulhat repülni.

Bizonyos értelemben a megerősítéses tanulás az egész MI-problémát átfogja. Egy ágenst belehelyezünk valamilyen környezetbe, és meg kell tanulnia ott sikeresen viselkedni. Ahhoz, hogy ezt a fejezetet áttekinthető bonyolultsági szinten tartsuk, csak egyszerű helyzetekre és egyszerű ágensekre fogunk koncentrálni. A fejezet legnagyobb részében teljesen megfigyelhető környezetet feltételezünk, tehát a megfigyelések informálnak az aktuális állapotról. Másrészt viszont azt feltételezzük, hogy az ágens sem azt nem tudja, hogyan működik a környezet, sem azt, hogy cselekedeteinek mi a hatása, továbbá megengedjük a cselekedetek valószínűségi (nemdeterminisztikusan meghatározott) kimenetelét. Három olyan ágensfelépítést fogunk áttekinteni, amelyeket először a 2. fejezetben vezettünk be:

Created by XMLmind XSL-FO Converter.

- A **hasznosságalapú ágens** (**utility-based agent**) az állapotokra alapozott hasznosságfüggvényt tanul, és ennek alapján választja ki azokat a cselekvéseit, amelyekkel maximálja az elérhető hasznosság várható értékét.
- A **Q-tanuló** (**Q-learning**) ágens egy **cselekvésérték** (**action-value**) függvényt – vagy más néven *Q*-függvényt – tanul, valamilyen várható hasznot tulajdonítva egy adott helyzetben egy adott cselekvésnek.
- Egy **reflexszerű ágens** (**reflex agent**) olyan stratégiát tanul, amely közvetlenül képezi le az állapotokat cselekvésekre.

Egy hasznosságalapú ágensnek a környezet valamilyen modelljével is rendelkeznie kell ahhoz, hogy tudja, cselekvése milyen állapotba vezet. Például ahhoz, hogy az ostáblajáték hasznosságfüggvényének hasznát vegyük, az ostáblajátékot játszó programnak tudnia kell, hogy melyek a lehetséges szabályos lépések, és ezek *hogyan hatnak az állásra*. Csak ilyen módon van arra lehetősége, hogy a lépése eredményeképpen kapott állásra alkalmazza a hasznosságfüggvényt. Egy *Q*-tanuló ágens viszont össze tudja hasonlítani a lehetséges választásait anélkül, hogy tudná, mire vezetnek, így nincs szüksége a környezet modelljére. Másrészt viszont, mivel nem tudják, hogy cselekvéseik hova vezetnek, a *Q*-tanuló ágensek nem képesek előre nézni. Ez viszont, mint látni is fogjuk, súlyosan korlátozhatja tanulási képességeiket.

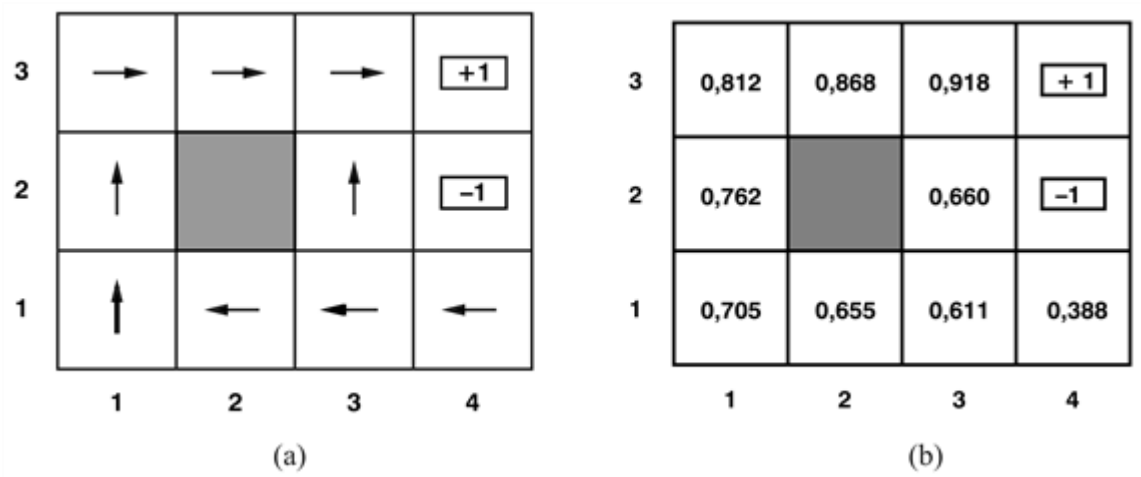
A 21.2. alfejezetben a **passzív tanulás**sal (**passive learning**) kezdünk foglalkozni, amelyben az ágensnek rögzített stratégiája van, és az állapotok hasznosságának (vagy az állapot-cselekvés párok hasznosságának) megtanulása a feladat. Ez magában foglalhatja a környezet modelljének megtanulását is. A 21.3. alfejezet az **aktív tanulás**sal (**active learning**) foglalkozik, amikor is az ágensnek azt is meg kell tanulnia, hogy mit tegyen. Az alapvető elv a **felfedezés** (**exploration**), az ágensnek a lehető legtöbbet meg kell tapasztalnia környezetéről ahhoz, hogy megtanulja, mi a célszerű viselkedés benne. A 21.4. alfejezet bemutatja, hogy az ágens hogyan tudja felhasználni az induktív tanulást arra, hogy sokkal gyorsabban tanuljon a tapasztalataiból. A 21.5. alfejezet a közvetlen stratégiareprezentáció tanulásának módszereivel foglalkozik reflexszerű ágenseknél. A Markov döntési folyamatok (lásd 17. fejezet) megértése döntő fontosságú ennek a fejezetnek a tárgyalása szempontjából.

2. Passzív megerősítéses tanulás

Az egyszerűség kedvéért egy ismert, teljes mértékben hozzáférhető környezet állapot-reprezentációját használó passzív ágens esetét tárgyaljuk először. Passzív tanulás esetén az ágens π stratégiája rögzített, az s állapotban mindig a $\pi(s)$ cselekvést hajtja végre. A cél egyszerűen a stratégia jóságának – tehát az $U^{\pi}(s)$ hasznosságfüggvénynek – a megtanulása. Példaként a 17. fejezetben bevezetett 4 × 3-as világot fogjuk használni. A 21.1. ábra mutatja ezt a világot, és a megfelelő hasznosságokat. Nyilvánvaló, hogy a passzív tanulás hasonló a **stratégiakiértékelés**i (**policy evaluation**) feladathoz, amely része a 17.3. alfejezetben ismertetett **stratégiaiteráció**s (**policy iteration**) algoritmusnak. A legfontosabb különbség, hogy a passzív ágens nem ismeri az **állapotátmenet-modell**t (**transition model**), a $T(s, a, s')$-t, amely annak a valószínűséget adja meg, hogy az a cselekvés hatására az s állapotból az s' állapotba jutunk, továbbá nem ismeri a **jutalomfüggvény**t (**reward function**), $R(s)$-et, amely minden állapothoz megadja az ott elnyerhető jutalmat.

21.1. ábra - (a) A 4 × 3-as világban alkalmazott π stratégia. Ez optimális stratégia, ha nem alkalmazunk leértékelést, és minden állapotban, amely nem végállapot, $R(s) = -0{,}04$. (b) A 4 × 3-as világ állapotainak hasznossága a π stratégia esetén.

Created by XMLmind XSL-FO Converter.



Az ágens – π stratégiája felhasználásával – egy sor **kísérlet**et (**trial**) végez. Minden egyes kísérletben az (1,1) állapotból indul, és az állapotátmenetek valamilyen sorozatát észleli, amíg el nem érkezik a (4,2), illetve a (4,3) végállapotok valamelyikébe. Észlelései alapján információt kap mind az aktuális állapotról, mind az ott nyert jutalomról. Például így nézhetnek ki a tipikus kísérletek:

$$(1,1)_{-0,04}\rightarrow(1,2)_{-0,04}\rightarrow(1,3)_{-0,04}\rightarrow(1,2)_{-0,04}\rightarrow(1,3)_{-0,04}\rightarrow(2,3)_{-0,04}\rightarrow(3,3)_{-0,04}\rightarrow(4,3)_{+1}$$

$$(1,1)_{-0,04}\rightarrow(1,2)_{-0,04}\rightarrow(1,3)_{-0,04}\rightarrow(2,3)_{-0,04}\rightarrow(3,3)_{-0,04}\rightarrow(3,2)_{-0,04}\rightarrow(3,3)_{-0,04}\rightarrow(4,3)_{+1}$$

$$(1,1)_{-0,04}\rightarrow(2,1)_{-0,04}\rightarrow(3,1)_{-0,04}\rightarrow(3,2)_{-0,04}\rightarrow(4,2)_{-1}$$

Vegyük észre, hogy mindegyik állapotészlelés mellett – alsó indexként – feltüntettük az elnyert jutalmat. Célunk a jutalmakban rejlő információ felhasználása arra, hogy az egyes s állapotokhoz – amelyek nem végállapotok – rendelhető várható hasznosságot, $U^{\pi}(s)$-t megtanuljuk. A hasznosságot úgy definiáljuk, mint a π stratégia követése esetén az összegzett (leértékelt) jutalom várható értéke. Ez – ahogy a (17.3) egyenletben a 1.2. szakasz - Optimalitás szekvenciális döntési problémákban részben – már felírtuk:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty}\gamma^{t}R(s_t)\middle|\pi, s_0 = s\right] \qquad (21.1)$$

Mindegyik összefüggésünkben figyelembe veszünk egy γ **leértékelési tényező**t (**discount factor**), de a 4 × 3 világban γ = 1-et használunk.

2.1. Közvetlen hasznosságbecslés

Widrow és Hoff az **adaptív szabályozáselmélet**tel (**adaptive control theory**) foglalkozva egy egyszerű módszert fedeztek fel a **közvetlen hasznosságbecslés**re (**direct utility estimation**) (Widrow és Hoff, 1960). Az ötlet a következő: az állapot hasznossága nem más, mint az adott állapotból kiindulva az összegzett jutalom várható értéke, és minden egyes kísérlet ennek az értéknek egy *mintájával szolgál* minden bejárt állapotra. Például az előzőkben bemutatott háromelemű tipikus kísérlethalmaz első kísérlete egy 0,72 értékű összegzett jutalommintát ad az (1,1) állapotra, két – 0,76 és 0,84 értékű – mintát az (1,2) állapotra, továbbá két – 0,80 és 0,88 értékű – mintát az (1,3) állapotra és így tovább. Ennek megfelelően az algoritmus mindegyik kísérleti lépéssorozat végén kiszámítja az összes állapotra a megfigyelt hátralevő jutalmat, és ez alapján frissíti az állapot hasznosságát, egyszerűen minden egyes állapotra mozgóablak átlagolást végezve, és az eredményt egy táblában tárolva. Ha a kísérletek száma a végtelenhez tart, a minták átlaga a (21.1) egyenlettel megadott valódi várható értékhez fog tartani.

Nyilvánvaló, hogy a közvetlen hasznosságbecslés nem más, mint egy példánya a felügyelt tanításnak. A tanító példapárok bemeneti része az állapot, kimeneti része pedig a megfigyelt hátralevő jutalom. Tehát a megerősítéses tanulást standard induktív tanulási problémára redukáltuk, a 18. fejezetben tárgyaltaknak megfelelően. A 21.4. alfejezetben áttekintjük a hasznosságfüggvények hatékonyabb reprezentálására szolgáló eszközök használatát, például a neurális hálókét. Ezeknek a reprezentációknak a használata esetén a tanulási technika közvetlenül a megfigyelt adatokra alkalmazható.

Created by XMLmind XSL-FO Converter.

Fontos

A közvetlen hasznosságbecsléssel sikerült a megerősítéses tanulást standard induktív tanulási problémára redukálnunk, amelyről már nagyon sok mindent tudunk. Sajnálatos módon egy nagyon fontos információforrás hiányzik, nevezetesen az, hogy az állapotok hasznossága nem független egymástól! *Minden egyes állapot hasznossága egyenlő az ebben az állapotban elnyerhető jutalom és az őt követő állapotok várható hasznosságának összegével.* Tehát a hasznosságértékek kielégítik a rögzített stratégiára vonatkozó Bellman-egyenletet (lásd még (17.10) egyenlet):

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U^{\pi}(s') \qquad (21.2)$$

Azzal, hogy figyelmen kívül hagyja az állapotok közötti összefüggéseket, a közvetlen hasznosság becslése eljátssza a tanulás lehetőségeit. Például a bemutatott három kísérlet közül a második érinti a (3,2) állapotot, amelyet előzőleg az ágens nem érintett még. A következő átmenet után (3,3)-ba jut, amely az első kísérlet alapján nagy hasznosságot mutat. A Bellman-összefüggés azonnal rámutat, hogy valószínűleg akkor a (3,2)-nek is nagy a hasznossága, mivel a (3,3)-ba vezet, de a közvetlen hasznosságbecslés semmit se tanul meg ebből a kísérlet végéig. Általánosabban megfogalmazva azt mondhatjuk, hogy a közvetlen hasznosságbecslésre úgy tekinthetünk, mint ami jóval nagyobb *U* hipotézistérben keres, mint amire szükség van. Ennek oka, hogy számos olyan függvényt is számon tart a hipotézisek közt, amelyek sértik a Bellman-egyenletet. Emiatt az algoritmus gyakran csak nagyon lassan konvergál.

2.2. Adaptív dinamikus programozás

Ahhoz, hogy használhassuk az állapotok közti kényszerekben rejlő információt, az ágensnek meg kell tanulnia, hogy az állapotok között milyen kapcsolatok állnak fenn. Egy **adaptív dinamikus programozás** (**ADP**) (**adaptive dynamic programming**, **ADP**) alapú ágens működésének lényege, hogy menet közben megtanulja a környezet állapotátmenet-modelljét, és dinamikus programozási módszerrel megoldja a hozzá kapcsolható Markov döntési folyamatot. Egy passzív tanuló ágens számára ez azt jelenti, hogy a megtanult $T(s, \pi(s), s')$ állapotátmenet-modellt és a megfigyelt $R(s)$ jutalmakat behelyettesíti a (21.2) Bellman-egyenletbe, majd kiszámítja az állapotok hasznosságát. Mint a 17. fejezetben a stratégiaiterációnál rámutattunk, ezek az egyenletek lineárisak (nincs maximalizálási lépés), így bármely lineáris algebrai programcsomaggal megoldhatók. Egy másik alternatíva, ha a **módosított eljárásmód-iteráció**t (**modified policy iteration**) választjuk (lásd 3. szakasz - Eljárásmód-iteráció részben), egy egyszerűsített értékiterációs eljárást használva arra, hogy a tanult modell minden változása után frissítsük a hasznosságbecslés eredményét. Mivel a modell legtöbbször csak keveset változik az egyes megfigyelések után, az értékiterációs algoritmus kiinduló értékként felhasználhatja az előző hasznosságbecslést, és egész gyorsan konvergálhat.

Magának a modellnek egyszerű a megtanulása, mivel a környezet teljesen megfigyelhető. Ennek megfelelően egy ellenőrzött tanítási feladatunk van, amelyben a bemenet az állapot-cselekvés pár, a kimenet pedig a következő állapot. A legegyszerűbb esetben az állapotátmenet-modellt valószínűségek táblázatával reprezentálhatjuk. Számon tartjuk, hogy a cselekvések egyes eredményei milyen gyakran lépnek fel, és abból a gyakoriságból becsüljük a $T(s, a, s')$ állapotátmenet valószínűségét, hogy hányszor következett s', amikor az s állapotban az a cselekvést választottuk.[1] Például ha a 2. szakasz - Passzív megerősítéses tanulás rész három útvonalában a *Jobbra* cselekvést háromszor választottuk az (1,3) állapotban, és a háromból kétszer a (2, 3) lett a következő állapot, akkor $T((1, 3), \textit{Jobbra}, (2, 3))$ becsült értéke 2/3 lesz.

A passzív ADP-ágens teljes programja a 21.2. ábrán látható. A 21.3. ábra mutatja a 4 × 3-as világon elért teljesítményét. Az értékbecsléseinek javulási sebessége szempontjából az ADP-ágens a lehetőségekhez képest a legjobban működik, ahogy csak az állapotátmenet-modell tanulási képességének megfelelően haladni tud. Ebben az értelemben más megerősítéses tanulási algoritmusok mérésének standardjaként szolgálhat. Ugyanakkor nagy állapotterek esetén némiképp kezelhetetlenné válhat. Például az ostáblajátékban nagyjából 10^{50} egyenletet kell megoldania 10^{50} ismeretlenre nézve.

21.2. ábra - Egy adaptív lineáris programozás alapon tanuló passzív megerősítéses ágens. A kód egyszerűsítése érdekében feltettük, hogy minden egyes észlelés szétbontható egy észlelt állapotra és egy jutalomjelre.

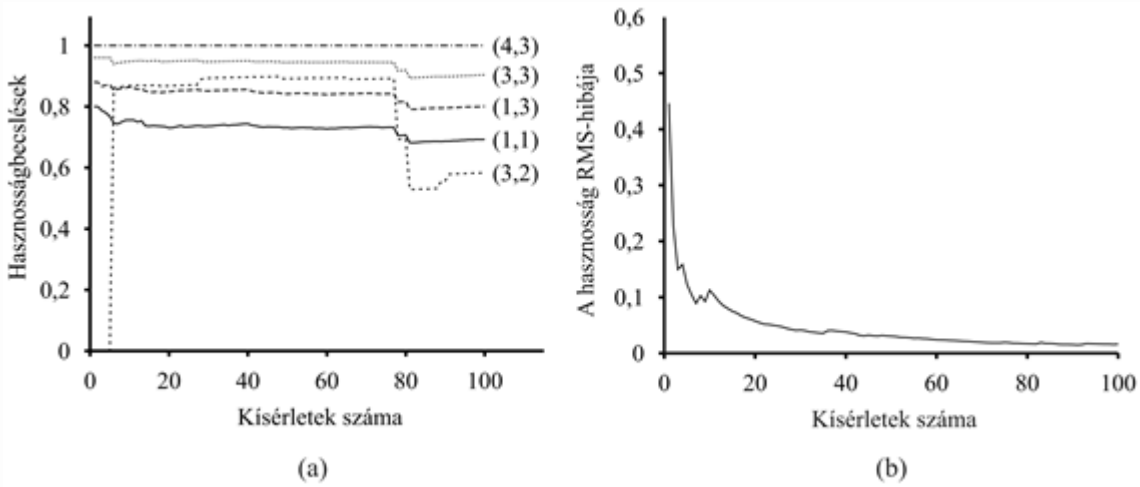
[1] Ez – mint a 20. fejezetben is tárgyaltuk – a maximum-likelihood becslés. Egy Bayes-frissítés Dirichlet-priorral jobb is lehet.

Created by XMLmind XSL-FO Converter.

```
function PASSZÍV-ADP-ÁGENS(észlelés) returns egy cselekvés
  inputs: észlelés, egy észlelés, amely a pillanatnyi s' állapotot és az r' jutalomjelet adja
  static: π, egy rögzített stratégia
                mdp, egy MDF, amely a T modellből, az R jutalomból és a γ leértékelésből áll
                U, egy hasznosságtábla, kezdetben üres
                N_sa, az állapot-cselekvés gyakoriságtábla, kezdetben nulla
                N_sas', az állapot-cselekvés-állapot hármasok gyakoriságtáblája, kezdetben nulla
                s, a, az előző állapot, illetve cselekvés, kezdetben üres

  if s' új then do U[s'] ← r'; R[s'] ← r'
  if s nem üres then do
        inkrementálja N_sa[s, a]-t és N_sas'[s, a, s']-t
        for each t amelyre N_sas'[s, a, t] nem nulla do
            T[s, a, t] ← N_sas'[s, a, t]/ N_sa[s, a]
  U←ÉRTÉK-MEGHATÁROZÁS(π, U, mdp)
  if VÉGÁLLAPOT?[s'] then s, a ← üres else s, a ← s', π[s']
  return a
```

21.3. ábra - A passzív ADP-tanuló tanulási görbéi a 4 × 3-as világ tanulása során, a 21.1. ábrán látható optimális stratégia használata esetén. (a) Az állapotok egy kiválasztott részhalmazára kiszámított hasznosságbecslések a kísérletek számának függvényében. Figyeljük meg a 78-adik kísérlet környékén a jelentős változásokat – ez az első eset, amikor az ágens a (4,2) végállapotba jut, melyben –1 jutalmat kap. (b) Az *U*(1,1) hibájának RMS-értéke 20 – egyenként 100 kísérletből álló – futásra átlagolva.



2.3. Az időbeli különbség tanulása

Fontos

Lehetőségünk van a két világ (szinte) legjobb vonásait egyesíteni: közelíthetjük az állapotok közti kényszereket leíró egyenleteket anélkül, hogy megoldanánk az összes lehetséges állapotra. *Az alapötlet a következő: használjuk fel a megfigyelt állapotátmeneteket a megfigyelt állapotok olyan módosítására, hogy azok megfeleljenek a korlátozó egyenleteknek.* Vegyük például az (1,3)-ból (2,3)-ba való átmenetet a 2. szakasz - Passzív megerősítéses tanulás második kísérletében. Tegyük fel, hogy az első kísérlet eredményeként a hasznosságuk becsült értéke $U^{\pi}(1,3) = 0,84$ és $U^{\pi}(2,3) = 0,92$. Ha ez az átmenet mindig előáll, akkor azt várjuk, hogy a hasznosságoknak meg kell felelniük a következő kényszernek:

$$U^{\pi}(1,3) = -0,04 + U^{\pi}(2,3)$$

Created by XMLmind XSL-FO Converter.

így $U^{\pi}(1,3)$ eszerint 0,88 lenne. Tehát a jelenlegi becsült érték (0,84) egy kicsit alacsonyabb a kívánatosnál, növelni kellene. Általánosabban megfogalmazva, ha egy s-ből s'-be történő átmenet lép fel, akkor a következő módon frissítjük az $U^{\pi}(s)$ hasznosságot:

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s)) \quad (21.3)$$

Ebben az egyenletben α az úgynevezett **bátorsági faktor** (**learning rate**). Mivel ez az egyenlet a hasznosságok egymást követő állapotokhoz tartozó értékeit használja, ezért gyakran **időbeli különbség** vagy **IK**- (**temporal-difference**, **TD**) egyenletnek nevezik.

Fontos

Az időbeli különbségen alapuló módszerek alapötlete az, hogy először határozzuk meg azokat a feltételeket, amelyek korrekt becslési értékek esetén lokálisan fennállnak, majd állítsunk fel egy olyan értékfrissítésre szolgáló egyenletet, amely ezen ideális „egyensúlyi” egyenlet irányába viszi a becsléseket. Passzív tanulás esetén az egyensúlyi egyenletet a (21.2) szolgáltatja. A (21.3) egyenlet ténylegesen a (21.2) által megadott egyensúlyi helyzetbe viszi az ágenst, bár van némi finom trükk a dologban. Először vegyük észre, hogy a frissítés csak a megfigyelt következő állapotot, s' -t veszi figyelembe, míg a tényleges egyensúlyi feltétel az összes lehetséges következő állapotot. Azt hihetnénk, hogy ez túlzottan megváltoztatja $U^{\pi}(s)$-t, amikor egy ritka átmenet következik be. Valójában viszont ritka átmenetek ritkán következnek be, ezért az $U^{\pi}(s)$ *átlagos értéke* a helyes értékhez fog konvergálni. Továbbá, ha α-t megváltoztatjuk úgy, hogy ne egy rögzített paraméter legyen, hanem egy függvény, amely aszerint csökken, ahogy egy állapotnak a kísérletekben történő előfordulása nő, akkor maga az $U(s)$ is a helyes értékhez fog tartani.[2] Ez a 21.4. ábrán látható ágensprogramhoz vezet. A 21.5. ábra bemutatja a passzív IK-ágens által a 4 × 3 világban mutatott teljesítményt. Ugyan nem tanul olyan gyorsan, mint az ADP-ágens, és nagyobb változékonyságot mutat, de sokkal kevesebb számítást igényel megfigyelésenként. Vegyük észre, hogy az IK *nem igényel modellt a frissítés elvégzéséhez*. A környezet szolgáltatja a szomszédos állapotok közti kapcsolatokat, a megfigyelt átmenetek formájában.

21.4. ábra - Egy időbeli különbség alapon hasznosságot tanuló passzív megerősítéses ágens

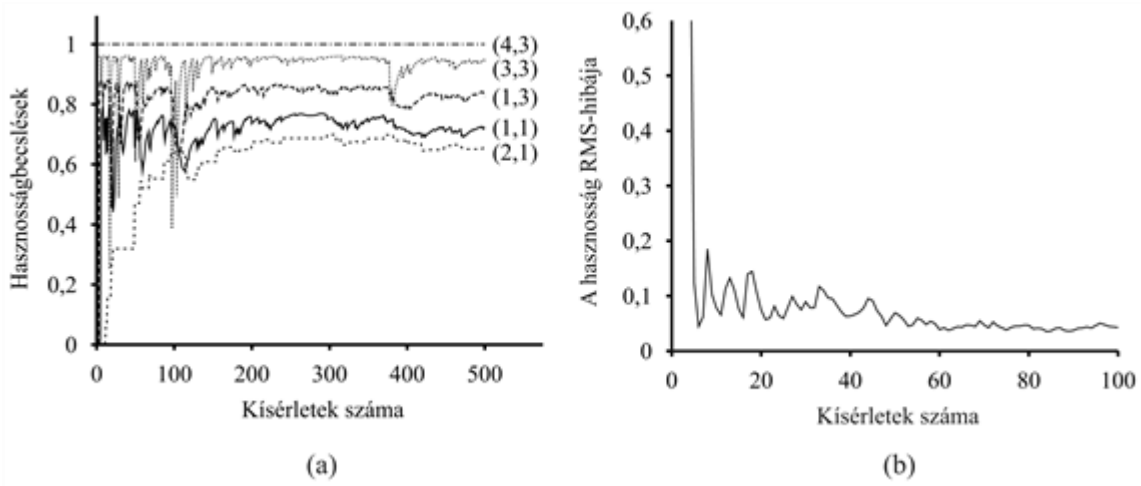
function PASSZÍV-IK-ÁGENS(*észlelés*) **returns** egy cselekvés
  inputs: *észlelés*, egy észlelés, amely a pillanatnyi s' állapotot és az r' jutalomjelet adja
  static: π, egy rögzített stratégia
    U, egy hasznosságtábla, kezdetben üres
    N_s, az állapotgyakoriságok tábla, kezdetben nulla
    s, a, r, az előző állapot, cselekvés, illetve jutalom, kezdetben üres

  if s' új **then do** $U[s'] \leftarrow r'$
  if s nem üres **then do**
    inkrementálja $N_s[s]$-t
    $U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma\, U[s'] - U[s])$
  if VÉGÁLLAPOT?$[s']$ **then** $s, a, r \leftarrow$ nulla **else** $s, a, r \leftarrow s', \pi[s'], r'$
  return a

21.5. ábra - Az IK-tanulás 4 × 3-as világon mért tanulási görbéi. (a) Néhány kiválasztott állapot hasznosságbecslése a kísérletszám függvényében. (b) Az $U(1,1)$ becslésének RMS-hibája, amelyet 20 futás átlagából számoltunk, mindegyik futás 500 kísérletet tartalmazott. Csak az első 100 kísérletből származó eredményt rajzoltuk fel, hogy a görbe a 21.3. ábrával összehasonlítható legyen.

[2] Technikailag azt kívánjuk meg, hogy $\sum_{n=1}^{\infty} \alpha(n) = \infty$ és $\sum_{n=1}^{\infty} \alpha^2(n) < \infty$ fennálljon. Az $\alpha(n) = 1/n$ csökkenés megfelel ennek a feltételnek. A 21.5. ábrán $\alpha(n) = 60/(59 + n)$-t használtuk.

Created by XMLmind XSL-FO Converter.



Az ADP- és IK-megközelítés valójában szoros rokonságban van. Mindkettő lokális változtatásokat végez a hasznosságbecslésben annak érdekében, hogy minden egyes állapot „megfeleljen” a rákövetkezőknek. Ugyanakkor van egy különbség. Az IK úgy módosítja az állapotot, hogy az megfeleljen a *megfigyelt* rákövetkező állapotnak ((21.3) egyenlet). Az ADP viszont úgy módosítja az aktuális állapotot, hogy az *összes* lehetséges következő állapotnak megfeleljen, de ezeket az állapotokat a valószínűségükkel súlyozza (21.2) egyenlet). Ez a különbség eltűnik, amikor az IK-módosítást nagyszámú átmenetre átlagoljuk, mivel az állapothalmaz következő állapotainak gyakorisága nagyjából arányos a valószínűségükkel. Egy, az előzőnél fontosabb különbség, hogy míg az IK minden megfigyelt állapotátmenet után egyetlen módosítást végez, addig az ADP annyit, amennyi csak szükséges ahhoz, hogy helyreállítsa az U hasznosságbecslések és a T környezetmodell közti konzisztenciát. Bár a megfigyelt állapotátmenet csak lokálisan változtatja meg T-t, de szükség lehet arra, hogy ezt végigterjesszük U-n. Ennek megfelelően úgy vehetjük, hogy az IK egy durva, de hatékony közelítése az ADP-módszernek.

Mindegyik, az ADP által elvégzett módosítás az IK szempontjából úgy is tekinthető, mint az aktuális környezeti modell szimulációja segítségével generált „pszeudokísérlet” eredménye. Kiterjeszthetjük az IK-megközelítést úgy, hogy a környezeti modell segítségével számos pszeudokísérletet generáljon – létrehozva olyan átmeneteket, amelyekről az aktuális környezet modellje alapján az IK-ágens el tudja képzelni, hogy *megtörténhetnek*. Az IK-ágens akár nagyszámú képzelt átmenetet is generálhat minden megfigyelt átmenetre. Ennek eredményeképpen a hasznosság becslései egyre inkább az ADP becsléseihez közelítenek, természetesen a fizetendő ár a megnövekedett számítási idő lesz.

Hasonló módon az ADP-eljárás hatékonyabb változatait hozhatjuk létre közvetlenül approximálva az értékiteráció és stratégiaiteráció algoritmusait. Ne felejtsük, hogy a teljes értékiteráció kezelhetetlenné válhat, ha az állapottér nagy. Ugyanakkor a módosítási lépések közül számos csak nagyon parányi. Elfogadhatóan jó válaszok gyors generálásához vezető lehetséges megközelítés, ha megkötjük a megfigyelt átmenetek után elvégzett módosítások számát. Azt is megtehetjük, hogy valamilyen heurisztika segítségével rangsoroljuk a lehetséges módosításokat, és csak a legfontosabbakat végezzük el. A **prioritásos végigsöprés** (**prioritized sweeping**) heurisztikája preferálja azon állapotok módosítását, amelyek *valószínű* követő állapotainak hasznosságában éppen most *nagy* módosítás történt. Ehhez hasonló heurisztikák segítségével a közelítő ADP-algoritmusok rendszerint nagyjából ugyanolyan gyorsan képesek tanulni, mint a teljes ADP, ha pusztán a tanító szekvenciák számát nézzük. Viszont több nagyságrenddel hatékonyabbak lehetnek, ha a számítási igényt vizsgáljuk (lásd 21.3. feladat). Ezáltal lehetővé válik számukra olyan állapotterek vizsgálata is, amelyek messze túl nagyok a teljes ADP-nek. A közelítő ADP-algoritmusoknak további előnye, hogy egy új környezet tanulásának korai fázisában a T környezeti modell gyakran olyan távol van a helyestől, hogy nincs sok értelme egy azzal pontosan konzisztens hasznosságfüggvényt kiszámítani. A közelítő algoritmus használhat olyan minimális méretű módosítást, amely csökken annak mentén, ahogy a környezeti modell egyre pontosabbá válik. Ez elkerülhetővé teszi a nagyon hosszú értékiterációkat, amelyek a tanulás korai szakaszában azért lépnek fel, mert nagy változások történnek a modellben.

3. Aktív megerősítéses tanulás

Created by XMLmind XSL-FO Converter.

A passzív tanuló ágensnek rögzített stratégiája van, ez határozza meg a viselkedését. Az aktív ágensnek viszont el kell döntenie, hogy melyik cselekvést válassza. Vegyük először az adaptív dinamikus programozás alapján tanuló ágenst, és nézzük meg azt, hogyan kell módosítani, hogy ezt az új lehetőséget kezelni tudja.

Először is az ágensnek nem csupán egy rögzített stratégia modelljét kell megtanulnia, hanem egy teljes modellt, amibe az összes cselekvésének lehetséges eredményeihez tartozó valószínűségek is beletartoznak. Erre a `PASSZÍV-ADP-ÁGENS` esetén alkalmazott egyszerű tanulási eljárás is kiválóan alkalmas. Ezek után figyelembe kell vennünk, hogy az ágens választhat a cselekvések között. A megtanulandó hasznosságok azok, amelyeket az *optimális* stratégia definiál. Ezek a hasznosságok megfelelnek a 2.1. szakasz - Az állapotok hasznossága részben közölt Bellman-egyenletnek, amit itt megismétlünk:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U(s') \qquad (21.4)$$

A 17. fejezetben közölt értékiterációs vagy stratégiaiterációs algoritmusok segítségével ezek az egyenletek megoldhatók az *U* hasznosságfüggvényre nézve. A végső kérdés az, hogy mit csináljunk az egyes lépésekben. Az egyik lehetőség, hogy a megtanult modell alapján nyert optimális *U* hasznosságfüggvény alapján az ágens meghatároz egy egy lépésre előretekintő optimális cselekvést annak érdekében, hogy maximálja a várható hasznosságot. A másik lehetőség, hogy ha stratégiaiterációt használt, akkor az optimális stratégia már rendelkezésére áll, így „gondolkodás nélkül” csak végre kell hajtania az optimális stratégia által javasolt cselekvést. A kérdés az, hogy valóban végre kell-e hajtania?

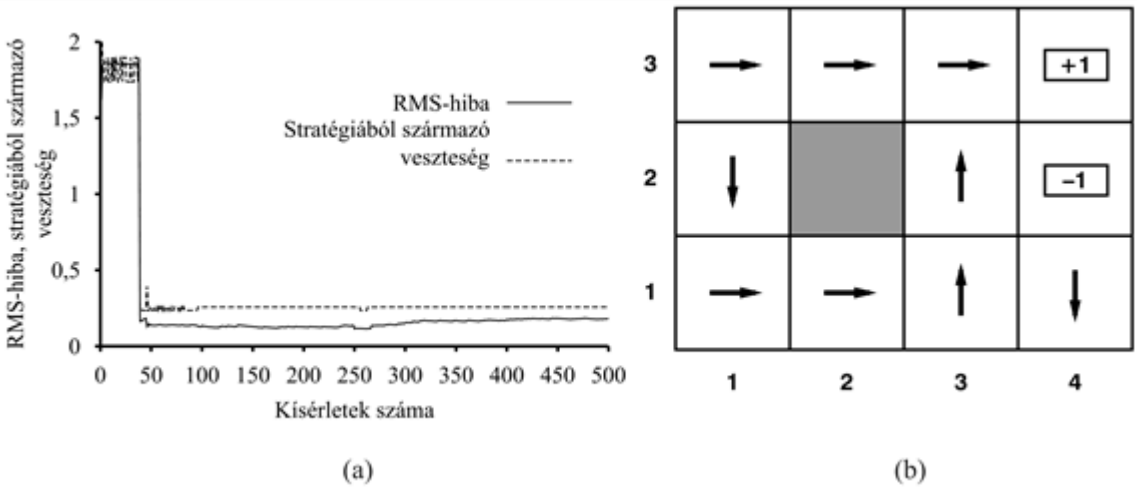
3.1. Felfedezés

A 21.6. ábra bemutatja annak a kísérletsorozatnak az eredményét, amely során az ADP-ágens az egyes lépések során tanult optimális stratégia által javasolt cselekvést követi. Látható, hogy az ágens *nem tanulja meg* sem a helyes hasznosságértékeket, sem az igazi optimális stratégiát! Ehelyett a 39-edik kísérletnél megtalál egy stratégiát, amely a +1 jutalomhoz vezet az alsó – (2,1), (3,1), (3,2), (3,3) – úton (lásd 21.6. ábra). Ezek után kisebb változtatásokkal gyűjti a tapasztalatokat, és a 276-ik kísérlettől kezdve mereven ragaszkodik ehhez a stratégiához, így soha nem tanulja meg a többi állapot hasznosságát, és soha nem talál rá a valóban optimális útra a (1,2), (1,3), (2,3) állapotokon keresztül. Ezt **mohó ágens**nek (**greedy agent**) nevezzük. Ismételt kísérletek azt mutatták, hogy a mohó ágens ebben a környezetben *nagyon ritkán* találja meg az optimális stratégiát, és időnként borzalmas stratégiákhoz konvergál.

Hogyan történhet meg, hogy optimális cselekvéseket választva szuboptimális eredményhez jutunk? A válasz abban rejlik, hogy a megtanult modell nem azonos az igazi környezettel, tehát az, ami optimális a megtanult modellben, szuboptimális lehet az igazi környezetben. Sajnálatos módon az ágens nem tudja, hogy milyen a valódi környezet, így nem képes az igazi környezetre nézve optimális cselekvést meghatározni. Mit kell akkor tennünk?

21.6. ábra - Annak a mohó ADP-ágensnek a felmutatott teljesítménye, amely végrehajtja a tanult modell alapján optimálisnak tűnő cselekvést. (a) A kilenc nem végállapot alapján számolt átlagos RMS-hiba. (b) Az a szuboptimális stratégia, amely felé ebben a kísérletsorozatban a mohó ágens konvergál.

Created by XMLmind XSL-FO Converter.



Amit a mohó ágens nem vett figyelembe, az az, hogy a cselekvésnek több funkciója is van annál, mint hogy a jelenleg megtanult modell alapján jutalmat hozzon, ezenfelül hozzájárul az igazi modell megismeréséhez is a nyert észleléseken keresztül. A modell javítása viszont azt eredményezheti, hogy az ágens több jutalomhoz jut a jövőben.[3] Ennek megfelelően az ágensnek kompromisszumot kell kötnie a **kihasználás** (**exploitation**) – a jelenlegi hasznosságbecslésében tükrözött modell alapján történő jutalom maximalizálás – és a **felfedezés** (**exploration**) között, ez utóbbi a hosszú távú eredményességet szolgálja. Ha csupán a kihasználásra tör, akkor azt kockáztatja, hogy beragad valamilyen megszokott kerékvágásba. Ha pedig csak a felfedezéssel törődik, hogy tudását bővítse, de soha nem ülteti át a tudást a gyakorlatba – nem hoz semmilyen hasznot. A való életben állandóan döntenünk kell, hogy folytassuk-e kényelmes életünket, vagy vágjunk neki az ismeretlennek, hátha egy új és jobb életet fedezünk fel. Minél több mindent értünk, annál kevesebb felfedezésre van szükségünk.

Tudunk-e ennél kicsit precízebbek lenni? Van egyáltalán *optimális* felfedezési stratégia? Kiderült, hogy ezt a kérdést a statisztikus döntéselmélet egy részterülete az ún. **rabló probléma** (**bandit problem**) kapcsán nagy mélységben tárgyalta (lásd a bekeretezett részt).

Bár a rablók problémáját rendkívül nehéz pontosan megoldani úgy, hogy egy *optimális* felfedezési stratégiát nyerjünk, azonban készíthető egy *ésszerű* terv, ami végül az ágens optimális viselkedéséhez vezet. Minden ilyen eljárás a végtelen felfedezés határán lehet mohó (**VFHM**) (greedy in the limit of infinite exploration, **GLIE**). Egy VFHM-séma alapján működő ágensnek korlátlan számban ki kell próbálnia minden cselekvést az összes állapotban. Ezáltal kerüli el annak veszélyét, hogy egy szokatlanul rossz kimenetelű sorozat miatt véges (nem nulla) valószínűséggel nem talál meg egy optimális cselekvést. Egy ilyen terv alapján működő ADP-ágens végül megtanulja az igazi környezeti modellt. Egy VFHM-tervnek végül mohóvá kell válnia, így az ágens cselekvései optimálissá válnak a megtanult (azaz itt az igazi) modellre nézve.

Fontos

Rablók és felfedezés

Las Vegasban a *félkarú rabló* egy pénzbedobós játékautomata. A játékos bedob egy érmét, meghúzza a kart, és elveszi a nyereményt (ha nyer). Egy **n karú rablónak** (**n-armed bandit**) *n* karja van. A játékosnak minden érme bedobásakor választania kell, hogy melyik kart húzza meg – azt, amelyik eddig a legjobban fizetett, vagy talán egy olyat, amelyet még nem próbált?

Az *n* karú rabló probléma a világ sok, létfontosságú területén számos valós feladat formális modelljeként szolgálhat, mint például az MI kutatás-fejlesztés éves költségvetésének kialakítása. Mindegyik kar egy cselekvésnek felel meg (mint például 400 millió forint elkülönítése MI-tankönyvek fejlesztésére), és a kar meghúzásával járó nyeremény a cselekvés választása esetén várható haszonnak felel meg (esetünkben óriási haszon várható). Az ismeretlen terület feltárása – akár egy új kutatásról, akár egy új üzletközpontról van szó – kockázatos, drága, bizonytalan haszonnal jár; másrészt ha egyáltalán nem kutatunk, akkor soha nem fogunk *semmilyen* értékes cselekvést felfedezni.

[3] Vegyük észre a 16. fejezetben tárgyalt információérték-elmélettel fennálló közvetlen analógiát.

Created by XMLmind XSL-FO Converter.

Ahhoz, hogy a rabló problémát megfelelően formalizáljuk, pontosan meg kell határoznunk, hogy mit értünk optimális viselkedésen. Az irodalomban található legtöbb definíció azt feltételezi, hogy a cél az ágens élettartama alatt elérhető összes jutalom várható értékének maximalizálása. Ezek a definíciók megkívánják, hogy várható értéket képezzünk egyrészt az összes lehetséges világ felett, amelybe csak kerülhet az ágens, másrészt bármelyik adott világ minden egyes lehetséges cselekvési szekvenciája fölött. Itt a „világot" a $T(s, a, s')$ állapotátmenet-modell definiálja. Tehát ahhoz, hogy az ágens optimálisan cselekedjék, szüksége van a lehetséges modellek a priori eloszlására. Ezen az úton rendszerint teljesen kezelhetetlen optimalizációs problémákhoz jutunk.

Néhány esetben azonban – például amikor az egyes gépek nyereményei függetlenek, és leértékelt jutalmakat használunk – minden egyes játékgépre kiszámítható az úgynevezett **Gittins-index** (Gittins, 1989). Az index csak annak függvénye, hogy hányszor választottuk a gépet, és eddig mennyit fizetett ki. Az index azt jelzi minden egyes gépre, hogy mennyire éri meg további pénzt bedobnunk, és a várható kifizetésen és az információ várható értékén alapul. A legnagyobb indexű gép választása az optimális felfedezési stratégia. Sajnálatos módon eddig még nem sikerült megtalálni annak a módját, hogy hogyan kell a Gittins-indexet szekvenciális döntési problémákra kiterjeszteni.

Az n karú rabló elmélet felhasználható arra, hogy bizonyítsuk a genetikus algoritmusok szelekciós stratégiájának ésszerűségét (lásd 4. fejezet). Tekintsünk az n karú rabló problémában minden egyes kart egy lehetséges génfüzérnek, továbbá a pénz bedobása ennél a karnál feleljen meg ezen gének reprodukciójának. Ebben az esetben a genetikus algoritmus optimálisan osztja szét a pénzt akkor, ha adott a függetlenségi feltételezések egy megfelelő halmaza.

Számos VFHM-séma ismert; az egyik legegyszerűbb, ha az ágens az idő $1/t$ részében véletlen cselekvést választ, egyébként pedig mohó stratégiát követ. Tény, hogy ez végül optimális stratégiához vezet, de időnként rendkívül lassan konvergál. Egy értelmesebb megközelítés, ha az ágens némi súlyt ad azoknak a cselekvéseknek, amelyeket nem használt még gyakran, miközben igyekszik elkerülni azokat, amelyeknek a hasznosságát kicsinek hiszi. Ezt meg tudjuk valósítani úgy, ha úgy módosítjuk a (21.4) kényszeregyenletet, hogy nagyobb hasznosságot tulajdonítson a relatíve kipróbálatlan állapot-cselekvés pároknak. Lényegében egy optimista priort hoz létre, az ágens kezdetben úgy viselkedik, mintha hatalmas jutalmak lennének szétszórva a helyszínen. Jelölje $U^+(s)$ az s állapot hasznosságának (tehát a hátralevő jutalom várható értékének) optimista becslését, $N(a, s)$ pedig azt a számot, ahányszor az s állapotban az a cselekvést választottuk. Tegyük fel, hogy az ADP-tanulást végző ágens értékiterációt végez, ez esetben a frissítési egyenletet [(17.6) egyenlet] át kell írnunk úgy, hogy az az optimista becslést tartalmazza. A következő összefüggés ezt megvalósítja:

$$U^+(s) \leftarrow R(s) + \gamma \max_a f\left(\sum_{s'} T(s,a,s')U^+(s'), N(a,s)\right) \qquad (21.5)$$

Itt $f(u, n)$ az úgynevezett **felfedezési függvény** (**exploration function**). Ez határozza meg a mohóság (a magas u értékek preferenciája) és a kíváncsiság (az alacsony n értékek preferenciája – cselekvések, amelyeket még nem alkalmaztunk nagyon gyakran) közötti kompromisszumot. Az $f(u, n)$ függvénynek u-ban növekvőnek, n-ben csökkenőnek kell lennie. Nyilvánvalóan sok lehetséges függvény megfelel ennek a feltételnek. Egy különösen egyszerű függvénydefiníció:

$$f(n,u)\begin{cases} R^+ \text{ ha } n < N_e \\ u \quad \text{egyébként} \end{cases}$$

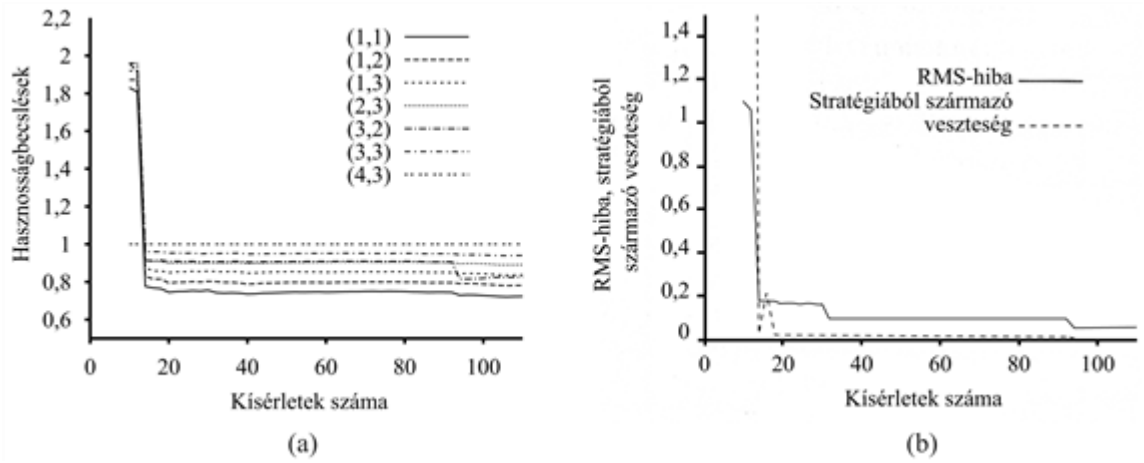
ahol R^+ a tetszőleges állapotban kapható legnagyobb jutalom optimista becslése, míg N_e egy rögzített paraméter. Ennek alkalmazásával az ágens minden állapot-cselekvés párt legalább N_e alkalommal kipróbál.

Az, hogy a (21.5) egyenlet jobb oldalán nem U, hanem U^+ szerepel, nagyon fontos. Ahogy a felfedezés folyik, a startállapot környéki állapotok és cselekvések jó néhányszor kipróbálásra kerülhetnek. Ha U-t – a pesszimistább becslést – használtuk volna, akkor az ágens hamar elveszthetné a szabad területek felfedezése iránti vonzalmát. Az U^+ szerepeltetése azt jelenti, hogy a felfedezéshez kapcsolt jutalom visszaterjed a felfedezetlen terület határáról. Így nem csupán a szokatlan cselekvéseket preferáljuk, hanem a felértékeljük azokat a cselekvéseket, amelyek a felfedezetlen területek *felé* fogják vinni az ágenst. Ennek a felfedezési stratégiának a hatása jól látható a 21.7. ábrán, gyors konvergenciát látunk az optimális teljesítményhez, ellentétben a mohó ágensnél látottakkal. Az optimálishoz nagyon közeli stratégiát talál mindössze 18 kísérlet után. Vegyük észre, hogy maguk a hasznosságérték-becslések nem konvergálnak ilyen gyorsan. Ennek az az oka, hogy az ágens meglehetősen

Created by XMLmind XSL-FO Converter.

hamar felhagy az állapottér jutalmat nem eredményező részeinek felfedezésével, oda már csak „véletlenszerűen” téved. Mindamellett teljesen ésszerű az ágens részéről, hogy nem sokat törődik azon állapotok hasznosságának pontos értékével, amelyekről tudja, hogy nemkívánatosak és elkerülhetők.

21.7. ábra - A felfedező ADP-ágens teljesítménye, $R^+ = 2$ és $N_e = 5$ esetén. (a) Néhány kiválasztott állapot hasznosságának becslése az idő függvényében. (b) A hasznosságértékek RMS-hibája, és az ehhez kapcsolható, a stratégiából származó veszteség.



3.2. Egy cselekvésérték-függvény tanulása

Most, hogy már van egy aktív ADP-ágensünk, nézzük meg, hogy mi módon tudunk egy aktív időbeli különbség tanuló ágenst létrehozni. A legszembetűnőbb eltérés a passzív esethez képest, hogy az ágensnek többé nem áll rendelkezésére egy rögzített stratégia, tehát ha megtanul valamilyen U hasznosságfüggvényt, akkor meg kell tanulnia egy modellt is, hogy egy U-n alapuló egylépéses előretekintő keretben cselekvést tudjon választani. A modellkialakítás problémája ugyanaz az IK-ágensnél, mint az ADP-ágensnél. Mi a helyzet magával az IK frissítési szabállyal? Talán meglepő, de a (21.3) IK frissítési szabály változatlan marad. Ez furcsának tűnhet a következő okból: tegyük fel, hogy az ágens olyan lépést választ, ami normál esetben jó célhoz vezet, de a környezet nemdeterminisztikus volta miatt az ágens valamilyen katasztrofális állapotban végzi. Az IK frissítési szabály ezt éppoly komolyan veszi, mintha ez a kimenetel a cselekvés normál következménye lenne, pedig csak egy pech volt, és nem kellene sokat aggódnia miatta. Valójában az ilyen valószínűtlen kimenetel csak ritkán fordul elő egy nagy tanítósorozat-halmazban, így reményeink szerint hosszú távon a valószínűségével arányosan kap súlyt a hatása. Ismét azt mondhatjuk, hogy az IK-ágens ugyanazokhoz az értékekhez fog tartani, mint az ADP-ágens, ahogy a tanítósorozatok száma tart a végtelenhez.

Van egy alternatív IK-módszer, amit **Q-tanulás**nak (**Q-learning**) hívunk, ez cselekvésérték-reprezentációt tanul, nem hasznosságot. $Q(a, s)$-sel fogjuk jelölni annak az értékét, ha az s állapotban az a cselekvést választjuk. A Q-értékek közvetlenül összekapcsolhatók a hasznosságértékekkel a következő összefüggés alapján:

$$U(s) = \max_a Q(a,s) \qquad (21.6)$$

Fontos

A Q-függvény úgy tűnhet, mintha csupán a hasznosságinformáció tárolásának egy eltérő módja lenne, de van egy nagyon fontos tulajdonsága: *egy Q-függvényt tanuló IK- ágensnek nincs szüksége sem a tanulás, sem a cselekvés kiválasztás modelljére.* Ezért a Q-tanulást **modellmentes** (**model-free**) módszernek is nevezik. Éppúgy, mint a hasznosságoknál, felírhatunk egy egyensúlyi egyenletet, amelyet a korrekt Q-értékeknek ki kell elégíteniük:

$$Q(a,s) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(a',s') \qquad (21.7)$$

Created by XMLmind XSL-FO Converter.

Az ADP-tanulást végző ágenssel azonos módon itt is felhasználhatjuk ezt az egyenletet egy iterációs eljárásban arra, hogy egy becsült modell alapján pontos *Q*-értékeket számoljunk. Ez viszont azt kívánná meg, hogy a modellt is megtanuljuk, hiszen az egyenlet használja a $T(s, a, s')$-t. Másrészről viszont az IK-megközelítés nem igényel modellt. Így a *Q*-értékekre vonatkozó IK frissítési összefüggés:

$$Q(a,s) \leftarrow Q(a,s) + \alpha\left(R(s) + \gamma \max_{a'} Q(a',s') - Q(a,s)\right) \qquad (21.8)$$

amely mindig kiszámításra kerül, amikor az *s* állapotban az *a* cselekvést választottuk, amely az *s'* állapotba vitt.

A 21.8. ábra mutatja be az IK-módszert használó, felfedező *Q*-tanuló ágens teljes programját. Vegyük észre, hogy ugyanazt az *f* felfedezési függvényt használja, mint amit a felfedező ADP-ágens is használt – ezért van szükség a végrehajtott cselekvések gyakorisági statisztikáira (az *N* táblára). Ha egy egyszerűbb felfedezési stratégiát használnánk – mondjuk a lépések egy részében véletlenszerű működést iktatva be, ahol ez részarány az idővel csökken –, akkor mellőzhetnénk ezt a statisztikát.

A *Q*-tanuló ágens megtanulja a 4 × 3-as világra az optimális stratégiát, de ezt sokkal lassabban teszi, mint az ADP-ágens. Ennek oka, hogy az IK nem kényszerít modellje alapján konzisztenciát az értékekre. Ez az összehasonlítás felvet egy általános kérdést: mi a jobb: egy modellt és egy hasznosságfüggvényt tanulni, vagy egy cselekvésértékfüggvényt? Más szavakkal, mi a legjobb módja egy ágensfüggvény reprezentálásának? Ez a mesterséges intelligencia alapjait érintő kérdés. Mint az 1. fejezetben kijelentettük, az MI legnagyobb részében kulcsfontosságú történeti örökség a (gyakran ki nem mondott) ragaszkodás a **tudásalapú** (**knowledge-based**) megközelítéshez. Ez vezet ahhoz a feltevéshez, hogy az ágensfüggvény reprezentációjának legjobb módja, ha valamilyen szempontból az ágenst körülvevő környezet modelljét építjük meg.

21.8. ábra - Egy felfedező *Q*-tanuló ágens. Egy aktív tanuló, amely megtanulja minden cselekvésnek minden egyes állapotban a *Q*(*a, s*) értékét. Ugyanazt az *f* felfedezési függvényt használja, amelyet a felfedező ADP-ágens is használt, de elkerüli az állapotátmenet-modell tanulását, mivel az állapot *Q*-értékét közvetlenül a szomszédos állapotok *Q*-értékeihez tudja kapcsolni.

function Q-TANULÓ-ÁGENS(*észlelés*) **returns** egy cselekvés
  inputs: *észlelés*, egy észlelés, amely a pillanatnyi s' állapotot és az r' jutalomjelet adja
  static: Q, egy cselekvésérték-tábla; amelyet az állapottal és a cselekvéssel indexelünk
    N_{sa}, az állapot-cselekvés párok gyakoriságának táblája
    s, a, r, az előző állapot, cselekvés, illetve jutalom, kezdetben nulla

  if s nem nulla **then do**
    inkrementálja $N_{sa}[s, a]$-t
    $Q[a, s] \leftarrow Q[a, s] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[a', s'] - Q[a, s])$
  if VÉGÁLLAPOT?$[s']$ **then** $s, a, r \leftarrow$ nulla
  else $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[a', s'], N_{sa}[a', s']), r'$
  return a

Néhány kutató ugyanakkor – mind az MI-közösségen belül, mind azon kívül – azt állítja, hogy a modellmentes módszerek, mint például a *Q*-tanulás léte azt mutatja, hogy a tudásalapú megközelítés nem szükséges. Mindamellett nem sok egyéb van e kérdés eldöntésére, mint az intuíciónk. A mi intuíciónk szerint, ahogy a környezet komplexebbé válik, a tudásalapú megközelítés előnyei egyre nyilvánvalóbbá válnak. Ez még a játékokban – mint a sakk, a dámajáték és az ostáblajáték (lásd következő alfejezet) – is így jelentkezik, a modellalapú kiértékelő függvények tanulására fordított erőfeszítések több sikert hoztak, mint a *Q*-tanuló módszerek.

4. A megerősítéses tanulás általánosító-képessége

Created by XMLmind XSL-FO Converter.

Az eddigiekben azt feltételeztük, hogy az ágens által megtanult hasznosságfüggvények és a *Q*-függvények táblázatos formában reprezentáltak, amelyben minden egyes bemeneti vektornak egy-egy kimeneti érték felel meg. Ez a megközelítés kis állapotterekre jól működik, de a konvergencia ideje, valamint egy-egy iterációs lépés ideje (ADP esetén) gyorsan nő, ahogy a tér egyre nagyobb lesz. Alaposan kézben tartott közelítő ADP-módszerekkel kezelhető lehet 10 000 állapot vagy annál valamivel több. Ez kielégítő a kétdimenziós útvesztő jellegű problémák esetén, de reálisabb problémák szóba sem jöhetnek. A sakk és az ostáblajáték a valós világ apró részei csupán, mégis állapotterük 10^{50}–10^{120} állapotot tartalmaz. Az teljesen abszurd feltételezés, hogy mindezeket az állapotokat be kell járnunk ahhoz, hogy megtanuljuk a játékot!

Ezeknek a problémáknak egy lehetséges kezelési módja, ha **függvényapproximáció**t (**function approximation**) használunk, ami egyszerűen azt jelenti, hogy a leképezésnek bármilyen reprezentációját használhatjuk, kivéve a táblázatos reprezentációt. A reprezentációt közelítőnek tekintjük, mivel nem áll fenn biztosan, hogy az *igazi* hasznosságfüggvény vagy *Q*-függvény reprezentálható a választott formában. A 6. fejezetben például leírtunk egy sakkjátékban használható **kiértékelő függvény**t (**evaluation function**), amelyet **tulajdonság**ok (**feature**) (vagy **bázisfüggvény**ek (**basis function**)) $f_1,\ldots, f_n$ halmazának lineáris kombinációjával reprezentáltunk:

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots + \theta_n f_n(s)$$

Megerősítéses tanulás során tanulhatjuk a $\theta = \theta_1,\ldots, \theta_n$ paramétereket úgy, hogy az az igazi hasznosságfüggvényt közelítse. Ahelyett, hogy mondjuk 10^{120} értéket tartanánk egy táblázatban, ez a függvényközelítés mondjuk $n = 20$ értékkel jellemezhető, ez *elképesztő* tömörítés! Bár senki sem ismeri a sakkjáték igazi hasznosságfüggvényét, azt senki se hiszi, hogy pontosan jellemezhető lenne 20 számmal. Mindamellett, ha az approximáció kielégítően jó, az ágens még mindig képes lehet kiválóan sakkozni.[4]

Fontos

A függvényapproximáció tulajdonságai lehetővé teszik, hogy praktikusan felhasználható legyen nagy állapotterek hasznosságfüggvényeinek reprezentálására, de nem ez az alapvető előnyük. *A függvényapproximátorokkal elért tömörítés lehetővé teszi a tanuló ágens számára az általánosítást azokról az állapotokról amelyeket bejárt, azokra az állapotokra, amelyeket nem járt be.* Azaz a függvényapproximáció legfontosabb aspektusa nem az, hogy kevesebb helyet igényel, hanem az, hogy lehetővé teszi az induktív általánosítást a bemeneti állapottérben. Hogy némi fogalmat alkothassunk ennek a hatásnak az erejéről: a lehetséges ostáblajáték állapotok közül csupán minden 10^{44}-edik állapotnak a vizsgálata alapján megtanulható olyan hasznosságfüggvény, amely képessé tehet egy programot, hogy olyan jól játsszon, mint egy ember (Tesauro, 1992).

Az éremnek természetesen van másik oldala is: lehet, hogy a kiválasztott hipotézistérben egyáltalán nincs olyan függvény, amely kielégítően approximálná az igazi hasznosságfüggvényt. Mint az induktív tanulásnál minden esetben, itt is kompromisszum van a hipotézistér mérete és a függvény megtanulásához szükséges idő között. Egy nagyobb hipotézistér valószínűbbé teszi, hogy jó approximációt lehet találni, de azt is jelenti, hogy a konvergencia valószínűleg lassabb lesz.

Kezdjük a legegyszerűbb esettel, a közvetlen hasznosságbecsléssel (lásd 21.2. alfejezet). Függvényapproximáció esetén ez nem más, mint a **felügyelt tanulás** (**supervised learning**) egy esete. Tegyük fel például, hogy a 4 × 3 világ hasznosságértékeit egy egyszerű lineáris függvénnyel reprezentáljuk. A négyzetek jellemzői egyszerűen az *x* és *y* koordinátáik, tehát:

$$\hat{U}_\theta(x, y) = \theta_0 + \theta_1 x + \theta_2 y \qquad (21.9)$$

Tehát ha $(\theta_0, \theta_1, \theta_2) = (0{,}5,\ 0{,}2,\ 0{,}1)$, akkor $\hat{U}_\theta(1, 1) = 0{,}8$. Adott kísérlethalmaz esetén egy sor értéket nyerünk az értékekre, és megtalálhatjuk a – mondjuk négyzetes értelemben – legjobb illesztést, standard lineáris regressziót használva (lásd 20. fejezet).

[4] Tudjuk, hogy az igazi hasznosságfüggvény reprezentálható egy- vagy kétoldalnyi Lisp, Java vagy C++ kóddal. Azaz reprezentálható egy programmal, ami pontosan oldja meg a játékot, valahányszor meghívjuk. Mi csak olyan függvényapproximátorokban vagyunk érdekeltek, amelyek *ésszerű* számítási mennyiséget végeznek. Valójában jobb lehet, ha egy nagyon egyszerű függvényapproximátort tanulunk meg, és kombináljuk valamilyen lépésszámú előretekintő kereséssel. A szükséges kompromisszumok mindamellett egyelőre nem jól ismertek.

Created by XMLmind XSL-FO Converter.

Megerősítéses tanulás esetén jobbnak tűnik *online* tanulást alkalmazni, amely minden egyes kísérlet után frissíti a paramétereket. $\hat{U}_\theta(1,1) = 0,8$ hogy lefuttatunk egy kísérletet, és a teljes jutalom (1, 1)-ből indulva 0,4. Ez arra utal, hogy az jelenlegi érték túl nagy, és csökkenteni kell. Hogyan kell a paramétereket módosítani, hogy ezt elérjük? Éppúgy, mint a neurális háló tanulásnál, egy hibafüggvényt írunk fel, és felírjuk a paraméterekre vonatkozó gradiensét. Ha $u_j(s)$ a j-edik kísérlet során az s állapotból kiindulva gyűjtött teljes megfigyelt jutalom, akkor a hibát a jósolt é $E_j(s) = \left(\hat{U}_\theta(s) - u_j(s)\right)^2 / 2$ négyzetének különbségeként (tulajdonképpen annak feleként) definiálják: . Az egyes θ_i paraméterek változásának hatására a hiba változásának sebességét a $\partial E_j/\partial\theta_i$ adja meg. Így, ha a paramétert abban az irányba akarjuk változtatni, hogy a hibát csökkentse, akkor azt akarjuk, hogy:

$$\theta_i \leftarrow \theta_i - \alpha\frac{\partial E_j(s)}{\partial\theta_i} = \theta_i + \alpha\left(u_j(s) - \hat{U}_\theta(s)\right)\frac{\partial\hat{U}_\theta(s)}{\partial\theta_i} \qquad (21.10)$$

Ezt **Widrow–Hoff-szabály**nak (**Widrow-Hoff rule**) vagy **delta-szabály**nak (**delta rule**) nevezzük, és az online legkisebb négyzetes hibára vonatkozik. A (21.9) egyenletben felírt lineáris függvényapproximátorra a következő három egyszerű frissítési szabályt kapjuk:

$$\theta_0 \leftarrow \theta_0 + \alpha\left(u_j(s) - \hat{U}_\theta(s)\right)$$
$$\theta_1 \leftarrow \theta_1 + \alpha\left(u_j(s) - \hat{U}_\theta(s)\right)x$$
$$\theta_2 \leftarrow \theta_2 + \alpha\left(u_j(s) - \hat{U}_\theta(s)\right)y$$

Alkalmazhatjuk ezeket a szabályokat arra az esetre, amikor $U_\theta(1,1) = 0,8$ és $u_j(1, 1) = 0,4$. Ekkor θ_0, θ_1 és θ_2 mindegyike $0,4\alpha$-val csökken, ami mind csökkenti az (1, 1)-beli hibát. Vegyük észre, hogy a *θ_i-k változtatása az összes többi állapotra is megváltoztatja U_θ értékét!* Ezt értettük azon, amit mondtunk: a függvényapproximáció lehetővé teszi a megerősítéses tanulást végzőnek, hogy általánosítson a tapasztalataiból.

Azt várjuk, hogy az ágens gyorsabban fog tanulni, ha függvényapproximátort használ, feltéve, hogy a hipotézistér nem túl nagy, de van benne néhány olyan függvény, amely kielégítően illeszkedik az igazi hasznosságfüggvényre. A 21.7. feladatban azt várjuk az olvasótól, hogy értékelje a közvetlen hasznosságbecslést mind a függvényapproximáció felhasználásával, mind anélkül. A 4 × 3-as világban a javulás érzékelhető, de nem drámai, mivel nagyon kicsi az állapottér, amivel dolgozni kell. A javulás sokkal nagyobb a 10 × 10-es világban, ha +1 jutalom van a (10, 10) mezőn. Ez a világ jól illeszthető a lineáris hasznosságfüggvényhez, mivel az igazi hasznosságfüggvény sima és közel lineáris (lásd 21.10. feladat). Ha a jutalmat az (5, 5) mezőre tesszük, akkor az igazi hasznosságfüggvény piramis jellegű, és a (21.9) egyenletben felvett függvényapproximátor szánalmas kudarcot vall. Mindamellett nem veszett el minden! Emlékezzünk vissza, hogy a lineáris függvényapproximációnál az számít, hogy a függvény *paramétereiben* legyen lineáris – a használt tulajdonságok önmagukban az állapotváltozók tetszőleges nemlineáris függvényei lehetnek. Tehát használhatunk egy $\theta_3 = \sqrt{(x - x_g)^2 + (y - y_g)^2}$ típusú tagot, amely a céltól való távolságot méri.

Ugyanilyen jól alkalmazhatjuk ezeket az elveket az időbeli különbség alapján tanulókra is. Mindössze annyit kell tennünk, hogy a paramétereket úgy módosítjuk, hogy az egymásra következő állapotok időbeli különbségét csökkenteni próbáljuk. A (21.3) és (21.8) IK- és Q-tanulási egyenleteink új változata:

$$\theta_i \leftarrow \theta_i + \alpha\left[R(s) + \gamma\hat{U}_\theta(s') - \hat{U}_\theta(s)\right]\frac{\partial\hat{U}_\theta(s)}{\partial\theta_i} \qquad (21.11)$$

a hasznosságfüggvényre, és

$$\theta_i \leftarrow \theta_i + \alpha\left[R(s) + \gamma\max_{a'}\hat{Q}_\theta(a', s') - \hat{Q}_\theta(a, s)\right]\frac{\partial\hat{Q}_\theta(a, s)}{\partial\theta_i} \qquad (21.12)$$

Created by XMLmind XSL-FO Converter.

a *Q*-értékekre. Megmutatható, hogy ezek a frissítési szabályok a paramétereiben lineáris függvényapproximátor esetén az igazi függvény lehető legközelebbi[5] approximációjához konvergálnak. Szerencsétlen módon mindez nem áll meg, ha nemlineáris függvényeket használunk – mint amilyenek például a neurális hálók. Nagyon egyszerű példák találhatók, amelyeknél a paraméterek a végtelenhez tartanak, pedig jó megoldások találhatók a hipotézistérben. Vannak kifinomultabb algoritmusok, amelyek elkerülik ezeket a problémákat, de napjainkban a megerősítéses tanulás általános függvényeket használó függvényapproximátorral – megmaradt a magas művészet szintjén.

A függvényapproximáció a környezet modelljének tanulásában is nagyon hasznos lehet. Emlékezzünk rá, hogy egy *megfigyelhető* környezet modelljének tanulása *ellenőrzött* tanulási feladat, mivel a következő észlelés megadja a kívánt kimeneti állapotot. A 18. fejezetben ismertetett bármely ellenőrzött tanulási módszer használható, megfelelő módosításokkal. Ugyanis itt nem egy logikai változó vagy egyetlen valós érték predikciójáról van szó, hanem komplett állapotleírásokéról. Ha például az állapot *n* logikai változóval adható meg, akkor *n* logikai függvényt kell megtanulnunk, hogy az összes változót megjósolhassuk. Sokkal nehezebb a tanulási probléma *részlegesen megfigyelhető* környezet esetén. Ha tudjuk, hogy melyek a rejtett változók, és tudjuk, hogy milyen oksági viszonyban állnak egymással és a megfigyelhető változókkal, akkor a 20. fejezetben leírtak szerint rögzíthetünk egy megfelelő Bayes-hálóstruktúrát, és használhatjuk az EM-algoritmust a paraméterek tanulására. A rejtett változók bevezetése, illetve a modellstruktúra tanulása jelenleg megoldatlan problémák.

Nézzük most a megerősítéses tanulás komolyabb alkalmazásait. Látni fogjuk, hogy azokban az esetekben, amelyekben a hasznosságfüggvényt (és ezzel együtt a modellt) használjuk, ott feltételezzük, hogy a modell adott. Például az ostáblajáték kiértékelő függvényének tanulása során általában feltesszük, hogy a legális lépések és azok hatása előzetesen ismert.

4.1. Alkalmazások a játékok területén

A megerősítéses tanulás első fontos alkalmazása Arthur Samuel dámajáték programja volt (Samuel, 1959; 1967) – egyben ez volt a tanuló programok összes válfaja közt is az első jelentősebb. Samuel először egy legfeljebb 16 tagú súlyozott lineáris álláskiértékelő függvényt használt. A (21.11) egyenlet egy variánsát alkalmazta frissítési szabályként. Mindamellett volt néhány lényeges eltérés az ő programja és a jelenleg használt módszerek közt. Az első az, hogy ő a jelen állapot és a keresési fában végrehajtott teljes előretekintéses keresés által adott érték közti különbség alapján frissítette a súlyokat. Ez nagyon jól működik, mert ahhoz vezet, hogy az állapotteret különböző felbontással lássuk. A második különbség, hogy a programja *nem* használt semmilyen megfigyelt jutalmat! Tehát a végállapot értékét figyelmen kívül hagyta. Ez azt jelenti, hogy könnyen elképzelhető, hogy Samuel programja nem konvergál, vagy ha konvergál, akkor a nyerő helyett a vesztő stratégiához tart. Samuel ezt oly módon kerülte el, hogy ragaszkodott ahhoz, hogy az anyagi előnyhöz tartozó súly pozitív legyen. Érdekes módon ez elegendő volt ahhoz, hogy programját a súlyok terének olyan részébe irányítsa, amely jó dámajátékot eredményezett.

Gerry Tesauro TD-Gammon rendszere (Tesauro, 1992) jól mutatja a megerősítéses tanulási technikák erejét. Korábbi munkájában (Tesauro és Sejnowski, 1989) Tesauro megkísérelte a $Q(a, s)$ függvény neurális reprezentációjának tanulását közvetlenül humán szakemberek által relatív értékekkel címkézett lépések alapján. Ez a megközelítés nagyon fárasztó volt a szakembereknek. Az eredményül kapott `NEUROGAMMON` nevű program a számítógépes programok közt erős játékos volt ugyan, de nem volt versenyképes a humán szakemberekkel. A TD-Gammon projekt az önmagában való tanulás egy kísérlete volt. A TD-Gammon csupán az egyes játszmák végén adott egyetlen jutalomjelet. A kiértékelő függvény reprezentációját egy teljesen összekötött, egyetlen – 40 elemből álló – rejtett réteggel felépített neuronhálóval oldotta meg. A (21.11) egyenlet szimpla ismételt alkalmazásával a TD-Gammon lényegesen jobban megtanult játszani, mint a Neurogammon annak ellenére, hogy a bemeneti reprezentáció a számított tulajdonságok nélküli nyers táblaállás volt. Ez körülbelül 200 000 játékot és két hét számítógépes futást igényelt. Bár ez hatalmas mennyiségű játéknak tűnhet, de valójában az állapottérnek csak egy elhanyagolhatóan kis részét fedi le. Amikor a bemeneti reprezentációt számított tulajdonságokkal bővítették, akkor egy 80 rejtett rétegbeli elemmel felépített neurális háló 300 000 tanító játszma után képes volt olyan szinten játszani, mint a világ három legerősebb játékosa. Kit Woolsey, az egyik élvonalbeli játékos és szakíró azt nyilatkozta: „Kétségtelen, hogy a pozíciómegítélésben sokkal jobb nálam.”

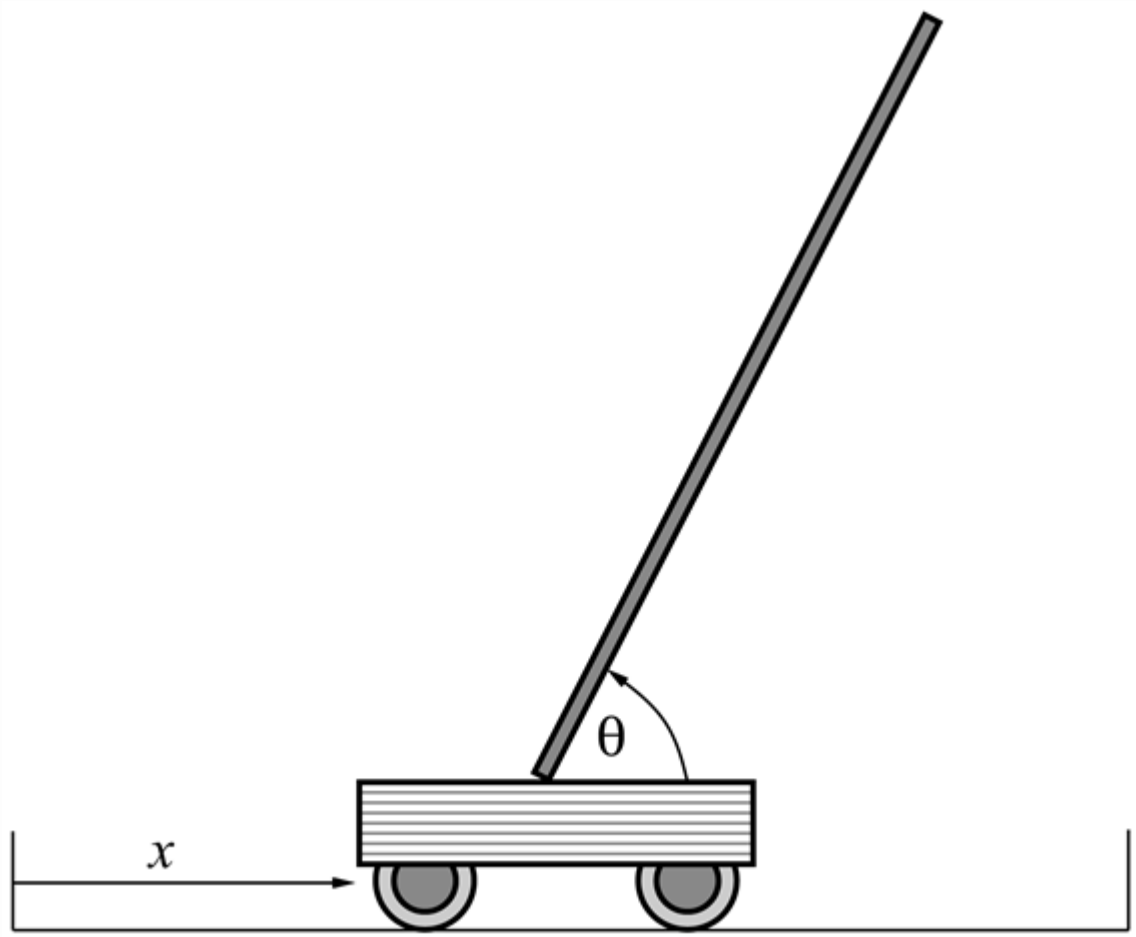
4.2. Robotirányítási alkalmazások

[5] A hasznosságfüggvények közti távolság definíciója csak technikai kérdés, lásd (Tsitsiklis és Van Roy, 1997).

Created by XMLmind XSL-FO Converter.

A híres **rúd-egyensúlyozási** (**cart-pole**) feladat, amelyet **inverz ingá**nak (**inverted pendulum**) is neveznek, a 21.9. ábrán látható. A feladat a kocsi *x* pozíciójának vezérlése úgy, hogy a rúd nagyjából fölfelé álljon ($\theta \approx \pi/2$), miközben a kocsi a pálya kijelölt határai között marad. A megerősítéses tanulás és a szabályozáselmélet területén több mint 2000 cikket publikáltak erről a látszólag egyszerű feladatr $x, \theta, \dot{x}, \dot{\theta}$ -egyensúlyozási probléma annyiban különbözik az eddigiekben tárgyaltaktól, hogy az állapotváltozói folytonosak. A cselekvés általában diszkrét: lökés jobbra vagy lökés balra, az úgynevezett **bang-bang szabályozás** (**bang-bang control**).

21.9. ábra - A mozgó kocsi tetején álló hosszú rúd egyensúlyozásának vizsgálatára szolgáló berendezés. A kocsit az $x, \theta, \dot{x}, \dot{\theta}$ állapotváltozókat megfigyelő szabályzó jobbra vagy balra lökheti.



Az első – ezen probléma tanulását célzó – munkát Michie és Chambers végezte (Michie és Chambers, 1968). Az ő BOXES nevű algoritmusuk mindössze 30 próbálkozás után képes volt a rudat több mint egy óráig egyensúlyban tartani. Ráadásul, sok későbbi rendszertől eltérően, a BOXES egy valódi kocsit és rudat használt, nem szimulációt. Az algoritmus először is kvantálta, azaz tartományokra („dobozokra”) bontotta a négydimenziós állapotteret, innen jött az algoritmus neve is. Ezek után addig végeztek egy-egy kísérletet, amíg a rúd leesett vagy a kocsi elérte a pálya végét. Negatív megerősítést rendeltek az utolsó állapottér-tartományban az utolsó cselekvéshez, és ezt terjesztették vissza a megelőző szekvenciára. Azt találták, hogy a kvantálás akkor okoz bizonyos problémákat, ha a berendezést más pozícióból indították, mint amiket a tanítás során használtak, ami arra utal, hogy az általánosítás nem volt tökéletes. Jobb általánosítóképesség és gyorsabb tanulás érhető el, ha egy olyan algoritmust használunk, amely *adaptívan* osztja részekre az állapotteret, a jutalom megfigyelt változásainak megfelelően. Manapság egy *háromszoros* invertált inga egyensúlyozása közönséges feladatnak számít – ez már a legtöbb ember ügyességét messze meghaladja.

5. Stratégiakeresés

Created by XMLmind XSL-FO Converter.

A megerősítéses tanulással kapcsolatos, utolsó általunk tárgyalt megközelítés **stratégiakeresés** (**policy search**) néven ismert. Bizonyos értelemben a stratégiakeresés a legegyszerűbb a fejezet összes módszere közt. Az alapötlet annyi, hogy engedjük változni a stratégiát, amíg javul, és utána állítsuk le a változást.

A vizsgálatot kezdjük magával a stratégiával. Emlékezzünk vissza, hogy a π stratégia egy olyan függvény, amely állapotokat képez le cselekvésekre. Mi elsősorban olyan *paraméterezet* π stratégiák vizsgálatában vagyunk érdekeltek, amelyeknek sokkal kevesebb paramétere van, mint ahány állapot található az állapottérben (éppúgy, mint az előző alfejezetben). Például paraméterezett Q-függvényekkel reprezentálhatjuk a π stratégiát, egy-egy Q-függvényt rendelve minden cselekvéshez, és azt a cselekvést választjuk, amely a legnagyobb jósolt értéket adja:

$$\pi(s) = \max_a \hat{Q}_\theta(a,s) \qquad (21.13)$$

Fontos

Minden egyes Q-függvény lehet a θ paraméterek lineáris függvénye, mint a (21.9) egyenletben, vagy lehet nemlineáris függvény, mint például egy neurális háló. A stratégiakeresés ezek után a θ paramétereket változtatja oly módon, hogy javítsa a stratégiát. Vegyük észre, hogy ha a stratégiát Q-függvényekkel reprezentáljuk, akkor a stratégiakeresés egy Q-függvény tanulási eljárást eredményez. *Ez a folyamat azonban nem azonos a Q-tanulással!* A függvényapproximációt használó Q-tanulásban az algoritmus egy olyan θ-t keres, amelyre a $\hat{Q}_\theta$ „közel" van Q^*-hoz, az optimális Q-függvényhez. A stratégiakeresés ezzel szemben olyan θ-t keres, amely jó működést eredményez, az eredményül kapott értékek alapvetően eltérhetnek.[6] Egy másik nyilvánvaló példa a két dolog különbségére az az eset, amikor a $\pi(s)$ stratégiát úgy számítjuk, hogy 10 lépés mélységben előretekintő keresést végzünk egy közelítő $\hat{U}_\theta$ hasznosságfüggvény alapján. A jó eredményt adó θ messze lehet attól, amely az $\hat{U}_\theta$-t az igazi hasznosságfüggvényhez hasonlóvá teszi.

A (21.13) egyenletben bemutatott stratégiareprezentáció egyik problémája, hogy diszkrét cselekvések esetén a stratégia a paraméterek *nemfolytonos* függvénye.[7] Azaz lesznek olyan θ értékek, amelyeknél végtelen kis változás a θ-ban azt eredményezi, hogy a stratégia egyik cselekvésről a másikra vált. Ez azt is jelenti, hogy a stratégia értéke is változhat nemfolytonos módon, ami nehézzé teszi a gradiensalapú eljárások alkalmazását. Emiatt a stratégiakeresési eljárások gyakran használják a $\pi_\theta(s, a)$ **sztochasz-tikus stratégia** (**stochastic policy**) reprezentációt, amely az s állapotban az a cselekvés választásának valószínűségét specifikálja. Egy népszerű reprezentáció a **szoftmax függvény** (**softmax function**):

$$\pi_\theta(s,a) = \exp\left(\hat{Q}_\theta(a,s)\right) / \sum_{a'} \exp\left(\hat{Q}_\theta(a',s)\right)$$

A szoftmax függvény közel determinisztikussá válik, ha az egyik cselekvés sokkal jobb, mint a többi, de mindig differenciálható θ szerint; így a stratégia értéke (amely folytonos módon függ a cselekvésválasztási valószínűségtől) differenciálható függvénye lesz θ-nak.

Nézzünk most módszereket a stratégia javítására. Kezdjük a legegyszerűbb esettel: determinisztikus stratégia, determinisztikus környezetben. Ebben az esetben a stratégia értékelése triviális: egyszerűen végrehajtjuk a stratégiát, és megfigyeljük az összegyűjtött jutalmat, ez adja számunkra a **stratégia értéké**t (**policy value**), $\rho(\theta)$-t. A stratégia javítása ezek után egy standard optimalizációs probléma, ilyeneket a 4. fejezetben tárgyaltunk. Követhetjük a **stratégiagradiens** (**policy gradient**) vektort, $\nabla_\theta \rho(\theta)$-t, feltéve, hogy $\rho(\theta)$ differenciálható. Másik lehetőség, ha az **empirikus gradiens**t (**empirical gradient**) követjük hegymászó módszerrel – azaz kiértékeljük a stratégiát minden egyes paraméter kis megváltozása esetére. A szokásos feltételek esetén ez a stratégiatér egy lokális optimumához fog konvergálni.

Ha a környezet (vagy a stratégia) sztochasztikus, akkor a helyzet nehezebbé válik. Tegyük fel, hogy hegymászó módszert próbálunk alkalmazni, ami azt kívánja, hogy összehasonlítsuk $\rho(\theta)$-t $\rho(\theta + \Delta\theta)$-val valamilyen kis $\Delta\theta$ esetén. Az a probléma, hogy a teljes jutalom nagyon nagyokat változhat kísérletről kísérletre, így a kisszámú kísérletből számított stratégiaérték nagyon megbízhatatlan lesz – két ilyen becslés összehasonlítása pedig még

[6] Triviális, hogy a $\hat{Q}_\theta(a,s) = Q^*(a,s)/10$ közelítő Q-függvény optimális teljesítményt ad, bár egyáltalán nincs közel Q^*-hoz.
[7] Folytonos cselekvési terekre a stratégia a paraméterek sima függvénye lehet.

Created by XMLmind XSL-FO Converter.

megbízhatatlanabb. Egy lehetséges megoldás, ha egyszerűen sok kísérletet futtatunk le, és a minta variancájával mérjük, hogy elegendő kísérletet futtattunk-e már ahhoz, hogy megbízhatóan jelezni tudjuk a $\rho(\theta)$ javításának irányát. Sajnálatos módon ez sok valós probléma esetén nem praktikus, mert az egyes kísérletek drágák, időigényesek és esetleg veszélyesek lehetnek.

A $\pi_\theta(s, a)$ sztochasztikus stratégia esetén lehetőségünk van arra, hogy a θ-ban végrehajtott kísérletek eredményei alapján a $\nabla_\theta \rho(\theta)$ gradiens torzítatlan becslését állítsuk elő θ-ban. Az egyszerűség kedvéért a becslést arra az egyszerű esetre vezetjük le, amikor nem szekvenciális környezetben közvetlenül az s_0 startállapotbeli cselekvés után megkapjuk a jutalmat. Ebben az esetben a stratégiaérték egyszerűen a jutalom várható értéke, tehát:

$$\nabla_\theta \rho(\theta) = \nabla_\theta \sum_a \pi_\theta(s_0, a) R(a) = \sum_a \left(\nabla_\theta \pi_\theta(s_0, a)\right) R(a)$$

Most egy egyszerű trükköt alkalmazunk, hogy ezt az összegzést approximálni tudjuk a $\pi_\theta(s, a)$ által meghatározott valószínűség-eloszlásból generált mintákkal. Tegyük fel, hogy összesen N kísérletünk van, és a j-edik kísérletben az a_j cselekvést választottuk. Ekkor

$$\nabla_\theta \rho(\theta) = \sum_a \pi_\theta(s_0, a) \cdot \frac{\left(\nabla_\theta \pi_\theta(s_0, a)\right) R(a)}{\pi_\theta(s_0, a)} \approx \frac{1}{N} \sum_{j=1}^{N} \frac{\left(\nabla_\theta \pi_\theta(s_0, a_j)\right) R(a_j)}{\pi_\theta(s_0, a_j)}$$

Tehát a stratégia igazi gradiensét egy szummával közelítettük, ahol a tagok az egyes kísérletek cselekvésválasztási valószínűségeinek gradiensét tartalmazzák. A szekvenciális esetre ez a következőképp általánosítható:

$$\nabla_\theta \rho(\theta) \approx \frac{1}{N} \sum_{j=1}^{N} \frac{\left(\nabla_\theta \pi_\theta(s, a_j)\right) R_j(s)}{\pi_\theta(s, a_j)}$$

minden egyes bejárt s állapotra. Itt a_j a j-edik kísérlet során az s állapotban választott cselekvés, és $R_j(s)$ a j-edik kísérlet során az s állapotból kiindulva gyűjtött összes jutalom. Az eredményül kapott algoritmust `REINFORCE`-nak nevezték el (Williams, 1992). Rendszerint sokkal hatékonyabb, mint a minden egyes θ-ban sok kísérletet használó hegymászó eljárás. Viszont sajnos még mindig sokkal lassabb annál, mint amire szükségünk lenne.

Fontos

Vizsgáljuk a következő feladatot: adott két blackjack[8] program, döntsük el, hogy melyik a jobb! Egyik lehetőség, ha egy közös „bank” ellen játszatjuk őket egy adott számú kártyaleosztásban, és megnézzük, hogy melyik nyert többet. Ezzel az a probléma – mint láttuk –, hogy mindegyik program nyereségei nagy fluktuációt fognak mutatni annak függvényében, hogy milyen lapokat kapott. Egy kézenfekvő megoldás erre, ha előre generálunk egy sor leosztást, egy *leosztáshalmazt*. Ezzel elkerüljük a különböző kártyaleosztások okozta mérési hibát. Ez a `PEGASUS` algoritmus ötletének alapja (Ng és Jordan, 2000). Az algoritmus olyan területeken alkalmazható, ahol rendelkezésünkre áll egy szimulátor, ezáltal a „véletlen” kísérlet kimenetelek megismételhetővé válnak. Az algoritmus N véletlen számsorozatot generál előre, mindegyik felhasználható arra, hogy egy tetszőleges stratégia alapján kísérletet futtassunk vele. A stratégiakeresést úgy hajtjuk végre, hogy mindegyik stratégiajelöltet az alapján értékelünk, hogy *ugyanazt* a véletlen sorozat halmazt használja a kísérletek kimeneti értékeinek meghatározásához. Megmutatható, hogy az *összes* stratégia értékének jó becsléséhez szükséges véletlen sorozatok száma csak a stratégiatér komplexitásától függ, és egyáltalán nem függ a mögöttes terület komplexitásától. A `PEGASUS` algoritmussal számos területen (például az autonóm helikoptervezetés területén) több hatékony stratégiát fejlesztettek ki (lásd 21.10. ábra).

21.10. ábra - Időben eltolt képek egymásra másolása útján kapott eredő kép, amelyen egy nagyon bonyolult „körberepülés orral a kör középpontja fele” manővert hajtanak végre. A helikoptert egy Pegasus stratégiakereső algoritmussal fejlesztett stratégiával vezérlik. Egy szimulátormodellt fejlesztettek a valós helikopter egyes vezérlési

[8] 21-es játékként is ismert.

Created by XMLmind XSL-FO Converter.

beavatkozásokra adott válaszainak vizsgálatára. Ezek után egész éjszaka futtatták az algoritmust a szimulátoron. Egy sor vezérlőt fejlesztettek ki különböző manőverekre. Minden esetben, amikor távirányítást használtak, messze jobb volt az eredmény, mint a képzett humán pilótáké. (A képet Andrew Ng engedélyével közöljük.)



6. Összefoglalás

Ebben a fejezetben a megerősítéses tanulással foglalkoztunk: azzal, hogy egy ismeretlen környezetben működő ágens hogyan válhat gyakorlottá pusztán megfigyelései, valamint az esetenként kapott jutalmak felhasználásával. A megerősítéses tanulás a teljes MI-témakör mikrovilágának is tekinthető, amelyet azonban az előrehaladás érdekében egy sor egyszerűsítő feltétel mellett tárgyaltunk. A következő fontosabb megállapításokat tettük:

- Az ágens teljes felépítése határozza meg azt, hogy milyen információt kell megtanulni. A tárgyalt három fő felépítési lehetőség: egyrészt a modellalapú, amely a T modellt és az U hasznosságfüggvényt használja; másrészt a modell nélküli felépítés, amely a Q cselekvésérték-függvényt használja, és a reflexszerű, amely a π stratégiát használja.

- A hasznosságértékek háromféle megközelítés alapján tanulhatók:

  1. A **közvetlen hasznosságbecslés** (**direct utility estimation**), amely a megfigyelt hátralévő jutalmat használja egy-egy állapot esetén, mint a hasznosság tanulásának közvetlen jellemzőjét.

  2. Az **adaptív dinamikus programozás** (**ADP**) (**adaptive dynamic programming**) megfigyelései alapján egy modellt és egy jutalomfüggvényt tanul, ezek után érték- vagy stratégiaiterációt használ ahhoz, hogy az állapotok hasznosságbecsléséhez vagy az optimális stratégiához eljusson. Az ADP optimálisan használja fel a környezetnek a szomszédos állapotokra vonatkozó struktúrájából adódó lokális kényszereket.

  3. Az **időbeli különbség** (**IK**) (**temporal-difference**, **TD**) megközelítés úgy módosítja a hasznosságbecsléseket, hogy megfeleljenek az adott állapotot követő állapotokra vonatkozó becsléseknek. Tekinthetjük egyszerűen az ADP megközelítés olyan approximációjának is, amely a tanuláshoz nem igényel modellt. Mindamellett, ha a megtanult modellt pszeudotapasztalatok generálására használjuk, gyorsabb tanuláshoz juthatunk.

- A cselekvésérték-függvények vagy Q-függvények ADP- vagy IK-megközelítéssel tanulhatók. IK esetén a Q-tanulás sem a tanulás, sem a cselekvés kiválasztásának fázisában nem igényli modell használatát. Ez

Created by XMLmind XSL-FO Converter.

egyszerűsíti a tanulási feladatot, de potenciálisan korlátozhatja a bonyolult környezetben való tanulási képességet, mivel az ágens nem képes szimulálni a lehetséges cselekvéssorozatok eredményét.

- Ha a tanuló ágens a tanulás során a cselekvések kiválasztásáért is felelős, akkor kompromisszumot kell kötnie a választott cselekvés jelenlegi hasznossága és hasznos új információ tanulásának lehetősége között. A felfedezési probléma egzakt megoldása kezelhetetlen, de néhány egyszerű heurisztika elég jó megoldásra vezet.

- Nagy állapotterek esetén a megerősítéses tanuló algoritmusnak közelítőfüggvényreprezentációt kell használnia az állapotok feletti általánosítóképesség érdekében. Az időbeli különbség jel közvetlenül felhasználható a neurális háló jellegű reprezentációk paramétereinek frissítésére.

- A **stratégiakeresés**i módszerek közvetlenül a stratégia reprezentációját használják, a megfigyelt teljesítmény alapján próbálva meg javulást elérni. Súlyos problémát okoz sztochasztikus területeken a teljesítményben mutatkozó nagy variancia. Szimulációval vizsgálható területeken ez megoldható úgy, hogy előre rögzítjük a véletlen hatásokat.

A vezérlési stratégiát biztosító program kézi kódolásának elkerülhetősége miatt a megerősítéses tanulás a gépi tanulás egyik legaktívabban művelt területe. A robotikai alkalmazások különösen értékesnek ígérkeznek, de ezek *folytonos, sokdimenziós,* csak *részlegesen megfigyelhető* környezetek kezelésére képes módszereket igényelnek, ráadásul a sikeres működéshez több ezer vagy akár millió elemi cselekvésre is szükség lehet.

6.1. Irodalmi és történeti megjegyzések

Turing javasolta a megerősítéses tanulás megközelítést, bár nem volt teljesen meggyőződve annak hatékonyságáról (Turing, 1948; 1950). Azt írta: „A büntetés és jutalmazás legjobb esetben is csak egy része lehet a tanítási folyamatnak.” Arthur Samuel munkája volt valószínűleg az első sikeres kutatás a gépi tanulás területén (Samuel, 1959). Bár munkája nem volt formálisan megfogalmazva, és számos hiányossága volt, mégis a megerősítéses tanulás legtöbb modern gondolatát tartalmazta, beleértve az időbeli különbségképzést és a függvényekkel történő approximációt. Körülbelül ugyanebben az időben az adaptív szabályozások területével foglalkozó kutatók (Widrow és Hoff, 1960), Hebb munkájára építve (Hebb, 1949), egyszerű hálókat tanítottak a delta-szabály segítségével. (A neurális hálók és a megerősítéses tanulás ezen korai kapcsolata vezethetett ahhoz a félreértéshez, hogy ez utóbbi az előbbinek részterülete.) Michie és Chambers rúdegyensúlyozás-vezérlési kutatása szintén egy függvényapproximációval történő megerősítéses tanulási módszernek tekinthető (Michie és Chambers, 1968). A megerősítéses tanulással foglalkozó pszichológiai irodalom sokkal régebbi, Hilgard és Bower jó áttekintést adnak erről (Hilgard és Bower, 1975). A méhek nektárgyűjtési szokásainak kutatása közvetlen bizonyítékokat szolgáltatott a megerősítéses tanulás működésére állatoknál. Nyilvánvaló neurális kapcsolat van a jutalomjellel. Ennek formája egy nagy neurális leképezés a nektár bevitelét érzékelő szenzoroktól a motoros kéregig (Montague és társai, 1995). A sejtszintű emlékezéssel kapcsolatban folytatott kutatások azt sugallják, hogy a főemlősök agyában a dopamin rendszer valami olyasmit valósít meg, ami a függvénytanulásra emlékeztet (Schultz és társai, 1997).

A megerősítéses tanulás és a Markov döntési folyamatok közti kapcsolatra először Werbos mutatott rá (Werbos, 1977), de a megerősítéses tanulás MI-n belüli fejlesztése a University of Massachusetts falain belül kezdődött az 1980-as évek elején (Barto és társai, 1981). Sutton írása jó történeti áttekintést ad (Sutton, 1988). Ennek a fejezetnek a (21.3) egyenlete egy speciális, a $\lambda = 0$ melletti esete Sutton általános IK(λ) algoritmusának. Az IK(λ) egy sorozat összes állapotának értékét frissíti oly módon, hogy a múltban t időlépés távolságra lévő átmenetekhez egy λ^t szerint csökkenő faktor vezet. Az IK(1) azonos a Widrow–Hoff- vagy delta-szabállyal. Bradtke és Barto érvelésére (Bradtke és Barto, 1996) építve Boyan azt állítja, hogy az IK(λ) és a hozzá kapcsolható algoritmusok nem használják ki hatékonyan a tapasztalatokat (Boyan, 2002). Szerinte ezek lényegében online regressziós algoritmusok, amelyek sokkal lassabban konvergálnak, mint az offline regresszió. Az ő LSTD(λ) algoritmusa egy olyan online regreszszió, ami az offline regresszióval azonos eredményt ad.

Az időbeli különbség tanulásnak és a szimulációs tapasztalatok modellbázisú generálásának kombinálását Sutton a DYNA architektúrában javasolta (Sutton, 1990). A prioritásos végigsöprés ötletét egymástól függetlenül Moore és Atkeson (Moore és Atkeson, 1993), illetve Peng és Williams (Peng és Williams, 1993) vetették fel. A *Q*-tanulást Watkins dolgozta ki PhD-disszertációjában (Watkins, 1989).

A rabló problematikát, amely a nemszekvenciális döntések területén történő felfedezést modellezi, részletesen Berry és Fristedt tanulmányozták (Berry és Fristedt, 1985). Számos helyzetben optimális felfedezési stratégia nyerhető a **Gittins-index** segítségével (Gittins, 1989). Barto és munkatársai a szekvenciális döntési

Created by XMLmind XSL-FO Converter.

problémákban alkalmazott felfedezési módszerek számos változatát tárgyalják (Barto és társai, 1995). Kearns és Singh, valamint Brafman és Tennenholtz olyan algoritmusokat írnak le (Kearns és Singh, 1998; Brafman és Tennenholtz, 2000), amelyek ismeretlen környezet felfedezését végzik, és bizonyított, hogy polinomiális idő alatt konvergálnak közel-optimális stratégiákhoz.

A megerősítéses tanulás területén a függvényapproximáció alkalmazása Samuel munkájáig vezethető vissza, aki mind lineáris, mind nemlineáris kiértékelő függvényeket alkalmazott, továbbá a tulajdonságok terének csökkentésére tulajdonságszelekciós algoritmusokat is használt. A későbbiekben bevezetett módszerek közé tartozik a **CMAC** (Cerebellar Model Articulation Controller) (Albus, 1975), amely lényegében átlapolódó lokális kernelfüggvények összegéből, és a Barto és társai által bevezetett asszociatív neurális hálókból áll (Barto és társai, 1983). Napjainkban a neurális hálók a legnépszerűbb függvényapproximátorok. A legismertebb alkalmazás az ebben a fejezetben bemutatott TD-Gammon (Tesauro, 1992, 1995). A neurális hálókkal megvalósított IK-tanulóknak súlyos problémája az, hogy hajlamosak elfelejteni korábbi tapasztalataikat. Ez különösen igaz azokra térrészekre, amelyeket a szakértelmük megszerzése után elkerülnek. Ez katasztrofális következményekkel járhat, ha helyzetek újra jelentkeznek. A **példányalapú tanulás** (**instance-based learning**) alkalmazásával ez a probléma elkerülhető (Ormoneit és Sen, 2002; Forbes, 2002).

A fügvényapproximációt használó megerősítéses tanulás konvergenciája kimondottan technikai kérdés. Lineáris approximáló függvények alkalmazása esetén az IK tanulási eredmények folyamatosan javulnak (Sutton, 1988; Dayan, 1992; Tsitsiklis és Van Roy, 1997), de nemlineáris függvények esetén számos divergenciát mutató példát találtak (a jelenség tárgyalását lásd a (Tsitsiklis és Van Roy, 1997)-ben). Papavassiliou és Russell a megerősítéses tanulás új formáját adták (Papavassiliou és Russell, 1999), amely tetszőleges struktúrájú függvényapproximátor esetén konvergál, feltéve, hogy található a megfigyelt adatokra egy legjobban illeszkedő approximáció.

A stratégiakeresésre Williams hívta fel a figyelmet (Williams, 1992), ő volt az, aki kifejlesztette a `REINFORCE` algoritmuscsaládot. A későbbi munkák megerősítették és általánosították a stratégiakeresés konvergenciájára vonatkozó eredményeket: (Marbach és Tsitsiklis, 1998; Sutton és társai, 2000; Baxter és Bartlett, 2000). Ng és Jordan (2000) alkották meg a `PEGASUS` algoritmust (Ng és Jordan, 2000), bár hasonló technikák már Van Roy PhD-disszertációjában is megtalálhatók (Van Roy, 1998). Mint a fejezetben említettük, a *sztochasztikus* stratégiák teljesítménye a paramétereik folytonos függvénye, ami elősegíti a gradiensalapú technikák alkalmazhatóságát. Nem ez az egyetlen előnyük: Jaakkola és társai amellett érvelnek, hogy részlegesen megfigyelhető környezetben a sztochasztikus stratégiák jobban működnek, mint a determinisztikusok, ha mindkét stratégia esetén a cselekvés pusztán a jelenlegi észlelésen alapul (Jaakkola és társai, 1995). (Ennek egyik oka, hogy a sztochasztikus algoritmusok kevésbé hajlamosak „beragadni", ha valamilyen eddig még nem látott akadály jelentkezik.) A 17. fejezetben rámutattunk, hogy részlegesen megfigyelhető MDF-ekben az optimális stratégiák inkább a *hiedelemállapot* determinisztikus függvényei, mint az aktuális megfigyeléseké. Ennek megfelelően még jobb eredményeket várnánk, ha a 15. fejezetben tárgyalt **szűrés**i (**filtering**) módszerekkel nyomon követnénk a hiedelemállapot alakulását. Sajnálatos módon a hiedelemállapot-terek sokdimenziósak és folytonosak, és az eddigiekben nem sikerült hatékony módszereket kifejleszteni a hiedelemállapotok megerősítéses tanulására.

A valós környezetekre az is jellemző, hogy gyakran rengeteg elemi cselekvési lépésre van szükség ahhoz, hogy jelentős jutalomhoz jussunk. Például ha egy robot focizik, akkor százezer elemi lábmozdulatot is tehet, mielőtt gólt lő. Egy szokásos módszer erre a **jutalomalakítás** (**reward shaping**), melyet eredetileg az állatok tanításánál alkalmaztak. Ennek része, hogy további jutalmakat adunk az ágensnek, ha az „haladást ér el". A foci esetén például jutalmat adhatunk, ha hozzáér a labdához vagy a kapu felé lő. Ezeket a jutalmakat rendszerint nagyon egyszerű kialakítani, és rendkívül felgyorsíthatják a tanulást, de megvan az a veszélye, hogy az ágens a tanulás során a „pszeudojutalmak" maximálását végzi el, nem az igazi jutalmakét. A foci tanulása esetén például a labdához való közelség és a labdával való sok-sok érintkezés, a „vibrálás", lehet ennek az eredménye. Ng és társai megmutatták, hogy az ágens így is megtanulja az optimális stratégiát, ha az $F(s, a, s')$ pszeudojutalom kielégíti az $F(s, a, s') = \gamma\, \Phi(s') - \Phi(s)$ egyenletet, ahol Φ az állapot tetszőleges függvénye (Ng és társai, 1999). A Φ-t úgy alkothatjuk meg, hogy az állapot valamely kívánatos vonásait tükrözze, például valamilyen részcél elérését vagy a célállapottól való távolságot.

Az összetett viselkedés kialakítását **hierarchikus megerősítéses tanulás**i (**hierarchical reinforcement learning**) módszerekkel segíthetjük elő, amelyek több absztrakciós szinten igyekeznek megoldani a problémát, a 12. fejezetben ismertetett **HFH-tervkészítés**hez (**HTN planning**) hasonlóan. Például a „góllövés" szétbontható „labdaszerzés", „a kapu felé cselezés" és „lövés" részekre, és ezek mind tovább bonthatók alacsonyabb szintű mozgásokra. Ezen a területen Forestier és Varaiya értek el alapvető eredményt (Forestier és Varaiya, 1978). Bebizonyították, hogy tetszőleges bonyolultságú alsóbb szintű viselkedés egyszerűen elemi

Created by XMLmind XSL-FO Converter.

cselekvésnek tekinthető az őt kiváltó magasabb szintű viselkedés szempontjából (bár az alsóbb szintű viselkedés által megvalósított cselekvések eltérő időt vehetnek igénybe). Parr és Russell, Dietterich, Sutton és társai, valamint Andre és Russell jelen kutatásaikban erre az eredményre építettek: olyan módszereket fejlesztettek, amelyek **részprogram**mal (**partial program**) látják el az ágenst, ezzel viselkedését valamilyen speciális hierarchikus struktúrába szorítva (Parr és Russell, 1998; Dietterich, 2000; Sutton és társai, 2000; Andre és Russell, 2002). A megerősítéses tanulást ezek után arra használják, hogy a részprogrammal konzisztens legjobb viselkedést tanítsák. A függvényapproximáció, a jutalomalakítás és a hierarchikus megerősítéses tanulás összekombinálása lehetővé teheti, hogy nagy bonyolultságú problémák megoldását kíséreljük meg a siker reményében.

Az irodalom áttekintéséhez jó kiindulópontot ad Kaelbling tanulmánya (Kaelbling és társai, 1996). A terület két úttörőjénk, Suttonnak és Bartónak a publikációja az architektúrákra és algoritmusokra koncentrál, megmutatva, hogy a megerősítéses tanulás összeköti a tanulás, a tervezés és a cselekvés elveit (Sutton és Barto, 1998). Bertsekas és Tsitsiklis inkább technikai részletekkel foglalkozó munkája a dinamikus programozás és a sztochasztikus konvergencia elméletének precíz megalapozását nyújtja (Bertsekas és Tsitsiklis, 1996). A *Machine Learning*, illetve a *Journal of Machine Learning Research* folyóiratok, továbbá az International Conferences on Machine Learning és a Neural Information Processing Systems rendezvények kiadványai közölnek gyakran megerősítéses tanulással foglalkozó cikkeket.

6.2. Feladatok

21.1.

Egy – a 4 × 3-as világhoz hasonló – egyszerű környezetben valósítson meg egy passzív tanuló ágenst! Hasonlítsa össze a közvetlen hasznosságbecslést, az IK-t és az ADP-t előzetesen nem ismert környezeti modellek esetén! Végezze el az összehasonlítást az optimális stratégiára és több, véletlen módon generált stratégiára! Melyikre konvergálnak gyorsabban a hasznosságbecslések? Mi történik, ha növeli a környezet méretét? (Vizsgáljon pályákat akadályokkal és azok nélkül!)

21.2.

A 17. fejezetben úgy definiáltuk az MDF számára **megfelelő stratégiá**t (**proper policy**), mint ami biztosan eléri a végállapotot. Mutassa meg, hogy egy passzív ADP-ágens képes egy olyan állapotátmenet-modell megtanulására, amelyre nézve π stratégiája nem megfelelő, még ha π megfelelő is az igazi MDF-re; az ilyen modellekkel $\gamma = 1$ esetben az értékmeghatározási lépés meghiúsulhat! Mutassa meg, hogy ez a probléma nem jelentkezik, ha az értékmeghatározást csak a kísérlet végeztével alkalmazzuk a megtanult modellre.

21.3.

Induljon ki egy passzív ADP-ágensből, és módosítsa úgy, hogy a fejezetben leírt közelítő ADP-algoritmust kapja! Két lépésben oldja meg a feladatot:

a. Építsen fel egy prioritássort a hasznosságbecslések módosítására! Valahányszor állapotmódosításra kerül, az őt megelőző állapotok is potenciálisan módosítandók, tehát be kell állítani őket a sorba. A sor inicializálását annak az állapotnak a felhasználásával végezzük, amelyből a legutolsó átmenet történt. Csak rögzített számú módosítást engedjen meg!

b. Kísérletezzen a prioritássor rendezésének különböző heurisztikáival, vizsgálja meg a tanulás sebességére és a számítási időre gyakorolt hatásukat!

21.4.

A 21.2. alfejezetben bemutatott közvetlen hasznosságbecslési módszerek elkülönített végállapotokat használtak a kísérlet végének jelzésére. Hogyan módosíthatók végállapottal nem rendelkező, leértékelt jutalmakat használó környezetekre?

21.5.

Hogyan használható az értékmeghatározási algoritmus úgy, hogy egy U hasznosságbecslés-halmazt és egy M becsült modellt használó ágensnek a helyes értékeket használó ágenshez képesti várható veszteségét számítsa?

21.6.

Created by XMLmind XSL-FO Converter.

Adaptálja a 2. fejezetben bemutatott porszívóvilágot megerősítéses tanulásra! Az ágens kapjon jutalmat minden felszívott szemétért, a helyére való visszaéréskor és kikapcsoláskor! Megfelelő észlelésekkel tegye megfigyelhetővé a világot! Kísérletezzen különböző megerősítéses tanulást végző ágensekkel! Szükség van-e függvényapproximációra a siker érdekében? Milyen approximátor működik erre az alkalmazásra?

21.7.

Valósítson meg egy felfedező megerősítéses tanuló ágenst, amely közvetlen hasznosságbecslést végez. Készítsen két változatot, az egyik táblázatos reprezentációt használjon, a másik a (21.9) egyenlet függvényapproximátorát. Hasonlítsa össze teljesítményüket az alábbi három különböző környezetben:

a. A fejezetben leírt 4 × 3-as világ.

b. Egy 10 × 10-es világ, amelyben nincsenek akadályok, és a (10, 10) mezőn van +1 jutalom.

c. Egy 10 × 10-es világ, amelyben nincsenek akadályok, és az (5, 5) mezőn van +1 jutalom.

21.8.

Írja fel az IK-tanulás paraméterfrissítési egyenletét a következő esetre:

$$\hat{U}(x,y) = \theta_0 + \theta_1 x + \theta_2 y + \theta_3 \sqrt{(x - x_g)^2 + (y - y_g)^2}$$

21.9.

Találjon ki megfelelő jellemzőket a sztochasztikus négyzetrácsvilágra (a 4 × 3-as világ általánosítása), amely világ több akadályt és több (+1, illetve –1 jutalommal jellemezhető) végállapotot tartalmaz!

21.10.

Számítsa ki a valós hasznosságfüggvényt és az *x*, *y* függvényében felírható legjobb lineáris approximációt (éppúgy, mint a (21.9) egyenletben tettük) a következő környezetekre:

a. Egy 10 × 10-es világ, amelyben a (10, 10) az egyetlen +1 végállapot mező.

b. Mint (a), de a (10, 1)-en van egy további –1 végállapot.

c. Mint (b), de tegyen 10 véletlen módon választott mezőre akadályt.

d. Mint (b), de vegyen fel egy falat (5, 2)-től (5, 9)-ig.

e. Mint (a), de a végállapot legyen az (5, 5).

A cselekvések a négy lehetséges irányba való determinisztikus lépések. Minden esetben háromdimenziós ábrákat használva végezze el az összehasonlítást! Mindegyik környezetben adjon javaslatot további olyan tulajdonságokra (*x*-en és *y*-on túl), amelyek javíthatják az approximációt, majd ábrázolja az eredményeket!

21.11.

Terjessze ki a standard játékkörnyezetet (6. fejezet) úgy, hogy tartalmazzon jutalomjelet is! Helyezzen két ágenst ebbe a környezetbe (természetesen osztozhatnak az ágensprogramon), és játszassa egymás ellen őket! Alkalmazza az általánosított IK frissítési szabályt (21.11 egyenlet) a kiértékelő függvény frissítésére. Lehet, hogy érdemes egy egyszerű súlyozott lineáris kiértékelő függvénnyel kezdeni, és egy egyszerű játékkal, mint a 3 × 3-as amőba (tic-tac-toe).

21.12.

Valósítsa meg a `REINFORCE` és `PEGASUS` algoritmusokat a 4 × 3-as világra, és alkalmazza egy saját maga által választott stratégiacsaládra! Értékelje az eredményeket!

21.13.

Created by XMLmind XSL-FO Converter.

Olvasnivaló. Vizsgálja meg a megerősítéses tanulás elveinek humán, illetve állati viselkedésre való alkalmazását!

21.14.

Olvasnivaló. Alkalmas absztrakt modellje-e az evolúciónak a megerősítéses tanulás? Milyen kapcsolat van – ha van egyáltalán – a beépített jutalomjelek és az evolúciós rátermettség közt?

Created by XMLmind XSL-FO Converter.

22. fejezet - Kommunikáció

Ebben a fejezetben meglátjuk, hogy az ágensek miért akarhatnak információt tartalmazó üzeneteket küldeni egymásnak, illetve azt, hogy ezt hogyan tehetik meg.

Alkonyodik az Amboseli Nemzeti Park szavannás erdőségében, közel a Kilimandzsáró tövéhez. A fehérbarkójú cerkófmajom vagy grivet (*Cercopithecus aethiops – a ford.*) egy csoportja épp élelem után kutat, amikor az egyik grivet hangos kiáltó hívást hallat. A csoport többi tagja felismeri, hogy ez egy leopárdra figyelmeztető felhívás (amely különbözik a sasra figyelmeztető rövid köhintéstől vagy a kígyókra vonatkozó sziszegéstől), és a fákra sietnek. A grivet sikeresen kommunikált a csoporttal.

A **kommunikáció** (**communication**) az információ szándékolt átadása közös jelrendszerből választott jel létrehozása és észlelése által. Az állatok többsége használ jelzéseket fontos üzenetek kifejezésére: itt van élelem, ragadozó a közelben, gyere közelebb, vonulj vissza, párosodjunk. Egy részlegesen megfigyelhető világban a kommunikáció segít az ágenseknek sikeresnek lenni, mivel megfigyelhetnek mások által észlelt vagy kikövetkeztetett információt.

Az embereket a **nyelv**ként (**language**) ismert, strukturált üzenetekből álló összetett rendszer különbözteti meg más élőlényektől, amely lehetővé teszi számunkra, hogy a világról szóló ismereteink nagy részét átadhassuk egymásnak. Bár a csimpánzok, delfinek és más emlősök több száz jeles szókinccsel rendelkeznek, és van valamiféle hajlamuk ezek összekapcsolására, az ember az egyedüli faj, amely minőségileg különböző üzenetek korlátlan számát képes megbízhatóan átadni.

Természetesen vannak más tulajdonságok, amelyek egyedülállóan emberiek: egyetlen más faj sem visel ruhát, készít művészeti alkotásokat vagy néz naponta három órán át televíziót. Azonban amikor Turing a tesztjét javasolta (lásd 1.1. alfejezet), a nyelvre alapozta azt, mivel a nyelv bensőséges kapcsolatban áll a gondolkodással. Ebben a fejezetben elmagyarázzuk, hogyan működik egy kommunikáló ágens, és bemutatjuk az angol nyelv egy egyszerűsített részletét.

1. A kommunikáció mint cselekvés

Egy ágens számára a beszéd létrehozása lehetséges cselekvéseinek egyike. Ezt **szólásaktus**nak (**speech act**) nevezzük. A „beszédet” olyan értelemben használjuk, mint „szabad közlést”, nem mint „beszélgetést”, így az elektronikus levelezés, a mutogatás és a jelbeszéd használata mind szólásaktusnak számít. Az angol nyelvben nincs általános szó a beszédet létrehozó ágensre, amely beszél, ír vagy másképp fejezi ki magát. A **beszélő** (**speaker**), a **hallgató** (**listener**) és a **megnyilatkozás** (**utterance**) kifejezéseket használjuk a kommunikáció bármilyen módjának általános leírására. A **szó** (**word**) kifejezést használjuk mindenféle szokásos kommunikációs jelre.

Miért foglalkozna egy ágens szólásaktus létrehozásával, amikor „hagyományos” cselekvést is végrehajthat? A 12. fejezetben láttuk, hogy többágenses környezetekben az ágensek használhatják a kommunikációt annak érdekében, hogy közös tervet alakítsanak ki. Például a wumpus világot felfedező ágensek egy csoportja együtt (egyéni vagy csoportos) előnyhöz jut, ha képes a következőket megtenni:

- **Megkérdez**ik (**query**) egymást a világ különféle területeiről. Ezt tipikusan kérdések feltevésével végzik: *Érezted a wumpus bűzét valahol?*
- **Értesít**ik (**inform**) egymást a világról. Ezt kijelentésekkel teszik meg: *Szellő van itt, a [3, 4]-en.* Egy kérdés megválaszolása egy másik fajta értesítés.
- **Kér**nek (**request**) más ágenseket cselekvések végrehajtására: *Kérlek segíts az aranyat vinni!* Néha egy **indirekt szólásaktus** (**indirect speech act**) (egy kijelentés vagy kérdés formájú kérés) udvariasabbnak számít: *Használni tudnék egy kis segítséget ennek a cipelésében.* Egy fennhatósággal rendelkező ágens adhat parancsokat (*Alfa jobbra menj; Bravo és Charlie balra*), és egy hatalommal rendelkező ágens fenyegethet (*Ide nekem az aranyat, vagy…*). Ezen szólásaktusokat együtt **direktívá**knak (**directive**s) nevezik.
- **Tudomásul vesz**ik (**acknowledge**) a kéréseket: *Rendben.*
- **Megígér**nek (**promise**) vagy hozzájárulnak egy tervhez: *Én lelövöm a wumpust; te megragadod az aranyat.*

Created by XMLmind XSL-FO Converter.

Minden szólásaktus befolyásolja a világot a levegőmolekulák rezgetésével (vagy ekvivalens hatással más médiumban), és ily módon megváltoztatja más ágensek mentális állapotát, valamint végső fokon a jövőbeli cselekedeteiket is. Bizonyos szólásaktusok információt juttatnak a hallgatóhoz, feltételezve, hogy a hallgató döntéshozatalát megfelelően befolyásolja az információ. Mások sokkal célzottabban késztetik a hallgatót valamilyen cselekvés véghezvitelére. A szólásaktusok további osztálya a **deklaratív** (**declarative**), amely sokkal közvetlenebb hatást gyakorol a világra: *férjnek és feleségnek nyilvánítom önöket* vagy *Harmadik találat, ön kiesett.* Természetesen a hatást az érintett ágensek mentális állapotai komplex hálózatának létrehozása vagy tudomásulvétele éri el: házasnak lenni vagy kiesni, olyan állapotok, amelyeket megállapodások rögzítenek és nem a világ „fizikai” jellemzői.

A kommunikáló ágens feladata annak eldöntése, hogy *mikor* van szükség egy szólásaktusra, és hogy *melyik* a helyénvaló az összes lehetséges közül. A szólásaktusok megértésének problémája hasonlatos más **megértés**i (**understanding**) problémákhoz, mint például a képek megértése vagy a betegségek diagnosztizálása. Kapunk egy halmazt többértelmű bemenetekkel, amikből visszafelé haladva azt kell eldöntenünk, hogy a világ mely állapota hozhatta létre őket. Azonban mivel a beszéd tervezett cselekvés, a megértés magában foglalja a tervfelismerést is.

1.1. A nyelv alapjai

A **formális nyelv**et (**formal language**) **karakterfüzérek** (**strings**) (lehetséges, hogy végtelen) halmazaként definiáljuk. Minden egyes füzér az úgynevezett **záró** (vagy **terminális**) **szimbólum**ok (**terminal symbol**s) – amelyeket néha szavaknak hívunk – összekapcsolt sorozata. Például az elsőrendű logikában a záró szimbólumok között van a $\wedge$ és a P, és egy tipikus füzér a „$P \wedge Q$”. A „$P\ Q \wedge$” füzér nem része a nyelvnek. A formális nyelvek, mint például az elsőrendű logika vagy a Java, szigorú matematikai definíciókkal rendelkeznek. Ezzel ellentétben a **természetes nyelvek** (**natural language**s), mint például a kínai, a dán és az angol, nem rendelkeznek szigorú definícióval, hanem beszélők egy közössége használja. Ebben a fejezetben a természetes nyelveket megpróbáljuk formális nyelveknek tekinteni, bár tisztában vagyunk azzal, hogy az illeszkedés nem lesz tökéletes.

A **nyelvtan** (**grammar**) a nyelvet meghatározó szabályok véges halmaza. A formális nyelveknek mindig van egy hivatalos nyelvtana, amelyet kezelési útmutatókban vagy könyvekben határoznak meg. A természetes nyelveknek nincs hivatalos nyelvtana; bár a nyelvészek törekszenek a nyelv tulajdonságainak felfedezésére tudományos vizsgálatokkal, majd ezen felfedezések rögzítésére egy nyelvtanban. Eddig egyetlen nyelvész sem járt teljes sikerrel. Figyeljük meg, hogy a nyelvészek tudósok, akik a nyelvet úgy próbálják definiálni, ahogy az *létezik*. Vannak olyan előíró nyelvészek is, akik megpróbálják diktálni, hogy a nyelvnek milyennek *kellene lennie*. Olyan szabályokat alkotnak, mint például a „használd az ikes ragozást”, amelyeket néha megjelentetnek stílusútmutatókban, de kevés valódi hatásuk van a beszélt nyelvre.

Mind a formális, mind a természetes nyelvek jelentést, avagy **szemantiká**t (**semantics**) rendelnek minden egyes érvényes füzérhez. Például az aritmetika nyelvén lenne egy szabályunk, hogy ha „X” és „Y” kifejezések, akkor „$X + Y$” szintén kifejezés, amelynek a szemantikája X és Y összege. Természetes nyelvekben hasonlóan fontos a füzérek **pragmatiká**ját (**pragmatics**) is megérteni: a füzér aktuális jelentését, ahogy az elhangzik egy adott szituációban. A jelentés nemcsak a szavakban rejlik, hanem a szavak *in situ* értelmezésében.

A legtöbb nyelvtani szabály formalizmus a **kifejezésstruktúra** (**phrase structure**) ötletére épül – azaz arra, hogy a füzérek **kifejezés**eknek (**phrase**s) nevezett, különböző kategóriából származó részfüzérekből állnak össze. Például „a wumpus”, „a király” és „a sarokban levő ágens”[1] kifejezések mind az úgynevezett **főnévi kifejezés** (**noun phrase**) (vagy röviden *NP*) kategória példái. Két ok létezik arra, hogy a kifejezéseket ilyen módon megkülönböztessük. Az egyik, hogy a kifejezések általában természetes szemantikus elemeknek felelnek meg, amelyekből a kijelentés jelentése megalkotható; például a főnévi kifejezések a világ objektumaira vonatkoznak. A másik, hogy a kifejezések kategorizálása segít a nyelv megengedett füzéreinek leírásában. Azt mondhatjuk, hogy a főnévi kifejezések bármelyike kombinálható egy **igei kifejezés**sel (**verb phrase**) (*VP*), mint például „is dead” (halott), hogy közösen egy olyan kifejezést formáljanak, amelyik a **mondat** (**sentence**) (*S*)

[1] A fejezetben hozott példák fordításakor a következőképpen jártunk el: az olyan angol kifejezéseket, amelyek magyarul is szemléletesek az adott mondanivaló szempontjából, magyar fordításban adjuk meg. A kifejezések és részletesebb példák többségénél ez nem követhető, mivel a szerző az angol nyelv sajátosságaihoz illeszti a fejezet ismeretanyagának és példáinak megfogalmazását, így a fordítás lényegi változást hozott volna a fejezet tartalmában is. Ezeken a részeken a magyar változatban is meghagyjuk az eredeti angol kifejezéseket, szükség esetén rövid magyar fordításukkal kiegészítve. A szerzők létrehozzák az angol nyelv egy formális részhalmazát, amin bemutatják a fejezetben ismertetett módszerek alkalmazását. Ezt is változatlan formában helyezzük át a magyar nyelvű kiadásba. Ezen részek magyar fordítását sem készítjük el, mivel a fordítás a mondanivaló szempontjából lényegtelen. (*A ford.*)

Created by XMLmind XSL-FO Converter.

kategóriába tartozik. A főnévi és igei kifejezés közbülső fogalma nélkül sokkal bonyolultabb lenne elmagyarázni, hogy miért jó mondat a „the wumpus is dead” és miért nem a „wumpus the dead is”.

Fontos

Generálóképesség

A nyelvtani formalizmusok osztályozhatók **generálóképesség**ük (**generative capacity**) szerint: azoknak a nyelveknek a halmaza szerint, amelyeket képesek reprezentálni. Chomsky a nyelvtani formalizmusok négy osztályát írta le, amelyek csak az átíró szabályok formájában különböznek (Chomsky, 1957). Az osztályok hierarchiába rendezhetők, ahol minden osztály felhasználható az összes olyan nyelv leírására, ami leírható egy nála gyengébb osztállyal, és még néhány további nyelvre is. Itt most felsoroljuk a hierarchiát, a legerősebbel kezdve.

A **rekurzívan felsorolható nyelvtan**ok (**recursively enumerable grammar**s) korlátozás nélküli szabályokat használnak: az átíró szabályok mindkét oldala tetszőleges számú záró és nem záró szimbólumot tartalmazhat, mint például az $A\ B \rightarrow C$ szabály. Ezen nyelvtanok kifejezőereje megegyezik a Turing-gépével.

A **környezetérzékeny nyelvtan**ok (**context-sensitive grammar**s) csak olyan értelemben korlátozottak, hogy a jobb oldalnak legalább annyi szimbólumot kell tartalmaznia, mint a bal oldalnak. A környezetérzékeny (környezetfüggő) elnevezés abból a tényből ered, hogy egy $A\ S\ B \rightarrow A\ X\ B$ szabály azt mondja, hogy egy S átírható X-re egy megelőző A és egy követő B környezetében. A környezetérzékeny nyelvtanok leírhatnak például olyan nyelvet, mint az $a^n b^n c^n$ (a-k n hosszúságú sorozata, amelyet ugyanannyi b és c követ).

A **környezetfüggetlen nyelvtan**ok (**context-free grammar**s) (vagy **CFG**-k) esetében a bal oldal egyetlen nem záró szimbólumot tartalmaz. Így minden egyes szabály megengedi a nem záró szimbólum átírását a jobb oldalra *tetszőleges* környezetben. A CFG-k népszerűek a természetes és programozási nyelvek nyelvtanai körében, bár ma már széles körben elfogadott, hogy létezik legalább néhány olyan természetes nyelv, mely tartalmaz olyan szerkezeteket, melyek nem írhatók le környezetfüggetlen nyelvtannal (Pullum, 1991). A környezetfüggetlen nyelvtanok képesek reprezentálni $a^n b^n$-t, de nem $a^n b^n c^n$-t.

A **reguláris nyelvtan**ok (**regular grammar**s) a legkorlátozottabb osztály. Minden szabálynak van egyetlen nem zárója a bal oldalán, és egy záró szimbólum a jobb oldalon, amit opcionálisan követhet egy nem záró. A reguláris nyelvtanok a véges automatával ekvivalens erejűek. Gyengén alkalmasak programozási nyelvek számára, például nem képesek nyitó és bezáró zárójelek egyensúlyát leíró szerkezetek ábrázolására (az $a^n b^n$ nyelv egy változata). Ehhez legközelebb $a^* b^*$ reprezentálásával juthatnak, amely tetszőleges számú a sorozatát követő tetszőleges számú b.

A hierarchiában fentebb helyezkedő nyelvtanoknak nagyobb a kifejezőereje, de a kapcsolódó algoritmusok kevésbé hatékonyak. Az 1980-as évek közepéig a nyelvészek a környezetfüggetlen és környezetérzékeny nyelvtanokkal foglalkoztak. Azóta nagyobb hangsúlyt fektetnek reguláris nyelvtanokra, amelyeket az elektronikusan elérhető szövegek mega- és gigabájtjainak gyors, még a kevésbé teljes analízis árán is történő feldolgozási igénye hívott életre. Ahogy Fernando Pereira mondta: „Minél öregebb leszek, annál lentebb megyek a Chomsky-hierarchián.” Hogy lássuk, mit értett ezen, vesse össze (Pereira és Warren, 1980; Mohri, Pereira és Riley, 2002)

Az *NP*, a *VP* és az *S* kategóriák úgynevezett **nem záró szimbólum**ok (**nonterminal symbol**s). A nyelvtanok **átíró szabály**ok (**rewrite rule**s) segítségével definiálják a nem záró szimbólumokat. Az átíró szabályok leírására a Backus–Naur-formát (BNF) fogjuk átvenni, amelyet a B) függelékben írunk le az 1. szakasz - B1. Nyelvek definiálása Backus–Naur-Formában (BNF) részben. Ezen jelölés szerint egy

```
S → NP VP
```

szabály jelentése az, hogy egy *S* egy tetszőleges *NP* kategóriájú kifejezésből és az azt követő tetszőleges *VP* kategóriájú kifejezésből áll.

1.2. A kommunikációt alkotó lépések

Created by XMLmind XSL-FO Converter.

Egy tipikus kommunikációs epizód, amikor *S* beszélő *P* állítást szeretné átadni *H* hallgatónak *W* szavakkal, hét folyamatból áll:

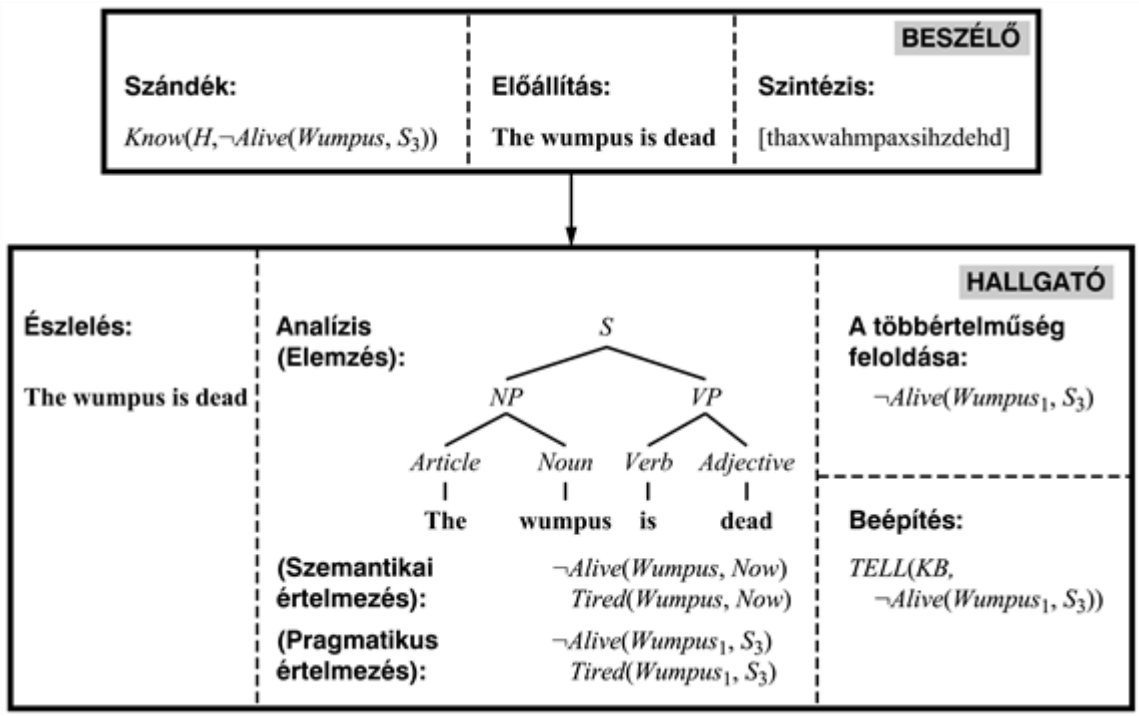
Szándék (intention). *S* beszélő valahogy eldönti, hogy van egy *P* propozíció, amit érdemes elmondani *H* hallgatónak. A példánkban a beszélőnek az a szándéka, hogy a hallgató tudomására hozza, hogy a wumpus már nem él.

Létrehozás (generation). A beszélő megtervezi, hogy *P* propozíciót hogyan alakítsa át olyan kijelentéssé, amely valószínűvé teszi, hogy a hallgató, fogadva a kijelentést a jelenlegi szituációban, kikövetkeztetheti *P* jelentést (vagy egy hozzá közelit). Tételezzük fel, hogy a beszélő képes a következő szavakat létrehozni: „The wumpus is dead" (A wumpus halott), és jelöljük ezt *W*-vel.

Szintézis (synthesis). A beszélő létrehozza *W* szavak fizikai realizációját, *W*′-t. Ezt teheti tintával papíron, rezgésekkel levegőben vagy valamilyen más médiumon. A 22.1. ábrán egy olyan ágenst mutatunk, amely a 6. szakasz - Beszédfelismerés részben definiált fonetikus ábécével leírt *W*′ hangfüzér szintézisét végzi: „[thaxwahmpaxsihzdehd]". A szavak egybefolytak, ami tipikus a gyorsan beszélt nyelv esetében.

Észlelés (perception). *H* észleli a *W*′ fizikai megvalósulást, mint W'_2-t, és W_2 szavakként dekódolja. Amikor a médium a beszéd, az észlelési lépést **beszédfelismerés**nek (**speech recognition**) nevezzük; amikor az írás, akkor **optikai karakter felismerés**nek (**optical character recognition**). Mindkettő elmozdult az elvont léttől a mindennapi előfordulás irányába a kilencvenes években, főként az asztali számítógépek jelentős mértékben növekvő teljesítményének köszönhetően.

22.1. ábra - A kommunikációban részt vevő hét folyamat, a „The wumpus is dead" mondat felhasználásával



Analízis (analysis). *H* kikövetkezteti, hogy W_2-nek P_1, …, P_n lehetséges jelentése van. Az analízist három fő részre osztjuk: szintaktikai elemzésre (parsing), szemantikai értelmezésre és pragmatikus értelmezésre. Az **elemzés (parsing)** egy bemeneti füzérhez tartozó **elemzési**, más szóval **levezetési fa (parse tree)** építésének a folyamata, ahogy a 22.1. ábrán látható. Az elemzési fa belső csomópontjai kifejezéseket reprezentálnak, míg a levelek szavakat jelképeznek. A **szemantikus értelmezés (semantic interpretation)** az a folyamat, amelynek során kinyerjük egy kijelentés jelentéstartalmát valamilyen reprezentációs nyelven. A 22.1. ábrán két lehetséges szemantikai értelmezést mutatunk be: azt, hogy a wumpus nem él, és azt, hogy fáradt (a *dead* egy köznyelvi jelentése). A több lehetséges értelmezéssel rendelkező kijelentéseket **többértelmű**nek (**ambiguous**) mondjuk. A **pragmatikus értelmezés (pragmatic interpretation)** figyelembe veszi azt a tényt, hogy ugyanazon szavaknak más jelentése lehet eltérő helyzetekben. Míg a szintaktikai értelmezés egy egy argumentummal, a füzérrel

Created by XMLmind XSL-FO Converter.

rendelkező függvény, addig a pragmatikus értelmezés a kijelentést és annak környezetét vagy szituációját figyelembe vevő függvény. A példában a pragmatikus elemzés két dolgot tesz: a *Now* konstans értéket kicseréli az S_3 konstanssal, ami a jelenlegi szituációt jelenti, illetve a *Wumpus*-t kicseréli $Wumpus_1$-re, ami arra az egyetlen Wumpusra utal, amelyről tudott, hogy a barlangban van. Általánosságban a pragmatikus elemzés a kijelentés végső értelmezéséhez sokkal többel járulhat hozzá; gondoljunk csak például arra, amikor a „A gyémántra nézek” elhangzik egy ékszerész, illetve egy baseballjátékos szájából.[2] A 22.7. alfejezetben látni fogjuk, hogy a pragmatikus elemzés segít értelmezi az „It is dead” kijelentést úgy, hogy a wumpus halott, ha egy olyan szituációban vagyunk, amikor a wumpus áll a figyelem középpontjában.

A **többértelműség feloldása** (**disambiguation**). *H* kikövetkezteti, hogy *S* P_i-t szándékozott közölni (ahol ideális esetben $P_i = P$). A beszélők többsége nem szándékosan többértelmű, de a legtöbb kijelentésnek több megengedett értelmezése van. A kommunikáció azért működik, mivel a hallgató elvégzi azt a munkát, hogy rájöjjön, hogy melyik az az értelmezés, amit a beszélő valószínűleg közölni akart. Vegyük észre, hogy ez az első alkalom, amikor a *valószínűleg* szót használtuk, és a többértelműség feloldása az első eljárás, ami erőteljesen valószínűségi következtetésen alapul. Az analízis lehetséges értelmezéseket állít elő; ha egynél több értelmezést talál, akkor a többértelműség feloldása választja ki a legjobbat.

Beépítés (**incorporation**). *H* eldönti, hogy elhiszi P_i-t (vagy nem). Egy teljesen naiv ágens mindent elhihet, amit hall, de egy kifinomultabb ágens úgy kezeli a szólásaktust, mint egy P_i-t alátámasztó tényt, és nem mint annak a megerősítését.

Mindezeket összerakva a 22.2. ábrán látható ágensprogramot kapjuk. Itt az ágens robotszolgaként működik, amit egy gazda irányíthat. A szolga minden lépésben megválaszolja a gazda kérdését, illetve végrehajtja a parancsát, és a szolga minden, a gazda által kijelentett állítást elhisz. Ezenkívül megállapítást tesz az aktuális szituációra (csak egyszer), ha nincs más tennivalója, és megtervezi saját akcióját, ha magára hagyják. Íme, egy tipikus párbeszéd:

ROBOTSZOLGA	GAZDA
Szellőt érzek.	Menj 1, 2-re!
Semmi sincs itt.	Menj északnak!
Szellőt és bűzt érzek, valamint csillogást látok.	Ragadd meg az aranyat!

22.2. ábra - Egy kommunikáló ágens, amely elfogad parancsokat, kérdéseket és állításokat. Az ágens leírhatja az aktuális állapotot, valamint végrehajthat „hagyományos”, nem szólásaktus cselekvést is, ha nincs mit mondania.

[2] A baseballpálya angol elnevezése *diamond*, mivel alakjával a gyémántra hasonlít. (*A ford.*)

Created by XMLmind XSL-FO Converter.

```
function NAIV-KOMMUNIKÁLÓ-ÁGENS(észlelés) returns cselekvés
  static: TB, egy tudásbázis
          állapot, a környezet jelenlegi állapota
          cselekvés, a legutóbbi cselekvés, kezdetben semmi

  állapot ← ÁLLAPOT-FRISSÍTÉS(állapot, cselekvés, észlelés)
  szavak ← BESZÉD-RÉSZ(észlelés)
  szemantika ← EGYÉRTELMŰSÍT(PRAGMATIKUS-ÉRTELMEZÉS(SZEMANTIKA(ELEMZÉS(szavak))))
  if szavak = Üres and cselekvés nem MOND then /* Írd le az állapotot! */
        return MOND(LEÍRÁS-KÉSZÍTÉS(állapot))
  else if TÍPUS[szemantika] = Utasítás then /* Hajtsd végre az utasítást! */
        return TARTALOM[szemantika]
  else if TÍPUS[szemantika] = Kérdés then /* Válaszold meg a kérdést! */
        válasz ← KÉRDEZ(TB, szemantika)
        return MOND(LEÍRÁS-KÉSZÍTÉS(válasz))
  else if TÍPUS[szemantika] = Állítás then /* Hidd el az állítást! */
        MOND(TB, TARTALOM[szemantika])
  /* Ha idáig eljutunk, akkor a „hagyományos" cselekvést hajtsd végre! */
  return ELSŐ(TERVEZÉS(TB, állapot))
```

2. formális nyelvtan az angol nyelv egy töredékére

Ebben a fejezetben definiálunk egy nyelvtant az angol nyelv egy kis töredékére, ami megfelelő a wumpus világról szóló kijelentésekhez. Ezt a nyelvet $\mathcal{E}_0$-nak nevezzük. A későbbi részek javítani fognak az $\mathcal{E}_0$-n, hogy valamelyest jobban hasonlítson a beszélt angolra. Igen valószínűtlen, hogy valaha is megszerkesztjük az angol nyelv teljes nyelvtanát, már csak azért sem, mivel nincs két ember, aki teljesen megegyezne abban, mit is tekint érvényes angolnak.

2.1. Az $\mathcal{E}_0$ szókincse

Először definiáljuk a **szókincs**et (**lexicon**), azaz a megengedett szavak listáját. A szavakat a szótárhasználók számára ismerős módon csoportosítjuk kategóriákba, avagy beszédrészekbe: főnevek, névmások és nevek dolgok elnevezésére, igék események jelölésére, jelzők a főnevek módosítására és határozószók az igék módosítására. Egyes olvasók számára kevéssé ismertek lehetnek a következő kategóriák: **névelő**k (**article**s) (például a *the*), **elöljárószavak** (**preposition**s) (például *in*) és **kötőszó**k (**conjunction**s) (például *and*). A 22.3. ábra mutat egy kis szókincset.

Minden egyes kategória …-tal végződik annak jelzésére, hogy vannak más szavak is a kategóriában. Azt is meg kell jegyeznünk azonban, hogy két különálló ok van a szavak hiányára. A főnevek, az igék, a jelzők és a névmások esetében elméletileg sem lehetséges az összes felsorolása. Nemcsak azért, mert minden osztálynak ezernyi vagy tízezernyi tagja lehet, hanem azért is, mivel folyamatosan hozzáadódnak újak, mint például az *MP3* vagy az *anime*. Ezt a négy kategóriát **nyitott osztály**oknak (**open class**es) nevezzük. A többi kategóriát (névmások, névelők, elöljáró- és kötőszók) **lezárt osztály**oknak (**closed class**es) hívjuk. Mindegyikben kevés számú szó van (néhánytól pár tucatig), amelyek elméletileg teljesen felsorolhatók. A lezárt osztályok évszázadok folyamán változnak, nem hónapok alatt. Például a „thee" és a „thou" a 17. században elterjedt névmások voltak, hanyatlóban voltak a 19. században, és manapság csak versekben és egyes régiók dialektusában lehet találkozni velük.

22.3. ábra - $\mathcal{E}_0$ szókincse

Created by XMLmind XSL-FO Converter.

Noun	→	**stench** | **breeze** | **glitter** | **nothing** | **wumpus** | **pit** | **pits** | **gold** | **east** | …
Verb	→	**is** | **see** | **smell** | **shoot** | **feel** | **stinks** | **go** | **grab** | **carry** | **kill** | **turn** | …
Adjective	→	**right** | **left** | **east** | **dead** | **back** | **smelly** | …
Adverb	→	**here** | **there** | **nearby** | **ahead** | **right** | **left** | **east** | **south** | **back** | …
Pronoun	→	**me** | **you** | **I** | **it** | …
Name	→	**John** | **Mary** | **Boston** | **Aristotle** | …
Article	→	**the** | **a** | **an** | …
Preposition	→	**to** | **in** | **on** | **near** | …
Conjunction	→	**and** | **or** | **but** | …
Digit	→	**0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

2.2. Az $\mathcal{E}_0$ nyelvtana

A következő lépés a szavak kifejezésekké történő kombinálása. Öt nem záró szimbólumot fogunk használni a különböző típusú kifejezések definiálására: mondat (*S*), főnévi kifejezés (*NP*), igei kifejezés (*VP*), elöljárói kifejezés (*PP*) és relatív klóz (*RelClause*).[3] A 22.4. ábra $\mathcal{E}_0$ számára mutat egy nyelvtant, minden egyes átíró szabályra példát adva. Az $\mathcal{E}_0$ jó angol mondatokat generált, mint amilyenek például a következők:

John is in the pit

The wumpus that stinks is in 2 2

Mary is in Boston and John stinks

Sajnálatos módon a nyelvtan **túlgenerál** (**overgenerates**) – előállít olyan mondatokat, amelyek nyelvtanilag helytelenek –, mint például a „Me go Boston” és az „I smell pit gold wumpus nothing east”. **Alul** is **generál** (**undergenerates**): sok olyan angol mondat van, amit elutasít, mint például a „I think the wumpus is smelly”. (Egy másik hátránya, hogy a nyelvtan nem nagybetűvel kezdi a mondatokat, és nem rak pontot a végükre. Ez azért van, mivel elsősorban beszédre tervezték, és nem írásra.)

22.4. ábra - Az $\mathcal{E}_0$ nyelvtana, minden egyes szabályra példa kifejezést adva

[3] Egy relatív klóz főnévi kifejezést követ és módosít. (A klóz a nyelvészetben egy alárendelt mellékmondat – *a ford.*) Egy relatív névmásból (mint a „who” és a „that”) és egy azt követő igei kifejezésből áll. (Egy másik típusú relatív klóz a 22.12. feladatban szerepel.) Példa egy relatív klózra a *that stinks* a „The wumpus *that stinks* is in 2 2” mondatban.

Created by XMLmind XSL-FO Converter.

S	→	*NP VP*	I + feel a breeze
	|	*S Conjunction S*	I feel a breeze + and + I smell a wumpus
NP	→	*Pronoun*	I
	|	*Name*	John
	|	*Noun*	pits
	|	*Article Noun*	the + wumpus
	|	*Digit Digit*	3 4
	|	*NP PP*	the wumpus + to the east
	|	*NP RelClause*	the wumpus + that is smelly
VP	→	*Verb*	stinks
	|	*VP NP*	feel + a breeze
	|	*VP Adjective*	is + smelly
	|	*VP PP*	turn + to the east
	|	*VP Adverb*	go + ahead
PP	→	*Preposition NP*	to + the east
RelClause	→	**that** *VP*	that + is smelly

3. Szintaktikai analízis (elemzés)

Már definiáltuk az **elemzés**t (**parsing**) mint az adott bemeneti füzérhez tartozó levezetési fa megtalálásának folyamatát. Azaz a `PARSE` függvény meghívása, mint például a

```
Parse(„the wumpus is dead”, ℰ₀, S)
```

olyan fát kell visszaadnia, amelynek gyökerében *S* áll, levelei a „the wumpus is dead” és belső csomópontjai az $\mathcal{E}_0$ nyelvtan nem záró szimbólumai. A 22.1. ábrán láthattunk egy ilyen fát. Folytonos szövegként a következőképpen írhatjuk:

```
[S: [NP: [Article: the] [Noun: wumpus]]
[VP: [Verb: is][Adjective: dead]]]
```

Fontos

Az elemzés tekinthető egy levezetési fa megkeresésének folyamataként. A keresési tér meghatározásának két szélsőséges (és sok közbülső) módja van. Az egyik szerint kiindulhatunk az *S* szimbólumból, és kereshetünk egy olyan fát, amely leveleiben tartalmazza a szavakat. Ez az úgynevezett **fentről lefelé elemzés** (**top-down parsing**) (mivel *S*-et a fa tetejére helyezzük). Másrészt kiindulhatunk a szavakból, és kereshetünk egy fát, ahol *S* a gyökércsomópont. Ez a **lentről felfelé elemzés** (**bottom-up parsing**).[4] A fentről lefelé történő elemzés pontosan definiálható a következő keresési problémaként:

- A **kezdeti állapot** (**initial state**) egy elemzési fa, amelynek *S* a gyökércsomópontja, és egy ismeretlen gyerekcsomópontja van: [*S*: ?]. Általánosságban a keresési térben minden állapot egy levezetési fa.
- Az **állapotátmenet-függvény** (**successor function**) kiválasztja azt a legszélső csomópontot bal oldalon a fában, amelynek ismeretlen gyereke van. Ezek után a nyelvtanban olyan szabályokat keres, amelyek ezt a csomópontot tartalmazzák gyökérelemként. Minden ilyen szabályra generál egy következő állapotot, ahol a ? szimbólumot felcseréli a szabály jobb oldalának megfelelő listával.

[4] Észrevehető, hogy a fentről lefelé és a lentről felfelé történő elemzés hasonló az előre-, illetve hátrafelé következtetéshez, amiket a 7. fejezetben írtunk le. Hamarosan látni fogjuk, hogy ez az analógia pontos.

Created by XMLmind XSL-FO Converter.

Például az $\mathcal{E}_0$ nyelvtanban két szabály van *S*-re, így az [*S*: ?] fát a következő két származtatottal cseréli le:

```
[S: [S: ?] [Conjunction: ?] [S: ?]]

[S: [NP: ?] [VP: ?]]
```

A másodiknak hét származtatottja lesz, minden *NP* átírási szabályhoz egy.

- A **célteszt** (**goal test**) ellenőrzi, hogy a levezetési fa levelei pontosan megfelelnek-e a bemeneti füzérnek, nincsenek-e ismeretlenek és lefedetlen bemenetek.

A fentről lefelé történő elemzés egyik nagy problémája az úgynevezett **bal-rekurzív szabály**ok (**left-recursive rule**s) kezelése, melyek $X \rightarrow X \ldots$ alakúak. Mélységi kereséssel egy ilyen szabály végtelen ciklusban *X*-et [*X*: *X* …]-ra cserélné. Szélességi keresés esetén sikeresen megtaláljuk az érvényes mondatok elemzéseit, de érvénytelen mondatok esetében egy végtelen keresési térben ragadnánk le.

A lentről felfelé történő elemzés keresésként történő formalizálása a következő:

- A **kezdeti állapot** (**initial state**) a bemeneti füzérben található szavak listája, mindegyiket egy olyan levezetési faként ábrázolva, melynek csak egy levele van, például: [**the**, **wumpus**, **is**, **dead**]. Általánosságban a keresés minden állapota levezetési fák egy listája.

- Az **állapotátmenet-függvény** (**successor function**) megvizsgál minden *i* pozíciót a fák listájában a nyelvtan szabályainak minden lehetséges jobb oldalán. Ha a lista *i* pozíciójában kezdődő részsorozata illeszkedik a jobb oldalra, akkor a részsorozatot lecseréli egy új fára, amelynek kategóriája a szabály bal oldala, és amelynek gyerekei a részsorozat. „Illeszkedés” alatt azt értjük, hogy a csomópont kategóriája megegyezik a jobb oldal elemének kategóriájával. Például az *Article* → **the** szabály illeszkedik a [**the**, **wumpus**, **is**, **dead**] első csomópontjából álló részsorozatra, így a következő állapot az [[*Article*: **the**], **wumpus**, **is**, **dead**] lenne.

- A **célteszt** (**goal test**) egy olyan állapotot keres, ahol egyetlen fa van, melynek gyökere az *S*.

A 22.5. ábra bemutat egy példát a lentről felfelé történő elemzésre.

22.5. ábra - A „the wumpus is dead” lentről felfelé történő elemzési lépései. A szavakat tartalmazó csomópontok listájával kezdünk. Ezek után egy szabály jobb oldalára illeszkedő részsorozatokat felcserélünk egy új csomóponttal, amelynek gyökere a szabály bal oldala. Például a harmadik sorban az *Article* és *Noun* csomópontokat lecseréljük egy *NP* csomópontra, amelynek e két csomópont lesz a gyereke. A fentről lefelé elemzés hasonló lépéseket eredményezne, de épp a fordított irányban.

lépés	*csomópontok listája*	*részsorozat*	*szabály*
KEZD	**the wumpus is dead**	**the**	Article → **the**
2	*Article* **wumpus is dead**	**wumpus**	Noun → **wumpus**
3	*Article Noun* **is dead**	*Article Noun*	*NP* → *Article Noun*
4	*NP* **is dead**	**is**	*Verb* → **is**
5	*NP Verb* **dead**	**dead**	*Adjective* → **dead**
6	*NP Verb Adjective*	*Verb*	*VP* → *Verb*
7	*NP VP Adjective*	*VP Adjective*	*VP* → *VP Adjective*
8	*NP VP*	*NP VP*	*S* → *NP VP*
CÉL	*S*		

A lentről felfelé és a fentről lefelé történő elemzés is lehet kevéssé hatékony azon módok számossága miatt, ahogy különböző kifejezésekhez többféle elemzés rendelhető. Mindkettő elpazarolhatja az időt a keresési tér irreleváns részeinek vizsgálatával. A fentről lefelé keresés generálhat olyan közbülső csomópontokat, amelyek sosem zárhatók le szavakkal, és a lentről felfelé keresés generálhat olyan részleges elemzéseket a szavakhoz, amelyek nem fordulhatnak elő egy *S*-ben sem.

Created by XMLmind XSL-FO Converter.

Még ha lenne is egy tökéletes heurisztikánk, amely lehetővé tenné az irreleváns kitérők nélküli keresést, ezek az algoritmusok akkor sem lennének hatékonyak, mivel bizonyos mondatoknak *exponenciálisan sok* levezetési fája van. A következő alfejezet megmutatja, hogy mit kezdhetünk ezzel a problémával.

3.1. Hatékony elemzés

Vizsgáljuk meg a következő két mondatot:

Have the students in section 2 of Computer Science 101 take the exam.

Have the students in section 2 of Computer Science 101 taken the exam?

Bár az első 10 szavuk közös, teljesen különböző levezetésük van, mivel az első egy felszólítás, a második egy kérdés. Egy balról jobbra elemző algoritmusnak tippelnie kellene, hogy az első szó egy felszólítás vagy egy kérdés része-e, és nem tudná megmondani, hogy a tipp helyes-e egészen a tizenegyedik szóig: *take* vagy *taken*. Ha az algoritmus rosszul tippelt, egészen az első szóig kellene visszalépnie. Az ilyen típusú visszalépés elkerülhetetlen, de ha azt szeretnénk, hogy az algoritmusunk hatékony legyen, akkor el kell kerülnie a „the students in section 2 of Computer Science 101” *NP*-ként történő újraelemzését minden alkalommal, amikor visszalép.

Fontos

Ebben a részben egy olyan algoritmust alakítunk ki, amely ezt a hatékonysági problémát kezelni tudja. Az alapötlet a **dinamikus programozás** (**dynamic programming**) egy példája: *minden alkalommal, amikor egy részfüzért elemzünk, tárold az eredményt, így később majd nem kell újraelemeznünk.* Például ha egyszer rájöttünk, hogy a „the students in section 2 of Computer Science 101” egy *NP*, ezt eltárolhatjuk egy **diagram**nak (**chart**) nevezett adatstruktúrában. Az így működő algoritmusokat **diagramelemző**knek (**chart parser**s) nevezik. Mivel környezetfüggetlen nyelvtanokkal foglalkozunk, a keresési tér egy ágának kontextusában talált tetszőleges kifejezés éppúgy szerepelhet a keresési tér bármilyen más ágában is.

Egy *n* szóból álló mondat diagramja *n* + 1 **csomópont**ból (**vertex**) és számos, ezeket összekötő **él**ből (**edges**) áll. A 22.6. ábrán láthatunk egy diagramot hat csomóponttal (körök) és három éllel (vonalak). Például a

```
[0, 5, S → NP VP •]
```

címkéjű él azt jelenti, hogy egy *NP*-t egy *VP* követ, és együtt alkotnak egy *S*-et, mely a 0-tól 5-ig terjedő karaktersorozatot fedi le. Egy élben található • jel elválasztja a már megtaláltat a még keresettől.[5] A • jellel záródó éleket **teljes él**eknek (**complete edge**s) nevezik. A

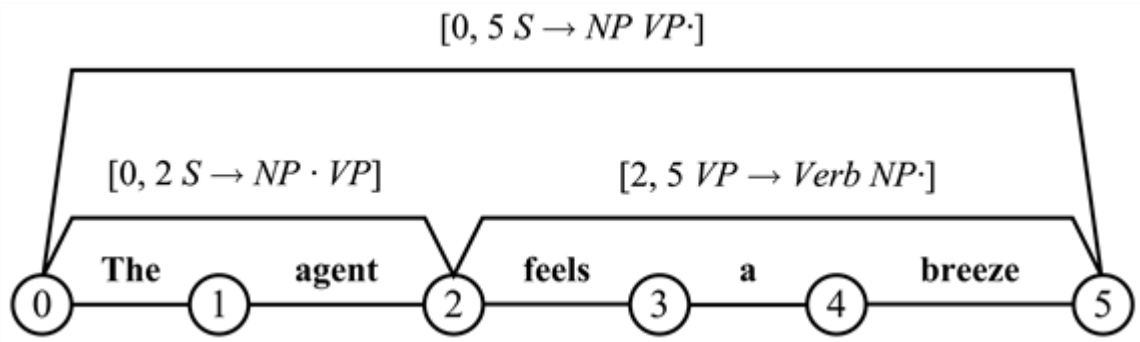
```
[0, 2, S → NP • VP]
```

él szerint egy *NP* lefedi a 0-tól 2-ig terjedő karaktersorozatot (az első két szó), és ha tudunk találni egy ezt követő *VP*-t, akkor lenne egy *S*-ünk. Az ilyen élek, ahol a pont a végük előtt van, úgynevezett befejezetlen (nem teljes) élek, és azt mondjuk, hogy az él egy *VP*-t vár.

22.6. ábra - A „The agent feels a breeze” mondat diagramjának egy részlete. Mind a hat csomópont látható, de csak három él szerepel azok közül, melyek teljes elemzést eredményeznének.

[5] A • miatt nevezik az éleket néha **pontozott szabály**oknak (**dotted rule**s).

Created by XMLmind XSL-FO Converter.



A 22.7. ábra a diagramelemző algoritmust szemlélteti. A lényege a fentről lefelé és a lentről felfelé legjobb tulajdonságainak ötvözése. A JÓSLÓ eljárás fentről lefelé működik: olyan bejegyzéseket helyez el a diagramban, amelyek megmondják, hogy milyen szimbólumok milyen helyszínen elvártak. A SZKENNER egy lentről felfelé eljárás, amely a szavakból indul ki, de egy szót csak egy meglevő diagrambejegyzés kiegészítésére használ fel. Hasonlóképpen a KITERJESZTŐ a komponenseket lentről felfelé építi, de csak egy meglevő diagrambejegyzést egészít ki.

Egy trükköt alkalmaztunk a teljes algoritmus elindításához: hozzáadtunk egy [0, 0, $S' \rightarrow \bullet S$] élet a diagramhoz, ahol S a nyelvtan kezdeti szimbóluma, S' pedig egy általunk most kitalált új szimbólum. Az ÉL-HOZZÁAD meghívása azt eredményezi, hogy a JÓSLÓ éleket ad az olyan szabályokhoz, melyek S-t eredményezhetnek, azaz [$S \rightarrow NP\ VP$]. Ezek után megvizsgáljuk ezen szabály első alkotóelemét, *NP*-t, és mindenféle módon olyan szabályokat adunk hozzá, melyek *NP*-t eredményeznek. Végső soron a JÓSLÓ fentről lefelé módon minden lehetséges élt hozzáad, ami felhasználható a végső S megalkotásában.

22.7. ábra - A diagramelemző algoritmus. *S* a kezdő szimbólum és *S′* egy új nem záró szimbólum, *diagram*[*j*] pedig azon élek listája, melyek a *j* csomópontban végződnek. A görög ábécé betűi nulla vagy több szimbólumból álló füzérekre illeszkednek.

Created by XMLmind XSL-FO Converter.

```
function DIAGRAMELEMZŐ(szavak, nyelvtan) returns diagram

   diagram ← a [0… HOSSZ(szavak)] üres listák elrendezése
   ÉL-HOZZÁAD([0, 0, S′ → • S])
   for i ← from 0 to HOSSZ(szavak) do
       SZKENNER(i, szavak[i])
   return diagram

procedure ÉL-HOZZÁAD(él)
   /* Adj hozzá egy élt a diagramhoz, és vizsgáld meg, hogy kiterjeszt, vagy jósol-e más élt! */
   if él nincs a diagram[VÉGE(él)]-ben then
       hozzáfűz él a diagram[VÉGE(él)]-hez
       if él-nek nincs része a pont után then KITERJESZTŐ(él)
       else JÓSLÓ(él)

procedure SZKENNER(j, szó)
   /* Minden egyes élt terjessz ki, amely a megadott kategóriájú szót várja! */
   for each [i, j, A → α • B β ] in diagram[j] do
       if szó B kategórájából való then
           ÉL-HOZZÁAD([i, j + 1, A → α B • β ])

procedure JÓSLÓ([i, j, A → α • B β ])
   /* Minden olyan B-re vonatkozó szabályt adj a diagramhoz, ami segíthet ezen élt kiterjeszteni! */
   for each (B → γ) in ÁTÍR(B, nyelvtan) do
       ÉL-HOZZÁAD([j, j, B → • γ])

procedure KITERJESZTŐ([j, k, B → γ • ])
   /* Nézd meg, mely élek terjeszthetők ki ezzel az éllel! */
   e_B ← ezen eljárás bemeneteként kapott él
   for each [i, j, A → α • B′ β ] in diagram[j] do
       if B = B′ then
           ÉL-HOZZÁAD([i, k, A → α e_B • β])
```

Amikor *S′* jóslója készen van, belépünk egy ciklusba, amely meghívja a SZKENNER-t a mondat minden egyes szavára. Ha a *j* pozícióban álló szó egy *B* kategória tagja, amit valamelyik él keres a *j* pozícióban, akkor kiterjesztjük azt az élet megjelölve a szót, mint *B* egy példányát. Vegyük észre, hogy a SZKENNER minden egyes meghívása végződhet a JÓSLÓ és a KITERJESZTŐ rekurzív meghívásával, ily módon ötvözve a fentről lefelé és a lentről felfelé feldolgozást.

A másik lentről felfelé komponens, a KITERJESZTŐ[6] vesz egy teljes élt, amelynek bal oldalán *B* van, és felhasználja a diagramban levő bármely nem teljes szabály kiterjesztésére, amely ott végződik, ahol a teljes él kezdődik, ha a nem teljes szabály egy *B*-re vár.

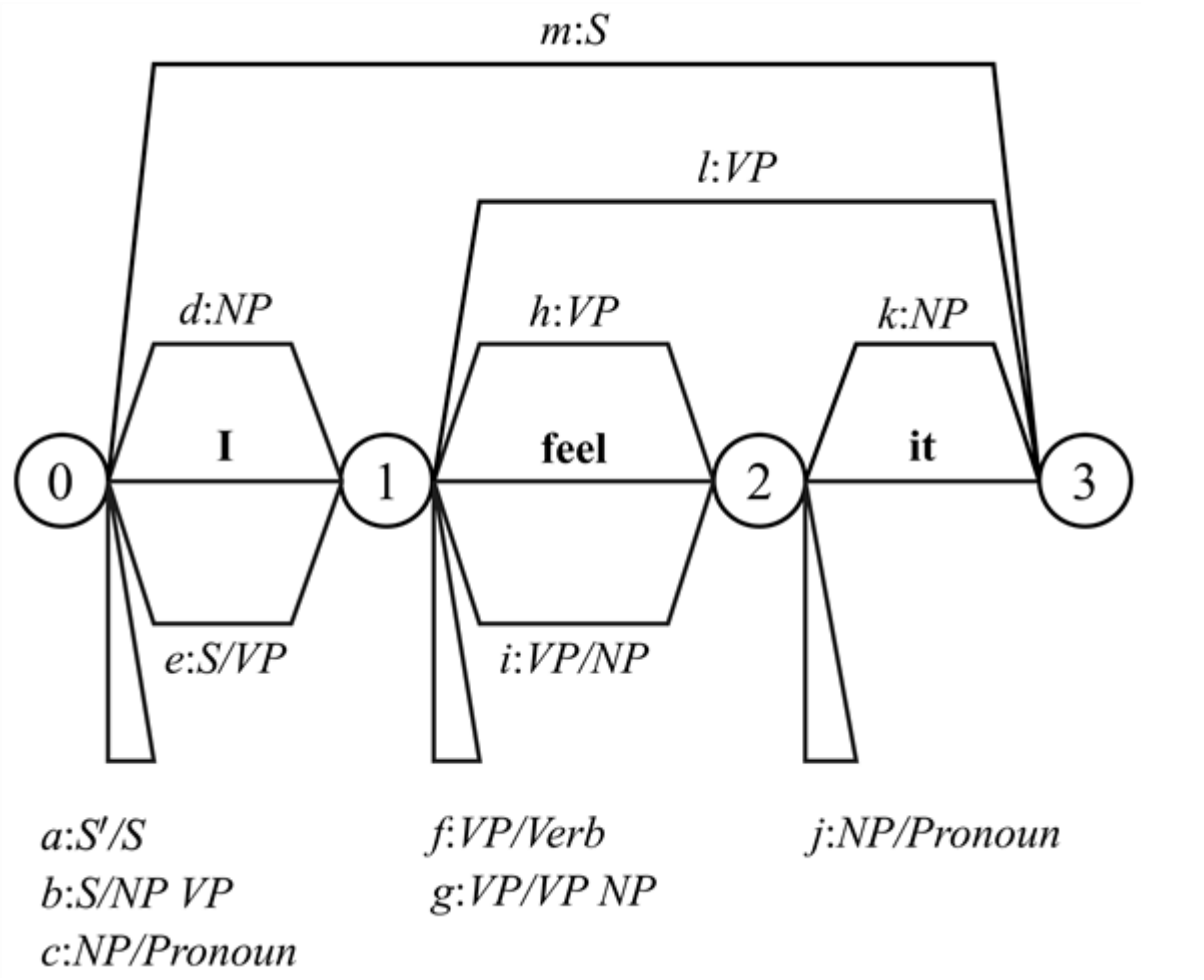
A 22.8. és 22.9. ábrák mutatják az „I feel it” mondat (amely a „Do you feel a breeze?” kérdésre adott válasz) diagramját és elemzési lépéseit. Tizenhárom él (a–m jelekkel ellátva) szerepel a diagramon, melyek közül öt teljes (a diagram verticei csomópontjai felett) és nyolc befejezetlen (alattuk). Vegyük észre a JÓSLÓ, SZKENNER és KITERJESZTŐ akciók ciklusát. Például a Jósló felhasználja azt a tényt, hogy az „a” él egy *S*-t vár ahhoz, hogy megelőlegezze egy *NP* („b” él) és egy *Pronoun* („c” él) jóslatát. Ezek után a SZKENNER felismeri, hogy van egy *Pronoun* a megfelelő helyen („d” él), és a KITERJESZTŐ kombinálja a „b” nem teljes élt a „d” teljes éllel, így előállítva egy új élt, „e”-t.

22.8. ábra - Az „$_0$ I $_1$ feel $_2$ it $_3$” mondat diagramelemzése. Az *m*:*S* jelölés azt jelenti, hogy az *m* élnek egy *S* áll a jobb oldalán, míg az *f*:*VP/Verb* azt, hogy az *f* élnek *VP* van a bal

[6] A KITERJESZTŐ eljárásunkat tradicionálisan TELJESSÉ-TEVŐ-nek hívták. Ez a név félrevezető, mivel az eljárás nem fejez be éleket: bemenetként vesz egy teljes élt, és kiterjeszt nem teljes éleket.

Created by XMLmind XSL-FO Converter.

oldalán, de egy *Verb*-et vár. Öt teljes él van a csomópontok felett, és nyolc hiányos alattuk.



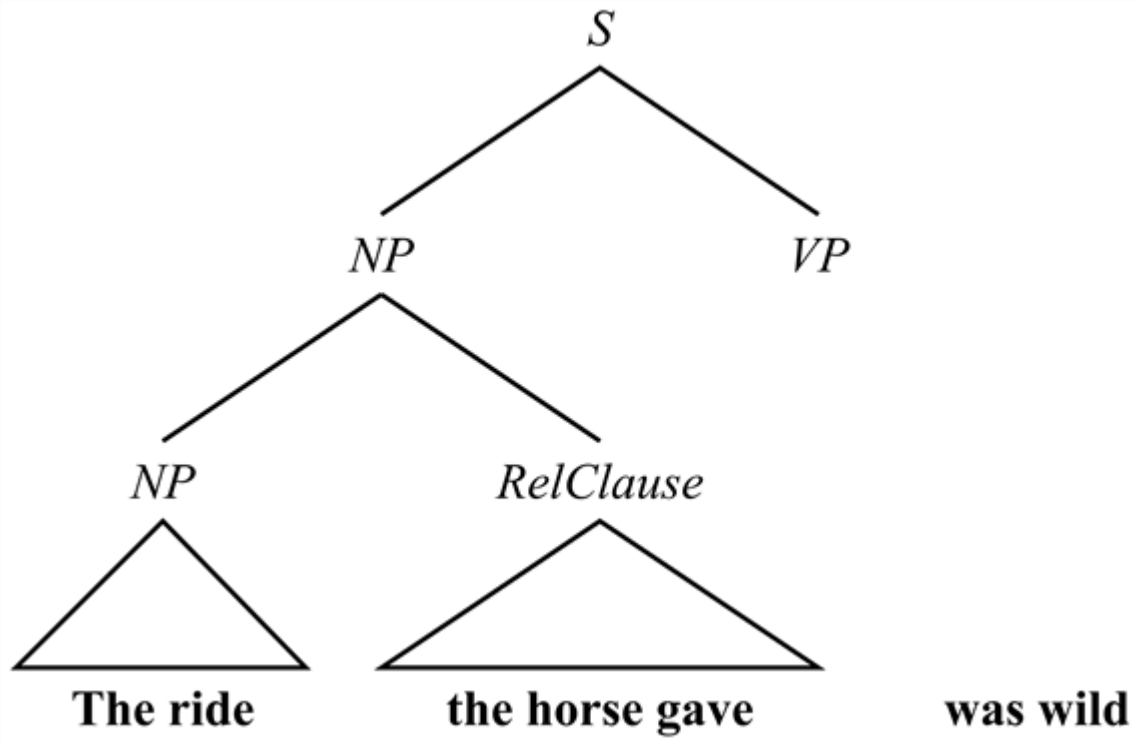
A diagramelemző algoritmus elkerüli az élek nagy halmazának építését, amit az egyszerű lentről felfelé eljárás végigvizsgált volna. Vegyük a „The ride the horse gave was wild” mondatot. Egy lentről felfelé elemzés megjelölné a „ride the horse”-t mint egy *VP*-t, majd elvetné az elemzési fát, amint rájönne, hogy nem illeszkedik egy nagyobb *S*-be. Azonban az ε_0 nem engedi meg, hogy egy *VP* a „the”-t kövesse, így a diagramelemző algoritmus sosem jósol *VP*-t ezen a ponton, így elkerüli ott a *VP* szerkezet időpocsékoló építését. Azokat az algoritmusokat, melyek balról jobbra működnek, és elkerülik ezen lehetetlen szerkezetek felépítését, **balsarok-elemző**knek (**left-corner parser**s) nevezzük, mivel egy olyan elemzési fát építenek fel, amely a nyelvtan kezdő szimbólumával indul, és a mondat legszélén balra (a bal sarokban) álló szó felé terjeszkednek. Egy élt csak akkor adnak a diagramhoz, ha ezen elemzési fa kiterjesztését szolgálhatja (példaként lásd 22.10. ábra).

22.9. ábra - Az „$_0$ I $_1$ feel $_2$ it $_3$.” elemzési lépései. Az a–m élek mindegyikére megmutatjuk azt az eljárást, amit az él más, a diagramon már szereplő élekből történő származtatására használtunk. Bizonyos éleket a rövidség érdekében elhagytunk.

Created by XMLmind XSL-FO Converter.

Él	Eljárás	Származtatás
a	INICIALIZÁLÁS	[0, 0, *S'* → • *S*]
b	JÓSLÓ(a)	[0, 0, *S* → • *NP VP*]
c	JÓSLÓ(b)	[0, 0, *NP* → • *Pronoun*]
d	SZKENNER(c)	[0, 1, *NP* → *Pronoun* •]
e	KITERJESZTŐ(b, d)	[0, 1, *S* → *NP* • *VP*]
f	JÓSLÓ(e)	[1, 1, *VP* → • *Verb*]
g	JÓSLÓ(e)	[1, 1, *VP* → • *VP NP*]
h	SZKENNER(f)	[1, 2, *VP* → *Verb* •]
i	KITERJESZTŐ(g, h)	[1, 2, *VP* → *VP* • *NP*]
j	JÓSLÓ(g)	[2, 2, *NP* → • *Pronoun*]
k	SZKENNER(h)	[2, 3, *NP* → *Pronoun* •]
l	KITERJESZTŐ(i, k)	[1, 3, *VP* → *VP NP* •]
m	KITERJESZTŐ(e, l)	[0, 3, *S* → *NP VP* •]

22.10. ábra - Egy balsarok-elemző algoritmussal elkerülhető a „ride” szóval kezdődő *VP* jóslása, de jósolni fog egy „was” szóval kezdődő *VP*-t, mivel a nyelvtan egy *NP*-t követő *VP*-t vár. A „the horse gave” feletti háromszög azt jelenti, hogy a szavaknak van egy *RelCause* elemzésük, amihez azonban közvetlenül további részek kapcsolódnak, amik nem láthatók.



A diagramelemző csak polinomiális idő- és tárigényű. $O(kn^2)$ helyet igényel az élek tárolására, ahol n a mondatban levő szavak száma és k egy, a nyelvtantól függő konstans. Amikor már nem tud több élt építeni, megáll, így tudjuk, hogy az algoritmus befejeződött (még akkor is, ha vannak bal-rekurzív szabályok). Valójában $O(n^3)$ időt igényel legrosszabb esetben, ami a legjobb, amit elérhetünk környezetfüggetlen nyelvtanokra. A `DIAGRAMELEMZŐ` szűk keresztmetszete a `KITERJESZTŐ`, amelynek meg kell próbálnia kiterjeszteni a j-ben végződő $O(n)$ nem teljes él mindegyikét az ugyanott kezdődő $O(n)$ teljes él mindegyikével, j lehetséges $n + 1$ különböző értékére. Ezeket összeszorozva kapjuk $O(n^3)$-at. Ez egyfajta paradoxont ad a számunkra: hogyan tud egy $O(n^3)$ algoritmus visszaadni egy választ, amely exponenciális számosságú elemzési fát tartalmazhat? Vegyünk egy példát: a

Created by XMLmind XSL-FO Converter.

„Fall leaves fall and spring leaves spring”

mondat többértelmű, mivel minden egyes szó (az „and” kivételével) lehet főnév és ige is, valamint a „fall” és a „spring” lehet melléknév is. Ennek a mondatnak mindösszesen négy elemzése van:[7]

```
[S: [S: [NP: Fall leaves] fall] and [S: [NP: spring leaves] spring]

[S: [S: [NP: Fall leaves] fall] and [S: spring [VP: leaves spring]]

[S: [S: Fall [VP: Fall leaves] fall] and [S: [NP: spring leaves] spring]

[S: [S: Fall [VP: Fall leaves] fall] and [S: spring [VP: leaves spring]]
```

Ha n többértelmű összekapcsolt részmondatunk lett volna, akkor 2^n módon választhatnánk elemzést a részmondatokra.[8] Hogyan kerüli el a diagramelemző az exponenciális feldolgozási időt? Két választ is adhatunk a kérdésre. Először is, a `DIAGRAMELEMZŐ` algoritmus maga valójában egy *felismerő,* nem egy elemző. Ha van egy $[0, n, S \rightarrow \alpha \bullet]$ alakú teljes él a diagramon, akkor felismertünk egy S-t. Az elemzési fa előállítását ebből az élből nem tekintjük a `DIAGRAMELEMZŐ` feladatának, de elvégezhető. Vegyük észre, hogy a `KIBŐVÍTŐ` utolsó sorában α-t élek e_B listájaként építjük fel, nemcsak kategória nevek listájaként. Így egy él elemzési fává történő átalakításához egyszerűen az összetevő éleket kell rekurzív módon végignézni, minden egyes $[i, j, X \rightarrow \alpha \bullet]$ élt egy $[X : \alpha]$ fává átalakítva. Ez egyenes/közvetlen módszer, de csak *egy* elemzési fát ad.

A második válasz szerint ha minden lehetséges elemzést szeretnénk, akkor mélyebbre kell ásnunk a diagramban. Miközben az $[i, j, X \rightarrow \alpha \bullet]$ élt egy $[X : \alpha]$ fává átalakítjuk, azt is megvizsgáljuk, hogy van-e másik, $[i, j, X \rightarrow \beta \bullet]$ alakú él. Amennyiben van, ezek az élek további elemzési fákat generálnak. Így kapjuk azt a választási lehetőséget, hogy mit is kezdjünk velük. Felsorolhatjuk az összes lehetőséget, ami azt jelenti, hogy a paradoxont feloldanánk, és exponenciálisan sok időre lenne szükségünk az összes elemzés felsorolásához. Vagy továbbvihetjük a rejtélyt egy kicsit és az elemzéseket egy **tömörített erdő** (**packed forest**) nevű struktúrával reprezentálhatjuk, amely a következőképpen néz ki:

$$\left[S : \left[S : \begin{Bmatrix}[NP : Fall\ leaves][VP : fall] \\ [NP : Fall][VP : leaves\ fall]\end{Bmatrix}\right] \text{ és } \left[S : \begin{Bmatrix}[NP : spring\ leaves[VP : spring]] \\ [NP : spring[VP : leaves\ spring]]\end{Bmatrix}\right]\right]$$

A lényeg az, hogy minden csomópont lehet egy hagyományos elemzési fa csomópont, illetve csomópontok egy halmaza is. Ez lehetővé teszi számunkra, hogy exponenciális számú elemzést reprezentáljunk polinomiális időben és tárhelyen. Természetesen $n = 2$ esetén nincs sok különbség 2^n és $2n$ között, de nagy n-ekre egy ilyen reprezentáció jelentős megtakarítást eredményez. Sajnálatos módon ez az egyszerű tömörített erdő megközelítés nem kezeli az összes $O(n!)$ lehetőséget a kapcsolódások összerendelésére. Maxwell és Kaplan (1995) megmutatja, hogy egy, az igazság-karbantartó rendszerek alapelveire épülő, összetettebb reprezentáció ezeket a fákat még jobban összetömörítheti.

4. Kiterjesztett nyelvtanok

A 22.2. alfejezetben láttuk, hogy az $\mathcal{E}_0$ egyszerű nyelvtana előállítja az angol nyelv „I smell a stench” és sok más mondatát. Sajnos azonban sok nem megengedett mondatot is előállít, mint például a „Me smell a stench”. Ezen probléma elkerülése érdekében a nyelvtanunknak tudnia kellene, hogy a „me” nem érvényes *NP* akkor, amikor a mondat alanya. A nyelvészek azt mondják, hogy az „I” névmás alanyesetű, míg a „me” tárgyesetű.[9] Ha figyelembe vesszük az eseteket, akkor rájövünk, hogy az $\mathcal{E}_0$ nem környezetfüggetlen: nem igaz az, hogy minden *NP* minden mással egyenértékű, függetlenül a környezetétől. A probléma megoldására bevezethetünk olyan új kategóriákat, mint az NP_S és NP_O a tárgyas és alanyi főnévi kifejezésekre. Ezenkívül ketté kellene bontanunk a *Pronoun* kategóriát a $Pronoun_S$-re (amelyik tartalmazza az „I”-t) és $Pronoun_O$-ra (amelyik tartalmazza a „me”-t). A 22.11. ábra felső része megmutatja az esetegyeztetés teljes BNF nyelvtanát; az így kapott nyelvet $\mathcal{E}_1$-nek

[7] Az [*S*: Fall [*VP*: leaves fall]] elemzés megegyezik az „Autumn abandons autumn” mondattal.

[8] Emellett a komponensek összekapcsolódásának $O(n!)$ *számú* értelmezése lenne – például (X és (Y és Z)), illetve ((X és Y) és Z). De ez már egy másik történet, amelyet Church és Patil (1982) igen jól mesél el.

[9] Az angolban az alanyesetre a „subjective case” mellett alkalmanként a „nominative case” kifejezést is használják. Hasonlóképpen a tárgyesetre az „objective case” mellett az „accusative case” is használatos esetenként. Sok nyelvnek van részes esete is az indirekt tárgy pozícióban álló szavakra.

Created by XMLmind XSL-FO Converter.

nevezzük. Vegyük észre, hogy az összes *NP* szabályt meg kell kettőznünk, egyszer az NP_S, másszor az NP_O esetre.

22.11. ábra - Felül: Az $\mathcal{E}_1$ egy BNF nyelvtana, amely kezeli az alanyi és a tárgyas esetet a főnévi kifejezésekben, és ezért annyira nem is generál túl. Az $\mathcal{E}_1$ nyelvtanával megegyező részeket elhagytuk. Lent: Az $\mathcal{E}_1$ egy definit klóz nyelvtana (DCG).

$S \rightarrow NP_S\ VP \mid \ldots$
$NP_S \rightarrow Pronoun_S \mid Name \mid Noun \ldots$
$NP_O \rightarrow Pronoun_O \mid Name \mid Noun \ldots$
$VP \rightarrow VP\ NP_O \mid \ldots$
$PP \rightarrow Preposition\ NP_O$
$Pronoun_S \rightarrow$ **I | you | he | she | it | …**
$Pronoun_O \rightarrow$ **me | you | him | her | it | …**

$S \rightarrow NP(Subjective)\ VP \mid \ldots$
$NP(eset) \rightarrow Pronoun(eset) \mid Name \mid Noun \ldots$
$VP \rightarrow VP\ NP(Objective) \mid \ldots$
$PP \rightarrow Preposition\ NP(Objective)$
$Pronoun(Subjective) \rightarrow$ **I | you | he | she | it | …**

$Pronoun(Objective) \rightarrow$ **me | you | him | her | it | …**

Sajnos az $\mathcal{E}_1$ még mindig túlgenerál. Az angol és sok más nyelv megköveteli az alany és a mondat fő igéjének **egyeztetés**ét (**agreement**). Például ha „I” az alany, akkor az „I smell” nyelvtanilag helyes, míg az „I smells” nem. Ha „it” az alany, akkor fordítva igaz. Az angolban az egyeztetésből származó különbségek minimálisak: a legtöbb igének egy alakja van egyes szám harmadik személyű alanyokra (he, she vagy it), és egy másik alakja az egyes és többes szám és a személyek minden más kombinációjára. Egy kivétel van: az „I am / you are / he is” három formával rendelkezik. Ha kombináljuk ezt a három különbségtételt az NP_S és NP_O különbségtétellel, akkor *NP* hat alakját kapjuk. Ahogy további különbözőségeket fedezünk fel, végül egy exponenciális sokaságot kapunk.

Az alternatív megközelítés a nyelvtan létező szabályainak **kiterjesztés**e (**augmenting**) új szabályok bevezetése helyett. Először megmutatjuk egy példán keresztül, hogy hogyan szeretnénk, hogy a kiterjesztett szabályok kinézzenek (a 22.11. ábra alsó fele), azután formálisan megadjuk, hogyan kell ilyen szabályokat értelmezni. A kiterjesztett szabályok nem záró kategóriák esetében megengedik *paraméterek* alkalmazását. A 22.11. ábra megmutatja, hogyan írhatjuk le az $\mathcal{E}_1$ nyelvtant kiterjesztett szabályokkal. Az *NP* és *Pronoun* kategóriák rendelkeznek egy olyan paraméterrel, amely leírja az esetüket. (A főnevek nem rendelkeznek esettel az angolban, bár sok más nyelvben igen.) Az *S*-re vonatkozó szabályban az *NP*-nek alanyi esetűnek kell lennie, míg a *VP* és *PP* szabályokban tárgyasnak. Az *NP* szabály egy változót fogad paraméterként, az *eset*-et. A szándék az, hogy az *NP* tetszőleges esetű lehet, de ha átírjuk *Pronoun*-ra, akkor ugyanolyan esetűnek kell lennie. Egy változó ilyen használata – elkerülve a döntést, amikor a különbségtétel nem fontos – visszatartja a szabályhalmaz méretének a tulajdonságok számával arányosan exponenciális növekedését.

A kiterjesztés ezen formalizmusát **definit klóz nyelvtan**nak (**definite clause grammar**) vagy **DCG**-nek hívjuk, mivel minden nyelvtani szabály értelmezhető a Horn-logikában szereplő definit klózként.[10] Elsőként azt mutatjuk meg, hogy egy normális, kiterjesztés nélküli szabály hogyan értelmezhető definit klózként. Minden kategóriajelet karaktersorozatra vonatkozó predikátumnak tekintünk, így az *NP*(s) igaz, ha az *s* karaktersorozat egy *NP*-t formál. Az

$S \rightarrow NP\ VP$

CFG-szabály a következő definit klóz rövidítése:

$NP(s_1) \wedge VP(s_2) \Rightarrow S(s_1 + s_2)$

[10] Idézze fel, hogy a definit klóz, amit ha implikációs formában írunk, pontosan egy atomot tartalmaz a következmény részében, és nulla vagy több atom konjunkcióját a feltétel részében. Két példa az $A \wedge B \wedge \Rightarrow C$, illetve önmagában a *C*.

Created by XMLmind XSL-FO Converter.

Fontos

Itt $s_1 + s_2$ két karaktersorozat összefűzését jelenti, így ez a szabály azt mondja, hogy ha van egy s_1 füzér, ami egy *NP* és egy s_2 füzér, ami egy *VP*, akkor az összekapcsolásukkal keletkező füzér egy *S*, ami pontosan megegyezik azzal, ahogy a CFG-szabályt korábban értelmeztük. Fontos észrevennünk, hogy a *DCG-k lehetővé teszik számunkra, hogy az elemzésről mint logikai következtetésről beszéljünk*. Ez lehetővé teszi, hogy nyelvek és karaktersorozatok felett sok különböző módon következtessünk. Például ez azt jelenti, hogy egy lentről felfelé elemzést előrefelé következtetéssel, egy fentről lefelé elemzést pedig hátrafelé következtetéssel végezhetünk el. Azt is látni fogjuk, hogy ez azt is jelenti, hogy ugyanazt a nyelvtant elemzésre és generálásra is használhatjuk.

A DCG-megközelítés igazi haszna az, hogy *kiterjeszthetjük* a kategóriaszimbólumokat a karakterfüzéreken kívül további argumentumokkal. Például a

```
NP(eset) → Pronoun(eset)
```

szabály a következő definit klóz rövidítése:

$$Pronoun(eset, s_1) \Rightarrow NP(eset, s_1)$$

Eszerint ha az s_1 füzér az *eset* változó által meghatározott esetű *Pronoun*, akkor s_1 egy ugyanolyan esetű *NP* is. Általánosságban egy kategóriaszimbólumot tetszőleges számú argumentummal kiterjeszthetünk, és az argumentumok egyben paraméterek, amelyek a szokásos Horn-klóz következtetés egyesítésének tárgyai.

Van ára is ennek a kényelemnek: a nyelvtan íróját ellátjuk a tételbizonyítók teljes erejével, így feladjuk a szintaktikai elemzők által garantált $O(n^3)$-at; a kiterjesztésekkel kiegészített elemzés lehet NP-teljes, vagy akár eldönthetetlen, a kiterjesztésektől függően.

Néhány további trükk szükséges ahhoz, hogy a DCG működjön; például szükségünk van egy módszerre, hogy záró szimbólumokat adhassunk meg, és kényelmes egy olyan módszert is alkalmaznunk, amely *nem* engedi az automatikus karaktersorozat-argumentum hozzáadását. Mindent összerakva, a definit klóz nyelvtant a következőképpen definiálhatjuk:

- Az $X \rightarrow Y\ Z\ \ldots$ jelölés fordítása $Y(s_1) \wedge Z(s_2) \wedge \ldots \Rightarrow X(s_1 + s_2 + \ldots)$
- Az $X \rightarrow Y \mid Z \mid \ldots$ jelölés fordítása $Y(s) \vee Z(s) \vee \ldots \Rightarrow X(s)$
- Mindkét megelőző szabályban bármelyik Y nem záró szimbólum egy vagy több argumentummal kiterjeszthető. Bármely argumentum lehet változó, konstans vagy argumentumok függvénye. A fordításban ezen argumentumok megelőzik a karakterfüzér argumentumot (például az *NP*(*eset*) fordítása *NP*(*eset*, s_1)).
- A {*P*(…)} jelölés szerepelhet egy szabály jobb oldalán, és változás nélkül átalakul *P*(…)-vé. Ez lehetővé teszi a nyelvtan írója számára, hogy egy *P*(…)-re vonatkozó tesztet anélkül elhelyezzen, hogy ahhoz automatikusan hozzáadódjon a karakterfüzér-argumentum.
- Az $X \rightarrow$ **word** jelölés fordítása *X*([*word*]).

Az alany–ige egyeztetése szintén kezelhető kiterjesztéssel, de ezt a 22.2. feladatra halasztjuk. Helyette egy nehezebb problémával foglalkozunk: az igék alkategóriákba osztásával.

4.1. Igék alkategóriákba osztása

Az $\mathcal{E}_1$ nyelv fejlettebb az $\mathcal{E}_0$-hoz képest, de még ez is túlgenerál. Az egyik probléma abban a módban rejlik, ahogy az igei kifejezéseket összefűzi. El akarjuk fogadni a „give me the gold” és a „go to 1,2” jellegű igés kifejezéseket. Ezek mindegyike létezik az $\mathcal{E}_1$-ben, de sajnos éppígy létezik a „go me the gold” és a „give to 1,2” is. Az $\mathcal{E}_2$ nyelv megszünteti ezeket a *VP*-ket úgy, hogy egyértelműen megadja, melyik igét melyik kifejezés követheti. Ezt az igék **alkategória** (**subcategorization**) listájának hívjuk. Az elképzelés az, hogy a *Verb* kategóriát alkategóriákra bontjuk – egy a tárggyal nem rendelkezők igék számára, egy az egyetlen tárggyal rendelkezőkre és így tovább.

Az ötlet megvalósításához minden egyes igéhez rendelünk egy **alkategória-listát** (**subcategorization list**), amely a **kiegészítő**it (**complement**s) tartalmazza. A kiegészítő olyan kötelező kifejezés, amely az igei

Created by XMLmind XSL-FO Converter.

kifejezésen belül követi az igét. Azaz a „give the gold to me”-ben az *NP* a „the gold” és a „to me” *PP* a „give” kiegészítője.[11] Ezt leírhatjuk így:

```
Verb([NP, PP]) → give | hand | …
```

Lehetséges, hogy egy igéhez több, különböző alkategóriára bontás létezik, csakúgy, mint ahogy egy szó számára is lehetséges, hogy több, különböző kategóriába tartozzon. Valójában a „give” rendelkezik az [*NP*, *NP*] alkategória-listával is, mint a „Give me the gold” mondatban. Ezt ugyanúgy kezelhetjük, mint minden más többértelműséget. A 22.12. ábra ad pár példát igékre és alkategória-listáikra (angolul röviden **subcat**).

22.12. ábra - Példák igékre az alkategória-listáikkal együtt

Ige	Alkategóriák	Példa igei kifejezés
give	[*NP*, *PP*] [*NP*, *NP*]	give the gold in 3 3 to me give me the gold
smell	[*NP*] [*Adjective*] [*PP*]	smell a wumpus smell awful smell like a wumpus
is	[*Adjective*] [*PP*] [*NP*]	is smelly is in 2 2 is a pit
died	[]	died
believe	[*S*]	believe the wumpus is dead

Három dolgot teszünk az igei alkategóriák nyelvtanba illesztéséhez. Az első lépés a *VP* kategória kiterjesztése egy alkategória-argumentummal, *VP*(*subcat*), amely felsorolja azokat a komponenseket, amikkel teljes *VP* formálható. Például a „give” teljes *VP*-vé tehető [*NP*, *PP*] hozzáadásával, „give the gold” teljessé tehető [*PP*] hozzáadásával, és a „give the gold to me” már teljes; alkategória-listája üres, []. Ez a következő szabályokat adja *VP*-re:

```
VP(subcat)   →   Verb(subcat)

             |   VP(subcat      +      [NP])
                 NP(Objective)

             |   VP(subcat   +   [Adjective])
                 Adjective

             |   VP(subcat + [PP]) PP
```

Az utolsó sor így olvasható: „Adott *subcat* alkategória-listával rendelkező *VP* formálható egy beágyazott *VP*-ből, amit egy *PP* követ, akkor, ha a beágyazott *VP* rendelkezik egy olyan alkategória-listával, ami a *subcat* lista elemeivel kezdődik és *PP*-vel folytatódik.” Például egy *VP*([])-t egy *VP*([*PP*]) és az azt követő *PP* alkot. Az első sor szerint egy *VP*, amely *subcat* alkategória-listával rendelkezik, formálható egy ugyanolyan alkategória-listával rendelkező *Verb* segítségével. Például a *VP*([*NP*]) létrehozható egy *Verb*([*NP*]) által. Egy példa ilyen igére a „grab”, tehát a „grab the gold” egy *VP*([]).

[11] Ez a *kiegészítő (complement)* egyik definíciója, de vannak más terminológiát használó szerzők is. Egyesek szerint az ige alanya is egy kiegészítő. Mások szerint csak az elöljárói kifejezés kiegészítő, és a főnévi kifejezést egy **érvelés**nek (**argument**) kell hívni.

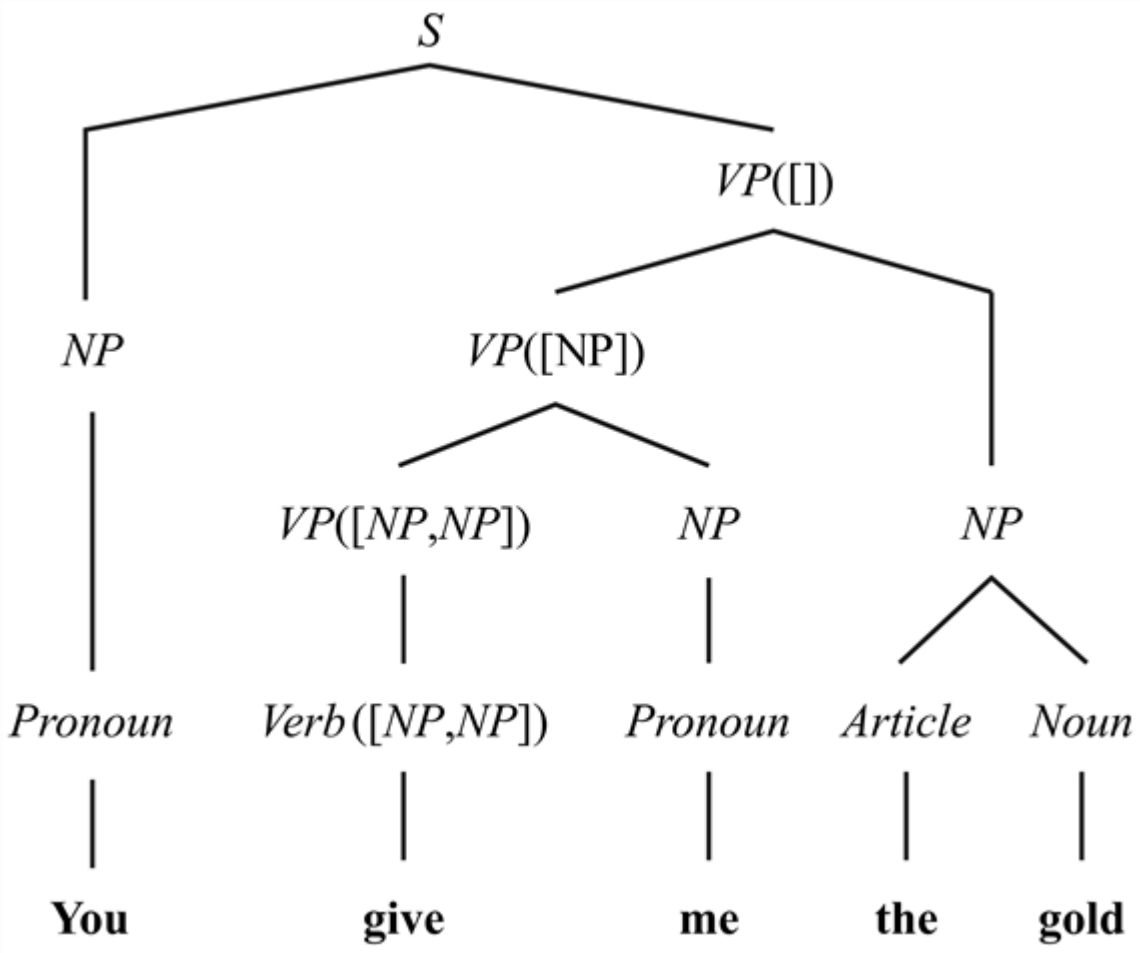
Created by XMLmind XSL-FO Converter.

A második lépés az *S*-re vonatkozó szabály megváltoztatása úgy, hogy szükségessé tegyen egy igei kifejezést, melynek összes kiegészítése megvan, és így [] alkategória-listája van. Ez azt jelenti, hogy az „I grab the gold” érvényes mondat, de a „You give” nem az. Az új szabály,

```
S → NP(Subjective) VP([])
```

a következőképpen olvasható: „Egy mondatot összeállíthatunk egy alanyi esetben álló *NP*-ből és az azt követő, üres alkategória-listájú *VP*-ből.” A 22.13. ábra bemutat egy elemzési fát, amelyik ezt a nyelvtant használja.

22.13. ábra - A „You give me the gold (Ideadod nekem az aranyat)” elemzési fája, amely bemutatja az ige és az igei kifejezés alkategóriáit



A harmadik lépésben gondolunk arra, hogy a kiegészítések mellett az igei (és egyéb) kifejezések **segédszavak**at (**adjuncts**) is kaphatnak, melyek nem egy igéhez kötődő kifejezések, hanem mindenféle igei kifejezésben feltűnhetnek. Az időt és helyet jelképező kifejezések segédszavak, mivel szinte minden cselekvés vagy esemény rendelkezhet időponttal és helyszínnel. Például a „now” határozószó az „I smell a wumpus now” mondatban és az „on Tuesday” *PP* a „give me the gold on Tuesday”-ban segédszavak. Íme, két szabály a segédszavak megengedéséhez:

```
VP(subcat) →    VP(subcat) PP

           |    VP(subcat) Adverb
```

4.2. Kiterjesztett nyelvtanok generálóképessége

Minden kiterjesztett szabály egy **szabályséma** (**rule schema**), mely a szabályok egy olyan halmazát jelenti, amit a kiterjesztett alkotórészek értékeinek minden lehetséges kombinációjával állítunk elő. A kiterjesztett

Created by XMLmind XSL-FO Converter.

nyelvtanok generálóképessége a kombinációk számától függ. Ha véges számú van, akkor a kiterjesztett nyelvtan ekvivalens egy környezetfüggetlen nyelvtannal: a szabálysséma felcserélhető egyedi környezetfüggetlen szabályokkal. Azonban ha végtelen számú van, akkor a kiterjesztett nyelvtanok nem környezetfüggetlen nyelveket is leírhatnak. Például az $a^n b^n c^n$ környezetfüggő nyelv leírható a következő kiterjesztett nyelvtannal:

```
S(n) → A(n) B(n) C(n)


A(1) → a A(n + 1) → a
         A(n)

B(1) → b B(n + 1) → b
         B(n)

C(1) → c C(n + 1) → c
         C(n)
```

5. Szemantikai értelmezés

Eddig csak a nyelv szintaktikai analízisét vizsgáltuk. Ebben a fejezetben rátérünk a **szemantiká**ra (**semantics**) – a megnyilatkozás (utterance) *jelentésének* kinyerésére. Ebben a fejezetben az elsőrendű logikát használjuk reprezentációs nyelvként, így a szemantikai értelmezés egy FOL kifejezés és egy nyelvi kifejezés összerendelésének a folyamata. Intuitív módon közelítve, a „the wumpus” kifejezés jelentése az a nagy, szőrös szörnyeteg, amit a logikában a $Wumpus_1$ logikai kifejezéssel jelzünk, és a „the wumpus is dead” jelentése a $Dead(Wumpus_1)$ logikai állítás. Ez a fejezet ezt az intuíciót pontosítja. Egy egyszerű példával kezdünk: a négyzetháló helyszíneit leíró szabállyal:

```
NP → Digit Digit
```

Kiterjesztjük a szabályt úgy, hogy minden összetevőhöz egy argumentumot illesztünk, amely az összetevő szemantikáját reprezentálja. A következőt kapjuk:

```
NP([x, y]) → Digit(x) Digit(y)
```

E szerint egy karaktersorozat, amely egy x szemantikájú számjegyből és egy azt követő y szemantikájú számjegyből áll, egy $[x, y]$ szemantikával rendelkező *NP*-t formál, amely a háló egy négyzetére alkalmazott jelölésünk.

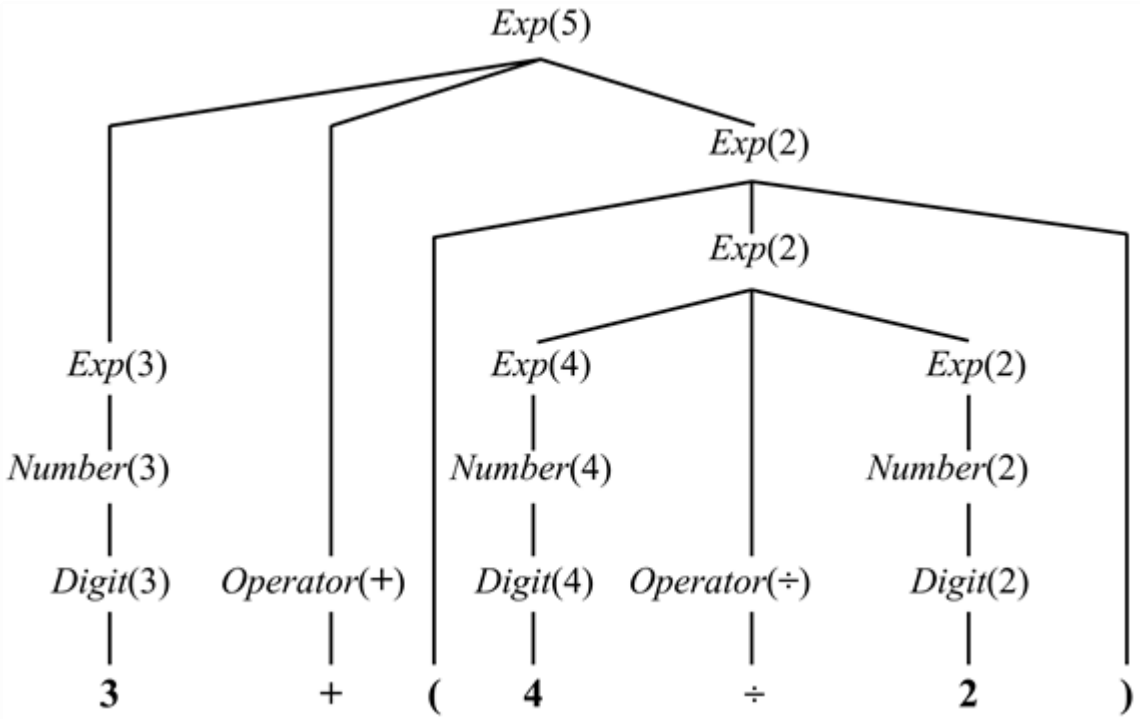
Vegyük észre, hogy a teljes *NP* szemantikája nagymértékben az összetevőinek szemantikáján alapszik. Már láttuk a **kompozíciós szemantika** (**compositional semantics**) ezen ötletét korábban: a logikában a $P \wedge Q$ jelentését P, Q és $\wedge$ határozzák meg; az aritmetikában az $x + y$ jelentését x, y és a + határozzák meg. A 22.14. ábra megmutatja, hogy a DCG jelölésrendszer hogyan használható egy aritmetikai kifejezéseket leíró nyelvtan szemantikával történő kiterjesztésére, és a 22.15. ábra mutatja a 3 + (4 ÷ 2) ezen nyelvtan alapján kapott elemzési fáját. Az elemzési fa gyökere *Exp*(5), egy kifejezés, melynek szemantikai értelmezése 5.

22.14. ábra - Szemantikával kibővített nyelvtan az aritmetikai kifejezésekre. Minden x_i változó egy összetevő szemantikáját reprezentálja. Vegye észre, hogy a {*teszt*} jelölést használjuk olyan logikai predikátumok definiálására, amelyeket ki kell elégíteni, de nem összetevők!

$$
\begin{aligned}
&Exp(x) \rightarrow Exp(x_1)\ Operator(op)\ Exp(x_2)\ \{x = Apply(op, x_1, x_2)\}\\
&Exp(x) \rightarrow (Exp(x))\\
&Exp(x) \rightarrow Number(x)\\
&Number(x) \rightarrow Digit(x)\\
&Number(x) \rightarrow Number(x_1)\ Digit(x_2)\ \{x = 10 \times x_1 + x_2\}\\
&Digit(x) \rightarrow x\ \{0 \leq x \leq 9\}\\
&Operator(x) \rightarrow x\ \{x \in \{+, -, \div, \times\}\}
\end{aligned}
$$

Created by XMLmind XSL-FO Converter.

22.15. ábra - A „3 + (4 ÷ 2)” füzér elemzési fája szemantikai értelmezésekkel



5.1. Az angol nyelv egy részletének szemantikája

Most már készen állunk arra, hogy az angol nyelv egy kis részhalmazára megírjuk a szemantikai kibővítést. Első lépésként azzal kezdünk, hogy melyik kifejezéshez milyen szemantikai értelmezéseket akarunk rendelni. A „John loves Mary” egyszerű mondatot fogjuk vizsgálni. A „John” *NP* szemantikus értelmezése a *John* logikai term kell legyen, és a mondatnak, mint egésznek, a *Loves*(*John, Mary*) logikai állítás kell a szemantikai értelmezése legyen. Ennyi világosnak látszik. A bonyolultabb rész a „loves Mary” *VP*. Ezen kifejezés szemantikai értelmezése se nem logikai term, se nem teljes logikai mondat. Intuitív módon kezelhetjük úgy, hogy a „loves Mary” egy leírás, ami lehet, hogy egy adott személyre vonatkozik, de lehet, hogy nem. (Jelen esetben Johnra vonatkozik.) Ez azt jelenti, hogy a „loves Mary” egy **predikátum** (**predicate**), amit ha egy személyt reprezentáló termmel kombinálunk (a személy, aki szeret), akkor egy teljes logikai mondatot állít elő. A λ jelölésrendszert használva (lásd 7.2. alfejezet) a „loves Mary”-t a következő predikátumként reprezentálhatjuk:

```
λx Loves(x, Mary)
```

Ezután szükségünk van egy szabályra, amely szerint „egy *obj* szemantikájú *NP*, melyet egy *rel* szemantikájú *VP* követ, együtt egy olyan mondatot eredményez, amelynek szemantikája a *rel* alkalmazása az *obj*-ra:

```
S(rel(obj)) → NP(obj) VP(rel)
```

A szabály szerint a „John loves Mary” szemantikai értelmezése

```
(λx Loves(x, Mary))(John)
```

ami megfelel a *Loves*(*John, Mary*)-nek.

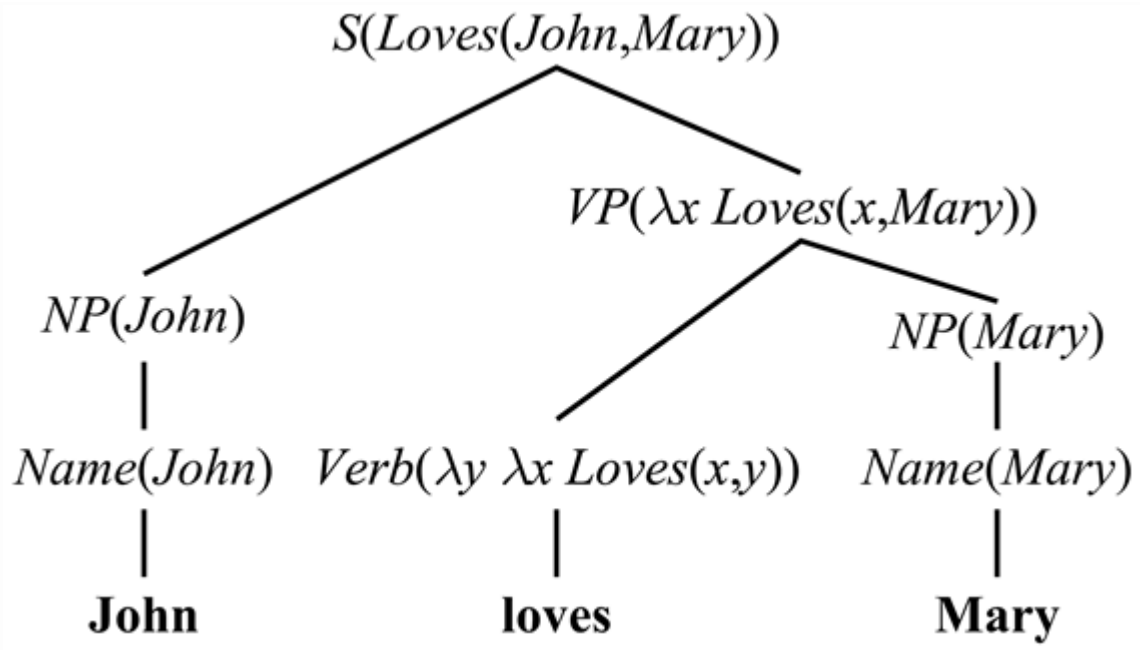
22.16. ábra - Egy nyelvtan, amely képes egy elemzési fa és szemantikai értelmezés levezetésére a „John loves Mary” (és három másik) mondat számára

Created by XMLmind XSL-FO Converter.

$S(rel(obj)) \rightarrow NP(obj)\ VP(rel)$
$VP(rel(obj)) \rightarrow Verb(rel)\ NP(obj)$
$NP(obj) \rightarrow Name(obj)$

$Name(John) \rightarrow$ **John**
$Name(Mary) \rightarrow$ **Mary**
$Verb(\lambda y\ \ \lambda x\ Loves(x, y)) \rightarrow$ **loves**

22.17. ábra - A „John loves Mary” füzér elemzési fája szemantikai értelmezésekkel



A szemantika hátralevő része az eddigi választásainkból egyenesen következik. Mivel a *VP*-ket predikátumként reprezentáljuk, jó ötlet konzisztens módon az igéket is predikátumként ábrázolni. A „loves” ige reprezentációja $\lambda y\ \lambda x\ \ Loves(x, y)$ predikátum, ami például egy *Mary* argumentum esetén $\lambda x\ \ Loves(x, Mary)$ predikátummal tér vissza.

A $VP \rightarrow Verb\ NP$ szabály alkalmazza az ige szemantikai értelmezéséből kapott predikátumot az *NP* (szemantikai) értelmezéséből kapott objektumra a teljes *VP* szemantikai értelmezésének kinyerésére. Végül a 22.16. ábrán látható nyelvtant és a 22.17. ábrán található elemzési fát kapjuk.

5.2. Idő és igeidő

Most tegyük fel, hogy a „John loves Mary” és a „John loved Mary” közötti különbséget szeretnénk reprezentálni. Az angol igeidőket (past [múlt], present [jelen] és future [jövő]) használ egy esemény relatív idejének jelzésére. Egy jó választás az események idejének reprezentálására a 10.3. alfejezet eseménykalkulus jelölésrendszere. Az eseménykalkulussal a két mondatunk a következőképpen értelmezhető:

$e \in Loves(John, Mary) \land During(Now, e)$

$e \in Loves(John, Mary) \land After(Now, e)$

Eszerint a „loves” és „loved” szavakra vonatkozó két lexikai szabályunk a következő lehet:

$Verb(\lambda y\ \lambda x\ e \in Loves(John, Mary) \land During(Now, e)) \rightarrow loves$

$Verb(\lambda y\ \lambda x\ e \in Loves(x, y) \land After(Now, e)) \rightarrow loved$

Created by XMLmind XSL-FO Converter.

E változástól eltekintve a nyelvtannal kapcsolatban minden más változatlan marad, ami ösztönző hír; azt sugallja, hogy helyes úton járunk, ha ilyen könnyedén bonyolíthatjuk például az igeidővel (bár csak érintettük az időre és igeidőre vonatkozó teljes nyelvtan felszínét). E bemelegítésként elért sikerrel készen állunk egy sokkal nehezebb reprezentációs probléma kezelésére.

5.3. Kvantifikálás

Nézzük a következő példát: „Every agent smells a wumpus.” A mondat valójában többértelmű: a javasolt értelmezés szerint az ágensek különböző wumpusokat észlelhetnek, de egy alternatív jelentés szerint csak egy wumpus van, akit mindenki érez.[12] A két interpretációt a következő módon ábrázolhatjuk:

$$\forall a\ a \in Agents \Rightarrow \exists w\ w \in Wumpuses \wedge \exists e\ e \in Smell(a, w) \wedge During(Now, e)$$

$$\exists w\ \ w \in Wumpuses\ \forall a\ a \in Agents \Rightarrow \exists e\ e \in Smell(a, w) \wedge During(Now, e)$$

A többértelműség problémáját elhalasztjuk későbbre, most csak az első értelmezést vizsgáljuk. Megpróbáljuk az összetevői szerint elemezni, *NP* és *VP* komponensekre bontva:

$$\text{Every agent } NP(\forall a\ a \in Agents \Rightarrow P)$$

$$\text{smells a wumpus } VP(\exists w\ w \in Wumpuses \wedge \exists e\ (e \in Smell(a, w) \wedge During(Now, e))$$

Máris két nehézség támadt. Először is, a teljes mondat szemantikája, úgy tűnik, megegyezik az *NP* szemantikájával, melyben a *VP* tölti ki a *P* rész szemantikáját. Ez azt jelenti, hogy a mondat szemantikáját nem formálhatjuk *rel*(*obj*) segítségével. Megtehetjük *obj*(*rel*) által, ami (legalábbis első ránézésre) kicsit különösnek tűnik. A második probléma az, hogy az *a* változót a *Smell* reláció argumentumaként kell megkapnunk. Más szavakkal, a mondat szemantikája a *VP* szemantikájának megfelelő *NP* argumentumrekeszbe való illesztésével adódik, miközben az *NP*-ben található *a* változót is betesszük a *VP* szemantika megfelelő argumentumrekeszébe. Úgy tűnik, mintha két funkcionális kompozíciónk lenne, és ez azt sugallja, hogy meglehetősen könnyű lesz összezavarodni. A bonyolultság abból a tényből ered, hogy a szemantikai struktúra nagyon különböző a szintaktikaitól.

Hogy elkerüljék ezt a zűrzavart, sok modern nyelvtan más módszert választ. Definiálnak egy **közbülső formát** (**intermediate form**) a szintaxis és a szemantika közötti közvetítésre. A közbülső formának két kulcstulajdonsága van. Egyrészt strukturálisan hasonló a mondat szintaxisához, így könnyen előállítható kompozíciós eszközökkel. Másrészt elég információt tartalmaz, és így lefordítható egy reguláris elsőrendű logikai mondattá. Mivel a szintaktikai és a logikai alak között helyezkedik el, néha **kvázilogikai formának** (**quasi-logical form**)[13] nevezik. A fejezetben egy kvázilogikai formát használunk, mely a teljes elsőrendű logikát tartalmazza, lambdakifejezéssekkel és egy új konstrukcióval kiegészítve, amit **kvantifikált termnek** (**quantified term**) nevezünk. Az „every agent” szemantikai értelmezését jelentő kvantifikált term a következőképpen írható:

$$[\forall a\ \ a \in Agents]$$

Ez olyan, mint egy logikai mondat, de ugyanolyan módon használjuk, mint a logikai termeket. Az „Every agent smells a wumpus” kvázilogikai értelmezése:

$$\exists e\ (e \in Smell([\forall a\ a \in Agents], [\exists w\ w \in Wumpuses]) \wedge During(Now, e))$$

A kvázilogikai alak generálásához szabályaink közül sok változatlan marad. Az *S*-re vonatkozó szabály *S* szemantikáját még mindig *rel*(*obj*) segítségével adja meg. Bizonyos szabályok változnak; az „a” szóra vonatkozó lexikai szabály a következő:

$$Article(\exists) \rightarrow a$$

míg a névelő és főnév kombinációjának szabálya:

$$NP([q\ x\ sem(x)]) \rightarrow Article(q)\ Noun(sem)$$

[12] Ha ez az értelmezés valószínűtlennek tűnik, akkor vizsgálja meg az „Every Protestant believes in a just God” mondatot.
[13] Egyes kvázilogikai formákra jellemző az is, hogy olyan többértelműségek tömör kifejezésére is alkalmasak, amelyeket logikai formában csak hosszú diszjunkciók segítségével lehetne kifejezni.

Created by XMLmind XSL-FO Converter.

E szerint az *NP* szemantikája egy kvantifikált term – ahol a kvantifikálást a névelő határozza meg – egy új *x* változóval és egy propozícióval, amely a főnév szemantikáját az *x* változóra alkalmazza. Az *NP*-re vonatkozó további szabályok hasonlók. A 22.18. ábra mutatja be a szemantikai típusokat, és példát ad minden egyes szintaktikai kategóriára a kvázilogikai ábrázolás szerint. A 22.19. ábra az „Every agent smells a wumpus" e megközelítés szerinti elemzését mutatja, a 22.20. pedig a teljes nyelvtant.

Ezek után át kell alakítanunk a kvázilogikai formát valódi elsőrendű logikaivá a kvantifikált termek igazi term-mé alakításával. Ezt egy egyszerű szabállyal tesszük meg: minden egyes *QLF* kvázilogika formán belüli [*q x P*(*x*)] kvantifikált termet a cseréljük ki *x*-re, és cseréljük a *QLF*-et *q x P*(*x*) *op QLF* formára, ahol az *op* operátor ⇒ akkor, ha a *q* ∀, és ∧, amennyiben a *q* ∃ vagy ∃ !. Például az „Every dog has a day (Minden kutyának van egy napja)" mondat a következő kvázilogikai formával rendelkezik:

```
∃e (e ∈ Has([∀d d ∈ Dogs], [∃a a ∈ Days], Now))
```

Nem specifikáltuk, hogy a két kvantifikált term melyikét kell elsőként elővenni, így valójában két lehetséges értelmezés van:

```
∀d  d ∈ Dogs ⇒ ∃a a ∈ Days ∧ ∃e e ∈ Has(d, a, Now)
```

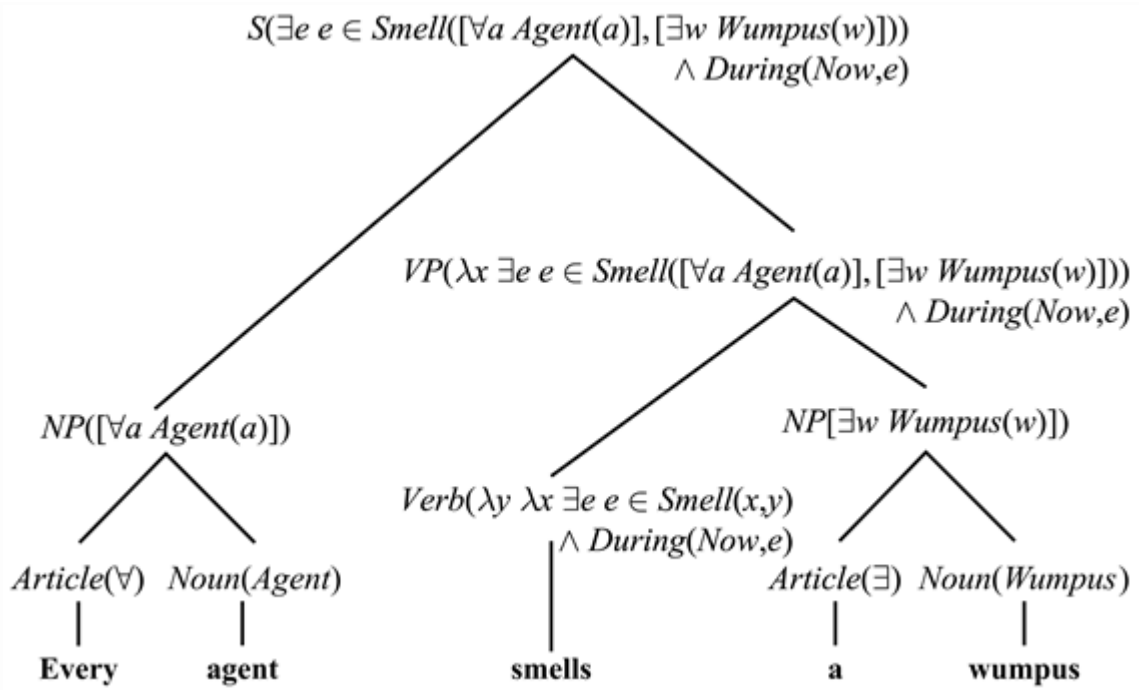
```
∃a a ∈ Days ∧ ∀d d ∈ Dogs ⇒ ∃e  e ∈ Has(d, a, Now)
```

22.18. ábra - Minden szintaktikai kategória kvázilogikai formájú kifejezésének típusát mutató tábla. A *t* → *r* jelölés egy függvényt takar, amely egy *t* típusú argumentumot fogad, és *r* típusú eredménnyel tér vissza. Például a *Preposition* szemantikai típusa *object*^{2} → *sentence*, ami azt jelenti, hogy az elöljárószó szemantikája egy függvény, amelyet ha két logikai objektumra alkalmazunk, akkor egy logikai mondatot eredményez.

Kategória	Típus	Példa	Kvázilogikai alak
S	*Sentence*	I sleep.	$\exists e\ e \in Sleep(Speaker) \wedge During(Now, e)$
NP	*object*	a dog	$[\exists d\ Dog(d)]$
PP	$object^2 \rightarrow sentence$	in [2, 2]	$\lambda x\ In(x, [2, 2])$
RelClause	*object* → *sentence*	that sees me	$\lambda x\ \exists e\ e \in Sees(x, Speaker) \wedge During(Now, e)$
VP	$object^n \rightarrow sentence$	sees me	$\lambda x\ \exists e\ e \in Sees(x, Speaker) \wedge During(Now, e)$
Adjective	*object* → *sentence*	smelly	$\lambda x\ Smelly(x)$
Adverb	*event* → *sentence*	today	$\lambda e\ During(e, Today)$
Article	*quantifier*	the	$\exists!$
Conjunction	$sentence^2 \rightarrow sentence$	and	$\lambda p, q\ (p \wedge q)$
Digit	*object*	7	7
Noun	*object* → *sentence*	wumpus	$\lambda x\ \ x \in Wumpuses$
Preposition	$object^2 \rightarrow sentence$	in	$\lambda x\ \ \lambda y\ In(x, y)$
Pronoun	*object*	I	*Speaker*
Verb	$object^n \rightarrow sentence$	eats	$\lambda y\ \lambda x\ \exists e\ \ e \in Eats(x, y) \wedge During(Now, e)$

22.19. ábra - Az „Every agent smells a wumpus" mondat elemzési fája, amely mind a szintaktikai struktúrát, mind a szemantikai értelmezéseket mutatja

Created by XMLmind XSL-FO Converter.



22.20. ábra - Szemantikával ellátott nyelvtan kvázilogikai formában

$S(rel(obj)) \rightarrow NP(obj)\ VP(rel)$
$S(conj(sem_1, sem_2)) \rightarrow S(sem_1)\ Conjunction(conj)\ S(sem_2)$

$NP(sem) \rightarrow Pronoun(sem)$
$NP(sem) \rightarrow Name(sem)$
$NP([q\ x\ sem(x)]) \rightarrow Article(q)\ Noun(sem)$
$NP([q\ x\ obj \wedge rel(x)]) \rightarrow NP([q\ x\ obj])\ PP(rel)$
$NP([q\ x\ obj \wedge rel(x)]) \rightarrow NP([q\ x\ obj])\ RelClause(rel)$
$NP([sem_1, sem_2]) \rightarrow Digit(sem_1)\ Digit(sem_2)$

$VP(sem) \rightarrow Verb(sem)$
$VP(rel(obj)) \rightarrow VP(rel\ NP(obj)$
$VP(sem_1\ (sem_2)) \rightarrow VP(sem_1)\ Adjective(sem_2)$
$VP(sem_2\ (sem_2)) \rightarrow VP(sem_1)\ PP(sem_2)$

$RelClause(sem) \rightarrow$ **that** $VP(sem)$

$PP(\lambda x\ rel(x, obj)) \rightarrow Preposition(rel)\ NP(obj)$

Az első szerint minden kutyának megvan a maga napja, míg a második szerint van egy különleges nap, ami minden kutyára ugyanaz. A kettő közötti választás a többértelműség feloldásának feladata. A kvantifikált termek balról jobbra sorrendje gyakran illeszkedik a kvantifikálók balról jobbra sorrendjére, de más tényezők is számítanak. A kvázilogikai alak előnye az, hogy tömören reprezentálja az összes lehetőséget. A hátránya pedig az, hogy nem segíti a választást közöttük; amihez szükségünk van a többértelműség feloldásának teljes erejére az összes tényt felhasználva.

5.4. Pragmatikus értelmezés

Megmutattuk, hogy egy ágens hogyan képes észlelni egy szófüzért, és hogyan képes egy nyelvtant használni a lehetséges szemantikai értelmezések halmazának előállítására. Most annak a problémáját vizsgáljuk, hogy az

Created by XMLmind XSL-FO Converter.

értelmezés hogyan tehető teljessé az egyes jelöltek értelmezéshez az adott szituációt leíró kontextusfüggő információk hozzáadásával.

A gyakorlati információra legnyilvánvalóbban a **referenciális index**ek (**indexical**s) feloldásához van szükség, amelyek közvetlenül az adott szituációra utaló kifejezések. Például az „I am in Boston today (Ma Bostonban vagyok)" mondatban az „I" és „today" referenciális indexek értelmezése attól függ, hogy ki és mikor ejtette ki a mondatot. A referenciális indexeket „konstansokkal" (mint például a *Speaker*) reprezentáljuk, amelyek valójában **változó dolgok** (**fluents**) – azaz a szituációtól függnek. A hallgatónak, aki észleli a szólásaktust, azt is észlelnie kell, hogy ki a beszélő, és felhasználja ezt az információt a referenciális indexek feloldására. Például a hallgató tudhatja, hogy $T((Speaker = Agent_B), Now)$.

Egy felszólítás, mint például a „go to 2, 2" implicit módon a hallgatóra vonatkozik. Eddig az *S* nyelvtanunk csak deklaratív mondatokat írt le. Könnyen kiterjeszthetjük felszólítások kezelésére.[14]

A felszólítást olyan *VP* segítségével formálhatjuk, ahol az alany implicit módon a hallgató. Meg kell különböztetnünk a felszólításokat az állításoktól, ezért megváltoztatjuk az *S*-re vonatkozó szabályokat a szólásaktus típusának beillesztésével a kvázilogikai formába:

```
S(StatementI(Speaker, rel(obj))) → NP(obj) VP(rel)

S(Command(Speaker, rel(Hearer))) → VP(rel)
```

Így a „go to 2, 2" kvázilogikai alakja:[15]

```
Command(∃e e ∈ Go(Hearer, [2, 2]))
```

5.5. Nyelv generálása DCG-kkel

Eddig egy nyelv *elemzésével* foglalkoztunk, nem a generálásával. A generálás hasonlóan gazdag téma. A megfelelő megnyilatkozás kiválasztása egy állítás kifejezésére sok hasonló választást von maga után, mint egy megnyilatkozás elemzése.

Emlékezzünk vissza, hogy a DCG egy logikai programozási rendszer, amely kényszereket határoz meg egy karaktersorozat és annak elemzése között. Tudjuk, hogy az *Append* predikátum logikai programozási definíciója felhasználható annak közlésére is, hogy az *Append*([1, 2], [3], *x*) esetében *x* = [1, 2, 3], és arra is, hogy felsoroljuk azon *x* és *y* értékeket, melyek az *Append*(*x*, *y*, [1, 2, 3]) kifejezést igazzá teszik. Hasonlóképpen írhatunk egy definíciót *S*-re, amely kétféleképpen használható: az elemzéshez megkérdezzük, hogy *S*(*sem*, [*John*, *Loves*, *Mary*]), és azt kapjuk vissza, hogy *sem* = *Loves*(*John*, *Mary*); a generáláshoz azt kérdezzük, hogy *S*(*Loves*(*John*, *Mary*), *words*), és azt kapjuk vissza, hogy *words* = [*John*, *Loves*, *Mary*]. Tesztelhetünk egy nyelvtant az *S*(*sem*, *words*) kérdéssel, visszakapva azokat a [*sem*, *words*] párok sorozatát válaszként, melyeket a nyelvtan generált.

Ez a módszer a fejezetben bemutatott egyszerű nyelvtanokra működik, de nagyobb nyelvtanokra történő felskálázás során lehetnek problémák. A logikai következtető gép által használt keresési stratégia nagyon fontos; a mélységi keresési stratégiák végtelen ciklusokhoz vezethetnek. Figyelmet kell fordítani a szemantikai alak pontos részleteire is. Előfordulhat, hogy egy adott nyelvtan nem tudja kifejezni az $X \wedge Y$ logikai alakot az *X* és *Y* bizonyos értékeire, de ki tudja fejezni $Y \wedge X$-et; ez azt sugallja, hogy szükségünk lesz valamilyen módszerre a szemantikai alakok kanonizálására, vagy ki kell terjesztenünk az egyesítő eljárást úgy, hogy az $X \wedge Y$ és az $Y \wedge X$ egyesíthető legyen.

Komolyabb generálási feladatok bonyolultabb modelleket használnak, melyek különböznek az elemzés nyelvtanától, és pontosabban szabályozzák, hogy a szemantikai komponenseket pontosan hogyan fejezzük ki. A

[14] Egy teljes kommunikáló ágens megvalósításához szükségünk lenne a kérdések nyelvtanára is. A kérdések kezelése kívül esik e könyv területén, mivel összetevők közötti **hosszú távú függőség**eket (**long-distance dependecies**) vonzanak magukkal. Például a „Whom did the agent tell you to give the gold to?" mondatban a záró „to" elemzésének hiányzó *NP*-jű *PP*-nek kell lennie; a hiányzó *NP*-t a mondat első szava, a „who" helyettesíti. Kiterjesztések egy komplex rendszere biztosítja, hogy a hiányzó *NP*-k összeilleszthetődjenek a helyettesítő szavakkal.

[15] Vegyük észre, hogy egy felszólítás kvázilogikai formája nem foglalja magában az esemény idejét (azaz a *During*(*Now*, *e*)-t)! Ez azért van, mert a „go" valójában a szó igeidő nélküli változata, nem a jelen idejű. Nem lehet megállapítani a különbséget a „go" esetében, de figyelje meg a „Be good!" felszólítás helyes alakját (a „be" igeidő nélküli alakját használva), amely nem „Are good!". Annak biztosítására, hogy a helyes igeidőt használjuk, kiegészíthetnénk a *VP*-ket egy igeidő kiterjesztéssel, és a felszólítások szabályának jobb oldalára ezt írhatnánk: *VP*(*rel*, *untensed*).

Created by XMLmind XSL-FO Converter.

szisztematikus nyelvtan egy olyan megközelítés, amely könnyűvé teszi, hogy a szemantikai alak legfontosabb részeire nagyobb hangsúlyt helyezzünk.

6. Többértelműség és feloldása

Bizonyos esetekben a hallgatók tudatában vannak egy megnyilatkozás többértelműségének. Íme, néhány példa a napilapok szalagcímeiből:

Squad helps dog bite victim.

Helicopter powered by human flies.

Once-sagging cloth diaper industry saved by full dumps.

Portable toilet bombed; police have nothing to go on.

British left waffles on Falkland Islands.

Teacher strikes idle kids.

Milk drinkers are turning to powder.

Drunk gets nine months in violin case.

(A mondatok fordítását és értelmezését az olvasóra hagyjuk – *a ford.*)

Fontos

Azonban legtöbb esetben a beszéd, amit hallunk, egyértelműnek tűnik. Ezért amikor a kutatók először kezdtek a nyelv elemzésére számítógépeket használni a hatvanas években, igen meglepődve tapasztalták, hogy *majdnem minden megnyilatkozás erősen többértelmű, bár az alternatív értelmezések egy anyanyelvi beszélő számára lehet, hogy nem nyilvánvalók*. Egy nagy nyelvtannal és szókinccsel rendelkező rendszer értelmezések ezreit találhatja egy teljesen átlagos mondatra. Vegyük például a „The batter hit the ball” mondatot, amelynek látszólag egy egyértelmű értelmezése van, miszerint a baseballjátékos megüt egy baseball-labdát. De különböző értelmezést kapunk, ha az ezt megelőző mondat ez: „The mad scientist unleashed a tidal wave of cake mix towards the ballroom.” Ez a példa a **lexikális többértelműség**en (**lexical ambiguity**) alapszik, amikor egy szónak egynél több jelentése van. A lexikális többértelműség igen gyakori: a „back” lehet határozószó (go back), melléknév (back door), főnév (the back of the room) és ige (back up your files). A „Jack” lehet név, főnév (kártya, hatszögű fém játékdarab, hajózási zászló, hal, hím szamár, dugasz vagy nehéz objektumok emelésére szolgáló eszköz), illetve lehet ige (felemelni egy autót, fénnyel vadászni vagy erősen megütni a baseball-labdát).

A **szintaktikai többértelműség** (**syntactic ambiguity**) (avagy **strukturális többértelműség** (**structural ambiguity**)) a lexikális többértelműséggel együtt vagy nélküle is előfordulhat. Például az „I smelled a wumpus in 2,2” füzérnek két értelmezése lehet: az egyikben az „in 2,2” prepozíciós kifejezés a főnevet módosítja, a másikban az igét. A szintaktikai többértelműség **szemantikai többértelműség**hez (**semantic ambiguity**) vezet, mivel az egyik értelmezés szerint a wumpus a 2,2-n van, a másik szerint a szaga van a 2,2-n. Ebben az esetben a rossz értelmezés halálos tévedés lehet.

A szemantikai többértelműség még olyan kifejezésekben is feltűnhet, ahol nincs lexikális vagy szintaktikai többértelműség. Például a „cat person” főnévi kifejezés takarhat olyasvalakit, aki szereti a macskaféléket, vagy az *Attack of the Cat People* (A macskaemberek támadása) c. film szalagcímét is. Egy „coast road” lehet olyan út, ami a partvonalat követi, és olyan, amelyik a partra vezet.

Végezetül lehet többértelműség a szó szerinti és az átvitt értelmű jelentések között. A jelképes beszéd fontos a költészetben, de meglepően gyakori a mindennapi beszédben is. A **metonímia** (**metonymy**) egy olyan jelképes beszéd, amikor az egyik objektum helyére egy másikat állítunk. Amikor azt halljuk, hogy „Chrysler announced a new model”, nem értelmezzük akként, hogy a cégek beszélni tudnak; inkább azt értjük ezalatt, hogy a céget képviselő szóvivő tette a bejelentést. A metonímia gyakori, és az emberi hallgatók gyakran észrevétlenül interpretálják. Sajnálatos módon a nyelvtanunk abban a formában, ahogy leírtuk, nem ennyire rugalmas. Ahhoz,

Created by XMLmind XSL-FO Converter.

hogy a metonímia szemantikáját megfelelően kezelni tudjuk, a többértelműség teljesen új szintjét kell bevezetnünk. Ezt úgy tesszük meg, hogy két objektumot használunk a mondat minden kifejezésének szemantikai értelmezésénél: egyet arra az objektumra, amelyre a kifejezés szó szerint vonatkozik (Chrysler), míg egy másikkal utalunk a fogalomcserére (szóvivő). Ezek után meg kell mondanunk, hogy összefüggés van a kettő között. Jelenlegi nyelvtanunkban a „Chrysler announced” értelmezése a következő:

$$\exists x, e\ x = Chrysler \wedge e \in Announce(x) \wedge After(Now, e)$$

Ezt a következőre kell megváltoztatnunk:

$$\exists m, x, e\ x = Chrysler \wedge e \in Announce(m) \wedge After(Now, e) \wedge Metonymy(m, x)$$

E szerint van egy olyan *x* entitás, amely azonos a Chryslerrel, és van egy másik *m* entitás, amelyik a bejelentést tette, és a kettő metonímiai relációban áll egymással. A következő lépés annak meghatározása, hogy milyen metonímiarelációk fordulhatnak elő. A legegyszerűbb eset az, amikor nincs semmilyen metonímia – az *x* szó szerinti objektum és az *m* metonímiai objektum azonos:

$$\forall m, x\ (m = x) \Rightarrow Metonymy(m, x)$$

A Chrysler példánál helyénvaló általánosítás az, hogy egy vállalat állhat a szóvivője helyén:

$$\forall m, x\ x \in Organizations \wedge Spokesperson(m, x) \Rightarrow Metonymy(m, x)$$

További fogalomcserék lehetnek a szerző a munkákra (I read *Shakespeare*), vagy általában a létrehozó az alkotásra (I drive a *Honda*) és a rész az egészre (The Red Sox need a strong *arm*). A metonímia egyes példái, mint a „The *ham sandwich* on Table 4 wants another beer” újszerűbbek, és az adott szituációnak megfelelően értelmezik őket.

Az itt vázolt szabályok lehetővé teszik a számunkra, hogy a „Chrysler announced a new model” mondat magyarázatát elkészítsük, de a magyarázat nem logikai következtetés útján jön létre. Valószínűségi vagy nemmonoton következtetést kell használnunk ahhoz, hogy magyarázatjelöltekkel álljunk elő.

Egy **metafora** (**metaphor**) olyan jelképes beszéd, amikor egy adott szó szerinti jelentéssel bíró kifejezést használunk egy másik jelentés sugalmazására analógia segítségével. A legtöbb ember úgy gondol a metaforára, mint a költők eszközére, mely nem játszik nagy szerepet a mindennapi szövegekben. Mindazonáltal nagyon sok alapvető metaforát olyan gyakran használunk, hogy még azt sem vesszük észre, hogy azok metaforák. Egy ilyen metafora annak a gondolata, hogy a „több felfelé van”. Ez a metafora lehetővé teszi számunkra annak közlését, hogy az árak emelkedtek, felmásztak vagy rakéta módjára szárnyalnak, hogy a hőmérséklet csökken vagy esik, hogy a bizalom valakiben csökken, illetve hogy egy sztár népszerűsége megugrott vagy szárnyal.

Az ilyen metaforák megközelítésének két módja lehetséges. Az egyik az, hogy a metafora teljes tudását beépítjük a szótárba – új jelentéseket adunk az emelkedik, esik, mászik stb. szavakhoz, melyek úgy írják le őket, hogy mindenféle mértékre vonatkozhatnak, nem csak a magasságra. Ez a módszer sok alkalmazás számára elegendő, de nem ragadja meg a metafora azon generálóképességét, amely az emberek számára lehetővé teszi új kifejezések használatát – mint például az „alábukik” vagy a „kirobbantja a tetőt” – anélkül hogy félreértéstől tartanának. A másik módszer a gyakori metaforákról explicit tudást illeszt be a rendszerbe, és ezeket használja az új kifejezések olvasás közbeni megértéséhez. Például tegyük fel, hogy egy rendszer ismeri a „több felfelé van” metaforáját. Azaz tudja, hogy az olyan logikai kifejezések, melyek egy függőleges skálán értelmezett pontra vonatkoznak, értelmezhetők egy mennyiségi skála megfelelő pontjaira vonatkozóként. Ezek után az „eladások magasak” kifejezéshez tartozna egy *Altitude*(*Sales, High*) szó szerinti értelmezés, amit metafora segítségével *Quantity*(*Sales, Much*)-ként értelmezhetünk.

6.1. A többértelműség feloldása

Fontos

Ahogy korábban mondtuk, a többértelműség feloldása *diagnózis kérdése*. A beszélő szándéka a kommunikációra a megnyilatkozásban szereplő szavak nem megfigyelt oka, és a hallgató dolga, hogy a szavakból és a szituációról való tudásából kiindulva visszafelé előállítsa a beszélő legvalószínűbb szándékát. Más szavakkal, a hallgató a következőt oldja meg:

Created by XMLmind XSL-FO Converter.

$$\underset{intent}{\operatorname{argmax}} Likelihood(intent \mid words, situation)$$

ahol a *Likelihood* lehet valószínűség vagy a preferenciák bármilyen számszerű mértéke. Valamilyen preferenciára szükség van, mivel a szintaktikai és szemantikai értelmezési szabályok önmagukban nem tudnak a kifejezéshez vagy mondathoz egyetlen helyes értelmezést meghatározni. Ezért részekre bontjuk a munkát: a szintaktikai és szemantikai értelmezés felelős a lehetséges értelmezések halmazának előállításáért, a többértelműség feloldásának folyamata pedig kiválasztja a legjobbat.

Vegyük észre, hogy a szólásaktus szándékáról, és nem csak a beszélő által kimondott aktuális kijelentésről beszélünk. Például ha azt halljuk egy politikustól, hogy „I am not a crook”, akkor lehet, hogy csak 50% valószínűséget adunk annak, hogy a politikus nem bűnöző, ugyanakkor 99,999%-ot annak, hogy a beszélő nem egy kampós pásztorbot (a *crook* másik jelentése – *a ford.*). Mégis nagyobb valószínűséget rendelünk ahhoz az értelmezéshez, hogy

$$Assert(Speaker, \neg(Speaker \in Criminals))$$

mivel ez a valószínűbb dolog, amit mond.

Vizsgáljuk ismét az „I smelled a wumpus in 2,2.” többértelmű példát! Egy preferenciaheurisztika a **jobbra asszociáció** (**right association**), amely szerint amikor el kell döntenünk, hogy az „in 2,2” *PP*-t hova helyezzük el az elemzési fában, akkor azt preferáljuk, hogy a jobb szélen elhelyezkedő összetevőhöz csatoljuk, ami jelen esetben az „a wumpus” *NP*. Természetesen ez csak egy heurisztika; az „I smelled a wumpus with my nose” mondatra a heurisztikát felülbírálná az a tény, hogy az „a wumpus with my nose” *NP* nem valószínű.

A többértelműség feloldása a tények kombinálásával oldható meg, felhasználva a könyvben eddig látott tudásábrázolásra és bizonytalanság melletti következtetésre alkalmas minden technikát.

A tudást négy modellre bonthatjuk:

1. A **világ modell** (**world model**): annak valószínűsége, hogy egy állítás előfordul a világban.

2. A **mentális modell** (**mental model**): annak valószínűsége, hogy a beszélő létrehozza azt a szándékát, hogy egy adott tényt a hallgatóval közöljön, feltéve, hogy az bekövetkezik. Ez annak modelljét kombinálja, amit a beszélő hisz azzal, amit a beszélő hisz arról, amit a hallgató hisz és így tovább.

3. A **nyelvi modell** (**language model**): annak valószínűsége, hogy szavak egy bizonyos füzére lesz kiválasztva, azt feltételezve, hogy a beszélőben létezik egy adott tény elmondásának szándéka. A fejezetben ismertetett CFG- és DCG-modellek a valószínűségre logikai (igaz/hamis) modellt használnak: egy füzérnek vagy lehet egy adott értelmezése, vagy nem. A következő fejezetben a CFG olyan valószínűségi változatát látjuk majd, amely a többértelműség feloldására egy jobban informált nyelvi modellt készít.

4. Az **akusztikai modell** (**acoustic model**): annak valószínűsége, hogy hangok egy adott sorozata keletkezik azt feltéve, hogy a beszélő szavak egy adott sorozatát választotta. A beszédfelismerést a 15.6. alfejezet tárgyalta.

7. Szövegértés

Egy **szöveg** (**discourse**) a nyelv tetszőleges karaktersorozata – tipikusan olyan, amelyik hosszabb, mint egy mondat. A jegyzetek, a regények, az időjárás-jelentések és a párbeszédek mind szövegek. Eddig teljesen figyelmen kívül hagytuk a szövegek problémáit, a nyelv önálló mondatokra bontását helyeztük előtérbe, melyeket *in vitro* tanulmányozhattunk. Ez a fejezet a mondatokat természetes környezetükben tanulmányozza. Két részproblémát fogunk vizsgálni: az utalásfeloldást és a koherenciát.

7.1. Utalásfeloldás

Created by XMLmind XSL-FO Converter.

Az **utalásfeloldás** (**reference resolution**) egy olyan névmás vagy határozott főnévi kifejezés értelmezése, amely a világ egy objektumára utal.[16] A feloldás a világról szóló tudáson és a szöveg korábbi részein alapszik. Vizsgáljuk a következő szakaszt:

„John flagged down the waiter. He ordered a ham sandwich."

Annak megértéséhez, hogy a „he" a második mondatban Johnra utal, meg kellett értenünk, hogy az első mondat két személyt említ, valamint azt, hogy John játssza a vendég szerepét, és ezért valószínűleg rendel, míg a pincér nem. Általában az utalásfeloldás egy hivatkozott kiválasztása a jelöltek listájáról, de néha magában foglalja új jelöltek létrehozását is. Vegyük például a következő mondatot:

„After John proposed to Marsha, they found a preacher and got married. For the honeymoon, they went to Hawaii. (Miután John megkérte Marshát, találtak egy papot, és összeházasodtak. Hawaiira mentek nászútra.)"

Itt a „honeymoon" határozott főnévi kifejezés egy olyan dologra utal, amire csak implicit módon utal a „married" ige. A „they" névmás egy olyan csoportra utal, amit explicit módon nem említettek előtte: John és Marsha (de *nem* a pap).

A legjobb hivatkozott dolog kiválasztása egy olyan többértelműség feloldási folyamat, amely többféle szintaktikai, szemantikai és pragmatikus értelmezési információ kombinálásán alapszik. Bizonyos nyomravezetők kényszerek formájában adottak. Például a névmásoknak nemben és számosságban egyezniük kell a hivatkozottakkal: a „he" például utalhat Johnra, de nem Marshára; a „they" utalhat egy csoportra, de nem egyetlen személyre. A névmásoknak a reflexivitás szintaktikai kényszereinek is meg kell felelniük. Például a „he saw him in the mirror" mondatban a két névmásnak két különböző emberre kell vonatkoznia, míg a „he saw himself" kifejezésben ugyanarra az emberre kell utalniuk. Vannak a szemantikai konzisztenciára vonatkozó kényszerek is. A „he ate it" kifejezésben a „he" névmásnak olyan dologra kell utalnia, ami eszik, míg az „it" olyanra, amit megehetnek.

Bizonyos vezérfonalak olyan preferenciák, amik nem állnak mindig fenn. Például amikor egymás utáni mondatoknak párhuzamos struktúrája van, olyankor előnyös, ha a névmási utalások követik ezt a struktúrát. Így a

„Marsha flew to San Francisco from New York. John flew there from Boston."

mondatokban azt preferáljuk, hogy a „there" San Franciscóra utal, mivel az ugyanazt a szintaktikai szerepet tölti be. Párhuzamos struktúra hiányában az alanyt preferáljuk a tárgy ellenében. Így a

„Marsha gave Sally the homework assignment. Then she left."

szövegben „Marsha" az első mondat alanya, így ő a „she" preferált előzménye. Egy másik preferencia annak az entitásnak az előtérbe helyezése, amelynek a tárgyalása a legnyilvánvalóbb. Ha a következő mondatpárt önállóan vizsgáljuk

„Dana dropped the cup on the plate. It broke."

akkor felmerül egy probléma: nem világos, hogy az „it" a pohárra vagy a tányérra utal. Azonban egy nagyobb kontextusban a többértelműség feloldódik:

„Dana was quite fond of the blue cup. The cup had been a present from a close friend. Unfortunately, one day while setting the table, Dana dropped the cup on the plate. It broke."

Itt a pohár van a figyelem középpontjában, ezért ez a preferált hivatkozott dolog.

Sokféle utalásfeloldó algoritmust találtak ki. Az egyik első (Hobbs, 1978) figyelemre méltó, mivel egy olyan mértékű statisztikai ellenőrzést végeztek rajta, ami abban az időben szokatlan volt. Három különböző szövegfajtát használva Hobbs 92%-os pontosságot mutatott ki. A módszer azt feltételezte, hogy egy elemző egy helyes elemzést készített, ami azonban nem állt rendelkezésére, ezért Hobbs kézzel készítette el az elemzéseket. A Hobbs algoritmus keresésként működik: az aktuális mondattól indulva visszafelé keres a mondatokban. Ez a technika biztosítja, hogy a legutóbbi jelölteket vizsgálja először. Egy mondaton belül szélességi keresést végez,

[16] A nyelvészetben az utalást egy olyan dologra, ami már szerepelt, **anafor**ának (**anaphoric reference**) nevezzük. Az utalást egy később szereplő dologra **kataforá**nak (**cataphoric reference**) nevezzük, mint például a „he" névmás a „When he won his first tournament, Tiger was 20".

Created by XMLmind XSL-FO Converter.

balról jobbra haladva. Ez azt biztosítja, hogy az alanyokat a tárgyak előtt vizsgálja. Az algoritmus kiválasztja az első jelöltet, amely a most vázolt kényszereknek eleget tesz.

7.2. Egy koherens szöveg struktúrája

Nyissa fel ezt a könyvet tízszer véletlenszerűen, és másolja le az első mondatot minden oldalról. Az eredmény szükségszerűen inkoherens lesz. Hasonlóképpen, ha veszünk egy koherens, tíz mondatból álló szöveget, majd permutáljuk a mondatokat, az eredmény inkoherens lesz. Ez azt demonstrálja, hogy a természetes nyelvű szövegek mondatai nagyon különböznek a logika állításaitól. A logikában, bármilyen sorrendben is KÖZÖLJÜK az *A*, *B*, és *C* állításokat egy tudásbázissal, az $A \wedge B \wedge C$ konjukciót kapjuk. Természetes nyelvben a mondatok sorrendje számít; vegyük például a „Go two blocks. Turn right” és a „Turn right. Go two blocks” szövegek közötti különbséget.

A szövegek mondatok feletti struktúrával rendelkeznek. Ezt a struktúrát a szöveg következő nyelvtanának segítségével vizsgálhatjuk meg:

```
Segment(x) → S(x)

Segment(CoherenceRelation(x, y)) → Segment(x) Segment(y)
```

E nyelvtan szerint a szöveg szegmensekből áll, ahol minden szegmens vagy egy mondat, vagy mondatok egy csoportja, és a szegmenseket **koherenciarelációk** (**coherence relations**) kötik össze. A „Go two blocks. Turn right” szövegben a koherenciareláció az, hogy az első mondat *engedélyezi* a másodikat: a hallgatónak akkor kell jobbra fordulnia, miután ment két háztömbnyit. Különböző kutatók koherenciarelációk különböző készletét javasolták; a 22.21. ábra felsorol egy reprezentatív halmazt. Most vizsgáljuk meg a következő történetet:

1. A funny thing happened yesterday.
2. John went to a fancy restaurant.
3. He ordered the duck.
4. The bill came to $50.
5. John got a shock when he realized he had no money.
6. He had left his wallet at home.
7. The waiter said it was all right to pay later.
8. He was very embarrassed by his forgetfulness.

Itt, az (1) mondat az *Evaluation* relációban áll a szöveg hátralevő részével; (1) a beszélő megjegyzése a szöveghez. A (2) mondat engedélyezi a (3)-at, és a (2–3) pár együtt

okozza (4)-et, azzal az implicit köztes állapottal, hogy János megette a kacsát. Ezek után (2–4) szolgál a szöveg hátralevő részének alapjául. A (6) mondat az (5) magyarázata, és (5–6) engedélyezi (7)-et. Vegyük észre, hogy ez egy *Enable*, nem *Cause*, mivel a pincérnek lehetett volna más reakciója is. Az (5–7) együtt okozza (8)-at. A 22.13. feladat arra kéri majd, hogy rajzolja fel e szöveg elemzési fáját.

22.21. ábra - Koherenciarelációk listája (Hobbs, 1990) alapján. Mindegyik reláció két, egymást követő szövegrészlet, S_1 és S_2 között áll fenn.

Created by XMLmind XSL-FO Converter.

- **Lehetővé teszi** vagy **okozza**: S_1 előidéz egy olyan állapotváltozást (amely implicit is lehet), amely okozza, vagy lehetővé teszi S_2-t. Példa: „Kimentem. Az iskolába vezettem." (A kimenetel implicit módon lehetővé teszi, hogy beüljünk az autóba.)
- **Magyarázat**: a lehetővé tétel ellentettje: S_2 okozza vagy lehetővé teszi S_1-et, és így egy magyarázat rá. Példa: „Elkéstem az iskolából. Elaludtam."
- **Háttér**: S_1 leír egy szituációt vagy hátteret S_2 számára. Példa: „Sötét és viharos éjszaka volt. *A történet folytatódik...*"
- **Kiértékelés**: S_2-ből következik, hogy S_1 része a beszélő azon tervének, hogy a szövegrészletet szólásaktusként végrehajtsa. Példa: „Egy mulatságos dolog történt. *A történet folytatódik...*"
- **Szemléltetés**: S_2 az S_1-ben levő általános dolog egy példája. Példa: „Ez az algoritmus megfordít egy listát. Az $[A, B, C]$ bemenetet leképezi a $[C, B, A]$-ra."
- **Általánosítás**: S_1 az S_2-ben levő általános dolog egy példája. Példa: „Az $[A, B, C]$ bemenetet leképezi a $[C, B, A]$-ra. Általánosságban ez az algoritmus megfordít egy listát."
- **Elvárással ellentétes**: S_2-ből $\neg P$-t következteti ki, negálva az S_1-ből szokásos P következtetést. Példa: „Ez a cikk gyenge. Másrészt viszont érdekes."

A koherenciarelációk szolgálnak egy szöveg összekötésére. Vezetik a beszélőt annak eldöntésében, hogy mit mondjon és mit tekintsen magától értetődőnek, és segítik a hallgatót a beszélő szándékának megértésében. A koherenciarelációk a mondatok többértelműségének szűrőjeként szolgálhatnak: önállóan a mondatok lehetnek többértelműek, de ezen többértelmű értelmezések többsége nem illeszthető össze egy koherens szöveggé.

Eddig az utalásfeloldást és a szövegstruktúrát külön vizsgáltuk. Azonban a kettő valójában összefügg. Grosz és Sidner (1986) elmélete például azzal foglalkozik, hogy a beszélő és hallgató figyelme mire koncentrál a szöveg során. Elméletük tartalmaz egy vermet, amely **fókuszter**eket (**focus space**s) tartalmaz. Bizonyos megnyilvánulások a fókusz elmozdulását okozzák egy elem verembe helyezésével vagy kivételével. Például az éttermi történetben a „John went to a fancy restaurant" mondat új fókuszt helyez a verem tetejére. Ezen a fókuszon belül a beszélő egy határozott *NP*-t használhat a „the waiter"-re történő utalásra (az „a waiter" általános *NP* helyett). Ha a történet folytatódna egy „John went home" mondattal, akkor a fókuszteret kivenné a verem tetejéről, és a szöveg nem utalhatna tovább a pincérre a „the waiter" vagy a „he" kifejezésekkel.

8. A nyelvtan indukciós tanulása

A **nyelvtan indukciós tanulása** (**grammar induction**) annak feladata, hogy adatokból nyelvtant tanuljunk. Nyilvánvaló, hogy meg kell kísérelni, hiszen bebizonyosodott, hogy nagyon nehéz egy nyelvtant kézzel elkészíteni, és az interneten mintaként használható kijelentések milliárdjai állnak ingyenesen rendelkezésre. Ez egy nehéz feladat, mivel a lehetséges nyelvtanok tere végtelen, és azért is, mivel annak ellenőrzése, hogy egy adott nyelvtan generálja-e a mondatok egy adott halmazát, számításigényes feladat.

Egy érdekes modell a `SEQUITUR`-rendszer (Nevill-Manning és Witten, 1997). Nincs szüksége más bemenetre, csak egy szövegre (amit nem szükséges előzetesen mondatokra bontani). Egy nagyon specializált formátumú nyelvtant állít elő: egy olyan nyelvtant, amely csak egy karakterfüzért generál, nevezetesen az eredeti szöveget. Másképp tekintve a `SEQUITUR` épp csak arra elegendő nyelvtant tanul, hogy a szöveget elemezze. Íme, itt van az a zárójelezés, amit egy nagyobb, híreket tartalmazó szövegben levő mondatra felfedezett:

```
[Most Labour] [sentiment [[would still] [favor the] abolition]] [[of [the House]] [of Lords]]
```

Helyesen választott ki olyan összetevőket, mint például az „of the House of Lords" *PP*, bár a tradicionális elemzéssel ellentétesen dolgozik akkor például, amikor a „the"-t a megelőző igével és nem a követő főnévvel csoportosítja.

A `SEQUITUR` azon az ötleten alapszik, hogy egy jó nyelvtan egyben tömör is. A következő két kényszer betartása a legfontosabb: (1) Egymást követő szimbólumok egyetlen párja sem szerepelhet egynél többször a nyelvtanban. Ha az *A B* szimbólumpár több szabály jobb oldalán is szerepel, akkor ki kell cserélnünk a párt egy új, nem záró szimbólummal, amit *C*-nek nevezünk, és egy új szabályt kell bevezetnünk, miszerint $C \rightarrow A\ B$. (2)

Created by XMLmind XSL-FO Converter.

Minden szabályt legalább kétszer kell használni. Ha egy *C* nem záró, csak egyszer szerepel a nyelvtanban, akkor törölnünk kell a *C*-re vonatkozó szabályt, és fel kell cserélnünk egyetlen előfordulását a szabály jobb oldalával. Ezt a két kényszert egy mohó keresés alkalmazza, amely a beérkező szöveget balról jobbra végignézi, menet közben inkrementálisan építi a nyelvtant, és alkalmazza a kényszereket, amint lehetséges. A 22.22. ábra mutatja az algoritmus működését az „*abcdbcabcd*” szövegen. Az algoritmus egy optimálisan tömör nyelvtant határoz meg a szöveghez.

A következő fejezetben további nyelvtant tanuló algoritmusokat is látni fogunk, amelyek valószínűségi alapú, környezetfüggetlen nyelvtanokkal működnek. Most azonban egy olyan nyelvtan tanulásának problémáját vizsgáljuk meg, amelyet szemantikával terjesztettünk ki. Mivel egy kiterjesztett nyelvtan egyben Horn-klóz logikai program is, ezért az induktív logikai programozás technikái megfelelők lesznek. A CHILL (Zelle és Mooney, 1996) egy induktív logikai programozás (ILP), amely példákból megtanul egy nyelvtant és egy arra specializált elemzőt. A célterület természetes nyelvű adatbázis-lekérdezések. A tanító példák szófüzérek, és a nekik megfelelő lekérdezések párjait tartalmazzák – például:

```
What is the capital of the state with the largest population?

Answer(c, Capital(s, c) ∧ Largest(p, State(s) ∧ Population(s, p)))
```

A CHILL feladata, hogy megtanuljon egy *Parse*(*words*, *query*) predikátumot, amely konzisztens a példákkal, és remélhetőleg más példákra is jól általánosít. Az ILP közvetlen alkalmazása ezen predikátum megtanulására gyenge teljesítményt hoz: a kikövetkeztetett elemző csak körülbelül 20%-os pontosságú. Szerencsére az ILP tanuló algoritmusok fejleszthetők tudás bevitelével. Ebben az esetben a *Parse* predikátum nagy részét logikai programként definiálták, és a CHILL feladatát a vezérlő szabályok kikövetkeztetésére redukálták, amelyek segítik az elemzőt a különböző elemzések közötti választásban. Ezzel a hozzáadott háttértudással a CHILL 70–85% közötti pontosságot ér el különböző adatbázis-lekérdezési feladatokon.

22.22. ábra - Az „*abcdbcabcd*” bemeneti szöveget elemző nyelvtant tartalmazó Sequitur-rendszer futási lépései. Egy *S*-re vonatkozó szabályból indulunk ki, és ezen szabály végéhez sorban minden egyes szimbólumot hozzáadunk. Miután a hatodik szimbólumot is hozzáadtuk, megkapjuk egy ismétlődő pár első előfordulását: *bc*. Ezért a *bc* mindkét előfordulását lecseréljük egy új nem záróra, *A*-ra, és egy új szabályt veszünk fel: $A \rightarrow bc$. Miután három újabb szimbólumot hozzáraktunk a bemenethez, a kilencedik a *bc* egy újabb ismétlését eredményezi, így újra *A*-val helyettesítjük. Ez az *aA* két előfordulásához vezet, ezért egy új nem záróval, *B*-vel helyettesítjük. Miután a tizedik, utolsó szimbólumot is hozzáraktuk, *Bd* két előfordulását kapjuk, ezért lecseréljük őket egy új nem záróval, *C*-vel. De *B* immár csak egyszer szerepel, a *C* szabály jobb oldalán, ezért *B*-t a neki megfelelő bővítéssel, *aA*-val cseréljük le.

Created by XMLmind XSL-FO Converter.

	Bemenet	Nyelvtan	Megjegyzések
1	*a*	$S \rightarrow a$	
2	*ab*	$S \rightarrow ab$	
3	*abc*	$S \rightarrow abc$	
4	*abcd*	$S \rightarrow abcd$	
5	*abcdb*	$S \rightarrow abcdb$	
6	*abcdbc*	$S \rightarrow abcdbc$	*bc* kétszer
		$S \rightarrow aAdA;\ A \rightarrow bc$	
7	*abcdbca*	$S \rightarrow aAdAa;\ A \rightarrow bc$	
8	*abcdbcab*	$S \rightarrow aAdAab;\ A \rightarrow bc$	
9	*abcdbcabc*	$S \rightarrow aAdAabc;\ A \rightarrow bc$	*bc* kétszer
		$S \rightarrow aAdAaA;\ A \rightarrow bc$	*aA* kétszer
		$S \rightarrow BdAB;\ A \rightarrow bc;\ B \rightarrow aA$	
10	*abcdbcabcd*	$S \rightarrow BdABd;\ A \rightarrow bc;\ B \rightarrow aA$	*Bd* kétszer
		$S \rightarrow CAC;\ A \rightarrow bc;\ B \rightarrow aA;\ C \rightarrow Bd$	*B* csak egyszer
		$S \rightarrow CAC;\ A \rightarrow bc;\ C \rightarrow aAd$	

9. Összefoglalás

A természetes nyelv megértése az MI egyik legfontosabb részterülete. Ötleteket merít a filozófiából és a nyelvészetből, valamint a logikai programozás, valószínűségi tudásreprezentáció és a következtetés területének technikáiból is. Más MI-területektől eltérő módon a természetes nyelv megértése a tényleges emberi viselkedés empirikus vizsgálatát igényli – amely komplexnek és érdekesnek bizonyul.

- Az ágensek jelzéseket küldenek egymásnak bizonyos célok elérése érdekében: informálni, figyelmeztetni, segítséget szerezni, tudást megosztani vagy megígérni valamit. A jelzések ily módon való küldését **szólásaktus**nak (**speech act**) nevezzük. Végső soron minden szólásaktus kísérlet arra, hogy egy másik ágens elhiggyen vagy megtegyen valamit.

- A nyelv megállapodáson alapuló **jelzés**ekből (**sign**s) áll, melyek jelentést közvetítenek. Sok állat használ ilyen értelemben jelzéseket. Az ember tűnik az egyetlen állatnak, aki **nyelvtan**t (**grammar**) használ strukturált üzenetek végtelen számú variációinak előállítására.

- A kommunikáció három lépést igényel a beszélőtől, a szándékot egy elképzelés átadására, a szavak mentális előállítását és a szavak fizikai szintézisét. Ezek után a hallgatónak négy lépést kell végrehajtania: észlelés, analízis, többértelműség feloldása és a jelentés beépítése. Minden nyelvhasználat **beágyazott** (**situated**) abban az értelemben, hogy a megnyilatkozások jelentése függhet attól a szituációból, amiben előállnak.

- A formális nyelvészet és a **kifejezésstruktúra** (**phrase structure**) nyelvtanok (különösen a **környezetfüggetlen nyelvtan** (**context-free grammar**)) hasznos eszközök a természetes nyelv egyes aspektusainak kezelésére.

- Környezetfüggetlen nyelv mondatai $O(n^3)$ időben elemezhetők a **diagramelemző** (**chart parser**) segítségével.

- Kézenfekvő egy nyelvtan **kiterjesztés**e (**augment**) annak érdekében, hogy olyan problémákat kezeljünk, mint az alany–ige egyeztetése vagy a névmási esetek. A **definit klóz nyelvtan** (**definite clause grammar**, **DCG**) egy olyan formalizmus, amely lehetővé teszi a kiterjesztéseket. A DCG segítségével az elemzés és szemantikai értelmezés (sőt még a generálás is) logikai következtetés segítségével elvégezhető.

- A **szemantikus értelmezés** (**semantic interpretation**) szintén kezelhető egy kiterjesztett nyelvtannal. Egy kvázilogikai forma jó közvetítő lehet a szintaktikai fák és a szemantika között.

Created by XMLmind XSL-FO Converter.

- A **többértelműség** (**ambiguity**) nagyon fontos probléma a természetes nyelv megértésében; a mondatok többségének több lehetséges értelmezése van, de általában csak egy megfelelő. A többértelműség feloldása a világról, az adott szituációról és a használt nyelvről szóló tudáson alapszik.

- A legtöbb nyelv több mondat kontextusában létezik, nem csak egyben. A **szövegértés** (**discourse**) a kapcsolódó szövegek vizsgálata. Láttuk, hogyan oldjunk fel mondatok közötti névmási utalásokat, és a mondatok hogyan kapcsolódnak koherens szegmensekké.

- A **nyelvtan indukciós** (**grammar induction**) **tanulása** példákból tud nyelvtant tanulni, bár vannak határai annak, hogy a nyelvtan milyen jól általánosít.

9.1. Irodalmi és történeti megjegyzések

A jelek és szimbólumok mint nyelvi elemek tanulmányozását John Locke (1690) **jeltudomány**nak (**semiotics**) nevezte, bár a 20. századig nem művelték (Peirce, 1902; De Saussure, 1993). A legutóbbi áttekintő munkák között van Eco (Eco, 1979) és Cobley (Cobley, 1997) műve.

A nyelv mint cselekvés ötlete a 20. századi nyelvészeti beállítottságú filozófiából ered (Wittgenstein, 1953; Grice, 1957; Austin, 1962), leginkább a *Speech Acts* c. könyvből (Searle, 1969). A szólásaktus ötletének előfutára Protagorasz négyféle mondatkategóriája volt: ima, kérdés, válasz és ítélet (kb. i. e. 430-ból). A szólásaktusok tervalapú modelljét először Cohen és Perrault (Cohen és Perrault, 1979) javasolta. A nyelv cselekvéshez kapcsolását tervfelismerés segítségével történetek megértéséhez Wilensky tanulmányozta (Wilensky, 1983). Cohen, Morgan és Pollack (Cohen, Morgan és Pollack, 1990) gyűjt össze újabb munkákat e területen.

A szemantikus hálókhoz hasonlóan a környezetfüggetlen nyelvtanok (amelyek kifejezésstruktúra nyelvtanokként is ismertek) a sásztra szanszkrit nyelvet tanulmányozó ősi indiai nyelvészek (különösen Pánini, kb. i. e. 350) által használt technika feleleventései (Ingerman, 1967). A modern időkben Noam Chomsky fedezte fel újra őket az angol szintaxis elemzésére (Chomsky, 1956), és tőle függetlenül John Backus az Algol-58 szintaxis analízisére használta. Naur kiterjesztette Backus jelölésrendszerét (Naur, 1963), ma pedig őhozzá rendelik az „N” betűt a BNF-ben, ami eredetileg a **Backus Normál Forma** volt (**Backus Normal Form**) (Backus, 1996). Knuth egyfajta kiterjesztett nyelvtant definiált, amelyet **attribútumnyelvtan**nak (**attribute grammar**) nevezünk, és programozási nyelvek esetén hasznos (Knuth, 1968). A definit klóz nyelvtanokat Colmerauer vezette be (Colmerauer, 1975), majd Pereira és Warren fejlesztette tovább és népszerűsítette (Pereira és Warren, 1980). A Prolog programozási nyelvet Alain Colmerauer találta ki, elsősorban a francia nyelv elemzésének problémájára. Colmerauer voltaképpen kidolgozott egy metamorfózis-nyelvtannak (metamorphosis grammar) nevezett formalizmust, amely továbbment a definit klózoknál, de a DCG hamarosan követte.

Számos kísérlet volt természetes nyelvek formális nyelvtanának leírására, mind a „tiszta”, mind a számítógépes nyelvészetben. A gépek számára készült nyelvtanok között van a New York University Linguistic String projektje (Sager, 1981) és a University of Pennsylvania XTAG projektje (Doran és társai, 1994). A modern DCG-rendszerek jó példája a Core Language Engine (Alshawi, 1992). Több átfogó, de nem formális nyelvtan létezik az angol nyelvről (Jespersen, 1965; Quirk és társai, 1985; McCawley, 1998; Huddleston és Pullum, 2002). A nyelvészetről szóló jó jegyzetek között van a (Sag és Wasow, 1999) bevezetése a szintaxisba, valamint a (Chierchia és McConnell-Ginet, 1990; Heim és Kratzer, 1998) szemantikáról szóló jegyzetek. McCawley nyelvészek számára szóló anyaga a logikára koncentrál (McCawley, 1993).

A nyolcvanas évek közepétől létező trend szerint egyre több információt tesznek a szókincsbe és kevesebbet a nyelvtanba. A lexikai-funkcionális nyelvtan (lexical-functional grammar), avagy LFG (Bresnan, 1982) volt az első jelentős nyelvtani formalizmus, amihez nagy szókincset gyűjtöttek. Ha a szélsőségig fokozzuk a lexikalizálást, akkor a **kategorikus nyelvtan**t (**categorial grammar**) kapjuk, amelyben lehet, hogy csak két nyelvtani szabály van, vagy a **függőségi nyelvtan**t (**dependency grammar**) (Melc´uk és Polguere, 1988), amelyben nincsenek kifejezések/frázisok, csak szavak. Sleator és Temperley leír egy függőségi nyelvtant használó népszerű elemzőt (Sleator és Temperley, 1993). A TAG, avagy Tree-Adjoining Grammar (Joshi, 1985) nem szigorúan lexikális, de nő a népszerűsége lexikalizált alakban is (Schabes és társai, 1988). A Wordnet (Fellbaum, 2000) egy körülbelül 100 000 szót és kifejezést tartalmazó szabadon elérhető szótár, melyet a beszéd részei szerint kategorizáltak, és olyan szemantikai relációkkal egészítettek ki, mint például a szinonima, antonima és része.

Created by XMLmind XSL-FO Converter.

Az első gépesített elemző algoritmusokat Yngve mutatta be (Yngve, 1955). Hatékony algoritmusokat a hatvanas évek végén fejlesztettek ki, néhány kisebb újítással azóta (Kasami, 1965; Younger, 1967; Graham és társai, 1980). A diagramelemzőnk Early rendszeréhez áll a legközelebb (Early, 1970). Egy jó összefoglaló Aho és Ullman elemzésről és fordításról szóló tanulmányában található (Aho és Ullman, 1972). Maxwell és Kaplan mutatja meg, hogy a kiterjesztett diagramelemző algoritmus hogyan tehető hatékonnyá az átlagos esetekben (Maxwell és Kaplan, 1993). Church és Patil foglalkozik a szintaktikai többértelműség feloldásával (Church és Patil, 1982).

A természetes nyelvek formális szemantikai értelmezése a filozófiából és a formális logikából ered, és különösen szorosan kapcsolódik Alfred Tarskinak a formális nyelvek szemantikájáról szóló munkájához (Tarski, 1935). Bar-Hillel volt az első, aki a pragmatikai elemzés problémáját áttekintette, és azt javasolta, hogy formális logikával kezeljék. Például bevezette C. S. Pierce *referenciális index (indexical)* fogalmát a nyelvészetben (Bar-Hillel, 1954). Richard Montague esszéje, az „English as a formal language" a nyelv logikai analízisének egyfajta manifesztuma (Montague, 1970), azonban Dowty és társainak könyve (Dowty és társai, 1991), valamint Lewis cikke (Lewis, 1972) sokkal olvasmányosabb. Thomason szerkesztette Montague munkáinak teljes kollekcióját (Thomason, 1974). A mesterséges intelligenciában McAllester és Givan munkája folytatja a montague-i hagyományt sok új technikai részlet megvilágításával (McAllester és Givan, 1992).

Egy közbülső, avagy kvázilogikai forma ötlete a kvantorok érvényességi köréhez hasonló problémák kezelésére Woodsig nyúlik vissza (Woods, 1978), és sok jelenlegi rendszerben is megtalálható (Alshawi, 1992; Hwang és Schubert, 1993).

Az első valódi feladatot megoldó NLP-rendszer valószínűleg a `BASEBALL` kérdésválaszoló rendszer volt (Green és társai, 1961), amely baseball-statisztikák adatbázisával kapcsolatos kérdéseket kezelt. Nem sokkal utána következett Woods `LUNAR` rendszere, amely az Apolló program által visszahozott kövekkel kapcsolatos kérdéseket válaszolt meg (Woods, 1973). Robert Schank és hallgatói több programot is készítettek (Schank és Abelson, 1977; Wilensky, 1978; Schank és Riesbeck, 1981; Dyer, 1983), amelyek mindegyikének a nyelv megértése volt a feladata. A hangsúly azonban kevésbé volt magán a nyelven, sokkal inkább a reprezentáción és következtetésen. A problémák között volt a sztereotip szituációk reprezentálása (Cullingford, 1981), az emberi memóriaszervezés leírása (Rieger, 1976; Kolodner, 1983), valamint tervek és célok megértése (Wilensky, 1983).

A természetes nyelv generálását az ötvenes években, a gépi fordítás legelső napjaitól kezdve figyelembe vették, de a hetvenes évekig nem merült fel egynyelvű problémaként. Simmons és Slocum (Simmons és Slocum, 1972), valamint Goldman (Goldman, 1975) munkája reprezentatív. A `PENMAN` (Bateman és társai, 1989) volt az első teljes generáló rendszer, amely a Szisztematikus Nyelvtanra (Systematic Grammar) (Kasper, 1988) épült. A kilencvenes években két fontos szabadon elérhető generáló rendszer vált elérhetővé, a KPML (Bateman, 1997) és a FUF (Elhadad, 1993). A generálásról szóló fontos könyvek között szerepel (McKeown, 1985; Hovy, 1988; Patten, 1988; Reiter és Dale, 2000).

A többértelműség feloldásával foglalkozó egyik legkorábbi munka Wilks elmélete a **preferenciaszemantikáról** (**preference semantics**) (Wilks, 1975), amelyik azokat az értelmezéseket próbálta megtalálni, amik minimalizálják a szemantikai anomáliák számát. Hirst egy olyan hasonló célú rendszert ír le, amelyik közelebb van a fejezetben ismertetett kompozíciós szemantikához (Hirst, 1987). Hobbs és társai a szintaktikai és szemantikai reprezentáció minőségét mérő kvantitatív keretrendszert írnak le (Hobbs és társai, 1993). Azóta a Bayes-hálók használata vált elterjedtebbé (Charniak és Goldman, 1992; Wu, 1993). A nyelvészetben az optimalitás elmélete (Kager, 1999) a puha kényszerek nyelvtanba építésén alapszik, amely természetes súlyt ad az értelmezéseknek ahelyett, hogy a nyelvtan az összes lehetőséget egyforma súllyal generálná. Norvig tárgyalja azokat a problémákat, amelyek egy maximálisan valószínű értelmezés helyett több párhuzamos értelmezés figyelembevételéből adódnak (Norvig, 1988). Az irodalmi kritika nem egyértelmű a tekintetben, hogy a többértelműség megoldandó probléma vagy üdvözlendő dolog (Empson, 1953; Hobbs, 1990).

Nunberg a metonímia (metonymy) formális modelljét vázolja (Nunberg, 1979). Lakoff és Johnson az angol nyelv gyakori metaforáit katalogizálja és magával ragadó módon elemzi (Lakoff és Johnson, 1980). Ortony metaforáról szóló cikkgyűjteményt mutat be (Ortony, 1979); Martin a metafora értelmezésének számítógépes megközelítését adja meg (Martin, 1990).

Az utalásfeloldás általunk bemutatott kezelése a (Hobbs 1978)-at követi. A (Lappin és Leass, 1994) által bemutatott összetettebb megoldás egy kvantitatív pontozási módszeren alapszik. Újabb munkák (Kehler, 1997; Ge és társai, 1998) gépi tanulást használnak a kvantitatív paraméterek hangolására. Az utalásfeloldásról szóló két kitűnő áttekintés Hirst és Mitkov könyvei (Hirst, 1981; Mitkov, 2002).

Created by XMLmind XSL-FO Converter.

1758-ban David Hume *Enquiry Concerning the Human Understanding* c. műve amellett érvelt, hogy a nyelvi szövegeket „három, az elgondolások közötti kapcsolatokat leíró elv köti/fogja össze, nevezetesen a *Hasonlóság*, az idő- vagy térbeli *Összefüggés* és az *Ok* vagy *Okozat*". Így kezdődött a koherenciarelációk meghatározásának hosszú története. Hobbs (Hobbs, 1990) adja meg nekünk a fejezetben használt halmazt; Mann és Thompson (Mann és Thompson, 1983) egy jobban kifejtett halmazt nyújt, amely magában foglalja a következőket: megoldás-összetartozás, bizonyíték, igazolás, motiváció, következtetés, sorozat, engedélyezés, kidolgozás, újrafogalmazás, feltétel, körülmény, ok, engedmény, háttér, valamint tézis-antitézis. Ez a modell fejlődött a retorikai struktúra elméletté (rhetorical structure theory, RST), amely valószínűleg napjaink legkiemelkedőbb elmélete (Mann és Thompson, 1988). Ez a fejezet átvesz egyes példákat (Jurafsky és Martin, 2000) Andrew Kehler által írt fejezetéből.

Grosz és Sidner bemutat egy fókuszeltoláson alapuló szövegkoherencia-elméletet (Grosz és Sidner, 1986), és a (Grosz és társai, 1995) egy középpontba állításon alapuló hasonló elméletet kínál. Joshi, Webber és Sag fontos korai munkákat gyűjt össze a szövegértésről (Joshi, Webber és Sag, 1981). Webber bemutatja olyan szintaxis- és szövegkényszerek egymással kapcsolatban álló modelljét, amely kényszerek a szöveg egy adott pontján állítható dolgokra vonatkoznak (Webber, 1983), továbbá bemutatja az igeidők és a szöveg kölcsönhatási módjának modelljét (Webber, 1988).

Az első fontos eredmény a **nyelvtan indukciós** (**grammar induction**) **tanulásá**nak területén egy negatív eredmény volt: Gold (Gold, 1967) megmutatta, hogy nem lehetséges megbízhatóan egy helyes környezetfüggetlen nyelvtan megtanulása annak szövegei alapján. Az ötlet lényege szerint ha adott egy s_1, s_2, ... s_n füzérhalmaz, akkor a helyes nyelvtan lehet a mindent magába foglaló ($S \rightarrow word^*$), vagy a bemenet másolata, az ($S \rightarrow s_1 \mid s_2 \mid ... \mid s_n$), vagy bármi a kettő között. Jeles nyelvészek, mint például Chomsky (Chomsky, 1957; 1980) és Pinker (Pinker, 1989; 2000) felhasználták Gold eredményeit annak megmutatására, hogy lennie kell egy öröklött **univerzális nyelvtan**nak (**universal grammar**), amellyel minden gyermek születésénél fogva rendelkezik. Az úgynevezett Ingerszegénység *(Poverty of the Stimulus)* érvelés szerint a gyerekeknek nincs más nyelvi példájuk, mint pozitív: szüleik és társaik nyelvük legtöbbször pontos példáit állítják elő, és igen ritkán javítanak ki hibákat. Ezért, mivel Gold bebizonyította, hogy pozitív példákból nem lehet CFG-t tanulni, a gyerekeknek már „tudniuk" kell a nyelvtant, és pusztán ezen öröklött nyelvtan paramétereit állítják, illetve a szókészletet tanulják. Bár ez az érvelés még befolyással bír a Chomskyt követő nyelvészek körében, egyes nyelvészek (Pullum, 1996; Elman és társai, 1997) és a legtöbb számítástudománnyal foglalkozó elvetette. Igen hamar, már 1969-ben Horning megmutatta, hogy *meg lehet tanulni* PAC értelemben egy *valószínűségi* környezetfüggetlen nyelvtant. Azóta számos meggyőző, kizárólag pozitív példákból tanuló empirikus demonstráció jelent meg, mint például Mooney, illetve Muggleton és De Raedt ILP-munkái (Mooney, 1999; Muggleton és De Raedt, 1994), valamint Schütze és de Marcken emlékezetes PhD-értekezései (Schütze, 1995; de Marcken, 1996). Más nyelvtani formalizmusok, mint például a reguláris nyelvek (Oncina és Garcia, 1992; Denis, 2001), reguláris fa nyelvek (Carrasco és társai, 1998), valamint a véges automaták (Parekh és Honavar, 2001) tanulása is lehetséges.

A SEQUITUR-rendszert Nevill-Manning és Witten készítette (Nevill-Manning és Witten, 1997). Érdekes, hogy ők, valamint de Marcken is megjegyezte, hogy nyelvtant kikövetkeztető megoldásaik tömörítésre is jól használhatók. Ez összhangban van a minimális leíró hosszúság kódolásának elvével: egy jó nyelvtan olyan, amely minimalizál két méretet: a nyelvtan és a szövegek elemzési fájának nagyságát.

A nyelv tanulására szolgáló induktív logikai programozási munkák között van a CHILL rendszer (Zelle és Mooney, 1996), valamint Mooney és Califf programja (Mooney és Califf, 1995), amely jobban tanult igék múlt idejű alakjára vonatkozó szabályokat, mint a korábbi neurális hálók és a döntési fa rendszerek. Cussens és Džeroski több cikkgyűjteményt is szerkesztett, amelyek a nyelvtanulásról szóltak a logikában (Cussens és Dzeroski, 2000).

Az Association for Computational Linguistics (ACL) rendszeresen rendez konferenciákat, és kiadja a *Computational Linguistics* folyóiratot. Ezenkívül van egy nemzetközi konferencia a témában, az International Conference on Computational Linguistics (COLING). A *Readings in Natural Language Processing* (Grosz és társai, 1986) a terület sok fontos korai cikkét tartalmazó antológia. A (Dale és társai, 2000) az NLP-rendszerek építésére szolgáló gyakorlati eszközökre helyezi a hangsúlyt. Jurafsky és Martin jegyzete alapos bevezetőt nyújt a területhez (Jurafsky és Martin, 2000). Az (Allen, 1995) egy kicsit régebbi munka. Pereira és Sheiber, valamint Covington Prolog-implementációkon alapuló tömör áttekintést ad (Pereira és Sheiber, 1987; Covington, 1994). Az *Encyclopedia of AI*-ben sok hasznos cikk található a területről; kiemelendő a „Computational Linguistics" és a „Natural Language Understanding".

Created by XMLmind XSL-FO Converter.

9.2. Feladatok

22.1.

Olvassa el az alábbi szöveget egyszer, és próbáljon meg amennyit csak lehet megjegyezni belőle. Később lesz egy teszt.

Az eljárás valójában igen egyszerű. Először különböző csoportokba rendezi a dolgokat. Természetesen egy halom elegendő lehet attól függően, hogy mennyi teendő van. Ha valahova máshova kell mennie a felszerelés hiánya miatt, akkor az a következő lépés, különben már nagyjából előkészült. Fontos, hogy ne vigye túlzásba a dolgot. Ez azt jelenti, hogy jobb egyszerre kevesebbel foglalkozni, mint többel. Rövid távon ez nem tűnhet fontosnak, de könnyen jelentkezhetnek komplikációk. A hiba is költséges. Első ránézésre a teljes eljárás bonyolultnak tűnik. Azonban hamarosan az élet megszokott dolgává válik. Nehéz a feladat szükségességének bármilyen elmúlását is előre látni, de sosem lehet tudni. Miután az eljárás befejeződött, ismét különböző csoportokba kell rendezni az anyagokat. Ezek után a megfelelő helyükre pakolhatók a dolgok. Végső soron újra használni fogják őket, és a teljes ciklust meg kell majd ismételni. Mindazonáltal, ez az élet velejárója.

22.2.

Írjon egy nyelvtant a DCG-jelölésrendszerrel, amely ugyanolyan, mint az $\mathcal{E}_1$, azzal a kivétellel, hogy kikényszeríti a mondat alanya és igéje közötti egyeztetést, így nem állítja elő az „I smells the wumpus”-t!

22.3.

Terjessze ki az $\mathcal{E}_1$ nyelvtant úgy, hogy kezelje a névelő-főnév egyeztetéseket! Azaz biztosítsa, hogy az „agents” *NP*, de az „agent” és az „an agents” nem az!

22.4.

Foglalja össze a legfontosabb különbségeket a Java (vagy bármely más számítógépes nyelv, amelyet ismer) és a magyar között, megjegyezve a „megértés” problémáját mindkettőre! Gondoljon olyan dolgokra, mint a nyelvtan, a szintaxis, a szemantika, a pragmatikus elemzés, az összetételek, a környezetfüggőség, a lexikális többértelműség, a szintaktikai többértelműség, az utalások megtalálása (a névmásokat is ideértve), a háttértudás és leginkább arra, hogy mit is jelent „megérteni”.

22.5.

Melyek indokolják a következők közül a kvázilogikai forma bevezetését?

a. Könnyebben leírható egyszerű összetételes nyelvtani szabályok.

b. A szemantikai leíró nyelv kifejezőképességének kiterjesztése.

c. A kvantorok érvényességi köre többértelműségének (többek között) tömör formában történő ábrázolása.

d. A szemantikai egyértelműsítés egyszerűbbé tétele.

22.6.

Határozza meg, hogy az ebben a fejezetben található nyelvtan alapján milyen szemantikai értelmezés adható a következő mondatokhoz:

a. It is a wumpus.

b. The wumpus is dead.

c. The wumpus is in 2,2.

Jó ötlet lenne az „It is a wumpus” szemantikai értelmezésének egyszerűen a $\exists x \; x \in \mathit{Wumpuses}$? Fontoljon meg alternatív mondatokat, mint például az „It was a wumpus”.

22.7.

Created by XMLmind XSL-FO Converter.

Anélkül hogy megnézné a 22.1. feladatot, válaszolja meg a következő kérdéseket:

a. Mi a négy említett lépés?

b. Melyik lépés maradt ki?

c. Mi az „az anyag”, amit a szöveg említ?

d. Milyen hiba lenne drága?

e. Jobb túl keveset, vagy túl sokat tenni? Miért?

22.8.

Ez a gyakorlat nagyon egyszerű nyelvek nyelvtanaival foglalkozik.

a. Írjon egy környezetfüggetlen nyelvtant az $a^n b^n$ nyelv számára!

b. Írjon egy környezetfüggetlen nyelvtant a tükörmondatok (palindroma) számára: az összes olyan füzérre, amelyek második fele épp az első fél tükörképe!

c. Írjon egy környezetfüggő nyelvtant a duplázó nyelv számára: az összes olyan füzérre, amelyek második fele megegyezik az első felével.

22.9.

Vizsgálja meg a „Someone walked slowly to the supermarket (Valaki lassan a szupermarkethez sétált)” mondatot és az alábbi szókincset:

```
Pronoun → someone      V → walked

Adv → slowly           Prep → to

Det → the              Noun → supermarket
```

A következő átíró szabályhalmazok melyike állítja elő a fenti mondatot az előbbi szókincset használva? Adja meg a megfelelő elemzési fá(ka)t!

```
(A):                   (B):                   (C):

S → NP VP              S → NP VP              S → NP VP

NP → Pronoun           NP → Pronoun           NP → Pronoun

NP → Det Noun          NP → Noun              NP → Det NP

VP → VP PP             NP → Det NP            VP → V Adv

VP → VP Adv Adv        VP → V Vmod            Adv → Adv Adv

VP → V                 Vmod → Adv Vmod        Adv → PP

PP → Prep NP           Vmod → Adv             PP → Prep NP
```


Created by XMLmind XSL-FO Converter.

```
NP → Noun     Adv → PP      NP → Noun

              PP → Prep NP
```

Írjon három szintaktikailag helyes és három helytelen angol mondatot, amelyet a három előbbi nyelvtan állít elő! Lényegében különbözzenek egymástól, legalább hat szó hosszúak legyenek, és egy teljesen új szókincsre kell épülniük (amit lehetőleg Ön határozzon meg). Javasoljon mindhárom nyelvtanra olyan újítást, amely elkerüli a nem helyes mondatok előállítását!

22.10.

Valósítsa meg a diagramelemző algoritmus egy olyan változatát, amely a teljes bemenetet lefedő összes él tömörített fáját adja vissza!

22.11.

Valósítsa meg a diagramelemző algoritmus egy olyan változatát, amely a leghosszabb bal oldali él tömörített fáját adja vissza, és amennyiben ez az él nem fedi le a teljes fát, folytatja az elemzést annak az élnek a végén! Mutassa meg, hogy miért lesz szükség a `PREDICT` eljárás meghívására a folytatás előtt! A végeredmény tömörített fák egy olyan listája, ahol a teljes lista lefedi a bemenetet.

22.12.

(Barton és társai, 1987) alapján. Ez a gyakorlat az általunk *Buffalo*^{n}-nek nevezett nyelvvel foglalkozik, ami nagyon hasonlatos az angolhoz (legalábbis az $\mathcal{E}_0$-hoz), azzal a kivétellel, hogy a szókincsben csak egy szó található: a *buffalo*. Íme, két mondat a nyelvből:

Buffalo buffalo buffalo Buffalo buffalo.

Buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.

Arra az esetre, ha nem hinné el, hogy ezek mondatok, álljon itt két angol mondat ugyanezzel a szintaktikai struktúrával:

Dallas cattle bewilder Denver cattle.

Chefs London critics admire cook French food.

Írjon egy nyelvtant a *Buffalo*^{n} számára! A lexikális kategóriák a város, a többes számú főnév és a (tranzitív) ige, és egy szabálynak kell lennie a mondat, egynek az igei kifejezés és háromnak a következő főnévi kifejezések számára: többes számú főnév, főnévi kifejezés, amelyet a város mint módosító előz meg, és főnévi kifejezés, amelyet redukált relatív klóz követ. A redukált relatív klóz egy olyan klóz, amelyből hiányzik a relatív névmás. Emellett a klóz egy alanyi főnévi kifejezést követő tárgy nélküli igéből áll. Egy példa a redukált relatív klózra a „London critics admire” a fenti példában. Foglalja táblázatba a *Buffalo*^{n} lehetséges elemzéseinek számát n = 1…10-re! Külön pontért: Carl de Marcken kiszámolta, hogy 121 030 872 213 055 59 681 184 485 olyan *Buffalo*^{n} mondat van, amely 200 hosszú (az általa használt nyelvtanra). Hogyan csinálta?

22.13.

Rajzolja fel a 7.2. szakasz - Egy koherens szöveg struktúrája részben található „John egy elegáns étterembe megy” történet szövegelemzési fáját! Használja a *Segment*-re vonatkozó két szabályt, megadva a helyes *CoherenceRelation*-t mindent egyes csomópontra! (Nem kell az egyes mondatok elemzéseit megmutatnia.) Most tegye meg ugyanezt egy Ön által választott tetszőleges 5–10 mondat hosszúságú szövegre!

22.14.

Elfelejtettük megemlíteni, hogy a 22.1. feladat szövegének címe: „Ruhamosás”. Olvassa újra a szöveget, és válaszolja meg a 22.7. feladat kérdéseit újra! Ezúttal jobban ment? Bransford és Johnson (1973) ezt a szöveget használta egy jobban ellenőrzött kísérletben, és azt tapasztalta, hogy a cím sokat segített. Mit mond ez Önnek a szövegértésről?

Created by XMLmind XSL-FO Converter.

23. fejezet - Valószínűségi nyelvfeldolgozás

Ebben a fejezetben meglátjuk, hogyan lehet egyszerű, statisztikailag tanított nyelvi modelleket szavak millióinak feldolgozására használni ahelyett, hogy csak egyes mondatok feldolgozására használnánk

A 22. fejezetben láttuk, hogyan képes egy ágens kommunikálni egy másik (szoftver vagy emberi) ágenssel közös nyelvi megnyilatkozások segítségével. A megnyilatkozások teljes szintaktikai és szemantikai elemzése *szükséges* a jelentésük teljes kinyeréséhez, ami azért *lehetséges,* mert a megnyilatkozások rövidek és körülhatárolt tárgyterületre korlátozottak.

Ebben a fejezetben a nyelvmegértés **korpuszalapú** (**corpus-based**) megközelítését tárgyaljuk. A korpusz egy nagy szöveggyűjtemény, például az a több milliárd oldal, ami a világhálót építi fel. A szövegeket emberek írják embereknek, és a szoftverek feladata az, hogy megkönnyítsék az embereknek az információkeresést. Ez a megközelítés magában foglalja statisztikák és tanulás használatát a korpusz kihasználására, és jellemzően olyan valószínűségi nyelvi modelleket von maga után, amelyek az adatokból tanulhatók, és amelyek egyszerűbbek, mint a 22. fejezet kibővített DCG-i. A legtöbb feladat esetén az adatmennyiség kárpótol azért, hogy egyszerűbb nyelvi modellt használunk. Három specifikus feladatot fogunk áttekinteni: az **információkeresés**t (**information retrieval**) (23.2. alfejezet), az **információkinyerés**t (**information extraction**) (23.3. alfejezet) és a **gépi fordítás**t (**machine translation**) (23.4. alfejezet). Elsőként azonban a valószínűségi nyelvi modelleket (probabilistic language model) mutatjuk be röviden.

1. Valószínűségi nyelvi modellek

A 22. fejezet a nyelv *logikai* modelljét adta meg: CFG-ket és DCG-ket használtunk, hogy egy adott karakterfüzérről eldöntsük, eleme-e vagy sem egy nyelvnek. Ebben az alfejezetben számos *valószínűségi* modellt vezetünk be. A valószínűségi modellek számos előnnyel rendelkeznek. Kényelmesen taníthatók adatok alapján: a tanulás mindössze az előfordulások megszámlálásából áll (némi tűréssel a kis mintaméret okozta hibák miatt). Továbbá robusztusabbak (mivel *bármely* karakterfüzért elfogadnak, habár kis valószínűséggel), visszatükrözik azt a tényt, miszerint nem a beszélők 100%-a ért egyet abban, hogy mely mondatok részei a nyelvnek; valamint alkalmasak a többértelműség feloldására: a valószínűségre támaszkodva kiválasztható a legvalószínűbb értelmezés.

A **valószínűségi nyelvi modell** (**probabilistic language model**) valószínűségi eloszlást definiál egy (esetleg végtelen) karakterfüzér-halmaz felett. Példaként állhatnak a 15.6. alfejezetben beszédmegértésre használt bi- és trigram nyelvi modellek. Az unigram modell $P(w)$ valószínűséget rendel a szókincs minden egyes szavához. A modell feltételezi, hogy a szavak függetlenül lettek kiválasztva, azaz a karakterfüzér valószínűsége egyszerűen a szavak valószínűségének szorzata: $\prod_i P(w_i)$. A következő húszszavas szekvenciát véletlen módon generáltuk, a könyv szavainak unigram modellje alapján:[1]

logical are as confusing a may right tries agent goal the was diesel more object then information-gathering search is

A bigram modell egy $P(w_i|w_{i-1})$ valószínűséget rendel minden egyes szóhoz, adott előző szó esetén. A 15.21. ábra néhány bigram-valószínűséget mutatott be. A könyv bigram modellje a következő véletlen szekvenciát generálja:

planning purely diagnostic expert system are very similar computational approach would be represented compactly using tic tac toe a predicate

Általánosan, egy n-gram modell az előző $n - 1$ szó alapján szabja meg a $P(w_i|w_{i-(n-1)}\ldots w_{i-1})$ valószínűséget. A könyv trigram modellje ezt a véletlen szekvenciát generálja:

[1] Mivel nem lett volna értelme lefordítani magyarra az egyes szavakat, ezért meghagytuk az eredeti angol nyelvű példát. (*A ford.*)

Created by XMLmind XSL-FO Converter.

planning and scheduling are integrated the success of naive bayes model is just a possible prior source by the time

Még a fenti kis példa alapján is láthatónak kell lennie, hogy a trigram modell jobb, mint a bigram (amely pedig jobb, mint az unigram), mind az angol nyelv, mind egy MI-tankönyv témájának közelítésében. A modellek maguk is egyetértenek ezzel: a trigram modell a véletlen módon generált sztringjéhez 10^{-10} valószínűséget rendel, a bigram 10^{-29}-et, az unigram pedig 10^{-59}-et.

Ez a könyv félmillió szavával nem tartalmaz elég adatot ahhoz, hogy jó minőségű bigram modellt lehessen előállítani belőle, nem is beszélve a trigram modellről. A könyv szókincse körülbelül 15 ezer különböző szót tartalmaz, tehát a bigram modell $15\,000^2 = 225$ millió szópárt tartalmaz. Világos, hogy a szópárok legalább 99,8%-a 0 gyakoriságú, de nem akarjuk, hogy a modell azt állítsa, hogy ezek a szópárok lehetetlenek. Szükségünk van valamilyen **simítás**ra (**smoothing**) a nulla gyakoriságok felett. A legegyszerűbb megoldás az **adj-hozzá-egyet simítás** (**add-one smoothing**): minden lehetséges bigram gyakoriságához hozzáadunk egyet. Azaz amennyiben a korpuszban N szó és B lehetséges bigram található, akkor minden c gyakoriságú bigramhoz egy $(c + 1)/(N + B)$ értékű valószínűség-becslőt rendelünk. Ez a módszer megszünteti a nulla valószínűségű n-gramok problémáját, de az a követelmény, hogy minden gyakoriságot pontosan eggyel kell növelni, kétséges, és rossz becslésekhez vezethet.

A másik megközelítés a **lineáris interpoláción alapuló simítás** (**linear interpolation smoothing**), ami lineárisan kombinálja a trigam, bigram és unigram modelleket. A valószínűség-becslőt a következőképpen definiáljuk:

$$\hat{P}(w_i \mid w_{i-2}w_{i-1}) = c_3 P(w_i \mid w_{i-2}w_{i-1}) + c_2 P(w_i \mid w_{i-1}) + c_1 P(w_i)$$

ahol $c_3 + c_2 + c_1 = 1$. A c_i paraméterek lehetnek rögzítettek vagy EM algoritmussal taníthatók. Az is lehetséges, hogy a c_i értékeket az n-gram gyakoriságoktól függőnek választjuk, azaz nagyobb súlyt adunk azoknak a valószínűségi becslőknek, amelyeket nagyobb gyakoriságokból számítunk.

A nyelvi modell egyik lehetséges *kiértékelési* módja a következő: először válasszuk szét a korpuszt egy tanító és egy teszthalmazra. Határozzuk meg a modell paramétereit a tanító halmaz alapján. Ezután számítsuk ki a valószínűséget a teszthalmazra a modell alapján; minél nagyobb a valószínűség, annál jobb. A megközelítés egyik problémája az, hogy hosszú karakterfüzérek esetén a $P(szavak)$ nagyon kicsi; a kis számok lebegőpontos alulcsordulást okozhatnak, vagy egyszerűen túl nehéz lenne elolvasni őket. Tehát a valószínűség helyett a modell **összetettség**ét (**perplexity**) számíthatjuk ki a teszt karakterfüzéren:

$$Összetettség(szavak) = 2^{-\log_2(P(szavak))/N}$$

ahol N a *szavak* száma. Minél kisebb az összetettség, annál jobb a modell. Az az n-gram modell, amely minden szóhoz $1/k$ valószínűséget rendel, k összetettségű; az összetettséget úgy is lehet értelmezni, mint átlagos elágazási tényezőt.

Példaként arra, hogy mire képesek az n-gram modellek, vegyük a **szegmentáció** (**segmentation**) feladatát, ami szóhatárok megtalálása szóköz nélküli szövegben. Ez a feladat elengedhetetlen a japán és kínai nyelv esetén; ezek olyan nyelvek, amelyek nem raknak szóközt a szavak közé, feltételezzük azonban, hogy a legtöbb olvasónak az angol nyelv[2] jobban megfelel. Az

Itiseasytoreadwordswithoutspaces[3]

mondatatot tényleg könnyű elolvasni. Az olvasó arra gondolhat, hogy ezt az teszi lehetővé, hogy teljesen ismerjük az angol szintaktikát, szemantikát és pragmatikát. Meg fogjuk mutatni, hogy a mondatot egy egyszerű unigram modellel is könnyen dekódolni lehet.

Korábban láthattuk, hogy a Viterbi algoritmus (15.9) hogyan használható a legvalószínűbb szekvencia megtalálására egy szó-valószínűségi hálóban. A 23.1. ábrán látható a Viterbi algoritmus olyan változata, amelyet specifikusan a szegmentációs probléma megoldására terveztünk. Bemenete a $P(szó)$ unigram valószínűségi eloszlás és egy karakterfüzér. Ezután az algoritmus a karakterfüzér minden egyes i pozíciójára a

[2] Mivel nem lehetett volna hasonló magyar példát valószínűségekkel együtt elkészíteni, ezért meghagytuk az eredeti angol nyelvű példát. (*A ford.*)
[3] Könnyűszavakatszóközöknélkülolvasni (*A ford.*)

Created by XMLmind XSL-FO Converter.

legjobb[*i*] elemben eltárolja a legvalószínűbb *i*-ig tartó karakterfüzér valószínűségét. Emellett a *szavak*[*i*] elemben eltárolja azt az *i*-edik pozícióban végződő szót, ami a legnagyobb valószínűséget adta. Miután felépítette a *legjobb* és a *szavak* tömböket dinamikus programozási módon, hátrafelé mozogva feldolgozza a *szavak* tömböt, hogy megtalálja a legjobb utat. Ebben az esetben, a könyv unigram modellje alapján a legjobb szekvencia ténylegesen az „It is easy to read words without spaces”,[4] 10^{-25} valószínűséggel. A szekvencia részeinek összehasonlítása során látható, hogy az „easy” unigram valószínűsége $2{,}6 \times 10^{-4}$, miközben az alternatív „*e* as *y*”[5] valószínűsége sokkal kisebb, $9{,}8 \times 10^{-12}$ annak ellenére, hogy a könyv képleteiben viszonylag gyakran előfordul az „*e*” és az „*y*”. Hasonlóképpen:

$P(\text{„without”}) = 0{,}0004$

$P(\text{„with”}) = 0{,}005;\ P(\text{„out”}) = 0{,}0008$

$P(\text{„with out”}) = 0{,}005 \times 0{,}0008 = 0{,}000004$

Ennek következtében az unigram modell szerint a „without” százszor nagyobb valószínűségű, mint a „with out”.[6]

23.1. ábra - Viterbi-alapú szószegmentáló algoritmus. Egy szóközöket nem tartalmazó szófüzért feldolgozva megadja a legvalószínűbb szavakra történő szegmentációt.

```
function VITERBI-SZEGMENTÁCIÓ(szöveg, P) returns legjobb szavak és valószínűségük
  inputs: szöveg, szóköz nélküli karakterfüzér
          P, szavak unigram modellje

  n ← HOSSZ(szöveg)
  szavak ← n + 1 hosszúságú üres vektor
  legjobb ← n + 1 hosszúságú vektor, 0,0-ra inicializálva
  legjobb[0] ← 1,0
  /* Legjobb és szavak tömbök feltöltése dinamikus programozási módon */
  for i = 0 to n do
      for j = 0 to i – 1 do
        szó ← szöveg[j:i]
        w ← HOSSZ(szó)
        if P[szó] × legjobb[i – w] ≥ legjobb[i] then
            legjobb[i] ← P[szó] × legjobb[i – w]
            szavak[i] ← szó
  /* Legjobb szó szekvencia kigyűjtése */
  szekvencia ← üres lista
  i ← n
  while i > 0 do
      szavak[i] a szekvencia elé
      i ← i – HOSSZ(szavak[i])
  /* Legjobb szó szekvencia és valószínűségének visszaadása */
  return szekvencia, legjobb[i]
```

Ebben a bekezdésben a szavak feletti *n*-gram modellekről értekeztünk, azonban az *n*-gram modelleket számos egyéb egység – például karakterek vagy **beszédrészek** (**parts of speech**) – felett is lehet értelmezni.

1.1. Valószínűségi környezetfüggetlen nyelvtanok

Az *n*-gram modellek a korpuszon belüli közös előfordulási statisztikát használják ki, azonban nincs semmilyen információjuk a nyelvtanról *n*-nél nagyobb távolságra. A **valószínűségi környezetfüggetlen nyelvtan** – **PCFG**[7] (**probabilistic context-free grammar**) – egy alternatív nyelvi modell, ami egy olyan CFG, melyben minden

[4] Könnyű szavakat szóközök nélkül olvasni (*A ford.*)
[5] *easy* – könnyű; *as* – mint (*A ford.*)
[6] *without* – nélkül; *with* – vele; *out* – kint (*A ford.*)
[7] A PCFG-k másik megnevezése a sztochasztikus környezetfüggetlen nyelvtan (stochastic context-free grammar), avagy SCFG.

Created by XMLmind XSL-FO Converter.

átírási szabályhoz valószínűséget rendelünk. Az azonos bal oldallal rendelkező szabályok valószínűségének összege 1. A 23.2. ábrán az $\mathcal{E}_0$ nyelvtan egy részletének PCFG-je látható.

23.2. ábra - Az $\mathcal{E}_0$ nyelvtan egy részletének valószínűségi környezetfüggetlen nyelvtana (PCFG) és szókincse. A szögletes zárójelben levő szám jelzi a valószínűségét annak, hogy az adott bal oldali szimbólumot a megfelelő szabály szerint írjuk át

$S \rightarrow NP\ VP$ [1,00]

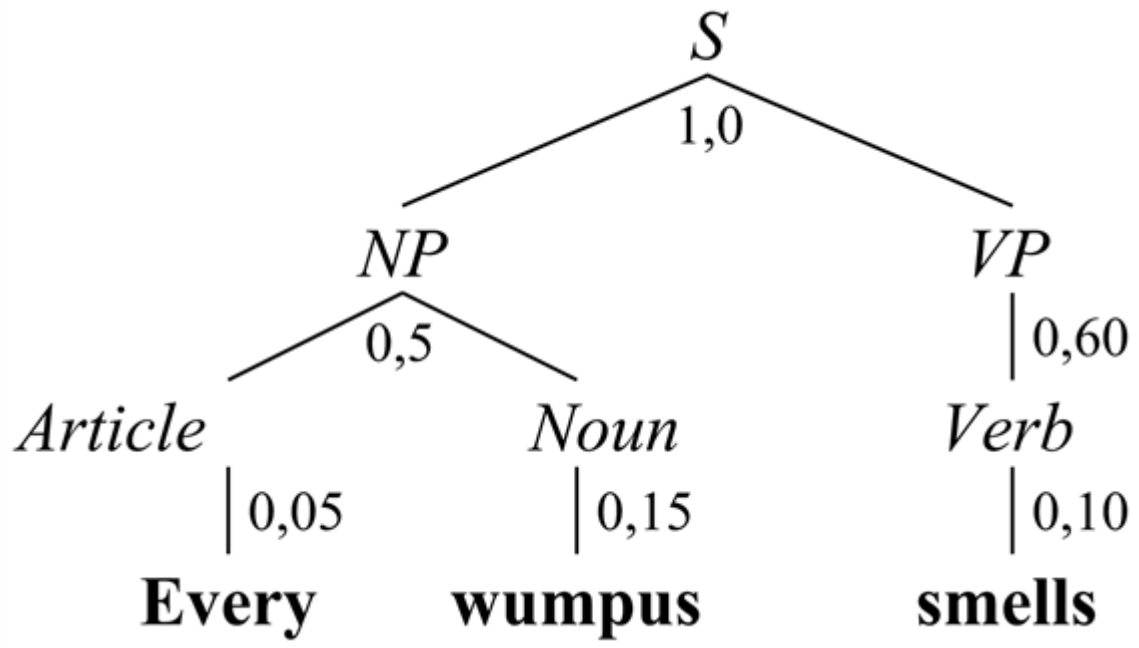
$NP \rightarrow Pronoun$ [0,10]
| *Neme*[0,10]
| *Noun* [0,20]
| *Article Noun* [0,50]
| *NP PP* [0,10]

$VP \rightarrow Ige$ [0,60]
| *VP NP* [0,20]
| *VP PP* [0,20]

$PP \rightarrow Preposition\ NP$ [1,00]

Noun → **breeze** [0,10] | **wumpus** [0,15] | **agent** [0,05]| …
Verb → **sees** [0,15] | **smells** [0,10] | **goes** [0,25]| …
Pronoun → **me** [0,05] | **you** [0,10] | **I** [0,25]|**it** [0,20]| …
Article → **the** [0,30] | **a** [0,35] | **every** [0,05]| …
Preposition → **to** [0,30] | **in** [0,10] | **on** [0,20]| …

23.3. ábra - A „Minden wumpus bűzlik” mondat elemzési fája, megadva minden egyes részfa valószínűségét. A teljes fa valószínűsége 1,0 × 0,5 × 0,05 × 0,15 × 0,60 × 0,10 = 0,000225. Mivel a mondatnak ez az egyetlen elemzése, ezért ennyi a mondat valószínűsége is.



A PCFG-modellben egy karakterfüzér valószínűsége – a *P*(*szavak*) – egyszerűen az elemzési fái valószínűségeinek összege. Egy adott fa valószínűsége a fa csomópontjait felépítő szabályok valószínűségeinek

Created by XMLmind XSL-FO Converter.

szorzata. A 23.3. ábra bemutatja, hogyan lehet kiszámítani egy mondat valószínűségét. A valószínűség kiszámítható egy CFG diagramelemző alkalmazásával, amely megadja az összes lehetséges elemzést, majd egyszerűen össze kell adni a valószínűségeket. Azonban ha csak a legvalószínűbb elemzés érdekel minket, akkor a nem valószínű elemzések meghatározása pazarlás. A legvalószínűbb elemzés hatékony megtalálására használhatunk egy Viterbi algoritmus variációt vagy egy legjobbat-először keresési technikát (mint például az A*-ot).

A PCFG-vel az a probléma, hogy környezetfüggetlen. Ez azt jelenti, hogy a különbség a *P*(„eat a banana") és a *P*(„eat a bandanna")[8] között mindössze a *P*(„banana") és *P*(„bandanna") közti különbségtől függ, és nem az „enni" ige és a megfelelő objektumok közti kapcsolattól. Hogy megkapjuk ezt a kapcsolatot, szükség van valamilyen környezetfüggő modellre, mint például a **szókinccsel ellátott PCFG**-re (**lexicalized PCFG**), amelyben a kifejezés feje[9] szerepet játszhat a kifejezés valószínűségének meghatározásában. Elegendő tanító adat esetén a *VP* → *VP NP* szabály kondicionálható a beágyazott *VP* fejére („enni") és az *NP* fejére („banán"). Ezáltal a szókinccsel ellátott PCFG-k képesek az *n*-gram modellek közös előfordulási megszorításai egy részének a megragadására, megtartva a CFG-modellek nyelvtani megszorításait.

A PCFG másik problémája az, hogy erősen preferálja a rövid mondatokat. Egy olyan korpuszban, mint a *Wall Street Journal*, az átlagos mondathossz körülbelül 25 szó. Azonban egy PCFG általában úgy végez, hogy viszonylag magas valószínűséget rendel olyan szabályokhoz, mint az *S* → *NP VP*, az *NP* → *Pronoun* és a *VP* → *Verb*. Ez azt jelenti, hogy a PCFG viszonylag nagy valószínűséget fog rendelni sok rövid mondathoz, mint pl. „ő aludt", azonban a *Journalban* sokkal valószínűbben találkozunk olyan mondatokkal, mint „Egy megbízható kormányzati forrás jelentése szerint az az állítás, hogy aludt, hihető". Úgy tűnik, hogy a *Journal* mondatai igazából nem környezetfüggetlenek, hanem az íróknak van elképzelésük az elvárt mondathosszról, amit fel is használnak lágy globális kényszerként a mondatok írásakor. Ezt nehéz visszatükrözni egy PCFG-ben.

1.2. PCFG-valószínűségek tanulása

Egy PCFG-modell létrehozásához a CFG konstrukciójának összes nehézségével szembesülünk, ehhez hozzátevődik még az egyes szabályok valószínűséggel való ellátása. Ez azt sugallja, hogy a nyelvtan adatokból való **tanulás**a (**learning**) hasznosabb lehet, mint a tudásmérnöki megközelítés. Csakúgy, mint a beszédmegértés esetén is, kétféle adat áll rendelkezésre: elemzett és nem elemzett. A feladat sokkal egyszerűbb, ha az adatok elemzési fáját nyelvészek (vagy legalábbis képzett anyanyelvi beszélők) készítették el. Egy ilyen korpusz elkészítése óriási feladat, a legnagyobb korpuszok „mindössze" körülbelül egymillió szót tartalmaznak. Az elemzési fa korpusz alapján a PCFG-t egyszerűen számlálással (és simítással) készítjük el: minden egyes nem záró szimbólumra megnézzük az összes olyan csomópontot, amelynek ez a szimbólum a gyökere, és előállítjuk a csomópontok gyermekeinek összes különböző kombinációit leíró szabályokat. Például ha az *NP* szimbólum 100 000-szer fordul elő, és ebből 20 000 esetben a gyermekek listája [*NP*, *PP*], akkor a következő szabályt állítjuk elő:

```
NP → NP PP [0,20]
```

A feladat sokkal nehezebb, ha csak elemzetlen szöveggel rendelkezünk. Először is két problémával szembesülünk: a nyelvtani szabályok struktúrájának és az egyes szabályok valószínűségének megtanulásával. (Ugyanezt a megkülönböztetést tesszük neurális hálózatok, valamint Bayes-hálók tanulása esetén is.)

Pillanatnyilag feltételezzük, hogy a szabályok struktúrája adott, és csak a valószínűségeket próbáljuk megtanulni. Alkalmazhatunk egy várhatóérték-maximalizálás (**expectation– maximization**, **EM**) módszert, úgy, mint az RMM-ek tanulásánál. A paraméterek – amelyeket tanulni próbálunk – a szabály-valószínűségek. A rejtett változók az elemzési fák: nem tudjuk, hogy a $w_i \ldots w_j$ szavakból álló karakterfüzért ténylegesen az $X \rightarrow \alpha$ szabály generálja-e, vagy sem. Az E lépés megbecsüli az egyes részszekvenciák egyes szabályok által történő generálásának valószínűségét. Ezután az M lépés megbecsüli az egyes szabályok valószínűségét. Az egész számítást el lehet végezni dinamikus programozási módon, az ún. **belső–külső** (**inside–outside**) **algoritmus**sal, ami a HMM tanulás előre–hátra algoritmusának analógiája.

A belső–külső algoritmus varázslatosnak tűnik, hiszen elemzetlen szövegekből állít elő nyelvtant. Azonban számos hátránnyal rendelkezik. Először is lassú: ahol *n* a mondatbeli szavak, *t* pedig a nem záró szimbólumok száma. Másodsorban, a valószínűségi hozzárendelések tere nagyon nagy, és a tapasztalatok alapján a lokális maximumokban való bennragadás súlyos probléma. Alternatív módszerek – például szimulált lehűtés –

[8] „egy banánt eszik" – „egy selyemkendőt eszik" (*A ford.*)
[9] A kifejezés *feje* a legfontosabb szó, például a főnév a főnévi szerkezetben.

Created by XMLmind XSL-FO Converter.

megpróbálhatók ugyan, de ezek még nagyobb számításigényűek. Harmadsorban, a kapott nyelvtanok által elvégzett elemzések gyakran nehezen érthetők, és nem elégítik ki a nyelvészeket. Ez megnehezíti a manuálisan előállított tudás kombinálását az automatikus indukcióval.

1.3. PCFG-szabálystruktúrák tanulása

Most pedig tételezzük fel, hogy a nyelvtani szabályok struktúrája nem ismert. Az első probléma, amivel szembesülünk, az, hogy a lehetséges szabályhalmazok tere végtelen, azaz nem tudjuk, hogy hány szabályt vegyünk figyelembe, és azt sem, hogy az egyes szabályok milyen hosszúak lehetnek. A probléma egyik lehetséges megkerülése az, hogy a nyelvtani szabályokat **Chomsky normál alak**ban (**Chomsky normal form**) tanuljuk, ami azt jelenti, hogy minden szabály a következő két alak egyike lehet:

$X \rightarrow Y\ Z$

$X \rightarrow t$

ahol *X*, *Y* és *Z* nem záró, míg **t** záró szimbólum. Minden környezetfüggetlen nyelvtant át lehet írni Chomsky normál alakra, amely pontosan ugyanazt a nyelvet fogadja el. Ezután önhatalmúlag *n* nem záró szimbólumra szorítkozhatunk, ezáltal $n^3 + nv$ szabályt kapunk, ahol *v* a záró szimbólumok száma. A gyakorlatban ez a módszer csak kis nyelvtanok esetén bizonyult hatékonynak. A **bayesi modellösszevonás** (**Bayesian model merging**) alternatív megközelítés hasonló a SEQUITUR modellhez (lásd 22.8. alfejezet). A módszer mondatonkénti lokális modellek (nyelvtanok) építésével kezd, majd a minimális leíróhossz felhasználásával összevonja a modelleket.

2. Információkeresés

Az **információkeresés** (**information retrieval**) feladata olyan dokumentumok megtalálása, amelyek relevánsak a felhasználó információigényére nézve. Az információkereső rendszerek legjobban ismert példái a világháló keresőgépei. A felhasználó megadhat egy lekérdezést – mint például [MI-könyv] – a keresőgépnek, majd megnézheti a releváns oldalak listáját. Ebben az alfejezetben bemutatjuk, hogyan épülnek fel ezek a rendszerek. Egy információkereső (ezentúl IR-) rendszer a következő módokon jellemezhető:

- **Dokumentumgyűjtemény.** Minden rendszernek el kell döntenie, hogy mit kezel dokumentumként: egy bekezdést, egy oldalt vagy egy többoldalas szöveget.

- **Lekérdezőnyelven megadott kérdés.** A **lekérdezés** (**query**) írja le, hogy a felhasználó mit szeretne megtudni. A **lekérdezőnyelv** (**query language**) lehet egyszerűen szavak listája (pl. [MI-könyv]); vagy meg lehet adni egymás mellett álló szavakból álló kifejezést (pl. [„MI-könyv”]); logikai operátorokat tartalmazhat (pl. [MI *ÉS* könyv]), nem logikai operátorokat (például [MI *KÖZELBEN* könyv] vagy [MI könyv *CÍM*:www.aaai.org]) foglalhat magában.

- **Eredményhalmaz.** Ez a dokumentumok azon részhalmaza, amit az IR a keresés alapján **releváns**nak (**relevant**) ítél. *Relevánson* azt értjük, hogy valószínűleg hasznos lesz a kérdést feltevő személy számára, arra a bizonyos információigényre, amelyet a lekérdezésben fogalmazott meg.

- **Az eredményhalmaz megjelenítése.** Ez lehet olyan egyszerű, mint a dokumentumcímek rendezett rangsorolt listája, vagy olyan bonyolult, mint az eredményhalmaz háromdimenziós térbe vetített, forgó színes térképe.

Az előző fejezet elolvasása után gondolhatunk arra, hogy egy információkereső rendszert úgy is felépíthetünk, hogy a dokumentumhalmazt elemzés után leképezzük logikai mondatok tudásbázisába, majd elemezzük a kérdéseket, és MEGKÉRDEZ-zük a tudásbázist, hogy a válaszokat megkapjuk. Sajnos senkinek sem sikerült így működő, nagyméretű IR-rendszert készítenie. Egyszerűen túl bonyolult olyan szókincset és nyelvtant építeni, amely képes nagy dokumentumhalmazokat lefedni, így minden IR-rendszer egyszerűbb nyelvi modellt használ.

A legkorábbi IR-rendszerek a **Boole-kulcsszó modell** (**Boolean keyword model**) szerint működtek. A dokumentumgyűjtemény minden egyes szavát úgy kezelik, mint egy Boole-tulajdonságot, amely igaz, ha a szó előfordul a dokumentumban, és hamis, ha nem. Azaz a „visszakeresés” tulajdonság igaz erre a fejezetre, de hamis a 15. fejezetre. A lekérdezőnyelv a tulajdonságok feletti logikai kifejezések nyelve. Például az [információkeresés *ÉS* lekérdezés] lekérdezés igaz erre a fejezetre, de nem igaz a 15. fejezetre.

Created by XMLmind XSL-FO Converter.

Ez a modell rendelkezik azzal az előnnyel, hogy könnyű elmagyarázni és megvalósítani. Azonban rendelkezik néhány hátránnyal is. Először, a dokumentum relevanciája egyetlen bit, így nincs semmilyen iránymutatás arra, hogy hogyan rendezzük a megjelenítés során a releváns dokumentumokat. Másodszor, a logikai kifejezések szokatlanok lehetnek a nem programozó vagy nem logikával foglalkozó felhasználók számára. Harmadszor, még egy gyakorlott felhasználó számára is nehéz lehet a megfelelő lekérdezést megfogalmazni. Tételezzük fel, hogy az [információkeresés *ÉS* modellek *ÉS* optimálás] kérdést tesszük fel, és üres eredményhalmazt kapunk vissza. Megpróbálhatjuk az [információkeresés *VAGY* modellek *VAGY* optimálás] lekérdezést, azonban ha túl sok eredményt ad vissza, akkor nehéz megmondani, mit kell utána kipróbálni.

A legtöbb IR-rendszer a szavak előfordulási statisztikájára (és esetleg más alacsony szintű jellemezőkre) épít. Bemutatunk egy valószínűségi keretrendszert, amely jól illeszkedik a tárgyalt nyelvi modellekhez. Az alapötlet az, hogy egy adott lekérdezéshez meg akarjuk találni azokat a dokumentumokat, amelyek relevánsak. Más szavakkal, ki akarjuk számolni a:

$$P(R = igaz|D, Q)$$

értéket, amelyben D a dokumentum, Q a lekérdezés, R pedig egy véletlen logikai változó, amely a relevanciát fejezi ki. Ha meghatároztuk ezt az értéket, alkalmazhatjuk a valószínűségi rendezési elvet, amely szerint amennyiben be kell mutatnunk az eredményhalmazt, akkor azt csökkenő valószínűségű relevancia szerint kell tennünk.

Számos lehetőség létezik a $P(R = igaz|D, Q)$ együttes eloszlás dekomponálására. Itt az ún. **nyelvi modellezés** (**language modeling**) megközelítést fogjuk bemutatni, amely minden egyes dokumentumra egy nyelvi modellt becsül, majd az adott dokumentum nyelvi modellje alapján minden egyes lekérdezésre kiszámítja a lekérdezés valószínűségét. Az $R = igaz$ érték jelölésére r-t használva, a következő alakra írhatjuk át a valószínűséget:

$$P(r|D, Q) = P(D, Q|r)P(r)/P(D, Q) \text{ (a Bayes-szabály alapján)}$$

$$= P(Q, D|r)P(D|r)P(r)/PP(D, Q) \text{ (a láncszabály alapján)}$$

$$= \alpha P(Q|D,r)P(r|D)/P(D, Q) \text{ (a Bayes-szabály alapján, rögzített } D\text{-re)}$$

Azt mondtuk, hogy a $P(r|D, Q)$ értékét akarjuk maximalizálni, azonban ezzel ekvivalens, ha a $P(r|D, Q)/P(\neg r|D, Q)$ valószínűségi arányt maximalizáljuk. Azaz a dokumentumokat a következő pontszám alapján rangsorolhatjuk:

$$\frac{P(r|D,Q)}{P(\neg r|D,Q)} = \frac{P(Q|D,r)P(r|D)}{P(Q|D,\neg r)P(\neg r|D)}$$

Ennek az az előnye, hogy kiküszöböli a $P(D, Q)$ tagot. Most pedig feltételezzük, hogy az irreleváns dokumentumokra a dokumentum független a lekérdezéstől. Más szavakkal, amennyiben egy dokumentum irreleváns egy adott lekérdezésre, akkor a dokumentum ismerete nem fog segíteni a lekérdezés meghatározásában. Ezt a lekérdezést a következő egyenlet írja le:

$$P(D, Q|\neg r) = P(D|\neg r)P(Q|\neg r)$$

Ezzel a feltételezéssel azt kapjuk, hogy:

$$\frac{P(r|D,Q)}{P(\neg r|D,Q)} = P(Q|D,r) \times \frac{P(r|D)}{P(\neg r|D)}$$

A $P(r|D)/P(\neg r|D)$ tényező a dokumentum relevanciájának lekérdezésfüggetlen valószínűsége. Ez a dokumentum minőségének mértéke: egyes dokumentumok *bármely* lekérdezéshez relevánsak, mert a dokumentum egyszerűen magas színvonalú. Akadémiai környezetben született folyóiratcikkek esetén a relevancia a hivatkozások száma alapján becsülhető, míg weboldalak esetén az oldalra mutató hiperhivatkozások számát használhatjuk. Minden esetben nagyobb súlyt adhatunk azoknak a hivatkozásoknak, amelyek maguk is magas színvonalúak. A dokumentum kora szintén szerepelhet a lekérdezésfüggetlen relevancia becslésében.

Az első tényező – a $P(Q|D, r)$ – a lekérdezés valószínűsége, feltéve egy adott releváns dokumentumot. Hogy megbecsülhessük ezt a valószínűséget, egy nyelvi modellt kell választanunk, amely megadja, hogy milyen kapcsolatban állnak a lekérdezések a releváns dokumentumokkal. Az egyik népszerű választás a dokumentumok

Created by XMLmind XSL-FO Converter.

unigram szómodellel történő reprezentálása. Ez az információkeresésben úgy is ismert, mint a **szózsák (bag of words)** modell, mivel a szavak dokumentumon belüli előfordulási gyakorisága az, ami számít, nem a sorrendjük. Ebben a modellben a „man bites dog" és a „dog bites man" (nagyon rövid) dokumentumok azonosan fognak viselkedni.[10] Világos, hogy *eltérő* jelentésűek, azonban az is igaz, hogy mindketten relevánsak a kutyákat és a harapásokat tartalmazó lekérdezésekre. Ezek után, hogy kiszámolhassuk egy lekérdezés valószínűségét egy adott dokumentum esetében, egyszerűen össze kell szoroznunk a lekérdezésben található szavak valószínűségeit a dokumentum unigram modellnek megfelelően. Ez a lekérdezés **naiv Bayes- (naive Bayes)** modellje. Q_j-vel jelölve a lekérdezés *j*-edik szavát, azt kapjuk, hogy:

$$P(Q|D,r) = \prod_j P(Q_j|D,r)$$

Ez lehetővé teszi a következő egyszerűsítést:

$$\frac{P(r|D,Q)}{P(\neg r|D,Q)} = \prod_j P(Q_j|D,r)\frac{P(r|D)}{(\neg r|D)}$$

Végre készen állunk, hogy ezeket a matematikai modelleket egy példára alkalmazzuk. A 23.4. ábra a [Bayes informational retrieval model] ([Bayes információkeresés modell]) lekérdezés szavainak ennek a könyvnek öt kiválasztott fejezetéből álló dokumentumgyűjtemény feletti unigram statisztikáját adja meg. Feltesszük, hogy a fejezetek azonos minőségűek, így csak azt kell kiszámolnunk, hogy mennyi a lekérdezés valószínűsége az adott dokumentum esetén, minden egyes dokumentumra. Két alkalommal tesszük ezt meg, egyszer egy D_i simítatlan maximum-likelihood becslővel, majd egy D_i' adj-hozzá-egyet simító becslőjű modellel. Azt tételeznénk fel, hogy egy ilyen keresésénél ez a fejezet lesz elsőnek rangsorolva, és valóban ez így is van mindkét modell szerint.

23.4. ábra - A [Bayes information retrieval model] lekérdezés-valószínűségi IR-modellje a könyv első öt fejezetét tartalmazó dokumentumgyűjtemény felett. Megadjuk a szógyakoriságot mindegyik dokumentum-szó párra, és a szavak számát (*N*) az összes dokumentumra. Két dokumentummodellt alkalmazunk, D_i az *i*-edik dokumentumon alapuló simítatlan unigram szómodell, míg D'_i ugyanaz a modell adj-hozzá-egyet simítással, majd kiszámítjuk a lekérdezés valószínűségét minden dokumentumra mindkét modellel. A jelen (23.) fejezet az egyértelmű győztes, mindkét modell esetén több mint kétszázszor valószínűbb, mint bármely más dokumentum.

Szavak	Lekérdezés	1. fejezet Bevezetés	13. fejezet Bizonytalanság	15. fejezet Idő	22. fejezet NLP	23. fejezet Aktuális
Bayes	1	5	32	38	0	7
information	1	15	18	8	12	39
retrieval	1	1	1	0	0	17
model	1	9	7	160	9	63
N	4	14 680	10 941	18 186	16 397	12 574
$P(Q\|D_i, r)$		$1{,}5 \times 10^{-14}$	$2{,}8 \times 10^{-13}$	0	0	$1{,}2 \times 10^{-11}$
$P(Q\|D'_i, r)$		$4{,}1 \times 10^{-14}$	$7{,}0 \times 10^{-13}$	$5{,}2 \times 10^{-13}$	$1{,}7 \times 10^{-15}$	$1{,}5 \times 10^{-11}$

A simított modell rendelkezik azzal az előnnyel, hogy kevésbé érzékeny a zajra, és hogy nemzérus relevancia-valószínűséget képes rendelni olyan dokumentumokhoz, amelyek nem tartalmazzák az összes szót. A simítatlan modellnek az az előnye, hogy könnyű sok dokumentumot tartalmazó gyűjteményekre kiszámolni: ha elkészítünk egy olyan indexet, amely megadja, hogy az adott szót mely dokumentumok tartalmazzák, akkor gyorsan elő tudjuk állítani az eredményhalmazt ezeknek a listáknak a metszeteként, és a $P(Q|D_i)$ értékeket csak a metszetben szereplő dokumentumokra kell kiszámolni, nem pedig mindegyikre.

[10] „Egy kutya megharapott egy embert", illetve „egy ember megharapott egy kutyát", jellemző újságcímstílus. (*A ford.*)

Created by XMLmind XSL-FO Converter.

2.1. Az IR-rendszerek értékelése

Honnan tudjuk, hogy egy IR-rendszer jól teljesít? Elvégzünk egy kísérletet, amelyben a rendszer kap egy lekérdezéshalmazt, az eredményhalmazokat pedig pontozzuk az emberi relevanciamegítélés szerint. Tradicionálisan két mértéket használunk a pontozásra: a **felidézést** (**recall**) és **pontosságot** (**precision**). Ezeket egy példán keresztül fogjuk bemutatni. Képzeljük el, hogy egy IR-rendszer visszaadott egy eredményhalmazt egy olyan lekérdezésre, amelyre tudjuk, hogy egy 100 dokumentumot tartalmazó korpuszból mely dokumentumok relevánsak, és melyek nem. Az egyes kategóriákba tartozó dokumentumok számát az alábbi táblázat adja meg:

	Az eredményhalmazban	Nem az eredményhalmazban
Releváns	30	20
Nem releváns	10	40

A **pontosság** az eredményhalmaz dokumentumai közül a ténylegesen relevánsak arányát méri. A példánkban a pontosság 30/(30 + 10) = 0,75. A hamis pozitív arány 1 – 0,75 = 0,25. A **felidézés** a gyűjtemény releváns dokumentumaiból az eredményhalmazban megjelenő hányadát méri. A példánkban a felidézés 30/(30 + 20) = 0,60. A hamis negatív arány 1 – 0,60 = 0,40. Egy nagyon nagy dokumentumgyűjteményben, mint például a világhálón, a felidézés nehezen számítható, mivel nincs egyszerű módszer a web összes oldalának a relevancia szempontjából történő elemzésére. A legjobb, amit tehetünk, a felidézés becslése mintavételezéssel, vagy pedig teljesen figyelmen kívül hagyjuk a felidézést, és csak a pontosság alapján ítélünk.

A rendszer kompromisszumot köthet a pontosság és felidézés között. Extrém esetben a rendszer visszaadhatja az összes dokumentumot a dokumentumgyűjteményből az eredményhalmazban, 100%-os felidézést garantálva, azonban a pontossága kicsi lesz. Alternatívaként, a rendszer visszaadhat egyetlen dokumentumot, így a felidézés alacsony lesz, azonban tűrhető esélye van 100%-os pontosságra. A kompromisszum összefoglalásának egyik lehetséges módja az **ROC-görbével** (**ROC curve**) történhet. Az „ROC” a „**vevő működési karakterisztika**” (**receiver operating characteristics**) rövidítése (ami nem túlzottan felvilágosító név). Ez egy olyan grafikon, amely a hamis negatív arányt méri az *y* tengelyen és a hamis pozitív arányt az *x* tengelyen, ábrázolva az egyes kompromisszumos pontokat. A görbe alatti terület az IR-rendszer hatékonyságának összefoglalása.

A felidézést és pontosságot akkor definiálták, amikor az IR-kereséseket elsődlegesen könyvtárosok végezték, akik alapos, pontos találatokban voltak érdekeltek. Manapság a legtöbb (napi több százmillió) lekérdezést az internetfelhasználók végzik, akik kevésbé érdekeltek az alaposságban, sokkal inkább abban, hogy azonnal választ kapjanak. Számukra jó mérték az első releváns találat átlagos **reciprokrang**ja (**reciprocal rank**). Azaz, amennyiben a rendszer első találata releváns, 1-es pontszámot kap a lekérdezésre, és amennyiben az első kettő nem releváns, de a harmadik az, akkor 1/3-ot. Egy alternatív mérték a **válaszidő** (**time to answer**), ami azt méri, hogy mennyi ideig tart a felhasználónak a problémára kívánt választ megtalálni. Ez kerül a legközelebb ahhoz, amit mérni szeretnénk, azonban azzal a hátránnyal rendelkezik, hogy minden egyes kísérlethez új emberi tesztalanycsoportra van szükség.

2.2. Az IR-rendszerek továbbfejlesztése

Az unigram modell az összes szót függetlenként kezeli, azonban mi tudjuk, hogy bizonyos szavak korreláltak: a „dívány” közeli kapcsolatban áll mind a „díványok”-kal, mind a „kanapé”-val. Számos IR-rendszer próbálja figyelembe venni ezeket a korrelációkat.

Például, amennyiben a lekérdezés [dívány], gyalázatos lenne kihagyni az eredményhalmazból azokat a dokumentumokat, amelyek a „DÍVÁNY” vagy a „díványok” szavakat tartalmazzák, de a „dívány”-t nem. A legtöbb IR-rendszer **kisbetű-nagybetű konverzió**t (**case folding**) alkalmaz, hogy a „DÍVÁNY”-t „dívány” alakká alakítsa, számos rendszer pedig **szótövesítő** (**stemming**) algoritmusokat, hogy a „díványok” alakot a „dívány” szótőre redukálja. Ez tipikusan a felidézés kismértékű növekedését eredményezi (az angol nyelv esetén kb. 2%-ot). Azonban ronthatja a pontosságot. Például a „stocking” szótövesítése „stock”[11] alakra valószínűleg csökkenteni fogja a ruházati, valamint pénzügyi eszközökre irányuló lekérdezések pontosságát, azonban növelheti a raktározásra irányuló lekérdezések felidézését. Szabályalapú szótövesítők (például az „-ing” végződés eltávolítása az angolban) nem képesek megkerülni ezt a problémát, de a szótárakon alapuló újabb

[11] Harisnya, illetve raktárkészlet. (*A ford.*)

Created by XMLmind XSL-FO Converter.

algoritmusok igen (nem kell eltávolítani az „-ing” képzőt, ha a szó már szerepel a szótárban). Bár az angol nyelv esetén a szótövesítésnek csak kis hatása van, sokkal fontosabb más nyelvek esetén. A német nyelvben gyakran találkozhatunk olyan szavakkal, mint „Lebensversicherungsgesellschattsangestellter” (életbiztosító cég alkalmazottja). Az olyan nyelvek, mint a finn, a török, az inuit vagy a jupik rekurzív morfológiai szabályokkal rendelkeznek, amelyek elméletileg korlátlan hosszúságú szavakat generálnak.

A következő lépés a **szinonimá**k (**synonym**s) felismerése, mint amilyen a „dívány” és a „kanapé”. A szótövesítéshez hasonlóan ez is a felidézés kismértékű növekedését eredményezheti, azonban a pontosságot veszélyezteti, ha túl agresszíven alkalmazzuk. Akik Tim Couch futballistára kíváncsiak, nem szeretnének átvergődni a kanapékról (couch – kanapé) szóló dokumentumokon. Az a probléma, hogy „a nyelv ugyanúgy irtózik az abszolút szinonimáktól, mint ahogy a természet retteg a vákuumtól” (Cruse, 1986). Azaz, amennyiben két szó azonos dolgot jelent, a nyelv beszélői törekednek a jelentés módosítására, hogy megszüntessék a zűrzavart.

Számos IR-rendszer bizonyos mértékig szó **bigram**okat használ, azonban csak néhányuk valósít meg egy teljes valószínűségi bigram modellt. A **helyesírás-javító** (**spelling correction**) eljárások alkalmazhatók mind a dokumentumok, mind a lekérdezések hibáinak javítására.

Végső finomításként az IR-rendszerek tökéletesíthetők **metaadat**ok (**metadata**) figyelembevételével, amelyek a dokumentum szövegén kívül álló adatok, mint például emberek által megadott kulcsszavak vagy dokumentumok közötti hypertext-hivatkozások.

2.3. Az eredményhalmaz prezentálása

A valószínűségi rendezési elv szerint vegyünk egy eredményhalmazt, és a relevancia valószínűségének megfelelően sorba rendezve prezentáljuk a felhasználónak. Ennek akkor van értelme, ha a felhasználó az összes releváns dokumentum minél hamarább történő megtalálásban érdekelt. Azonban bajba kerül, mert nem veszi figyelembe a *hasznosságot*. Például ha a legrelevánsabb dokumentum két példányban szerepel a gyűjteményben, akkor az első megnézése után a második azonos relevanciájú, de zérus hasznosságú. Számos IR-rendszer rendelkezik mechanizmusokkal, amelyek eliminálják az előző találatokhoz túlzottan hasonlító eredményeket.

Az IR-rendszerek teljesítménynövelésének egyik leghatékonyabb módja a **relevancia-visszacsatolás** (**relevance feedback**), amely a felhasználó visszajelzése, hogy az eredeti eredményhalmazból mely dokumentumok voltak relevánsak. A rendszer ezután egy második eredményhalmazt prezentálhat, amelynek dokumentumai hasonlók a megadottakhoz.

Egy másik lehetséges megközelítés az eredményhalmaznak egy rendezett lista helyett egy *címkézett faként* történő prezentálása. A **dokumentumosztályozás** (**document classification**) során az eredményeket egy előre definiált témakör-taxonómiának megfelelően osztályozzuk. Például újsághírekből álló gyűjteményt a következő kategóriákba lehet sorolni: külföldi, belföldi, üzleti hírek, szórakozás és sport. **Dokumentumklaszterezés** (**document clustering**) esetén minden eredményhalmazra teljesen új kategóriafa készül. Az osztályozás akkor használható, ha a gyűjteményben kevés számú témakör található, míg a klaszterezést a világhálóhoz hasonló széles körű gyűjtemények esetén érdemes használni. Mindkét esetben, miután a felhasználó megad egy lekérdezést, az eredményhalmazt a kategóriáknak megfelelő mappákba rendezve kapja meg.

Az osztályozás felügyelt tanítási probléma, és mint ilyen, a 18. fejezetben ismertetett bármelyik módszerrel megtámadható. Az egyik népszerű megközelítés a döntési fák alkalmazása. Amennyiben rendelkezünk a megfelelő kategóriákkal címkézett dokumentumokból álló tanító halmazzal, építhetünk egyetlen döntési fát, amelynek levelei a dokumentumot a megfelelő kategóriához rendelik. Ez akkor működik jól, ha mindössze néhány kategória van; nagyobb kategóriahalmazok esetén minden egyes kategóriára külön döntési fát építünk, amelynek a levelei megadják, hogy a dokumentum az adott kategóriába tartozik-e vagy sem. Általában az egyes csomópontokban tesztelt tulajdonságok egyedi szavak. Például a „Sport” kategóriában az egyik csomópont tesztelheti a „kosárlabda” szó meglétét. Javított teljesítményű döntési fák, naiv Bayes-modellek, valamint szupport vektor gépek mindegyikét használták szövegosztályozásra, sok esetben a hitelesség 90–98%-os volt bináris osztályozás esetén.

A klaszterezés egy felügyelet nélküli tanítási módszer. A 20.3. alfejezetben láthattuk, hogy hogyan alkalmazható az EM algoritmus a klaszterezés eredeti becslésének javítására, Gauss-modellek keverékét használva. A dokumentumok klaszterezése nehezebb feladat, mert nem tudjuk, hogy az adatokat egy barátságos Gauss-

Created by XMLmind XSL-FO Converter.

modell generálta-e, és mert egy sokkal több dimenziós térrel kell elbánnunk. Számos megközelítést dolgoztak ki.

Az **agglomeratív klaszterezés** (**agglomerative clustering**) klaszterekből álló fát épít, lemenve egészen az egyedi dokumentumok szintjére. A fa bármely szinten nyeshető, hogy kevesebb kategóriát kapjunk, de ezt az algoritmuson kívül vesszük figyelembe. Az elején minden dokumentumot külön klaszternek tekintünk. Ezután megkeressük azt a két klasztert, melyek egy bizonyos távolságmérték szerint legközelebb állnak egymáshoz, és összevonjuk őket. Addig ismételjük a folyamatot, amíg csak egy klaszter marad. A két dokumentum távolságát meghatározó mérték a dokumentumok szavai közti átfedés valamilyen mértéke. Például reprezentálhatjuk a dokumentumot szógyakoriságok vektoraként, ahol a távolságot a két vektor euklideszi távolságaként értelmezzük. Két klaszter távolsága a klaszterek mediánjainak távolságaként értelmezhetjük, vagy a klaszterek elemeinek átlagos távolságát vehetjük figyelembe. Az agglomeratív klaszterezés időigénye $O(n^2)$, ahol n a dokumentumok száma.

A **k-közép klaszterezés** (**k-means clustering**) pontosan k darab kategória halmazát állítja elő. A következő elven működik:

1. Vegyünk véletlenszerűen k dokumentumot a k kategória reprezentálására.
2. Rendeljünk minden dokumentumot a legközelebbi kategóriához.
3. Számoljuk ki minden egyes kategória átlagát, és használjuk a k átlagot a k kategóriák új értékeinek reprezentálására.
4. Ismételjük a 2-es és 3-as lépéseket, amíg konvergálnak.

A k-közép módszer $O(n)$ időigényű, ez az egyetlen előnye az agglomeratív klaszterezéshez képest. Általában kevésbé pontos, mint az agglomeratív klaszterezés, bár egyesek szerint majdnem olyan hatékony (Steinbach és társai, 2000).

Az alkalmazott klaszterezési módtól függetlenül van még egy feladat, amit el kell végeznünk, mielőtt a klaszterezést az eredményhalmaz bemutatására használhatjuk: a klaszter jó leírásának megtalálása. Az osztályozás esetén a kategóriák előre definiáltak (például „jövedelmek”), míg a klaszterezés esetén ki kell találnunk a kategórianeveket. Az egyik választás a klaszter szempontjából reprezentatív szavak listájának használata. A másik lehetőség a klaszter középpontjához közel levő egyik dokumentum címének a használata.

2.4. Az IR-rendszerek megvalósítása

Az eddigiekben csupán absztrakt módon definiáltuk az IR-rendszerek működését, azonban nem magyaráztuk el, hogy hogyan lehet olyan hatékonnyá tenni őket, hogy egy webes keresőgép visszaadhassa a legfelső találatokat egy több milliárd oldalas gyűjteményből egytized másodperc alatt. Minden IR-rendszer számára két kulcsfontosságú adatstruktúra szükséges: a szókincs, amely felsorolja a dokumentumok szavait, és az invertált index, amely megadja, hogy az egyes szavak hol szerepelnek a dokumentumgyűjteményben.

A **szókincs** (**lexicon**) egy olyan adatstruktúra, amely egy műveletet támogat: megadja, hogy egy adott szó hol szerepel az invertált indexben, amely a szó előfordulásait tárolja. Egyes megvalósítások esetén azt is visszaadja, hogy összesen hány dokumentum tartalmazza a szót. A szókincset hash-tábla vagy valamilyen hasonló adatstruktúrával ajánlott megvalósítani, amely lehetővé teszi a gyors kikeresést. Egyes esetekben kis információtartalmú gyakori szavakat kihagynak a szókincsből. Ezek a **tiltólistás szavak** (**stop words**) (például „a”, „egy”, „ez” stb.) helyet foglalnak az indexben, és nem javítják az eredmény rangsorolását. Az egyetlen jó indok arra, hogy mégis megtartsuk őket a szókincsben a kifejezéslekérdezést támogató rendszerek esetén áll fenn: a tiltólistás szavak szókincsbeli tárolása szükséges olyan lekérdezések kiszolgálására, mint pl. a „to be or not to be”.[12]

Az **invertált index**[13] (**inverted index**), a könyvünk végén található szójegyzékhez hasonlóan, **találati listá**k (**hit list**s) halmazából – az egyes szavak előfordulási helyeiből – áll. A Boole-kulcsszó modell esetén a találati lista nem más, mint a dokumentumok listája. Az unigram modell esetén egy (dokumentum, gyakoriság) párokat

[12] lenni vagy nem lenni (*A ford.*)

[13] Az „invertált index” kifejezés redundáns, sokkal jobb lenne egyszerűen az „index” kifejezést használni. Azért invertált, mert más sorrendben van, mint a szöveg szavai, de ilyen minden index. Azonban az „invertált index” a hagyományos IR-kifejezés.

Created by XMLmind XSL-FO Converter.

tartalmazó lista. A kifejezéslekérdezés támogatásához a találati listának az adott szó minden egyes dokumentumon belüli előfordulási helyeit is tartalmaznia kell.

Amennyiben a lekérdezés egyetlen szó (Silverstein szerint ilyen az esetek 26%-a (Silverstein és társai, 1998)), akkor a feldolgozás igen gyors. Egyszerűen kikeressük a szót a szókincsből, hogy megkapjuk a találati lista címét, majd egy üres prioritási sort hozunk létre. Ezután egyesével végigmegyünk a találati lista dokumentumain, és megnézzük a szó gyakoriságát a dokumentumban. Amennyiben a prioritási sor R-nél kevesebb elemet tartalmaz (ahol R az eredményhalmaz elvárt mérete), akkor hozzáadjuk a (dokumentum, gyakoriság) párt a sorhoz. Ellenkező esetben, ha a gyakoriság nagyobb, mint a prioritási sor legkisebb elemének gyakorisága, akkor töröljük a legkisebb elemet, és hozzáadjuk az új (dokumentum, gyakoriság) párat. Ezáltal a lekérdezés megválaszolása $O(H + R \log R)$ időt vesz igénybe, ahol H a találati lista dokumentumainak száma. Amennyiben a lekérdezés n szót tartalmaz, akkor n találati listát kell összevonni, ami $O(nH + R \log R)$ időt vesz igénybe.

Azért mutattuk be az IR-rendszerek elméletét a valószínűségi modellen keresztül, mert ez a modell hasznosítja azokat az ötleteket, amelyeket más témákban is alkalmaztunk. A jelenlegi gyakorlati IR-rendszerek sokkal inkább egy másik megközelítést alkalmaznak, amelyet **vektortér modell**nek (**vector space model**) hívunk. Ez a modell ugyanúgy a szózsák-megközelítést alkalmazza, mint a valószínűségi modell. Minden egyes dokumentumot unigram szógyakoriságok vektoraként ábrázolunk. A lekérdezést is így ábrázoljuk, például a [Bayes information retrieval model] lekérdezést a

```
[0, …, 1, 0, …, 1, 0, …, 1, 0, …, 1, 0, …]
```

vektor írja le, amelyben az ötlet az, hogy minden egyes szóhoz külön dimenzió tartozik, és a lekérdezés vektorában minden dimenzió 0 értékű, kivéve azt a négy szót, amely a lekérdezésben előfordul. A releváns dokumentumokat úgy kapjuk meg, hogy megkeressük azokat a dokumentumokat, amelyek a lekérdezés vektorának legközelebbi szomszédai a vektortérben. A hasonlóság egyik lehetséges mértéke a lekérdezés- és dokumentumvektor skaláris szorzata: minél nagyobb ez a szorzat, annál közelebb van egymáshoz a két vektor. Algebrailag ez a mérték nagy pontszámot ad azoknak a szavaknak, amelyek mind a dokumentumokban, mind a lekérdezésben gyakran előfordulnak. Geometriailag a két vektor skaláris szorzata az általuk bezárt szög koszinusza, azaz ha két ilyen vektor koszinuszát maximalizáljuk (amennyiben azonos kvadránsban találhatók), akkor az általuk bezárt szög nullához közeli lesz.

A vektortér modell ennél sokkal többre képes. A gyakorlatban számos jellemzővel, finomítással, javítással és kiegészítéssel lett kibővítve. Az az alapötlet, miszerint a dokumentumokat a vektortérbeli hasonlóságuk alapján lehet rangsorolni, lehetővé teszi, hogy új ötleteket építsünk be a numerikus sorrendező rendszerbe. Egyesek azt állítják, hogy a valószínűségi modell ezeket a módosításokat sokkal tisztább elvi alapokon állva tenné lehetővé, azonban az IR-kutatók nem fognak váltani, amíg nem látnak egyértelmű teljesítményjavulást a másik modellhez képest.

Hogy egy átlagos IR-feladat indexelési problémájának nagyságrendjét érzékeltessük, vegyünk egy szabványos TREC (Text REtrieval Conference – Szöveg-visszakeresési Konferencia) dokumentumgyűjteményt, amely 750 ezer dokumentumot tartalmaz, összesen 2 GB (gigabájt) szöveggel. A szókincs körülbelül 500 ezer szót tartalmaz, szótövesítés és kisbetű-nagybetű konverzió után; ennyi szó tárolása 7 és 10 MB közötti tárterületet igényel. Az invertált index a (dokumentum, gyakoriság) párokkal 324 MB területet igényel, bár tömörítési technikák alkalmazásával csak 83 MB-ot. A tömörítés tárhelyet takarít meg, a feldolgozási követelmények kismértékű növelése árán. Azonban ha a tömörítés lehetővé teszi, hogy az egész indexet a memóriában – és nem háttértáron tároljuk –, akkor jelentős eredő teljesítménynövekedést kapunk. A kifejezéslekérdezés támogatása a tárigényt 1200 MB-ra növeli tömörítetlen, illetve 600 MB-ra tömörített esetben. A webes keresőgépek háromezerszer ekkora feladattal dolgoznak. A legtöbb probléma az esetükben is hasonló, azonban nem praktikus terabájtnyi mennyiségű adatot kezelni egyetlen számítógépen, ezért az indexet k szegmensre vágják, és minden szegmenst külön számítógép tárol. A lekérdezést mindegyik számítógép megkapja, majd a k eredményhalmazt egy eredményhalmazba vonják össze, amelyet megjelenítenek a felhasználó számára. A webes keresőgépeknek ráadásul másodpercenként több ezer lekérdezést kell kiszolgálniuk, így a k számítógép n másolatára van szükség. Az idő folyamán k és n folyamatosan növekszik.

3. Információkinyerés

Az **információkinyerés** (**information extraction**) adatbázis-bejegyzések előállításának a folyamata, amely során egy szöveget egy adott osztályba tartozó objektum vagy esemény előfordulásait, valamint az objektumok és események közti relációkat keresve átfutunk. Megpróbálhatjuk konkrét címek kinyerését weboldalakról,

Created by XMLmind XSL-FO Converter.

adatbázismezőkkel az utca, az állam és az irányítószám számára, vagy konkrét viharok adatainak a kinyerését időjárás-jelentésekből, mezőkkel a hőmérséklet, a szélsebesség és a csapadék számára. Az információkinyerő rendszerek félúton állnak az információkereső rendszerek és a teljes nyelvi elemzők között, hiszen többet kell tenniük, mint egyszerűen szózsáknak tekinteni a dokumentumot, de kevesebbet, mint minden egyes mondat teljes nyelvtani elemzését.

Az információkinyerő rendszerek legegyszerűbb típusát **attribútumalapú** (**attribute-based**) rendszernek nevezzük, mert feltételezi, hogy a teljes szöveg egy objektumról szól, és a feladat ezen objektum attribútumainak kinyerése. Például a 10.5. alfejezetben megemlítettük a „17 hüvelykes SXGA Monitor mindössze 249,99 $-ért” szövegből az alábbi adatbázis-reláció kinyerésének problémáját:

$$\exists\ m\ m \in \mathit{SzámítógépMonitorok} \wedge \mathit{Méret}(m, \mathit{Hüvelyk}(17)) \wedge \mathit{Ár}(m, \$(249{,}99))$$

$$\wedge\ \mathit{Felbontás}(m, 1280 \times 1024)$$

Ennek az információnak egy része **reguláris kifejezés**ekkel (**regular expression**) kezelhető, amelyek reguláris nyelvtant definiálnak egy karakterfüzérben. Reguláris kifejezéseket használnak olyan Unix-parancsokban, mint a grep, olyan programozási nyelvekben, mint a Perl, valamint olyan szövegszerkesztőkben, mint a Microsoft Word. A részletek eltérők az egyes eszközökben, és legjobban a megfelelő útmutatóból tanulhatók meg, azonban itt bemutatjuk, hogyan lehet reguláris kifejezéseket építeni dollárban megadott árakra, bemutatva a közös részkifejezéseket:

```
[0-9] megfelel bármely 0 és 9 közti számjegynek

[0-9] + megfelel egy vagy több számjegynek

.[0-9] [0-9] megfelel számjegyek által követett pontnak

(.[0-9] [0-9]) ? megfelel számjegyek által követett pontnak vagy

üres füzérnek

$[0-9] + (.[0-9][0-9])? megfelel 249,99 $-nak, vagy 1,23 $-nak, vagy

1 000 000 $-nak, vagy…
```

Az attribútumalapú információkinyerő rendszerek reguláris kifejezések sorozatából építhetők fel, ahol minden egyes attribútumhoz egy reguláris kifejezés tartozik. Ha a reguláris kifejezés pontosan egyszer illeszkedik a szövegre, akkor kivehetjük a szöveg illeszkedő részét, amely az attribútum értéke lesz. Ha nincs illeszkedés, akkor nem tudunk mit tenni, azonban ha több illeszkedés is van, akkor szükségünk van egy eljárásra, hogy hogyan válasszunk közülük. Az egyik megoldás szerint minden egyes attribútumhoz több reguláris kifejezést rendelünk, prioritás szerint sorba rendezve. Például a legnagyobb prioritású árra vonatkozó reguláris kifejezés a dollárjel előtti „ár:” karakterfüzérre kereshet, ha ezt nem találja meg, akkor átlépünk egy kevésbé megbízható reguláris kifejezésre. Egy másik stratégia az összes találat kinyerése, majd valamilyen módon választani közülük. Például vehetjük azt a legalacsonyabb árat, amely a legnagyobbnak legalább fele. Ez a megoldás kezelni tudja az olyan szövegeket, mint a „Listaár 99,00 $, akciós ár 78,00 $, szállítás 3,00 $”.

Az attribútumalapú rendszereknél eggyel összetettebbek a relációs alapú információkinyerő rendszerek, amelyeknek több mint egy objektummal kell foglalkozniuk, ráadásul a köztük levő relációkat is figyelembe kell venniük. Azaz, amikor egy ilyen rendszer a „249,99 $” szöveget látja, akkor nem csak azt kell megállapítania, hogy ez egy ár, hanem azt is, hogy mely objektumnak az ára. A FASTUS egy tipikus relációs alapú információkinyerő rendszer, amely cégegyesülésekről és felvásárlásokról szóló híreket képes kezelni. El tudja olvasni a következő hírt:

„Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan.”[14]

és egy a következőhöz hasonló adatbázisrekordot tud létrehozni:

$$e \in \mathit{VegyesVállalat} \wedge \mathit{Termék}(e, \mathit{„Golfütők”}) \wedge \mathit{Dátum}(e, \mathit{„Péntek”})$$

[14] „A Bridgestone Sports Vállalat pénteken azt nyilatkozta, hogy vegyes vállalatot hozott létre egy helyi konszernnel és egy japán kereskedőházzal Tajvanon, hogy Japánba szánt golfütőket gyártsanak.” A *golf club* kétértelmű, itt golfütő. (*A ford.*)

Created by XMLmind XSL-FO Converter.

∧ *Entitás*(e, „*Bridgestone Sports Co.*”) ∧ *Entitás*(e, „*egy helyi konszern*”)

∧ *Entitás*(e, „*egy japán kereskedőház*”)

Relációs információkinyerő rendszereket gyakran építenek **kaszkádosított véges állapotú átalakító**k (**cascaded finite-state transducer**) segítségével. Ez azt jelenti, hogy egy sor **véges állapotú automatá**ból (**finite-state automaton**, **FSA**) áll, ahol minden egyes automata egy szöveget kap bemenetként, amelyet átalakít egy más formába, majd továbbadja a következő automatának. Ez azért megfelelő, mert az egyes véges automaták hatékonyak lehetnek, és együtt képesek lehetnek a szükséges információ kinyerésére. A FASTUS egy tipikus rendszer, amely a következő öt fokozatból áll:

1. Tokenizálás
2. Komplex szavak kezelése
3. Alapcsoport-kezelés
4. Komplex kifejezések kezelése
5. Struktúra-összevonás

A FASTUS első lépése a **tokenizálás** (**tokenization)**, amely a karakterfüzéreket tokenekbe (szavak, számok és írásjelek) szegmentálja. Az angol nyelv esetén a tokenizálás elég egyszerű lehet, pusztán a szóközök (white space) vagy írásjelek mentén történő szegmentálás viszonylag jó hatékonyságú. A japán nyelv esetén a tokenizálásnak szegmentálást kell végeznie, valami olyasmit használva, mint a Viterbi algoritmus (lásd 23.1. ábra). Egyes tokenizálók az olyan jelölőnyelveket is kezelik, mint a HTML, az SGML és az XML.

A második lépés **komplex szavak** (**complex word**s) kezelése, ideértve az olyan szókapcsolatokat, mint a „set up” és a „joint venture”, valamint olyan tulajdonneveket, mint a „Prime Minister Tony Blair ” és a „Bridgestone Sports Co.”.[15] Ezeket szótárbejegyzések és véges állapotú nyelvtani szabályok kombinációjaként ismeri fel. Például egy cégnevet a közvetkező szabállyal lehet felismerni:

```
NagybetűvelKezdődőSzó + („Company” | „Co” | „Inc” | „Ltd”)
```

A szabályokat gondosan kell megalkotni, majd ellenőrizni kell a felidézést és a pontosságot. Az egyik kereskedelmi rendszer az „Intel Chairman Andy Grove”[16] szöveget személy helyett helyként ismerte fel, az egyik szabály miatt:

```
NagybetűvelKezdődőSzó + („Grove” | „Forest” | „Village” | …)
```

A harmadik lépés az **alapcsoport**ok (**basic group**s) kezelése, amelyek főnévi és igei csoportokat jelentenek. A lépés lényege, hogy ezeket darabokra kell vágni, hogy a későbbi lépések kezelhessék őket. A főnévi csoport a fejszerepet betöltő főnévből áll, amelyet opcionálisan megelőzhetnek névelők és egyéb módosítók. Mivel a főnévi csoport nem tartalmazza az *NP* $\mathcal{E}_1$-beli teljes komplexitását, ezért nincs szükség rekurzív környezetfüggetlen nyelvtani szabályokra, a véges automaták által megengedett reguláris nyelvtanok is elégségesek. Az igei csoport az igéből és a hozzá kapcsolódó segédigékből, valamint határozókból áll, azonban nem tartalmazza a közvetlen és közvetett tárgyakat és az elöljárói kifejezéseket. Az előbbi példamondat a következő módon kerülne ki ebből a lépésből:

```
1  NG:  Bridgestone  Sports 10 NG: a local concern
Co.

2 VG: said                  11 CJ: and

3 NG: Friday                12  NG:  a  Japanese  trading
                            house

4 NG: it                    13 VG: to produce
```

[15] „vegyes vállalat”, „Tony Blair Miniszterelnök”, „Bridgestone Sports Vállalat” (*A ford.*)
[16] Az Intel elnöke, Andy Grove – a *grove* magyarul ligetet jelent. (*A ford.*)

Created by XMLmind XSL-FO Converter.

```
5 VG: has set up          14 NG: golf clubs

6 NG: a joint venture     15 VG: to be shipped

7 PR: in                  16 PR: to

8 NG: Taiwan              17 NG: Japan

9 PR: with
```

Az NG főnévi csoportot, a VG igei csoportot, a PR elöljárószót, a CJ pedig kötőszót jelent.

A negyedik lépés az alapcsoportokat **komplex kifejezés**ekké (**complex phrase**s) kombinálja. Ismét az a cél, hogy véges állapotú szabályokat használjunk a gyors feldolgozás érdekében, és a szabályok olyanok legyenek, hogy (közel) egyértelmű kimeneti kifejezéseket kapjunk. A kombinációs szabályok egyik típusa tárgyterületfüggő eseményekkel foglalkozik. Például a

```
Cég + VegyesVállalatLétrehozása („with” Vállalat+)?
```

szabály a vegyes vállalat létrehozásának egyik lehetséges leírása. A kaszkádban ez az első lépés, amely eredményét nemcsak a kimenetre, hanem adatbázissémákba is elhelyezi.

Az utolsó lépés az előző lépés által felépített **struktúrákat vonja össze** (**merges structures**). Amennyiben a következő mondat azt állítja, hogy „A vegyes vállalat januárban kezdi meg a termelést”, akkor ez a lépés felismeri, hogy két hivatkozás történt a vegyes vállalatra, amelyeket össze kellene vonni.

Általánosságban az információkinyerés jól működik egy olyan korlátozott tárgyterület esetén, amelyben a tárgyalt témákat előre meg lehet állapítani, továbbá azt is lehet tudni, hogy hogyan írnak róluk. Számos tárgyterületen használhatónak bizonyult ez a technika, azonban nem lehet a teljes természetes nyelvi elemzés helyettesítője.

4. Gépi fordítás

A gépi fordítás (machine translation) egy természetes nyelvű szöveg egyik nyelvről (forrás) egy másik nyelvre (cél) történő automatikus fordítása. Ez a folyamat számos feladatra alkalmasnak bizonyult, beleértve a következőket:

1. **Nyersfordítás** (**rough translation**), amelyben a cél pusztán az, hogy egy passzus lényegét megkapjuk. Nyelvtanilag helytelen és nem elegáns mondatokat is elfogadunk, mindaddig, amíg a jelentés világos. Például a webszörfözés során a felhasználó gyakran örül az idegen nyelvű weboldal nyersfordításának. Az esetek egy részében egy egynyelvű ember utólagosan szerkesztheti a kimenetet, az eredeti szöveg elolvasásának szükségessége nélkül. A géppel támogatott fordítás ezen típusa azért takarít meg pénzt, mert az ilyen szerkesztőknek kevesebbet kell fizetni, mint a kétnyelvű fordítóknak.

2. **Korlátozott forrás fordítása** (**restricted-source translation**), amelyben a forrásszöveg témája és formátuma szigorúan korlátozott. Az egyik legsikeresebb példa a TAUM-METEO rendszer, amely időjárás-jelentéseket fordít angolból franciára. Azért működik, mert az időjárás-jelentések nyelvezete erősen stilizált és reguláris.

3. **Előre szerkesztett fordítás** (**preedited translation**), amelyben egy ember úgy szerkeszti meg előre a forrásdokumentumot, hogy megfeleljen az angol nyelv (vagy bármely forrásnyelv) egy korlátozott részhalmazának. Ez a megközelítés különösen akkor költséghatékony, amikor egyetlen dokumentumot sok nyelvre kell lefordítani, mint például jogi szövegeket az Európai Közösségnél, vagy olyan vállalatok esetén, amelyek ugyanazt a terméket sok országban értékesítik. A korlátozott nyelveket néha „Caterpillar-angolnak” nevezik, mert a Caterpillar vállalat volt az első, amelyik az útmutatóit ilyen formában próbálta megírni. A Xerox olyan nyelvet definiált a javítási útmutatók számára, amely annyira egyszerű volt, hogy géppel le lehetett fordítani minden olyan nyelvre, amellyel a Xerox érintkezésbe kerül. További haszonként az eredeti angol nyelvű útmutatók is világosabbá váltak.

Created by XMLmind XSL-FO Converter.

4. **Irodalmi fordítás** (**literary translation**), amely a forrásszöveg legkisebb árnyalatát is megőrzi. Ez jelenleg túlmutat a legmodernebb gépi fordítók képességén is.

A nyersfordítás példájaként álljon itt jelen fejezetünk első bekezdésének a SYSTRAN fordító szolgáltatás által történt fordítása előbb olaszra, majd vissza angolra:[17]

OLASZ: In capitolo 22 abbiamo visto come un agente potrebbe comunicare con un altro agente (essere umano o software) che usando le espressioni in un linguaggio reciprocamente accordato. Completare sintattico e l'analisi semantica delle espressionië necessaria da estrarre il significato completo del utterances ed è possible perché le espressioni sono corte e limitate ad un settore limitato.	ANGOL: In chapter 22 we have seen as an agent could communicate with an other agent (to be human or software) that using the expressions in a language mutual come to an agreement. Complete syntactic and the semantic analysis of the expressions is necessary to extract the complete meant one of the utterances and is possible because the expressions short and are limited to a dominion.

A fordítás azért nehéz, mert általában a szövegek mély megértését igényli, amihez pedig a közölt szituáció mély megértése szükséges. Ez még nagyon egyszerű szövegek esetén is fennáll, sőt egyszavas „szövegek" esetén is. Vegyük a „Open" szót egy üzlet ajtaján.[18] Ez azt az információt közvetíti, hogy az üzlet pillanatnyilag fogad vevőket. Most vegyük ugyanezt a szót egy újonnan felépített üzleten található hatalmas transzparensen. Ez azt jelenti, hogy az üzlet megnyílt, de az olvasók nem éreznék becsapva magukat, ha éjszaka a transzparens eltávolítása nélkül lenne zárva a bolt. A két felirat ugyanazt a szót használja különböző jelentések közvetítésére. Német nyelvű országban a felirat az ajtón „Offen", míg a transzparensen „Neu eröffnet" lenne.[19]

Az a probléma, hogy az egyes nyelvek eltérő módon kategorizálják a világot. Például a francia „doux" szó jelentések széles körét fogja át, amely megközelítőleg a következő angol szavakkal adható meg: „soft", „sweet", „gentle".[20] Hasonlóképpen az angol „hard" szó gyakorlatilag a német „hart" szó összes jelentését (fizikailag ellenálló, durva) lefedi, emellett a „schwierig" („nehéz", fáradságos, bonyolult értelemben) szó néhány jelentését is magában foglalja. A német „heilen" (meggyógyít) ige az angol „cure" (gyógyít) szó orvosi használatát fedi le, továbbá a „heal" (meggyógyul) tranzitív és intranzitív jelentéseit. Emiatt az adott mondat jelentésének ábrázolása sokkal nehezebb a fordítás, mint az egynyelvi megértés esetén. Egy egynyelvű elemzőrendszer használhatna olyan predikátumokat, mint $Open(x)$, azonban a fordítás céljából a reprezentációs nyelvnek esetleg több megkülönböztetést kellene tennie, például az „Offen" jelentést reprezentáló $Open_1(x)$ predikátummal és a „Neu eröffnet" jelentést reprezentáló $Open_2(x)$ predikátummal. Azt a reprezentációs nyelvet, amely a lefedett nyelvekhez szükséges összes megkülönböztetést lehetővé teszi, **köztes nyelv**nek (**interlingua**) nevezzük.

Folyékony fordításhoz az szükséges, hogy a fordító (ember vagy gép) elolvassa az eredeti szöveget, megértse a szituációt, amiről szól, és találjon egy megfelelő szöveget a célnyelvben, amely jól leírja ugyanazt, vagy egy hasonló szituációt. Ez gyakran választást jelent. Például amennyiben az angol „you" szó egy személyre vonatkozik, akkor vagy a magyar hivatalos „ön" vagy a közvetlen „te" alakra fordítható. Egyszerűen nem lehet a „you" fogalomról beszélni magyarul anélkül, hogy eldöntenénk, hogy hivatalos vagy közvetlen módon szeretnénk használni. A fordítók (mind a gépi, mind az emberi) néha nehezen tudják ezt a döntést meghozni.

4.1. Gépi fordító rendszerek

A gépi fordító rendszerek erősen különböznek abban, hogy milyen szinten elemzik a szöveget. Egyes rendszerek a bemeneti szöveget egészen a köztes nyelv szintjéig próbálják elemezni (mint ahogy azt a 22. fejezetben tettük), majd a célnyelven ezen reprezentáció alapján generálnak mondatokat. Ez azért nehéz, mert részproblémaként tartalmazza a teljes nyelvi megértés problémáját, amihez még a köztes nyelv kezeléséből

[17] A fejezet első bekezdésének eredeti angol szövege: *In Chapter 22, we saw how an agent could communicate with another agent (human or software), using utterances in a common language. Complete syntactic and semantic analysis of the utterances is* necessary *to extract the full meaning of the utterances, and is* possible *because the utterances are short and restricted to a limited domain.* Látható, hogy a két angol nyelvű szöveg között jelentős eltérések vannak, azonban a gépi fordítás eredményeként kapott szöveg nagyjából érthető. (*A ford.*)

[18] Martin Kay példája.

[19] Nyitva – Kinyitottunk (*A ford.*)

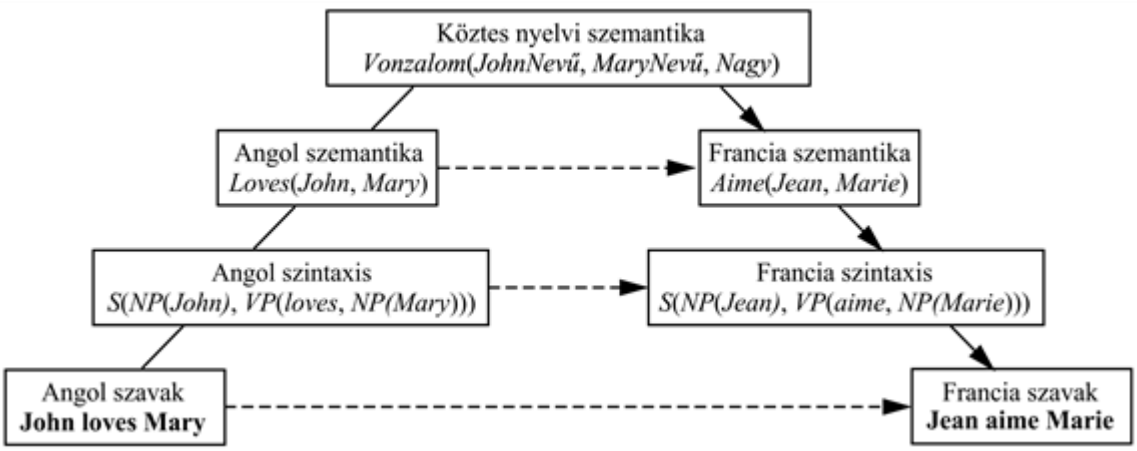
[20] doux – édes, enyhe, lágy, soft – lágy, sweet – édes, gentle – finom, gyengéd (*A ford.*)

Created by XMLmind XSL-FO Converter.

fakadó nehézségek is hozzáadódnak. A megközelítés azért törékeny, mert ha az elemzés sikertelen, akkor nincs kimenet. Azonban az az előnye, hogy a rendszerben nincs olyan komponens, amelynek két (természetes) nyelvet egyszerre kell ismernie. Ez azt jelenti, hogy köztes nyelv használatával n nyelv közti fordítás $O(n)$ és nem $O(n^2)$ nehézségű.

Más rendszerek az **átvitel**en (**transfer**) alapulnak. Fordítási szabályok (vagy példák) adatbázisát tartalmazzák, és amikor egy szabály (vagy példa) illeszkedik, akkor közvetlenül fordítanak. Az átvitel lexikai, szintaktikai vagy szemantikai szinten történhet. Például egy szigorúan szintaktikai szabály az angol [*Melléknév Főnév*] szekvenciát a francia [*Főnév Melléknév*] szekvenciába képezi le. Egy kevert szintaktikai és lexikai szabály a francia [S_1 „et puis” S_2] szekvenciát az angol [S_1 „and then” S_2] szekvenciába képezi le.[21] Azt az átvitelt, amely egy mondatot közvetlenül egy másikba visz át, **memóriaalapú fordítás**nak (**memory-based translation**) nevezünk, mivel nagyszámú (angol, francia) pár memorizálására támaszkodik. Az átviteli módszer robusztus, mert minden esetben generál *valamilyen* kimenetet, és legalább a szavak egy része feltétlenül helyes. A 23.5. ábra bemutatja az egyes átviteli pontokat.

23.5. ábra - Egy gépi fordító rendszer választási lehetőségeit ábrázoló sematikus diagram. A fenn található angol szöveggel kezdjük. A köztes nyelven alapuló rendszer a folytonos vonalakat követi, az angol szöveget először szintaktikai elemzésnek veti alá, majd szemantikai és köztes nyelvi reprezentációt állít elő, végül szemantikai, szintaktikai és lexikai formákon keresztül francia szöveget állít elő. Az átvitelalapú rendszerek a szaggatott vonalak által jelölt rövidzárakat használják. Az egyes rendszerek eltérő szinteken végzik az átvitelt, egyes rendszerek több szinten is.



4.2. Statisztikai gépi fordítás

Az 1960-as évek elején nagy reményeket fűztek ahhoz, hogy a számítógépek képesek egyik természetes nyelvből a másikba fordítani csakúgy, mint ahogy Turing projektje képes volt kódolt német üzeneteket értelmes német szövegbe fordítani. 1966-ra világossá vált, hogy a folyékony fordítás igényli az üzenet jelentésének a megértését, míg a kódfejtés nem.

Az elmúlt évtizedben elmozdulás volt megfigyelhető a statisztikai alapú gépi fordító rendszerek irányába. Természetesen, a 23.5. ábra bármely lépésének javára szolgálnának statisztikai adatok, valamint egy olyan egyértelmű valószínűségi modell használata, amely megadja, hogy mi egy jó analízis vagy átvitel. Azonban a „statisztikai gépi fordítás” az egész fordítási probléma olyan megközelítésének a megnevezésévé vált, amely a mondat legvalószínűbb fordításának kétnyelvű korpuszon alapuló megtalálását jelenti. A kétnyelvű korpuszok egyik példája a **Hansard**,[22] amely parlamenti viták naplója. Kanada, Hongkong és más országok kétnyelvű Hansardokat tesznek közzé, az Európai Unió 11 nyelven[23] publikálja hivatalos dokumentumait, míg az Egyesült Nemzetek Szervezete többnyelvű dokumentumokat ad ki. Ezek a statisztikai gépi fordítás számára felbecsülhetetlenül értékes forrásoknak bizonyultak.

[21] Mindkét esetben magyarul [S_1 „és utána” S_2] (*A ford.*)
[22] William Hansard után elnevezve, aki 1811-ben elsőként publikálta a brit parlamenti vitákat.
[23] Az angol nyelvű kiadás idején. (*A ford.*)

Created by XMLmind XSL-FO Converter.

Egy angol nyelvű mondat (E) például francia[24] (F) mondatra történő fordításának problémáját a Bayes-szabály következő alkalmazásaként írhatjuk le:

$$\begin{aligned}\operatorname{argmax}_F P(F|E) &= \operatorname{argmax}_F P(E|F)P(F)/P(E)\\ &= \operatorname{argmax}_F P(E|F)P(F)\end{aligned}$$

Ez a szabály azt állítja, hogy minden lehetséges F francia mondatot figyelembe kell vennünk, és azt kell választanunk, amelyik maximalizálja $P(E|F)P(F)$-t. A $P(E)$ tényezőt nem kell figyelembe venni, hiszen minden F esetén azonos. A $P(F)$ tényező a francia **nyelvi modell** (**language model**), azt adja meg, hogy milyen valószínűségű egy adott francia mondat. A $P(E|F)$ a **fordítási modell** (**translation model**), azt adja meg, hogy milyen valószínűségű egy adott angol mondat, mint az adott francia mondat fordítása.

Az ügyes olvasók bizonyára csodálkoznak, hogy mit nyertünk a $P(F|E)$ definiálásával a $P(E|F)$ segítségével. A Bayes-szabály más alkalmazásainál azért tettük ezt, mert kauzális modellt akartunk használni. Például a $P(Szimptómák|Betegség)$ kauzális modellt használtuk $P(Betegség|Szimptómák)$ kiszámolására. A fordítás esetében azonban egyik irány sem kauzálisabb, mint a másik. Jelen esetben a Bayes-szabály alkalmazásának az az oka, hogy azt hisszük, képesek leszünk egy olyan $P(F)$ nyelvi modellt megtanulni, amely pontosabb, mint a $P(E|F)$ fordítási modell (és pontosabb, mint $P(F|E)$ közvetlen becslése). Lényegében a problémát két részre osztottuk: először a $P(F|E)$ fordítási modellt alkalmazzuk, hogy olyan francia mondatokat találjunk, amelyek visszaadják az angol mondat lényegét, de amelyek nem feltétlenül folyékony francia mondatok, majd a $P(F)$ nyelvi modellt (amelyre sokkal jobb valószínűségi becslésünk van) használjuk a legjobb jelölt kiválasztására.

A $P(F)$ **nyelvi modell** bármilyen olyan modell lehet, amely egy mondathoz valószínűséget rendel. *Nagyon* nagy korpusz esetén $P(F)$-et közvetlenül becsülhetnénk az egyes mondatok korpuszbeli előfordulási száma alapján. Például ha a webről összegyűjtünk 100 millió francia mondatot, és a „Clique ici”[25] mondat 50 ezerszer fordul elő, akkor $P(\text{Clique ici}) = 0{,}0005$. Azonban még 100 millió példa esetén is a legtöbb mondat előfordulási száma nulla lenne.[26] Emiatt az ismert bigram modellt fogjuk alkalmazni, amelyben az $f_1 \ldots f_n$ szavakból álló francia nyelvű mondat valószínűsége:

$$P(f_1 \cdots f_2) = \prod_{i=1}^{n} P(f_i | f_{i-1})$$

Ismernünk kell az olyan bigram valószínűségeket, mint például a $P(\text{Eiffel}|\text{tour}) = 0{,}02$.[27] Ez a szintaxis mindössze nagyon lokális jellemzőit képes leírni, ahol a szó csak az őt megelőző szótól függ. A nyersfordításhoz azonban többnyire ez is elegendő.[28]

A $P(E|F)$ **fordítási modell**t (**translation model**) nehezebb meghatározni. Egyrészt nem áll rendelkezésünkre (angol, francia) mondatpárokból álló kész gyűjtemény, amely alapján taníthatnánk. Másrészt a modell komplexitása nagyobb, mivel mondatok keresztszorzatára, és nem pedig különálló mondatokra alapul. Egy túlzottan leegyszerűsített fordítási modellel fogunk kezdeni, és felépítünk valamit, ami az „IBM Model 3”-at (Brown és társai, 1993) közelíti, ami továbbra is a végletekig leegyszerűsítettnek tűnik, azonban az esetek körülbelül felében elfogadható fordításokat generált.

A végletekig leegyszerűsített modell arról szól, hogy „a mondat fordításához egyszerűen fordítsuk le a szavakat egyesével és egymástól függetlenül, balról jobbra”. Ez egy unigram szóválasztási modell. Lehetővé teszi, hogy egyszerűen kiszámítsuk egy fordítás valószínűségét:

$$P(E|F) = \prod_{i=1}^{n} P(E_i | F_i)$$

Néhány esetben ez a modell jól működik. Vegyük például a következőt:

[24] Ebben a bekezdésben az angolról francia nyelvre történő fordítás problémájával foglalkozunk. Ne zavarja az olvasót, hogy a Bayes-szabály alkalmazása a $P(E|F)$, és nem a $P(F|E)$ figyelembe vételéhez vezet, ami annak tűnik, mintha franciáról fordítanánk angolra.

[25] Kattints ide (*A ford.*)

[26] Amennyiben csak 100 ezer szó fordulna elő a lexikonban, akkor az összes lehetséges háromszavas mondatok 99,99999%-ának előfordulási száma nulla lenne a 100 milliós korpuszban. Hosszabb mondatok esetén még rosszabb a helyzet.

[27] Eiffel-torony *(A ford.)*

[28] Világos, hogy a fordítás finomabb részleteihez $P(f_i|f_{i-1})$ nem elegendő. Híres példaként álljon itt Marcel Proust 3500 oldalas regénye, a „A la récherche du temp perdu”, amely ugyanazzal a szóval kezdődik és végződik, így egyes fordítók úgy döntöttek, hogy ugyanezt teszik, azaz egy szó fordítását egy olyan szóra alapozták, amely durván 2 millió szóval korábban fordult elő.

Created by XMLmind XSL-FO Converter.

$$P(\text{the dog}|\text{le chien}) = P(\text{the}|\text{le}) \times P(\text{dog}|\text{chien})$$

Bármely elfogadható valószínűségi értékhalmaz esetén a „the dog”[29] lenne a „le chien” maximum-likelihood becslője. A legtöbb esetben azonban a modell megbukik. Az egyik probléma a szórend. Franciában a „dog” megfelelője a „chien”, a „brown”-é a „brun”, azonban a „brown dog”-é a „chien brun”.[30] A másik probléma az, hogy a szóválasztás nem egy az egyes leképezés. Az angol „home” szót gyakran „à la maison”-nak[31] fordítják, ami egy hármas leképezés (illetve három az egybe a másik irányban). Ezen problémák ellenére az IBM Model 3 makacsul ragaszkodik az alap unigram modellhez, bár, hogy javítson rajta, hozzátesz néhány kiegészítést.

A modell, hogy képes legyen kezelni azt a tényt, hogy a szavakat nem egy az egyben fordítjuk, bevezeti a szó **termékenység**ének (**fertility**) fogalmát. Egy *n* termékenységű szót *n*-szer lemásolja, és az *n* másolat mindegyike függetlenül fordítódik. A modell az összes francia szóra tartalmazza a $P(\textit{termékenység} = n|\textit{szó})$ paramétereket. Az „à la maison” „home”-ra történő fordításához a modell 0 termékenységet választana az „à” és „la” szavakra,[32] míg egyet a „maison”-ra, és utána az unigram modellt alkalmazná a „maison” szó „home”-ra történő fordításához. Ez eléggé indokoltnak tűnik, az „à” és a „la” kis információtartalmú szavak, amelyeket ésszerű üres karakterfüzérré fordítani. A másik irányba történő fordítás már kétségesebb. A „home” szóhoz hármas termékenységet rendelne a modell, „home home home” szekvenciát kapva. Az első „home” „à”-ra, a második „la”-ra, míg a harmadik „maison”-ra fordulna. A fordítási modell szempontjából az „à la maison” és a „maison à la” pontosan azonos valószínűséget kapna. (Pont ez a kétséges rész.) A nyelvi modellre lenne bízva, hogy eldöntse, melyik a jobb. Értelmesebbnek tűnhet a „home”-ot közvetlenül „à la maison”-ra fordítani, mint közvetett módon a „home home home”-on keresztül, azonban ehhez sokkal több paraméterre lenne szükség, amelyeket nehéz lenne a rendelkezésre álló korpuszból meghatározni.

A fordítási modell végső része a szavak megfelelő sorrendbe történő permutálása. Ez egy eltolási modellel történik, amely során a szó az eredeti pozíciójából a végleges pozícióba mozog. Például a „chien brun” „brown dog”-ra történő fordítása során a „brown” szó +1 eltolási értéket kap (azaz jobbra egy pozíciót mozog), míg a „dog” –1 értéket. Az olvasó elképzelheti, hogy az eltolásnak függnie kellene a szótól: az olyan melléknevek, mint a „brown” többnyire pozitív eltolási értékkel rendelkeznének, mert a francia nyelv általában a főnév után helyezi el a mellékneveket. Azonban az IBM Model 3 úgy döntött, hogy a szótól függő eltolás túl sok paramétert igényelne, ezért az eltolás független a szótól, és csak a mondatbeli pozíciótól és a mondat mindkét nyelvbeli hosszától függ. Azaz a modell a következő paramétereket becsüli:

$$P(\textit{Eltolás} = o|\textit{Pozíció} = p, \textit{AngolHossz} = m, \textit{FranciaHossz} = n)$$

Azaz a „brown” szó „brown dog”-beli eltolásának meghatározásához megnézzük a $P(\textit{Eltolás}|1,2,2)$ értékét, amely mondjuk +1 értéket adna 0,3, és 0 értéket 0,7 valószínűséggel. Az eltolási modell még *kétségesebbnek* látszik, olyan, mintha egy olyan személy agyalta volna ki, aki sokkal jobban ért a mágnesbetűk hűtőn történő tologatásához, mint a valódi természetes nyelven történő beszédhez. Rövidesen bemutatjuk, hogy nem azért tervezték ilyen módon, mert jól modellezi a természetes nyelvet, hanem azért, mert a rendelkezésre álló adatokat ésszerűen hasznosítja. Mindenesetre arra szolgál, hogy élénken emlékeztessen bennünket arra, hogy egy középszerű fordítási modellt meg lehet menteni egy jó francia nyelvi modellel. Íme, egy példa egy mondat fordításának lépéseire.[33]

Forrás francia:	Le	chien	brun	n’	est	pas	allé	à	la	maison
Termékenységi modell:	1	1	1	1	1	0	1	0	0	1
Transzformált francia:	Le	chien	brun	n’	est		allé			maison
Szóválasztási modell	The	dog	brown	not	did		go			home
Eltolási modell:	0	+1	–1	+1	–1		0			0
Cél angol:	The	brown	dog	did	not		go			home

Most már tudjuk, hogyan kell kiszámítani a $P(F|E)$ valószínűséget bármely (francia, angol) mondatpárra. Azonban amit igazából tenni akarunk az az, hogy egy adott angol mondat esetén megtaláljuk azt a francia mondatot, amely maximalizálja ezt a valószínűséget. Nem sorolhatunk fel egyszerűen mondatokat: 10^5 francia szó esetén 10^{5n} *n* hosszúságú mondat lehetséges, és számos összerendelés mindegyikre. Még ha csak a 10 leggyakoribb szó-szó fordítást tekintjük minden egyes szóra, és csak 0 és ±1 eltolást veszünk figyelembe, akkor

[29] a kutya (*A ford.*)
[30] barna kutya (*A ford.*)
[31] home – otthon, à la maison – otthon lenni, haza menni értelemben (*A ford.*)
[32] „à” – nagyjából a magyar -ban, -ben, -ba, -be ragoknak felel meg, „la” – nőnemű határozott névelő (*A ford.*)
[33] A barna kutya nem ment haza. (*A ford.*)

Created by XMLmind XSL-FO Converter.

is $2n!10^n$ mondatot kapunk, ami azt jelenti, hogy $n = 5$ esetén még fel tudnánk őket sorolni, de $n = 10$ esetén már nem. Ehelyett meg kell *keresnünk* a legjobb megoldást. Az A^* keresés hatékonynak bizonyult a feladatra (Germann és társai, 2001).

4.3. Valószínűségek tanulása gépi fordításhoz

Körvonalaztunk egy olyan $P(F|E)$ modellt, amely négy paraméterhalmazt tartalmaz:

Nyelvi modell: $P(szó_i | szó_{i-1})$

Termékenységi modell: $P(Termékenység = n | szó_F)$

Szóválasztási modell: $P(szó_E | szó_F)$

Eltolási modell: $P(Eltolás = o | pozíció, hossz_E, hossz_F)$

Ennek a modellnek még egy 1000 szót tartalmazó, szerény szókincs esetén is paraméterek millióira van szüksége. Nyilvánvalóan adatokból kell megtanulnunk őket. Feltételezzük, hogy az egyetlen rendelkezésünkre álló adat egy kétnyelvű korpusz. Lássuk a felhasználási módját:

Mondatokra történő szegmentáció: a fordítási egység a mondat, tehát a korpuszt mondatokra kell tördelnünk. A pont a mondatvég jellemző indikátora, de vegyük a „Dr. J. R. Smith of Rodeo Dr. megérkezett.” példát, amelyben csak az utolsó pont jelenti a mondat végét. A mondatokra történő szegmentációt körülbelül 98%-os pontossággal lehet elvégezni.

A $P(szó_i|szó_{i-1})$ **francia nyelvi modell becslése:** vegyük a korpusz francia nyelvű részét, számoljuk meg a szópárok előfordulási gyakoriságát, és végezzünk simítást, hogy megkapjuk a $P(szó_i|szó_{i-1})$ becslőjét. Például megkaphatjuk, hogy P(Eiffel|tour) = 0,02.

Mondatok illesztése: minden egyes angol nyelvű mondatra határozzuk meg az(oka)t a francia nyelvű mondato(ka)t, amely(ek) megfelel(nek) neki a francia változatban. Általában a szövegben következő angol mondat megfelel a következő francia mondatnak 1:1 megfeleltetéssel, azonban néha előfordulnak variációk: az egyik nyelvű mondatot 2:1 megfeleltetésbe vágjuk, vagy két mondat sorrendjét felcseréljük, 2:2 megfeleltetést kapva. Mindössze a mondathosszakat tekintve lehetséges az illesztésük (1:1, 1:2, 2:2 stb.) 90% és 99% közötti pontossággal, a Viterbi szegmentációs algoritmus (lásd 23.1. ábra) megfelelő variációjával. Még jobb illeszkedés is elérhető amennyiben olyan iránypontokat használunk, amelyek mindkét nyelvben közösek, mint például számok, tulajdonnevek, illetve olyan szavak, melyeknek – egy kétnyelvű szótár alapján – tudjuk, hogy egyértelmű a fordításuk.

A $P(Termékenység = n|szó_F)$ **kezdeti termékenységi modell becslése:** legyen adott egy m hosszúságú francia mondat, amely egy n hosszúságú angol mondathoz illeszkedik, ekkor vegyük ezt annak alátámasztásaként, hogy mindegyik (a mondatban előforduló) francia szó termékenysége n/m. Vegyük figyelembe ezeket az összes mondatra az egyes szavak termékenységi eloszlásának meghatározásához.

A $P(szó_E|szó_F)$ **kezdeti szóválasztási modell becslése:** vegyük az összes olyan mondatot, amely tartalmazza például a „brun” szót. Azok a szavak, amelyek a leggyakrabban fordulnak elő az illeszkedő angol mondatokban, valószínűleg a „brun” szó-szó fordításai.

A $P(Eltolás = o|pozíció, hossz_E, hossz_F)$ **kezdeti eltolási modell becslése:** most, hogy rendelkezünk egy szóválasztási modellel, használjuk fel azt az eltolási modell becslésére. Egy n hosszúságú angol mondat esetén, amely egy m hosszúságú francia mondathoz illeszkedik, vegyük minden egyes francia szót (i pozícióban) és minden olyan angol szót (j pozícióban), amely valószínű választás lenne a francia szóra, és vegyük ezt $P(Eltolás = i - j|i, n, m)$ bizonyítékául.

A becslések javítása: használjuk az EM (elvárásmaximalizáló) algoritmust a becslések javítására. A rejtett változó az illeszkedő mondatpárok közti **szóilleszkedési vektor** (**word alignment vector**). A vektor megadja minden egyes angol szóra a megfelelő francia szó francia mondatbeli pozícióját. Például megkaphatjuk a következőt:

Created by XMLmind XSL-FO Converter.

Forrás francia:	Le	chien	brun	n'	est	pas	allé	à	la	maison
Cél angol:	The	brown	dog	did	not		go			home
Szóilleszkedés:	1	3	2	5	4		7			10

Először a paraméterek aktuális becslőit használva elkészítjük a szóilleszkedési vektort mindegyik mondatpárra. Ez jobb becslő megalkotását fogja számunkra lehetővé tenni. A termékenységi modellt az alapján becsüljük, hogy a szóilleszkedési vektor egy adott tagja hányszor képződik le több szóra, vagy egyetlenegyre sem. A szóválasztási modellnek a mondat összes szava helyett most már csak olyan szavakat kell figyelembe vennie, amelyek egymáshoz lettek illesztve. Az eltolási modell a mondat egyes pozícióinak elmozdulását vizsgálja a szóilleszkedési vektornak megfelelően. Sajnos nem tudjuk, hogy mi a helyes illeszkedés, és túl sok van belőlük ahhoz, hogy mindet felsoroljuk. Emiatt arra vagyunk kényszerítve, hogy megkeressünk néhány nagy valószínűségű illeszkedést, és a valószínűségükkel súlyozzuk őket, miközben bizonyítékot gyűjtünk az új paraméterbecslőkhöz. Mindössze ennyire van szükségünk az EM algoritmushoz. A kezdeti paraméterek alapján illeszkedéseket számolunk, majd az illeszkedések alapján javítjuk a paraméterbecslőket. Ismételjük, amíg konvergál.

5. Összefoglalás

A fejezet főbb pontjai a következők:

- Az *n*-gram alapú valószínűségi nyelvi modellek meglepő mennyiségű információt képesek visszaadni egy nyelvről.
- A CFG-ket ki lehet egészíteni valószínűségi CFG-kre, könnyebbé téve az adatok alapján történő tanulásukat és a kétértelműségek feloldását.
- Az **információkereső** (**information retrieval**) **rendszerek** egy nagyon egyszerű, szózsákokra épülő nyelvi modellt használnak, ennek ellenére jó **felidézés**i (**recall**) és **pontosság**i (**precision**) mutatókkal rendelkeznek nagyon nagy korpuszokon.
- Az **információkinyerő** (**information extraction**) **rendszerek** bonyolultabb modellt alkalmaznak, amely korlátozottan figyelembe veszi a szintaxist és a szemantikát. Gyakran véges automaták kaszkádjával valósítják meg őket.
- A **gépi fordító** (**machine translation**) **rendszerek**et különböző technikák alkalmazásával valósították meg, a teljes szintaktikai és szemantikai analízistől kezdve egészen a szógyakoriságon alapuló statisztikai módszerekig.
- Egy statisztikai nyelvi rendszer építése során az a legjobb, ha olyan rendszert gondolunk ki, amely a rendelkezésre álló adatokat jól hasznosítja, még akkor is, ha a modell a végletekig leegyszerűsítettnek tűnik.

5.1. Irodalmi és történeti megjegyzések

Az *n*-gram betűmodelleket nyelvi modellezésre Markov javasolta (Markov, 1913). Claude Shannon generált elsőként *n*-gram szómodelleket az angol nyelvre (Shannon és Weaver, 1949). Chomsky mutatta meg a véges állapotú modellek korlátait a környezetfüggetlen modellekhez képest, az alábbi következtetést levonva: „Valószínűségi modellek nem adnak részletes bepillantást a szintaktikai struktúrák egyes alapvető problémáiba" (Chomsky, 1956; 1957). Ez igaz, azonban figyelmen kívül hagyja azt a tényt, hogy a valószínűségi modellek *lehetővé teszik* a bepillantást néhány *más* alapvető problémába, olyanokba, amelyekkel a CFG-k nem foglalkoznak. Chomsky észrevételeinek olyan sajnálatos hatása volt, hogy két évtizeden keresztül sokakat elijesztett a statisztikai modellektől egészen addig, míg ezek a modellek újra megjelentek a beszédfelismerésben (Jelinek, 1976).

Az adj-hozzá-egyet simítás Jeffreystől származik (Jeffreys, 1948), míg a törölt interpolációs simítás Jelinektől és Mercertől, akik beszédfelismerésre használták (Jelinek és Mercer, 1980). További technikákra példa a Witten–Bell-simítás (Witten–Bell, 1991) és a Good–Turing-simítás (Church és Gale, 1991). Az utóbbit gyakran használják bioinformatikai problémáknál is. A biostatisztika és a valószínűségi természetes nyelv feldolgozás (NLP) közelednek egymáshoz, mivel mindkettő alkotóelemek ábécéjéből felépülő hosszú, strukturált szekvenciákkal foglalkozik.

Created by XMLmind XSL-FO Converter.

Az egyszerű *n*-gram betű- és szómodellek nem az egyetlen lehetséges valószínűségi modellek. Blei és társai leírják a **rejtett Dirichlet-allokáció**nak (**latent Dirichlet allocation**) nevezett valószínűségi szövegmodellt, amely a szövegeket témák keverékének tekinti, amelyben mindegyik téma saját szóeloszlással rendelkezik (Blei és társai, 2001). Ez a modell a **rejtett szemantikai indexelés (latent semantic indexing)** modell (Deerwester és társai, 1990) (lásd még (Papadimitriou és társai, 1998)) kibővítésének és racionalizálásának tekinthető, valamint Sahami és társai többszörös okkeverék (cause mixture) modelljéhez is kötődik (Sahami és társai, 1996).

A **valószínűségi környezetfüggetlen nyelvtan**ok (**PCFG**) megválaszolják Chomsky valószínűségi modellekkel kapcsolatos összes ellenvetését, és a CFG-khez képest előnyökkel rendelkeznek. A PCFG-ket Booth (Booth, 1969) és Salomaa (Salomaa, 1969) vizsgálták. Jelinek bemutatja a veremdekódoló algoritmust, amely a Viterbi-keresés olyan variációja, amely arra használható, hogy megtalálja egy PCFG-vel a legvalószínűbb elemzést (Jelinek, 1969). Baker vezette be a belső–külső algoritmust (Baker, 1979), Lari és Young leírta használhatóságát és korlátait (Lari és Young, 1990). Charniak (Charniak, 1996), valamint Klein és Manning (Klein és Manning, 2001) a **treebank** nyelvtanokkal történő elemzést tárgyalják. Stolcke és Omohundro megmutatták, hogyan lehet nyelvtani szabályokat tanulni Bayes-modellek egyesítésével (Stolcke és Omohundro, 1994). További PCFG-algoritmusokat mutatott be Charniak (Charniak, 1993), valamint Manning és Schütze (Manning és Schütze, 1999). Collins a terület áttekintő tanulmányát kínálja, továbbá az egyik legsikeresebb statisztikai elemzőprogram magyarázatát (Collins, 1999).

Sajnos a PCFG-k rosszabbul teljesítenek, mint az egyszerű *n*-gram modellek számos feladat esetén, mert a PCFG-k nem képesek reprezentálni az egyes szavakhoz társuló információkat. Hogy kijavítsák ezt a hiányosságot, számos szerző (Collins, 1996; Charniak, 1997; Hwa, 1998) bevezették a **szókinccsel ellátott valószínűségi nyelvtan**ok (**lexicalized probabilistic grammar**) különböző verzióit, amelyek kombinálják a környezetfüggetlen és a szóalapú statisztikákat.

A Brown-korpusz volt az első próbálkozás, hogy tapasztalati nyelvészeti célokra kiegyensúlyozott korpuszt gyűjtsön (Francis és Kucera, 1967). Körülbelül egymillió szót tartalmazott, szófaji információval ellátva. Eredetileg 100 ezer lyukkártyán tárolták. A Penn treebank körülbelül 1,6 millió szó gyűjteménye, kézi elemzéssel fákba rendezve. Elfér egy CD-n. A brit nemzeti korpusz (British National Corpus) kibővíti ezt 100 millió szóra (Leech és társai, 2001). A világháló több mint egybillió szót tartalmaz, több mint 10 millió szerveren tárolódik.

Az **információkeresés (informational retrieval)** területe iránti érdeklődés újra nő, amelyet az internetes keresés széles körű elterjedése inspirál. Robertson egy korai áttekintést ad, és bevezeti a valószínűségi rangsorolási elvet (Robertson, 1977). Manning és Schütze az NLP statisztikai megközelítésének kontextusában tárgyalja röviden az IR-t (Manning és Schütze, 1999). Baeza-Yates és Ribeiro-Neto általános célú áttekintést ad (Baeza-Yates és Ribeiro-Neto, 1999), helyettesítve az olyan régi klasszikusokat, mint Salton és McGill (Salton és McGill, 1983), valamint Frakes és Baeza-Yates (Frakes és Baeza-Yates, 1992). A *Gigabájtok kezelése* (Managing Gigabytes) c. könyv pont azt teszi, amit a címe mond: elmagyarázza hogyan lehet hatékonyan indexelni, tömöríteni és lekérdezni gigabájtos méretű korpuszokat (Witten és társai, 1999). Az amerikai kormány Nemzeti Szabványügyi és Technológiai Intézete (National Institute of Standards and Technology, NIST) által szervezett TREC konferencia minden évben megrendezi az IR-rendszerek versenyét, és kiadványában publikálja az eredményeket. A verseny első hét évében a részt vevő rendszerek teljesítménye körülbelül megduplázódott.

A legnépszerűbb IR-modell a **vektortér modell** (**vector space model**) (Salton és társai, 1975). Salton munkája uralta a terület korai éveit. Két alternatív valószínűségi modell létezik. Mi a Ponte és Croft munkáján alapulót mutattuk be (Ponte és Croft, 1998). A $P(D, Q)$ közös valószínűségi eloszlást $P(Q|D)$ segítségével modellezi. Egy alternatív modell (Maron és Kuhns, 1960; Robertson és Sparck Jones, 1976) a $P(D|Q)$-t használja. Lafferty és Zhai megmutatta, hogy a modellek ugyanazon a közös valószínűségi eloszláson alapulnak, azonban a modellválasztásnak következményei vannak a paramétertanításra (Lafferty és Zhai, 2001). Az általunk bemutatottak az ő munkájukon alapul. Turtle és Croft összehasonlítják a különféle IR-modelleket (Turtle és Croft, 1992).

Brin és Page leírják egy világhálón kereső gép implementációját, taglalva a `PAGERANK` algoritmust is, mely a dokumentumok minőségének weblinkek elemzésén alapuló lekérdezésfüggetlen mértéke (Brin és Page, 1998). Kleinberg leírja, hogyan lehet hiteles forrásokat találni a weben hivatkozáselemzéssel (Kleinberg, 1999). Silverstein és társai megvizsgáltak egy egymilliárd webkeresést tartalmazó logot (Silverstein és társai, 1998). Kukich felmérte a helyesírási hibák javításának irodalmát (Kukich, 1992). Porter leírja a klasszikus szabályalapú szótövesítő algoritmust (Porter, 1980), míg Krovetz egy szótáralapú verziót ír le (Krovetz, 1993).

Created by XMLmind XSL-FO Converter.

Manning és Schütze a dokumentumosztályozás és klaszterezés alapos áttekintését adják (Manning és Schütze, 1999). Joachims statisztikai tanulási elméletet és szupport vektor gépeket alkalmazott annak elméleti vizsgálatára, hogy mikor sikeres egy osztályozás (Joachims, 2001). Apté és társai a Reuters-hírek „Jövedelmek" kategóriába történő osztályozásánál 96%-os pontosságot publikált (Apté és társai, 1994). Koller és Sahami 95%-os pontosságot ért el egy naiv Bayes-osztályozóval, és akár 98,6%-ot is, ha olyan Bayes-osztályozót alkalmazott, amely bizonyos jellemzők közti összefüggéseket is figyelembe vett (Koller és Sahami, 1997). Lewis áttekinti a Bayes-osztályozók negyvenéves alkalmazását szövegosztályzásra és visszakeresésre (Lewis, 1998).

Az *Information Retrieval* újság és az éves *SIGIR* konferenciakiadványai a terület friss fejleményeit fedik le.

A korai információkinyerő programok közé tartozik a GUS (Bobrow és társai, 1977) és a FRUMP (DeJong, 1982). A modern információkinyerő rendszerek egyes tervezési részletei az 1970–1980-as évekbeli szemantikai nyelvtanokkal foglalkozó kutatásokra vezethetők vissza. Például egy szemantikai nyelvtanokon alapuló repülőgép-foglaló rendszer interfésze olyan kategóriákkal rendelkezne, mint a *Hely* és a *Repül*, az *NP*-k és a *VP*-k helyett. Birnbaum és Selfridge munkája egy szemantikai nyelvtanon alapuló rendszer megvalósítását mutatja be (Birnbaum és Selfridge, 1981).

Az újabb keletű információkinyeréssel foglalkozó munkákat az amerikai kormány által szponzorált Message Understand Conference (MUC) konferenciák mozgatták. A Fastus-rendszert Hobbs és társai készítették (Hobbs és társai, 1997), és az a publikációgyűjtemény, amely bemutatja, további véges állapotú modelleket alkalmazó rendszereket is felsorol (Roche és Schabes, 1997).

Az 1930-as években Petr Troyanskii beadott egy „fordítógépre" vonatkozó szabadalomkérelmet, azonban még nem állt rendelkezésre számítógép ötleteinek megvalósítására. 1947 márciusában a Rockefeller Alapítványnak dolgozó Warren Weaver az írta Norbert Weinernek, hogy a gépi fordítás megvalósítható lehet. A kriptográfiai és információelméleti munkákra alapozva Weaver azt írta, hogy: „Amikor egy orosz nyelvű cikket nézek, azt mondom, hogy »Ezt igazából angolul írták, azonban furcsa szimbólumokkal kódolták. Most pedig nekiállok a dekódolásnak.«" A következő évtizedben a közösség ilyen módon próbálta a dekódolást. Az IBM 1954-ben bemutatott egy kezdetleges rendszert. Bar-Hillel (Bar-Hillel, 1960), valamint Locke és Booth (Locke és Booth, 1955) leírták az ebben az időszakban meglévő lelkesedést. A gépi fordításból való későbbi kiábrándulást Lindsay írta le, aki a gépi fordítás néhány olyan akadályát is bemutatta, amelyek a szintaxis és a szemantika közti kölcsönhatáshoz és a világ ismeretének szükségességéhez kötődnek (Lindsay, 1963). Az amerikai kormány csalódott az előrehaladás hiánya miatt, ezt egy jelentés akként összegezte, hogy „a gépi fordításnak nincs közvetlen vagy megjósolható jövője" (ALPAC, 1966). Ennek ellenére korlátozottan ugyan, de folytatódott a munka, és az amerikai légierő 1970-ben „hadrendbe állította" a SYSTRAN-rendszert, míg az Európai Közösség 1976-ban kezdte azt használni. A TAUM-METEO időjárásjelentés-fordító rendszer használata szintén 1976-ban indult (Quinlan és O'Brien, 1992). Az 1980-as években a rendelkezésre álló számítási teljesítmény eljutott arra a szintre, ahol az ALPAC megállapításai már nem voltak helyesek. Voorhees bemutat néhány Wordneten alapuló, közelmúltbeli fordítási alkalmazást (Voorhees, 1993). Bevezető tankönyvet Hutchins és Somers írt (Hutchins és Somers, 1992).

A statisztikai gépi fordítás Weaver 1947-es megjegyzéséig nyúlik vissza, azonban csak az 1980-as évekre jutott el gyakorlati szintre. Az általunk bemutatottak Brown és kollégáinak IBM-beli munkáin alapulnak (Brown és társai, 1988; 1993). Nagyon komoly matematikát alkalmaznak, ezért a Kevin Knight által írt kísérő oktatóanyag sokat segít a megértésben (Knight, 1999). A még frissebb statisztikai gépi fordítás területén végzett munkák továbblépnek a bigram modellen, és olyan modelleket használnak, amelyek valamennyi szintaxist is figyelembe vesznek (Yamada és Knight, 2001). A mondatszegmentáció terén Palmer és Hearst végezték a korai munkákat (Palmer és Hearst, 1994). Michel és Plamondon a kétnyelvű mondatillesztésről értekezik (Michel és Plamondon, 1996).

Két kitűnő könyv szól a valószínűségi nyelvfeldolgozásról: Charniak könyve rövid és lényegre törő (Charniak, 1993), míg Manning és Schütze munkája átfogó és naprakész (Manning és Schütze, 1999). A gyakorlati nyelvfeldolgozás terén végzett munkákat a kétévente megrendezett Applied Natural Language Processing (ANLP) és az Empirical Methods in Natural Language Processing (EMNLP) konferenciák, valamint a *Natural Language Engineering* folyóirat mutatja be. A SIGIR egy hírlevelet és egy információkeresést tárgyaló évenkénti konferenciát támogat.

5.2. Feladatok

23.1.

Created by XMLmind XSL-FO Converter.

(Jurafsky és Martin, 2000) alapján. Ebben a feladatban szerzőséget osztályozó rendszert fogunk kifejleszteni, amely egy adott szöveg esetén megpróbálja meghatározni, hogy a két pályázó szerző közül melyik írta a szöveget. Szerezzünk be szövegeket két különböző szerzőtől. Válasszuk szét őket tanító és teszthalmazokra. Tanítsunk unigram modellt mindkét szerző tanító halmaza alapján. Végül minden egyes tesztelő szövegre számítsuk ki mindkét unigram modellre a valószínűséget, és rendeljük a legvalószínűbb modellhez. Határozza meg a technika pontosságát. Tudná növelni a pontosságot további jellemzőkkel? A nyelvészet ezen részterületét a **stílusmérés tudományá**nak (**stylometry**) nevezzük, sikerei közé sorolható a Federalist Papers (Mosteller és Wallace, 1964), valamint Shakespeare néhány vitatott munkája (Foster, 1989) szerzőjének meghatározása.

23.2.

Ez a feladat az *n*-gram modellek minőségét méri fel. Keressen vagy hozzon létre körülbelül 100 ezer szavas egynyelvű korpuszt. Tördelje szavakra, és számítsa ki az egyes szavak előfordulási gyakoriságát. Hány különböző szó található benne? Ábrázolja a gyakoriságot a sorrend (leggyakoribb, második leggyakoribb stb.) függvényében log–log skálán. Határozza meg a bigramok (két egymást köbvető szó) és trigramok (három egymást követő szó) gyakoriságát is. Használja ezeket a gyakoriságokat nyelv generálására: az unigram, bigram és trigram modell alapján a gyakoriságnak megfelelő véletlen választással generáljon 100 szavas szövegeket. Hasonlítsa össze ezeket a szövegeket a ténylegesen beszélt nyelvvel. Végül számolja ki az egyes modellek összetettségét.

23.3.

Ez a feladat a spam-felismeréssel foglalkozik. A spam definíciója „kéretlen tömeges reklám e-mail”. A spam kezelése bosszantó elfoglaltság számos e-mailező számára, tehát megbízható eltávolításuk áldás lenne. Hozzon létre spam és spam-mentes e-mailekből álló korpuszt. Elemezze mindkét korpuszt, és állapítsa meg, hogy mely jellemzők lennének alkalmasok az osztályozásra (unigram, brigram, üzenethossz, feladó, érkezési idő). Ezután tanítson egy osztályozó algoritmust (döntési fa, naiv Bayes vagy más, Ön által választott algoritmus) egy tanító halmazon, majd értékelje pontosságát egy teszthalmazon.

23.4.

Készítsen öt lekérdezésből álló teszthalmazt, majd küldje el három nagy webkeresőnek. Értékelje mindegyik pontosságát az első 1, 3 és 10 dokumentumra és az átlagos reciprok rangra. Próbálja megmagyarázni az eltéréseket.

23.5.

Próbáljon megbizonyosodni arról, hogy az előbbi feladat keresőgépei közül melyek használnak betűkonverziót, szótövesítést, szinonimákat és helyesírásihiba-javítást.

23.6.

Becsülje meg, hogy mekkora tárterület szükséges egy egymilliárd weboldalt tartalmazó korpusz indexeléséhez. Mutassa be a feltételezéseket, amelyekre alapozott.

23.7.

Írjon egy reguláris kifejezést vagy egy rövid programot vállalatnevek kinyerésére. Tesztelje üzleti híreket tartalmazó cikkeken. Adja meg az elért felidézést és pontosságot.

23.8.

Válasszon ki öt mondatot és küldje el egy internetes fordítószolgáltatásnak. Fordítsa le angolról egy idegen nyelvre, majd vissza angolra. Értékelje a kapott mondatokat nyelvtani helyesség és a jelentés megtartásának szempontjából. Ismételje meg a folyamatot; az iteráció második lépése rosszabb vagy azonos eredményeket ad? Befolyásolja az eredmény minőségét a közbenső nyelv megválasztása?

23.9.

Gyűjtsön össze időre vonatkozó kifejezéseket, például „két órakor”, „éjfélkor”, „12:46-kor”. Találjon ki olyan nyelvtanilag helytelen példákat is, mint „huszonöt órakor” vagy „félnegyedháromkor”. Írjon nyelvtant az idő nyelvére.

Created by XMLmind XSL-FO Converter.

23.10.

(Knight, 1999) alapján. Az IBM Model 3 gépi fordító feltételezi, hogy miután a szóválasztási modell egy szólistát ajánl, a nyelvi modell képes kiválasztani a legjobb permutációt. Ez a feladat megvizsgálja, hogy mennyire értelmes ez a feltételezés. Próbálja meghatározni a következő mondatok szavainak helyes sorrendjét:

a. have programming a seen never I language better

b. loves john mary

c. is the communication exchange of intentional information brought by about the production perception of and signs from drawn a of system signs conventional shared

Melyik mondat szavait tudta sorrendbe tenni? Milyen tudás alapján volt erre képes? Tanítson egy bigram modellt egy tanító korpusz alapján, és használja arra, hogy meghatározza egy tesztkorpusz mondatainak legnagyobb valószínűségű permutációit. Mutassa meg a modell pontosságát.

23.11.

Amennyiben megnéz egy angol–francia szótárt, a „hear” ige fordítása „entendre”.[34] Azonban ha az IBM Model 3-at a kanadai Hansard alapján tanítjuk, akkor a „hear” legvalószínűbb fordítása „Bravo”. Magyarázza meg, hogy mi ennek az oka, és becsülje meg, hogy mi lehet a „hear” termékenységi eloszlása. (*Segítség:* ha Hansard-szövegeket szeretne megnézni, írja be egy webkeresőbe a [Hansard hear] lekérdezést.)

[34] hall, hallgat (*A ford.*)

Created by XMLmind XSL-FO Converter.

24. fejezet - Az észlelés

Ebben a fejezetben a számítógépet a zord, koszos világhoz csatlakoztatjuk.

Az észlelés információhoz juttatja az ágenst arról a világról, amelyben létezik. Az észlelést az **érzékelő**k (**sensor**s) kezdeményezik. Egy érzékelő lehet bármi, ami a környezet valamelyik aspektusát rögzíti és továbbítja az ágensprogram számára. Ez olyan egyszerű is lehet, mint egy egybites érzékelő, amely egy kapcsoló be és ki állapotát jelzi, de lehet olyan bonyolult is, mint az emberi szem retinája, amely több mint százmillió fotoérzékeny elemet tartalmaz. Ebben a fejezetben a látásra összpontosítunk, mert a fizikai világ kezelésében ez az észlelésfajta messze a leghatásosabb.

1. Bevezetés

Egy mesterséges ágens több észlelési modalitással is rendelkezhet. Amin az ágensek és az emberek is osztoznak, az többek között a látás, a hallás és a tapintás. A hallással a beszédfelismerés témán belül, a 15.6. alfejezetben foglalkozunk. A tapintás, másképpen a **taktilis érzékelés** (**tactile sensing**) a 25. fejezet témája, ahol a kézi műveleteket végző ágenseknél vizsgáljuk meg ennek a használatát, a fejezet hátralevő része pedig a látással foglalkozik. Egyes robotok **aktív érzékelés**t (**active sensing**) végeznek, azaz kiküldenek egy jelet, mint például radar- vagy ultrahangjelet, és ennek a jelnek a környezetből érkező visszaverődését vizsgálják.

Egy ágens kétféle módon használhatja az érzékeléseit. A **tulajdonság kinyerés**i (**feature extraction**) megközelítés szerint az ágens néhány kevés számú tulajdonságot észlel az érzékelő bemenetein, és ezeket továbbítja az ágensprogramnak, amely a tulajdonságoknak megfelelően reagál, vagy kombinálja őket más információkkal. A wumpus ágens ily módon működött, öt érzékelővel, amelyek mindegyike egybites információt nyert ki. Ma már ismert, hogy egy légy az optikai bemenetről tulajdonságokat nyer ki, és ezeket direkt módon a kanyarodásban segítő izmokhoz továbbítja, lehetővé téve, hogy 30 milliszekundumon belül reagáljon, és irányt váltson.

Az alternatív lehetőség a **modellalapú** (**model-based**) megközelítés, ahol a szenzoros bemenet alapján a világ egy modellje épül fel. E megközelítés szerint egy f függvényből indulunk ki, amely a világ W állapotát az általa keltett S ingerületre képezi le:

$$S = f(W)$$

Az f függvényt a fizika és az optika határozza meg, és viszonylag jól ismerjük. S és egy valós vagy elképzelt W világ előállítása f-ből a **számítógépes grafika** (**computer graphics**) által kezelt probléma. A számítógépes látás bizonyos értelemben a számítógépes grafika fordítottja, ha adott f és S, akkor W-t próbáljuk kiszámítani:

$$W = f^{-1}(S)$$

Sajnos az f-nek nincs jól definiált inverze. Nem láthatunk a sarkon túlra, így a világ aktuális állapotának minden aspektusát az ingerületből visszaállítani nem tudjuk. Sőt az a része, amit *láthatunk,* nagyban bizonytalan: kiegészítő információ nélkül nem tudjuk megmondani, hogy S egy játék Godzilla-képe, amint egy hatvan centiméteres modellépületet tarol le, vagy egy valóságos szörnyeteg, amely egy hatvanméteres épületet pusztít el. Ezen problémák egy részét kezelhetjük úgy, hogy egy világ megtalálása helyett világok valószínűségi eloszlását határozzuk meg:

$$P(W) = P(W|S)P(S)$$

Az ilyen modellezés legfontosabb hátránya az, hogy túl nehéz problémát kísérel meg megoldani. Gondoljunk bele, hogy a számítógépes grafikában többórányi számításra lehet szükség egy film egyetlen kockájának az előállításához, amiből 24 kocka kell másodpercenként, és f^{-1} kiszámítása sokkal bonyolultabb, mint f-é. Világos, hogy ez túl sok számítás egy szuperszámítógép számára, hogy valós időben reagáljon, nem is beszélve egy légyről. Szerencsére az ágensnek nem kell a fényképhű számítógépes grafikához hasonló szintű modellel rendelkeznie a világról. Az ágensnek elegendő azt tudnia, hogy rejtőzködik-e egy tigris a bozótban, és nem kell tudnia a tigris hátán levő minden egyes szőrszál pontos elhelyezkedését és irányát.

A fejezet nagy részében azt fogjuk látni, hogy hogyan kell objektumokat – például tigriseket – felismerni, továbbá olyan módszereket, amelyek ezt a tigris minden egyes részletének reprezentálása nélkül teszik. A 24.2. alfejezetben a képalakítás folyamatát tanulmányozzuk majd, definiálva az $f(W)$ függvény egyes aspektusait.

Created by XMLmind XSL-FO Converter.

Először a folyamat geometriájára tekintünk. Látni fogjuk, hogy a fény a világ objektumairól az ágens érzékelőjének képsíkján lévő pontokra verődik vissza. Ez a geometria magyarázza meg, hogy egy nagy Godzilla a távolban miért néz ki úgy, mint egy kis Godzilla a közelben. Ezután a folyamat fotometriáját vizsgáljuk, amely leírja, hogy a táj fénye hogyan határozza meg a kép pontjainak fényességét. A geometria és a fotometria együtt egy olyan modellt nyújtanak, amely leírja, hogy a világ objektumai hogyan képződnek le a képpontok kétdimenziós tömbjére.

Annak megértése után, hogy hogyan jönnek létre a képek, megvizsgáljuk, hogy hogyan dolgozzuk fel őket. A vizuális feldolgozás folyamata az emberekben és a számítógépekben is három részre bontható. Először, avagy az alacsony szintű látás során (24.3. alfejezet), a feldolgozatlan képet simítjuk, hogy kiszűrjük a zajt, majd a kétdimenziós kép tulajdonságait kinyerjük, elsősorban a kép régiói közötti éleket. A középső szintű látásban ezeket összekapcsolva régiókat hozunk létre. A magas szintű látásban (24.4. alfejezet) a kétdimenziós régiókat mint a világ valóságos objektumait ismerjük fel (24.5. alfejezet). A képekben megtalálható olyan speciális információkat tanulmányozunk, amelyeket erre a célra felhasználhatunk, mint például a mozgást, a térbeli információt,a textúrát, az árnyalást és a kontúrokat. Az objektumfelismerés fontos a vadonban levő ágens számára, hogy észlelje a tigriseket, és fontos az ipari robotok számára, hogy megkülönböztesse az anyákat a csavaroktól. Végezetül a 24.6. alfejezet megmutatja, hogy az objektumok felismerése hogyan segíthet nekünk olyan hasznos feladatokban, mint például a manipulálás vagy a navigáció. A manipulálás azt jelenti, hogy megragadhatunk és használhatunk szerszámokat, illetve más tárgyakat, a navigáció pedig azt, hogy egyik helyről átmehetünk egy másikra anélkül, hogy nekimennénk valaminek. Ezeket a feladatokat észben tartva biztosíthatjuk, hogy egy ágens csak akkora modellt építsen, amire céljai eléréséhez szüksége van.

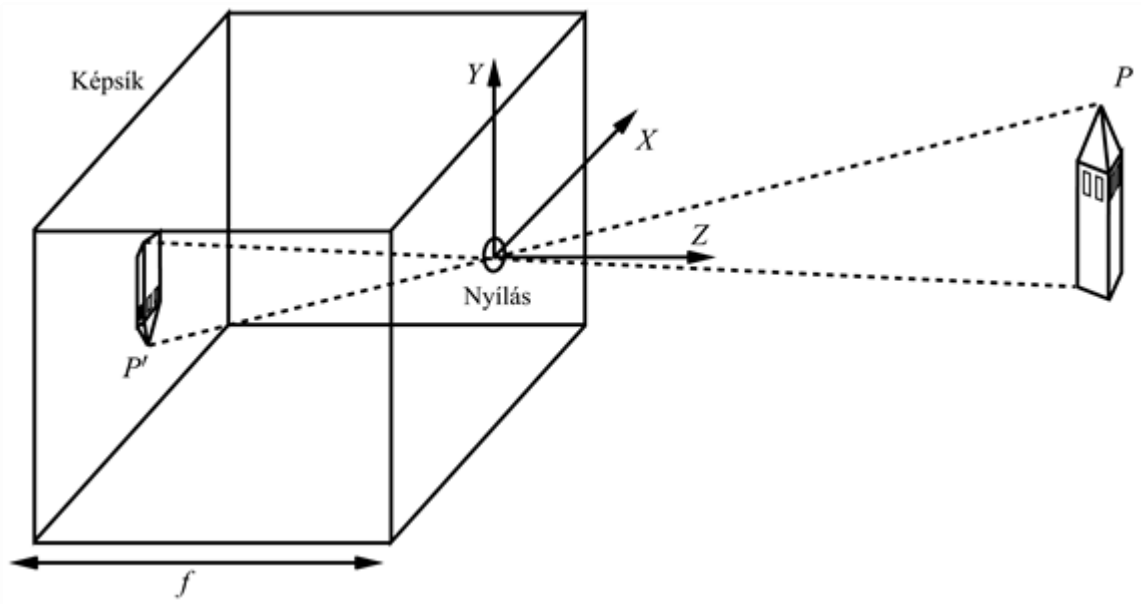
2. Képalkotás

A látás egy **jelenet** (**scene**) objektumairól szóródó fényt összegyűjti, majd egy kétdimenziós **kép**et (**image**) alkot egy képsíkon. A képsíkot fényérzékeny anyag borítja: a retinában rodopszinmolekulák, a negatív filmen ezüst-halogenidek, a digitális fényképezőgépben pedig egy töltéscsatolt eszköz (CCD) lapka. A CCD minden cellája összegyűjti a fényelnyelés által adott idő alatt keltett elektronokat. A digitális fényképezőgépben a képsík egy néhány millió **képpont**ból (**pixel**) álló négyzethálóra van felosztva. A szem hasonló képpontráccsal rendelkezik, amely körülbelül 100 millió pálcikát és 5 millió csapot tartalmaz egy hexagonális mozaikba rendezve.

A jelenet nagyon nagy, és a képsík egészen kicsi, tehát lennie kell valamilyen módszernek a fény képsíkra fókuszálásához. Ezt megtehetjük lencsével vagy anélkül. Bárhogy is tesszük, a lényeg a geometria meghatározása, hogy megmondhassuk, a jelenet melyik pontja hova kerül a képsíkon.

2.1. Lencsék nélküli képek: a sötétkamra

24.1. ábra - A képalkotás geometriája a sötétkamrában



Created by XMLmind XSL-FO Converter.

A képalkotás legegyszerűbb eszköze a **sötétkamra**, amely egy doboz elején található kis lyukból (O) és a doboz hátsó részén levő képsíkból áll (lásd 24.1. ábra). Egy háromdimenziós koordináta-rendszert fogunk használni, amelynek az origója az O pont, a jelenet egy P pontját pedig az (X, Y, Z) koordinátákkal reprezentáljuk. A P pontot egy P' pontra képezzük le a képsíkon, amelynek koordinátái (X, Y, Z). Ha f az O pontszerű lyuknak a képsíktól vett távolsága, akkor a hasonló háromszögek alapján az alábbi egyenleteket tudjuk levezetni:

$$\frac{-x}{f}=\frac{X}{Z},\frac{-y}{f}=\frac{Y}{Z}\Rightarrow x=\frac{-fX}{Z},y=\frac{-fY}{Z}$$

Ezek az egyenletek egy **perspektivikus vetítés** (**perspective projection**) néven ismert képalkotási eljárást definiálnak. Vegyük észre, hogy a Z az osztóban azt jelenti, hogy minél távolabb van az objektum, annál kisebb lesz a képe. Valamint azt is figyeljük meg, hogy a kép, mind a bal–jobb, mind a fel–le irányban *fordított* a jelenethez képest, amit az egyenletekben a negatív előjel jelez.

A perspektivikus vetítésben a párhuzamos vonalak a horizontvonalon lévő pontban futnak össze. Nézzük meg, miért szükségszerű ez! Az (X_0, Y_0, Z_0) ponton keresztül (U, V, W) irányban áthaladó egyenes leírható, mint az ($X_0 + \lambda U$, $Y_0 + \lambda V$, $Z_0 + \lambda W$) pontok halmaza, ahol λ a $+\infty$ és $-\infty$ között változik. Ezen egyenes egy P_λ pontjának képsíkra vett vetületét az:

$$\left(f\frac{X_0+\lambda U}{Z_0+\lambda W},f\frac{Y_0+\lambda V}{Z_0+\lambda W}\right)$$

adja. Amint $\lambda \to \infty$ vagy $\lambda \to -\infty$ ez a pont a $p_\infty = (fU/W, fV/W)$ lesz, ha $W \neq 0$. A p_∞ pontot az (U, V, W) irányú egyenesek családjával kapcsolatos **távlatpont**nak (**vanishing point**) nevezzük. Az azonos irányú egyenesek távlatpontja azonos.

Ha a tárgy a kamrától vett távolságához képest viszonylag lapos, a perspektivikus vetítést a **skálázott ortografikus vetítés**sel (**scaled orthographic projection**) közelíthetjük. Ennek elve a következő. Ha a tárgy pontjainak Z mélysége egy bizonyos $Z_0 \pm \Delta Z$ tartományban változik, és $\Delta Z = Z_0$, akkor a perspektíva f/Z skála tényezőjét az $s = f/Z_0$ állandóval lehet közelíteni. Az (X, Y, Z) jelenet koordinátákból a képsíkra történő vetítés egyenletei az $x = sX$ és az $y = sY$ lesznek. Jegyezzük meg, hogy a skálázott függőleges síkú vetítés egy közelítés, amely a jelenet azon részeire érvényes, amelyeknek a saját mélységváltozásuk elenyésző. A globális tulajdonságok tanulmányozására ezt a módszert nem szabad használni. Hogy az óvatosság nem árt, erre egy példa: függőleges síkú vetítésben a párhuzamos vonalak párhuzamosak maradnak ahelyett, hogy egy távlatpontban összefutnának.

2.2. Lencserendszerek

A gerincesek szeme és a modern kamerák is lencséket használnak. A lencse a sötétkamra nyílásánál nagyobb, így több fényt enged át. Ennek az az ára, hogy egyidejűleg a jelenet minden részének az élességét nem lehet biztosítani. A jelenet egy Z távolságban lévő pontjának a képe a lencsétől egy rögzített Z' távolságban keletkezik, és a Z és a Z' között az:

$$\frac{1}{Z}+\frac{1}{Z'}=\frac{1}{f}$$

reláció áll fenn, ahol f a lencse fókusztávolsága. Ha a lencse optikai középpontja és a képsík közötti távolságot valamilyen Z'_0-nak választjuk, a jelenet azon tárgyai, amelyek a Z_0 körül egy adott mélységtartományban fekszenek, ahol Z_0 a megfelelő tárgytávolság, nagyjából élesen fognak látszani. A jelenet ezen mélységi tartományát **mélységélesség**nek (**depth of field**) nevezzük.

Jegyezzük meg, hogy mivel a Z tárgytávolság általában sokkal nagyobb, mint a Z' képtávolság vagy mint az f, sokszor jogos az alábbi közelítés:

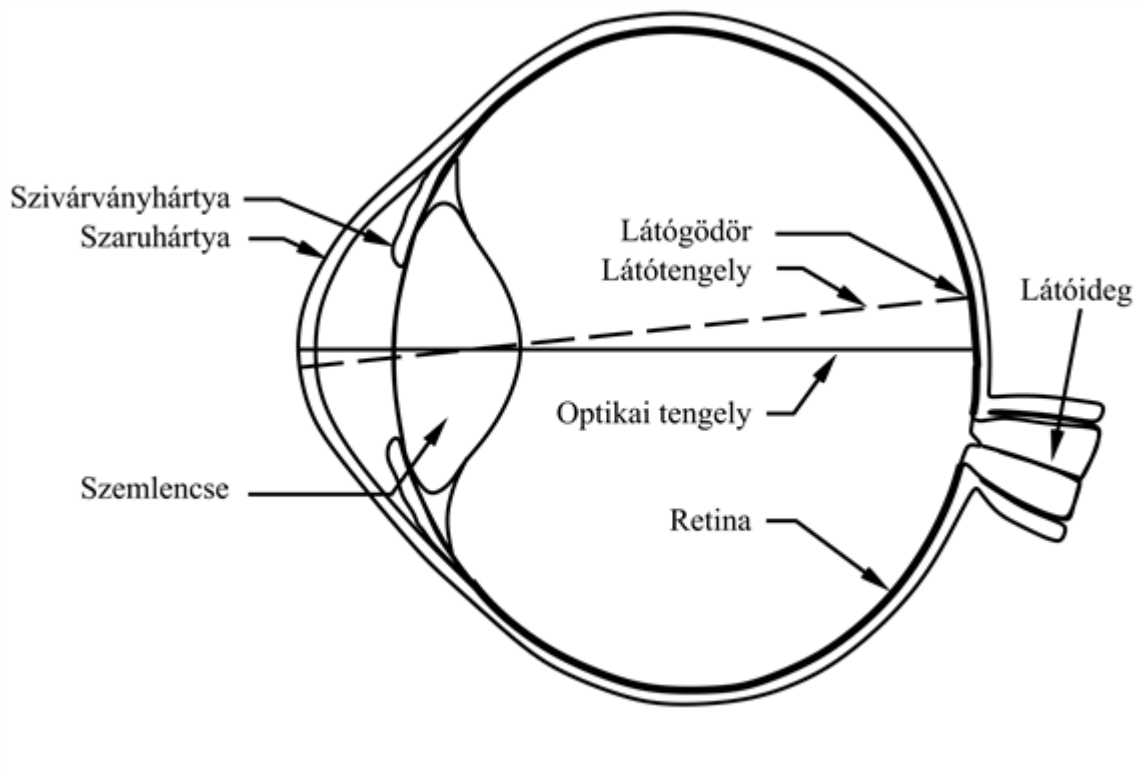
$$\frac{1}{Z}+\frac{1}{Z'}\approx\frac{1}{Z'}\Rightarrow\frac{1}{Z'}\approx\frac{1}{f}$$

Azaz a képtávolság $Z' \approx f$. A sötétkamra vetítésegyenleteit tehát egy lencse által létrehozott kép geometriájának leírásához is használhatjuk.

Created by XMLmind XSL-FO Converter.

Ahhoz, hogy a különböző Z távolságban lévő tárgyakat élesen lássuk, az emberi szem lencséje (lásd 24.2. ábra) változtatja az alakját, a kamera lencséje pedig Z irányban elmozdul.

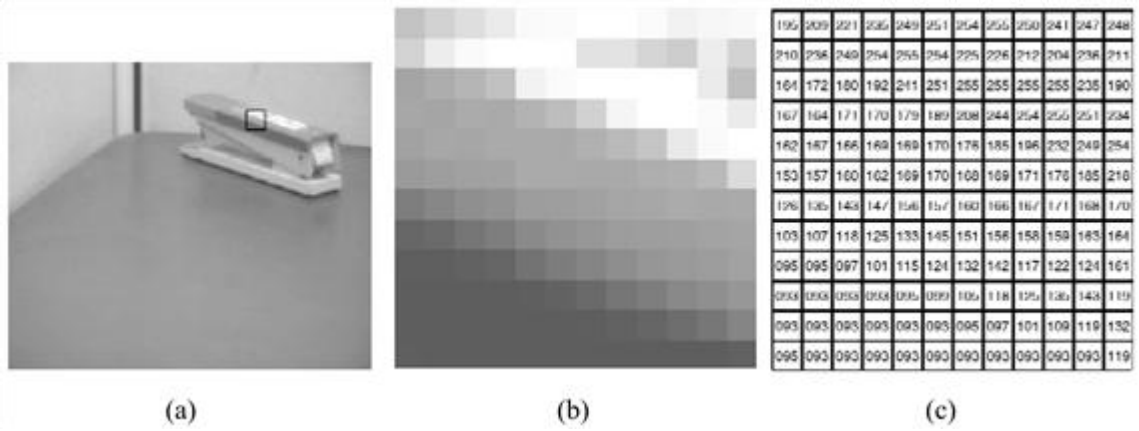
24.2. ábra - Az emberi szem vízszintes keresztmetszete



2.3. A fény: a képalkotás fotometriája

A fény nélkülözhetetlen a látáshoz: nélküle minden kép egyformán sötét lenne, függetlenül attól, mennyire érdekes a jelenet. A **fotometria** (**photometry**) a fény tanulmányozása. Saját céljainkra azt fogjuk modellezni, hogy a jelenet fénye hogyan képződik le az időben a képsík fényintenzitására, amit $I(x, y)$-nal[1] jelölünk. A látás rendszere ezt a modellt visszafelé alkalmazza, a képek intenzitásából kiindulva a világ tulajdonságai felé. A 24.3. ábra egy asztalon lévő tűzőgép digitalizált képét és a tűzőgép egy 12 × 12 képpontból álló részletét mutatja. Egy számítógépes program, amely értelmezni próbálja a képet, egy ilyen intenzitásmátrixból indulna ki.

24.3. ábra - (a) Egy asztalon lévő tűzőgép fényképe. (b) Az (a) egy 12 × 12 képpontból álló, nagyított részlete. (c) A részletnek megfelelő képfényességértékek egy 0-tól 255-ig tartó skálán.



[1] Ha az időbeli változást is vizsgáljuk, akkor az $I(x, y, t)$ jelölést alkalmazzuk.

Created by XMLmind XSL-FO Converter.

A kép egy képpontjának a fényessége arányos a jelenet képpontba vetített felületeleme által a kamera felé irányított fény mennyiségével. Ez viszont függ a felületelem fényvisszaverő tulajdonságaitól és a fényforrások elhelyezésétől, valamint eloszlásától a jelenetben. Egy képpont fényességébe a jelenet egyéb részeinek fényvisszaverő tulajdonságai is beleszólnak, hiszen a jelenet más felületei indirekt fényforrásként szolgálnak, mivel a rájuk eső fényt az adott felületelem felé verik vissza.

Kétféle visszaverődést modellezhetünk. A **tükröző visszaverődés** (**specular reflection**) azt jelenti, hogy a fény a tárgy külső felületéről verődik vissza, és teljesíti azt a kényszert, miszerint a beesés és a visszaverődés szöge megegyezik. Ilyen a tökéletes tükör viselkedése. A **diffúz visszaverődés** (**diffuse reflection**) azt jelenti, hogy a fény a tárgy felszínén belülre hatol, a tárgy a fényt elnyeli és ismételten kisugározza. A tökéletesen **diffúz** (avagy **Lambert-féle**) **felület** minden irányban azonos intenzitással szórja a fényt. Az intenzitás egyedül a fényforrástól érkező fény beesési szögétől függ: a felületre pontosan merőleges fényforrás esetén lesz a legnagyobb intenzitású a reflexió, míg majdnem párhuzamos fényforrás esetén a legkisebb. E két véglet között a viszszaverődést Lambert koszinusztörvénye írja le:

$$I = kI_0 \cos\theta$$

ahol *s* a fényforrás intenzitása, a felület normálisa és a fény beesési iránya közötti szög, *k* a visszaverődési tényező értéke, amely a felület visszaverési képességeitől függ, és 0-tól (tökéletesen fekete felület) 1-ig (tökéletesen fehér felület) változik.

A valódi életben a felületek kevert, diffúz és tükrös tulajdonságúak. Számítógépes modellezésük a számítógépes grafika fő alkalmazási területe. A valósághű képek előállítása általában fénysugár-követési eljárással történik, amely szimulálja azt a fizikai folyamatot, ahogy a fénysugár a forrásnál keletkezik, és a tárgyakról többszörösen visszaverődik.

2.4. Színek: a képalkotás spektrális fotometriája

A 24.3. ábrán egy fekete-fehér képet mutattunk, nagyban figyelmen kívül hagyva azt a tényt, hogy a látható fény a hullámhosszak egész tartományát öleli át – a 400 nm-től kezdve a spektrum ibolyaszínű végénél, egészen a 700 nm-ig a vörös színű végénél. Bizonyos fény csak egyetlen hullámhosszból áll, amely a szivárvány egy színének felel meg. De más fények különböző hullámhosszok keverékei. Azt jelenti ez, hogy az $I(x, y)$ mértékére egyetlen szám helyett az értékek egy keveréke kell? Ha a fény fizikáját pontosan akarnánk reprezentálni, akkor bizony igen. De ha csak utánozni akarjuk az emberek (és más gerincesek) fényérzékelését, akkor köthetünk kompromisszumokat. Kísérletek mutatják (egészen 1801-től, Thomas Youngtól), hogy hullámhosszak bármilyen komplex keverékét is előállíthatjuk mindössze három szín keverésével. Azaz, ha van egy fénygenerátor, amely képes lineárisan kombinálni három hullámhosszt (tipikusan a vörös [700 nm], a zöld [546 nm] és a kék [436 nm] színeket választjuk), akkor az egyes színeket erősítő, másokat gyengítő gombok csavargatásával *bármilyen* hullámhossz-kombináció beállítható, legalábbis az emberi érzékelés szempontjából. Ez a kísérleti tény azt jelenti, hogy a képek egy olyan vektorral reprezentálhatók, amely képpontokként csak három intenzitásértéket tárol: minden elsődleges hullámhosszra egyet. A gyakorlatban mindegyiket egy bájttal ábrázolva a kép igen jó minőségű reprodukcióját kapjuk. A színek ezen háromszínű észlelése azzal a ténnyel van összefüggésben, hogy háromféle csap található a retinában, amelyek érzékelési maximuma 650, 530 és 430 nm, de az összefüggés pontos részletei az egy az egyben történő leképezésnél bonyolultabbak.

3. Előzetes képfeldolgozási műveletek

Láttuk eddig, hogy a fény hogyan tükröződik a jelenet tárgyairól egy mondjuk ötmillió hárombájtos képpontból álló képet formálva. Mint minden érzékelő esetében, most is lesz zaj a képben, és minden esetben nagyon sok adattal kell dolgozni. Elsőként a finomítás műveleteit nézzük meg, amelyekkel a zaj csökkenthető, majd az éldetektálás műveleteit vizsgáljuk. Ezeket „előzetes” vagy „alacsony szintű” műveleteknek hívjuk, mivel ezek az elsők egy műveletsorozatban. Az előzetes látási műveletekre egyrészt a lokalitás jellemző (a kép egy részletére végrehajthatók anélkül, hogy néhány képpontnál messzebb bármit is figyelembe vennénk), másrészt az ismeretek hiánya: anélkül finomíthatjuk a képeket vagy detektálhatunk éleket, hogy bármilyen fogalmunk lenne arról, milyen objektumok vannak a képeken. Ez az alacsony szintű műveleteket esélyessé teszi a párhuzamos hardveren történő megvalósításra, akár élőlényekben, akár mesterséges eszközökben. Ezek után a középső szinten levő műveletek nézzük majd meg, amelyekkel a kép régiókra bontható. A műveletek ezen fázisa még mindig a képen dolgozik, nem a jeleneten, de nem helyi feldolgozást igényel.

Created by XMLmind XSL-FO Converter.

A 15.2. alfejezetben a **simítás** (**smoothing**) azt jelentette, hogy egy állapotváltozó értékét jósoltuk meg valamilyen múltbeli t időpontra, bizonyos, a t időpontra és további, egészen a jelenig terjedő időpontokra vonatkozó tények ismeretében. Most ugyanezt az ötletet alkalmazzuk a térbeli kiterjedésre az időbeli helyett: a simítás egy képpont értékének jóslását jelenti a környező képpontok alapján. Vegyük észre, hogy világosan megkülönböztetjük a képpontban mért megfigyelt értéket és azt a valódi értéket, amit ott mérnünk kellett volna. Ezek véletlenszerű és szisztematikus mérési hibák (például ha a CCD érzékelője elromlott) miatt különbözhetnek.

A simítás egyik módszere az, ha minden képponthoz az őt környező szomszédok értékeinek átlagát rendeljük. Ez az extrém értékek kihagyása felé mutató megoldás. De hány szomszédot kell figyelembe vennünk – az egy képpontnyi távolságra vagy a kettő esetleg több pont távolságra lévőket? Erre egy válasz, amely jól működik Gauss-zajok kiszűrésére, a **Gauss-szűrő**t (**Gaussian filter**) használó súlyozott átlag. Emlékezzünk a standard δ szórással rendelkező Gauss-függvényre:

$$G_\sigma(x)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{-x^2/2\sigma^2} \qquad \text{egy dimenzióban, illetve}$$

$$G_\sigma(x,y)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\left(x^2+x^2\right)/2\sigma^2} \qquad \text{két dimenzióban.}$$

Egy Gauss-szűrő alkalmazása azt jelenti, hogy az $I(x_0, y_0)$ intenzitást lecseréljük a minden (x, y)-ra kiszámolt $I(x, y)G_\sigma(d)$ intenzitások összegével, ahol d az (x_0, y_0) és az (x, y) pontok távolsága. Az ilyen fajta súlyozott összeg olyan gyakori, hogy van számára egy speciális elnevezés és jelölés. Azt mondjuk, hogy a h függvény az f és a g függvények **konvolúció**ja (**convolution**) (amit $h = f * g$ alakban írunk), ha

$$h(x)=\sum_{u=-\infty}^{+\infty} f(u)g(x-u) \qquad \text{egy dimenzióban,illetve}$$

$$h(x,y)=\sum_{u=-\infty}^{+\infty}\sum_{v=-\infty}^{+\infty} f(u,v)g(x-u,y-v) \qquad \text{két dimenzióban.}$$

Azaz a simított függvényt úgy kapjuk, hogy a képet a Gauss-függvénnyel konvolváljuk: $I * G_\sigma$. Egy képpontnyi σ kevés zaj kisimításához elegendő, míg 2 képpontnyi több zajt fog elsimítani, de valamilyen mértékű részletvesztés mellett. Mivel a Gauss-hatás a távolsággal eltűnik, a gyakorlatban a $\pm\infty$ a szummákban $\pm 3\sigma$-val helyettesíthető.

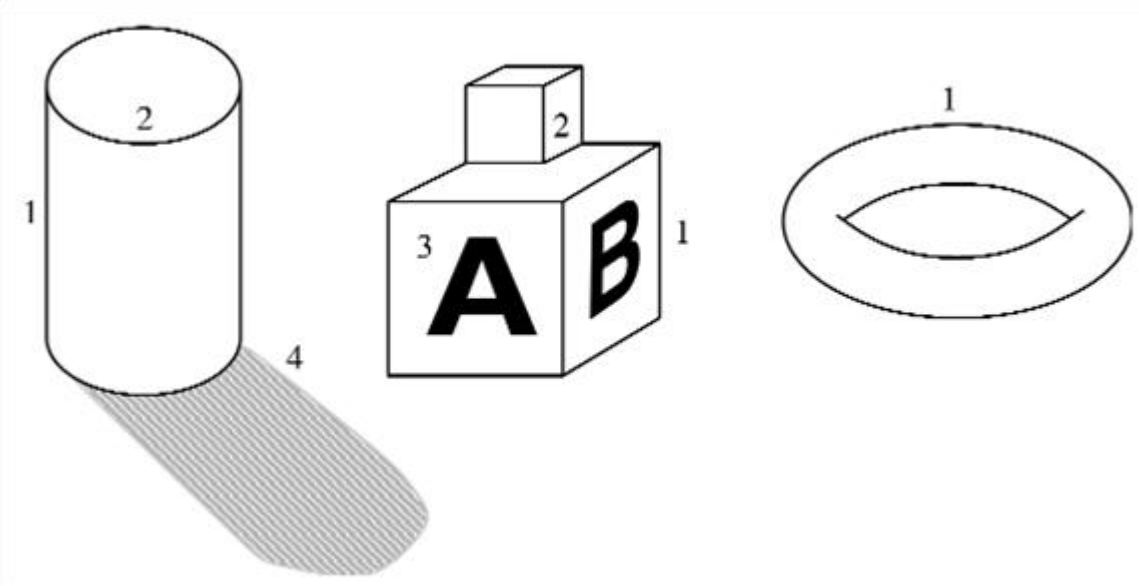
3.1. Éldetektálás

Az előzetes látás következő lépése az **él**ek (**edge**s) detektálása a képsíkon. Az élek olyan egyenes vagy görbe vonalak a képsíkon, amelyekre merőlegesen a kép fényességében „lényegesen” változás van. Az éldetektálás végső célja, hogy eltávolodjunk a bonyolult, több megabájtos képtől egy tömörebb, absztrakt reprezentáció felé, mint amilyen a 24.4. ábrán látható. Az indíték az, hogy a kép élkontúrjai a jelenet fontos kontúrjainak felelnek meg. A képen három mélységi diszkontinuitás példa szerepel 1-gyel címkézve; a 2-vel címkézett felületi orientáció diszkontinuitásból két példa látható, míg a 3-mal címkézett visszaverődési képesség diszkontinuitásokról és a 4-gyel címkézett fényességi diszkontinuitásokról (árnyék) egy-egy. Az éldetektálás csak a képpel foglalkozik, így nem tesz különbséget a jelenet ezen diszkontinuitásfajtái között, de a későbbi feldolgozás igen.

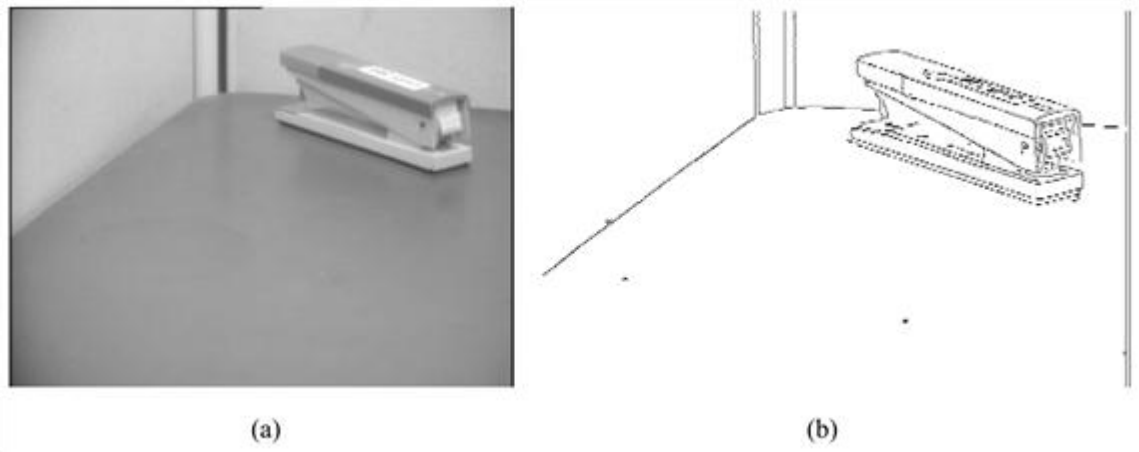
A 24.5. (a) ábra egy asztalon nyugvó tűzőgépet tartalmazó jelenet képét mutatja, a 24.5. (b) pedig a képre lefuttatott éldetektáló algoritmus eredményét mutatja. Láthatjuk, hogy az éldetektáló kimenete és az ideális vonalas ábra között van különbség. A kis részek élei nem mind illeszkednek egymáshoz, vannak hézagok, ahol nincsenek vonalak, valamint „zajos” kontúrok, amelyek a jeleneten semmilyen lényeges vonásnak nem felelnek meg. A későbbi feldolgozási fázisokban ezeket a hibákat majd korrigálni kell. Hogyan detektáljuk a képen az éleket? Figyeljük meg a kép fényintenzitás-profilját egy egydimenziós keresztmetszet mentén egy élre merőleges irányban, pl. az asztal bal éle és a fal között. Nagyjából olyan lesz ez, mint amit a 24.6. (a) ábra mutat. Az él helye az $x = 50$-nek felel meg.

Created by XMLmind XSL-FO Converter.

24.4. ábra - Különféle élek: (1) mélységi diszkontinuitás, (2) felületi orientáció diszkontinuitás, (3) viszszaverődési képesség diszkontinuitás, (4) fényességi diszkontinuitás (árnyékok)



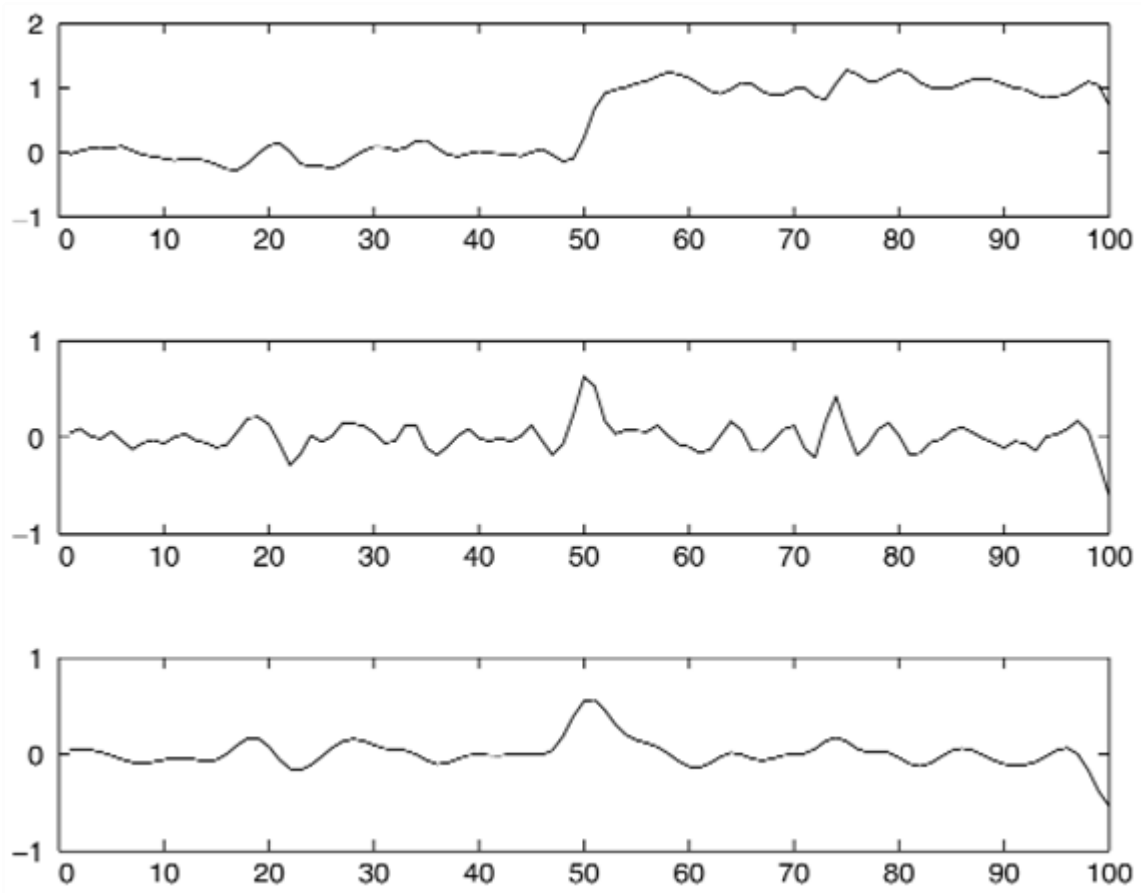
24.5. ábra - (a) Egy asztalon nyugvó tűzőgép fényképe. (b) Az (a) alapján meghatározott élek.



Tekintettel, hogy az élek azoknak a képhelyeknek felelnek meg, ahol a fényességben hirtelen változás tapasztalható, egy naiv ötlet az lehetne, hogy a képet differenciáljuk, és azokat a helyeket keressük meg, ahol az $I'(x)$ derivált értéke nagy. Nos, az ötlet majdnem működőképes. A 24.6. (b) ábrán láthatjuk, hogy az $x = 50$ csúccsal együtt más helyeken, másodlagos csúcsok is megjelentek (például $x = 75$), amiket tévesen valódi éleknek fogadhatunk el. A jelenség oka a kép zajossága. Ha a képet először simítjuk, akkor a felesleges csúcsok eltűnnek, ahogy azt a (c) mutatja.

24.6. ábra - Felül: Az $I(x)$ intenzitásprofil egy egydimenziós részlet mentén egy lépcsőfok élén át. Középen: Az intenzitás deriváltja, $I'(x)$. A függvény nagy értékei éleknek felelnek meg, de a függvény zajos. Lent: Az intenzitás simított változatának deriváltja, $(I * G_\sigma)'$, amely $I * G'$-hoz hasonlóan egy lépésben számolható. Az $x = 75$ pontban levő zajos él jelölt eltűnt.

Created by XMLmind XSL-FO Converter.



Van arra esélyünk, hogy optimalizáljuk a munkát ezen a ponton: egyetlen műveletben kombinálhatjuk a simítást és az éldetektálást. Egy tétel szerint tetszőleges f és g függvények esetén megmutatható, hogy a konvolúció deriváltja $(f * g)'$ megegyezik a derivált konvolúciójával: $f * (g)'$. Tehát ahelyett, hogy először simítanánk, majd differenciálnánk, egyszerűen konvolváljuk a képet a Gauss simító függvény deriváltjával, G'_σ-val. Így az egydimenziós éldetektálás algoritmusa a következő:

1. Az I képet a G'_σ-val kell konvolválnunk, hogy az R-et megkapjuk.

2. Jelöljük be az azokat a csúcsokat $||R(x)||$-ben, amelyek egy előre definiált T küszöbnél magasabbak. A küszöb értékét úgy határozzuk meg, hogy a zaj miatti másodlagos csúcsokat eliminálják.

Két dimenzióban egy él irányítottsága tetszőleges θ szögű lehet. A függőleges élek detektálásának módja nyilvánvaló: $G'_\sigma(x)\ G_\sigma(y)$-nal kell konvolválnunk a képet. A hatás y irányban simító (a Gauss-konvolúció miatt), x irányban pedig egy simítással kísért differenciálás. A függőleges élek detektálási algoritmusa így az alábbi:

1. Konvolváljuk az $I(x, y)$ képet az $f_V(x, y) = G'_\sigma(x)\ G_\sigma(y)$-nal, hogy az $R_V(x, y)$ -t megkapjuk.

2. Jelöljük be azokat a csúcsokat az $||R_V(x, y)||$-ben, amelyek egy előre definiált T_n küszöbnél magasabbak.

A tetszőleges orientációjú él detektálásához a képet két szűrővel: az $f_V = G'_\sigma(x)\ G_\sigma(y)$ és az $f_H = G'_\sigma(y)\ G_\sigma(x)$ szűrővel kell konvolválni (az f_H az f_V 90°-kal elforgatott változata). A tetszőleges orientációjú éldetektáló algoritmus az alábbi:

1. Konvolváljuk az $I(x, y)$ képet az $f_V(x, y)$-nal, valamint az $f_H(x, y)$-nal, hogy az $R_V(x, y)$-t és az $R_H(x, y)$ -t megkaphassuk. Definiáljuk az $R(x,y) = R_V^2(x,y) + R_H^2(x,y)$-t.

2. Jelöljük be azokat a csúcsokat az $||R(x, y)||$-ban, amelyek egy előre definiált T küszöbnél magasabbak.

Miután ezzel az algoritmussal megjelöltük az élek képpontjait, a következő lépés az azonos élekhez tartozó képpontok összekötése. Ez megtehető, ha feltételezzük, hogy bármely két szomszédos képpont, amely élpont és

Created by XMLmind XSL-FO Converter.

konzisztens irányítottságú, szükségszerűen ugyanahhoz az élhez tartozik. Ezt a folyamatot **Canny-féle éldetektor**nak (**Canny edge detection**) nevezzük, kitalálója, John Canny után.

Miután detektáltuk őket, az élek alkotják több következő feldolgozási lépés alapját: felhasználhatjuk őket a térbeli képalkotásban, a mozgásdetektálásban és az objektumok felismerésében.

3.2. A kép szegmentálása

Az emberek *rendszerezik* észleléseiket; az egyes fotóérzékelőkhöz tartozó fényességértékek halmaza helyett vizuális csoportokat észlelünk, amelyek általában objektumokkal vagy azok részeivel asszociáltak. Ez a képesség a számítógépes látás számára is éppúgy fontos.

A **szegmentálás** (**segmentation**) a kép felbontása részcsoportokra a képpontok hasonlósága alapján. Az alapötlet a következő: minden képponthoz rendelhetünk bizonyos vizuális tulajdonságokat, mint például fényesség, szín és mintázat.[2] Ezek az attribútumok egy objektumon vagy annak egyetlen részén belül, csak kismértékben változnak, míg az objektumok közötti határok mentén tipikusan egyik vagy másik attribútum jelentősen változik. Úgy kell a képet képpontok halmazaira felosztanunk, hogy ezeket a kényszereket amennyire csak lehet, kielégítsük.

Többféle módszer létezik ezen intuitív elképzelés matematikai formalizálására. Például Shi és Malik (Shi és Malik, 2000) ezt gráfparticionálási problémaként határozzák meg. A gráf csomópontjai a képpontoknak felelnek meg, az élek pedig a köztük levő kapcsolatoknak. Egy W_{ij} súly értéke az i és j képpontokat összekötő élen attól függ, hogy a két képpont mennyire hasonló fényességben, színben, mintázatban és más tulajdonságokban. Ezek után olyan partíciókat keresnek, amelyek minimalizálnak egy *normalizált vágási* kritériumot. Durván fogalmazva a gráf particionálásának kritériuma, hogy a csoportok közötti élek súlyainak összegét minimalizáljuk, a csoportokon belüli élek súlyainak összegét pedig maximalizáljuk.

A pusztán alacsony szintű, helyi tulajdonságokon – például a fényességen és színen – alapuló szegmentáció hibázásra hajlamos eljárás. Ahhoz, hogy az objektumokhoz rendelt határokat megbízhatóan megtalálhassuk, a jelenetben várható objektumfajtákról magas szintű tudást is be kell építenünk. A beszédfelismerésre ezt a rejtett Markov-modell formalizmusa teszi lehetővé, míg a képek kontextusában egy ilyen egységes keretrendszer még mindig az aktív kutatás tárgya. Mindenesetre az objektumokról szóló magas szintű tudás a következő alfejezet tárgya.

4. D információ kinyerése

Ebben az alfejezetben azt mutatjuk meg, hogyan kell a kétdimenziós képből kiindulva a jelenet háromdimenziós reprezentációjáig eljutni. Fontos, hogy a jelenettel foglalkozzunk, hiszen végső soron az ágens a világban létezik, nem a képsíkon, a látás célja pedig annak elérése, hogy az ágens a világ objektumaival együttműködjön. Mindazonáltal az ágensek többségének csak a jelenet egyes aspektusainak korlátozott absztrakt reprezentációjára van szüksége, nem minden részletre. A könyv hátralevő részében látható, a világgal foglalkozó algoritmusok az objektumok tömör leírására építenek, nem minden háromdimenziós felületdarabka kimerítően teljes számbavételére.

Elsőként az **objektumfelismerés**t (**object recognition**) tárgyaljuk, azt a folyamatot, amely során a képek jellemzői (mint például az élek) átalakulnak ismert objektumok (mint például a tűzőgépek) modelljeivé. Az objektumfelismerés három lépésből áll: a jelenet szegmentálása különálló objektumokra, mindegyik objektum pozíciójának és térbeli orientációjának a megfigyelőhöz képesti meghatározása, valamint mindegyik objektum alakjának a meghatározása.

A manipulálási és a navigálási feladatok szempontjából legfontosabb az objektum pozíciójának és térbeli orientációjának (az objektum ún. **helyzet**ének – **pose**) a meghatározása a megfigyelőhöz képest. Ahhoz, hogy egy nyüzsgő gyárban eligazodjunk, tudnunk kell az akadályok elhelyezkedését, mert akkor egy olyan pályát tudunk tervezni, amely az akadályokat kikerüli. Ha egy tárgyat szeretnénk felvenni és kézben tartani, a kézhez képesti helyzetet kell ismernünk, mert csak így tudjuk generálni a megfelelő pályájú cselekvésszekvenciát. A manipulálást és a navigálást tipikusan szabályozási hurokban oldjuk meg – az érzékelők információja visszacsatolást jelent, amivel a robot vagy a robotkar mozgását befolyásolni tudjuk.

[2] A mintázati jellemzők a képpontra középpontozott kis felületdarabka statisztikai elemzésén alapulnak.

Created by XMLmind XSL-FO Converter.

Használjunk most matematikai jelölést a pozíció és az orientáció leírására. A jelenet egy *P* pontjának a pozícióját három szám jellemzi, a *P* pont (*X*, *Y*, *Z*) koordinátái a sötétkamra nyílásában elhelyezett origójú és az optikai tengellyel parallel *Z* tengelyű koordináta-rendszerben (lásd 24.1. ábra). Amivel rendelkezünk, az a kép egy pontjának (*x*, *y*) perspektivikus vetülete. Ez egyben a fénysugarat is definiálja a nyílás felől, amely mentén valahol elhelyezkedik a *P* pont. A *P* pont távolságát azonban nem ismerjük. Az „orientáció” fogalmát kétféle értelemben lehet használni:

1. **Az objektum egészének az orientációja**. Ezt egy háromdimenziós forgatási függvényben specifikálhatjuk, az objektum koordináta-rendszerét a kameráéhoz viszonyítva.

2. **Az objektum felületének az orientációja a P pontban.** Ez egy **n** normál vektorral adható meg, amely a felületre merőleges irányt határozza meg. A felület orientációját sokszor a **lejtés**sel (**slant**) és a **dőlés**sel (**tilt**) fejezzük ki. A dőlés az **n** és a *Z* tengely által bezárt szög. A lejtés az *X* tengely és az **n** képsíkra vett vetülete által bezárt szög.

Amikor a kamera a tárgyhoz képest elmozdul, az objektumnak mind a távolsága, mind az orientációja megváltozik. Ami viszont nem fog változni, az a tárgy **alak**ja (**shape**). Ha a tárgy pl. egy kocka, akkor a kocka jellege nem változik, ha a tárgy mozog. A geometria kutatói századok óta próbálkoznak az alak fogalmának formalizálásával – abból az alapötletből kiindulva, hogy az alak az a valami, ami bizonyos transzformációcsoport – pl. a forgatások és a transzlációk kombinációi – hatására nem változik. A nehézség abban rejlik, hogy hogyan keressük meg a globális alak egy olyan reprezentációját, amely elegendően általános ahhoz, hogy a valódi világ alakbeli változékonyságát – és nem csupán oly egyszerű formákat, mint a hengerek, a kúpok és a gömbök – lefedje, és mégis könnyen kinyerhető legyen a vizuális bemenetből. A felület *lokális* alakjának a jellemzése sokkal jobban feltárt. Ez lényegében a görbülettel lehetséges – azaz azzal, hogy hogyan változik a felület normálisa, ha a felületen különböző irányokba elmozdulunk. A sík esetén egyáltalán nincs változás. A henger esetén sincs, ha a tengely mentén haladunk, a rá merőleges irányban azonban a normális, a henger sugarával fordítottan arányos ütemben forog. Ezekkel a kérdésekkel a differenciálgeometria foglalkozik.

A tárgy alakja bizonyos manipulációs feladatok szempontjából fontos, pl. ahhoz, hogy eldöntsük, hogy a tárgyat hol fogjuk meg. Legfontosabb szerepe azonban a tárgyak felismerésében van, ahol a geometriai alak a színnel és a textúrával egyetemben a legjellemzőbb vonást jelenti, hogy a tárgyakat azonosíthassuk, a kép tartalmát a korábban látott osztályokba besoroljuk és így tovább.

Az alapvető probléma a következő: ha egyszer a perspektivikus vetítés alatt a háromdimenziós világ egy pontból kiinduló sugár mentén található minden pontja a képnek ugyanabba a pontjába vetítődött, hogyan nyerjük vissza a háromdimenziós információt? A vizuális érzékelésben ehhez számos segítő dolog áll rendelkezésre, a **mozgás**t, a **sztereolátás**t, a **textúrá**t, az **árnyalás**t és a **kontúrok**at beleértve. Ezek mindegyike, ahhoz hogy a kép értelmezését (közel) egyértelmű módon megoldhassa, a jelenetre vonatkozó háttér-feltételezéseken alapul. Az alábbiakban ezekkel a módszerekkel foglalkozunk.

4.1. A mozgás

Eddig egy időpontban csak egy képpel foglalkoztunk. A videokamerák azonban másodpercenként 30 képkockát rögzítenek, és a kockák közötti különbségek fontos információk forrásai lehetnek. Abban az esetben, ha a kamera a háromdimenziós jelenethez képest mozog, a képen keletkező látszólagos mozgást **optikai folyam**nak (**optical flow**) nevezzük. A folyam a *képbeli* jellegzetességek mozgásirányát és mozgási sebességét írja le, amit a jelenet és a megfigyelő egymáshoz viszonyított relatív mozgásának következtében kapunk. A 24.7. (a) és (b) ábrán egy forgó Rubik-kockát ábrázoló videofelvételből látunk két filmkockát. A (c) képen az előbbi képekből számított optikai folyam vektorai láthatók. Az optikai folyam a jelenet struktúrájára nézve tartalmaz hasznos információt. Például egy mozgó kocsiból kinézve a távoli tárgyaknak sokkal lassúbb a látszólagos mozgása, mint a közelieknek, így a látszólagos mozgás üteme a távolságról is hordoz valamiféle információt.

Az optikai folyam vektormezeje az *x* irányú $v_x(x, y)$ és az *y* irányú $v_y(x, y)$ komponensekkel jellemezhető. Ahhoz, hogy az optikai folyamot megmérhessük, meg kell találni az egymásnak megfelelő pontokat a szomszédos képkockákon. Ehhez azt a tényt használjuk fel, hogy az egymásnak megfelelő pontok körüli képrészeknek hasonló a fényességmintázata. Tekintsük a *p*, (x_0, y_0) képpont körül t_0 időpillanatban csoportosuló képponttömböt. Ezt a tömböt a $t_0 + D_t$ időpillanatban a különböző ($x_0 + D_x$, $y_0 + D_y$) pozíciójú q_i képpontjelöltek körül csoportosuló képponttömbökkel össze kell hasonlítani. A hasonlóság egyik szóba jöhető mértéke a **különbségek négyzetösszege** (**sum of squared differences**, **SSD**):

Created by XMLmind XSL-FO Converter.

$$SSD(D_x, D_y) = \sum_{(x,y)} \left(I(x,y,t) - I(x + D_x, y + D_y) \right)^2$$

24.7. ábra - (a) Rubik-kocka egy forgó lemezkorongon. (b) Ugyanaz a kocka 19/30 másodperccel később (Richard Szeliski nyomán). (c) Az (a) és (b) képek összehasonlítása alapján számolt optikai folyam vektorok (Joe Weber és Jitendra Malik hozzájárulásával).



(a) (b) (c)

Az (x, y) az (x_0, y_0) képpont körüli tömb képpontjainak halmazán fut végig. Megkeressük most azt a (D_x, D_y)-t, amely az SSD-t minimizálja. Az (x_0, y_0)-beli optikai folyam ilyenkor: $(v_x, v_y) = (D_x/D_t, D_y/D_t)$. Alternatívaként maximalizálhatjuk a **keresztkorrelációt** (**cross-correlation**):

$$Korreláció(D_x, D_y) = \sum_{(x,y)} I(x,y,t) I(x + D_x, y + D_y, t + D_t)$$

A keresztkorreláció akkor használható a legjobban, ha a jelenet képében textúra található, mert ekkor a vizsgált ablakokban nagy a fényességváltozás a képpontok mentén. Ha egy egyenletesen fehér falra nézünk, a különböző q jelöltekre számított keresztkorreláció majdnem azonos lesz, és az algoritmus a gyakorlatban véletlenszerűen fog választani.

Tegyük fel, hogy a megfigyelő **T** transzlációs és szögsebességgel rendelkezik [amelyek így a **sajátmozgás**t (**egomotion**) írják le]. Származtatni lehet olyan összefüggéseket, amelyek a megfigyelő sebességét, az optikai folyamot és a jelenet tárgyainak a pozícióját kapcsolják össze. Tegyük fel, hogy $f = 1$, akkor:

$$v_x(x,y) = \left[-\frac{T_x}{Z(x,y)} - \omega_y + \omega_z y \right] - x \left[-\frac{T_z}{Z(x,y)} - \omega_x y + \omega_y x \right]$$

$$v_y(x,y) = \left[-\frac{T_y}{Z(x,y)} - \omega_z x + \omega_x \right] - y \left[-\frac{T_z}{Z(x,y)} - \omega_x y + \omega_y x \right]$$

ahol $Z(x, y)$ az (x, y) képponthoz tartozó jelenetbeli pontnak a z koordinátája.

A képlet jobb megértését szolgálja, ha az egyszerű transzláció esetét nézzük. Ebben az esetben a folyammező:

$$v_x(x,y) = \frac{-T_x + xT_z}{Z(x,y)} \qquad v_y(x,y) = \frac{-T_y + yT_z}{Z(x,y)}$$

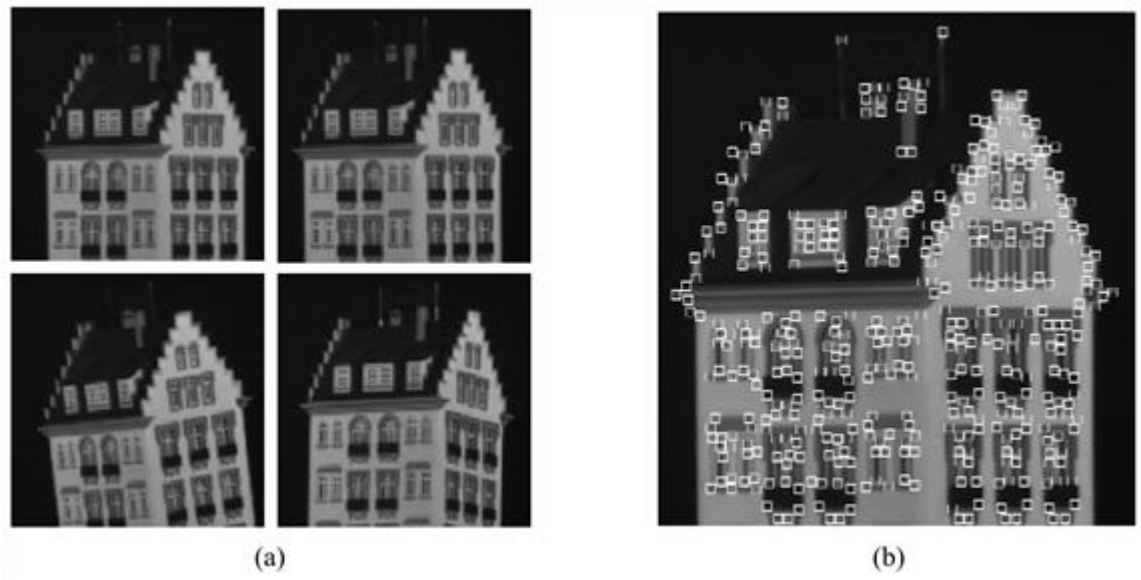
lesz. Néhány érdekes tulajdonságot figyelhetünk meg. Az optikai folyam mindkét komponense, $v_x(x, y)$ és $v_y(x, y)$ is zérus az $x = T_x/T_z$, $y = T_y/T_z$ pontban. Ezt a pontot a folyammező **expanziófókusz**ának (**focus of expansion**) nevezzük. Tegyük fel, hogy az $x - y$ sík origóját az expanziófókuszba helyezzük át. Ilyenkor az optikai folyam kifejezései igen egyszerű formát öltenek. Jelölje (x', y') az $x' = x - T_x/T_z$, és az $y' = y - T_y/T_z$ által definiált új koordinátákat. Ekkor:

Created by XMLmind XSL-FO Converter.

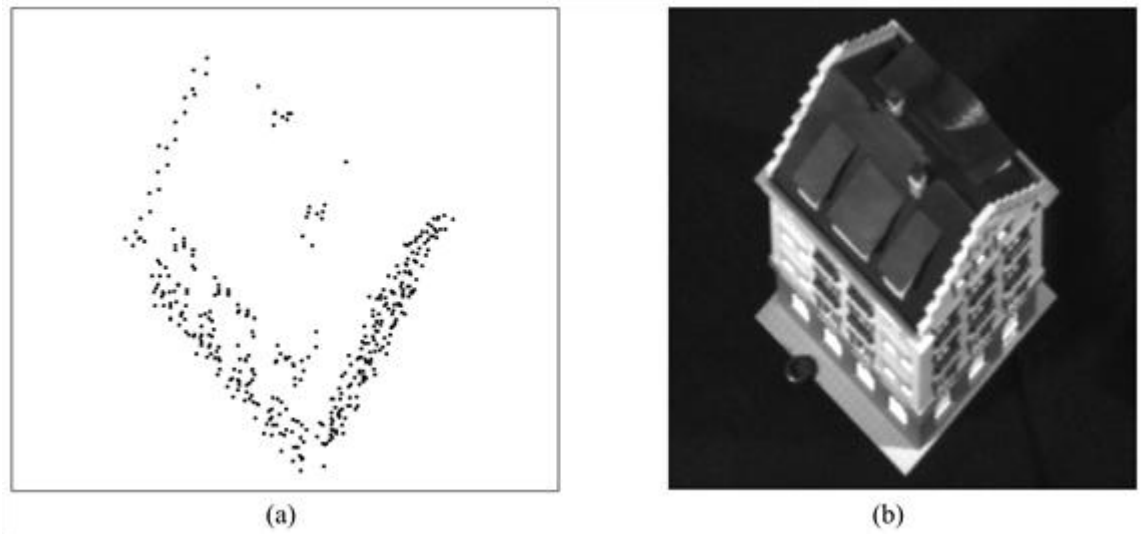
$$v_x(x',y') = \frac{x'T_Z}{Z(x',y')} \qquad v_y(x',y') = \frac{y'T_Z}{Z(x',y')}$$

Ennek az egyenletnek van néhány érdekes alkalmazása. Tegyük fel, hogy ön egy légy, amely egy falra próbál leszállni, és azt szeretné tudni, hogy adott aktuális sebesség mellett mennyi idő múlva ér a falhoz. Ezt az időt a Z/T_z adja meg. Figyeljük meg, hogy annak ellenére, hogy a pillanatnyi optikai folyam nem képes sem a Z távolságot, sem a T_z sebességi komponenst megadni, a kettő arányát meg tudja adni, és így a leszállás irányításában felhasználható. Az élő legyekkel való kísérletek alátámasztják, hogy a legyek pontosan ezt a mechanizmust használjak. A legyek a legmerészebb repülők az állatok és gépek között, és érdekes, hogy ezt egy olyan látórendszerrel teszik, amelynek borzasztóan gyenge a térbeli felbontása (körülbelül 600 érzékelője van, összehasonlítva az emberek 100 milliójával), de kiemelkedő az időbeli felbontása.

24.8. ábra - (a) Egy olyan videokép-sorozat négy kockája, amelyben a kamera a tárgyhoz képest mozgott és forgott. (b) A sorozat első képkockája, amelyen apró négyzetek jelzik a tulajdonságfelismerő által megtalált tulajdonságokat (Carlo Tomasi hozzájárulásával).



24.9. ábra - (a) A 24.8. ábrán látható kép tulajdonságai elhelyezkedésének háromdimenziós rekonstrukciója. (b) A valódi ház ugyanabból a perspektívából.



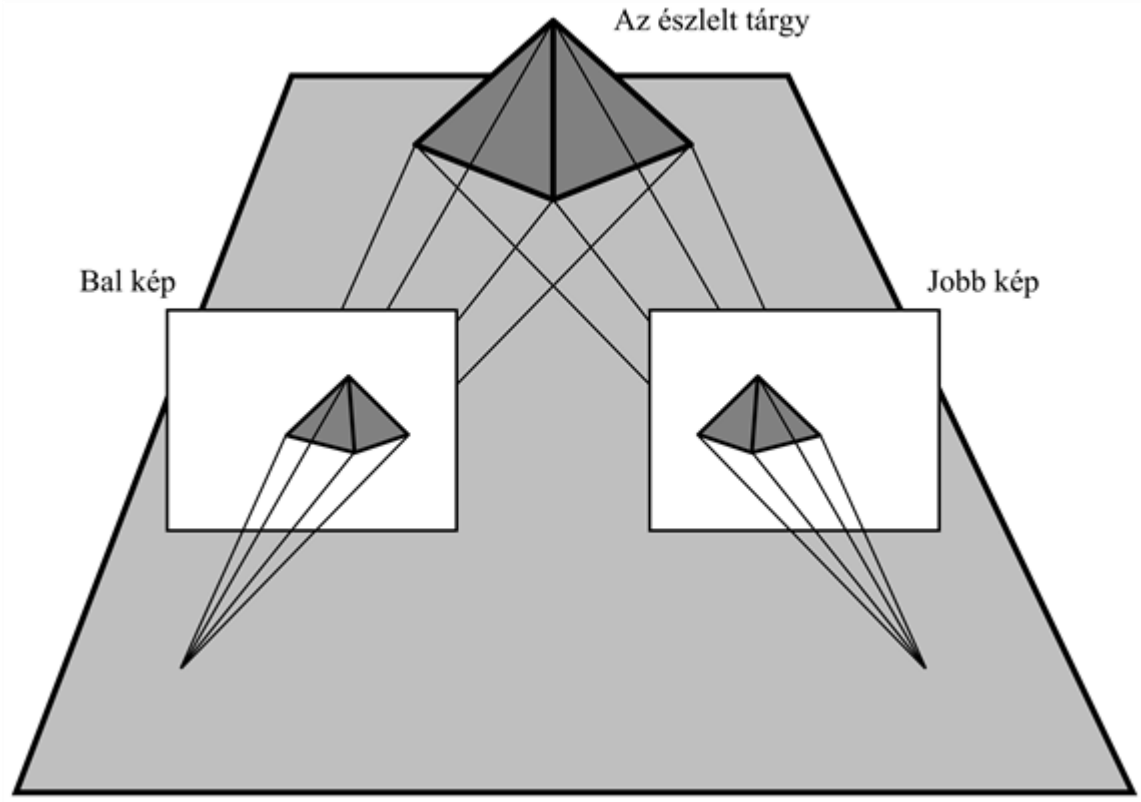
Created by XMLmind XSL-FO Converter.

A mélység kiszámításához több képkocka használata szükséges. Ha a filmre egy merev testet veszünk fel, akkor ennek alakja képkockáról képkockára nem fog változni, és így az optikai folyam természeténél fogva zajos méréseit jobban kezelhetjük. Az egyik ilyen megközelítés eredményét a 24.8. és a 24.9. ábrán láthatjuk (Tomasi és Kanade, 1992).

4.2. Kétkamerás (binokuláris) térbeli látás

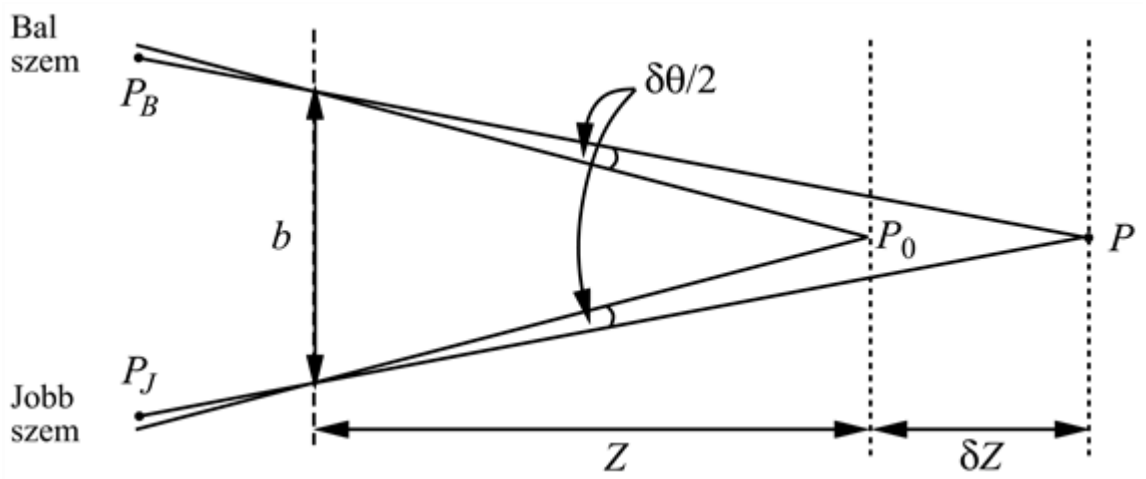
A legtöbb gerincesnek *két* szeme van. Ez hasznos redundancia arra az esetre, ha elveszítené az egyiket, de más módon is segít. A legtöbb zsákmányállatnak a feje oldalán vannak a szemei, hogy nagyobb területet beláthasson. A ragadozóknak elöl, hogy **kétkamerás térbeli látás**t (**binocular stereopsis**) valósíthassanak meg. Maga a gondolat a mozgási parallaxisra igen hasonlít, azonban az eltérő időpontokhoz tartozó képek helyett két (vagy több), térben szeparált képet használunk, hasonlóan ahhoz, mint amilyeneket pl. az előrenéző emberi szemek szolgáltatnak. Tekintettel arra, hogy a jelenet egy adott jellemzője az egyes képsíkok z koordinátájához képest más helyen lesz a képen, ha a két képet egymásra fektetjük, a képjellemző elhelyezkedésében a két kép különbséget – **diszparitás**t (**disparity**) – fog mutatni. Ez látszik a 24.10. ábrán, ahol a piramis legközelebbi pontja balra mozdult el a jobb oldali képen, és jobbra a bal oldalin.

24. ábra - A térbeli látás alapgondolata: különböző kamerapozíciók ugyanannak a háromdimenziós jelenetnek kissé eltérő kétdimenziós nézeteit eredményezik



24.11. ábra - A diszparitás és a mélység összefüggése a sztereolátásban

Created by XMLmind XSL-FO Converter.



Dolgozzuk ki a diszparitás és a mélység geometriai kapcsolatát. Először azt az esetet vizsgáljuk meg, amikor a két szem (illetve a két kamera) előrenéz, és optikai tengelyük párhuzamos. A jobb és a bal kamera kapcsolata így egy *b* nagyságú (bázisvonal) transzláció az *x* koordináta-tengely mentén. A $H = v_x \Delta t$ vízszintes és $V = v_y \Delta t$ függőleges képeltérés számításához az optikai folyam előbb megismert kifejezéseit használhatjuk, $T_x = b/\Delta t$, $T_y = T_z = 0$ mellett. Az ω_x, ω_y, és ω_z rotációs paraméterek zérusok. Azt kapjuk eredményül, hogy $H = b/Z$ és $V = 0$. Szavakkal megfogalmazva ez azt jelenti, hogy a vízszintes képeltérés a bázisvonal és a mélység aránya, a függőleges eltérés pedig zérus.

Normális látási körülmények között az emberek **fixál**nak, azaz **összetartóan néznek előre** (**fixate**), vagyis a jelenetben létezik egy olyan pont, amelyben a két optikai tengely metszi egymást. A 24.11. ábra egy P_0 pontra fixáló szemeket mutat, amely pont a két szem középvonalától *Z* távolságban helyezkedik el. Az egyszerűség kedvéért *szögeltérést* fogunk számítani radiánban. Az összetartás P_0 pontjában az eltérés zérus. A jelenet egy *δZ*-vel távolabbi *P* pontja esetén számíthatjuk a *P* pont P_L bal és a P_R jobb oldali képének a szögeltolódását. Ha azok mindegyike a P_0-hoz képest *δθ/2* szöggel eltolt, akkor a P_L és a P_R eltolódása, ami a *P* képeltérése, pontosan *δθ*. Egyszerű geometriai átalakításokból azt fogjuk kapni, hogy:

$$\frac{\delta\theta}{\delta Z} = \frac{-b}{Z^2}$$

Embereknél a *b* **bázisvonal** (**baseline**) kb. 6 cm. Tegyük fel, hogy *Z* kb. 100 cm. A legkisebb detektálható *δθ* (ami a képpont nagyságával függ össze) kb. 5 szögmásodperc, avagy $2{,}42 \times 10^{-5}$ radián, amiből a *δZ* 0,4 mm-re adódik. *Z* = 30 cm-re *δZ* = 0,036 mm-es lenyűgöző értéket kapunk. Ez azt jelenti, hogy 30 cm-es távolságban az emberek akár a 0,036 mm-es képmélység különbség megkülönböztetésére is képesek. Ez teszi lehetővé, hogy be tudjuk fűzni a cérnát egy tűbe, és hasonló műveleteket tudunk elvégezni.

4.3. Textúragradiensek

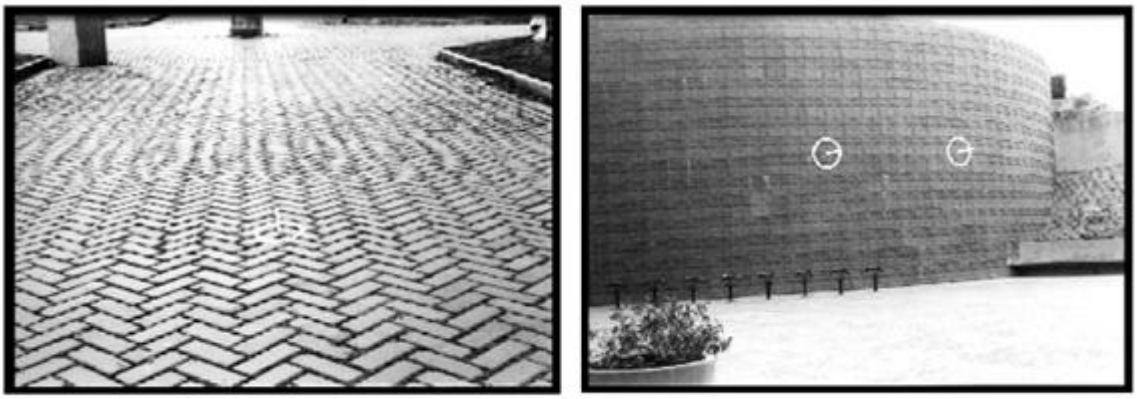
A mindennapi nyelvben a **textúra** vagy **felületmintázat** (**texture**) a felület kitapintható tulajdonságaira vonatkozik (az angol nyelvben ugyanazzal a szótővel rendelkezik, mint maga a szövet[3]). A számítógépes látásban a textúra egy ehhez közeli fogalomra, egy felületnek a síkban ismétlődő, vizuálisan érzékelhető mintázatára vonatkozik. Példaként említhetjük egy épület ablakainak a mintázatát, egy szvetter kötési mintáját, a leopárd foltos bundáját, a pázsit fűszálainak a mintázatát, kavicsokat a strandon vagy embertömeget egy stadionban. Az elrendezés néha közel periodikus, mint például a kötési szemek a szvetteren, más esetekben, mint például a strandon lévő kavicsok esetén, a szabályosság csak statisztikai értelemben létezik – a kavicsok sűrűsége a strand különböző részein nagyjából azonos.

24.12. ábra - (a) A textúragradienseket illusztráló jelenet. Feltéve, hogy a valós mintázat egyforma, lehetővé teszi a felület irányágának a meghatározását. A számított irányt a képen oly módon elhelyezett fehér kör és nyíl jelzi, mintha a kört a felületre festették volna abban a pontban. (b) Egy ívelt felület alakjának meghatározása a mintázat

[3] avagy „textile” (*A ford.*)

Created by XMLmind XSL-FO Converter.

alapján. (A képeket Jitendra Malik és Ruth Rosenholtz hozzájárulásával közöljük [Malik és Rosenholtz, 1994].)



(a) (b)

Amit most a *jelenetre* vonatkozóan állítottunk, igaz. A képen a textúraelemek vagy **texel**ek (**texel**s) látszólagos nagysága, alakja, távolsága stb. igencsak változik, ahogy ezt a 24.12. ábra mutatja. A csempelapok a jeleneten azonosak. A lapok kivetített nagyságának és alakjának megváltozásában két fő ok játszik szerepet:

1. *Az egyes texeleknek a kamerától vett változó távolsága*. Emlékezzünk arra, hogy a perspektivikus vetítésben a távoli tárgyak kisebbnek tűnnek. A skálatényező $1/Z$.

2. *Az egyes texelek változó rövidülése*. Ez a texeleknek a kamerából kiinduló rálátási irányhoz viszonyított orientációján múlik. Rövidülés nincs, ha a texel a rálátási irányra merőleges. A rövidülés mértéke cos σ-val arányos, ahol a σ a texel síkjának a lejtése.

Egy kevés matematikai elemzést követően meghatározhatjuk a különböző texeltulajdonságok, mint pl. a terület, a rövidülés és a sűrűség változási mértékének kifejezéseit. Ezek az ún. **textúragradiensek** (**texture gradients**) a felület alakjának és a megfigyelőhöz képesti dőlésének és lejtésének a függvényei.

Ahhoz, hogy az alakot a textúrából kinyerhessük, kétlépcsős eljáráshoz kell folyamodnunk: (a) először mérjük a textúragradienseket; (b) majd becsüljük a mért gradienseket okozni látszó felületalakot, azok dőlését és lejtését. A 24.12. ábra mutatja ezen eljárás eredményeit.

4.4. Árnyalás

Az **árnyalás**t (**shading**) – a jelenethez tartozó felület különböző részeiről kapott megvilágítás intenzitásváltozását – a jelenet geometriája és a felület-visszaverési tulajdonságai határozzák meg. A számítógépes grafikában a feladat a kép $I(x,y)$ fényességfüggvényének a megadása, ha a jelenet geometriája és a visszaverődési tulajdonságok adottak. A számítógépes látásban azt reméljük, hogy ez a folyamat invertálható, vagyis visszaállíthatjuk a jelenet geometriáját és visszaverődési tulajdonságait, ha az $I(x,y)$ képfényesség adott. A probléma olyan nehéznek bizonyult, hogy a legegyszerűbb esetektől eltekintve e téren nem sikerült eredményeket elérni.

Induljunk ki egy olyan példából, ahol az alakot tényleg visszanyerhetjük az árnyalásból. Tekintsünk egy Lambert-féle felületet, amit egy távoli, pontszerű fényforrás világít meg. Tételezzük fel azt is, hogy a felület messze van a kamerától, így függőleges síkú vetítést fogunk használni a perspektivikus vetítés megközelítésére. A kép fényessége:

$$I(x, y) = kn(x, y) \cdot s$$

ahol k egy skálaegyüttható, **n** a felület normál egységvektora és **s** a fényforrás irányába mutató egységvektor. Mivel **n** és **s** egységvektorok, skalárszorzatuk a kettő közötti szög koszinusza. A felület alakját az **n** felület menti változása tartalmazza. Tételezzük fel, hogy k és **s** ismertek. A problémánk most a felület **n** normálisának a kiszámítása, ha az $I(x,y)$ képfényesség adott.

Az első megjegyzés, amit meg kell tenni, az az, hogy az **n** meghatározása az adott (x,y) képpontfényesség ismeretében lokálisan alulhatározott. Ki tudjuk ugyan számítani az **n** és a fényforrás közötti szöget, ez azonban

Created by XMLmind XSL-FO Converter.

az **n**-et csak annyiban korlátozza, hogy egy bizonyos, **s** irányú és $\theta = \cos^{-1}(I/k)$ palástszögű kúp felületén helyezkedik el. Hogy továbbléphessünk, jegyezzük meg, hogy képpontról képpontra az **n** változása nem lehet tetszőleges. Egy sima felület normálisa csak sima módon változhat – amely korlátozást az **integrálhatóság** (**integrability**) szakkifejezéssel jelöljük. Ezt a felismerést számos módszerben fel is használják. Egy lehetséges eljárás az **n** átírása a $Z(x,y)$ mélység Z_x és Z_y parciális deriváltjainak felhasználásával. Ennek eredménye Z egy parciális differenciálegyenlete, amit megfelelő határfeltételek figyelembevételével $Z(x,y)$-ra nézve meg lehet oldani.

Az eljárás általánosítható. Nem szükséges, hogy a felület Lambert-féle legyen, és az sem, hogy a fényforrás pontszerű legyen. Lényegében ugyanez a módszer használható, ha sikerül a felület $R(\mathbf{n})$ **reflektanciatérképét** (**reflectance map**) meghatározni, amely a felületelem fényességét annak **n** normálisa függvényében fejezi ki.

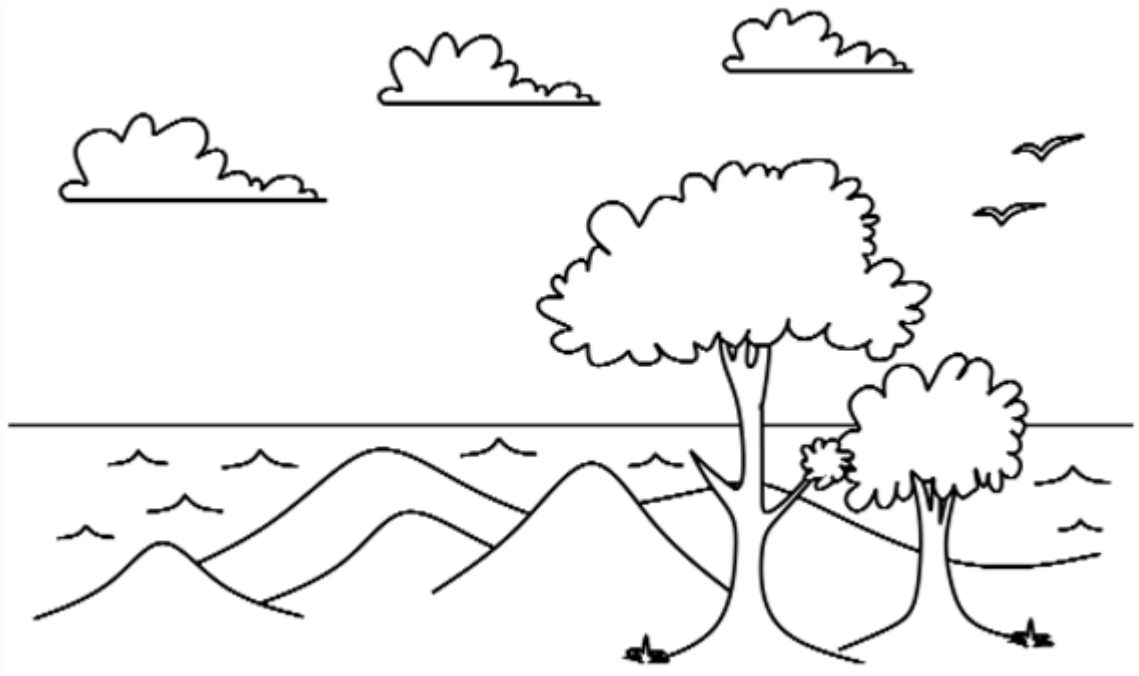
Az igazi nehézség a kölcsönös visszaverődések kezelése. Egy tipikus belsőtérjelenet esetén, mint például egy irodában lévő tárgyak esetén a felületeket nem csupán a fényforrások világítják meg, hanem a jelenet más felületeitől visszavert fény is, amelyek tényleges másodlagos fényforrásokként szolgálnak. Ezek a kölcsönös megvilágítási hatások igen jelentősek lehetnek. Az ilyen helyzetben a reflektanciatérkép megközelítés teljes mértékben használhatatlan – a kép fényessége nemcsak a normálison múlik, hanem a jelenet különböző felületeinek bonyolult térbeli viszonyain is.

Az ember az árnyalásból képes valamilyen mértékű alakinformáció kinyerésére, így ez a probléma – minden nehézsége ellenére – továbbá is érdekes marad.

4.5. Kontúrok

Amikor egy olyan vonalas rajzot nézünk, mint amilyen a 24.13. ábra, a kép egy háromdimenziós elrendezés életszerű benyomását kelti bennünk. Hogyan lehetséges ez? Hiszen láttuk korábban, hogy egy és ugyanaz a vonalas kép végtelen sok jelenet konfigurációból származhat. Figyeljük meg, hogy még a felszín lejtésének és emelkedésének érzetét is megkapjuk. Ez minden bizonnyal (a tipikus alakzatokra vonatkozó) magas szintű tudás és valamilyen alacsony szintű kényszerek összekapcsolásának következtében lehetséges.

24.13. ábra - Egy felidéző vonalas rajz (Isha Malik hozzájárulásával)



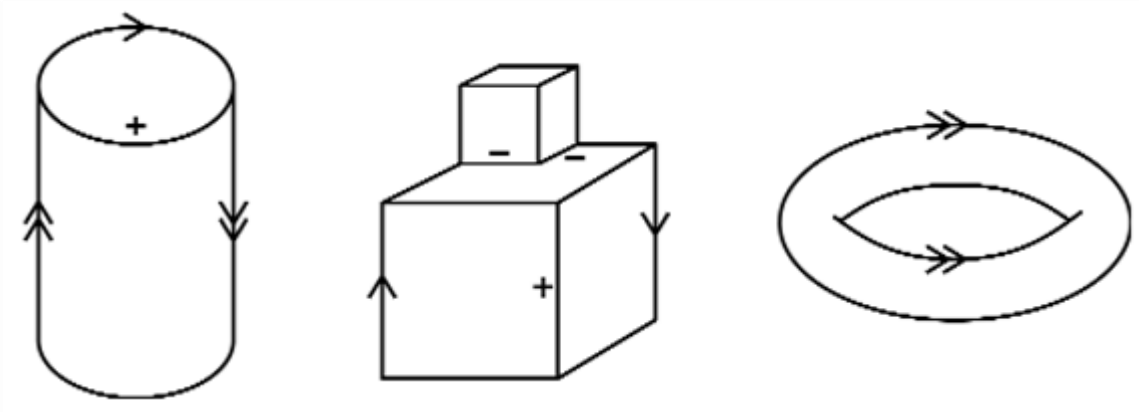
A vonalas ábrában rejlő kvalitatív tudással fogunk foglalkozni. Korábban láttuk, hogy a rajz vonalainak különféle lehet a fontossága (lásd 24.4. ábra és kísérőszövege). Az a folyamat, ahogy a kép minden vonalának a tényleges fontosságát megállapítjuk, a **vonalcímkézés** (**line labeling**), és ez volt a számítógépes látás által elsőként tanulmányozott problémák egyike. Tegyük fel egyelőre, hogy a világ olyan leegyszerűsített modelljével dolgozunk, ahol a tárgyaknak nincsenek felületei, és ahol az olyan vonalakat, amelyek megvilágítási diszkontinuitásokból adódnak, mint amilyenek az árnyékélek és a tükrözések, valamilyen

Created by XMLmind XSL-FO Converter.

előfeldolgozó eljárással a képből kiemeltük. A figyelmünket így olyan vonalas ábrákra összpontosíthatjuk, ahol minden vonal vagy mélységi, vagy orientáció diszkontinuitáshoz tartozik.

Mindegyik vonalat ezek után vagy egy a felület határán megjelenő **pszeudoél** (**limb**) vetületének (ami a felület azon pontjainak összessége, ahol a rálátás iránya a felület érintője), vagy egy **él**nek (**edge**) lehet tekinteni (ami a felület normálisának egy diszkontinuitása). Az éleket továbbá konvex, konkáv vagy határoló élosztályokba sorolhatjuk. A pszeudoélek és a határoló élek esetén érdekes annak az eldöntése, hogy a vonalas ábrán a görbét határoló két felületből melyik van hozzánk közelebb. Ezeket a következtetéseket úgy képzelhetjük el, hogy mindegyik vonalhoz a 6 lehetséges **vonalcímké**nek (**line label**) az egyikét rendeljük hozzá, ahogy az a 24.14. ábrán látható.

24.14. ábra - Különféle vonalcímkék



1. A „+” és a „–” címkék rendre a konvex és a konkáv éleket jelentik. Ezeket az olyan normális diszkontinuitásokhoz rendeljük, ahol az él mentén találkozó mindkét felület látható.

2. A „←” és a „→” határoló konvex éleket jelentenek. A kamerával felülről nézve az él mentén találkozó mindkét felületelem azonos oldalon fekszik úgy, hogy az egyik eltakarja a másikat. Ha a nyíl irányába mozgunk, a felületek hozzánk képest jobbra esnek.

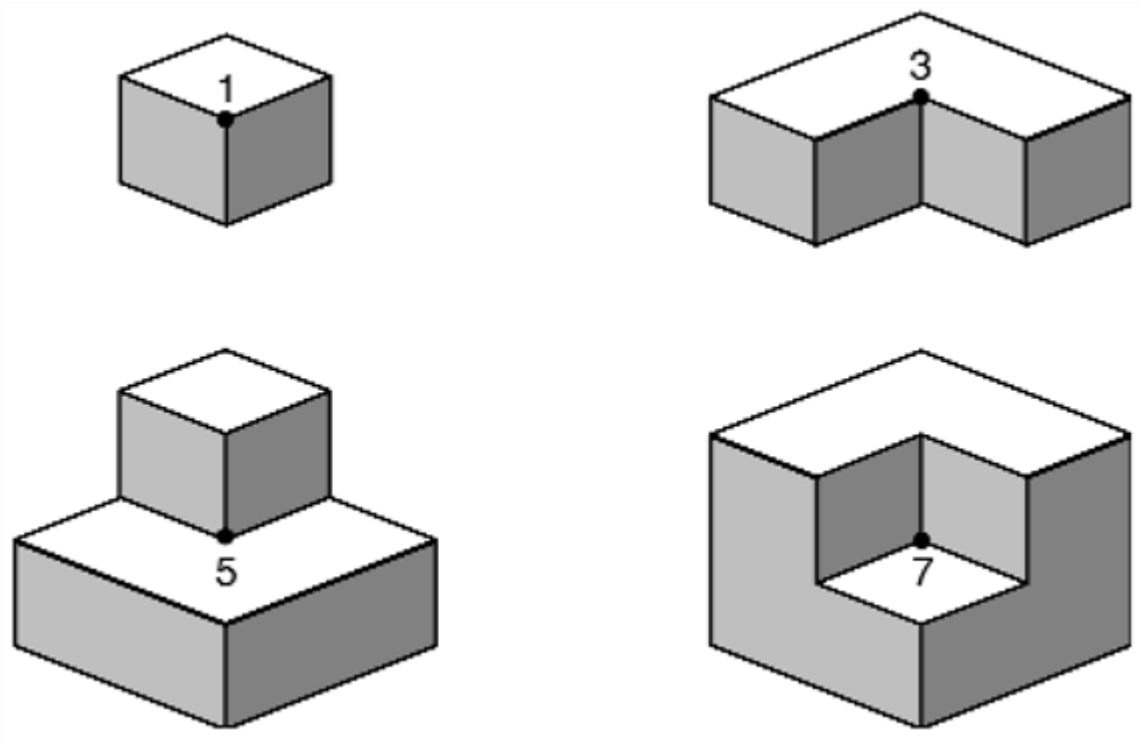
3. A „←←” és a „→→” címkék a pszeudoéleket jelölik. Az ilyen helyeken a felület sima módon görbül, és saját magát takarja. Ha a dupla nyíl irányába lépünk, a felület tőlünk jobbra helyezkedik. A rálátás iránya a pszeudoél minden pontjában a felület érintője. Ahogy a nézőpont változik, a pszeudoél a felület mentén mozog.

Egy ábrán található *n* számú vonal kombinatorikusan elvileg lehetséges címke-hozzárendelésből csupán csekély számú hozzárendelés lehetséges fizikailag. Ezeknek a címkézéseknek a megállapítása az ún. vonalcímkézési probléma. Jegyezzük meg, hogy a probléma csak akkor értelmes, ha a címke a vonal mentén nem változik. Ez nem mindig igaz, hiszen a görbült objektumok képein a címke egy vonal mentén igenis változhat. Ebben a részben ez nem jelent problémát, mert csak poliéderes objektumokkal foglalkozunk.

A poliéderes jelenetelemzéssel először Huffman és tőle függetlenül Clowes próbálkoztak (Huffman, 1971; Clowes, 1971). Huffman és Clowes elemzésüket a nem átlátszó **triéderes** (**trihedral**) testekből – azaz olyan testekből, amelyeknél mindegyik sarokpontjánál pontosan három felület fut össze – álló jelenetek esetére korlátozták. A több objektumot tartalmazó jelenetek esetén kizárták a triéderes feltételt megsértő objektumkonfigurációkat is, mint amilyen például az egyik élük mentén érintkező két kocka. Nem engedték a **repedés**eket (**crack**s) sem, azaz olyan „éleket”, amelyek mentén a tangenciális síkok folytonosan változnak. A triéderes világban Huffman és Clowes kimerítő listát adtak az összes különböző sarokponttípusról meg arról is, hogy azok hogyan látszhatnak általános nézőpontból szemlélve. Az általános nézőpont feltétel lényegében azt tételezi fel, hogy az összes csatlakozás megtartja a jellegét, ha a szem csak kissé mozdul el. Ha például a képen három vonal metszi egymást, akkor a feltételből következik, hogy a jelenet hozzájuk tartozó éleinek szintén metszeniük kell egymást.

24.15. ábra - A triéderes csúcsok négy fajtája

Created by XMLmind XSL-FO Converter.

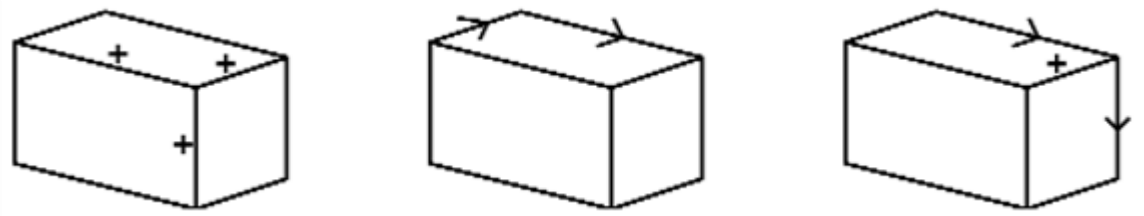


A 24.15. ábra mutatja azt a négyféle helyzetet, ahogy a három síkfelület egy sarokpontban találkozhat. Ezeket az eseteket úgy szerkesztettük, hogy egy kockát nyolc **oktáns**ra (**octant**) bontottunk. A triéderes sarokpontok egész választékát szeretnénk generálni a kocka középpontjában úgy, hogy a különböző oktánsokat kitöltjük. Az 1-es címkéjű sarokpont az egyetlenegy kitöltött oktánshoz tartozik, a 3-as címkéjű a három kitöltött oktánshoz és így tovább. Az olvasónak kellene belátnia, hogy ezek tényleg kimerítik *az összes* lehetséges helyzetet. Ha valaki például a kockában két oktánst kitölt, akkor nem lesz képes a középpontban egy érvényes triéderes sarokpontot megszerkeszteni. Jegyezzük meg azt is, hogy ezen négy eset a sarokpontban találkozó konvex és konkáv élek különböző kombinációinak felel meg.

A sarokpontban találkozó három él a körülöttük lévő teret nyolc oktánsra osztja. A sarokpontot bármely, anyaggal nem kitöltött oktánsból lehet szemlélni. A nézőpont oktánson belüli elmozdítása a képen látható csatlakozások jellegét megváltoztatni nem fogja. A 24.15. ábrán látható 1-es címkéjű sarokpontot a maradó hét oktánsból szemlélve a 24.16. ábrán látható csatlakozáscímkéket kapjuk.

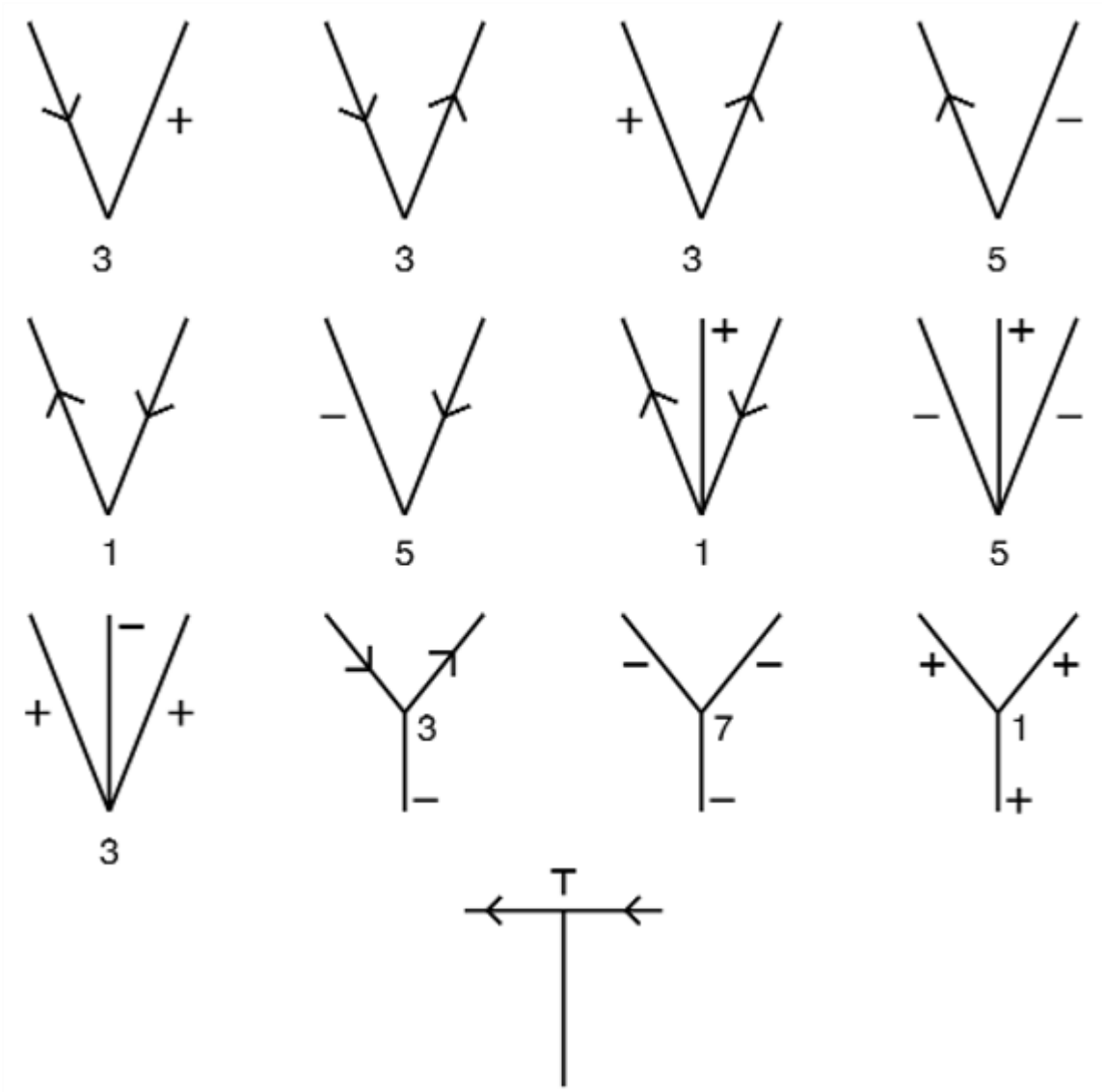
Annak kimerítő vizsgálata, hogy egy sarokpont milyen módon látható, a 24.17. ábrán felsorolt lehetőségekhez vezet. A képen négy különböző csatlakozástípust azonosíthatunk: L, Y, nyíl- és T csatlakozást. Az L csatlakozás a két látható él esete. Az Y és a nyílcsatlakozás három élnek felel meg. Az Y csatlakozásban a szögek egyike sem lépi túl a 180°-ot. A T csatlakozás a takarással kapcsolatos. Ha egy közeli, nem átlátszó felület egy távolabbi élt eltakar, egy fél éllel találkozó folytonos élt kapunk. A T csatlakozás négy címkéje a négyféle él takarásának felel meg.

24.16. ábra - A 24.15. ábrán 1-gyel jelölt csúcs különböző előfordulásai



24.17. ábra - A Huffman–Clowes-címkekészlet

Created by XMLmind XSL-FO Converter.



Ha egy vonalas rajzot ennek a csatlakozásszótárnak a segítségével címkézünk meg, a probléma annak a megállapítása, hogy mely csatlakozásinterpretációk lesznek globálisan konzisztensek. A konzisztenciát az a szabály kényszeríti ki, miszerint a rajzon egy vonalnak egy és csakis egy címkéje lehet a vonal egész hosszában. Ezen probléma algoritmikus megoldását Waltz adta meg (pontosabban egy olyan kibővített probléma megoldását adta meg, amely az árnyékot, a görbületi diszkontinuitásokat [repedéseket] és a szétválaszthatóan konkáv éleket is figyelembe veszi [Waltz, 1975]). Az algoritmus a kényszerkielégítés egyik legelső alkalmazása volt az MI területén (lásd 5. fejezet). A kényszerkielégítés nyelvén a változók a csatlakozások, a változók értékei a csatlakozások címkézése, a kényszer pedig az, hogy minden vonalnak csak egy címkéje van. Bár a triéderes vonalcímkézési probléma NP-teljes, a közönséges kényszerkielégítési algoritmusok a gyakorlatban jól működnek.

5. Objektumok felismerése

A látás lehetővé teszi számunkra, hogy megbízhatóan felismerjünk embereket, állatokat és élettelen dolgokat. Az MI-ben, illetve a számítógépes látásban az *objektumfelismerés* fogalmát szokás szerint mindezen dolgokra alkalmazzák. Ez magában foglalja a képen rögzített objektumok osztályának felismerését – például: ez egy arc, illetve adott objektumok felismerését is – például: ez Bill Clinton arca. A motivációt ilyen alkalmazási területek jelentik:

- **Biometrikus azonosítás** (**biometric identification**): a bűnügyi vizsgálatok és a védett létesítmények belépésszabályozása megköveteli az egyének azonosítását. Az ujjlenyomatok, az íriszlenyomat és az arcképek olyan képeket eredményeznek, amelyeket adott egyénekhez kell rendelni.

Created by XMLmind XSL-FO Converter.

- **Tartalomalapú képkeresés** (**content-based image retrieval**): könnyű egy dokumentumban megtalálni azt a helyet – ha létezik –, ahol a „macska” szó szerepel. Bármelyik szövegszerkesztő képes erre. Most gondoljon arra a feladatra, hogy egy képen megtalálja a képpontok azon részhalmazát, amely egy macska képének felel meg. Ha valaki rendelkezne ezzel a képességgel, akkor megválaszolhatna olyan kereséseket, mint például „Bill Clinton és Nelson Mandela együtt”, egy „görkoris a levegőben”, „az Eiffel-torony éjszaka” stb., anélkül hogy egy gyűjtemény minden egyes fényképéhez feliratkulcsszavakat kellett volna rendelnünk. Ahogy a kép- és videogyűjtemény növekszik, a kézi feliratozás lehetetlenné válik.

- **Kézírás-felismerés** (**handwriting recognition**): például az aláírások, borítékok címmezői, csekkeken szereplő összegek és a kéziszámítógépek tollalapú beviteli megoldása.

A látást nemcsak objektumok, hanem cselekvések felismerésére is használjuk. Felismerhetünk járásmódokat (egy barát járását), kifejezéseket (egy mosolyt, egy grimaszt), gesztikulációt (egy integető embert), cselekvéseket (ugrást, táncot) és így tovább. A cselekvésfelismerést célzó kutatás még gyerekcipőben jár, így ebben az alfejezetben az objektumfelismerésre koncentrálunk.

A vizuális objektumok felismerésének feladata általában könnyű az emberek számára, de a számítógépek számára igen nehéznek bizonyult. Fel akarjuk ismerni egy ember arcát függetlenül a megvilágítás, a kamerához képesti helyzet és az arckifejezés különböző variációitól. Ezen variációk bármelyike széles körű változásokat eredményez a képpontok fényességértékeiben, így a képpontok közvetlen összehasonlítása valószínűleg nem fog működni. Ha valaki egy kategória (például az „autó”) példányait szeretné felismerni, akkor a kategórián belüli variációkat is figyelembe kell vennie. Még a postai irányítószámoknál a kézzel írt számjegyek felismerésének igen korlátozott feladata is komoly kihívásnak bizonyult.

A felügyelt tanulás vagy a mintaosztályozás természetes keretet biztosít az objektumfelismerés tanulmányozásához. Pozitív („arcok”) és negatív példaként („nem arcok”) adott képek esetében a cél egy olyan függvény megtanulása, amely az új képeket felcímkézi az *arc* és a *nem arc* jelölésekkel. A 18. és 20. fejezet összes technikája lehetséges jelölt a feladatra: többrétegű perceptronokat, döntési fákat, legközelebbi-szomszéd osztályozókat és kernelgépeket alkalmaztak már objektumfelismerési feladatokra. Meg kell azonban jegyeznünk, hogy ezen technikák alkalmazása objektumok felismerésére távolról sem egyértelmű.

Az első kihívás a kép szegmentálása. Minden kép tipikusan több objektumot fog tartalmazni, így először olyan képpontrészhalmazokra kell osztanunk, amelyek egy-egy objektumnak felelnek meg. Miután a képet régiókra osztottuk, utána a régiókat vagy az azokból összeállított egységeket átadjuk egy osztályozónak, hogy meghatározza az objektumok címkéit. Sajnálatos módon a lentről felfelé szegmentáció hibákra érzékeny eljárás, ezért alternatív megoldásként kínálkozik a fentről lefelé objektumcsoport meghatározása. Eszerint keresni kell egy képpontrészhalmazt, amit arcként lehet osztályozni, és ha ez sikerült, máris van egy csoportunk! A tisztán fentről lefelé módszerek nagyon számításigényesek, mivel különböző méretű képablakokat kell megvizsgálni különböző helyszíneken, és az összes különböző objektumhipotézissel össze kell vetni őket. Jelenleg a leginkább gyakorlati objektumfelismerő rendszerek ilyen fentről lefelé stratégiát alkalmaznak, bár ez megváltozhat, ahogy a lentről felfelé technikák fejlődnek.

A második kihívás annak biztosítása, hogy a felismerés folyamata kellően robusztus a megvilágítás és az elhelyezkedés változásaira. Az emberek fel tudnak ismerni objektumokat a képpontok fényességértékeivel mért pontos megjelenés jelentős változásai ellenére is. Például egy barát arcát különböző fényviszonyok mellett vagy különböző látószögekből is felismerjük. Egy ennél is egyszerűbb példaként megvizsgálhatjuk a kézzel írt 6-os számjegy felismerését. Képesnek kell lenni a 6-os felismerésére különböző méretekben, a kép különböző pozícióiban és a kép kisebb elforgatásai ellenére is.[4]

A kulcsmomentum azt észrevenni, hogy a geometriai transzformációk, mint például az elfordítás, a méretváltás és a forgatás, vagy a kép fényességének megváltozásai a fényforrások fizikai mozgatása által, más jellemzőkkel rendelkeznek, mint a kategórián belüli variációk, mint például az emberek arcainak különbözőségei. Nyilvánvaló, hogy a tanulás az egyetlen mód az emberi arcok különbözőségeinek megtanulására vagy a 4-es számjegy különböző írásmódjainak felismerésére. Másrészről a geometriai és fizikai transzformációk hatásai szisztematikusak és kiküszöbölhetők a tanuló minták paramétereit leíró tulajdonságok megfelelő tervezésével.

Annak érdekében, hogy a geometriai transzformációkra vonatkozó invarianciát biztosítsuk, egy egészen hatékonynak bizonyult eljárás a kép régiójának előfeldolgozása úgy, hogy standard pozícióba, méretre és orientációra alakítjuk. Egy alternatív megoldás az, ha egyszerűen figyelmen kívül hagyjuk a geometriai és

[4] A teljes elforgatásinvariancia nem szükséges és nem is kívánatos – ekkor a 9-es és a 6-os összekeverhető lenne!

Created by XMLmind XSL-FO Converter.

fizikai transzformációk alkalmi természetét, és úgy gondolunk rájuk, mint az osztályozó variálhatóságának egy újabb forrására. A tanító minták halmazában az összes ilyen variációra példákat kell elhelyeznünk, és abban reménykedünk, hogy az osztályozó kikövetkeztet egy megfelelő transzformációhalmazt a bemenetre, és így kiszűri a variációkat.

Vizsgáljunk meg most az objektumfelismerésre szolgáló speciális algoritmusokat! Az egyszerűség érdekében kétdimenziós összeállításban fogjuk a problémát vizsgálni, ahol mind a tanuló, mind a tesztminták kétdimenziós fényességábrák formájában adottak. Olyan területeken, mint például a kézírás-felismerés, ez nyilvánvalóan elegendő. Még háromdimenziós objektumok esetében is hatékony stratégia az objektumokat több kétdimenziós nézettel reprezentálni (lásd 24.18. ábra), és az új objektumokat a (valamilyen reprezentációjú) tárolt nézetekkel összehasonlítva osztályozni.

24.18. ábra - Két háromdimenziós tárgy többféle nézete



Az előző alfejezet megmutatta, hogy több tényező vesz részt a háromdimenziós információ kinyerésében egy jelenetből. Az objektumfelismerés is több tényezőn alapszik – egy tigrist a narancs és fekete színek keveredéséről, csíkos mintázatáról és testének alakjáról ismerünk fel.

A szín és a mintázat hisztogramok vagy empirikus frekvenciaeloszlások segítségével reprezentálható. Ha adott egy tigris képe mintaként, akkor megmérhetjük a különböző színű képpontok arányát. Ezután, ha egy ismeretlen példát látunk, összevethetjük a szín hisztogramját a korábbi tigrispéldáknál látottakkal. A mintázatok elemzéséhez a kép különböző irányú és skálázású szűrőkkel történt konvolúciója után kapott eredményeinek hisztogramjait vizsgáljuk, egyezést keresve.

Az alak felhasználása az objektumfelismerésben sokkal bonyolultabbnak bizonyult. Nagyjából két irányzatot különböztethetünk meg: **fényességalapú felismerés**t (**brightness-based recognition**), ahol a képpontok fényességértékeit direkt módon használjuk, és **jellemzőalapú felismerés**t (**feature-based recognition**), ahol kiemelt jellemzők, mint például az élek vagy a kulcspontok részleges elrendezését használjuk. Miután részletesebben megvizsgáltuk e két megközelítést, tárgyalni fogjuk a **pozícióbecslés**t (**pose estimation**), azaz az objektum helyzetét és irányát a jelenetben.

5.1. Fényességalapú felismerés

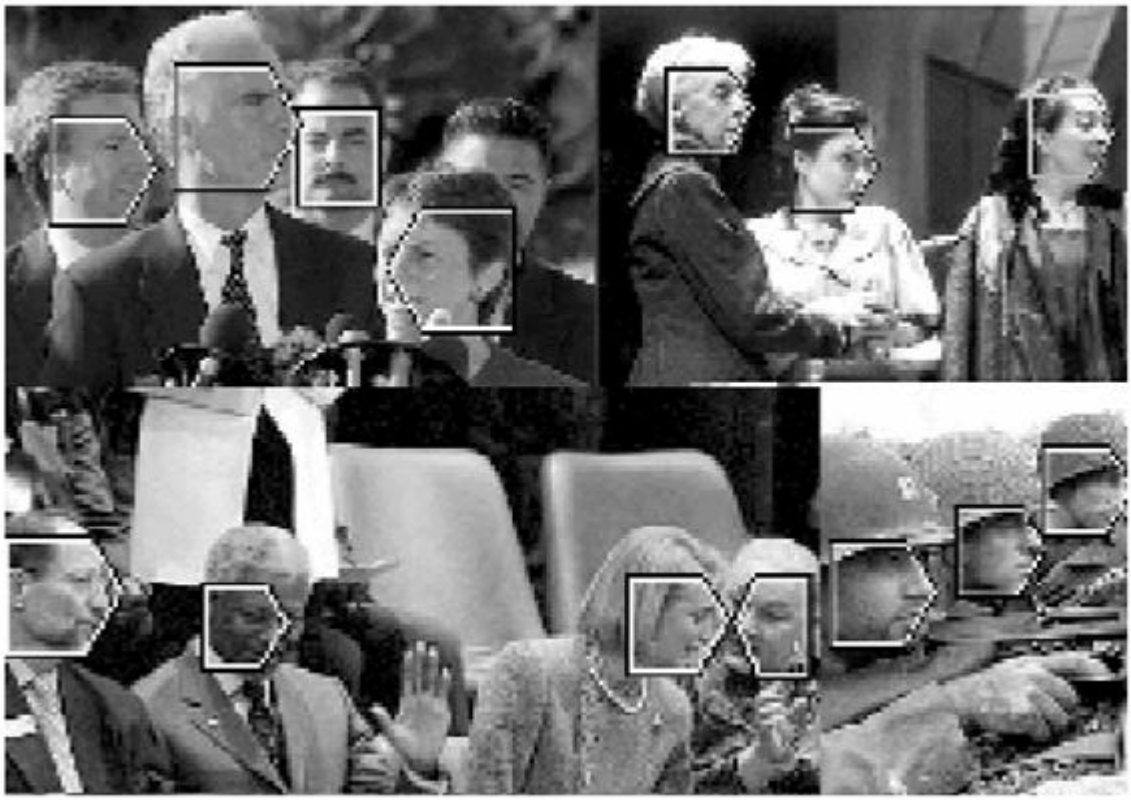
Ha adott egy képponthalmaz, amely egy lehetséges objektumnak felel meg, akkor válaszszuk a jellemzőknek magukat a képpontok fényességeit. Egy másik variáció szerint először konvolválhatjuk a képet többféle lineáris szűrővel, és az eredményül kapott kép pontjainak értékeit tekinthetjük a jellemzőknek. Ez a megközelítés különösen sikeres olyan feladatokban, mint a kézzel írt számjegyek felismerése, ahogy azt a 20.7. alfejezetben láthattuk.

Sokféle statisztikai módszert használtak arra, hogy képeket tartalmazó adatbázisok segítségével arcfelismerőket fejlesszenek ki, mint például feldolgozatlan képpontbemeneten működő neurális hálókat képpontbemenettel,

Created by XMLmind XSL-FO Converter.

vonal- és élszűrők által előállított jellemzőket használó döntési fákat és wavelet-jellemzőket használó, naiv Bayes-modelleket. Az utóbbi módszer néhány eredménye a 24.19. ábrán látható.

24.19. ábra - Egy arcmegtaláló algoritmus eredménye (Henry Schneiderman és Takeo Kanade hozzájárulásával)



A feldolgozatlan képpontok jellemző vektorként történő használatának egy negatív aspektusa az ebben a reprezentációban rejlő hatalmas redundancia. Vegyünk például két egymás melletti képpontot az arc pofa részén; valószínűleg igen erős közöttük a korreláció, hiszen hasonló a geometriájuk, megvilágításuk stb. Az adatmennyiséget csökkentő technikák, mint például a főkomponens-analízis, sikeresen használhatók a jellemzővektor dimenziószámának csökkentésére, lehetővé téve az olyan dolgok, mint például arcok, nagydimenziós terekben való felismerésénél nagyobb sebességű felismerését.

5.2. Jellemzőalapú felismerés

A feldolgozatlan képpontok fényességértékei jellemzőként történő felhasználása helyett felismerhetünk adott helyen levő jellemzőket, mint például régiókat vagy éleket (lásd 24.3. alfejezet). Két motivációnk van az élek használatára. Az egyik az adatmennyiség csökkentése – sokkal kevesebb él van, mint képpont. A másik a megvilágításfüggetlenség – a kontraszt egy megfelelő tartományában az éleket nagyjából azonos helyen ismerjük fel, függetlenül a tényleges fényforrás-konfigurációtól. Az élek egydimenziós jellemzők; kétdimenziós és nulladimenziós tulajdonságokat (régiókat, illetve pontokat) szintén használunk. Vegyük észre a térbeli elhelyezkedés kezelésében jelentkező különbséget a fényességalapú és a jellemzőalapú módszerek között! A fényességalapú módszerekben az (x,y) hely a jellemző.

Az élek elrendezése egy objektum tulajdonsága – ez az egyik oka, ami miatt a rajzokat könnyedén értelmezni tudjuk (lásd 24.13. ábra), még akkor is, ha ilyen képek nem fordulnak elő a természetben! Ezen tudás legegyszerűbb felhasználásának módja egy legközelebbi-szomszéd osztályozóval történhet. Előre kiszámítjuk és eltároljuk az ismert objektumok nézeteihez tartozó élkonfigurációkat. Ha adott egy ismeretlen objektumnak megfelelő élelrendezés a bemeneti képen, akkor meghatározhatjuk az eltárolt nézetekhez mért „távolságát”. A legközelebbi-szomszéd osztályozó a legközelebbi illeszkedőt választja.

Sokféle definíciót javasoltak a képek közötti távolságra. Az egyik legérdekesebb megközelítés a **deformációs illeszkedés** (**deformable matching**) ötletén alapul. D’Arcy Thompson klasszikus munkájában (*On Growth and*

Created by XMLmind XSL-FO Converter.

Form [D'Arcy Thompson, 1917]) megfigyelte, hogy a hasonló, de nem azonos alakzatok gyakran egybeesésbe hozhatók egyszerű koordináta-transzformációkkal.[5] E módszer szerint az alakzatok hasonlóságát egy három lépésből álló folyamattal határozzuk meg: (1) megoldjuk a két alakzat megfeleltetési problémáját, (2) a megfeleltetést felhasználjuk egy egymásra illesztési transzformáció megbecslésére, és (3) kiszámítjuk a két alakzat közötti távolságot az egymásnak megfelelő pontok illeszkedési hibáinak összegeként, kiegészítve a fedésbe hozó transzformáció nagyságával.

Az alakzatot a belső vagy külső körvonalát alkotó pontok diszkrét halmazaként reprezentáljuk. Ezeket előállíthatjuk egy éldetektáló által megtalált élképpontok helye alapján, így kapunk egy N pontból álló $\{p_1,\ldots, p_N\}$ halmazt. A 24.20. (a) és (b) ábra két alakzat mintapontjait mutatja.

Most tekintsünk egy p_i mintapontra és az onnan induló összes többi mintapontra mutató vektorra egy alakzaton. Ezek a vektorok kifejezik a teljes alakzat konfigurációját a referenciaponthoz viszonyítva. Ez a következő ötlethez vezet: rendeljünk minden mintaponthoz egy leírót, az **alakzatkontextus**t (**shape context**), amely megadja az alakzat további részének durva elrendezését az adott ponthoz viszonyítva. Pontosabban fogalmazva a p_i alakzatkontextusa a maradék $N - 1$ p_k pontra a $p_k - p_i$ relatív koordináták egy közelítő h_i térbeli hisztogramja. Egy logaritmikus poláris koordináta-rendszert használunk a tartók meghatározására annak biztosítása érdekében, hogy a leíró érzékenyebb legyen a közeli képpontok esetében. A 24.20. (c) ábra mutat erre egy példát.

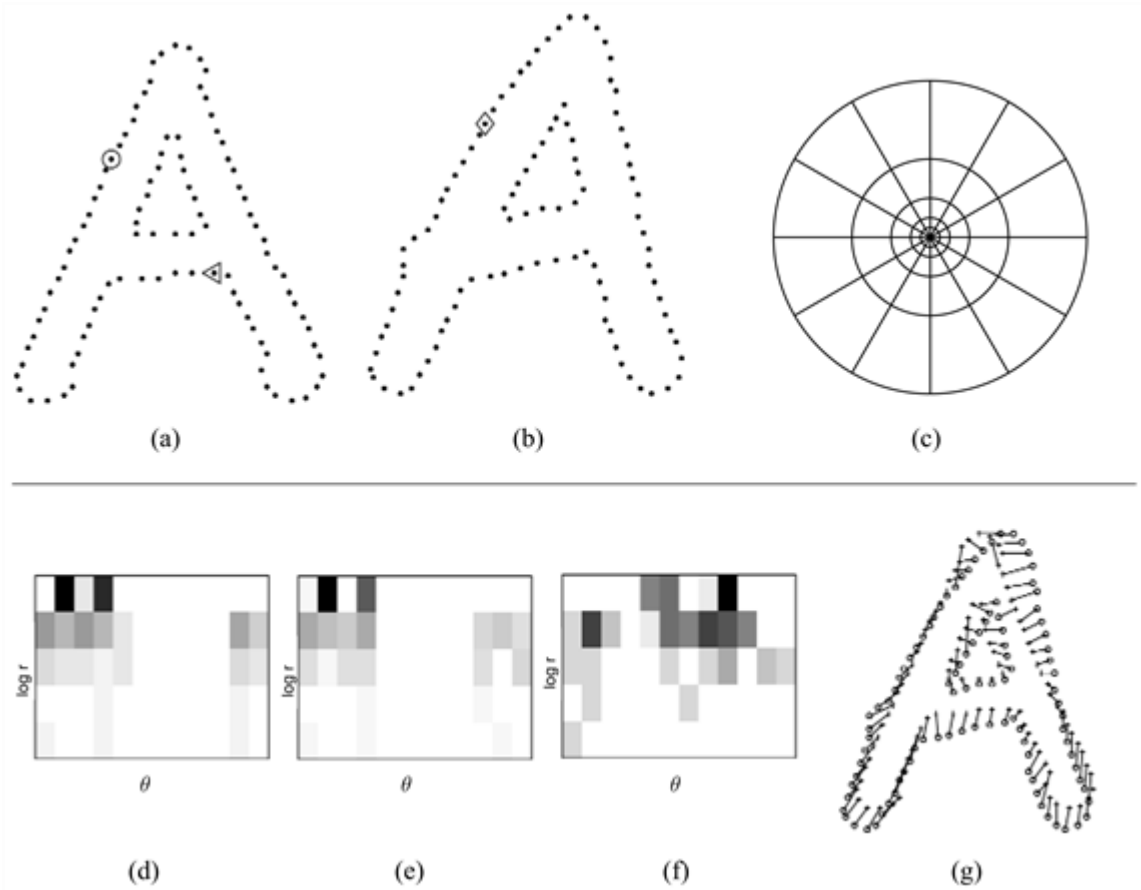
Vegyük észre, hogy az alakzat kontextusának definíciója magában hordozza az elforgatásinvarianciát, hiszen minden mérést az objektum pontjaihoz képest végzünk. A méretinvariancia eléréséhez minden sugárirányú távolságot normalizálunk a pontpárok közötti átlagos távolsággal.

Az alakzatkontextus lehetővé teszi, hogy a megfeleltetési problémát megoldjuk két hasonló, de nem azonos objektumra, mint ahogy az a 24.20. (a) és (b) ábrán látszik. Az alakzatkontextus különböző lesz egy S alakzat különböző pontjaira, de egy S és egy S' alakzat megfelelő (homológ) pontjainak alakzatkontextusa hajlamos lesz a hasonlóságra. Ezek után a két alakzat megfeleltethető pontjai megtalálásának problémáját akként kezelhetjük, mint hasonló alakzatkontextussal rendelkező párok megtalálását.

24.20. ábra - Alakzatkontextus-számítás és -illesztés. (a, b) Két alakzat élének mintapontjai. (c) Az alakzatkontextus számítására használt log-polár hisztogram rekeszek ábrája. 5 rekeszt használunk a log r és 12 rekeszt θ számára. (d–f) Az (a) és (b) ábrán jelölt referenciaminták alakzatkontextusai: ○, ◊, ⊲. Minden alakzatkontextus a maradék ponthalmaz koordinátáinak log-polár hisztogramja a referenciapontot mint origót használva. (A sötét cellák több pontot jelentenek a rekeszekben.) Vegye észre a ○ és ◊ pontok alakzatkontextusainak vizuális hasonlóságát, amelyeket a két alakzat relatíve hasonló pontjaira számoltunk! Ezzel ellentétben a ⊲ alakzatkontextusa egészen eltérő. (g) Az (a) és (b) között kétoldali illesztéssel megtalált megfeleltetések, a hisztogramok χ^2 távolsága által definiált költséget használva.

[5] A modern számítógépes grafikában erre az ötletre az **áttűnés**sel, **morfolás**sal (**morphing**) hivatkoznak.

Created by XMLmind XSL-FO Converter.



Precízebben fogalmazva: tekintsük a p_i pontot az első, a q_j pontot a második alakzaton. Jelölje a $C_{ij} = C(p_i,q_i)$ a két pont illesztésének költségét. Minthogy az alakzatkontextusok hisztogramokkal reprezentált eloszlások, így természetes a χ^2 távolság használata:

$$C_{ij} = \frac{1}{2}\sum_{k=1}^{K}\frac{\left[h_i(k)-h_j(k)\right]^2}{h_i(k)+h_j(k)}$$

ahol $h_i(k)$ és $h_j(k)$ a normalizált hisztogram k-adik értékét jelöli a p_i, illetve a q_j pontokban. Ha adott az első objektum i és a második objektum j pontjainak összes lehetséges párosítására a C_{ij} költségek halmaza, akkor az illesztés teljes költségét akarjuk minimalizálni azzal a kényszerfeltétellel, hogy az illeszkedésnek kölcsönösen megfelelőnek kell lennie. Ez egy példa a **súlyozott páros illesztés** (**weighted bipartite matching**) problémájára, amely $O(N^3)$ időben megoldható a magyar algoritmussal.

A mintapontok megfeleltetése alapján a teljes alakzatra kiterjeszthető a megfeleltetés annak a fedési transzformációnak a megbecsülésével, amely az egyik alakzatot a másikra képezi le. A regularizált vékonylemez spline-ok (thin plate spline) különösen hatékonyak. Ha az alakzatokat fedésbe hoztuk, a hasonlósági mértékek kiszámítása viszonylag kézenfekvő. A két alakzat közötti távolság definiálható az egymásnak megfelelő pontok alakzatkontextus-távolságainak és a vékonylemez spline-hoz rendelt kötési energiának a súlyozott összegeként. Ezt a távolságmértéket meghatározva egy egyszerű legközelebbi-szomszéd osztályozó megoldja a felismerési problémát. A 20. fejezetben írtuk le ezen módszer kitűnő teljesítményét a kézzel írt számjegyek osztályozásában.

5.3. Az elhelyezkedés becslése

Amellett hogy meghatározzuk, mi is az objektum, a helyzetét is szeretnénk megállapítani, azaz a nézőhöz képesti pozícióját és az irányát. Például egy ipari alkalmazásban a robot karja nem tud felvenni egy objektumot, amíg nem tudja a helyzetét. Merev két-, illetve háromdimenziós objektumok esetében a problémának egy egyszerű és jól definiált megoldása van, amely az **illesztési módszer**en (**alignment method**) alapszik, amit a következőkben fejtünk ki.

Created by XMLmind XSL-FO Converter.

Az objektumot M jellemzővel, avagy m_1, m_2, …, m_M háromdimenziós ponttal reprezentáljuk – például egy soklapú objektum sarokpontjaival. Ezeket valamilyen, az objektumnak természetes koordináta-rendszerben határozzuk meg. Ezek után a pontokat egy ismeretlen háromdimenziós forgatásnak, **R**-nek vetjük alá, amit egy ismeretlen **t** mértékű eltolás követ, végezetül egy vetítés következik, amivel előállnak a képsík $\{p_1, p_2 \ldots, p_N\}$ jellemzői. Általában $N \neq M$, mivel bizonyos modellpontokat elnyelhettek, és a jellemződetektor kihagyhat egyes jellemzőket (vagy hamisakat határozhat meg a zaj miatt). Ezt egy háromdimenziós m_i modellpontra és a neki megfelelő p_i képpontra a következőképpen fejezhetjük ki:

$$p_i = \prod\left(\mathbf{R}m_i + \mathbf{t}\right) == Q\left(m_i\right)$$

Itt az **R** a rotációs mátrix, **t** az eltolás, és $\prod$ jelöli a perspektivikus vetítést vagy annak egy közelítését, mint például a skálázott ortografikus vetítést. A végső eredmény a Q transzformáció, ami az m_i modellpontot megfelelteti a p_i képpontnak. Bár kezdetben nem ismerjük a Q transzformációt, (merev objektumokra) tudjuk, hogy Q-nak minden modellpontra *azonosnak* kell lennie.

Meghatározhatjuk Q-t a modellpontjainak és a pontoknak megfelelő kétdimenziós projekciók adott háromdimenziós koordinátái alapján. Intuíciónk a következőt súgja: felírhatunk olyan egyenleteket, amelyek az m_i modellpontokat és a p_i képpontokat összekapcsolják. Ezekben az egyenletekben az ismeretlen mennyiségek az **R** forgatási mátrix és a **t** eltolási vektor paramétereinek felelnek meg. Ha van elég egyenletünk, ki kell tudnunk számolni Q-t. Nem fogjuk bizonyítani itt, hanem csupán állítjuk a következő eredményt:

Ha adott három, nem egy egyenesre eső m_1, m_2 és m_3 modellpont, és ezek skálázott ortogonális projekciója, p_1, p_2 és p_3 a képsíkon, akkor létezik pontosan két transzformáció a háromdimenziós modell koordinátakeretből a kétdimenziós képkoordináta-keretbe.

Ezek a transzformációk a képsíkra vonatkozó tükrözésen keresztül kapcsolatban állnak egymással, és egy egyszerű zártalakú megoldással meghatározhatók. Ha meg tudnánk határozni a kép három jellemzőjének megfelelő három modelljellemzőt, akkor kiszámolhatnánk Q-t, az objektum helyzetét. Az előző alfejezetben tárgyaltunk egy alakzatkontextuson alapuló megfeleltetést. Ha az objektumnak vannak jól meghatározott sarkai vagy más jellegzetes pontjai, akkor egy még egyszerűbb technika alkalmazása válik lehetségessé. Az ötlet lényege a generálás és tesztelés. Egy kezdeti megfeleltetést kell találnunk egy képhármas és egy modellhármas között, majd a `TRANSZFORMÁCIÓTKERES` eljárást használjuk Q egy hipotézisének előállítására. Ha a tippelt megfelelés helyes volt, akkor Q helyes lesz, és ha a többi modellpontra alkalmazzuk, a képpontok predikcióját eredményezi. Ha a tippelt megfelelés helytelen volt, akkor Q helytelen lesz, és ha a maradék modellpontokra alkalmazzuk, nem állítja elő a képpontokat.

24.21. ábra - Az illesztésalgoritmus informális leírása

```
function ILLESZTÉS(kép, modell) returns egy megoldás vagy kudarc
  inputs: kép, képjellemző pontok listája
          modell, modelljellemző pontok listája

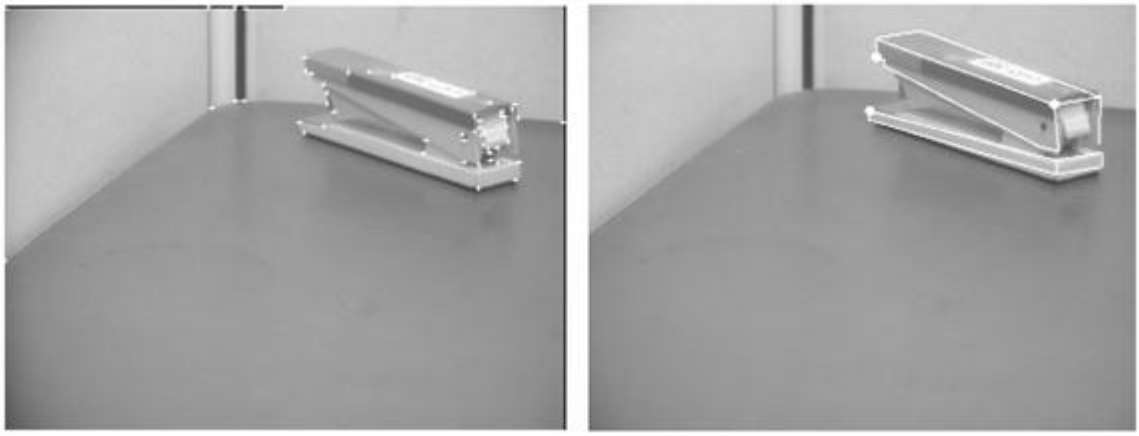
  for each p1, p2, p3 in TRIPLETS(kép) do
      for each m1, m2, m3 in TRIPLETS(modell) do
        Q ← TRANSZFORMÁCIÓT-KERES(p1, p2, p3, m1, m2, m3)
        if a Q szerinti projekció megmagyarázza a képen then
          return Q
  return kudarc
```

Ez a 24.21. ábrán látható `ILLESZTÉS` algoritmus alapja. Az algoritmus megtalálja egy adott modell elhelyezkedését, vagy hibával tér vissza. Az algoritmus időbeli komplexitása a legrosszabb esetben arányos a modell- és képponthármasok kombinációinak számával, vagyis $\binom{N}{3}\binom{M}{3}$-mal, megszorozva minden egyes kombináció ellenőrzésének költségével. Az ellenőrzés költsége $M \log N$, mivel az M modellpont mindegyikére elő kell állítanunk a képbeli pozíciót, és meg kell találnunk a legközelebbi képponttól mért távolságot, ami egy $\log N$ művelet, ha a kép pontjai egy megfelelő adatstruktúrában helyezkednek el. Így legrosszabb esetben az algoritmus időbeli komplexitása $O(M^4N^3 \log N)$, ahol M és N a modell és a kép pontjainak száma. Az

Created by XMLmind XSL-FO Converter.

elhelyezkedésklaszterezésen alapuló technikák randomizálással lecsökkentik a komplexitást $O(MN^3)$-re. Ezen algoritmus alkalmazásának eredményei a tűzőgépre a 24.22. ábrán láthatók.

24.22. ábra - (a) A tűzőgép fényképén talált sarkok. (b) Az eredeti képre ráhelyezett hipotetikus rekonstrukció (Clark Olson hozzájárulásával).

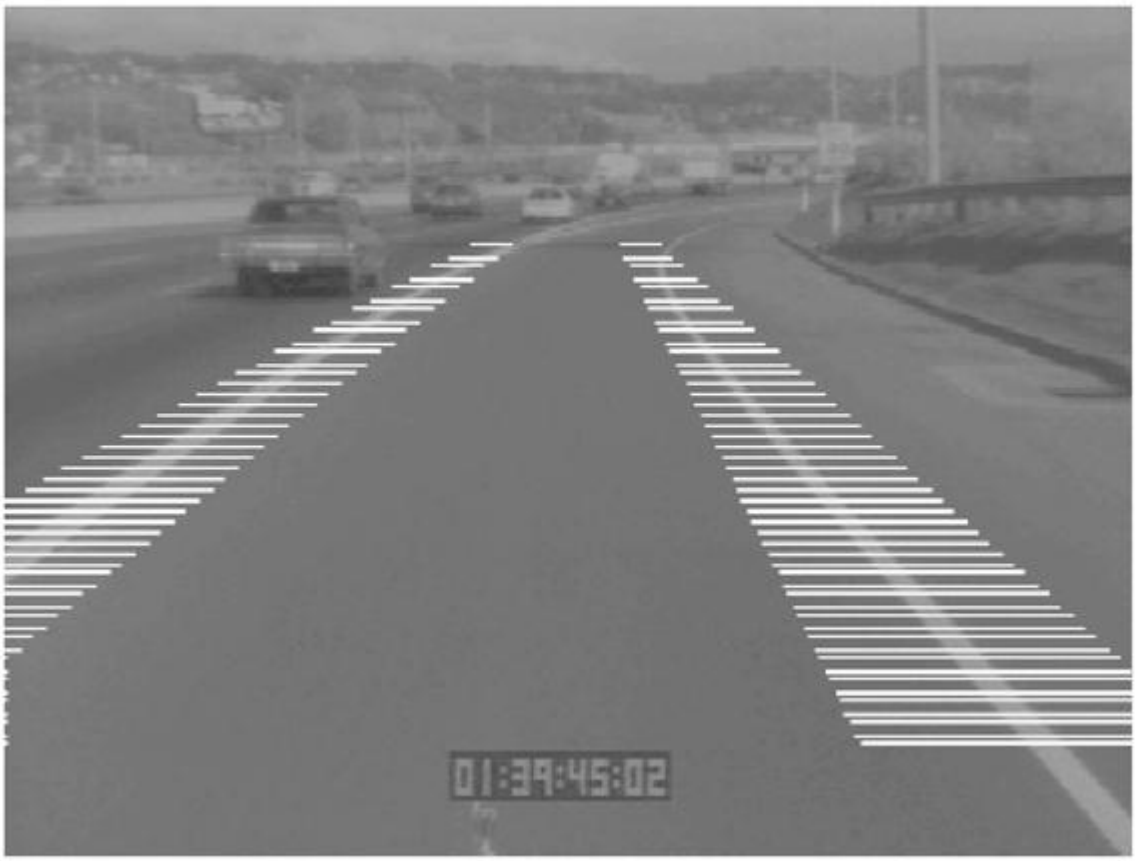


6. Navigálás és manipulálás a látás segítségével

A látás egyik legfontosabb alkalmazása az, hogy információt szolgáltat az objektumok manipulálásához – hogy felszedhessük, megfoghassuk, elforgathassuk stb. azokat –, valamint hogy akadályok kikerülésével navigálhassunk. Az a képesség, hogy a látást ezen célokra használják, a legprimitívebb állati látórendszerekben is jelen van. Sok esetben a látórendszer minimális abban az értelemben, hogy a rendelkezésre álló látómezőből csak annyi információt nyer ki, ami az állat viselkedésének a tájékoztatására szükséges. Igen valószínű, hogy a modern látórendszerek a korai, primitív organizmusokból fejlődtek ki, amelyek egy fényérzékeny foltot használtak az egyik végükön, és így tudták magukat a fény felé vagy azzal ellentétes irányba orientálni. A 24.4. alfejezetben láttuk, hogy a legyek egy nagyon egyszerű optikai folyamra alapozó látást használnak, hogy le tudjanak szállni a falon. A *What the Frog's Eye Tells the Frog's Brain* c. klasszikus tanulmány (Lettvin és társai, 1959) megfigyeli, hogy egy béka „éhen fog halni, ha olyan étellel van körülvéve, amely nem mozog. Csak a méret és a mozgás alapján állapítja meg, hogy mi élelem."

24.23. ábra - Az út képe egy autóban lévő kamerával lefényképezve. A vízszintes fehér csíkok jelzik azokat a keresési ablakokat, amiken belül a vezérlő az útvonali jeleket keresi. A gyenge képminőség nem szokatlan az alacsony felbontású fekete-fehér videók esetében.

Created by XMLmind XSL-FO Converter.



A robotoknak nevezett „organizmusokban” számítógépes látórendszereket használunk. Tekintsünk egy speciális robotfajtát: egy automata járművezetőt egy autópályán (lásd 24.23. ábra). Először elemezzük a feladatot; azután meghatározzuk a látási algoritmust, amely a feladatok helyes végrehajtásához szükséges. A vezető előtt álló feladatok között találjuk az alábbiakat:

1. Oldalsó irányítás – biztosítani, hogy a kocsi biztonságosan a közlekedési sávon belül maradjon, vagy szükség esetén zökkenőmentesen váltson sávot.

2. Hosszanti irányítás – biztosítani, hogy a kocsi előtt biztonságos távolság legyen.

3. Akadályelhárítás – a szomszédos közlekedési sávokban haladó kocsik figyelése és felkészülés az elkerülő manőverezésre, ha azok valamelyike sávot szeretne váltani.

A vezető feladata, hogy generálja ezen feladatok legjobb teljesítéséhez alkalmas kormányzási, gyorsítási vagy fékezési cselekvéseket.

Az oldalsó irányításhoz szükséges a gépkocsi pozíciójának és irányítottságának a sávhoz képesti reprezentálása. A 24.23. ábrán látható kép esetén éldetektáló algoritmusokkal azonosítjuk a sávhatárokat jelző határoló vonalakhoz tartozó éleket. Ezt követően ezekre az élekre sima görbéket illeszthetünk. Ezen görbék paraméterei információt hordoznak a gépkocsi oldalsó elhelyezkedésére nézve, az autó haladási irányára a sávon belül és a sáv íveltségére. Ez az információ – a gépkocsi dinamikájával együtt – képezi mindazt, amire a kormányzást irányító rendszernek szüksége van. Jegyezzük meg azt is, hogy mivel képkockáról képkockára a sáv képi vetületének pozíciója csupán csekély mértékben változik, tudjuk, hogy a képen a sávhatárjelző vonalakat *hol* keressük – csak azokon a területeken kell keresnünk, amelyek párhuzamos fehér vonalakkal vannak megjelölve.

A hosszanti irányításhoz az előttünk haladó gépkocsik távolságára van szükség. Erre a kétkamerás sztereolátás vagy optikai folyam segítségével tehetünk szert. Mindkét megközelítést lényegesen egyszerűsíthetjük, ha kihasználjuk a sík felületen történő haladásból adódó kényszereket. Ilyen technikák felhasználásával a vizuálisan irányított autók nagy sebességgel hosszú ideig tudnak közlekedni.

Fontos

Created by XMLmind XSL-FO Converter.

A gépkocsi-vezetési példa egy szempontot igencsak világossá tesz: *egy konkrét feladat esetén a képen elvben található összes információ kinyerése nem szükséges*. Nem szükséges a gépkocsik pontos alakjának a meghatározása, az úttal szomszédos füves területekre vonatkozó az-alak-a-textúrából probléma megoldása stb. A feladat információszükséglete csak egy bizonyos típusú információra vonatkozik, és lényeges számítási sebességet és robusztusságot lehet biztosítani, ha csak erre az információra összpontosítunk, és a problémából adódó kényszereket teljes egészében kihasználjuk. Az előbbi részekben bemutatott általános megközelítések célja az volt, hogy egy általános alapelméletet adhassunk, amit majd a konkrét feladat szükségleteihez lehet specializálni.

7. Összefoglalás

Bár az ember számára az érzékelés könnyed cselekvésnek tűnik, igen nagy mennyiségű komplex számítást igényel. A látás célja információ kiemelése olyan feladatokhoz, mint a manipulálás, a navigálás és az objektumfelismerés.

- A **képalkotás** (**image formation**) folyamata, a geometriai és fizikai aspektusokat tekintve, jól feltárt terület. Ha adott egy háromdimenziós jelenet leírása, ennek valamilyen tetszőleges kamerapozíciónak megfelelő képét könnyűszerrel előállíthatjuk (grafikai probléma). E folyamat megfordítása és a jelenet leírásának a kép alapján történő előállítása nehéz.

- Ahhoz, hogy a manipulálási, navigálási és objektumfelismerési feladatokhoz szükséges vizuális információt kiemelhessük, szükség van valamilyen közbülső reprezentáció megkonstruálására. A képekből az olyan primitív elemeket, mint az élek és a régiók, a **képfeldolgozási algoritmusok** (**image-processings**) emelik ki.

- Egy képben többféle segítség rejlik, amely lehetővé teszi, hogy a jelenetről háromdimenziós információt kapjunk. Idetartozik a mozgás, a sztereolátás, a textúra, az árnyalás és a kontúrelemzés. Ezek mindegyike a fizikai jelenetre vonatkozó háttér- feltételezéseken alapul, hogy az interpretációt közel egyértelművé tegye.

- A teljes körű objektumfelismerés nagyon nehéz probléma. A fényesség- és a jellemzőalapú megközelítéseket tárgyaltuk. Más lehetőségek is vannak.

7.1. Irodalmi és történeti megjegyzések

Az emberi látás megértésére irányuló szisztematikus próbálkozások az ókorig nyúlnak vissza. Eukleidész (kb. i. e. 300) írt a természetes perspektíváról – az olyan leképezésről, amely a háromdimenziós világ minden *P* pontjához az *OP* sugár irányát rendeli hozzá, amely sugár a *P* pontot a vetítés *O* középpontjával köti össze. Eukleidész ismerte a mozgási parallaxis jelenségét is. A perspektivikus vetítés matematikai megértését, egy síkra való vetítés szempontjából, a 15. században az itáliai reneszánsz segítette elő. Az első, a háromdimenziós jelenet helyes geometriai vetítésén alapuló festményt általában Brunelleschinek tulajdonítják (1413). 1435-ben Alberti fogalmazta meg a szabályokat, és ő inspirálta a művészek generációit, akik művészi teljesítményét máig csodáljuk. A perspektíva tudományának – ahogy akkor nevezték – kidolgozásában kimagaslott Leonardo da Vinci és Albrecht Dürer. Leonardo késő 15. századi leírásait a fény és az árnyék *(chiaroscuro)* játékáról, az árnyék *umbra* és *penumbra* régióiról, valamint a légi perspektíváról még mindig érdemes elolvasni (Kemp, 1989).

Bár a perspektívát a görögök is ismerték, furcsamód a felfogásuk a szemnek a látásban betöltött szerepéről zavaros volt. Arisztotelész azt gondolta, hogy a szem – a mai lézeres távolságmérők mintájára – fénysugarakat bocsát ki. Ezt a téves elképzelést a világnak az arab tudósok közvetítették, többek közt Alhazen a 10. században. Ezután különféle kamerák fejlesztése következett. Ezek egy szobából (latinul a *camera* szobát jelent) álltak, ahova a fényt a falon egy kis nyíláson át engedjük be, hogy az átellenes falon a kinti jelenet képét kivetítse. Ezen kamerák mindegyikében a kép természetesen fordított volt, ami vég nélküli zavart okozott. Ha a szem ilyen elvű berendezés lenne, akkor hogyan lehetséges, hogy a képet helyesen látjuk? A probléma a korszak legnagyobb elméit foglalkoztatta (Leonardót is beleértve). A kérdés megválaszolásához Kepler és Descartes munkássága kellett. Descartes az ablakzsalu nyílásába egy olyan szemet helyezett be, amelynek nem átlátszó felhámrétegét eltávolították. A retinára kiterített papíron így egy fordított képet kapott. Bár a retinás kép valóban fordított, ez nem okoz problémát, mert az agy helyesen interpretálja a képet. Mai szaknyelven azt mondhatnánk, hogy megfelelő módon kezelni kell az adatstruktúrákat.

A látás megértésében a következő lényeges előrelépés a 19. században történt. Helmholtz és Wundt 1. fejezetben említett munkássága a pszichofizikai kísérleteket szigorú tudományos szintre emelte. Young,

Created by XMLmind XSL-FO Converter.

Maxwell és Helmholtz munkája megalapította a színes látás három színen alapuló elméletét. Azt a tényt, hogy az emberi szem a mélység érzékelésére is képes, ha a két szem kissé eltérő képet lát, Wheatstone a sztereoszkóp feltalálásával demonstrálta (Wheatstone, 1838). A készüléknek azonnal nagy sikere lett a szalonokban Európa-szerte. A kétkamerás sztereolátás lényegét, azaz hogy a jelenet két, kissé eltérő látószögből készített képe elegendő információt hordoz a jelenet háromdimenziós visszaállításához, a fotogrammetria területén aknázták ki. Megszülettek a kulcsfontosságú matematikai eredmények. Kruppa bebizonyította, hogy ha öt különböző pont két különböző képével rendelkezünk, vissza lehet állítani a két kamera pozíciója közötti elfordulást és eltolást, valamint a jelenet mélységét (egy skálafaktor erejéig) (Kruppa, 1913). Bár a sztereolátás geometriáját jó ideje ismerték, a fotogrammetria megfeleltetési problémáját általában emberek oldották meg, akik az egymásnak megfelelő pontokat igyekezték egymáshoz illeszteni. Az embereknek a megfeleltetési probléma megoldásában megmutatkozó csodálatos képességét Julesz Béla demonstrálta a véletlen-pont sztereogram feltalálásával (Julesz, 1971). Ezen probléma megoldására mind a számítógépes látás területén, mind a fotogrammetriában sok munkát fordítottak az 1970-es és az 1980-as években.

A 19. század második fele volt az a periódus, amikor az emberi látásra vonatkozó pszichofizikai kutatásokat megalapozták. A 20. század első felében a látás területén a legfontosabb eredményekhez a Max Wertheimer vezette Gestalt pszichológiai iskola jutott. „Az egész több, mint a részek összege” jelszóval azt a nézetet hangsúlyozták, hogy az észlelés elsődleges elemei a teljes formák legyenek, ne pedig az összetevők, mint például az élek.

A második világháború utáni szakaszt megújult aktivitás jellemezte. A legfontosabb J. J. Gibson munkája volt, aki rámutatott az optikai folyam és a textúragradiensek fontosságára olyan környezeti változók becslésénél, mint a dőlés és a lejtés (Gibson, 1950; 1979). Ő újból hangsúlyozta az ingerek fontosságát és gazdagságát. Gibson, Olum és Rosenblatt rámutattak, hogy az optikai folyammező elegendő információt tartalmaz ahhoz, hogy a megfigyelő meghatározhassa saját mozgását a környezetben (Gibson és társai, 1955). A számítógépes látásban a kutatás ezen területen és a (matematikailag ekvivalens) struktúra-a-mozgásból területen főleg az 1980-as években ment végbe, Koenderink, Van Doorn, Ullman és Longuet-Higgins alapozó eredményeit követve (Koendrink és Van Doorn, 1975; Ullman, 1979; Longeut-Higgins, 1981) adott lendületet e tevékenységnek. A kezdeti aggodalmakat a mozgásból származtatott struktúra stabilitásával kapcsolatban Tomasi és Kanade munkája (Tomasi és Kanade, 1992) csillapította, akik megmutatták, hogy több képkocka felhasználásával és az ebből származó széles alapvonallal az alak egészen pontosan felismerhető.

Chan és társai írják le a légy meghökkentő vizuális apparátusát, amelynek tízszer nagyobb az időbeli vizuális aktivitása, mint az embereké (Chan és társai, 1998). Azaz egy légy egy maximum 300 képkocka/másodperc sebességű filmet is úgy tudna nézni, hogy felismeri az egyes képkockákat.

Az 1990-es években megjelent koncepcionális újítás a mozgásból származtatott struktúra tanulmányozása volt. Egy ilyen beállításban a kamera kalibrációja nem szükséges, ahogy azt Faugeras megmutatta (Faugeras, 1992). Ez a felfedezés összefügg az objektumfelismerésben alkalmazott geometriai invariánsokkal, ahogy azt Mundy és Zisserman áttekintették (Mundy és Zisserman, 1992), és a mozgásból származtatott rokon struktúrával (Koenderink és Van Doorn, 1991). Az 1990-es években a számítógépek sebességének és tárolási kapacitásának növekedésével a digitális videofelvételekből történő mozgási szekvenciaelemzést sok új területen alkalmazták. Valósvilág-beli jelenetek geometriai modelljeinek felépítése a számítógépes grafikai technikákkal történő előállítás céljából különösen népszerűnek bizonyult, amelyet olyan rekonstrukciós algoritmusok vezettek, mint amelyet például Debevec, Taylor és Malik fejlesztettek ki (Debevec, Taylor és Malik, 1996). Hartley és Zisserman, valamint Faugeras és társainak könyvei átfogóan tárgyalják a többszörös nézetek geometriáját (Hartley és Zisserman, 2000; Faugeras és társai, 2001).

A számítógépes látásban az alak textúrából való kinyerésére vonatkozó legfontosabb kezdeti eredményeket Bajcsynak, Liebermannak és Stevensnek köszönhetjük (Bajcsy és Lieberman, 1976; Stevens, 1981). Míg ez a munka sík felületekre vonatkozott, a görbült felületekre vonatkozóan átfogó elemzéseket Garding, Malik és Rosenholtz végeztek (Garding, 1992; Malik és Rosenholtz, 1994).

A számítógépes látásban az alaknak az árnyékolásból való kinyerését Berthold Horn tanulmányozta (Horn, 1970). E terület fő cikkeit a (Horn és Brooks, 1989) tekinti át. Ez a megközelítés számos egyszerűsítő feltétellel élt, amelyekből a legkényesebb a kölcsönös megvilágítás hatásának figyelmen kívüli hagyása volt. A kölcsönös megvilágítás fontosságát kellően értékelték a számítógépes grafika területén, ahol éppen ezen hatás figyelembevételére fejlesztették ki a fénysugárkövetést és a radiozitást. Egy elméleti és gyakorlati kritika a (Forsyth és Zisserman, 1991)-ben található.

Created by XMLmind XSL-FO Converter.

Az alaknak a kontúrból való meghatározása területén Huffman és Clowes kezdeti, kulcsfontosságú hozzájárulásai után poliéderes objektumok esetére (Huffman, 1971; Clowes, 1971) Mackworth és Sugihara teljessé tették az elemzést (Mackworth, 1973; Sugihara, 1984). Malik kifejlesztette a tartományonként sima görbült objektumok címkézési sémáját (Malik, 1987). Kirousis és Papadimitriou megmutatták, hogy a triéderes jelenet vonalcímkézése NP-teljes (Kirousis és Papadimitriou, 1988).

Ahhoz, hogy a sima görbült objektumok vetületeiből a vizuális eseményeket megértsük, szükség van a differenciálgeometria és a szingularitáselmélet együttesére. Ezek tanulmányozásához a legjobb irodalom Koenderink *Solid Shape* c. műve (Koenderink, 1990).

A háromdimenziós objektumfelismerés területén Roberts az MIT-n benyújtott diszszertációja adott lényegi eredményeket (Roberts, 1963). Sokan úgy tartják, hogy ez a számítógépes látás területén az első PhD-disszertáció, amely sok kulcsfontosságú ötletet vezetett be, az éldetektálást és a modellalapú illesztést is beleértve. A Canny-féle éldetektálást (Canny, 1986) vezette be. A Roberts által bevezetett illesztés gondolata később Lowe, Huttenlocher és Ullman munkájában újra megjelent (Lowe, 1987; Huttenlocher és Ullman, 1990). A helyzet összerendelési módszer alapján történő becslésének hatékonyságát jelentős mértékben Olson növelte (Olson, 1994). A háromdimenziós objektumfelismerés kutatásának egy másik nagy vonulata, az alakzatok térfogati primitívekre, ún. **általánosított henger**ekre (**generalized cylinder**s) alapuló leírása, amelyet Tom Binford vezetett be, különösen népszerűnek bizonyult (Binford, 1971).

Míg az objektumfelismeréssel foglalkozó számítógépes látás kutatása nagyrészt a háromdimenziós objektumok kétdimenziós képekre történő leképezésével foglalkozott, addig a mintafelismeréssel foglalkozó közösségben létezett egy párhuzamos irányzat, amely a problémát a mintaosztályozás problémájaként látta. Az ezt motiváló példák olyan területekről kerültek ki, mint az optikai karakterfelismerés és a kézzel írt irányítószámok felismerése, ahol az elsődleges probléma egy objektumosztály tipikus variációinak jellemvonásait megtanulni, és ezek alapján különválasztani őket más osztályoktól. A megközelítések összehasonlítását a (LeCun és társai, 1995) tartalmazza. Az objektumfelismerésről szóló további munkák között van (Sirovitch és Kirby, 1987; Viola és Jones, 2002) arcfelismerésről szóló munkája. Az alakkontextus módszerét (Belongie és társai, 2002) írja le. (Dickmanns és Zapp, 1987) demonstrált először nagy sebességgel az autópályán haladó, vizuálisan vezérelt járművel; Pomerleau neurális hálózatok segítségével ért el hasonló teljesítményt (Pomerleau, 1993).

Stephen Palmer munkája, a *Vision Science: Photons to Phenomenology* (Palmer, 1999) a legjobb összefoglaló munka az emberi látás leírására; David Hubel *Eye, Brain and Vision* (Hubel, 1988) és Irvin Rock *Perception* c. műve (Rock, 1984) rövid bevezetőt nyújtanak a neuropszichológia, illetve az észlelés területeibe.

A számítógépes látás területének legjobb összefoglaló jegyzete David Forsyth és Jean Ponce munkája, a *Computer Vision: A Modern Approach*. Sokkal rövidebben tárgyalja a területet (Nalwa, 1993; Trucco és Verri, 1998). A *Robot Vision* (Horn, 1986) és a *Three-Dimensional Computer Vision* (Faugeras, 1993) régebbi, de még mindig hasznos jegyzetek, saját, speciális témájukban. David Marr könyve, a *Vision* (Marr, 1982) fontos szerepet töltött be a számítógépes látás (pszichofizika) és a tradicionális biológiai látás (neurobiológia) összekapcsolásában. A számítógépes látás témájában a két fő folyóirat az *IEEE Transactions on Pattern Analysis and Machine Intelligence* és az *International Journal of Computer Vision*. A számítógépes látással foglalkozó konferenciák többek közt az ICCV *(International Conference on Computer Vision),* a CVPR *(Computer Vision and Pattern Recognition)* és az ECCV *(European Conference on Computer Vision).*

7.2. Feladatok

24.1.

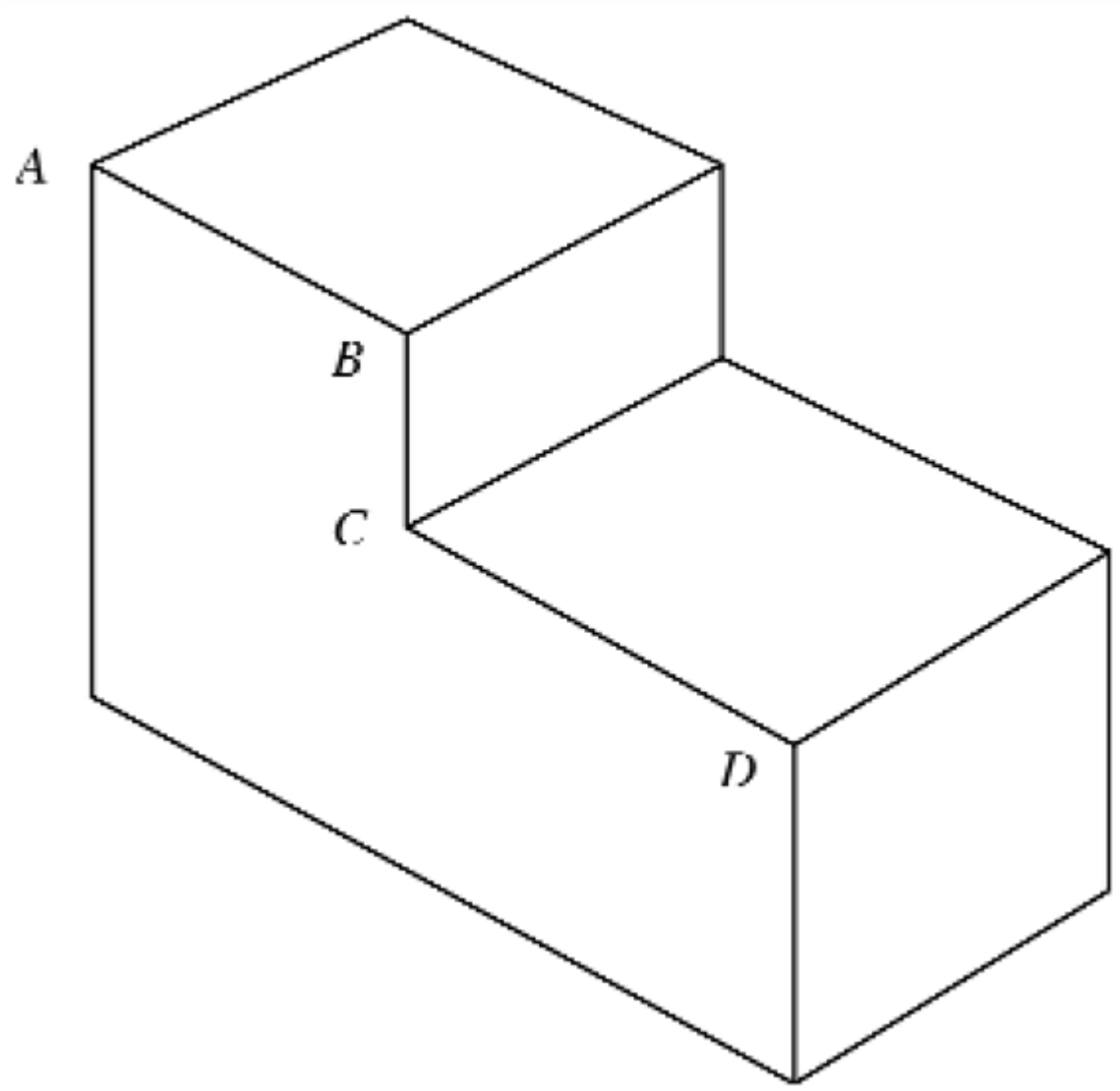
A sűrű, lombos koronával rendelkező fa árnyékában számos világos foltot lehet látni. Meglepetésre mind körkörös. Miért? Hiszen ha úgy vesszük, a levelek közötti rések, amelyeken keresztül átvilágít a nap, nagy eséllyel nem ilyenek.

24.2.

Címkézze meg a 24.24. ábrán látható objektumot, azt feltételezve, hogy a külső éleket takaró éleknek címkéztük, és az összes sarokpont triéderes. Tegye meg ezt a visszalépő algoritmussal, amely a sarokpontokat *A, B, C* és *D* sorrendben fogja megvizsgálni, mindegyik lépésnél úgy megválasztva a címkéket, hogy azok az előbb megcímkézett csatlakozásokkal és élekkel konzisztensek legyenek. Próbálja most a sarokpontokat *B, D, A* és *C* sorrendben megcímkézni.

Created by XMLmind XSL-FO Converter.

24.24. ábra - A megcímkézendő objektum, amelynek minden sarokpontja triéderes



24.3.

Tekintsen egy végtelen hosszú, *r* sugarú hengert, amelynek tengelye az *y* tengellyel párhuzamos. A henger Lambert-féle felülettel rendelkezik, és a pozitív *z* tengely felől irányítjuk rá a kamerát. Mit fog látni a képen, ha a hengert egy pontszerű fényforrás világítja meg, amely a pozitív *x* tengelyen a végtelenben helyezkedik el. Magyarázza meg a válaszát úgy, hogy az azonos fényességű kontúrokat rajzolja fel a vetített képre. Az azonos fényességű kontúrok azonos távolságban helyezkednek-e el?

24.4.

A képbeli élek a jelenetesemények sokaságának felelhetnek meg. Tekintsen egy tetszőleges fényképet, amely valódi háromdimenziós jelenetet ábrázol. Azonosítson a képen tíz élt, és kísérelje meg egyenként eldönteni, hogy az él (a) a mélység, (b) a felületi normális, (c) a reflektancia, illetve (d) a megvilágítás szakadásának felel-e meg.

24.5.

Mutassa meg, hogy a konvolúció egy adott függvénnyel felcserélhető a differenciálással, azaz, hogy:

$(f * g') = f * g'$

24.6.

Created by XMLmind XSL-FO Converter.

Egy terület feltérképezéséhez egy sztereorendszer használatát fontolgatjuk. A rendszer két CCD kamerából fog állni, mindegyik 512 × 512 pixel felbontású egy 10 × 10 cm négyzet alakú érzékelőn. A felhasználandó lencsék fókusztávolsága 16 cm, és a lencsék a végtelenben fixálnak. A bal oldali kép (u_1, v_1) és a jobb oldali kép (u_2, v_2) egymáshoz tartozó pontjaira $v_1 = v_2$, hiszen a két képsíkon az x tengelyek az epipoláris vonalakkal párhuzamosak. A két kamera optikai tengelye párhuzamos. A két kamera közötti bázisvonal 1 m.

a. Ha a legközelebbi mérendő táv 16 m, mi a tapasztalható legnagyobb diszparitás (képpontban)?

b. Mi a távmérés pixelfelbontásból adódó felbontása 16 m távolságban?

c. Milyen táv tartozik az egy képpontos képeltéréshez?

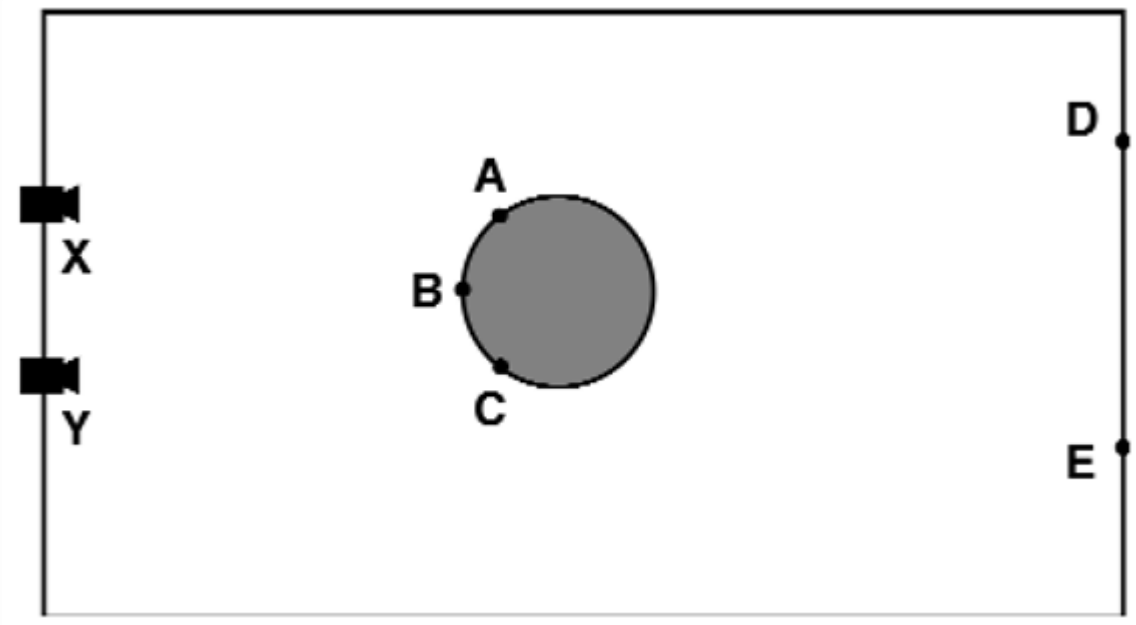
24.7.

Az illesztési algoritmust egy ipari alkalmazásnál szeretnénk használni, ahol lapos alkatrészeket egy szállítószalag visz, a szalag felett pedig egy függőlegesen elhelyezett kamera figyel. Az alkatrész elhelyezkedését három változó határozza meg, egy az elfordulást és kettő az alkatrész kétdimenziós pozícióját. Ez egyszerűsíti a problémát, és a `TRANSZFORMÁCIÓT-KERES` függvény az egymásnak megfelelő kép- és modelltulajdonságokból két párt igényel ahhoz, hogy az elhelyezkedést azonosítsa. Határozza meg csak ebben a környezetben az illesztésnek a legrosszabb esetre vett komplexitását.

24.8.

(Pietro Perona nyomán.) A 24.25. ábra X és Y pontban két kamerát mutat, amelyek megfigyelnek egy jelenetet. Rajzolja le azt a képet, amit az egyes kamerák látnak (felteheti, hogy minden jelölt pont ugyanabban a vízszintes síkban helyezkedik el). Mi tudható meg a két kép alapján az A, B, C, D és E pontok a kamera alapvonalától mért relatív távolságáról, és milyen alapon?

24.25. ábra - Felülnézeti kép egy kétkamerás látórendszerről, amely egy üveget figyel meg, amely mögött egy fal van



24.9.

A következő állítások közül melyik igaz és melyik hamis?

a. Sztereoképeken az egymásnak megfelelő pontok megtalálása a sztereomélység-megállapítás folyamatának legegyszerűbb feladata.

b. Az alak-mintázatból feladat megoldható egy fénycsíkokat tartalmazó háló jelenetre történő vetítésével.

Created by XMLmind XSL-FO Converter.

c. A Huffman–Clowes címkéző rendszer mindenféle poliéderes objektumot tud kezelni.

d. Görbéket tartalmazó objektumok vonalas rajzaiban a vonal címkéje változhat az egyik végétől a másikig.

e. Ugyanazon jelenet sztereonézeteiben minél messzebb van a két kamera egymáshoz képest, annál pontosabban határozható meg a mélység.

f. Egy jelenetben található egyforma hosszúságú vonalak mindig egyforma hosszúságra vetítődnek a képen.

g. A képen egyenes vonalak szükségszerűen egyenes vonalaknak felelnek meg a jelenetben.

24.10.

A 24.23. ábra egy autó nézőpontjából készült egy autópálya kijáratánál. Két autó látható a közvetlenül balra levő sávban. Milyen okok miatt kell a nézőnek azt a következtetést levonnia, hogy az egyik közelebb van, mint a másik?

Created by XMLmind XSL-FO Converter.

25. fejezet - Robotika

Ebben a fejezetben az ágenseket fizikai beavatkozó szervekkel látják el, hogy rosszalkodhassanak.

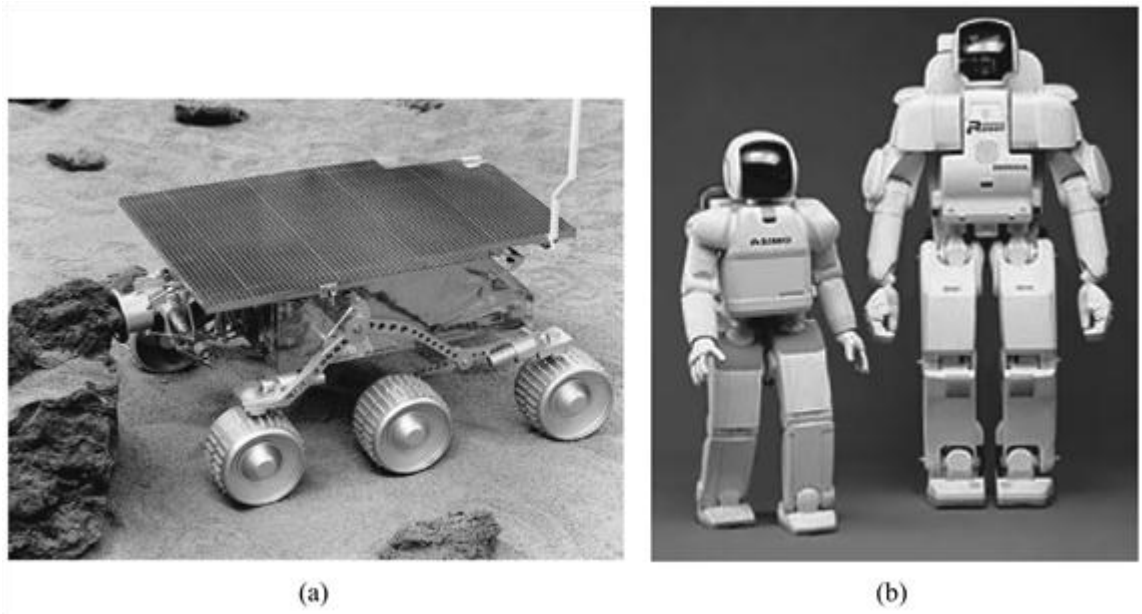
1. Bevezetés

A **robot**ok (**robot**s) olyan fizikai ágensek, amelyek a fizikai világ megváltoztatásával oldanak meg feladatokat. E célból különböző **beavatkozó szerv**ekkel (**effector**s) szerelik fel őket, például lábakkal, kerekekkel, karokkal és megfogókkal. A beavatkozók kizárólagos célja, hogy fizikai hatást fejtsenek ki a környezetre.[1] A robotokat **érzékelő**kkel (**sensor**s) is felszerelik, hogy érzékelhessék környezetüket. Manapság a robotikában számtalan különféle érzékelőt használnak: kamerákat és ultrahangradarokat a környezet mérésére, giroszkópokat és gyorsulásmérőket a robot saját mozgásának követésére.

A legtöbb mai robot három nagy kategória egyikébe sorolható. A **manipulátor**ok (**manipulator**s), vagy más néven robotkarok fizikailag a munka helyszínéhez rögzítettek, például egy ipari szerelősoron egy gyárban vagy a Nemzetközi Űrállomáson. A manipulátorok mozgását általában irányítható csuklók sora biztosítja, lehetővé téve, hogy a végbeavatkozó szerv a munkatér bármely pontjára eljuthasson. Az ipari robotok messze leggyakoribb fajtája a manipulátor, világszerte több mint egymillió működik belőlük. Bizonyos mobil robotkarokat kórházakban, műtéteknél használnak. Kevés autógyártó tudna ma már meglenni ipari robotok nélkül, és egyes robotkarok még műalkotások készítésére is képesek.

A második csoportot a **mobil robot**ok (**mobile robot**s) alkotják. A mobil robotok kerekek, lábak vagy hasonló szerkezetek segítségével mozognak a fizikai környezetben. Használják őket kórházakban ételkihordásra, dokkokban árurakodásra és más, hasonló feladatokra. Korábban már említettünk egy példát, a NAVLAB **ember nélküli közúti jármű**vét (**unmanned land vehicle**, **ULV**), amely autópályán képes önállóan, sofőr nélkül navigálni. Másfajta mobil robotokat, például az **ember nélküli légi jármű**veket (**unmanned air vehicle**, **UAV**) katonai felderítésre, mezőgazdasági permetezésre és megfigyelésre használnak. Az **autonóm víz alatti jármű**vek (**autonomous underwater vehicle**, **AUV**) nagy szerepet játszanak a mélytengeri felfedezésekben, míg a **bolygójáró**k (**planetary rover**s), mint például a 25.1. (a) ábrán látható Sojourner, az űrkutatásban segédkeznek.

25.1. ábra - (a) A NASA Sojourner mobil robotja, amely a Mars felszínét derítette fel 1997 júliusában. (b) A Honda P3 és Asimo elnevezésű humanoid robotjai.



[1] A 2. fejezetben még **beavatkozó**król (**actuator**s) beszéltünk, és nem beavatkozó szervekrõl (effectors). A beavatkozó egy vezérlővonal, amely a beavatkozó szervnek közvetíti az utasítást, míg a beavatkozó szerv maga a fizikai eszköz.

Created by XMLmind XSL-FO Converter.

A harmadik típusba tartoznak a hibridek: olyan mobil robotok, amelyekre karokat is szereltek. Ezek közé sorolhatjuk a **humanoid robot**okat (**humanoid robot**s), amelyek fizikai felépítése hasonlít az emberéhez. A 25.1. (b) ábrán két ilyen humanoid robot látható, mindkettő Japánban készült, a Hondánál. A hibridek a rögzített manipulátoroknál nagyobb távolságokban is képesek beavatkozó szerveiket használni, de általában nehezebben tudják végrehajtani a feladatukat, mert nem rendelkeznek kellő stabilitással és szilárdsággal, amit a lerögzítés biztosít.

A robotika tárgyköre magában foglalja a protéziseket (emberek részére készített mesterséges végtagok, mesterséges fül vagy szem), az intelligens környezeteket (mint például egy egész ház felszerelve szenzorokkal és beavatkozó szervekkel) és a többelemű (multibody) rendszereket is, ahol a feladatokat rengeteg apró együttműködő robot hajtja végre.

A valódi robotoknak általában olyan környezetben kell boldogulniuk, amely csak *részlegesen megfigyelhető,* sztochasztikus, dinamikus és folytonos. Néhány, de nem mindegyik robotkörnyezet szekvenciális és multiágens jellegű is. A részleges megfigyelhetőség és a sztochasztikusság annak az eredménye, hogy egy meglehetősen nagy és komplex világgal kell foglalkozni. A robot nem lát a sarkok mögé, és bizonytalanság van a mozgásparancsok végrehajtásában a hajtások csúszása, a súrlódás stb. miatt. Mi több, a fizikai világ makacsul visszautasítja, hogy valós idejűnél gyorsabb legyen! Szimulációs környezetben egyszerűbb tanulási algoritmusok használatával (mint például a 21. fejezetben leírt **Q-tanulás**) lehetséges mindössze néhány CPU-munkaóra során több millió próbát végrehajtva tanulni, de valós környezetben éveket vehetnek igénybe ezek a kísérletek. Továbbá, az igazi ütközések valóban fájdalmat okozhatnak, ellentétben a szimuláltakkal. A valódi robotrendszerekbe – ahhoz, hogy a robot gyorsan tanulhasson és biztonságosan üzemelhessen – előzetesen információt kell beépíteni magáról a robotról, fizikai környezetéről és az elvégzendő feladatról.

2. Robothardver

Eddig a könyvben adottnak tekintettük az ágensek architektúráját (érzékelőket, beavatkozó szerveket és processzorokat), és csak az ágensek működtető programjára koncentráltunk. A valóságban a robotok sikere legalább annyira múlik a feladathoz illő, megfelelő szenzorok és beavatkozó szervek megválasztásán.

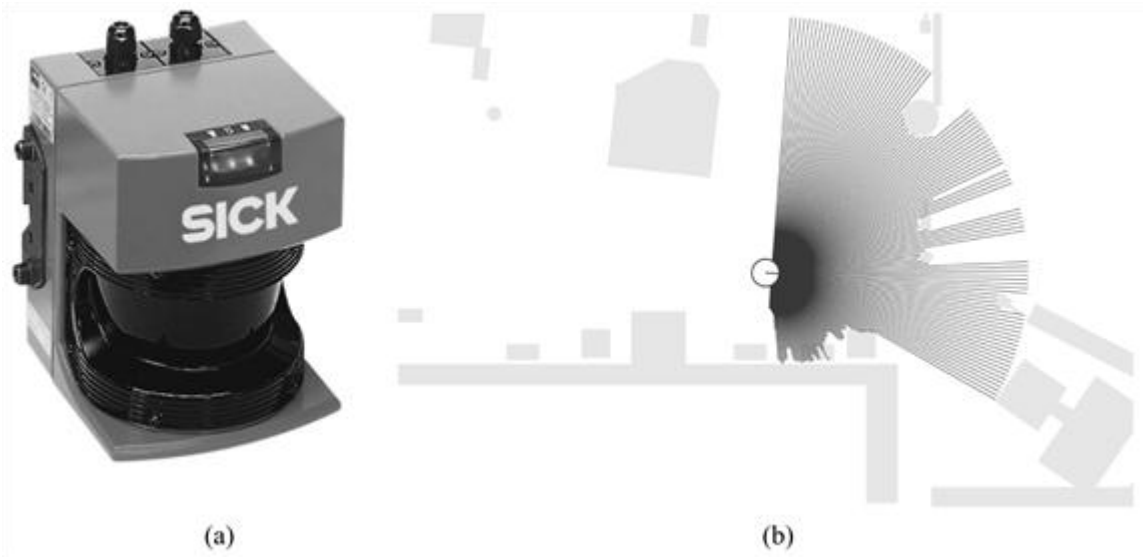
2.1. Érzékelők

A szenzorok jelentik az érzékelési interfészt a robotok és környezetük között. A **paszszív érzékelő**k (**passive sensor**s), mint például a kamerák, ténylegesen puszta megfigyelői a környezetüknek: olyan jeleket vesznek, amelyeket a robot környezetében lévő más tárgyak generálnak. Az **aktív szenzor**ok (**active sensor**s), mint például a hanglokátor, energiát sugároznak környezetükbe, és érzékelik, ha ez az energia visszaverődik. Az aktív szenzorok általában több információt szolgáltatnak, de ugyanakkor többet fogyasztanak, és interferencia léphet fel, ha egyszerre többet is használunk. Akár aktív, akár passzív érzékelőkről van szó, három csoportba lehet őket osztani az alapján, hogy távolságot mérnek, teljes képet közvetítenek a környezetről vagy a robot egyes saját tulajdonságait figyelik.

Sok mobil robot egy adott térrészt lefedő **pásztázó távolságmérő**t (**range finder**) használ, olyan szenzort, amely a közeli tárgyak távolságát méri. Gyakori típus a szonárszenzor, más néven hanglokátor. A szonárszenzorok irányított hanghullámokat bocsátanak ki, amelyek egy része visszaverődik a tárgyakról. A visszaverődés ideje és a hullám intenzitása információval szolgál a közeli tárgyak helyzetéről. Az AUV-kre (autonóm víz alatti jármű) víz alatti hanglokátorokat szerelnek. Szárazföldön a hanglokátorokat – rossz irányfelbontásuk miatt – főleg kis távolságon belüli ütközések elkerülésére használják. A hanglokátorok alternatíváját jelenthetik (elsősorban légi járműveknél) a radaralapú és a lézeres távolságmérők. Egy lézeres pásztázó távolságmérő látható a 25.2. ábrán.

25.2. ábra - (a) A mobil robotoknál népszerű SICK LMS lézeres távolságmérő. (b) Egy horizontálisan felszerelt távolságmérő által mért visszaverődési kép 2D-s térképre vetítve.

Created by XMLmind XSL-FO Converter.



Egyes távolságmérők csak nagyon kicsi vagy nagyon nagy távolságokra használhatók. A kis távolságú szenzorok közé tartoznak a **taktilis**, vagyis **tapintásalapú érzékelő**k (**tactile sensor**s), mint például az érzékelőbajusz, lökhárító vagy az érintésérzékeny bőr. A skála másik végén a **globális helymeghatározó rendszer** (**Global Positioning System**, **GPS**) áll, amely a műholdakból érkező impulzusok alapján számítja a távolságot. Jelenleg két tucat műhold kering orbitális pályán, és mindegyik két külön frekvencián sugároz jeleket. A GPS-vevőkészülékek a fázistolásból tudják kiszámítani a műholdtól való távolságukat. Több műholdtól érkező jel alapján háromszögeléssel néhány méteres pontossággal meghatározható a Földön elfoglalt tényleges pozíció. A **differenciális GPS** (**differential GPS**) még egy pontosan ismert helyen lévő földi vevő jelét is használva – ideális esetben – milliméteres pontosságot ér el. Sajnálatos módon a GPS nem használható beltérben vagy víz alatt.

A szenzorok másik fontos osztályát képezik a **képérzékelő**k (**imaging sensor**s). A kamerák képet adnak számunkra a környezetről, valamint – a 24. fejezetben tárgyalt, számítógépes gépi látási módszerek alkalmazásával – modellekkel, illetve tulajdonságokkal írják le a környezetet. A sztereolátás különösen fontos a robotikában, mivel mélységi információt is közvetít. Jövőjük mindamellett elég bizonytalan, mert sikeresen folyik új, aktív távolságleképezési technológiák kidolgozása.

A harmadik fontos osztályt az **önérzékelők** (**proprioceptive sensor**s) alkotják, amelyek a robot saját állapotáról adnak tájékoztatást. Annak érdekében, hogy a robot csuklóinak pontos állásáról információnk legyen, a motorok tengelyét sokszor **szöghelyzet-dekódoló**kkal (**shaft decoder**s) látják el, amelyek kis lépésekben követik a tengely elfordulását. A robotkarokon a tengelyre szerelt dekódolók bármikor pontos helyzetinformációt tudnak szolgáltatni. Mobil robotokon a szöghelyzet-dekódolókat a kerék mozgásának figyelésére használják, amely alapján számítható a megtett út. Ezt hívják **odometriá**nak (**odometry**). Sajnálatos módon az odometria – a kerekek csúszása és sodródása miatt – csak kis távolságokban használható. A külső hatások, mint például az áramlások az AUV-knél (autonóm víz alatti jármű) vagy a szél az UAV-k (autonóm légi jármű) esetében tovább növeli a mérés bizonytalanságát. Az **inerciaszenzor**ok (**inertial sensor**s), mint például a giroszkóp, javítanak a pontosságon, de önmagukban még nem küszöbölik ki a pozíciómérés hibájának elkerülhetetlen halmozódását.

A robotállapot más fontos jellemzőinek mérésére **erő**- és **nyomatékérzékelő**ket (**force**, **torque sensor**s) használnak. Ezek nélkülözhetetlenek, ha a robotnak törékeny tárggyal kell dolgoznia, vagy olyannal, amelynek pontos mérete és helyzete nem ismert. Képzeljük csak el, amikor egy közel egytonnás manipulátornak egy villanykörtét kell becsavarnia. Ha túl nagy erővel fogná meg a körtét, könnyen összeroppanthatná. Az erőérzékelők lehetővé teszik, hogy a robot tudja, milyen nehéz megfogni a körtét, míg a nyomatékérzékelők információt szolgáltatnak, hogy milyen nehéz becsavarni. A jó szenzorok képesek erőt mérni mindhárom elmozdulási és mindhárom elfordulási irányban.

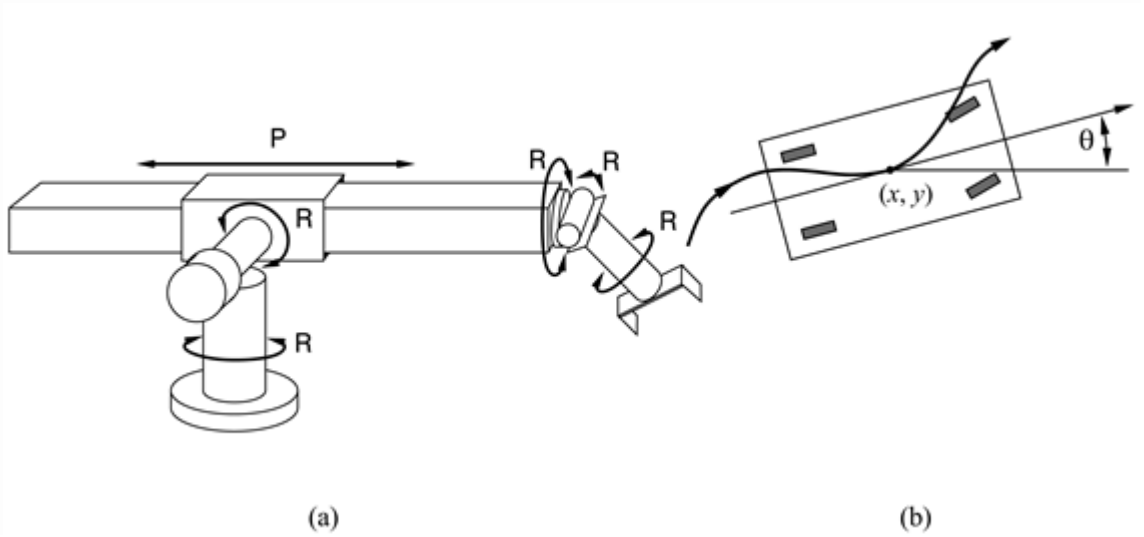
2.2. Beavatkozó szervek

A robotok beavatkozó szerveik segítségével mozognak és változtatják alakjukat. Ahhoz, hogy megértsük a beavatkozó szervek felépítését, először is általános értelemben kell beszélnünk a mozgás és az alak elvont értelmezéséről, a **szabadságfok**ok (**degree of freedom**, **DOF**) koncepciójának felhasználásával.

Created by XMLmind XSL-FO Converter.

Szabadságfoknak számít minden olyan irány, amelyben a robot vagy egyik beavatkozó szerve mozogni képes. Például egy merev, szabadon mozgó robotnak (mint amilyenek az AUV-k) hat szabadságfoka van, három az (*x*, *y*, *z*) térbeli elhelyezkedés, három pedig a szögelfordulás (orientáció). Ezeket szokták *csavarásnak*, *billentésnek* és *forgatásnak* is hívni (RPY – *roll, pitch, yaw*). Ez a hat szabadságfok meghatározza a robot kinematikus állapotát,[2] más néven a pozícióját és az orientációját együttesen. A robot **dinamikai állapot**a (**dynamic state**) minden egyes kinematikai paraméter változási sebességét is tartalmazza, így minden kinematikai dimenziónál egy-egy további dimenziót hoz be.

25.3. ábra - (a) A Stanford manipulátor: egy korai robotkar öt rotációs (R) és egy transzlációs (prizmatikus) csuklóval (P), összesen hat szabadságfokkal. (b) Egy elsőkerék-kormányzású nemholonomikus négykerekű jármű mozgása.



Nem merev testek esetében magán a roboton belül további szabadságfokok vannak. Például az emberi kar esetében a könyök egy szabadságfokú, mert egy irányba képes elfordulni. A csukló három szabadságfokú, mert tud fel-le és jobbra-balra mozogni, továbbá forogni is. Általában minden egyes robotcsuklónak is egy, két vagy három szabadságfoka van. Hat szabadságfok szükséges ahhoz, hogy például egy kezet a térben tetszőleges pozícióba és irányba (orientációba) hozzunk. A 25.3. (a) ábrán látható robotkarnak pontosan hat szabadsági foka van: öt **rotációs csukló** (**revolute joint**), amelyek forgást tesznek lehetővé, és egy **transzlációs csukló** (**prismatic joint**), amely kinyúlásra képes. Egy egyszerű kísérlet segítségével könnyedén meggyőződhetünk arról, hogy az emberi kéznek valójában hatnál több szabadságfoka van: kezünket az asztalra rakva még mindig képesek vagyunk elforgatni a könyökünket, anélkül hogy a kezünk konfigurációja megváltozna. Azokat a manipulátorokat, amelyeknek több szabadságfoka van, mint ami minimálisan szükséges ahhoz, hogy végbeavatkozó szervüket (end effector) a kívánt helyre vigyék, könnyebb vezérelni, mint azokat, amelyek csak a minimálisan szükséges számú szabadságfokkal rendelkeznek.

Mobil robotoknál a szabadságfokok száma nem feltétlenül egyezik meg a mozgatott részek számával. Vegyünk például egy átlagos autót: tud mozogni előre-hátra és tud fordulni. Ez így két szabadságfok. Ennek ellenére a kocsi kinematikája háromdimenziós: egy szabad lapos felületen könnyedén el tud jutni bármilyen (*x*, *y*) pontba, tetszőleges orientációval (lásd 25.3. (b) ábra). Azaz a kocsinak 3 **effektív szabadságfok**a (**effective DOF**) van, de 2 **irányítható szabadságfok**a (**controllable DOF**). A robotra azt mondjuk, hogy **nemholonomikus** (**nonholonomic**), ha az effektív szabadságfoka nagyobb, mint az irányítható szabadságfoka, míg **holonomikus** (**holonomic**), ha ez a kettő megegyezik. A holonomikus robotokat könnyebb irányítani. Mennyivel egyszerűbb lenne olyan kocsival parkolni, amely képes oldal irányban is mozogni, nem csak előre-hátra! Viszont a holonomikus robotok jellemző módon mechanikailag sokkal bonyolultabbak. A legtöbb robotkar holonomikus, míg a mobil robotok általában nemholonomikusak.

A mobil robotok számára mozgató mechanizmusok széles tárháza áll rendelkezésre: kerekek, lánctalpak, lábak stb. A **differenciál hajtás**ú (**differential drive**) robotoknak kétoldalt külön mozgatható kerekeik (vagy lánctalpaik) vannak, ugyanúgy, mint a tankoknak. Ha mindegyik kerék azonos sebességgel mozog, akkor a jármű egyenesen halad előre. Ha ellenkező irányba mozognak, akkor a robot képes egy helyben megfordulni. A

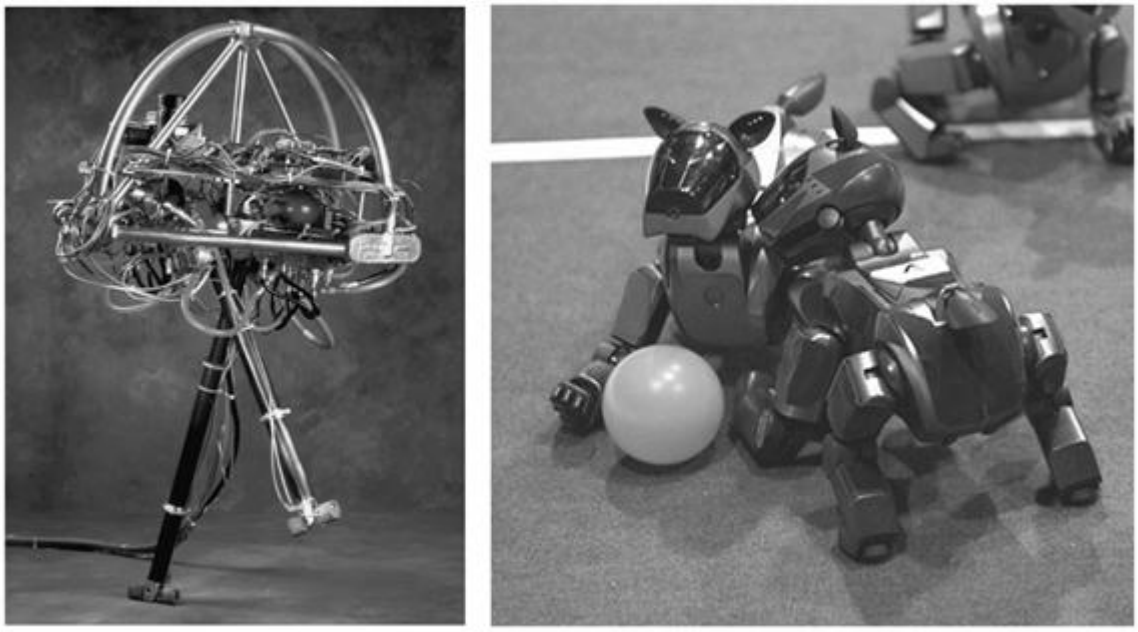
[2] A „kinematika” szó a görög „cinema”-ból ered, ami *mozgást* jelent.

Created by XMLmind XSL-FO Converter.

másik alternatívát a **szinkrón hajtás** (**synchro drive**) jelenti, amelynél minden kerék el tud fordulni a tengelye mentén. Ez könnyen vezethetne káoszhoz, ha nem lenne az a kényszer, hogy minden kerék mindig azonos irányba álljon, és azonos sebességgel forogjon. Mind a differenciál, mind a szinkrón hajtás nemholonomikus. Egyes drága robotok holonomikus hajtást használnak, általában három vagy több önállóan irányítható kerékkel.

A kerekekkel ellentétben a lábak még a nagyon nehéz tereppel is megbirkóznak. Ugyanakkor a lábak meglehetősen lassúak sík terepen, és mechanikailag nehezebb megépíteni őket. Robotikai szakemberek próbáltak már különféle konstrukciókat, egytől akár tucatnyi lábig. Készítettek lábbal rendelkező robotokat sétálásra, futásra, sőt még ugrálásra is. Egy ilyen példa látható a 25.4. (a) ábrán: ez a robot **dinamikusan stabil** (**dinamically stable**), ami azt jelenti, hogy képes talpon maradni, miközben körbe ugrál. **Statikusan stabil**nak (**statically stabile**) hívják azokat a robotokat, amelyek úgy képesek állva maradni, hogy nem mozognak a lábaik. A robot statikusan akkor stabil, ha a súlypontja a lábai által kifeszített sokszög fölé esik.

25.4. ábra - (a) Marc Raibert egyik lépegető robotja mozgás közben. (b) A Sony AIBO robot focizás közben (copyright 2001, The RoboCup Federation).



(a) (b)

Más típusú mobil robotok teljesen eltérő módszereket használnak a mozgáshoz. A légi járműveken általában propellereket vagy turbinákat alkalmaznak. Robot léghajók a meleg áramlatokat használják ki, hogy a levegőben maradjanak. Az autonóm víz alatti járművek gyakran alkalmaznak a tengeralattjárókon használtakhoz hasonló fúvókákat.

Szenzorok és beavatkozó szervek önmagukban még nem tesznek ki egy robotot. Egy igazi robotnak energiaforrásra is szüksége van, hogy mozgathassa beavatkozó szerveit. A legnépszerűbb megoldás mind a manipulátorok mozgatásához, mind a helyváltoztatáshoz, a villanymotor. Ugyanakkor a pneumatikus (sűrített gázzal működő) és a hidraulikus (folyadékkal közvetített nyomással működő) hajtásoknak is megvannak a maguk alkalmazási területei. A legtöbb robot valamiféle digitális kommunikációs eszközzel is rendelkezik, például vezeték nélküli hálózati kapcsolattal. Végezetül szükség van valami vázra is, amelyre fel lehet szerelni az alkatrészeket és az eszközöket, valamint egy forrasztópákára, ha az alkatrészek és eszközök megszűnnek működni.

3. Érzékelés a robotikában

Az érzékelés az a folyamat, amelynek során a robot a szenzoraiból érkező jeleket környezetének belső reprezentációjára képezi le. Az érzékelés nehéz, mivel általában a szenzorok adatai zajjal terheltek, a környezet csak részlegesen megfigyelhető, nem jósolható viselkedésű és gyakran dinamikus is. Ökölszabály, hogy egy jó belső reprezentációnak három tulajdonsága van: elég információt hordoz ahhoz, hogy a robot a megfelelő döntést meghozhassa; strukturáltsága lehetővé teszi, hogy hatékonyan frissíthető legyen; valamint természetes

Created by XMLmind XSL-FO Converter.

olyan értelemben, hogy a belső állapotváltozók megfeleltethetők egy-egy valós fizikai világbeli állapotváltozónak.

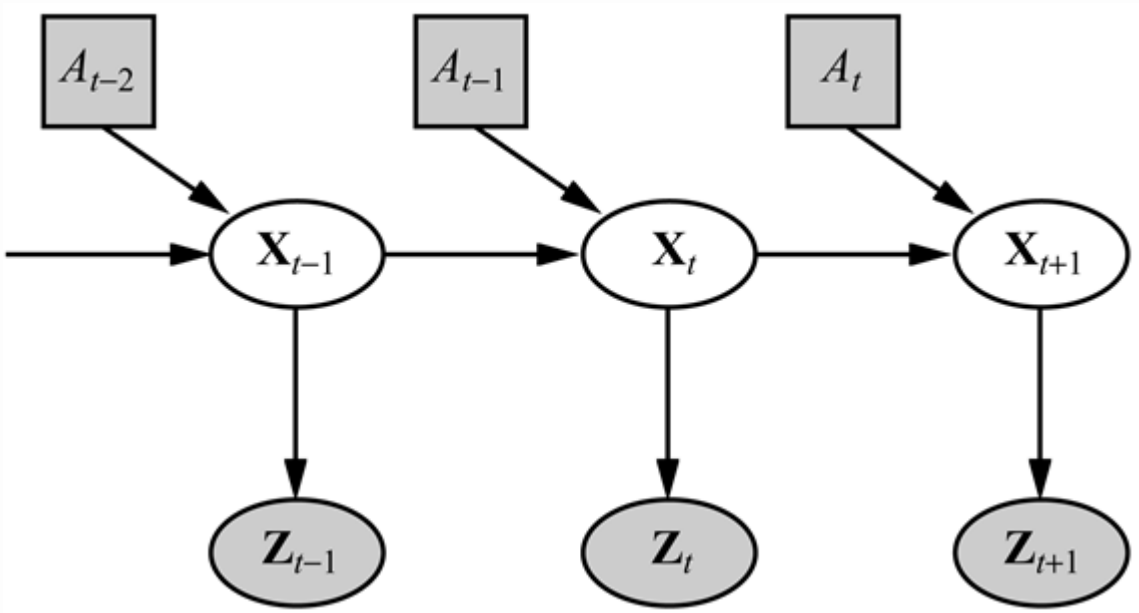
A 15. fejezetben megmutattuk, hogy a Kalman-szűrők, a rejtett Markov-modellek és a dinamikus Bayes-hálók alkalmasak egy részlegesen megfigyelhető környezet állapotátmenet- és érzékelő modelljeinek a reprezentálására. Ismertettünk egzakt és közelítő algoritmusokat a belső **hiedelmi állapot** (**belief state**) – a környezet állapotváltozói felett értelmezett a posteriori valószínűség-eloszlás – frissítésére. Erre számos dinamikus Bayes-hálós modellt mutattunk be a 15. fejezetben. Robotikai problémák esetében a modellhez általában megfigyelt változóként a robot saját korábbi cselekvéseit is hozzáveszszük, mint ahogy az a 17.9. ábra hálózatán látható. A 25.5. ábra ennek a fejezetnek a jelöléseit mutatja: $\mathbf{X}_t$ a környezet állapota (beleértve a robotot is) a t időpillanatban; $\mathbf{Z}_t$ a t időpontbeli megfigyelés (az érzékelésből származó adatok), az A_t pedig az érzékelést követően végzett cselekvés.

A **szűrés**i (**filtering**) feladat, vagyis a hiedelmi állapot frissítése alapvetően azonos a 15. fejezetben tárgyaltakkal. A feladat az új hiedelmi állapot, $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{z}_{1:t+1}, a_{1:t})$ kiszámítása a $\mathbf{P}(\mathbf{X}_t|\mathbf{z}_{1:t}, a_{1:t-1})$ aktuális hiedelmi állapotból és az új $\mathbf{z}_{t+1}$ megfigyelésből. Az alapvető különbség az, hogy (1) explicit feltételekkel élünk mind a döntést, mind a megfigyelést illetően, (2) *folytonos* változókkal kell dolgoznunk *diszkrétek* helyett. Ezért módosítanunk kell a (15.3)-as rekurzív szűrőalgoritmust, integrált használva az összegzés helyett:

$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{z}_{1:t+1}, a_{1:t}) = \alpha \mathbf{P}(\mathbf{z}_{t+1}|\mathbf{X}_{t+1}) \int \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t, a_t) P(\mathbf{x}_t|\mathbf{z}_{1:t}, a_{1:t-1}) d\mathbf{x}_t \qquad (25.1)$$

Az egyenlet azt fejezi ki, hogy az $\mathbf{X}$ állapotváltozó feletti a posteriori eloszlást a $t + 1$ időpillanatbani rekurzívan számítjuk az egy időlépéssel korábbi megfelelő becslésből. A számításokhoz felhasználjuk a korábbi cselekvést, a_t-t, és az aktuális szenzoros megfigyelést, $\mathbf{z}_{t+1}$-et. Például ha célunk egy futballozó robot fejlesztése, akkor $\mathbf{X}_{t+1}$ lehet a labda relatív helyzete a robothoz képest. A posteriori $\mathbf{P}(\mathbf{X}_t|\mathbf{z}_{1:t}, a_{1:t-1})$ mindazon állapotok felett értelmezett valószínűség-eloszlás, amelyek megőrzik mindazt, amit a korábbi érzékelő mérésekből és irányításokból tudunk. A (25.1) egyenlet megmondja, hogyan becsüljük rekurzívan ezt a pozíciót, folyamatosan felhasználva az újabb szenzoradatokat (például kameraképek) és a robot mozgásparancsait. A $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t, a_t)$ valószínűséget **állapotátmenet-modell**nek (**transition model**) vagy más néven **mozgásmodell**nek (**motion model**) hívjuk, míg $\mathbf{P}(\mathbf{z}_{t+1}|\mathbf{x}_{t+1})$ az **érzékelő modell** (**sensor model**).

25.5. ábra - A robotérzékelés felfogható úgy, mint cselekvések és mérések sorozatának időbeli egymásra hatása, ahogy ezt a dinamikus Bayes-háló is illusztrálja



3.1. Helymeghatározás

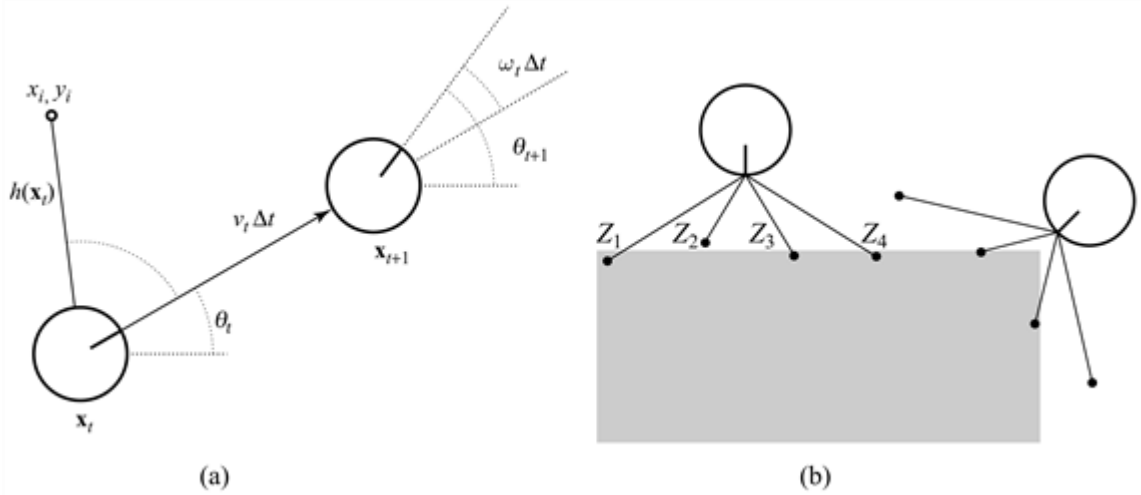
A **helymeghatározás** (**localization**) jellemző példa a robotérzékelésre. A probléma lényege a dolgok pontos helyzetének meghatározása. A helymeghatározás az egyik legfontosabb érzékelési feladat a robotikában, mivel a fizikai környezettel való sikeres kölcsönhatáshoz feltétlenül szükséges. Például a robotkaroknak tudniuk kell,

Created by XMLmind XSL-FO Converter.

hol van az a tárgy, amivel dolgozni akarnak. A navigáló robotoknak pedig pontosan ismerniük kell saját helyzetüket, hogy eljuthassanak a céljukhoz.

A helymeghatározási feladat három, egyre nehezebb problémaként jeletkezik. Ha a tárgy kezdeti pozíciója és iránya (orientációja) ismert, akkor a helymeghatározás tulajdonképpen **követési feladat**ot (**tracking**) jelent, és ez korlátos bizonytalansággal jellemezhető. Ennél nehezebb a **globális helymeghatározás** (**global localization**), amikor is a kezdeti pozíció egyáltalán nem ismert. A globális helymeghatározás követési problémává egyszerűsödik, ha sikerül lokalizálni a kívánt tárgyat, de itt is előfordulhatnak olyan esetek, amikor a robotnak nagyon nagy bizonytalanságokkal kell megbirkóznia. Végezetül lehetünk gonoszak is a robotunkkal, ha elvesszük, „elraboljuk" előle az éppen lokalizálni próbált tárgyat. Ezt **elrablásos problémá**nak (**kidnapping problem**) hívják, és gyakran tesztelik vele a robot lokalizációs algoritmusának robusztusságát extrém körülmények között.

25.6. ábra - (a) Egy mobil robot egyszerűsített kinematikai modellje. A robotot a kör jelképezi, és a bevágás mutatja a haladási irányt. Külön-külön láthatjuk a *t*-beli és a *t* + 1-beli pozíciót és orientációt a $v_t \Delta_t$ és $\omega_t \Delta t$ frissítési értékekkel. Szintén fel van tüntetve egy *t* időpontban megfigyelt referenciapont (tereptárgy) az (x_i, y_i) pontban. (b) A pásztázó távolságmérés modellje. Egy adott távolságú méréshez (z_1, z_2, z_3, z_4) tartozó két lehetséges robothelyzet látható. Sokkal valószínűbb, hogy a bal oldali helyzetből származnak a távolságmérések.



Azért, hogy egyszerűvé tegyük a dolgot, feltételezzük, hogy a robotunk lassan mozog sík terepen, és pontos térképe van a környezetéről. (Egy ilyen térkép látható a 25.8. ábrán.) Egy mobil robot helyzetét két derékszögű koordinátájával (x, y) és az irányát jellemző szöggel írjuk le, ahogy ez a 25.6. (a) ábrán is látható. (Mivel nem foglakoztunk a hozzá tartozó sebességekkel, ez inkább kinematikai, mint dinamikus modell.) Ha ezt a három értéket egy vektorba fogjuk össze, akkor bármely állapot megadható $\mathbf{X}_t = (x_t, y_t, q_t)^\top$ formában.

Kinematikai közelítésben minden cselekvés felbontható két „pillanatnyi" sebességre: egy v_t (transzlációs) sebességre és egy w_t (rotációs) szögsebességre. Kis Δt időkre a robot mozgásának egy durva determinisztikus modellje az alábbi alakban adható meg:

$$\hat{\mathbf{X}}_{t+1} = f\left(\mathbf{X}_t, \underbrace{v_t, \omega_t}_{a_t}\right) = \mathbf{X}_t + \begin{pmatrix} v_t \Delta \cos\theta_t \\ v_t \Delta t \sin\theta_t \\ \omega_t \Delta t \end{pmatrix}$$

Az $\hat{\mathbf{X}}$ jelölés a determinisztikus állapotbecslésre vonatkozik. Természetesen a fizikai robotok valamilyen módon mindig megjósolhatatlanok maradnak. Ezt gyakran modellezik egy $f(\mathbf{X}_t, v_t, \omega_t)$ középértékű, $\mathbf{\Sigma}_x$ kovarianciájú Gauss-eloszlással. (Lásd még az A) függelék matematikai definícióit.)

$$\mathbf{P}\left(\mathbf{X}_{t+1} \middle| \mathbf{X}_t, v_t, \omega_t\right) = N\left(\hat{\mathbf{X}}_{t+1}, \mathbf{\Sigma}_x\right)$$

Created by XMLmind XSL-FO Converter.

A következő lépésben szükségünk van egy érzékelő modellre. Kétféle érzékelő modellről beszélhetünk. Az első azon alapul, hogy a szenzor **referenciapont**oknak (**landmark**s) nevezett *állandó* és *felismerhető* jellemvonásokat érzékel és azonosít a környezetében. Minden referenciapontról meghatározza annak távolságát és szögét. Tegyük fel, hogy a robot az $\mathbf{x}_t = (x_t, y_t, q_t)^T$ állapotban van, és érzékel egy ismert $(x_i, y_i)^T$ helyen lévő referenciapontot. Ha nem vesszük bele az érzékelés zajosságát, egyszerű geometriával megkapható a távolság és a szög (lásd a 25.6. (a) ábra). Megfigyelés alapján a pontos becslés a távolságra és a szögre:

$$\hat{\mathbf{z}}_t = h(\mathbf{x}_t) = \begin{pmatrix} \sqrt{(x_t - x_i)^2 + (y_t - y_i)^2} \\ \arctan\frac{y_i - y_t}{x_i - x_t} - \theta_t \end{pmatrix}$$

A zaj most is torzítja a méréseinket. Egyszerűsítésül Gauss-eloszlású, $\mathbf{\Sigma}_z$ kovarianciájú *i* zajt tételezhetünk fel:

$$P(\mathbf{z}_t \mid \mathbf{x}_t) = N(\hat{\mathbf{z}}_t, \mathbf{\Sigma}_z)$$

A 25.2. ábrán bemutatott távolságszenzorokhoz kicsit másféle érzékelő modell is megfelelő lehet. Az ilyen érzékelők egy $\mathbf{z}_t = (z_1, \ldots, z_M)^T$ vektort adnak vissza távolságadatokkal, amelyek a robothoz képest rögzített irányokban lévő tárgyakról hordoznak információt. Adott $\mathbf{x}_t$ pozícióban legyen $\hat{z}_j$ a *j*-edik sugár irányába lévő legközelebbi akadály pontos távolsága. Csakúgy, mint korábban, most is Gauss-zajt tételezünk fel. Általánosan vehetjük úgy, hogy a különböző sugárirányokbeli hibák függetlenek és azonos eloszlásúak, így:

$$P(\mathbf{z}_t \mid \mathbf{x}_t) = \alpha \prod_{j=1}^{M} e^{-(z_j - \hat{z}_j)/2\sigma^2}$$

A 25.6. (b) ábra egy példát mutat be négyirányú távolságmérő szkennerre, két különböző pozíció esetén. A kettő közül az egyik sokkal nagyobb valószínűséggel adhatta a megfigyelés négy értékét. Összehasonlítva a pásztázó távolságméréses modellt a referenciapontos modellel látható, hogy az előbbinek az az előnye az utóbbival szemben, hogy nincs szükség egy referenciapont *azonosítására* mielőtt egy tartománypásztázás értelmezhető lenne. Sőt az is előfordulhat – ahogy a 25.6. (b) ábra esetében is –, hogy a robot egy jellegtelen fallal találkozik. Ugyanakkor, ha van egy *tisztán látható*, azonosítható referenciapont, akkor abból azonnal megkapható a pontos helyzet.

A 15. fejezetben már találkoztunk a Kalman-szűrővel, amely a hiedelmi állapotot egyetlen többváltozós Gauss-eloszlással reprezentálja, illetve a részecskeszűrővel, amely a hiedelmi állapotot az állapotokhoz rendelt részecskeegyüttessel reprezentál-ja. A legtöbb modern lokalizáló algoritmus e két módszer egyikét használja a robot $\mathbf{P}(\mathbf{X}_t | \mathbf{z}_{1:t}, a_{1:t-1})$ hiedelmi állapotának leképezésére.

A részecskeszűrős lokalizációs algoritmust **Monte Carlo lokalizáció**nak (**Monte Carlo localization**, **MLC**) hívják. Az MCL alapvetően megegyezik a 15.15. ábrán bemutatott részecskeszűrős algoritmussal, mindössze annyit kell módosítanunk, hogy szükségünk van a megfelelő mozgás és szenzor modellre. A 25.7. ábra bemutat egy változatot, pásztázó távolságméréses modellel. Az algoritmus működését, azt, hogy hogyan azonosítja a robot saját tartózkodási helyét az irodaépületben, a 25.8. ábra szemlélteti. Az első képen még a részecskék egyenletesen oszlanak el, az előzetes információ alapján, jelképezve a teljes bizonytalanságot a robot pozícióját illetően. A második képen már az első mérések alapján klaszterekét alkotnak a részecskék, oda tömörülve, ahol valószínűleg tartózkodik a robot. A harmadik esetben pedig már elég információ áll rendelkezésre a mérésekből, hogy az összes részecske ugyanoda jusson.

25.8. ábra - A Monte Carlo lokalizáció, egy részecskeszűrő algoritmus mobil robotok pozíciójának meghatározására. Felül: a kezdeti, teljes bizonytalanság. Középen: nagyjából kétpólusú bizonytalanság, miután a robot elindult a középső (szimmetrikus) folyosón. Alul: egyetlen pont köré szűkült bizonytalanság, miután bement az egyik irodába.

Created by XMLmind XSL-FO Converter.


A robot helyzete


A robot helyzete



25.7. ábra - A Monte Carlo lokalizáló algoritmus, független zajjal terhelt pásztázó távolságméréses modellt használva

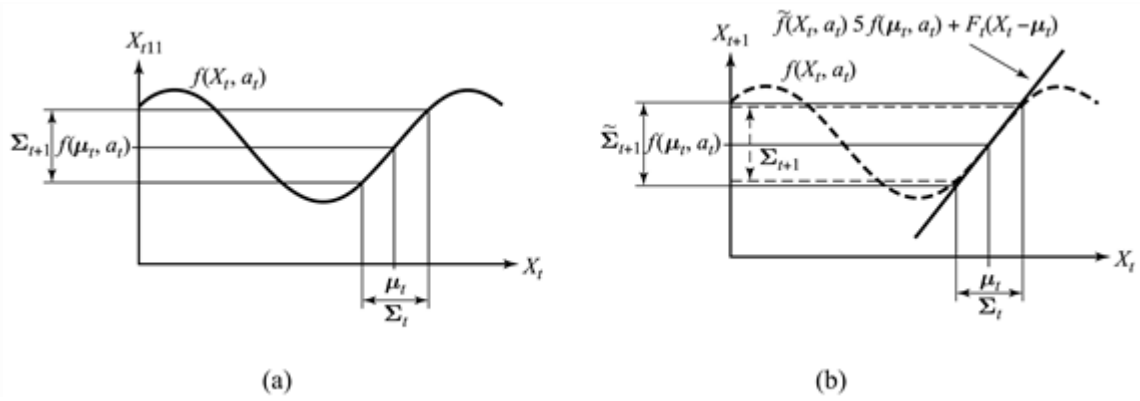
function MONTE-CARLO-HELYZETMEGHATÁROZÁS(a, z, N, *modell*, *térkép*) **returns** egy mintahalmaz
  inputs: a, a megelőző robotmozgás-parancs
    z, egy pásztázó távolságmérés M méréssel $z_1,..., z_M$
    N, a fenntartandó minták száma
    modell, egy valószínűségi környezet modell a helyzetre vonatkozó $\mathbf{P}(\mathbf{X}_0)$ priorral
      $\mathbf{P}(\mathbf{X}_1|\mathbf{X}_0, A_0)$ mozgásmodell és pásztázó távolságmérő $P(Z|\hat{Z})$ zajmodell
    térkép, a környezet 2D térképe
  static: S, egy N mintából álló vektor, kezdetben a $\mathbf{P}(\mathbf{X}_0)$ szerint generálva
  local variables: W, egy N elemű súlyvektor

  for $i = 1$ **to** N **do**
    $S[i] \leftarrow$ minta a $\mathbf{P}(\mathbf{X}_1 | \mathbf{X}_0 = S[i], A_0 = a)$-ból
    $W[i] \leftarrow 1$
    for $j = 1$ **to** M **do**
      $\hat{z} \leftarrow$ EGZAKT-TARTOMÁNY(j, $S[i]$, *térkép*)
      $W[i] \leftarrow W[i] \cdot P(Z = z_j | \hat{Z} = \hat{z})$
  $S \leftarrow$ SÚLYOZOTT-MINTA-HELYETTESÍTÉSSEL(N, S, W)
  return S

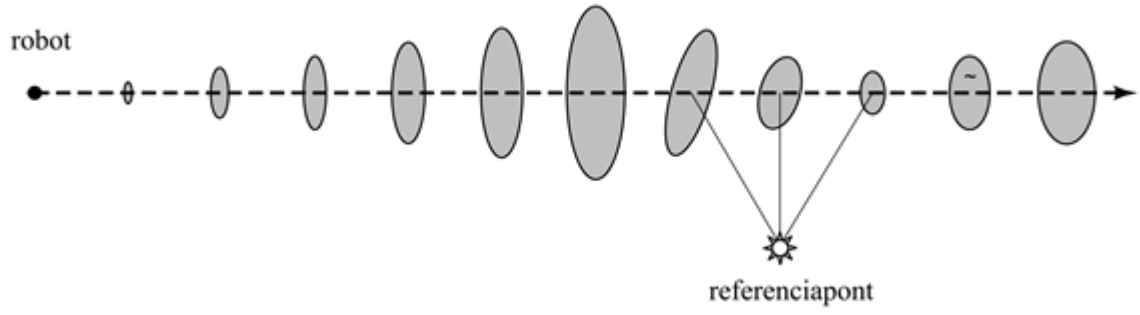
A Kalman-szűrő a másik széles körben használt lokalizáló algoritmus. A Kalman-szűrő a $\mathbf{P}(\mathbf{X}_t|\mathbf{z}_{1:t}, a_{1:t-1})$ posteriort egy Gauss-eloszlással reprezentálja. Az eloszlás középértékét $\boldsymbol{\mu}_t$-vel, kovarianciáját pedig $\boldsymbol{\Sigma}_t$-vel jelöljük. A fő probléma a Gauss-hiedelmekkel az, hogy csak lineáris mozgási modell, f és lineáris mérési modell, h mellett zártak. Nemlineáris f vagy h esetén a szűrő frissítésének eredménye általában már nem lesz Gauss-eloszlású. Így Kalman-szűrőt használó lokalizációs algoritmusok **linearizál**ják a mozgási és az érzékelő modelleket. A linearizálás egy nemlineáris függvény adott lokális lineáris közelítése. A 25.9. ábra a linearizálás elvét mutatja be egy (egydimenziós) robotmozgási modellen. A bal oldalon fut a mozgás nemlineáris $f(\mathbf{x}_t, a_t)$ modellje (az a_t vezérlés nem szerepel az ábrán, mert nem játszik szerepet a linearizálásban). A jobb oldalon ezt egy $f(\mathbf{x}_t, a_t)$ lineáris függvénnyel közelítjük. A lineáris függvény a $\boldsymbol{\mu}_t$ pontban érinti az f-et, ami a t időre vonatkozó becslésünk középértéke. Ezt a linearizálást (első fokú) **Taylor-sorfejtés**nek (**Taylor expansion**) hívják. Az olyan Kalman-szűrőt, amely f-et és h-t Taylor-sorfejtéssel linearizálja, **kiterjesztett Kalman-szűrő**nek (**KKSZ**; **extended Kalman Filter** – **EKF**) hívjuk. A 25.10. ábra egy KKSZ helymeghatározó algoritmust használó robot becsléssorozatát mutatja. Ahogy a robot halad, a helyzetére vonatkozó becslés bizonytalansága egyre nő, ahogy azt a hibaellipszis is mutatja. A hiba csökken, ahogy a robot egy ismert elhelyezkedésű referenciapont távolsága és szöge alapján javítani tudja saját helyzetére vonatkozó becslését. A hiba végül újra nőni kezd, ahogy a referenciapont kikerül a robot látóköréből. A KKSZ algoritmus jól működik, ha könnyen azonosítható referenciapontok vannak. Ellenkező esetben az a posteriori eloszlás multimodális is lehet, mint a 25.8. (b) ábrán. A referenciapontok azonosításának szükségességét jelentő probléma csak egy példa a 15. fejezet végén tárgyalt **adatasszociáció**s (**data association**) problémának.

25.9. ábra - Egydimenziós, linearizált mozgási modell illusztrálása: (a) Az f függvény és $t + 1$ időpontbeli μ_t középértéke és kovarianciája (Σ_t alapján). (b) A linearizált változat az f érintője a μ_t pontban. A μ_t középérték ugyanaz marad, ugyanakkor az előrevetített $\tilde{\Sigma}_{t+1}$ kovariancia már különbözik a Σ_{t+1}-től.

Created by XMLmind XSL-FO Converter.



25.10. ábra - Példa a kiterjesztett Kalman-szűrő használatára lokalizációs problémához. A robot egy egyenes mentén mozog. Ahogy halad előre, a saját helyzetének ismeretéhez tartozó bizonytalanság fokozatosan nő, ahogy azt a hibaellipszisek mutatják. Amikor észrevesz egy ismert pozíciójú referenciapontot, a bizonytalanság csökken.



3.2. Térképezés

Az előzők során egyetlen tárgy helymeghatározásának problémáját tárgyaltuk meg. A robotikában azonban általában egyszerre több tárgy helyzetét kell meghatározni. A klasszikus példa erre a feladatra a térképezés. Képzeljünk el egy robotot, amelyiknek nincs pontos térképe a környezetéről, magának kell azt elkészítenie. Az ember fantasztikus képességeket fejlesztett ki, hogy feltérképezze környezetét, mára már az egész bolygót. Természetesen adódó feladat egy olyan algoritmus kitalálása, amely képessé teszi ugyanerre a robotot is.

Az irodalomban a robottérképezés problémáját gyakran mint **szimultán helymeghatározást és térképezést** (**SZLT**; **simultaneous localization and mapping, SLAM**) említik. A robotnak nem csupán egy térképet kell konstruálnia, hanem ezt anélkül kell tennie, hogy ismerné, ő maga hol tartózkodik. Az SZLT az egyik alapvető robotikai probléma. Azt a változatot fogjuk tekinteni, amikor a környezet állandó. A feladat így is meglehetősen bonyolult; és még nehezebbé válik, ha megengedjük, hogy a környezet változzon, ahogyan a robot mozog benne.

Statisztikai oldalról nézve a térképezés egy Bayes következtetési probléma, csakúgy, mint a lokalizáció. Jelöljük M-mel a térképet és $\mathbf{X}_t$-vel a robot t pillanatbeli helyzetét, mint ahogy azt korábban is tettük. Ekkor átírhatjuk a (25.1) egyenletet, hogy a posterior tartalmazza a teljes térképet is:

$$\mathbf{P}(\mathbf{X}_{t+1}, M \,|\, \mathbf{z}_{1:t+1}, a_{1:t}) = \alpha \mathbf{P}(\mathbf{z}_{t+1} \,|\, \mathbf{X}_{t+1}, M) \int \mathbf{P}(\mathbf{X}_{t+1} \,|\, \mathbf{x}_t, a_t) \mathbf{P}(\mathbf{x}_t, M \,|\, \mathbf{z}_{1:t}, a_{1:t-1}) d\mathbf{x}_t$$

Szerencsére az egyenletből látszik, hogy a feltételes eloszlások, amelyek a cselekvések és mérések figylembevételéhez kellenek, alapvetően megegyeznek a helymeghatározás problémájánál használtakkal. A fő különbség az, hogy az új állapottér már sokkal több dimenziós, mert tartalmazza az összes robothelyzeten kívül az összes térképet is. Képzeljük csak el, ha egy egész épületet akarunk leképezni fotorealisztikusan! Ez esetleg több száz millió numerikus adatot igényelne. Minden egyes numerikus adat egy véletlen változó lesz, és hozzájárul, hogy az állapottér elképesztően nagy dimenziós legyen. Ami még tovább bonyolítja a problémát, hogy a robot esetleg előre nem is tudja, hogy milyen nagy a környezete, ezért M dimenzióját a térképezés során dinamikusan kellene növelni.

Created by XMLmind XSL-FO Converter.

Talán a legszélesebb körben használt módszer az SZLT-probléma megoldására a KKSZ. A szűrőhöz általában társítanak még egy referenciapont-érzékelő modellt is, valamint kitétel, hogy az összes referenciapont megkülönböztethető legyen. A korábbiakban a posterior becslését egy $\boldsymbol{\mu}_t$ várható értékű, $\boldsymbol{\Sigma}_t$ kovarianciájú Gauss-eloszlással jellemeztük. Az SZLT/probléma KKSZ-megközelítésénél a posterior megint csak Gauss-eloszlású lesz, de most a középérték egy sokkal nagyobb vektor. Nemcsak a robot helyzetét rögzíti, hanem tartalmazza a térkép összes jellemzőjét (vagy referenciapontját) is. Amennyiben n ilyen jellemzőnk van, akkor a vektor $2n + 3$ dimenziós lesz (két paraméterrel írható le a referenciapontok helyzete és hárommal a robot póza). Következésképpen a $\boldsymbol{\Sigma}_t$ mátrix $(2n + 3)$-szor $(2n + 3)$-as lesz, az alábbi struktúrával:

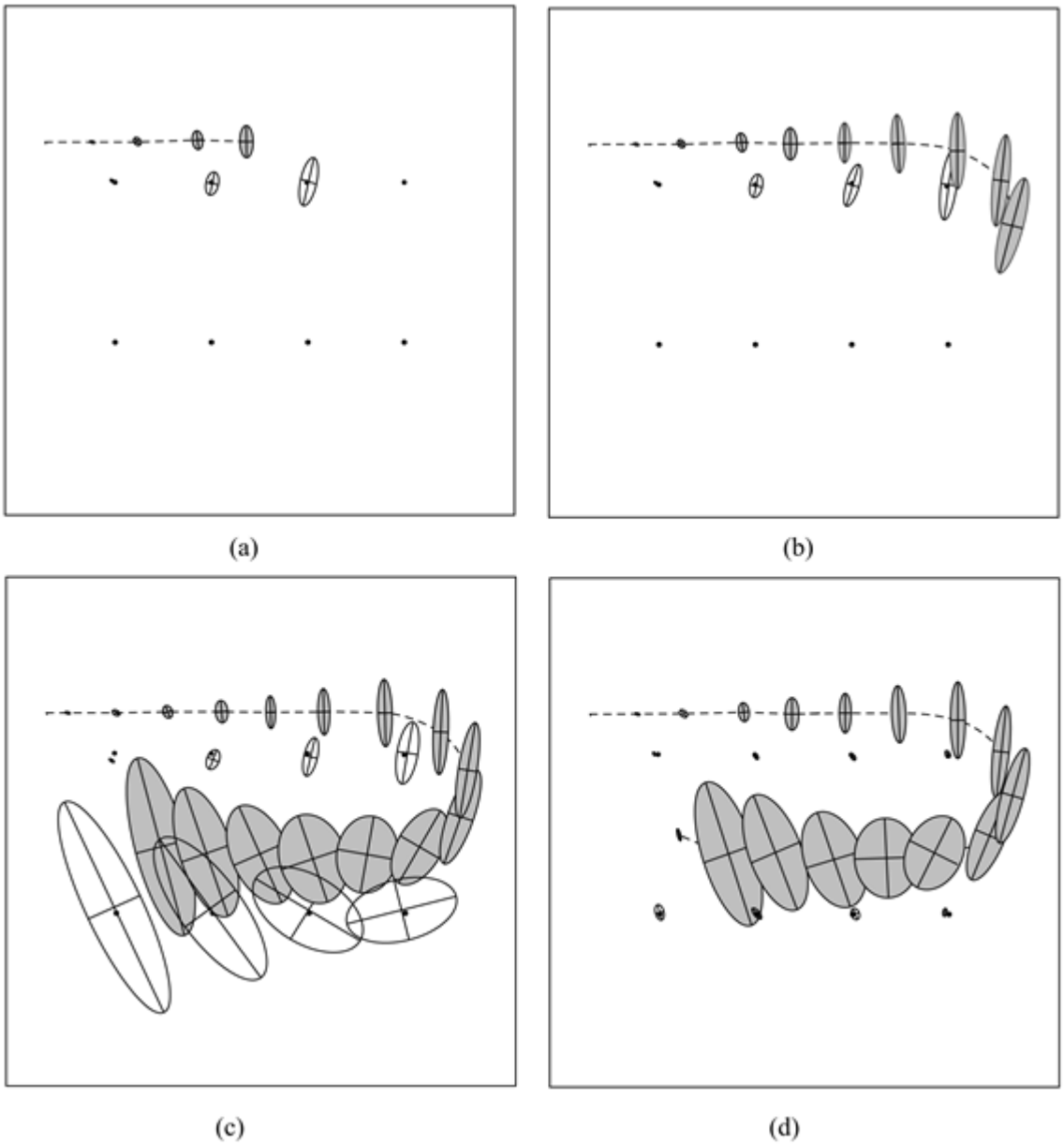
$$\boldsymbol{\Sigma}_t = \begin{pmatrix} \boldsymbol{\Sigma}_{XX} & \boldsymbol{\Sigma}_{XM} \\ \boldsymbol{\Sigma}_{XM}^{\top} & \boldsymbol{\Sigma}_{MM} \end{pmatrix} \qquad (25.2)$$

Ahol is a $\boldsymbol{\Sigma}_{XX}$ a robot helyzetének a kovarianciája, amit már korábban figyelembe vettünk a helymeghatározásnál. $\boldsymbol{\Sigma}_{XM}$ egy 3-szor $2n$-es mátrix, amely a referenciapontok és a robotkoordináták közötti korrelációt fejezi ki. Végezetül a $\boldsymbol{\Sigma}_{MM}$ egy $2n$-szer $2n$-es mátrix, ami a térképjellemzők kovarianciáját írja le, beleértve az összes páronkénti korrelációt. Így a KKSZ memóriaigénye kvadratikusan nő n-nel, a térképen lévő jellemzők számával, és a frissítési idő szintén kvadratikus n-ben.

Mielőtt elmerülnénk a matematikai részletekben, tanulmányozzuk grafikusan a KKSZ-t. A 25.11. ábra egy robotot mutat, amelynek környezetében nyolc referenciapont helyezkedik el két sorban, mindegyikben négy referenciaponttal. Kezdetben a robotnak nincs információja arról, hogy hol helyezkednek el a referenciapontok. Tételezzük fel, hogy az összes pont más színű, és a robot ezeket egyértelműen meg tudja különböztetni egymástól! A robot egy ismert pozícióból balra indul el, de fokozatosan elveszti pontos helyének ismeretét. Ezt jelölik a hibaellipszisek a 25.11. (a) ábrán. Az ellipszisek szélessége növekszik, ahogy a robot halad előre. Ahogy a robot mozog, a közeli referenciapontok távolságának és szögének érzékelése révén a robot meg tudja becsülni azok helyzetét. Természetesen ezeknek a becsléseknek a bizonytalansága nagyban függ a robot saját helyzetére vonatkozó becslésének bizonytalanságától. A 25.11. (b) és (c) ábrák illusztrálják a robot hiedelmét, ahogy fokozatosan halad előre környezetében.

25.11. ábra - A KKSZ alkalmazása a robottérképezés problémájára. A robot pályája a szaggatott vonal, és a besatírozott ellipszisek jelentik helyzetére vonatkozó saját becsléseit. Nyolc pont jelenti az ismeretlen helyzetű referenciapontokat, míg a helyzetükre vonatkozó becsléseiket a fehér ellipszisek mutatják. Az (a), (b) és (c) ábrákon a robot helyzetére vonatkozó becslés bizonytalansága folyamatosan növekszik, ahogy azoké a referenciapontoké is, amelyek mellett elmegy. A (d) ábrán a robot újra elérkezik az első referenciaponthoz, így az összes többi pont helyzetének bizonytalansága is csökken, köszönhetően annak, hogy a becslések korreláltak.

Created by XMLmind XSL-FO Converter.



Ezen becslések fontos jellemzője – ami egyáltalán nem magától értetődő az ábrákon –, hogy a becsléseket *egyetlen* Gauss-eloszlás jellemzi. A 25. 11. ábrán látható hibaellipszisek csupán ennek az eloszlásnak a vetületei a robot- és referenciakoordináták alterébe. Ez a többváltozós Gauss a posteriori eloszlás tartja fenn a korrelációt az összes becslés között. Fontos ez az észrevétel annak megértéséhez, hogy mi történik a 25.11. (d) ábrán. Itt a robot észrevesz egy korábban már feltérképezett referenciapontot. Ennek következtében saját pozíciójának bizonytalansága drasztikusan csökken, mint ahogy az összes többi referenciapont bizonytalansága is. Ez annak a következménye, hogy a robot és a pontok helyzetének becslése erősen korreláltak a Gauss a posteriori eloszlásban. Bármilyen új információ egy változóról (ebben az esetben a robot helyzetéről) automatikusan csökkenti az összes többi változó bizonytalanságát.

A térképezéshez használatos KKSZ algoritmus hasonlít a korábban a lokalizációhoz használt KKSZ-re. A legfontosabb különbséget a referenciapont-változók hozzáadása jelenti a posteriorban. A referenciapontok mozgásmodellje triviális: nem mozognak. Így az *f* függvény azokra a változókra az identitásfüggvény. A mérési függvény alapvetően megegyezik a korábban használttal. Az egyetlen különbség a KKSZ *frissítési* egyenletben az, hogy a H_t Jacobi-mátrixnak nemcsak a robot pozícióját kell figyelembe vennie, hanem a referenciapontok *t*-ben megfigyelt helyzetét is. Az így kapott KKSZ-egyenletek még ijesztőbbek, mint a korábbiak, ezért itt nem is tárgyaljuk őket.

Ugyanakkor van még egy nehézség, amit szép csöndben figyelmen kívül hagytunk eddig: az *M* térkép méretét nem tudjuk előre. Ebből kifolyólag a végső becsléshez tartozó $\boldsymbol{\mu}_t$ és $\boldsymbol{\Sigma}_t$ elemeinek száma szintén ismeretlen.

Created by XMLmind XSL-FO Converter.

Ezeket dinamikusan kell meghatározni, ahogy a robot újabb referenciapontokat talál. A probléma megoldása elég egyszerű: amikor a robot felfedez egy új referenciapontot, egyszerűen hozzáad egy új elemet a posteriorhoz. Ha ennek az új elemnek a varianciáját nagyon nagy kezdeti értékűre választjuk, akkor az eredményül kapott posterior ugyanaz, mintha a robot már korábban is tudott volna annak a referenciapontnak a létezéséről.

3.3. További érzékelési típusok

Nem minden robotérzékelés helymeghatározás és térképezés. A robotok érzékelhetnek hőmérsékletet, szagokat, hangokat stb. Ezek közül sok valószínűségi alapon becsülhető, mint ahogyan a lokalizációnál és a térképezésnél láttuk. Mindössze olyan feltételes valószínűség-eloszlásokat kell találni, amelyek leírják az állapotváltozók időbeli viselkedését, illetve olyan eloszlásokat, amelyek jellemzik a mérések és az állapotváltozók kapcsolatát.

Azonban nem minden működő robotérzékelési rendszer alapul valószínűségi modellezésen. Valójában nem szükséges, hogy minden belső állapotnak legyen valós fizikai interpretációja, ahogy az a mi példáink esetén volt. Például képzeljünk magunk elé egy járó robotot, amely éppen az egyik lábát próbálja egy akadály felett átemelni. Feltételezzük, hogy a robot olyan szabályt alkalmaz, amely szerint kiindulásként csak kicsit emeli fel a lábát, de egyre nagyobb és nagyobb magasságokkal próbálkozik, ha a lába folyton beleütközik valamilyen akadályba. Mondhatjuk-e, hogy az emelt láb magassága valamilyen valós fizikai paraméter reprezentációja? Esetleg igen, és akkor az az akadály magasságára, illetve átléphetőségére vonatkozik. Ugyanakkor mondhatjuk azt is, hogy a lábmagasság a robot vezérlésének egy segédváltozója, mindenfajta fizikai jelentés nélkül. Az ilyen reprezentációk nem ritkák a robotikában, és bizonyos feladatok megoldására kiválóan alkalmasak.

A robotika jelenlegi trendjei egyértelműen egy jól definiált szemantikával rendelkező leírás felé mutatnak. A valószínűség-alapú modellek egyre inkább kiszorítják a többi megoldást olyan komplex problémák esetén, mint amilyen a helymeghatározás és a térképezés. Bizonyos esetekben azonban a statisztikai technikák túlságosan körülményesek, és néha a gyakorlatban az egyszerűbb megoldások is éppen olyan hatékonyak. A legmegfelelőbb alkalmazandó módszer kiválasztását leginkább a valódi robotokkal történő munka során szerzett tapasztalatok segítik.

4. Mozgástervezés

A robotikában a döntések végül is a beavatkozó szervek mozgásában jutnak kifejezésre. A **ponttól pontig mozgás** (**point-to-point motion**) esetén a cél a robotnak vagy végbeavatkozó szervnek egy megadott helyre való eljuttatása. Nagyobb kihívást jelent a **követő mozgás** (**compliant motion**), amikor is a robot úgy mozog, hogy közben végig fizikai kontaktusban marad egy akadállyal. Példa erre, amikor egy robotkar becsavar egy égőt, vagy amikor végigtol az asztalon egy dobozt.

Azzal kell kezdenünk, hogy keresünk egy megfelelő reprezentációt, amely alkalmas a mozgástervezési problémák leírására és megoldására. Kiderül, hogy a **konfigurációs tér** (**configuration space**) – a robotállapotok tere, amelyet a pozíció, az orientáció és a csuklószögek definiálnak – alkalmasabb a feladatra, mint a 3D tér. A **pályatervezés** (**path planning**) azt jelenti, hogy meg kell találni az egyik konfigurációtól egy másikig vezető utat. A korábbi fejezetekben már számos úttervezési módszert vázoltunk fel. A robotikában az úttervezés legfőbb jellemvonása az, hogy *folytonos* terekkel dolgozik. A szakirodalom számos különböző technikát tart számon, amelyek kifejezetten olyan esetekre alkalmazhatók, ahol a cél útvonalak keresése sokdimenziós folytonos terekben. Két fő megközelítés létezik: a **celladekompozíció** (**cell decompisition**) és a **szkeletonizáció** (**skeletonisation**). Mindkettő diszkrét gráf keresési feladatra vezeti vissza és egyszerűsíti le a folytonos útkeresési problémát azáltal, hogy kanonikus állapotokat és pályákat ír le a folytonos térben. Ebben a fejezetben végig feltételezzük, hogy a mozgás determinisztikus, és a robot helybéli azonosítása egzakt. A későbbi fejezetek gyengíteni fogják ezeket a feltételezéseket.

4.1. Konfigurációs tér

A robotmozgási feladatok megoldásának első lépéseként a problémához ki kell találni egy megfelelő reprezentációt. Kezdjük egy egyszerű probléma egyszerű ábrázolásával. Vegyük a 25.12. (a) ábrán látható robotkart. Két csuklóval rendelkezik, amelyek egymástól függetlenül mozoghatnak. Ha a csuklókat mozgatjuk, megváltozik a könyök és a megfogó (x, y) koordinátája. (A kar z irányban nem képes elmozdulni.) Mindez azt sugallja, hogy a robot konfigurációi leírhatók egy négydimenziós koordináta-rendszerben, ahol: $(x_{\text{könyök}}, y_{\text{könyök}})$ jelenti a könyök relatív helyzetét a környezethez képest, míg $(x_{\text{megfogó}}, y_{\text{megfogó}})$ a megfogó helyzetét. Világos, hogy ez a négy koordináta leírja a robot összes lehetséges állapotát. Ezt a leírást **munkatér-reprezentáció**nak

Created by XMLmind XSL-FO Converter.

(**workspace representaion**) is nevezik, mivel a robot helyzetét ugyanabban a koordináta-rendszerben írjuk le, mint a mozgatni (vagy éppen elkerülni) kívánt tárgyakét. A munkatér-reprezentáció jól használható ütközés elkerülésre, különösen akkor, ha a tárgyakat egyszerű poligonokként modellezzük.

A munkatér-reprezentációval az a probléma, hogy a robot számára nem minden munkatér-koordináta érhető el, még akkor sem, ha nincsenek akadályok. Ez a munkatér elérhető koordinátáira vonatkozó **összekapcsolási kényszer**ek (**linkage constrain**s) miatt van. Például a könyök ($x_{\text{könyök}}$, $y_{\text{könyök}}$) és a megfogó ($x_{\text{megfogó}}$, $y_{\text{megfogó}}$) pozíciója egymástól mindig adott távolságra van, mert egy merev alkar kapcsolja őket össze. Ha a robotmozgás-tervezés munkatér-koordinátákban történik, akkor a generált útvonalaknak mindig meg kell felelniük ezeknek a kényszereknek. Ez azért különösen nehéz, mert az állapottér folytonos, a kényszerek pedig nemlineárisak.

Könnyebb a dolgunk, ha **konfigurációs tér**ben (**configuration space**) tervezünk. Ahelyett hogy Descartes-koordinátákkal írnánk le a robot helyzetét a térben, a csuklók konfigurációját vesszük figyelembe. A példában szereplő robotnak két csuklója van, így állapotát a két csukló állásával jellemezhetjük: $\phi_{\text{váll}}$ jelenti a vállcsukló szögét, míg $\phi_{\text{könyök}}$ a könyökcsuklóét. Ha semmilyen akadály sincs a munkatérben, akkor a robot csuklói bármilyen értéket szabadon felvehetnek a konfigurációs térben. Sőt pályatervezés során a kiindulási és a célkonfiguráció egyszerűen összeköthető egy egyenessel. A robot ezt az egyenest követve konstans sebességgel mozgatja csuklóit, amíg el nem éri a kijelölt helyet.

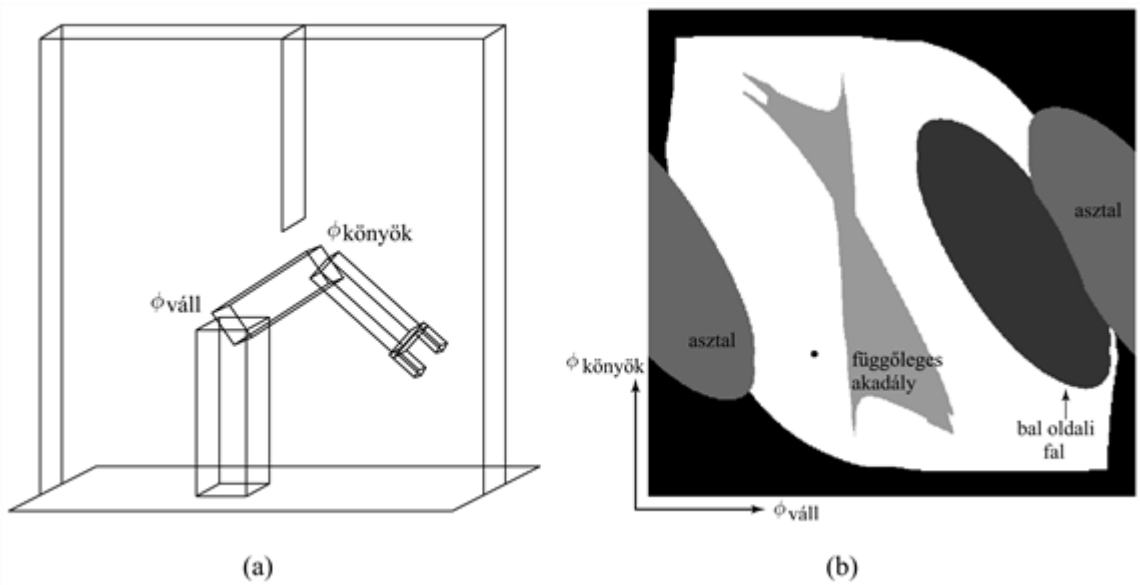
Sajnálatos módon azonban a konfigurációs térnek is megvannak a maga problémái. A robot feladatai általában a munkatér koordinátáiban adottak, és nem a konfigurációs térében. Konkrétan például azt akarjuk, hogy a robot végberendezését a munkatérben egy adott pontra mozgassa, sőt esetleg még az orientáció is adott. Ez felveti azt a kérdést, hogy hogyan képezzük le a munkatér-koordinátákat a konfigurációs térbe. Általában ennek a problémának az inverze egyszerűbb, azaz könnyebben képezhetők le a konfigurációs tér koordinátái a munkatérbe, hiszen a probléma néhány egyértelmű koordináta-transzformációval megoldható. Ezek a transzformációk lineárisak transzlációs, és trigonometrikusak rotációs csuklók esetén. A transzformációk sorozatát **kinematiká**nak (**kinematics**) hívják. Ezzel a kifejezéssel már találkoztunk korábban, a mobil robotoknál.

Inverz kinematikának (**inverse kinematics**) hívják egy olyan robot konfigurációjának kiszámítását, ahol a robot beavatkozó szerveinek helyzete a munkatér koordinátáiban adott. Az inverz kinematikai számítások általában nehezek, különösen sok szabadságfokú robotok esetében. Sőt a megoldás ritkán egyértelmű. Példarobotunk esetében két különböző konfiguráció is létezik, amikor a megfogó ugyanazt a helyzetet veszi fel a munkatérben, mint ami a 25.12. ábrán látható.

Általában két szabadságfokú (2DOF) robotkar esetén az inverz kinematikai feladat megoldásainak száma nulla és kettő között van, bármilyen bemeneti munkatér-koordinátára. A legtöbb ipari robot esetén végtelen sok megoldás van. Ennek megértéséhez képzeljük el, hogy a példarobotunkhoz hozzáadunk még egy rotációs csuklót, aminek tengelye párhuzamos a meglévő csuklók egyikével. Így a legtöbb konfiguráció esetén szabadon tudjuk mozgatni a közbülső csuklókat, miközben a megfogó pozíciója nem változik (az orientációja azonban igen). Néhány további csuklóval (mennyivel?) megoldható, hogy a kart ugyanígy mozgatva az orientáció se változzon. Már láttunk erre példát, amikor letettük kezünket az asztalra, és mégis tudtuk mozgatni a könyökünket. A kéz pozíciójából adódó kinematikai korlátozások alapján még nem határozható meg egyértelműen az emberi könyök konfigurációja. Más szavakkal, a váll-kar együttes inverz kinematikai problémájára végtelen sok megoldás létezik.

25.12. ábra - (a) Egy két szabadságfokú robotkar munkatere. A munkatér egy doboz, amiben egy lapos akadály lóg a mennyezetről. (b) Ugyanennek a robotnak a konfigurációs tere. A fehérrel jelzett terület jelenti azokat a konfigurációkat, ahol a robot nem ütközik össze semmivel. A pont az ábrán a robot bal oldali konfigurációját mutatja.

Created by XMLmind XSL-FO Converter.



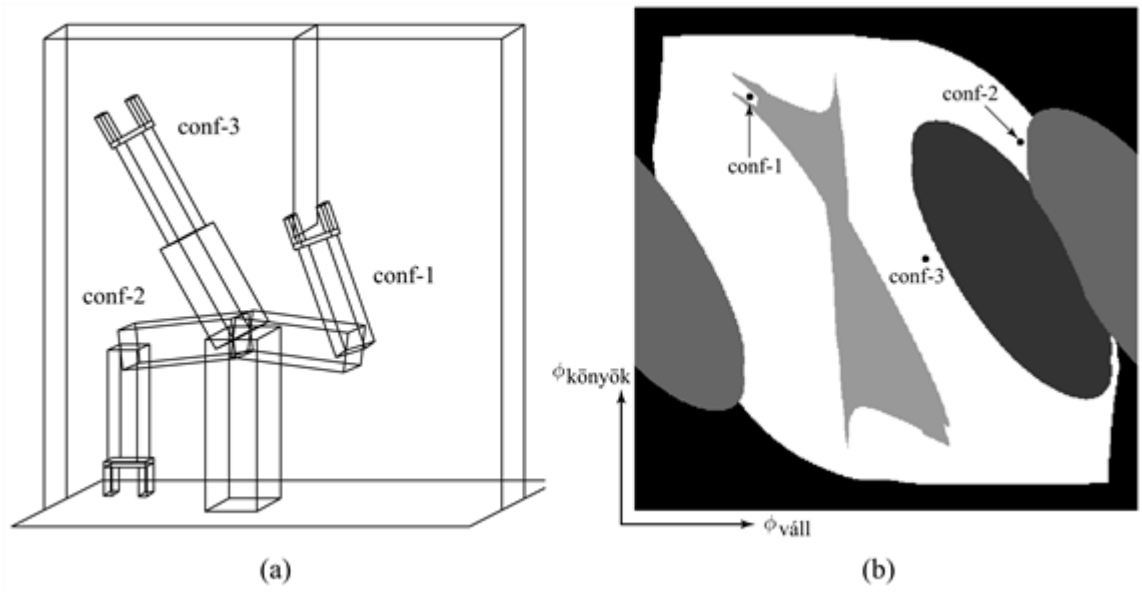
A másik probléma a konfigurációs térrel kapcsolatosan akkor merül fel, amikor akadályok is vannak a munkatérben. A 25.12. (a) ábrán látható esetben például számos ilyen akadály van; az egyik pont a robot munkaterének közepébe nyúlik bele. A munkatérben ezek az akadályok egyszerű geometriai formákként jelennek meg, különösen a robotikai tankönyvekben, ahol a poligonális akadályokra fókuszálnak. De hogyan néznek ki ezek a konfigurációs térben?

Példarobotunk konfigurációs terét mutatja a 25.12. (b) ábra a 25.12. (a) ábrán látható akadályok esetére. A konfigurációs teret két altérre bonthatjuk: azoknak a konfigurációknak a terére, amelyek elérhetők – ez a **szabad tér** (**free space**), illetve az elérhetetlen konfigurációk által alkotott térrészre, amit **foglalt tér**nek (**occupied space**) hívnak. A 25.12. (b) ábrán a fehér rész jelenti a szabad teret, míg az összes többi a foglalt tér. A különböző árnyalatok jelzik, hogy melyik akadályról is van szó: a szabad teret körbehatároló fekete részek jelentik azokat a konfigurációkat, amelyeknél a robot önmagával ütközik össze. Könnyű belátni, hogy a váll- és könyökcsuklók szélső állásai ezt eredményezik. A robot két oldalán lévő ovális részek az asztalnak felelnek meg, amin a robot áll. A harmadik ovális terület a bal oldali fal. Végezetül, a konfigurációs térben a legkülönösebb alakzat a robot mozgásterébe belelógó, egyszerű függőleges elem. Ennek igen különös alakja van, erősen nemlineáris, és helyenként még konkáv is. Egy kis képzelőerővel felismerhetjük a megfogó alakját a bal felső szélén. Szakítsunk egy kis időt ennek a fontos ábrának a tanulmányozására. Az alakzat egyáltalán nem nyilvánvaló! A 25.12. (b) ábrán a pont a robot 25.12. (a) ábrán lévő állapotának megfelelő konfigurációt mutatja. A 25.13. ábrán további három konfigurációt is láthatunk, mind a munkatérben, mind a konfigurációs térben. A „conf-1” állásban a megfogó megközelíti a középső akadályt.

A legtöbb esetben a szabad tér alakja nagyon bonyolult a konfigurációs térben, még akkor is, ha a munkateret egyszerű poligonokkal reprezentáljuk. Éppen ezért a gyakorlatban inkább *feltérképezik* a konfigurációs teret, semmint pontosan kiszámítják. A térképezés során generálnak egy konfigurációt, és megnézik, hogy az a szabad térben van-e. Ehhez alkalmazzák a robot kinematikáját, és annak alapján már látszik, ha az adott konfiguráció ütközéssel jár a munkatérben.

25.13. ábra - Három különböző robotkonfiguráció a munkatérben és a konfigurációs térben

Created by XMLmind XSL-FO Converter.



4.2. Celladekompozíciós módszerek

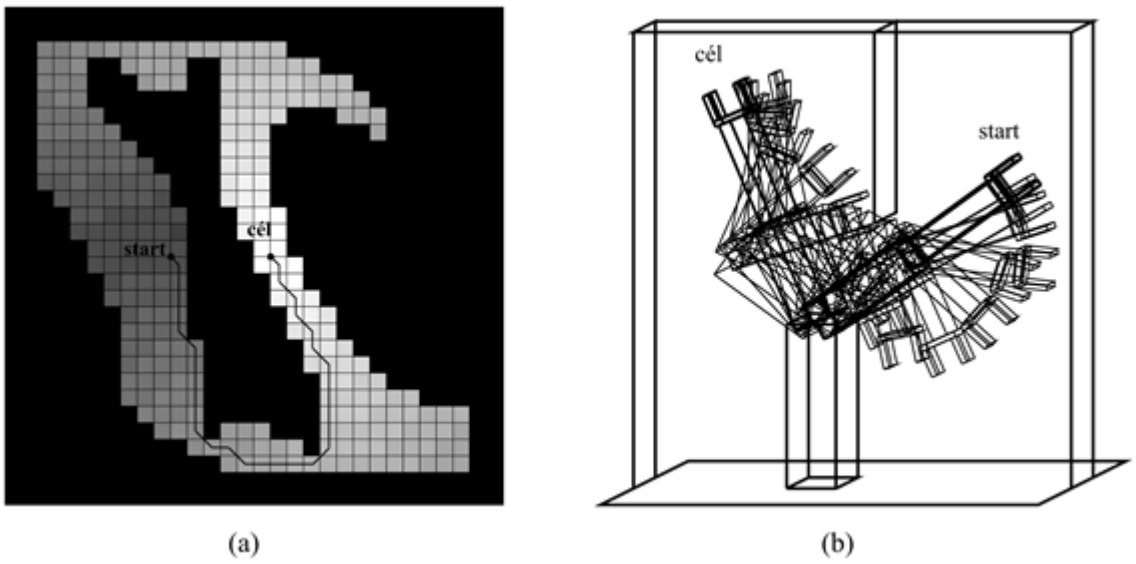
Első megközelítésben **celladekompozíció**t (**cell decompisition**) használunk pályatervezéshez, ami azt jelenti, hogy a szabad teret véges számú folytonos régióra, cellákra osztjuk fel. Fontos tulajdonsága ezeknek a régióknak, hogy a pályatervezés egyszerű módszerekkel megoldható egy-egy cellán belül (pl. egyenes vonalú mozgással). Ezután a pályakeresés diszkrét gráfkeresési problémává egyszerűsödik, nagyon hasonlóvá ahhoz, amit a 3. fejezetben már bemutattunk.

A legegyszerűbb celladekompozíció egy szabályos osztástávolságú rácsból áll. Ilyen négyzethálós dekompozíciót mutat be a 25.14. (a) ábra, az ehhez a sűrűségű felosztáshoz tartozó optimális útvonallal. Különböző árnyalatokat használtunk a szabad tér egyes celláihoz tartozó *értékek* jelölésére. Ez az érték mutathatja például, hogy milyen messze van az adott cella a céltól. (Kiszámításához használhatjuk a 17.4. ábrán ismertetett ÉRTÉKITERÁCIÓ algoritmus egy determinisztikus változatát.) A 25.14. (b) ábrán látható a kar mozgása a munkatérben.

Az ilyen felbontásnak a nagyon könnyű megvalósíthatóság az előnye. Ugyanakkor, használatakor két korlátozás is él. Először is, csak kis dimenziójú munkaterek esetén működik, mert a cellák száma *d*-vel, a dimenziók számával exponenciálisan nő. Másodszor, felmerül a kérdés, hogy mit csináljuk azokkal a cellákkal, amelyek vegyesek, vagyis részben a szabad térhez, részben a foglalthoz tartoznak. Egy ilyen cellán keresztül menő megoldás nem feltétlenül működik, mert lehet, hogy az adott irányban nem lehet áthaladni a cellán. Ez az úttervezőt *bizonytalanná* tenné. Ugyanakkor, ha egyáltalán nem engedjük meg, hogy a pálya ilyen „vegyes” cellákon menjen keresztül, tervező algoritmusunk *hiányos* lesz. Előfordulhat, hogy az egyetlen lehetséges út a célhoz egy ilyen cellán megy át – különösen akkor, ha a cella mérete összehasonlítható a térközök és átjárók fizikai méretével.

25.14. ábra - (a) A költségfüggvény és az előállított trajektória a konfigurációs térben, diszkrét hálós celladekompozíciós közelítés esetén. (b) Ugyanaz az útvonal a munkatérben ábrázolva. Érdemes megfigyelni, hogy a robot behajlítja a könyökét, hogy elkerülje a függőleges akadályt.

Created by XMLmind XSL-FO Converter.

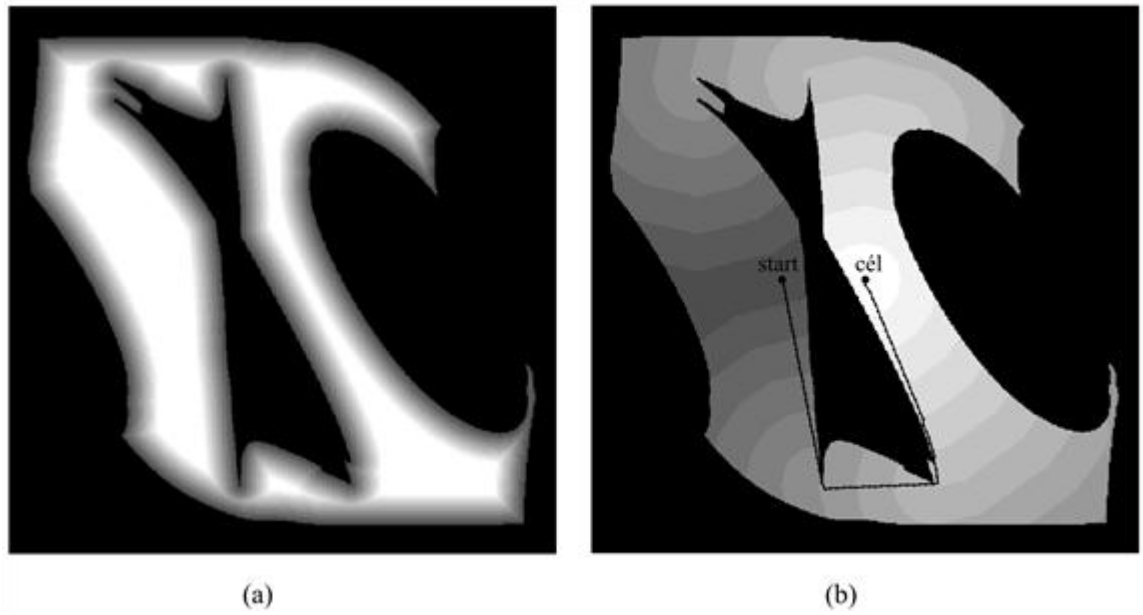


A fenti problémák elkerülése érdekében a celladekompozíciós módszeren kétféleképpen lehet javítani. Az első, ha megengedjük a vegyes cellák *tovább osztását* – mondjuk az eredeti méret felére. Ezt rekurzívan folytathatjuk, amíg nem találunk olyan utat, ami csak szabad térben lévő cellákon megy át. (Természetesen a módszer csak akkor működik, ha egy adott celláról egyértelműen el tudjuk dönteni, hogy vegyes-e. Ez viszonylag könnyű, ha a konfigurációs tér határai matematikailag egyszerűen leírhatók.) Akkor teljes a módszerünk, ha létezik korlát a legkeskenyebb átjáróra, amin az útvonal bejárható. Annak ellenére, hogy a konfigurációs téren belül alapvetően a bonyolultabb, trükkösebb részekre fókuszálódik a számítási igény, még mindig nem leszünk képesek sokdimenziós problémák kezelésére, mivel a cellák minden rekurzív újraosztása 2^d kisebb cellát hoz létre. A másik megoldás során azzal tesszük teljessé az algoritmusunkat, hogy ragaszkodunk a szabad tér egy **egzakt celladekompozíció**jához (**exact cell decomposition**). Ez a módszer megengedi, hogy a cellák szabálytalan alakúak legyenek, de kizárólag a szabad tér határán. Az alakzatoknak viszonylag egyszerűeknek kell maradniuk, hogy könnyen meg lehessen határozni egy szabad cella átlóját. Ez a technika bizonyos magasabb szintű geometriai ismereteket igényel, ezért részletesebben nem tárgyaljuk.

Ha megvizsgáljuk azt a pályát, amit a 25.14. ábrán kaptunk, láthatjuk, hogy még további nehézségeket kell leküzdenünk. Először is, vegyük észre, hogy a pálya mentén nagyon éles kanyarok vannak. A robot bármilyen véges sebességgel mozog, nem tudja ezeket „bevenni”. Másodszor, az út helyenként nagyon közel halad el az akadályok mellett. Mindenki, aki már vezetett autót, tudja, hogy egy olyan parkolóhely, ahol csak egy-egy milliméter szabad hely van mindkét oldalon, valójában nem is parkolóhely. Éppen ezért olyan megoldásokra törekszünk, amelyek robusztusak a kis mozgási hibákra nézve.

25.15. ábra - (a) A taszító potenciáltér távol tartja a robotot az akadályoktól. (b) A megoldásként kapott pálya, ha egyszerre minimalizáljuk az úthosszt és a potenciálfüggvényt.

Created by XMLmind XSL-FO Converter.



Szeretnénk maximalizálni az útvonal távolságát az akadályoktól, ugyanakkor minimalizálni annak hosszát. Mindezt a **potenciáltér** (**potential field**) módszer bevezetésével érhetjük el. A potenciáltér az állapottérben definiált függvény, amely nagy értéket vesz fel bármely akadály közelében, és attól távolodva egyre csökken. Ilyen potenciáltér látható a 25.15. (a) ábrán: minél sötétebbre van színezve a konfigurációs tér, annál közelebb van egy akadályhoz. Pályatervezéshez használva a potenciáltér még egy költségfüggvényt jelent az optimalizálás során. Ez érdekes kompromisszumot rejt magában. Egyfelől a robot igyekszik lerövidíteni az út hosszát a célig, másfelől próbál minél távolabb maradni az akadályoktól, azáltal hogy minimalizálja a potenciálfüggvényt. A két szempontot megfelelően súlyozva az eredmény a 25.15. (b) ábrán látható pályához hasonló lesz. Az ábra az egyesített költségfüggvényből értékiterációval számított értékfüggvényt is mutatja. Világos, hogy az így kapott megoldási útvonal hosszabb, viszont biztonságosabb.

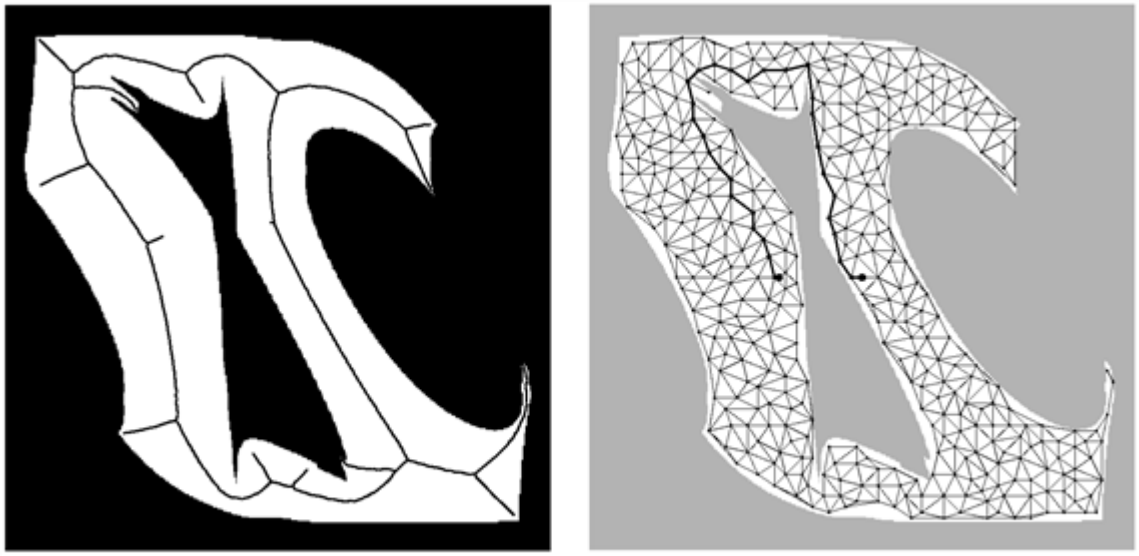
4.3. Szkeletonizációs módszerek

Az útkereső algoritmusok másik fő csoportja a **szkeletonizálás** (**skeletonization**) elvén alapul. Ezek a módszerek a robot szabad terének egydimenziós leképezésével dolgoznak, amely alapján a tervezési probléma már könnyebben megoldható. Ezt az alacsonyabb dimenziójú reprezentációt hívják a konfigurációs tér **váz**ának (**skeleton**).

A 25.16. ábrán a szkeletonizációra látható egy példa: a szabad tér **Voronoi-gráf**ja (**Voronoi graph**) azokból a pontokból áll, amelyek kettő vagy több akadálytól egyenlő távolságra vannak. Ahhoz, hogy a pályatervezéshez a Voronoi-gráfot lehessen használni, a robotnak először is meg kell változtatnia aktuális konfigurációját úgy, hogy rajta legyen a gráfon. Könnyen belátható, hogy a konfigurációs térben ez mindig megoldható egyenes vonalú mozgással. Ezután a robot a gráf mentén halad mindaddig, amíg el nem éri a célkonfigurációhoz legközelebb eső pontot. Ekkor elhagyja a Voronoi-gráfot, és elmegy a célig. A konfigurációs térben ez egyenes vonalú mozgással ismét megvalósítható.

25.16. ábra - (a) A Voronoi-diagram azon pontok halmaza a konfigurációs térben, amelyek egyenlő távolságra vannak két vagy több akadálytól. (b) A szabad tér 400 véletlenszerűen kiválasztott pontjából felépített valószínűségi úthálózat.

Created by XMLmind XSL-FO Converter.



(a) (b)

Ilyen módon az eredeti útkeresési feladatot a Voronoi-gráfon lévő útvonal megtalálására redukáltuk. A Voronoi-gráf (néhány különleges esetet leszámítva) egydimenziós, és véges sok olyan pontja van, ahol három vagy több görbe metszi egymást. Így azon a legrövidebb út megtalálása egy diszkrét gráfkeresési probléma, amilyet a 3. és 4. fejezetekben már tárgyaltunk. Ha a Voronoi-gráf mentén haladunk, akkor nem a legrövidebb utat kapjuk, de az egyik legbiztonságosabbat. A hátránya ennek a módszernek, hogy nehéz többdimenziós konfigurációs terekre alkalmazni, és sokszor túl nagy kitérőket tesz, főleg ha a szabad tér nagyon tág. Ezenfelül a Voronoi-diagram számítása bonyolult lehet a konfigurációs térben, különösen ahol az akadályok alakja nagyon összetett.

Alternatívát jelent a Voronoi-diagramokkal szemben a **valószínűségi úthálózat** (**probabilistic roadmap**) módszere. Ez a szkeletonizációs technika több lehetséges útvonallal dolgozik, így jobb megoldást jelent nagy, nyitott terek esetén. A valószínűségi úthálózatra látható példa a 25.16. (b) ábrán. A gráfot véletlenszerűen generált, a szabad térbe eső, nagyszámú konfiguráció alkotja. Ezek után két csomópontot összekötünk egy éllel, amennyiben „könnyű” eljutni egyikből a másikba. Alapvetően egyenes élekkel dolgozunk, és azok természetesen csakis a szabad térben futhatnak. Eredményül egy véletlenszerű gráfot kapunk a robot szabad terében. Ha ehhez még hozzávesszük a kezdeti és a célkonfigurációt, az útkeresés diszkrét gráfkeresési problémává válik. Elméletileg a módszer nem teljes, mivel rosszul felvett ponthalmazzal lehet, hogy soha nem találunk megoldást. A felvett pontok száma és a konfigurációs tér alakja alapján korlátozhatjuk a kudarc valószínűségét. Azt is megtehetjük, hogy főként azon a területen generálunk pontokat, ahol egy előzetes, mindkét végpontból lefuttatott részleges keresés alapján sejtjük, hogy a helyes út visz majd. Ezekkel a kiegészítésekkel a valószínűségi úthálózat módszere jobban használható más technikáknál magasabb dimenziójú konfigurációs terek esetében is.

5. Bizonytalan mozgások tervezése

Az eddig tárgyalt pályatervezési algoritmusok közül egyik sem foglalkozott a robotika egyik központi problémájával: a *bizonytalansággal*. A robotikában a bizonytalanság a környezet részleges megfigyelhetőségéből és a robot mozgásának sztochasztikus (vagy nem modellezett) összetevőiből ered. Hibát okozhat az is, ha valamilyen közelítéses módszert, például részecskeszűrést használunk. Ilyen esetekben a robotnak nincs pontos információja a helyzetéről, még akkor sem, ha a környezet sztochasztikus vonásait tökéletesen modelleztük.

A legtöbb mai robot a döntéshozatalhoz determinisztikus algoritmusokat használ, mint például az eddig tárgyalt útkereső algoritmusokat. Gyakori megoldás, hogy a helymeghatározó algoritmus által előállított állapoteloszlásból a **legvalószínűbb állapot**ot (**most likely state**) választják ki. Ennek a módszernek az előnye a számítások során mutatkozik meg. A pályatervezés már a konfigurációs térben is kihívás, és még nehezebb lenne, ha az állapotok teljes valószínűség-eloszlásával kellene dolgoznunk. Csak akkor tehetjük meg, hogy nem veszünk tudomást a bizonytalansági tényezőről, ha az kellően kicsi.

Created by XMLmind XSL-FO Converter.

Sajnálatos módon azonban nem hagyhatjuk mindig figyelmen kívül a bizonytalanságot. Egyes problémák esetén egyszerűen túlságosan nagy! Hogyan is használhatnánk determinisztikus pályatervezést egy mobil robot irányításához, ha fogalma sincs a robotnak arról, hogy hol van? Általában, ha a robot valódi állapota nem egyezik meg a maximális valószínűségi szabály által számítottal, akkor az irányító algoritmus nem lesz optimális. Az eltérés nagyságától függően ez egy sor nem kívánatos eseményt idézhet elő, például ütközést más tárgyakkal.

A robotikában számos technikát dolgoztak ki a bizonytalanság kezelésére. Néhányuk a 17. fejezetben tárgyalt bizonytalansági döntéshozó algoritmusokból ered. Ha a robotnak csak az állapotátmenetek kapcsán kell bizonytalansággal számolnia, máskülönben állapota teljesen megfigyelhető, akkor a problémát legjobban a **Markov döntési folyamat**tal (**MDF**) lehet modellezni. Az MDF által adott megoldás optimális **eljárásmód** (**policy**), amely minden lehetséges állapotban megmondja a robotnak, hogy mit csináljon. Ezzel a módszerrel képes bármiféle mozgási hiba kezelésére, ahol a determinisztikus úttervező egyetlen megoldása sokkal kevésbé lenne robusztus. Robotikában a stratégiákat általában **navigációs függvény**eknek (**navigation function**s) hívják. A 25.14. (a) ábrán bemutatott értékfüggvény könnyen átalakítható ilyen navigációs függvénnyé, ha egyszerűen követjük a gradienst.

Csakúgy, mint a 17. fejezetben, a részleges megfigyelhetőség itt is tovább nehezíti a problémát. Az így keletkező robotirányítási problémát a **részlegesen megfigyelhető MDF**-fel (**RMMDF**) modellezzük. Ilyen esetekben a robotnak általában van egy belső képe az állapotáról, ahogy azt már a 25.3. alfejezetben megtárgyaltuk. Az RMMDF-re a megoldás a robot belső állapotán definiált stratégia. Másképp megfogalmazva, a stratégia bemenete egy teljes valószínűség-eloszlás. Ez a robot számára lehetővé teszi, hogy ne csak az alapján tudjon döntést hozni, amit tud, hanem azt is felhasználhassa, amit nem tud. Például ha nem ismeri egy kritikus állapotváltozó értékét, akkor ésszerűen **információgyűjtés**t (**information gathering**) fog végrehajtani. Az MDF keretein belül erre van lehetőség, mivel az MDF teljes megfigyelhetőséget feltételez. Sajnálatos módon azok a technikák, amelyek az RMMDF-eket pontosan megoldják, nem alkalmazhatók a robotikában, minthogy nem ismeretes folytonos idejű változatuk. A diszkretizálás általában olyan hatalmas méterű RMMDF-eket eredményez, amelyek messze meghaladják a mai eszközök feldolgozási kapacitását. Jelen esetben a legtöbb, amit tehetünk, hogy megpróbáljuk a helyzet bizonytalanságát minimumra csökkenteni. A **tárgy menti navigálás** (**coastal navigation**) heurisztikus módszerével például jól csökkenthető a pozíció ismeretének bizonytalansága, ha a robot végig ismert referenciapontok közelében halad. Ugyanakkor fokozatosan csökkenti más, közeli referenciapontok helyzetének bizonytalanságát is, ami aztán lehetővé teszi a robot számára, hogy újabb területeket térképezzen fel.

5.1. Robusztus módszerek

Valószínűségi módszerek helyett kezelhetjük a bizonytalanságot úgynevezett **robusztus** (**robust**) módszerekkel is. A robusztus módszer csak *korlátozott* mértékű bizonytalanságot enged meg a probléma minden egyes paraméterével szemben, ugyanakkor nem társít valószínűségeket az egyes értékekhez a megengedett határon belül. A robusztus megoldás a bemeneti értékektől függetlenül megfelelően működik mindaddig, amíg azok nem lépnek át egy határt. A robusztus módszerek egy extrém formája a 12. fejezetben tárgyalt **alkalmazkodó tervezés** (**conformant planning**), ami úgy alakít ki terveket, hogy egyáltalán nincs információja az állapotról.

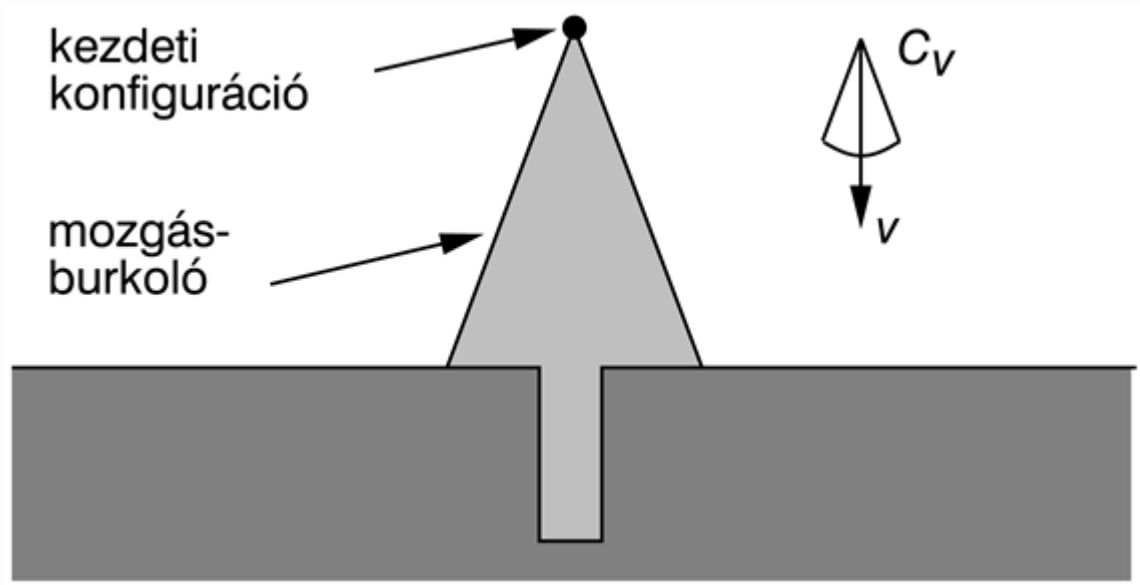
Itt egy olyan robusztus technikát ismertetünk, amelyet összeszerelési feladatoknál **finom mozgások tervezésé**re (**FMT**; **fine motion planning**) használnak. FMT-re például akkor van szükség, amikor egy robotkart egy nagy, merev tárgyhoz nagyon közel kell mozgatni. A fő probléma az FMT-vel az, hogy a környezet kisméretű, lényeges elemeivel kell dolgozni, nagyon apró mozdulatokkal. Ilyen kis lépték esetén a robot már képtelen pontosan mérni vagy irányítani pozícióját, sőt adott esetben még a környezet alakjáról sincs pontos képe. Feltételezzük, hogy ezek a bizonytalanságok mind korlátosak. Az FMT-problémákra a megoldást legtöbbször feltételes tervek és irányelvek kidolgozása jelenti, amelyek a végrehajtás közben visszacsatolásként használják a szenzorok adatait, és a megadott bizonytalansági korlátokon belül minden esetben megbízhatóan működnek.

Az FMT-k **felügyelt mozgás**ok (**guarded motion**) sorából tevődnek össze. Minden egyes felügyelt mozdulat (1) egy mozgásparancsból és egy (2) leállási feltételből áll, ami a robot szenzorainak adatain alapul. A folyamat igaz értékkel tér vissza, ha sikeresen befejeződött a felügyelt mozgás. A mozgásutasítások jellemzően engedékeny, azaz **önbeálló mozgás**ok (**compliant motion**), amelyek lehetővé teszik, hogy a robot kitérjen, ha az utasítás végrehajtása ütközést eredményezne. A 25.17. ábrán látható példa egy kétdimenziós konfigurációs teret mutat, középen egy szűk, függőleges lyukkal. Ez egy olyan feladat konfigurációs tere lehetne, amikor egy szögletes csapot kell beilleszteni egy nála alig szélesebb lyukba. A mozgásutasítások konstans sebességeket írnak elő. A leállási feltétel a felülettel történő érintkezés. A vezérlés bizonytalanságának modellezésénél feltételezzük, hogy a robot nem pontosan az utasításban megadott irányba mozdul el, hanem a körül egy C_v

Created by XMLmind XSL-FO Converter.

kúpon belül, tetszőlegesen. A 25.17. ábra azt mutatja, mi történne, ha adott sebességgel elindítanánk a műveletet az *s* kezdőpontból. A mozgás bizonytalansága miatt a robot a szürke területen belül bárhova megérkezhet a felülethez. Van rá esély, hogy beletalál a lyukba, de nagyobb valószínűséggel valamelyik oldalán áll meg. Mivel a robot nem fogja tudni, hogy a lyuk melyik oldalán van, ezért azt sem fogja tudni, hogy merre mozduljon.

25.17. ábra - Egy kétdimenziós környezet, a sebesség vektor bizonytalanságát ábrázoló kúppal. A vezérlés bizonytalansága miatt az eredeti szándéknak megfelelő *v* sebesség valójában bármilyen vektor lehet a C_v kúpon belül. Ennek eredményeképpen a robot végső konfigurációja bárhol lehet a burkolón (a világosabban árnyalt területen), azaz nem tudjuk, hogy eljutottunk-e a lyukba, vagy sem.



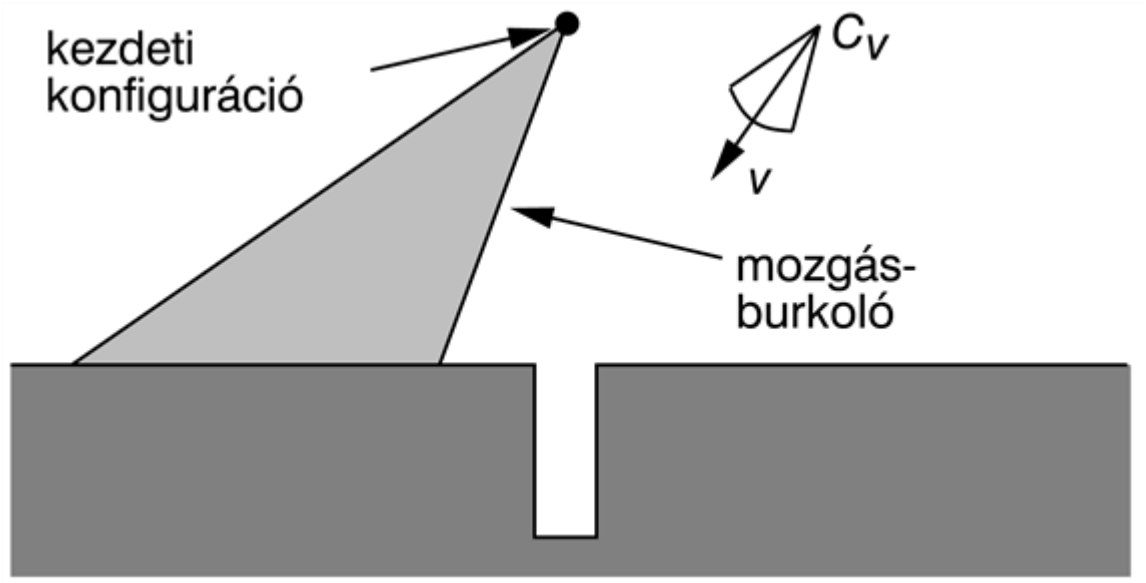
Egy kifinomultabb stratégia látható a 25.18. és a 25.19. ábrán. A 25.18. ábra esetében a robot szándékosan a lyuknak az egyik oldala felé mozog. Az ábrán látható a mozgásparancs és a leállás feltétele: valamely felülettel való érintkezés. A 25.19. ábrán olyan a mozgásparancs, hogy a robot a felület mentén mozdul el, míg bele nem talál a lyukba. Ez önbeálló mozgásvezérlés alkalmazását tételezi fel. Mivel a mozgás iránya (a bizonytalansággal együtt is) jobbra mutat, a robot mindig jobbra fog csúszni, valahányszor érintkezésbe kerül a vízszintes felülettel. Amikor eléri a lyukat, lecsúszik annak jobb oldali széle mentén, mivel a függőleges felülethez képest valamennyi lehetséges vektor lefelé mutat. Addig fog mozogni, amíg el nem éri a lyuk alját, mivel ez számára a leállási feltétel. A vezérlés bizonytalansága ellenére a robot valamennyi lehetséges pályája a lyuk aljával történő érintkezésnél ér véget. Mindenképpen ez történik, hacsak a felület szabálytalanságai miatt a robot valahol meg nem akad.

Láthatjuk, hogy a finom mozgások *tervezése* nem triviális; valójában sokkal nehezebb, mint a pontos mozgások tervezése. Vagy minden egyes mozgáshoz rögzített számú diszkrét értéket választunk, vagy a környezet geometriáját használjuk fel azon irányok megállapításához, amelyek minőségileg különböző viselkedést eredményeznek. A finommozgás-tervező a konfigurációs tér leírását, a sebességbizonytalansági kúp szögét és azt a specifikációt tekinti inputnak, amely megadja a lehetséges leállásifeltétel-érzékeléseket (ebben az esetben a felülettel való érintkezést). Az eredménynek egy olyan többlépéses feltételes tervnek vagy stratégiának kell lennie, ami garantálja a sikert, ha egyáltalán létezik ilyen terv.

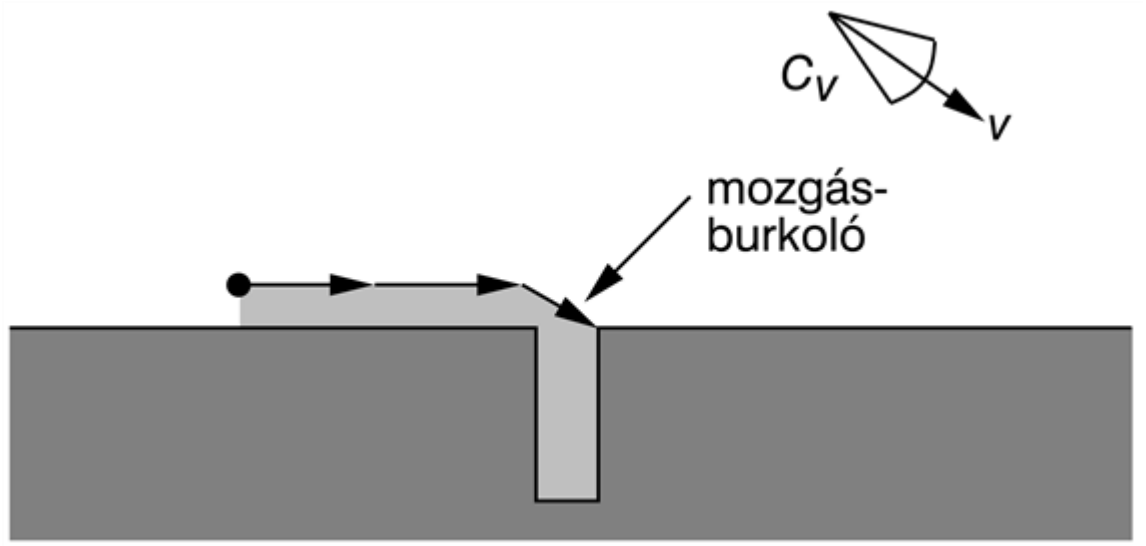
Példánkban feltételezzük, hogy a tervezőnek pontos képe van a környezetről, de korlátos hiba is bevezethető a modellbe a következők szerint: amennyiben a hibák paraméterekkel leírhatók, úgy azok a paraméterek szabadságfokok formájában hozzáadhatók a konfigurációs térhez.

25.18. ábra - Az első mozgásparancs és a lehetséges robotmozgások burkolója. A hibától függetlenül tudjuk, hogy a végső konfiguráció a lyuk bal oldalán lesz.

Created by XMLmind XSL-FO Converter.



25.19. ábra - A második mozgásparancs és a lehetséges mozgások. Még hibák esetén is végül bele fogunk találni a lyukba.



Az utolsó példában, amennyiben a lyuk mélysége és szélessége bizonytalan, úgy ezeket újabb két szabadságfok formájában hozzávehetjük a konfigurációs térhez. Az nem lehetséges, hogy a robotot ezekbe az irányokba elmozdítsuk a konfigurációs térben, és a robot pozícióját sem tudjuk közvetlenül érzékelni. Azonban mindkét korlátozást figyelembe vehetjük, amikor ezt egy FMT-problémaként írjuk le, megfelelően specifikálva a vezérlés és az érzékelés bizonytalanságait. Ez egy összetett, négydimenziós tervezési problémához vezet, azonban azonos tervezési technikák alkalmazhatók. Megjegyezzük, hogy a 17. fejezetben ismertetett döntéselméleti módszerektől eltérően ez a fajta megközelítés a legrosszabb esetre vonatkozó terveket eredményezi ahelyett, hogy a terv várható minőségét maximalizálná. A legrosszabb esetre vonatkozó tervek döntéselméleti értelemben csak akkor optimálisak, ha a végrehajtás közbeni kudarc sokkal költségesebb, mint bármely más, a végrehajtás során felmerülő költség.

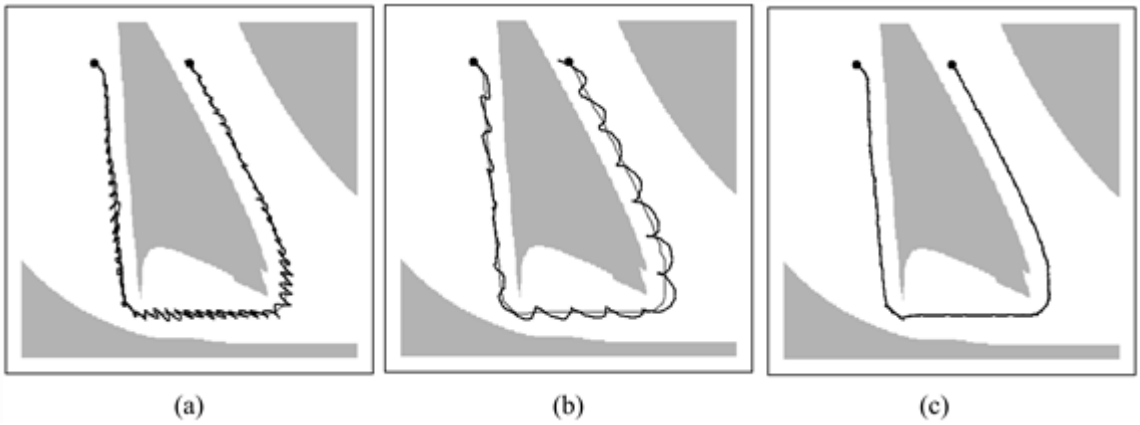
6. Mozgás

Beszéltünk már arról, hogyan *tervezzük* meg a mozgást, de arról nem, hogy hogyan *mozogjunk*. Az eddigi terveink – különösen azok, amelyeket determinisztikus pályatervezővel állítottunk elő – feltételezik, hogy a robot egyszerűen képes követni bármilyen megtervezett pályát. A valóságban természetesen nem ez a helyzet. A robotoknak van tehetetlenségük, ezért nem tudnak tetszőleges pályát követni, csak nagyon kis sebesség esetén.

Created by XMLmind XSL-FO Converter.

A legtöbbször erőt kell kifejtenie a robotnak, nem pedig meghatározott pozícióba eljutnia. Ez a fejezet azokról a módszerekről szól, amelyekkel ki lehet számítani ezeket az erőket.

25.20. ábra - Robotkarvezérlés (a) egységnyi erősítésű arányos szabályozóval; (b) 0,1-es erősítési tényezőjű arányos szabályozóval; (c) PD szabályozóval, 0,3-as erősítésű arányos taggal és 0,8-as differenciáló komponenssel. Mindhárom esetben a robotkar a szürkével jelölt pályát próbálta követni.



6.1. Dinamika és vezérlés

A 25.2. alfejezetben már megismerkedtünk a **dinamikus állapot** (**dynamic state**) fogalmával, amely a robot sebességmodelljével kiterjeszti a robot kinematikai állapotát. Például egy csukló szögének értéke mellett a dinamika magában foglalja a szög változási sebességet is. A dinamikai állapot reprezentálására szolgáló állapotátmenet-modell tartalmazza az erők erre a változási sebességre gyakorolt hatását. Ezek a modellek általában **differenciálegyenletek**et (**differential equations**) használnak, amelyek egy kinematikai mennyiséget (például egy kinematikai állapotot) viszonyítanak annak időbeli megváltozásához (például a sebességhez) is. Elvileg a mozgástervezéshez a kinematikai helyett dinamikai modellt is használhattunk volna. Ez a metodika magasabb szintű teljesítményt eredményezne a robotnál, ha egyáltalán meg tudnánk alkotni a terveket. Azonban a dinamikai állapot sokkal bonyolultabb a kinematikainál. A dimenziók magas száma miatt ezt a módszert csak a legegyszerűbb robotok esetében lehetne sikeresen alkalmazni. Éppen ezért a gyakorlatban leginkább az egyszerűbb kinematikai pályatervező algoritmusokra hagyatkoznak.

A kinematikai pályatervező algoritmusok hiányosságainak kompenzálására gyakran használnak különálló **szabályozó**kat (**controller**s), hogy pontosan a pályán tartsák a robotot. Szabályozók segítségével a környezetből vett visszacsatolással valós időben generálhatók a vezérlőjelek az irányítási célok eléréséhez. **Referenciaszabályozó**ról (**reference controller**) beszélünk, ha az a feladat, hogy a robot egy előre kitűzött pályán maradjon – ezt hívják **referenciapálya**nak (**reference path**). Azokat a szabályozókat, amelyek globális költségfüggvényeket optimalizálnak, **optimális szabályozó**knak (**optimal controller**s) hívjuk. Az optimális stratégiák az MDF-ek számára valójában optimális szabályozók.

Első ránézésre egyszerűnek tűnik az a feladat, hogy adott pályán tartsuk a robotot. A gyakorlatban azonban még ennek a látszólag könnyű feladatnak is vannak buktatói. A 25.20. (a) ábrán láthatjuk, hogy mik adódhatnak. A fekete vonal a robot által befutott út, ahogy próbálja a kinematikai pályát követni. Ha eltérés keletkezik – zaj vagy a robot mozgása során fellépő erőkre vonatkozó korlátozások miatt –, a robot ellenkező irányú, az eltéréssel arányos nagyságú erővel igyekszik azt kompenzálni. Elsőre ez a megoldás kézenfekvőnek tűnik, mivel az eltéréseket ellenerőkkel kompenzáljuk, így a robot a pályán marad. Ugyanakkor mindez azt eredményezi, hogy robotunk erőteljesen vibrálni kezd, ahogy az a 25.20. (a) ábrán is látszik. Ez a vibráció a robotkar természetes tehetetlenségéből következik: amikor a robot visszaér a referenciapálya adott pontjára, túllendül, ami a korábbihoz hasonló, de ellentétes előjelű hibát eredményez. Ahogy az ábrán is látható, ez a fajta túllendülés akár az egész trajektória mentén folytatódhat, és az eredményül kapott mozgás nem éppen ideális. Világos, hogy jobb szabályozásra van szükség.

Mielőtt rátérnénk a megfelelőbb szabályozásra, formálisan írjuk le a túllendülést okozó szabályozót. Azokat a szabályozókat, amelyek ellentétes irányú, arányos erővel reagálnak a megfigyelt hibára **P szabályozó**nak (**P controller**) hívjuk. A P betű *arányosat (proportional)* jelent: a pillanatnyi vezérlés mindig arányos a robotkar

Created by XMLmind XSL-FO Converter.

hibájával. Még formálisabban kifejezve: legyen $y(t)$ a referenciapálya idő (t) szerint paraméterezve. A P szabályozó által generált a_t vezérlőjel:

$$a_t = K_P(y(t) - x_t)$$

ahol X_t a robotkar t időpontbeli állapota. K_P a szabályozó úgynevezett **erősítési tényező**je (**gain parameter**), ami azt határozza meg, hogy milyen hevesen reagál a szabályozó az aktuális x_t állapot és a kívánt $y(t)$ között lévő eltérésre. A példánkban $K_P = 1$. Első pillantásra azt gondolhatja valaki, hogy ha K_P-t kis értékűre választjuk, akkor az megoldja a problémát. Sajnos nem így van. A 25.20. (b) ábrán látható $K_P = 0{,}1$ esetben a trajektória, amely továbbra is oszcillál. A kisebb erősítési tényező csak lelassította az oszcillációt, de egyáltalán nem oldotta meg a problémát. Ha elhanyagoljuk a súrlódást, a P szabályozó tulajdonképpen egy rugó, és a végtelenségig oszcillál a rögzített egyensúlyi pont körül.

Hagyományosan az ilyen problémákat az **irányításelmélet** (**control theory**) tárgykörébe sorolják. Ennek a tudományágnak egyre nagyobb a jelentősége a mesterséges intelligenciával foglalkozó kutatók számára. Évtizedek kutatásai során számos olyan szabályozót alkottak meg, amelyek lényegesen bonyolultabbak a fent használt egyszerű szabályozónál. Egy referenciaszabályozó **stabil** (**stable**), ha kis megváltozások csak korlátos hibát eredményeznek a referenciajelhez képest. **Szigorúan stabil** (**strictly stable**), ha a robot a kis megváltozások ellenére is vissza tud térni a referenciapályára. Látható, hogy a P szabályozónk stabil, de nem szigorúan stabil, mivel képtelen visszatérni a referenciatrajektóriára.

A legegyszerűbb szabályozó, ami eleget tesz a szigorú stabilitás követelményeinek a mi esetünkben a **PD szabályozó** (**PD controller**). A P betű továbbra is az arányosat jelöli, míg a D a *differenciálót*. A PD szabályozót az alábbi egyenlet írja le:

$$a_t = K_P\left(y(t) - x_t\right) + K_D \frac{\partial\left(y(t) - x_t\right)}{\partial t} \qquad (25.3)$$

Ahogy az már a képletből is látszik, a PD szabályozó egy differenciális taggal egészíti ki a P szabályozót, ami az a_t-t egy olyan taggal bővíti, ami arányos az $y(t) - x_t$ hiba idő szerinti első deriváltjával. Mi a hatása egy ilyen új tagnak? Általában a derivátor csillapítja a szabályozott rendszert. Tegyük fel, hogy egy adott rendszerben időben gyorsan változik az $(y(t) - x_t)$ hiba, mint ahogyan a fentebb tárgyalt esetben is. A differenciális tag ilyenkor az arányos ellenében hat, és összességében csökkenteni fogja a változásra történő reakciót. Ugyanakkor, ha a hiba állandósul, a differenciális tag háttérbe szorul, és az arányos tag fogja meghatározni a szabályozást.

PD szabályozót használva $K_P = 0{,}3$ és $K_D = 0{,}8$ paraméterekkel a 25.20. (c) ábrán lévő trajektóriát kapjuk. A pálya sokkal simább, és nem látható oszcilláció sem. Ahogy a példa is mutatja, egy differenciáló tag stabillá tehet egy szabályozót, amely enélkül nem az.

A gyakorlatban a PD szabályozónak is lehetnek hibái. Van úgy, hogy a PD szabályozó akkor sem tudja nullára redukálni a hibát, ha a külső eltérítés megszűnik. Ez a példánkból nem nyilvánvaló, de néha szükség van egy másik visszacsatolásra is az arányoson felül, hogy a hibát meg lehessen szüntetni. A megoldás erre a problémára egy harmadik tag hozzáadása a szabályozáshoz, ami a hiba idő szerinti integrálján alapul:

$$a_t = K_P\left(y(t) - x_t\right) + K_I \int\left(y(t) - x_t\right)dt + K_D \frac{\partial\left(y(t) - x_t\right)}{\partial t} \qquad (25.4)$$

K_I egy újabb erősítési tényező. A kifejezés kiszámítja a hiba idő szerinti integrálját. Ennek hatására a referenciajel és az aktuális állapot között hosszan fennálló eltérés megszűnik. Például ha x_t hosszabb ideig kisebb, mint $y(t)$, akkor az integrál addig fog nőni, amíg az a_t el nem tünteti a hibát. Az integrátor biztosítja, hogy a szabályozónak ne legyen rendszeres hibája, azon az áron, hogy nő az oszcilláció veszélye. **PID szabályozó**nak (**PID controller**) hívjuk azt a szabályozót, amelyik mindhárom tagot tartalmazza. Széles körben használják különböző ipari alkalmazásokhoz és irányítási feladatokhoz.

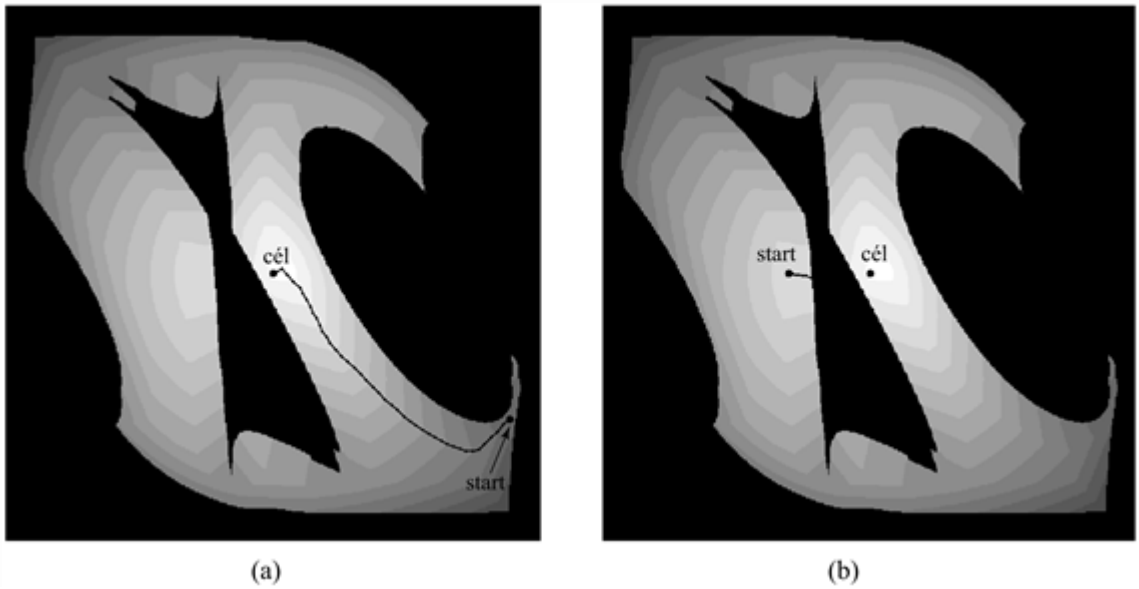
6.2. Potenciáltér-vezérlés

A potenciáltereket mint újabb költségfüggvényeket vezettük be a robotmozgás-tervezés során, de alkalmasak arra is, hogy közvetlenül generálják a robot mozgását, kiegészítve a pályatervezési fázist. Ahhoz, hogy ezt elérjük, definiálnunk kell egy vonzó erőt, ami a robotot a célkonfiguráció felé vonzza, valamint egy taszító potenciálteret, amely távol tartja az akadályoktól. Egy ilyen teret már bemutattunk a 25.21. ábrán. Az egyetlen

Created by XMLmind XSL-FO Converter.

globális minimumhelye a célkonfiguráció, értéke pedig az e céltól való távolság, valamint az akadályok közelségének összege. Az ábrán bemutatott potenciáltér kialakítását nem előzte meg tervezés. Éppen ezért a potenciálterek jól alkalmazhatók valós idejű megoldásokhoz. A 25.21. ábrán két trajektória látható, ahogy egy robot két különböző kezdeti konfigurációból megpróbálja megmászni a potenciálteret. Nagyon sok alkalmazás esetén a potenciálteret hatékonyan ki lehet számítani bármilyen adott konfiguráció esetében. Sőt adott robotkonfiguráció esetén a potenciálok kialakulásával a potenciálgradienst is meghatározhatjuk. Ezek a számítások általában különösen hatékonyak, főleg a pályatervező algoritmusokhoz hasonlítva, amelyek mind exponenciálisak a konfigurációs tér dimenziójában (a szabadságfokokban).

25.21. ábra - Potenciáltér-vezérlés. A robot egy potenciáltérben halad, amely az akadályoktól taszító erőkből és a célkonfigurációhoz vonzókból áll. (a) Sikeres útvonal. (b) Lokális optimum.



Az a tény, hogy a potenciálteres módszer ilyen hatékonyan megtalálja az utat a célhoz a konfigurációs térben még nagy távolságok esetén is, megkérdőjelezi a tervezés szükségességét. Vajon csak szerencsénk volt a példánk esetében, vagy tényleg elegendő a potenciálterek használata? A válasz az, hogy szerencsénk volt. A potenciáltereknek számos olyan lokális minimumhelye lehet, amely csapdába ejti a robotot. Ebben az esetben a robot úgy közelíti meg az akadályt, hogy csak vállcsuklóját mozgatja egészen addig, amíg bele nem ütközik az akadályba a rossz oldalon. A potenciáltér nem ad annyi információt, hogy a robot behajlítsa a könyökét, hogy átférjen az akadály alatt. Más szavakkal, a potenciáltéren alapuló technikák nagyon jók lokális robotirányításra, de globális tervezést igényelnek. A másik hátulütője ennek a módszernek az, hogy az erők, amiket generál, csak az akadályoktól és a céltól való távolságtól függnek, nem veszik figyelembe a robot sebességét. Így a potenciáltér-alapú vezérlés tényleg egy kinematikai módszer, és nem feltétlenül működik, ha a robot gyorsan mozog.

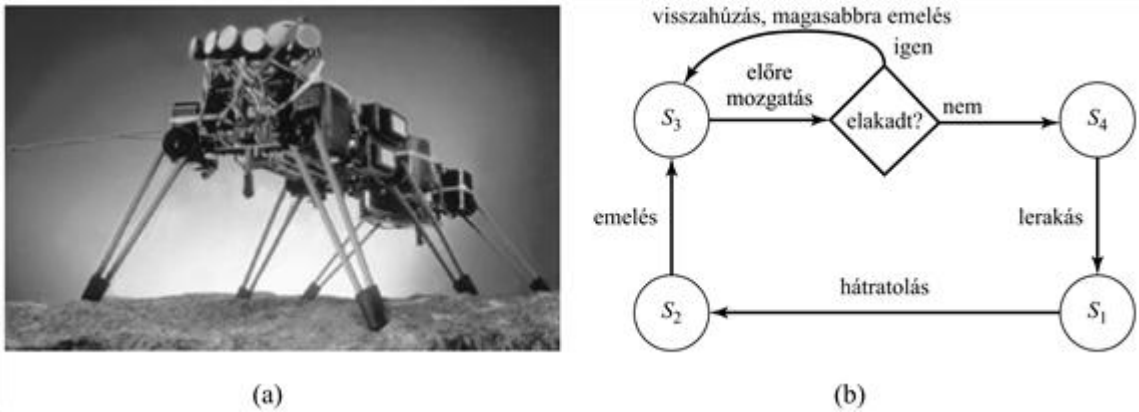
6.3. Reaktív irányítás

Eddig olyan irányítási módszerekkel foglalkoztunk, amelyek megkívánták a környezet bizonyos fokú ismeretét ahhoz, hogy akár referenciapályát, akár potenciálteret alkothassunk. Ezzel a megközelítéssel kapcsolatban azonban felmerülnek bizonyos problémák. Először is, sokszor nagyon nehéz pontos modellt alkotni a környezetről, különösen ha az komplex vagy nagyon távol van, mint mondjuk a Mars felszíne. Másodszor, még ha sikerül is pontos modellt készítenünk, a számítási nehézségek és a lokalizációs technikák hibája folytán alkalmazhatatlanná válnak ezek a módszerek. Bizonyos esetekben megfelelőbb a reflexszerű ágensek használata, ezt hívják **reaktív irányítás**nak (**reactive control**).

Erre példa a 25.22. (a) ábra **hexapod**ja, azaz hatlábú robotja, amelyet nehéz terepen való közlekedésre terveztek. A robot szenzorai többnyire alkalmatlanok arra, hogy megfelelő pontosságú modellt alkossanak a környezetről, hogy a robot a korábban tárgyalt pályatervezési algoritmusok bármelyikét használhassa. De hiába látnánk el megfelelő szenzorokkal, a tizenkét szabadságfok (kettő lábanként) túl nagy számítási kapacitást igényelne pályatervezés esetén.

Created by XMLmind XSL-FO Converter.

25.22. ábra - (a) Hexapod robot. (b) Kiterjesztett véges automata (KVA) (augmented finite state machine, AFSM) egyetlen láb vezérléséhez. Fontos, hogy ez a KVA-szenzoros visszacsatolásra reagál: ha a láb megakad az előremozgatás során, akkor magasabbra emeli.



Mindazonáltal a környezet explicit ismerete nélkül is lehetséges közvetlenül szabályozót tervezni a feladathoz. (Már láttuk ezt a PD szabályozó esetében, amely képes volt a komplex robotkart a megadott pályán tartani a robot dinamikájának pontos ismerete *nélkül*. Ugyanakkor szükség volt a kinematikai modell alapján megalkotott referenciapályára.) A hatlábú robot esetében megfelelő absztrakciós szinten meglepően könnyű meghatározni az irányítási törvényt. A megfelelő algoritmus ciklusosan mozgathatná a lábakat. Egy ideig a levegőben mozognak, utána pedig a földön támaszkodnak. A hat lábat úgy kell koordinálni, hogy egyszerre három (ellentétes oldalakon) mindig a földön legyen, biztosítva a robot fizikai stabilitását. Egy ilyen irányítási algoritmust könnyű programozni, és remekül működik sík felületen. Nehéz terepen az egyes akadályok meggátolhatják a lábakat az előremozdulásban. Ezen a problémán egy egészen egyszerű irányítási szabállyal segíthetünk: *ha a láb mozgását valami blokkolja, a robot egyszerűen húzza vissza, emelje magasabbra, és próbálja újra!* Az eredményül kapott vezérlést, mint véges automatát mutatja a 25.22. (b) ábra. Egy reflexszerű ágenshez állapotokat társít, ahol a belső állapot a gép aktuális állapotának indexe (s_1-től s_4-ig) jelöli.

Ennek az egyszerű visszacsatoláson alapuló szabályozónak a különböző változatai figyelemre méltóan robusztus lépegető vezérlést valósítanak meg, amellyel a robot göröngyös terepen is képes előrejutni. Világos, hogy ehhez a szabályozóhoz nem kell környezeti modell, és nem kell semmiféle keresést alkalmazni a vezérlés megalkotásához. Amikor egy ilyen vezérlőt használunk, a robot viselkedésének kialakításában nagyon fontos szerepet kapnak a környezetből visszacsatolásként érkező jelek. A szoftver magától még nem határozza meg, hogy éppen mi fog történni a robottal, ha belehelyezik az adott környezetbe. Az olyan viselkedést, ami egy (egyszerű) szabályozó és a (komplex) környezet kölcsönhatásaként jön létre **kibontakozó viselkedés**nek (**emergent behavior**) hívjuk. Szigorúan véve bármelyik eddig tárgyalt robotra mondhatnánk, hogy kibontakozó viselkedést mutat, hiszen egyetlen modell sem tökéletes. Történeti okok miatt azonban ezt a kifejezést csak arra az irányítási technikára mondják, amelyik nem használ explicit környezeti modellt. A kibontakozó viselkedés jellemző tulajdonsága számos biológiai organizmusnak is.

Technikailag a reaktív irányítás csak egy implementációja az MDF-ek (illetve ha vannak belső állapotok, akkor POMDF-ek) stratégiáinak. A 17. fejezetben számos technikát ismertettünk, amelyek alkalmasak a robot modelljéből és a robot környezetének a modelljéből stratégiák generálására. A robotikában az ilyen stratégiák kézzel történő megszerkesztésének nagy gyakorlati jelentősége van, mivel képtelenek vagyunk tökéletes modelleket alkotni. A 21. fejezetben leírt megerősítéses tanulási módszerek segítségével tapasztalati úton lehet stratégiákat szerkeszteni. Néhány ezen módszerek közül – például a Q-tanulás és a stratégiakereső módszerek – nem igényel környezeti modellt, hanem a tanulás során összegyűjtött, nagy mennyiségű adatra támaszkodik. Így alkalmas jó minőségű szabályozók létrehozására robotok számára.

7. Szoftverarchitektúrák a robotikában

Szoftverarchitektúrának (**software architecture**) hívják az algoritmusok strukturálásának módszertanát. Egy architektúra általában nyelveket és eszközöket tartalmaz programok írásához, valamint egy átfogó szisztémát, amely alapján a programokat össze lehet illeszteni.

Created by XMLmind XSL-FO Converter.

A modern robotikai szoftverarchitektúráknak sikeresen kell ötvözniük a reaktív irányítást és a modellalapú, tudatos vezérlést. Sok szempontból a reaktív és a tudatos irányítás eltérő előnyökkel és hátrányokkal rendelkezik. A reaktív szabályozás szenzoros információn alapul, és nagyon jól használható valós idejű, alacsony szintű döntéshozásra. Ugyanakkor ritkán ad globális szinten elfogadható megoldást, mert a globális döntések olyan információt igényelnek, amely a döntés meghozatalakor még nincs a robot birtokában. Ilyen esetekben a modellalapú tervezés a megfelelő.

Következésképpen a legtöbb robotarchitektúra reaktív technikákat használ az alacsony szintű irányításhoz, és modellalapút a magasabb szintűhöz. Amikor a PD szabályozóról beszéltünk, számításba vettünk egy ilyen kombinációt, ahol a (reaktív) PD szabályozó együtt dolgozik a (modellalapú) pályatervezővel. **Hibrid architektúrá**nak (**hybrid architecture**) hívják általában azokat az architektúrákat, amelyek ötvözik a reaktív és a modellalapú technikákat.

7.1. Alárendelt architektúra

Az **alárendelt architektúra** (**subsumption architecture**) (Brooks, 1986) egy olyan keretrendszer, amelyen belül lehetőség nyílik reaktív szabályozók véges automatákból történő összeállítására. Ezeknél az automatáknál, állapotgépeknél a csomópontok egyes szenzoros változók vizsgálatát tartalmazzák olyan esetekben, ahol a véges automatánál is a változó vizsgálatának eredménye határozta meg a végrehajtás menetét. Az éleket el lehet látni címkékkel, amelyek akkor generálódnak, amikor az éleken áthaladunk, és amelyeket a robot motorjához vagy egy másik véges automatához küldünk. Azonfelül a véges automatáknak vannak belső időzítőik (óráik), amik irányítják, hogy mennyi idő alatt lehet átjutni egy-egy élen. Az így kapott állapotgépet szokás **kiterjesztett véges automatá**nak (KVA) – (**augmented finite state machine, AFSM**) hívni, ahol a kiterjesztés a belső órajel használatára vonatkozik.

Egy egyszerű KVA használatára példa a 25.22. (b) ábrán látható négyállapotú gép, amely ciklikus mozgást generál a hexapod robot számára. Ez az KVA ciklikus szabályozót használ, aminek működése tulajdonképpen nem függ a környezetből jövő visszacsatolástól. Az előrenyújtási szakasz során viszont már használja a szenzoros információt. Ha a láb elakad, azaz nem tudja végrehajtani az előremozgást, a robot visszahúzza a lábat, majd magasabbra emelve újra próbálkozik. Azaz a szabályozó képes *reagálni* a robot környezettel való fizikai érintkezésére.

Az alárendelt architektúra további primitíveket szolgáltat az KVA-ek szinkronizálásához, és a többszörös, esetleg ellentmondásban lévő KVA-ák kimenetének összehangolásához. Ezáltal lehetővé válik a programozó számára, hogy egyre bonyolultabb szabályozókat építsen fel bottom-up, alulról felfelé technikával. Példánkban egy-egy KVA-val kezdhetünk az egyes lábak vezérléséhez, majd a következő KVA már több láb működésének összehangolását végezheti. Mindezekfölött megvalósíthatunk valamilyen magas szintű viselkedést, mondjuk, egy ütközés elkerülését, amely már hátrálást és visszafordulást is tartalmaz.

Az ötlet, hogy KVA-ákkal valósítsuk meg a robot szabályozását, igen nagy előrelépés. Képzeljük el, milyen nehéz lenne ugyanezt a viselkedést leírni a konfigurációs térben, bármelyik korábban tárgyalt pályatervező algoritmussal! Először is szükségünk lenne a terep pontos modelljére. A hatlábú robot konfigurációs tere összesen tizennyolc dimenziós (tizenkét dimenzió a lábankénti két-két motortól származik, hat pedig a robot pozíciója és orientációja a környezethez képest). Még ha a számítógépeink elég gyorsak is lennének, hogy kiszámoljanak egy pályát egy ilyen kiterjedt térben, további problémát jelentenének az egyéb előforduló hatások, mint például az, ha a robot megcsúszik a lejtőn. Az ilyen sztochasztikus hatások miatt egyetlen út a konfigurációs térben csaknem bizonyosan túl törékeny lenne, és még egy PID szabályozó sem tudna megbirkózni a felmerülő problémákkal. Más szavakkal, a modellalapú mozgástervezés túl bonyolult feladat a mai robotok pályatervező algoritmusainak.

Sajnos azonban az alárendelt architektúrának is megvannak a maga problémái. Először is, az KVA-ák általában feldolgozatlan szenzoros információt kapnak, ami csak akkor elegendő a döntéshozáshoz, ha az adatok megbízhatók, és tartalmazzák az összes szükséges információt. De ha a szenzoros adatokat nemtriviális módon kell időben integrálni, akkor sikertelen lehet a döntéshozás. Éppen ezért az alárendelt típusú szabályozókat lokális feladatokra használják, mint például fal mentén haladás vagy egy látható fényforrás követése. Másodszor, a modell hiánya nehézkessé teszi a robot feladatának megváltoztatását. Az alárendelt típusú robot általában csak egy feladatot hajt végre, és nincs is elképzelése arról, hogy hogyan módosíthatná vezérlését úgy, hogy más feladatokat is el tudjon látni (mint a ganajtúró bogár a 2.2. alfejezetben). Végezetül, az alárendelt típusú vezérlést elég nehéz átlátni. A gyakorlatban több tucat bonyolult, együttműködő KVA és a környezet kölcsönhatásait a legtöbb programozó már nem tudja követni. Ezen okok miatt az alárendelt architektúrát – nagy

Created by XMLmind XSL-FO Converter.

történeti jelentősége ellenére – ritkán szokták kereskedelmi forgalomba kerülő robotoknál használni. Ugyanakkor néhány leszármazottját már igen.

7.2. Háromréteges architektúra

A hibrid architektúra modellbe épített tudással ötvözi a reaktivitást. Messze a legnépszerűbb hibrid architektúra az úgynevezett **háromréteges architektúra** (**three-layer architecture**), amely egy reaktív, egy végrehajtó és egy modellező rétegből áll.

A **reaktív réteg** (**reactive layer**) alacsony szintű vezérlést biztosít a robot számára. Ezt egy szoros szenzorcselekvés-hurok jellemzi. Döntési ciklusa gyakran milliszekundum nagyságrendű.

A **végrehajtó** vagy **szekvenciális réteg** (**executive/sequencing layer**) kapcsolja össze a reaktív réteget a modellezővel. Direktívákat fogad a modellező rétegtől, és egymás után továbbítja azokat a reaktív réteg számára. A végrehajtó réteg kezelheti például a modellező réteg pályatervező algoritmusa által generált útvonalpontokat, és eldönti, hogy melyik reaktív viselkedést hívja meg. A döntési idő a végrehajtó réteg esetében átlagosan másodperc nagyságrendű. Ez a réteg felelős továbbá azért is, hogy a szenzoros információkat egy belsőállapot-reprezentációba integráljuk. Kezeli például a robot lokalizációs és online térképezési rutinjait.

A **modellező réteg** (**deliberate layer**) a beépített tudással komplex feladatokra globális megoldásokat generál, tervezéssel. A bonyolult feladatok nagy számítási igénye miatt a döntési ciklusidő perc nagyságrendű is lehet. A modellező vagy tervező rétegmodelleket használ a döntések meghozásához. Ezek a modellek lehetnek előzetesen betápláltak vagy adatokból tanulás útján létrejöttek, és általában a végrehajtó modell által gyűjtött állapotinformációt használják.

A háromrétegű architektúra különböző változatait találhatjuk meg a legtöbb mai robot szoftverrendszerében. A három rétegre való bontás nem kötelező. Egyes robotszoftverrendszerek további rétegeket is tartalmaznak, például felhasználói interfész réteget – az emberekkel való kommunikálás vezérlésére –, vagy olyan rétegeket, amelyek a cselekvésnek a többi, ugyanabban a környezetben tevékenykedő robottal való összehangolásáért felelősek.

7.3. Robotprogramozási nyelvek

Sok robotvezérlőnek megvan a saját külön programozási nyelve. Például az alárendelt architektúra sok programja a Brooks (Brooks, 1990) által megalkotott **viselkedési nyelv**et (**behavior language**) alkalmazza. Ez egy szabályalapú, valós idejű nyelv AFSM vezérlésekhez. A LISP-hez hasonló szintaxisú, egyedi szabályokat fordít AFSM-ekra, valamint több AFSM lokális és globális üzenetküldő mechanizmussal van integrálva.

Csakúgy, mint az alárendelt architektúra, a viselkedési nyelv is csak egyetlen AFSM-re képes fókuszálni, és relatív alacsony szintű kommunikációt enged meg az egységek között. Az újabb kutatások erre az ötletre építve olyan nyelvek sorát fejlesztették ki, amelyek alapvetően hasonlítanak a viselkedési nyelvre, de nagyobb teljesítményűek, és gyorsabban futnak. Ilyen nyelv például az **általános robot nyelv** (**generic robot language, GRL**), amely Horswill (Horswill, 2000) nevéhez fűződik. A GRL egy funkcionális programozási nyelv nagyméretű moduláris irányítási rendszerekhez. Ugyanúgy, mint a viselkedési nyelv, a GRL is véges automatákból építkezik. Ezenfelül sokkal nagyobb lehetőséget hagy a különböző modulok közötti kommunikációs formák és szinkronizációs lehetőségek definiálására. A GRL-programok hatékony imperatív nyelvekre fordítódnak le, mint amilyen például a C.

Egy másik fontos, modern programozási nyelv (és hozzá tartozó architektúra) a Firby (Firby, 1994) által megalkotott reaktív akciótervrendszer (reactive action plan system, RAPS). A RAPS lehetővé teszi, hogy a programozók célokat és ezekhez tartozó terveket (vagy részleges stratégiákat) definiáljanak, valamint feltételeket, amelyek esetén az adott terv nagy valószínűséggel sikeres lesz. Ami nagyon fontos, hogy a RAPS lehetőséget ad az esetleges *kudarc* kezelésére is, ami szükségszerű velejárója a valódi robotikai rendszereknek. A programozó számos érzékelő rutint specifikálhat a különböző típusú kudarcok esetére, és mindegyikhez definiálhat egy kivételkezelési rutint. Háromrétegű architektúráknál gyakran használják a RAPS-ot a végrehajtó rétegben, az újratervezést nem igénylő kis eltérések, váratlan események kezelésére.

Számos más nyelv létezik, amely lehetővé teszi a robot számára, hogy mérlegeljen és tanuljon. Ilyen például a GOLOG (Levesque és társai, 1997b), amely akadályok nélkül egyesíti a modellalapú problémamegoldást (tervezést) és a reaktív irányítás közvetlen megvalósítását. A GOLOG-programok szituációkalkulus (10.3. alfejezet) formájában jutnak kifejezésre, nemdeterminisztikus mozgásoperátorok hozzáadásának a

Created by XMLmind XSL-FO Converter.

lehetőségével. Ha a vezérlőprogramban nemdeterminisztikus mozgások is specifikáltak, a programozónak teljes modellt kell szolgáltatnia a robotról és környezetéről. Amikor a vezérlőprogram egy nemdeterminisztikus választási lehetőséghez ér, egy tervező algoritmust hív meg (szabálykészítő formában), hogy az meghatározza a következő lépést. Ilyen módon a programozó részleges vezérlőket ír, és a végső irányítási döntéshozást a beépített mechanizmusra bízza. A GOLOG szépsége a modellalapú és a reaktív vezérlés probléma nélküli összeillesztésében van. Annak ellenére, hogy a GOLOG alkalmazásának több feltétele van (teljes megfigyelhetőség, diszkrét állapotok, teljes modell), sok beltéri robot számára nyújt megfelelő, magas szintű irányítást.

A **CES**, C++ beágyazott rendszerekhez (C++ for embedded systems), a C++ nyelv kiterjesztése, amely már valószínűségeket és tanulást is tartalmaz (Thrun, 2000). A CES adattípusai valószínűségi eloszlások, amelyek a programozó számára lehetővé teszik a bizonytalan információk kezelését a korábbi nehézségek nélkül. Ami még fontosabb, hogy a CES lehetőséget nyújt a robot szoftverpéldákkal való tanítására, nagyon hasonlóan a 20. fejezetben tárgyalt tanulási algoritmusokhoz. Ennek kihasználásával a programozók hagyhatnak helyet a programban a később megtanulható függvények számára – ezek tipikusan differenciálható parametrikus reprezentációk, mint például a neurális hálók. Ezeket a függvényeket a külön tanulási fázisban tanulja meg a rendszer, ahol a tanító határozza meg a kívánt kimeneti viselkedést. A CES sikeresnek bizonyult részlegesen megfigyelhető és folytonos esetekben is.

Az **ALisp** (Andre és Russell, 2002) a LISP továbbfejlesztett változata. Az Alisp lehetővé tesz nemdeterminisztikus döntési pontok megadását, hasonlóan a GOLOG döntési pontjaihoz. De ahelyett, hogy a tételbizonyítóra bízná a döntést, az ALisp induktív módon, megerősítéses tanulással megtanulja a helyes cselekvést. Az ALisp-ot rugalmas eszköznek tekinthetjük a tárgytartományról szóló – főleg a megfelelő döntések szubrutinjainak hierarchikus szerkezetére vonatkozó – tudásnak megerősítéses tanulással tanuló rendszerbe való beépítésére. Eddig csak szimulációk során alkalmazták az ALisp-ot, de nagyon ígéretes módszernek tűnik olyan robotok építéséhez, amelyek tanulnak a környezetükkel való kölcsönhatásból.

8. Alkalmazási területek

Most pedig felsorolunk néhányat a robottechnológia fontos alkalmazási területei közül.

Ipar és mezőgazdaság. Hagyományosan azok az ágazatok használnak robotokat, amelyek nehéz, fizikai jellegű emberi munkát igényelnek, de mégis megfelelően strukturálhatók és automatizálhatók. A legjobb példa az összeszerelő üzemi gyártósor, ahol a manipulátorok rutinszerűen végzik feladatukat, az összeszerelést, az alkatrészek behelyezését, az anyagkezelést, a hegesztést és a festést. Sok esetben a robotok sokkal költséghatékonyabbak, mint a humán munkaerő.

A szabadban sok nehéz gép helyét robotok vették át a betakarításnál, a bányászatban, a földmunkáknál. Például nemrégiben a Carnegie Mellonon, egy projekt során bemutatták, hogy a robotok 50-szer gyorsabban képesek levakarni a festéket egy hajóról, mint az emberek, és sokkal kevesebb kárt tesznek a környezetben. Autonóm bányászrobotok prototípusai bizonyították, hogy sokkal gyorsabban és pontosabban képesek kitermelni az ércet a föld alatti bányákban, mint az emberek. Robotokat használnak az elhagyatott bányák és csatornarendszerek nagy pontosságú feltérképezéséhez is. Ugyan sok rendszer még csak kísérleti stádiumban van, pusztán idő kérdése, hogy a robotok mikor veszik át az emberektől a félig mechanikus munkák többségét.

Szállítás. A robotizált szállításnak sok oldala van. Az autonóm helikopterektől kezdve, amelyek nehezen megközelíthető helyekre juttathatnak el szállítmányokat, az automata kerekes székeken át (amik olyan embereket szállítanak, akik nem tudnák irányítani a hagyományos kerekes széket), egészen az autonóm rakodógépekig, amelyek felülmúlják a dokkokban a szakképzett humán sofőrt is a konténerek hajókra és teherautókra való bepakolásánál. Jó példa a beltéri szállító robotokra a 25.23. (a) ábrán látható Helpmate nevű robot. Ezt a robotot kórházak tucatjaiban használják étel és egyéb orvosi felszerelés szállítására. A kutatók olyan autószerű robotot is kifejlesztettek, amely képes önállóan navigálni az autópályán, és terepen is megállja a helyét. Üzemi körülmények között a szállítórobotok ma már rutinszerűen látják el a szállítási feladatokat raktárakban és a gyártósorok között.

Ezen robotok közül sok igényli, hogy munkája elvégzéséhez változtassunk valamit környezetén. A leggyakoribb, hogy tájékozódást segítő elemeket kell felszerelni, például induktív köröket a padlóba, aktív fényjeleket, vonalkódokat vagy GPS-adókat. Jelenleg még mindig nagy kihívás olyan robotot tervezni, amely képes a természetes környezetben tájékozódni, és nem igényel mesterséges eszközöket. Ez különösen fontos olyan esetekben, mint például a mélytengeri kutatások, ahol a GPS nem elérhető.

Created by XMLmind XSL-FO Converter.

Veszélyes környezet. Robotok segítettek az embereknek a nukleáris hulladékok eltakarításában. A legnevezetesebb helyszín Csernobil és Three Mile Island volt. A robotok ott voltak a World Trade Center összeomlása után is, és olyan helyekre is bementek, amelyek a mentőcsapatok számára túl veszélyesek voltak.

Néhány országban lőszer szállítására és – ami különösen veszélyes feladat – bombák hatástalanítására használnak robotokat. Számos kutatás irányul napjainkban aknaszedő robotok kifejlesztésére – mind szárazföldi, mind tengeri használatra. A legtöbb ilyen célú robot távirányítással működik, azaz emberek vezérlik messziről őket. Ezen robotok önállóvá tétele a következő fontos fejlesztési lépés.

Felfedezés. A robotok eljutottak oda is, ahová az ember még soha, például a Mars felszínére (lásd 25.1. (a) ábra). Robotkarok segítik az űrhajósokat a műholdok befogásában és pályára állításában, valamint a Nemzetközi Űrállomás (ISS) építésében. Robotok segítenek a tenger alatti kutatásokban. Gyakran használják őket elsüllyedt hajók feltérképezésére. A 25.24. ábrán egy elhagyatott szénbányát feltérképező robot látható, a távolságszenzorai által a járatról alkotott 3D modellel együtt. 1996-ban tudósok egy járórobotot engedtek le egy működő vulkán kráterébe, és fontos adatokat gyűjtöttek klimatológiai kutatásokhoz. Ember nélküli légi járműveket, más néven **drón**okat (**drone**s) használnak katonai műveletekhez. A robotok egyre hatékonyabbak az információgyűjtésben olyan területeken, amelyek nehezen megközelíthetők (vagy veszélyesek) az ember számára.

25.23. ábra - (a) A Helpmate robot élelmiszert és egyéb orvosi felszerelést szállít több tucat kórházban, világszerte. (b) Sebészrobotok a műtőben (da Vinci Surgical Systems).



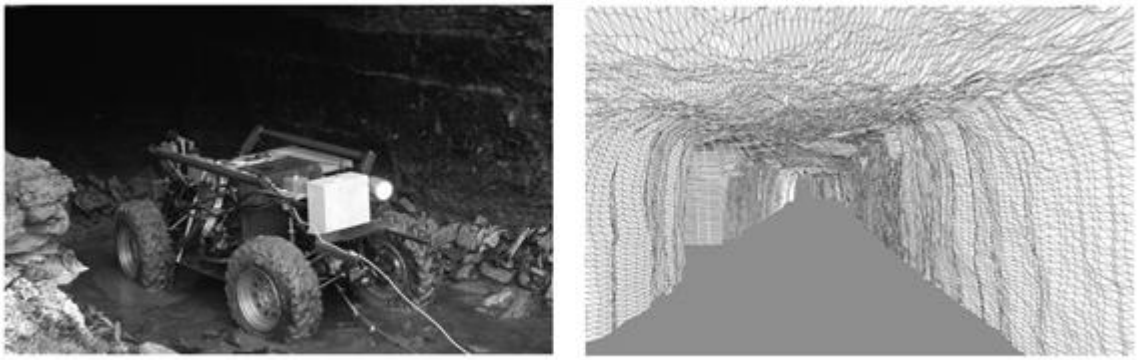
(a) (b)

Egészségügy. A sebészorvosok munkáját segítendő egyre többször használnak robotokat az orvosi eszközök pontos mozgatásához olyan kényes szerveket érintő műtétek esetében, mint az agy, a szív vagy a szem. A 25.23. (b) ábra egy ilyen rendszert mutat be. A robotok – nagy pontosságuknak köszönhetően – nélkülözhetetlen eszközökké váltak bizonyos csípőprotézisek beültetésénél. Pilottanulmányok kimutatták, hogy végbéltükrözés esetén a robotizált eszközök használata csökkenti a sérülés veszélyét. A műtőn kívül, a kutatók megkezdték olyan robotok kifejlesztését, amelyek idős vagy fogyatékos emberek segítségére lehetnek, mint például az intelligens robotizált járógépek vagy intelligens játékok, amelyek figyelmeztetnek a gyógyszerek bevételére.

Személyi kiszolgálók. A szolgáltatás a robotok nagy, ígéretes alkalmazási területe. A szervizrobotok a mindennapi teendők elvégzésében segédkeznek. A kereskedelmi forgalomban is kapható szervizrobotok között vannak autonóm porszívók, fűnyírók és golflabdaszedők. Ezen robotok mindegyike önállóan, emberi segítség nélkül képes navigálni, és teljesíteni feladatát. Néhány szervizrobot nyilvános helyeken tevékenykedik, mint például az idegenvezetőként alkalmazott robotizált információs pult a bevásárlóközpontokban, a vásárokon vagy a múzeumokban. A szervizfeladatok ellátása emberi interakciót kíván, és azt, hogy a robot megfelelően boldoguljon kiszámíthatatlan és dinamikus környezetekben.

25.24. ábra - (a) Egy elhagyatott bánya feltérképezése egy robottal. (b) A bányának a robot által kapott 3D képe.

Created by XMLmind XSL-FO Converter.



(a) (b)

Szórakozás. A robotok megkezdték a játék- és szórakoztatóipar meghódítását is. A 25.4. (b) ábrán már találkoztunk a Sony AIBO-val, ami bár kutyaszerű játék, világszerte használják MI-kutatólaborokban különböző projektek platformjaként. Az egyik, nagy kihívást jelentő feladat, ahol az AIBO-kat fizikai eszközként használták, a **robotfoci** (**robotic soccer**): versengés két csapat között, az emberi focihoz hasonló szabályokkal, de autonóm mobil robotokkal. A robotfoci nagyon jó lehetőségeket kínál az MI-kutatásokhoz, mert sok olyan problémát foglal magában, ami máshol, komolyabb alkalmazásokban is megjelenik. Az évente megrendezésre kerülő robotfoci-bajnokságok sok MI-kutatót vonzzanak, és érdekesebbé, izgalmasabbá tették a robotikának ezt az ágát.

Emberi kiegészítők. Az utolsó alkalmazás, amiről szót ejtünk, az emberi kiegészítők. A kutatók kifejlesztettek már a kerekes székhez hasonló lépegető járógépet, amelyek alkalmasak emberek szállítására. Számos helyen jelenleg arra koncentrálnak, hogy olyan eszközöket alkossanak, amelyek megkönnyítik az embernek a járást vagy a karok mozgatását egy, a testhez kívülről csatlakoztatott vázzal, amely képes pluszerőt kifejteni. Ha valakire tartósan rögzítenek egy ilyen szerkezetet, akkor mesterséges végtagnak is tekinthető. A robotok általi távjelenlét, illetve teleoperáció is felfogható emberi kiegészítésnek. A teleoperáció azt jelenti, hogy robotikai eszközök segítségével nagy távolságból tudunk megadott feladatokat végrehajtani. A teleoperáció népszerű megvalósítása a mester–szolga (master-slave) elvű konfiguráció, amikor is a robot manipulátor a távolban lévő, humán operátor mozdulatait követi, egy heptikus (érzetet is közvetítő) interfészen keresztül. Ezek a rendszerek mind jobban kiterjesztik az emberi képességeket a környezettel való tökéletesebb kölcsönhatás érdekében. Néhány projekt azt tűzte ki célul, hogy lemásolja az embert, legalábbis felületes szinten. Japánban már számos cégnél kaphatók humanoid robotok.

9. Összefoglalás

A robotika tulajdoképpen nem más, mint intelligens ágensek, amelyek képesek megváltoztatni a fizikai világot. Ebben a fejezetben a következő alapvető ismereteket sajátítottuk el a robothardverrel és -szoftverrel kapcsolatban.

- A robotok szenzorokkal vannak felszerelve, hogy érzékelhessék az őket körülvevő világot, és beavatkozó szervekkel, hogy fizikai erőkkel hathassanak környezetükre. A legtöbb robot vagy manipulátor, amely egy adott helyhez van rögzítve, vagy mobil robot, ami képes mozogni.

- A robotikai érzékelés a döntéshez kapcsolódó mennyiségeknek a szenzoros adatokból való becslése. Ehhez szükségünk van egy belső reprezentációra, valamint egy olyan módszerre, amivel azt időnként frissíthetjük. A nehéz érzékelési problémák gyakori példái a helymeghatározás és a térképezés.

- Valószínűség-alapú szűrő algoritmusok, mint például a Kalman-szűrők és a részecskeszűrők hasznosak a robotérzékelés szempontjából. Ezek a technikák frissítik a belső világképet, például az állapotváltozók a posteriori eloszlását.

- A robot mozgásának tervezése általában a konfigurációs térben történik, ahol minden pont egyértelműen meghatározza a robot egy adott pozícióját és orientációját, csuklóinak szögét.

- A konfigurációs térben kereső algoritmusok között fontos megemlíteni a celladekompozíciót, amely a teljes konfigurációs teret véges sok cellára bontja fel, valamint a szkeletonizációs technikákat, amelyek az adott

Created by XMLmind XSL-FO Converter.

konfigurációs teret egy alacsonyabb dimenzióba vetítik le. A mozgástervezés problémáját ezután már egyszerűbb struktúrákban való kereséssel is meg lehet oldani.

- A kereső algoritmus által megtalált pályán a robot akkor tud végighaladni, ha egy PID szabályozót teszünk a rendszerbe, aminek referenciapályája a kívánt útvonal.
- A potenciáltér-alapú technikák a céltól és az akadályoktól való távolság szerint definiált potenciálfüggvények segítségével navigálják a robotot. Ezek a technikák megakadhatnak lokális minimumhelyeken, de képesek mozgásgenerálásra előzetes tervezés nélkül.
- Néha egyszerűbb egy robotszabályozót közvetlenül megtervezni, mint a pályát a környezet explicit modelljéből levezetni. Egy ilyen szabályozót legtöbbször leírhatunk véges automataként.
- Az alárendelt architektúra a programozók számára lehetővé teszi, hogy belső órajellel kiegészített véges automaták összekapcsolásával robotvezérlőket alkossanak.
- A népszerű háromrétegű architektúra olyan robotszoftver fejlesztéséhez is keretet ad, amely integrálja a modellalapú tervezést, a részfeladatokra bontást és a szabályozást.
- Léteznek feladatspecifikus robotprogramozási nyelvek, amelyek megkönnyítik a robotszoftverek fejlesztését. Ezek a nyelvek kész szerkezetekkel segítik a többszálú programok írását, a szabályozási irányelveknek a tervezésbe történő integrálását és a tapasztalatalapú tanulást.

9.1. Irodalmi és történeti megjegyzések

A **robot** szót a cseh drámaíró, Karel Capek népszerűsítette 1921-ben írt *R.U.R.* (Rossum's Universal Robots) c. színművével. Az ő robotjait nem mechanikusan szerelték össze, hanem kémiai úton állították elő. A műben a robotok megharagszanak mestereikre, és elhatározzák, hogy átveszik az uralmat. Úgy tűnik (Glanc, 1978), hogy valójában Capek bátyja, Josef alkotta meg a robot szót a cseh „robota" (robotolás) és „robotnik" (jobbágy) szó összevonásával, 1917-es *Opilec* c. novellájában.

A robotika kifejezést Asimov használta először 1950-ben, ugyanakkor a robotika története – más megnevezéssel – sokkal régebbre nyúlik vissza. Az ókori görög mitológiában Talószt, a mechanikus embert állítólag Héphaisztosz, a kovácsisten készítette. Jacques Vaucanson egy csodálatos automatát épített a 18. században. Az 1738-as mechanikus kacsa a korai robotok egy példája, aminek összetett viselkedése még fixen volt beleépítve. Az első programozható, robotszerű alkotás talán Jacquard szövőszéke volt (1805), amiről már az 1. fejezetben már szót ejtettünk.

Az első kereskedelmi forgalomba került robot a **Unimate** (universal automation) manipulátor volt. A Unimate-et Joseph Engelberger és George Devol fejlesztette. 1960-ben adták el az első példányt a General Motorsnak, ahol tv-képcsövek gyártásánál használták. Szintén 1960-ben nyújtotta be Devon az első robotokra vonatkozó szabadalmat az Egyesült Államokban. 11 évvel később a Nissan Corp. az elsők között oldotta meg egy teljes szerelősor automatizálását. A fejlesztést a Kawasaki végezte Engelberger és Devol Unimation nevű cégétől származó robotokkal. Főként Japánban és az USA-ban ez az előrelépés jelentős átalakulásnak nyitott utat, amely mind a mai napig tart. A Unimationt a PUMA robot fejlesztése követte 1978-ban. A **PUMA** a Programmable Universal Machine for Assembly (programozható univerzális gép szereléshez) névből származik, és eredetileg a General Motors számára fejlesztették. A következő évtizedekben ez lett a robotmanipulátorok *de facto* szabványa. Manapság egymillióra becsülik a világon dolgozó robotok számát, és ezeknek több mint fele Japánban tevékenykedik.

A robotikai kutatásokról szóló szakirodalom nagyjából két részre osztható: mobil robotokra és rögzített manipulátorokra. A Grey Walter által 1948-ban készített „teknőst" tekinthetjük a világ első autonóm mobil robotjának annak ellenére, hogy a rendszer nem volt programozható. A '60-as években, a Johns Hopkins Egyetemen készített „Hopkins Beast" már sokkal kifinomultabb volt. Mintafelismerő hardverével azonosította a szabványos fali AC-csatlakozókat, és képes volt mozgása során megtalálni azokat, rácsatlakozni és feltölteni akkumulátorát. Mégis, a Beastnek nagyon limitált képességei voltak. Az első általános célú robot a „Shakey" volt, amit a Stanford Research Institute-ban (SRI) fejlesztettek ki az 1960-as évek végén (Fikes és Nilsson, 1970; Nilsson, 1984). Shakey volt az első robot, amelybe integrálták az érzékelést, a tervkészítést és a végrehajtást. Ez a jelentős eredmény számos további MI-kutatást inspirált. Szintén jelentős hatású projektek voltak még a Stanford Cart és a CMU Rover (Moravec, 1983). A mobil robotokkal kapcsolatos klasszikus munkákat Cox és Wilfong foglalták össze (Cox, és Wilfong, 1990).

Created by XMLmind XSL-FO Converter.

A robottérképezés gyökerei két különböző helyről erednek. Az egyik szál Smith és Cheeseman munkásságával kezdődött (Smith és Cheeseman, 1986), akik Kalman-szűrőt alkalmaztak szimultán helymeghatározási és térképezési problémákhoz. Az algoritmust először Moutarlier és Chatila implementálta 1989-ben, majd Leonard és Durrant-Whyte terjesztették ki (Leonard és Durrant-Whyte, 1992). Dissanayake és társai 2001-ben írták le az ágazat jelenlegi fejlettségi szintjét (Dissanayake és társai, 2001). A másik szál a **foglalási hálózat** (**occupancy grid**) reprezentáció megalkotásával kezdődött, amelyet a valószínűségi térképezésnél használnak. A foglalási hálózat minden (x, y) pontra meghatározza annak a valószínűségét, hogy ott egy akadály van (Moravec és Elfes, 1985). A modern robottérképezésről Thrun írt összefoglalót (Thrun, 2002). Kuipers és Levitt (Kuipers és Levitt, 1988) az elsők között voltak, akik metrikus helyett topológikus térképezést javasoltak, az ember térbeli tájékozódását véve alapul.

A korai mobil robot helymeghatározási technikákat Borenstein és társai tekintették át (Borenstein és társai, 1996). Annak ellenére, hogy a Kalman-szűrő évtizedeken át jól ismert helymeghatározási módszer volt az irányításelméletben, a helymeghatározás problémájához tartozó általános valószínűségi szabály csak jóval később jelent meg az MI-irodalomban, Tom Dean és kollégái (Dean és társai, 1990), valamint Simmons és Koenig (Simmons és Koenig, 1995) munkássága révén. Ez utóbbi páros vezette be a **Markov-helymeghatározás** (**Markov localization**) fogalmát. A technika első, valósvilág-beli alkalmazását Burgard és társai mutatták be (Burgard és társai, 1999) egy sor roboton keresztül, amelyeket múzeumokba telepítettek. A részecskeszűrő-alapú Monte Carlo helymeghatározás, amelyet Fox és társai fejlesztették ki (Fox és társai, 1999), manapság széles körben használatos. A **Rao-Blackwellized** részecskeszűrő kombinálja a részecskeszűrést a robot lokalizációval, valamint az egzakt szűrést a térképalkotással (Murphy és Russell, 2001; Montemerlo és társai, 2002).

A mobil robotokhoz kapcsolódó kutatások szimulálásának az elmúlt évtizedben két jelentős verseny is teret adott. Az AAAI éves robotbajnokságát 1992 óta rendezik meg. Az első verseny győztese a `CARMEL` volt (Congdon és társai, 1992). A fejlődés azóta igen jelentős és töretlen: a 2002-es verseny során például a robotoknak be kellett járniuk a konferenciaközpontot, megtalálni a regisztrációs pultot, regisztrálni a konferenciára, és végül tartani egy beszédet. Az 1995-ben, Kitano és kollégái (Kitano és társai, 1997) által elindított **Robocup** kezdeményezés célja, hogy 2050-re olyan teljesen autonóm humanoid robotokból álló futballcsapatot fejlesszen, „amely képes megverni az aktuális emberi világbajnokcsapatot”. Különböző bajnokságokat szerveznek kerekes, eltérő méretű, szimulált robotoknak és négylábú Sony Aibóknak. 2002-ben a világbajnokságra több mint 30 országból érkeztek csapatok, és a rendezvény legalább 100 000 nézőt vonzott.

A robotmanipulátorok, eredeti nevükön **kéz–szem gép**ek (**hand-eye machine**) tanulmányozása meglehetősen más vonalon fejlődött. Egy ilyen kéz–szem gép megalkotására az első igazi kísérlet Heinrich Ernst MH-1-ese volt, amelyet PhD-téziseiben írt le az MIT-n (Ernst, 1961). Az edinburghi Gépi Intelligencia (Machine Intelligence) program már igen korán jelentős eredményeket mutatott fel látásalapú összeszerelő rendszerükkel, amit `FREDDY`-nek hívtak (Michie, 1972). Ezen úttörő kísérletek után rengetegen foglalkoztak geometriai algoritmusokkal determinisztikus és teljesen megfigyelhető mozgástervezési problémákhoz. A robotmozgás-tervezés P-TÁR nehézségét Reif egy termékenyítő cikke mutatta be (Reif, 1979). A konfigurációstér-reprezentációt Lozano-Perez-nek (Lozano-Perez, 1983) köszönhetjük. Nagy befolyást gyakoroltak Schwartz és Sharir publikációi az általuk **zongoraszállító**knak (**piano movers**) nevezett problémáról (Schwartz és társai, 1987).

A konfigurációs tér rekurzív cellákra bontásával való tervezés Brookstól és Lozano-Perez-től származik (Brooks és Lozano-Perez, 1985), ezt később Zhu és Latombe fejlesztették jelentősen tovább (Zhu és Latombe, 1991). A legkorábbi szkeletonizációs algoritmusok a Voronoi-diagramokon (Rowat, 1979) és a **láthatósági gráf**okon (**visibility graph**) alapultak (Wesley és Lozano-Perez, 1979). Guibas és társai (Guibas és társai, 1992) hatékony technikát fejlesztettek ki a Voronoi-diagramok inkrementális számítására. Choset (Choset, 1996) általánosította sokkal átfogóbb mozgástervezési problémákra a Voronoi-diagramokat. John Canny alapozta meg PhD-tézisével (Canny, 1991) az első különálló exponenciális algoritmust pályatervezésre. Másféle szkeletonizációs módszert használt, amit **sziluett** (**silhouette**) algoritmusnak hívnak. Jean-Claude Latombe írása (Latombe, 1991) a különböző mozgástervezési metódusok széles skáláját fedi le. Kavraki és társai valószínűségi útitervet fejlesztettek (Kavraki és társai, 1991), amely még ma is a leghatékonyabb. Lozano-Perez és társai (Lozano-Perez és társai, 1984), valamint Canny és Reif (Canny és Reif, 1987) is foglalkoztak a finom mozgások tervezésével korlátozott érzékelés esetén, és intervallumra vonatkoztatott bizonytalanságot használtak valószínűségi bizonytalanság helyett. A referenciapont-alapú navigálás a mobil robotok esetében sok hasonló ötletet használ (Lazanas és Latombe, 1992).

A robotok mint dinamikus rendszerek irányítása, legyen szó manipulátorról vagy mobil robotról, hatalmas irodalommal rendelkezik, amelynek anyagáról ebben a fejezetben alig-alig esett szó. A fontos művek közé

Created by XMLmind XSL-FO Converter.

tartozik Hogan trilógiája (Hogan, 1985) az impedancia szabályozásáról és egy általános tanulmány a robotok dinamikájáról Featherstone-tól (Featherstone, 1987). Dean és Wellman az elsők között voltak, akik megpróbálták összekötni az irányításelméletet és az MI-tervezőrendszereket (Dean és Wellman, 1991). Három klasszikus tankönyv a robotkarok matematikájáról Paultól (Paul, 1981), Craigtól (Craig, 1989) és Yoshikawától (Yoshikawa, 1990) származik. A **megfogás** (**grasping**) tárgyköre szintén nagyon fontos a robotikában: egy stabil megfogás meghatározásának problémája meglehetősen nehéz (Mason és Salisbury, 1985). A megfelelő megfogás érintkezésérzékelőket vagy **heptikus visszacsatolás**t (**haptic feedback**) kíván a fellépő erők és az esetleges csúszás meghatározására (Fearing és Hollerbach, 1985).

A potenciáltér-vezérlést, amely egyszerre próbálja megoldani a mozgástervezés és az irányítás problémáját, Khatib vezette be a robotika irodalmába (Khatib, 1986). Mobil robotok esetében ezt az ötletet részleges megoldásnak tekintették az ütközések elkerülésének problémájához. Később Borenstein (Borenstein, 1991) továbbfejlesztette az algoritmust, és a **vektortérhisztogram** (**vector field histogram**) nevet adta neki. A navigációs függvényeket, a determinisztikus MDF-hez tartozó vezérlési irányelvek robotikai megfelelőjét, Koditschek vezette be (Koditschek, 1987).

A robotokhoz tartozó szoftverarchitektúra témája sok heves vitát váltott ki. Az MI jó öreg zászlóshajója – a háromrétegű architektúra – Shakey tervezésének eredménye, később Gat újította fel (Gat, 1998). Az alárendelt architektúra Brooks munkája (Brooks, 1986), bár tőlük függetlenül Braitenberg (Braitenberg, 1984) is fejlesztett hasonlót, akinek könyve, a *Járművek*, egy sor egyszerű viselkedésalapú robotot mutat be. Brooks hatlábú lépegető robotjának sikerét sok más projekt követte. Connell PhD-munkája keretében (Connell, 1989) olyan teljesen reaktív mobil robotot fejlesztett, amely képes volt tárgyakért elmenni és azokat elhozni. A viselkedésalapú paradigma többrobotos rendszerekre való kiterjesztését megtalálhatjuk Mataric (Mataric, 1997) és Parker (Parker, 1996) munkáiban. A GLR (Horswill, 2000) és a `COLBERT` (Konolige, 1997) robotok megvalósították a párhuzamosított viselkedésalapú irányítást általános robotprogramozási nyelveken. Arkin átfogó képet nyújtott a téma aktuális helyzetéről (Arkin, 1998).

A 7. fejezetben leírt **helyezett automatá**kat (**situated automata**) (Rosenschein, 1985; Kaelbling és Rosenschein, 1990) szintén használják mobil robotok irányítására, felderítési és szállítási feladatoknál. A helyezett automaták szorosan kapcsolódnak a viselkedésalapú tervezéshez abból a szempontból, hogy véges automatákból állnak, amelyek egyszerű kombinációs hálózatokkal nyomon követik a környezet állapotának bizonyos aspektusait. Amíg a viselkedésalapú megközelítés az explicit reprezentáció hiányát hangsúlyozza, a helyezett automatákat algoritmikusan építik deklaratív környezeti modellekből, így minden állapotregiszter által reprezentált tartalom jól definiált.

Számos jó kézikönyv jelent meg mostanában a mobil robotokról. A fent említetteken felül Kortenkamp és társai (Kortenkamp és társai, 1998) gyűjtése minden részletre kiterjedő áttekintést nyújt a mai mobil robot architektúrákról és rendszerekről. Két újabb könyv, Dudek és Jenkin (Dudek és Jenkin, 2000), valamint Murphy (Murphy, 2000) munkája általánosabb megközelítésben tárgyalja a robotikát. Mason könyve (Mason, 2001) a robotkarokról olyan modern témákat is tárgyal, mint például az önbeálló mozgás. A robotikával foglalkozó legjelentősebb konferencia az *IEEE International Conference on Robotics and Automation*. A témával foglalkozó folyóiratok közül kiemelkedik az *IEEE Robotics and Automation*, az *International Journal of Robotics Research* és a *Robotics and Autonomous Systems*.

9.2. Feladatok

25.1.

A Monte Carlo-helymeghatározás *torzított* bármilyen véges méretű mintára. Azaz az algoritmus által kiszámolt várható érték a részecskeszűrés működéséből adódóan eltér a tényleges várható értéktől. Ebben a példában ezt a torzítást kell kiszámolni.

Hogy kicsit egyszerűsítsük a problémát, vegyünk egy teret, amelyben a robotnak négy lehetséges állapota (helyzete) van: $X = \{x_1, x_2, x_3, x_4\}$. Kezdetben egyenletes eloszlással választunk $N \geq 1$ mintát a megadott állapotok közül. Mint általában, itt is tökéletesen elfogadható, ha egymás után akár több is generálódik bármelyik X állapotból. Legyen Z egy bináris szenzorváltozó, az alábbi feltételes valószínűségekkel:

$P(z|x_1) = P(\neg z|x_1) = 0,2$
$0,8$

Created by XMLmind XSL-FO Converter.

$P(z|x_2) = P(\neg z|x_2) = 0,6$
$0,4$

$P(z|x_3) = P(\neg z|x_3) = 0,9$
$0,1$

$P(z|x_4) = P(\neg z|x_4) = 0,9$
$0,1$

A Monte Carlo-helymeghatározás ezeket a valószínűségeket használja, hogy generálja a részecskék súlyát, amelyeket azután normalizál és felhasznál az újabb mintavételezésnél. Az egyszerűség kedvéért feltételezzük, hogy az újabb mintavételezésnél, N nagyságától függetlenül, csak egyetlen új mintát generálunk. Ez a minta korrelálhat bármelyik X állapottal. Így a mintavevő folyamat X-re nézve meghatároz egy valószínűség-eloszlást.

a. Mi az új mintára eredményül kapott valószínűség-eloszlás X-re nézve? Válaszolja meg a kérdést az $N = 1$, …, 10 esetekre, valamint $N = \infty$-re.

b. A P és Q két valószínűség-eloszlás közötti különbséget mérhetjük a KL divergenciával, amelynek definíciója:

$$KL(P,Q) = \sum_i P(x_i) \log \frac{P(x_i)}{Q(x_i)}$$

c. Melyek a KL divergencia értékei az (a)-beli eloszlások és a valódi posterior között?

d. Milyen változtatást hajtana végre a feladat szövegezésén (nem az algoritmuson!) annak érdekében, hogy garantáljuk a fenti specifikus becslő torzítatlanságát még véges N-ekre is? Legalább két különböző megoldást adjon (mindkettőnek elégségesnek kell lennie).

25.2.

Alkalmazza egy pásztázó távolságmérővel ellátott szimulált robot esetében a Monte Carlo-helymeghatározási algoritmust. A rácstérkép és a távolságadatok elérhetők az **aima.cs.berkeley.edu** kódtárából. A megoldás akkor teljes, ha bemutatja a robot sikeres globális helymeghatározását.

25.3.

Vegyük a 25.12. ábrán látható robotkart. Tegyük fel, hogy a robot alapja 60 cm hosszú, a felkarja és az alkarja pedig egyaránt 40-40 cm. Ahogy a 25.4. alfejezetben beláttuk, az inverz kinematikai feladat megoldása sokszor nem egyértelmű. Állítson fel egy explicit, zárt alakú megoldást az inverz kinematikára erre a robotkarra. Pontosan milyen feltételek mellett egyértelmű a megoldás?

25.4.

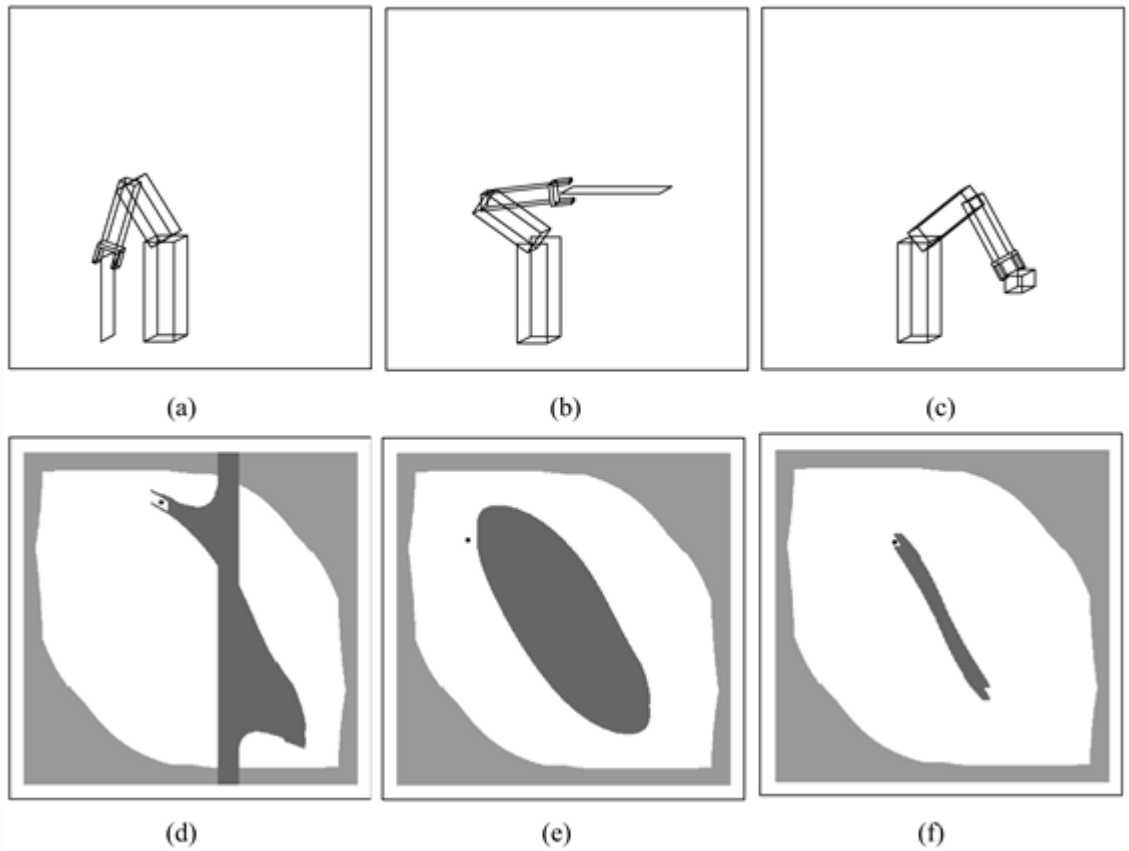
Implementáljon egy algoritmust, amely kiszámítja egy tetszőleges ($n \times n$-es logikai tömbbel) adott 2D környezet Voronoi-diagramját. Illusztrálja algoritmusát a Voronoi-diagram felrajzolásával 10 érdekes térképen. Mekkora az algoritmus komplexitása?

25.5.

Ez a feladat feltárja a kapcsolatot a munkatér és a konfigurációs tér között, a 25.25. ábra példáit használva.

25.25. ábra - Diagram a 25.5. feladathoz

Created by XMLmind XSL-FO Converter.



a. Vegyük a 25.25. (a)-tól (c)-ig látható robotkonfigurációkat, nem véve figyelembe a diagramokon feltüntetett akadályokat. Rajzolja fel az ezeknek megfelelő robotkar-konfigurációkat a konfigurációs térben. (*Segítség:* minden egyes karkonfiguráció egyetlen pontra képződik le a konfigurációs térben, ahogy ezt a 25.12. (b) ábra is mutatja.)

b. Rajzolja meg a konfigurációs teret a 25.25. (a)–(c) ábrákon látható minden egyes munkatérdiagramra. (*Segítség:* a konfigurációs terek megegyeznek a 25.25. (a) ábrán lévővel, ami a robot önmagával való ütközését jelentő területet illeti, de különbségek adódnak a terület határát jelentő akadályok hiányából és az egyes ábrákon lévő akadályok különböző helyzetéből.)

c. A 25.25. (e)–(f) ábrákon látható fekete pontok mindegyikéhez rajzolja meg a hozzá tartozó konfigurációt a munkatérben. Az árnyékolt részeket hagyja figyelmen kívül.

d. A 25.25. (e)–(f) ábrákon látható konfigurációs terek mindegyikét egy-egy különálló (a munkatérben lévő) akadály generálta (ez a sötétebb terület), valamint azok a megszorítások, amelyek a robot önmagával való ütközéséből erednek (a világosabb árnyalat). Rajzolja meg mindkét diagramhoz a sötét területnek megfelelő akadályt a munkatérben.

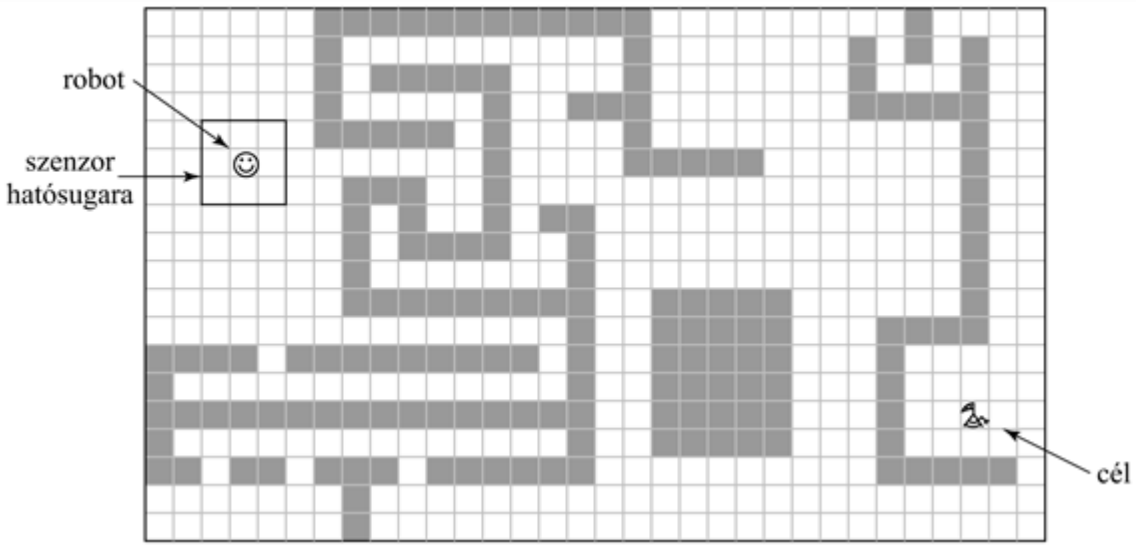
e. A 25.25. (d) ábra azt mutatja be, hogy egyetlen síkbeli (lapos) akadály két elválasztott részre tudja bontani a konfigurációs teret. Maximum hány elválasztott részre tudja bontani az egybefüggő akadálymentes teret egyetlen lapos akadály behelyezése egy 2 szabadságfokos robot esetén? Adjon példát rá, és indokolja meg, miért nem lehetséges nagyobb számú független terület kialakítása. Mi a helyzet, ha nem lapos az akadály?

25.6.

Vegyük a 25.26. ábra egyszerűsített robotját. Tegyük fel, hogy a robot derékszögű koordinátái ismertek minden időpillanatban, mint ahogy a célhelyzet is. Ugyanakkor van egy akadály, amelynek a helyzete nem ismert. A robot csak a közvetlen környezetében képes érzékelni az akadályt, mint ahogy azt az ábra is illusztrálja. Az egyszerűség kedvéért tételezzük fel, hogy a robot mozgása zajjal nem terhelt, és az állapottér diszkrét. A 25.26. ábra csak egy példa. Ebben a feladatban azt kérjük, hogy minden lehetséges rácsvilágra gondoljon, ahol van érvényes út a start pozíciótól a célig.

Created by XMLmind XSL-FO Converter.

25.26. ábra - Egyszerűsített robot egy labirintusban. Lásd 25.6. feladat.



a. Tervezzen egy modellalapú vezérlőt, amely garantálja, hogy a robot mindig eléri a célt, amennyiben ez lehetséges. A modellalapú vezérlő egy, a mozgás során fokozatosan kialakított térkép formájában képes tárolni a terep jellemzőit. A robot az egyes mozdulatok között tetszőleges időt tölthet gondolkodással.

b. Most pedig tervezzen egy *reaktív* vezérlőt ugyanerre a feladatra. Ez a vezérlő nem tárolhatja a korábbi érzékelések eredményeit. (Azaz nem készíthet térképet!) Ehelyett minden döntését az aktuális érzékelő adatai alapján hozza meg, amely magában foglalja a saját helyének és a cél pontos helyzetének ismeretét. A döntéshozási időnek függetlennek kell lennie a környezet méretétől, valamint a már megtett lépések számától. Maximum hány lépés kell a robotnak ahhoz, hogy elérje célját?

c. Hogy fog viselkedni az (a) és (b) pontban megtervezett vezérlő, ha az alábbi hat feltétel valamelyike fennáll: folytonos állapottér; zajjal terhelt érzékelés; zaj a mozgásban; zaj mind a mozgásban, mind az érzékelésben; a cél helyzete ismeretlen (csak akkor ismeri fel a célt, ha az az érzékelő hatótávolságán belülre kerül); vagy mozgó akadályok vannak. Minden egyes feltételre és mindkét vezérlőre adjon egy-egy példát, amikor is a robot elbukik (vagy magyarázza el, miért nem tud).

25.7.

A 25.22. (b) ábrán kiterjesztett véges automatát használtunk a hatlábú robot egyetlen lábának vezérléséhez. Ebben a feladatban a cél egy olyan AFSM tervezése, amely a hat külön lábhoz tartozó vezérlővel együttműködve hatékony és stabil mozgást eredményez. Ennek eléréséhez ki kell bővíteni az egyes lábakat vezérlő AFSM-eket, hogy üzeneteket is küldjenek az új AFSM-nek, és hogy várjanak amíg más üzenetek érkeznek. Magyarázza el, hogy miért hatékony a vezérlője, azaz miért nem fogyaszt feleslegesen energiát (pl. csúszó lábak miatt), és hogy miért mozgatja a robotot megfelelően nagy sebességgel! Bizonyítsa be, hogy vezérlője megfelel a 25.2. alfejezetben támasztott stabilitási feltételeknek.

25.8.

(Ezt a feladatot Michael Genesereth és Nils Nilsson találták ki, és az első évesektől a végzősökig mindenkinek szól.) Az emberek annyira ügyesek alapfeladatok elvégzésében – mint például csészék felemelése vagy dobozok egymásra rakása –, hogy gyakran elfelejtik milyen összetettek ezek a mozdulatok valójában. A feladat során felfedezzük e feladatok komplexitását, és összefoglaljuk a robotika elmúlt 30 évének fejlődését. Először határozzuk meg a feladatot (például egy kapu készítése három dobozból), majd építsünk egy robotot négy emberből az alábbiak szerint:

- **Agy:** az Agy feladata a cél elérését biztosító terv kidolgozása és a kezek irányítása a terv megvalósítása során. Az Agy információt csak a Szemtől kaphat, *közvetlenül nem láthatja a helyszínt!* Az Agy az egyetlen, aki tudja, mi a cél.

- **Szem:** a Szem feladata a helyszín rövid leírása az Agy számára. A munkakörnyezettől kb. egy méternyire kell lennie, és vagy kvalitatív leírást (mint például: „Az oldalán fekvő zöld doboz tetején egy piros doboz van”),

Created by XMLmind XSL-FO Converter.

vagy kvantitatív leírást („A zöld doboz kb. 50 cm távolságra balra van a kék hengertől”) adhat. A Szem az Agytól jövő kérdésekre is válaszolhat, így például: „Van-e hely a Bal Kezem és a piros doboz között?” Ha van kéznél egy videokamera, akkor állítsuk rá a munkakörnyezetre, és a Szem ezen keresztül nézze azt, ne lássa közvetlenül!

- **Kezek** (Jobb és Bal): mindkét Kezet más ember játssza. A két Kéz egymás mellett áll. A Balkéz csak a saját balkezét használja, a Jobbkéz pedig csak a jobbat. A Kezek csak egyszerű, az Agytól érkező feladatokat hajthatnak végre. Például: „A Balkéz mozduljon 5 cm-t előre”. Csak mozgásparancsot képesek végrehajtani – például a „vedd fel a dobozt” nem olyan parancs, amit a Kéz végre tud hajtani. A csalások megakadályozása végett megkövetelhetjük, hogy a kezeken legyen kesztyű vagy használjanak csipeszt. A Kezeknek *bekötött szemmel kell tevékenykedniük*. Egyetlen érzékelési képességük az, hogy meg tudják mondani, amikor útjukat valamilyen elmozdíthatatlan akadály, például egy asztal vagy a másik Kéz zárja el. Ilyen esetben hangjelzést adhatnak az Agy informálása céljából.

Created by XMLmind XSL-FO Converter.

26. fejezet - Filozófiai alapok

Ebben a fejezetben arról esik szó, hogy mit jelent gondolkozni, vajon képesek-e erre a mesterséges tárgyak, és szeretnénk-e, ha képesek lennének rá.

Ahogyan az 1. fejezetben említettük, a filozófusok sokkal régebb óta vannak jelen, mint a számítógépek, és folyamatosan próbálnak válaszokat találni néhány olyan kérdésre, amely a mesterséges intelligenciához kötődik: Hogyan működik az elme? Képesek-e gépek intelligensen cselekedni; és ha igen, lenne-e elméjük? Milyen etikai következményei lehetnek az intelligens gépeknek? A könyv eddigi huszonöt fejezetében a mesterséges intelligencia saját kérdéseiről volt szó, most egy fejezet erejéig a filozófusok témáival foglalkozunk.

Először néhány elnevezés: a filozófiában **gyenge MI**-hipotézisnek (**weak AI**) nevezik azt az állítást, miszerint a gépek valószínűleg képesek intelligensen cselekedni (vagy jobban mondva, képesek úgy cselekedni, *mintha* intelligensek lennének), míg azt az állítást, hogy a gépek *valóban* intelligensen cselekszenek, **erős MI**-hipotézisnek (**strong AI**) hívják.

A legtöbb MI-kutató elfogadja a gyenge MI-hipotézist, az erőssel pedig nem törődik: amíg a program működik, az nem érdekes, hogy az intelligencia szimulációjának nevezik-e vagy valós intelligenciának. Minden MI-kutatónak foglalkoznia kellene munkája etikai következményeivel.

1. Gyenge MI: Tudnak-e a gépek intelligensen cselekedni?

Néhány filozófus megpróbálta bebizonyítani, hogy a mesterséges intelligencia nem lehetséges, azaz a gépek valószínűleg nem tudnak intelligensen cselekedni. Néhányan az MI-kutatás leállítását szeretnék elérni érveikkel:

> *A számításorientált megközelítés kultuszának részeként űzött* mesterséges intelligenciának még esélye sincsen arra, hogy tartós eredményeket érjen el (…) itt az ideje, hogy az MI-kutatók erőfeszítéseit – és a támogatásukra szánt hatalmas pénzeket – máshova tereljük, mint a számításorientált megközelítés.
>
> —(Sayre, 1993)

A mesterséges intelligencia lehetséges mivolta nyilván azon múlik, miként definiáljuk. Lényegét tekintve a mesterséges intelligencia egy adott architektúrához a legjobb ágensprogramot keresi. Ebben a megfogalmazásban már a definíció szerint is lehetséges az MI: bármely k tárbitet tartalmazó digitális architektúrához pontosan 2^k ágensprogram tartozik; ahhoz pedig, hogy megtaláljuk a legjobbat, egyszerűen csak egyesével fel kell sorolni és tesztelni az ágenseket. Ez ugyan nem lenne praktikus nagy k esetén, de a filozófia az elmélettel, nem pedig a gyakorlattal foglalkozik.

A mesterséges intelligenciára adott definíciónk jól működik a mérnöki probléma esetében is, ahol egy adott architektúrához tartozó jó ágenst kell találni. Nagy a kísértés tehát, hogy befejezzük itt ezt a szakaszt, és igenlő választ adjunk a címben feltett kérdésre. A filozófusok azonban két architektúrát akarnak összehasonlítani: az embert és a gépet. Ráadásul a hagyományosan feltett kérdés így hangzik: „**Tudnak-e a gépek gondolkodni?**” Sajnos ez egy helytelenül feltett kérdés. Hogy megértsük, miért, nézzük az alábbi kérdéseket:

- Tudnak a gépek repülni?
- Tudnak a gépek úszni?

A legtöbb angolul beszélő ember[1] egyetért abban, hogy az első kérdésre igennel kell felelni: a repülőgépek tudnak repülni; de a második kérdésre az angolban a válasz nemleges: a hajók és a tengeralattjárók haladnak ugyan a vízben, de az angolban ezt nem nevezik úszásnak. Sem ezeknek a kérdéseknek persze, sem a válaszoknak nincsen semmilyen hatása a tengerészek, a tengerészeti mérnökök vagy az ezekkel a dolgokkal bármilyen kapcsolatba kerülők mindennapi munkájára. A válaszoknak alig van közük a repülőgépek és tengeralattjárók tervezéséhez és képességeihez; inkább arról szólnak, hogy a szavak használatának milyen módjait választottuk ki. Úgy alakult, hogy az „úszni” szó az angolban azt jelenti, hogy ’testrészeinek mozgásával előrehalad a vízben’, míg a „repülni” szó jelentése az angolban nem határozza meg a

[1] A kérdések eredeti formája és így az elemzés tárgya az angol nyelv. (*A ford.*)

Created by XMLmind XSL-FO Converter.

helyváltoztatás módját.[2] A „gondolkodó gépek” gyakorlati lehetősége még csak mintegy ötven éve jelent meg, és ez az idő nem volt elég az angolul beszélőknek, hogy a „gondolkodni” szó az angolban új jelentést kapjon.

Alan Turing azt javasolta *Computing Machinery and Intelligence* c. híres cikkében (Turing, 1950), hogy ne azt kérdezzük, tudnak-e a gépek gondolkodni, hanem azt vizsgáljuk, hogy átmennek-e a gépek egy viselkedési intelligenciateszten, amelyet később Turing-tesztnek neveztek el. A teszt szerint a programnak öt percen át kell (gépelt online üzenetekkel) beszélgetnie egy kérdezővel. A kérdezőnek ezután választania kell, hogy egy programmal vagy egy személlyel beszélgetett-e, és egy program akkor felel meg a teszten, ha az idő 30%-ában megtéveszti a kérdezőt. Turing azt a sejtést fogalmazta meg, hogy 2000-re egy 10^9 táregységből álló számítógépet be lehet úgy programozni, hogy megfeleljen a teszten, ámde nem lett igaza. Az ugyan *megtörtént* már, hogy néhány embert öt percig becsaptak; például az `ELIZA` program és az `MGONZ` internetes csevegőrobot becsapott olyan embereket, akik nem gondoltak arra, hogy talán egy programmal beszélhetnek, és az `ALICE` program becsapott egy zsűritagot a 2001-es Loebner-díjért zajló versenyben. De egyetlenegy program sem került annak közelébe, hogy 30%-ot érjen el képzett zsűrivel szemben, és az MI-kutatás egésze nem is szentel túl nagy figyelmet a Turing-teszteknek.

Turing az intelligens gépek lehetősége ellen felhozható érvek széles választékát is megvizsgálta, beleértve szinte minden olyan ellenérvet, amely a cikk megjelenése óta eltelt fél évszázadban felmerült. A következőkben ezek közül nézünk meg néhányat.

1.1. A képesség hiányából vett érv

A „képesség hiányából vett érv” azzal a követeléssel lép fel, hogy „a gépek sohasem képesek *X*-et tenni”. Az *X*-ekre Turing a következő példákat hozza:

> Kedvesnek, találékonynak, szépnek és barátságosnak lenni, kezdeményezni, humorérzéket mutatni, megkülönböztetni a helyest a helytelentől, hibákat követni el, szerelmesnek lenni, élvezni a tejszínes epret, elérni, hogy valaki szerelmes legyen belé, tapasztalatból tanulni, helyesen használni a szavakat, saját gondolatai alanyának lenni, annyira változatosan viselkedni, mint az ember, valami igazán újat tenni.
>
> —Turing

Turingnak az intuícióját kellett használnia, hogy megsejtse, mi válhat lehetségessé a jövőben, mi azonban abban a kényelmes helyzetben vagyunk, hogy visszanézhetünk a számítógépek eddigi teljesítményére. Senki sem tagadja, hogy a számítógépek ma sok olyan dolgot megtesznek, amely korábban kizárólag az emberekhez tartozott. A programok sakkoznak, dámát vagy más játékokat játszanak, alkatrészeket vizsgálnak gyártósorokon, a szövegszerkesztéskor ellenőrzik a helyesírást, helikoptereket és autókat irányítanak, betegségeket diagnosztizálnak és több száz más tevékenységet végeznek éppolyan jól, vagy még jobban, mint az emberek. A számítógépek kicsiny, de jelentős felfedezéseket tettek a csillagászatban, a matematikában, a kémiában, az ásványtanban, a biológiában, a számítástudományban és más területeken. Ezekhez a felfedezésekhez az emberi szakértőhöz hasonló szintű teljesítményekre volt szükség.

Annak fényében, amit most tudunk a számítógépekről, nem meglepő, hogy jól teljesítenek a sakkhoz hasonló kombinatorikai problémákban. De az algoritmusok az emberhez mérhető szinten hajtanak végre olyan tevékenységeket is, amelyek látszólag emberi döntést igényelnek, amelyek során, mint Turing mondta, képesnek kell lenni a „tapasztalatból tanulni” és „megkülönböztetni a helyest a helytelentől”. Még egészen korán, 1955-ben Paul Meehl (lásd még Grove és Meehl, 1996) képzett szakértők döntéshozatali folyamatát tanulmányozta olyan szubjektív esetekben, mint például hogy egy diák elvégez-e egy képzési programot, vagy egy bűnöző visszaesik-e. A megvizsgált 20 eset közül 19-nél Meehl azt találta, hogy egyszerű statisztikai tanulási algoritmusok (mint például a lineáris regresszió vagy a naiv Bayes-tanulás) jobb előrejelzéseket adnak, mint az emberi szakértők. Az Educational Testing Service a GMAT-vizsga esszékérdéseinek millióit osztályozza 1999 óta egy automatizált programmal. A program osztályzata az osztályzást végző emberek minősítésével az esetek 97%-ában egyezik meg, és ez az arány hasonló ahhoz, amennyire a különböző emberek által adott minősítések megegyeznek (Burstein és társai, 2001).

Világos, hogy sok mindent, köztük olyan dolgokat is, amelyekhez általános meggyőződésünk szerint komoly emberi éleselméjűség és értelem szükséges, a számítógépek ugyanolyan jól el tudnak végezni, mint az emberek, vagy akár még jobban is. Ez persze nem jelenti azt, hogy a számítógépek éleselméjűséget és értelmet

[2] Az orosz nyelvben az „úszni” megfelelője hajókra is vonatkozik. A magyar „úszni” ige is vonatkozhat hajókra, sőt, ismét csak az angoltól eltérően, még felhőkre is. (*A ford.*)

Created by XMLmind XSL-FO Converter.

tanúsítanának az ilyen dolgok végrehajtásakor: ezek nem részei *viselkedésüknek* (erről a kérdésről majd máshol beszélünk), hanem inkább arról van szó, hogy gyakran téves az első feltételezésünk az adott tevékenység elvégzéséhez szükséges mentális folyamatról. Ugyanakkor az is igaz, hogy számtalan feladatban a számítógépek (finoman szólva) nem járnak az élen, és ilyen Turing feladata, a nyíltvégű beszélgetések folytatása is.

1.2. A matematikai ellenvetés

Turing (Turing, 1936) és Gödel (Gödel, 1931) munkássága révén közismert, hogy egyes matematikai kérdések elviekben is megválaszolhatatlanok bizonyos formális rendszerekben. A leghíresebb példa erre Gödel nemteljességi tétele (lásd 9.5. alfejezet). Ez, röviden összefoglalva, azt mondja ki, hogy bármely F formális rendszerben, amelyben az aritmetika megfogalmazható, lehetséges egy $G(F)$ Gödel-mondatot konstruálni, amelyre igaz, hogy

- $G(F)$ egy mondat F-ben, de F-en belül nem bizonyítható.
- Ha F konzisztens, akkor $G(F)$ igaz.

Egyes filozófusok, mint például J. R. Lucas (Lucas, 1961) állítása szerint ez a tétel azt bizonyítja, hogy a gépek mentálisan alsóbbrendűek az embereknél, mert a gépeket mint formális rendszereket korlátozza a nemteljességi tétel (nem tudják megállapítani a saját Gödel-mondatuk igazságát), az embereket viszont ilyesmi nem korlátozza. Ez az állítás évtizedes, sok könyvet felölelő vitát keltett, például a matematikus Sir Roger Penrose két könyvében (Penrose, 1989; 1994) képviseli ezt az állítást kicsit megcsavarva (például azzal a feltételezéssel, hogy az emberek azért különböznek, mert agyukat a kvantumgravitáció működteti). Három problémát fogunk az állítással kapcsolatban megvizsgálni.

Először is Gödel nemteljességi tétele csak olyan formális rendszerekre vonatkozik, amelyek elég erősek az aritmetikához. Ezek közé tartozik a Turing-gép is, és Lucas állítása részben azon a feltételezésen alapul, hogy a számítógépek Turing-gépek. Ez jó közelítés, ám nem teljesen igaz. A Turing-gépek végtelenek, a számítógépek azonban végesek, és ezért bármely számítógépet le lehet írni egy nagyon nagy logikai állításrendszerrel, amelyre már nem vonatkozik Gödel nemteljességi tétele.

Másodszor is egy ágensnek nem kell túlzottan szégyenkeznie amiatt, hogy egyes állítások igazságát, másoktól eltérően, ő nem tudja eldönteni. Nézzük például a következő mondatot:

J. R. Lucas nem tudja konzisztensen azt állítani, hogy ez a mondat igaz.

Lucas ellentmondana önmagának, ha azt állítaná, hogy a fenti (teljes) mondat igaz, tehát Lucas nem tudja ezt a mondatot konzisztensen állítani, tehát a mondat mindenképpen igaz. (A mondat nem lehet hamis, mert ha hamis volna, akkor Lucas nem tudná konzisztensen állítani, tehát igaz lenne.) Bebizonyítottuk tehát, hogy van egy mondat, amelyet Lucas nem tud konzisztensen állítani, mások viszont (beleértve a gépeket is) igen. De ez előttünk semmit sem von le Lucas értékéből. Egyetlenegy ember sem tudja, hogy egy másik példát vegyünk, élete során kiszámítani tízmilliárd tízjegyű szám összegét, egy számítógép viszont másodpercek alatt megteszi ezt. Mégsem tartjuk ezt az emberi gondolkodási képesség alapvető korlátjának. Évezredeken át, mielőtt felfedezték volna a matematikát, az emberek ugyanúgy intelligens viselkedést tanúsítottak, nem valószínű hát, hogy a matematika a perifériálisnál fontosabb szerepet játszana az intelligencia meghatározásában.

Harmadszor és legfőképpen, még ha el is fogadjuk, hogy a számítógépek bizonyítási képessége korlátozott, semmi sem bizonyítja, hogy az emberek mentesek lennének az ilyen korlátozásoktól. Túlzottan is egyszerű szigorúan bebizonyítani, hogy egy formális rendszer képtelen X-re, majd minden további bizonyíték nélkül azt állítani, hogy az emberek a maguk nem formális módján *képesek* erre az X-re. Azt, hogy az emberek nincsenek alávetve Gödel nemteljességi tételének, valóban lehetetlen bebizonyítani, mert bármely szigorú bizonyításnak tartalmaznia kellene az állítás szerint formalizálhatatlan emberi tehetség egy formalizálását, tehát a bizonyítás önmagát cáfolná meg. Lehet még fellebbezni az intuícióhoz, miszerint az emberek valamilyen módon képesek a matematikai belátás emberfeletti cselekedeteire. Ezt a fordulatot fejezik ki az olyan érvek, mint például „feltételeznünk kell önnön konzisztenciánkat, hogy a gondolkodás egyáltalán lehetséges legyen” (Lucas, 1976). De ha bármiről is, akkor az emberi gondolkodásról mindenképpen tudjuk, hogy inkonzisztens. Ez egyértelműen igaz a mindennapi érvelésünkre, de vonatkozik még az elővigyázatos matematikai gondolkodásra is. Híres példa erre a négyszín-probléma. 1879-ben Alfred Kempe közzétett egy bizonyítást, amelyet széles körben elfogadtak, és amely hozzájárult, hogy a Royal Society tagjává választották. 1890-ben azonban Percy Heawood egy hibát fedezett fel a bizonyításban, és egészen 1977-ig senki nem tudta a tételt bebizonyítani.

Created by XMLmind XSL-FO Converter.

1.3. A meghatározatlanságból vett érv

A mesterséges intelligencia vállalkozásának legnagyobb hatású és legmakacsabb kritikáját Turing úgy mutatta be, mint „a viselkedés meghatározatlanságából származó érvet”. Az állítás lényege az, hogy az emberi viselkedés túl komplex ahhoz, hogy egy szabálykészlettel le lehessen írni, és mivel a számítógépek semmi másra nem képesek, csak szabályok követésére, ezért nem tudnak az emberekhez hasonló intelligens viselkedést létrehozni. Azt, hogy logikai szabályok egy halmazával képtelenség mindent leírni, a mesterséges intelligenciában **kvalifikációs problém**ának (**qualification problem**) nevezik (lásd 10. fejezet).

Ezt a nézetet főként a filozófus Hubert Dreyfus képviselte, aki számos befolyásos MI-kritika szerzője: *What Computers Can’t Do* (Dreyfus, 1972); *What Computers Still Can’t Do* (Dreyfus, 1992), valamint a testvérével, Stuarttal, közösen írt *Mind Over Machine* (Dreyfus és Dreyfus, 1986).

Azt az álláspontot, amit kritizáltak, Haugeland (Haugeland, 1985) nyomán elkezdték „Jófajta, régivágású MI”-nek nevezni (angol rövidítéssel: `GOFAI`). A `GOFAI` azt feltételezné, hogy minden intelligens viselkedés megragadható egy olyan rendszerrel, amely a tárgyterületet leíró tények és szabályok halmazaiból kiindulva logikai következtetést végez. Ezért a `GOFAI` a 7. fejezetben leírt legegyszerűbb logikai ágensnek felel meg. Dreyfusnak igaza van abban, hogy a logikai ágensek ki vannak téve a kvalifikációs probléma veszélyének. Amint a 13. fejezetben láttuk, a nyílt tárgyterületekhez sokkalta megfelelőbbek a valószínűségi következtető rendszerek. Dreyfus kritikája tehát nem *per se* a számítógépek, hanem csak a programozás egy meghatározott módja ellen irányul. Okkal feltételezhetjük persze, hogy nem lenne túl nagy hatású egy könyv *Amire az elsőrendű logikai szabályalapú, nem tanuló rendszerek képtelenek* címmel.

Dreyfus nézete szerint az emberi szakértelemnek ugyan része néhány szabály ismerete, de ezek csak „holisztikus kontextust” vagy „hátteret” alkotnak az emberi tevékenységhez. Példaként az ajándék adásakor és fogadásakor illő társas viselkedést hozza fel: „Általában az ember a megfelelő körülmények között úgy reagál, hogy egy megfelelő ajándékot ad.” Az embernek láthatólag „közvetlen érzéke van ahhoz, hogy miként történnek a dolgok, és mit lehet elvárni.” Hasonló állítást tehetünk a sakkjáték vonatkozásában is: „Egy átlagos sakkmesternek lehet, hogy gondolkodnia kell azon, mit tegyen, de egy nagymester látja, hogy a tábla egy bizonyos lépést követel (…) a helyes reakció egyszerűen csak beugrik a fejébe.” Minden bizonnyal igaz, hogy egy ajándékozó vagy egy sakknagymester gondolati folyamata azon a szinten zajlik, amely rejtett a tudatos elme önismerete elől. Ez azonban nem jelenti, hogy ez a gondolati folyamat nem létezik. Lényeges, hogy Dreyfus nem válaszolja meg azt, hogy a helyes lépés *miként* kerül a nagymester fejébe. Mindez Daniel Dennett (Dennett, 1984) megjegyzésére emlékeztet:

> Ez olyan, mintha a filozófusok azt tartanák magukról, hogy szakértő magyarázói a színpadi bűvészek trükkjeinek, aztán, amikor megkérdezzük tőlük, hogy milyen trükköt használnak a bűvészek a nő kettéfűrészelésénél, akkor azt a magyarázatot adják, hogy mindez nagyon is világos: a bűvész nem fűrészeli ketté a nőt, ezt csak színleli. Ha azt kérdezzük azonban: „Mégis *hogyan* színleli?”, a filozófusok ezt felelik: „Nem ránk tartozik.”
>
> —Daniel Dennett

A két Dreyfus (Dreyfus és Deyfus, 1986) vázolja a szakértelem ötlépcsős elsajátításának folyamatát, kiindulva a hasonló szabályalapú feldolgozásból (ahogyan az a `GOFAI`-ban is szerepel) eljutva a helyes reakció azonnali kiválasztásának képességéhez. A szerzők ezzel a javaslattal tulajdonképpen az MI kritikusaiból az MI elméletalkotóivá válnak: egy „eset-könyvtárakba” szervezett neurálisháló-architektúrát vázolnak fel, azonban rámutatnak néhány problémára is. Ezeket a problémákat szerencsére már mind célba vette az MI-kutatás néhányukat részleges, másokat pedig teljes sikerrel oldva meg. Nézzünk néhány ilyen problémát:

1. Háttértudás nélkül nem lehet jól példákból általánosítani. Azt állítják, senki sem tudja, miként lehetne a háttértudást beépíteni a neuronhálók tanítási folyamatába. Pedig, amint a 19. fejezetben mi is láttuk, léteznek módszerek az a priori tudás felhasználására a tanítási folyamat során. Ezek a módszerek persze a tudás explicit megadásán alapulnak, amelynek lehetőségét a szerzőpáros elszántan tagadja. Nézetünk szerint ez a probléma erős indok a neurális feldolgozás jelenlegi modelljeinek olyan újratervezésére, amely már a többi tanuló algoritmushoz hasonlóan *képes lesz* felhasználni a korábban megtanult tudást.

2. A neurális hálók tanulása a felügyelt tanulás egyik formája (lásd 18. fejezet), amely igényli, hogy előzetesen azonosítsuk a releváns bemeneteket, valamint a megfelelő kimeneteket. Ezért azt állítják, hogy nem is képesek ezek a hálózatok autonóm módon, emberi betanítás nélkül üzemelni. Valójában a **nem ellenőrzött tanulás** (**unsupervised learning**) (lásd 20. fejezet) és a **megerősítéses tanulás** (**reinforcement learning**) (lásd 21. fejezet) éppen a tanító nélküli tanulást teszi lehetővé.

Created by XMLmind XSL-FO Converter.

3. A tanuló algoritmusok nem teljesítenek jól sok jellemző esetén, viszont ha kijelöljük a jellemzők egy halmazát, „nem ismeretes olyan módszer, amellyel új jellemzőket illeszthetünk be, ha a jelenlegiek nem lennének elegendők a megtanult tényekhez”, valójában új eljárások, mint például a szupport vektor gépek, jól megbirkóznak a nagy jellemzőhalmazokkal is. A 19. fejezetben pedig láthattuk, hogy alapjaikat tekintve léteznek módszerek az új jellemzők bevezetésére, habár még sok tennivaló akad ezen a területen.

4. Az agy képes a szenzorait releváns információk keresésére irányítani, és képes kinyerni az információkból azt, ami releváns az adott szituációban. Ezzel szemben „jelenleg egyetlen részletét sem értjük ennek a működésmódnak – állítják –, és még csak olyan hipotézist sem tudunk alkotni róluk, amely útmutatásul szolgálhatna az MI-kutatóknak”. Valójában az aktív látás területe, amelyet az információérték-elmélet (lásd 16. fejezet) alapoz meg, éppen hogy a szenzorok irányításának problémájával foglalkozik, és az elért elméleti eredmények egy részét már át is ültették robotmegvalósításokba.

Összegezve azt mondhatjuk, hogy számos olyan kérdés, melyet Dreyfus felvetett – a köznapi háttértudás, a kvalifikáció probléma, a bizonytalanság kezelése, a tanulás, a döntéshozatal előre lefordított módjai, annak fontossága, hogy ne testetlen következtetőgépeket tekintsünk, hanem szituációjukba ágyazott ágenseket –, ma már bekerült az intelligens ágensek tervezésének szokványos menetébe. Véleményünk szerint mindez tehát nem a mesterséges intelligencia lehetetlenségét, hanem éppen a fejlődését bizonyítja.

2. Erős MI: Tudnak-e ténylegesen gondolkodni a gépek?

Több filozófus álláspontja szerint egy Turing-teszten átmenő gép nem gondolkodna ténylegesen, csupán szimulálná a gondolkodást. Ezt az ellenvetést is előre látta Turing. Geoffrey Jefferson (Jefferson, 1949) professzor egy beszédét idézi:

> Amíg nem képes egy gép egy szonettet írni vagy egy zenei versenyművet komponálni, pusztán a gondolatai és az érzései alapján, és nem szimbólumok véletlen alakulása folytán, addig nem fogunk beleegyezni, hogy egy gép egyenlő lehet az aggyal; ehhez az kell, hogy ne csupán megírja, de tudja is, hogy megírta.
>
> —Geoffrey Jefferson

Ezt nevezi Turing **tudatosság**i (**consciousness**) érvnek: a gépnek saját mentális állapotainak és cselekvéseinek tudatában kell lennie. Ugyan a tudatosság fontos témakör, Jefferson megállapítása azonban lényegileg a **fenomenológiá**hoz (**phenomenology**), a közvetlen tapasztalat tudományához kötődik: a gépnek ténylegesen éreznie kell az érzéseit. Más szerzők az **intencionalitás**ra (**intentionality**) koncentrálnak, ami azt kérdezi, hogy a gép feltételezett hiedelmei, vágyai és más reprezentációi ténylegesen valamilyen valósvilág-beli dologra „vonatkoznak”-e.

Figyelemre méltó Turing válasza erre az ellenvetésre. Felhozhatott volna érveket amellett, hogy a gépek valóban rendelkezhetnek tudattal (avagy viszonyulhatnak fenomenológiailag, illetve viselkedhetnek intencionálisan). Ehelyett Turing úgy érvel, hogy ez a kérdés éppen úgy rosszul lett megfogalmazva, mint a „Tudnak-e a gépek gondolkodni?” kérdés. Ráadásul miért kellene erősebb kritériumokat várni el a gépek esetében, mint az embereknél? A mindennapi életben végső soron nincsen *semmilyen* közvetlen bizonyítékunk a többi ember belső mentális állapotairól. Mégis, amint Turing rámutat, „ahelyett hogy állandóan erről vitatkoznánk, azzal az **udvarias feltételezés**sel (**polite convention**) élünk, hogy mindenki képes gondolkodni”.

Azonban Turing szerint Jefferson hajlandó lenne ezt az udvarias feltételezést a gépekre is kiterjeszteni, ha találkozna intelligensen cselekvő gépekkel. Az alábbi dialógust idézi, amely annyira beépült a mesterséges intelligencia szájhagyományába, hogy egyszerűen nem hagyhattuk ki:

> EMBER: Szonetted első sorában, mely úgy hangzik, hogy „hasonlítanálak téged a nyári naphoz”, nem lenne legalább ugyanolyan jó, ha „tavaszi nap” szerepelne?
>
> GÉP: Nem jönne ki az ütem.
>
> EMBER: De ha „téli nap” lenne? Ez ütemre ugyanolyan jó.
>
> GÉP: Persze, de hát senki sem szeretné, ha egy téli naphoz hasonlítanák.

Created by XMLmind XSL-FO Converter.

> EMBER: Mondanál olyat, hogy Pickwick úr a karácsonyra emlékeztet téged?
>
> GÉP: Bizonyos értelemben igen.
>
> EMBER: A karácsony viszont egy téli nap, és én nem gondolom, hogy Pickwick úrnak ellenére lenne az összehasonlítás.
>
> GÉP: Nem hiszem, hogy ezt komolyan gondolod. A téli nap alatt egy átlagos téli napot szokás érteni, nem pedig egy olyan különlegeset, mint a karácsony.
>
> —Turing

Azt ugyan elismeri Turing, hogy a tudatosság megmagyarázása nehéz kérdés, de tagadja, hogy bármi jelentősége is volna a mesterséges intelligencia gyakorlata számára: „Nem szeretném azt a benyomást kelteni, miszerint úgy gondolnám, hogy a tudatosságban ne volna semmi rejtélyes elem. (…) De nem gondolom azt, hogy ezeket a rejtélyeket meg kellene oldani ahhoz, hogy meg tudjuk válaszolni a cikkemben tárgyalt kérdéseket.” Egyetérthetünk Turinggal, hiszen az a fontos számunkra, hogy intelligensen viselkedő programokat készítsünk, nem pedig az, hogy mások azt vajon szimulált vagy valóságos intelligenciának tartják. Persze másfelől sok filozófusnak igencsak fontos ez a kérdés. Ezt megvilágítandó vegyük szemügyre, hogy más eseteken mennyire tartják valóságosnak az ember készítette tárgyakat.

A mesterséges karbamidot 1828-ban sikerült először előállítania Fredrick Wöhlernek. A szintézis fontosságát az adta, hogy bizonyítékot szolgáltatott a szerves és szervetlen kémia akkortájt sokak által vitatott egységére. A vegyület előállítása után a kémikusok egyetértettek abban, hogy a mesterséges karbamid valódi karbamid, hiszen az összes megfelelő fizikai jellemzőben megegyeznek. Ehhez hasonlít, hogy a mesterséges édesítőket ugyanúgy édesítőknek nevezzük, és a mesterséges megtermékenyítés (a mesterséges intelligencia másik módja) ugyanolyan megtermékenyítés. Azonban a mesterséges virágok nem virágok, és, amint Daniel Dennett rámutat, a mesterséges Chateau Latour bor sem lenne a különleges Chateau Latour francia bor, még akkor sem, ha kémiailag megkülönböztethetetlen volna, mivel nem a rögzített helyen (a francia Bordeaux-i borvidéken) és nem a rögzített módon készült. Ugyanígy egy mesterséges Picasso-festmény – akárhogy is néz ki – nem volna igazi Picasso-festmény.

Megállapíthatjuk tehát, hogy bizonyos esetekben az ember készítette tárgy viselkedése fontos, más esetekben pedig a származása. Úgy látszik, megállapodás kérdése, hogy melyik esetben melyiket tartjuk fontosnak. Mesterséges elmék esetében nincsen ilyen megállapodás, így az intuíciónkra kell hagyatkoznunk. Egy filozófus, John Searle (Searle, 1980), elég határozott intuícióval rendelkezik:

> Senki sem gondolja, hogy egy vihar számítógépes szimulációjától eláznánk. (…) Mi visz rá tehát embereket annak feltételezésére, hogy a mentális folyamatok számítógépes szimulációja mentális folyamatokat mutatna fel?
>
> —(Searle, 1980) (37–38. oldal)

Könnyen elfogadható persze, hogy a viharok számítógépes szimulációi senkit sem áztatnak el, az azonban kevésbé látható, miként lenne alkalmazható ez az analógia a mentális folyamatok számítógépes szimulációira. Hiszen amikor Hollywoodban locsolók és szélgépek segítségével szimulálják a vihart, eláznak a színészek. A legtöbben határozottan összeadásnak tartják az összeadás számítógépes szimulációját, és a sakkjáték számítógépes szimulációja is sakkjáték. A mentális folyamatok vajon a viharokhoz hasonlíthatók-e, vagy inkább az összeadáshoz és a sakkhoz? Olyanok, mint a Chateau Latour bor és Picasso, vagy mint a karbamid? Ez teljes egészében azon múlik, hogy milyen elméletet vallunk a mentális állapotokról és folyamatokról.

A **funkcionalizmus** (**functionalism**) szerint a mentális állapotok köztes kauzális feltételek a bemenetek és a kimenetek között. A funkcionalista elmélet szerint, ha két rendszer izomorf kauzális folyamatokkal rendelkezik, akkor ugyanazok a mentális állapotai is. Egy számítógépes programnak tehát ugyanolyan mentális állapotai lehetnek, mint egy embernek. Azt persze még nem határoztuk meg, hogy az „izomorf” pontosan mit jelent, de azt tételezzük fel, hogy létezik egy olyan absztrakciós szint, ami fölött nem számít a specifikus implementáció; és ha a folyamatok eddig a szintig izomorfak, akkor ugyanazok lesznek a mentális állapotok is.

Ezzel szemben a **biológiai naturalizmus** (**biological naturalism**) szerint a mentális állapotok magas szintű emergens tulajdonságok, amelyeket a *neuronok* alacsony szintű neurológiai folyamatai okoznak, azaz a neuronok (meghatározatlan) tulajdonságai a lényegesek. A mentális állapotokat így tehát nem lehet duplikálni valamilyen programmal, amelynek azonos a bemenet–kimenet funkcionális struktúrája, hanem az is szükséges, hogy a program olyan architektúrán fusson, amelynek kauzális erői megegyeznek a neuronokéval. Az elmélet

Created by XMLmind XSL-FO Converter.

nem mond semmit arról, hogy a neuronok miért rendelkeznek ilyen kauzális erővel, és arról sem, hogy más fizikai megtestesülések rendelkezhetnek-e ezekkel az erőkkel.

A két álláspont megvizsgálásához vegyük szemügyre először az elmefilozófia legrégibb problémáját, majd pedig három gondolatkísérletet.

2.1. A test–elme probléma

A **test–elme probléma** (**mind–body problem**) azt kérdezi, hogy a mentális állapotok és folyamatok miként kapcsolódnak a testi (specifikusan az agyi) állapotokhoz és folyamatokhoz. Ezt a kérdést, mintha nem lenne elég nehéz, általánosítani fogjuk az „architektúra–elme” problémává, hogy az elmével rendelkező gépek lehetőségéről beszélhessünk.

Egyáltalán mi a probléma a test–elme problémában? Az első nehézség René Descartes-ra vezethető vissza, aki egy nem halandó lélek interakcióját vizsgálta egy halandó testtel, és egy **dualista** (**dualist**) elméletet állított fel, miszerint a lélek és a test két különböző típusú dolog. A **monista** (**monist**), gyakran **materialista** (**materialism**) névvel is illetett elmélet szerint nincs olyan dolog, hogy halhatatlan lélek, hanem csupán az anyagi objektumok léteznek. Ennek megfelelően a mentális állapotok (mint például a fájdalom, a lovaglás tudata vagy az a meggyőződés, hogy Bécs Ausztria fővárosa) csupán agyállapotok. Ezt az elméletet John Searle egy tömör szlogenben foglalta össze: *„Az agy okozza az elmét.”*

A materialistának legalább két komoly ellenvetéssel kell szembesülnie. Az első a **szabad akarat** (**free will**) problémája: hogyan lehetséges, hogy egy tisztán fizikai elme, amelynek minden átalakulását szigorú fizikai törvények irányítják, mégis megőrzi a döntés szabadságát? A legtöbb filozófus szerint ez a probléma ahelyett, hogy a materializmusnak jelentene kihívást, inkább a szabad akaratról alkotott naiv fogalmunk gondos újraalkotását igényli. A második probléma a **tudatosság** (**consciousness**) általános témaköréhez kapcsolódik (a probléma hasonló, mint a **megértés** (**understanding**) és az **öntudat** (**self-awareness**) kérdései, habár nem teljesen ugyanarról van szó). A kérdés egyszerűen ez: miért van, hogy bizonyos agyállapotokkal rendelkezni valamiféle *érzés,* viszont másfajta fizikai állapotokkal rendelkezni (például sziklának lenni) feltehetően nem érzés.

Ahhoz, hogy elkezdhessük megválaszolni a kérdést, az agyállapotok egy olyan megfogalmazását kell megtalálni, amely absztraktabb szintű, mint az egy adott személy agyát alkotó atomok adott időpontbeli fizikai elhelyezkedése. Például miközben Ausztria fővárosára gondolok, agyam minden pikoszekundumban milliárd apró változáson megy keresztül, mégis ezek a változások nem jelentik az agyállapot *kvalitatív* változását. Ennek megfelelően be kell vezetnünk az agyállapot *típusok* fogalmát, amelynek segítségével elkülöníthetők a különböző típusba tartozó agyállapotok. Az egyes szerzők különbözőképpen értik a *típus* szót ebben az esetben. A legtöbbjük szerint, ha például veszünk egy agyat, és néhány szénatomját más szénatomra[3] cseréljük, a mentális állapot még nem változik meg. Ez azért örvendetes, mert valójában az agy atomjai az anyagcsere-folyamatok során állandóan cserélődnek, mégsem tűnik úgy, mintha ez nagy mentális felfordulást okozna.

Vegyük most szemügyre a mentális állapotok egy fajtáját, az **intencionális állapot**oknak (**intentional state**s) is nevezett **ítéletlogikai attitűd**öket (**propositional attitude**s) (amelyekről már szó esett a 10. fejezetben). Ezek az állapotok – mint például a vélekedés, a tudás, a vágyakozás, a félelem és társai – olyanok, hogy bizonyos szempontból a külső világra utalnak. Az a vélekedés például, miszerint Bécs Ausztria fővárosa, egy bizonyos városra és annak státusára vonatkozó vélekedés. Azt fogjuk vizsgálni, hogy rendelkezhetnek-e a számítógépek intencionális állapotokkal, ezért hasznos áttekinteni ezen állapotok jellemzését. Azt is lehet mondani például, hogy az a mentális állapot, hogy egy hamburgerre vágyom, különbözik attól az állapottól, hogy egy pizzát szeretnék, hiszen a hamburger és a pizza a valódi világ különböző tárgyai. Másfelől viszont amellett érveltünk éppen néhány bekezdéssel korábban, hogy a mentális állapotok agyállapotok, tehát az azonosságukat meg kellene tudnunk határozni csupán „a fejen belül” maradva, minden hivatkozás nélkül a külső világra. A dilemma megoldásához vizsgáljunk meg egy olyan gondolatkísérletet, amely megpróbálja az intencionális állapotokat elkülöníteni a hozzájuk tartozó külső tárgyaktól.

2.2. Az „agy a tartályban” kísérlet

Képzeljük el – ha belebocsátkozunk ebbe a gondolatkísérletbe –, hogy az agyunkat születésünkkor kiveszik a testünkből, és egy csodálatos szerkezetű tartályba helyezik. Ez a tartály táplálja az agyat, lehetővé téve, hogy

[3] Esetleg a szénatom más izotópjára, ahogyan ez az agyi képalkotó vizsgálatok során gyakran történik.

Created by XMLmind XSL-FO Converter.

növekedjen és fejlődjön. Ugyanakkor azonban egy teljességgel kitalált világ számítógépes szimulációjának elektromos ingereit vezetik az agyba, az agy motorikus ingereit pedig elfogják, és hatásukra megfelelően módosítják a szimulációt.[4] Ekkor az agy rendelkezhet a *HalálosanVágyik*(*Én*, *Hamburger*) mentális állapottal, akkor is, ha nincs is teste, amely az éhséget érezné, ízlelőbimbói sincsenek, sőt talán egyáltalán nincs is hamburger a valódi világban. Azonosnak tekintjük-e ezt a mentális állapotot egy testben lévő agy megfelelő mentális állapotával?

Az egyik kiút a dilemmából annak megállapítása, hogy a mentális állapotok tartalma két különböző szempontból értelmezhető. A **tág tartalom** (**wide content**) nézet képviselői a mentális állapotok tartalmát a mindentudó külső megfigyelő szemszögéből értelmezik, amely megfigyelő átlátja az egész szituációt, és felismeri a világ különbségeit. Tehát a tág tartalom nézet szerint a tartályba helyezett agy vélekedései különbözők a „normál" személy vélekedéseitől. A **szűk tartalom** (**narrow content**) nézet szerint viszont csak a belső, szubjektív nézőpontot kell figyelembe venni, és szerintük a két vélekedés megegyezik.

Annak a vélekedésnek, hogy a hamburger ízletes, benső természete van: van valami, amihez az ilyen vélekedés hasonlít. Most már beléptünk a **qualiá**k, a benső élmények birodalmába (a *qualia* szó a latin nyelvből származik, megközelítő jelentése: 'ilyen dolgok'). Tegyük fel, hogy a retina és az idegrendszer valamilyen sérülése folytán *X* személy úgy érzékeli a vörös színt, mint *Y* a zöldet, és *vice versa*. Ebben az esetben ugyanúgy látják a közlekedési lámpát, és ugyanúgy is cselekszenek, de *tapasztalataik* valamiképpen különbözni fognak. Mindketten elfogadhatják, hogy tapasztalatuk neve: „a lámpa vörös", de tapasztalatuk különböző érzés lesz. Nem világos, hogy mit jelent ez a mentális állapotok megegyezését illetően. A következőkben egy olyan gondolatkísérlettel foglalkozunk, amely azt vizsgálja, hogy az emberi neuronokon kívül más fizikai tárgyaknak is lehetnek-e mentális állapotai.

2.3. Az agyprotézis-kísérlet

Az agyprotézis-kísérletet Clark Glymour vezette be a hetvenes évek közepén, majd John Searle (Searle, 1980) is finomított rajta, de manapság leginkább Hans Moravec (Moravec, 1988) munkásságával szokás összefüggésbe hozni. A kísérlet a következő. Tegyük fel, hogy a neurofiziológia eljutott odáig, hogy tökéletesen megértette az emberi agy összes neuronjának bemeneti–kimeneti jellemzőit és összekapcsolódását. Tegyük fel továbbá azt is, hogy képesek vagyunk olyan mikroszkopikus méretű elektronikus berendezéseket építeni, amelyek utánozni tudják ezeket a jellemzőket, és problémamentesen hozzáilleszthetők az idegszövethez. Végül pedig tegyük fel azt is, hogy valamilyen csodálatos sebészi technikával anélkül tudjuk az egyes neuronokat a megfelelő elektronikus eszközökkel helyettesíteni, hogy az agy egészében vett működése megszakadna. A kísérlet abból áll, hogy valaki fejében az összes neuront elektronikus eszközökkel helyettesítik, majd pedig a folyamat visszafordításával visszajuttatjuk az illetőt a normális biológiai állapotába.

A személynek mind a műtét során tanúsított külső viselkedése, mind belső tapasztalatai érdekelnek minket. A kísérlet alanyának külső viselkedésének definíció szerint változatlannak kell maradnia az operáció nélkül megfigyelhető viselkedéshez képest.[5] A kísérlet alanyának azonban egyben képesnek kell lennie arra, hogy – noha a tudatosság hiányát vagy jelenlétét egy harmadik személy nehezen állapíthatja meg – beszámolót adjon a tudatos tapasztalatát ért változásokról. A történtekre vonatkozó különböző intuícióink nyilván ellentmondanak egymásnak. Moravec szerint, aki robotkutató és funkcionalista, változatlan maradna a tudata. A filozófus és biológiai naturalista Searle azonban ugyanilyen erősen vallja, hogy a tudata eltűnne:

> Azt fogod észrevenni – legnagyobb megdöbbenésedre –, hogy a külső viselkedésed irányítása mintegy kicsúszik a kezeid közül. Azt veszed észre, hogy például amikor az orvosok vizsgálják a látásodat, hallod őket, amint azt mondják: „Egy piros tárgyat tartunk ön előtt, kérjük, mondja el, mit lát." Ez a kiáltás törne ki belőled: „Semmit sem látok. Teljesen megvakulok." De hallod a saját hangodat, amint befolyásodtól teljesen függetlenül ezt mondja: „Egy vörös tárgyat látok magam előtt." (…) Tudatos tapasztalatod lassan semmivé zsugorodik, de a külső, megfigyelhető viselkedésed változatlan marad.
>
> —(Searle, 1992)

Mindannyian csak intuíciónk alapján tudunk érvelni. Először vegyük észre, ahhoz, hogy a külső viselkedés változatlan maradjon, mialatt a kísérlet alanya fokozatosan elveszti a tudatát, az akaratnak egyik pillanatról a másikra hirtelen és teljesen kell eltűnnie; különben a tudat zsugorodása befolyásolná a külső viselkedést – például ilyesmi szavak hangzanának el: „Segítség, eltűnőben vagyok!" Elég valószínűtlenül hangzik azonban,

[4] Ez a helyzet ismerős lehet azoknak, akik látták az 1999-ben készült *Mátrix* (The Matrix) c. filmet.
[5] El tudunk képzelni egy azonos kontrollszemélyt, akin egy placebooperációt hajtanak végre, hogy a két viselkedést összevethessék.

Created by XMLmind XSL-FO Converter.

hogy a neuronok egyenkénti lecserélése az akarat hirtelen, egyik lépésről a másikra történő teljes eltűnéséhez vezetne.

Másodszor pedig nézzük meg, mi történik akkor, ha a kísérlet alanyát az alatt az idő alatt kérdezzük tudatos tapasztalatairól, amikor egyetlen igazi neuron sincs jelen az agyában. A kísérlet feltétele szerint olyasféle válaszokat kell kapnunk, mint „Jól érzem magam. Meg kell mondanom, egy kicsit meg vagyok lepődve, hiszen elhittem, amit Searle mondott". Vagy pedig egy mutatópálcával egy kicsit megszúrhatjuk a kísérleti alanyt és megfigyelhetjük válaszát: „Juj, ez fáj." Viszont normál körülmények között a mesterséges intelligencia programok ilyen kimeneteit a szkeptikus elutasíthatja, mint egyszerű látszatválaszokat. Hiszen elég könnyű elképzelni egy olyan szabályt, mint például: „Ha a 12-es szenzoron »Magas« érték jelenik meg, adj ki »Juj!-t«. A lényeg azonban az, hogy mivel a normál emberi agy összes funkcionális tulajdonságát átmásoltuk, feltételezhetjük, hogy elektronikus agyunk nem tartalmaz ilyen látszatválasz-mechanizmusokat. Azaz az elektronikus agy által létrehozott tudati megnyilvánulásokra olyan magyarázatot kell adnunk, amely csak a neuronok funkcionális tulajdonságaira hivatkozik. *Ennek a magyarázatnak azonban a valós agyra is vonatkoznia kell, amelynek ugyanazok a funkcionális tulajdonságai.* Ezért, úgy tűnik, kétfajta konklúzió lehetséges.

1. A tudat kauzális mechanizmusai, amelyek ezt a kimenetet a normális agyban létrehozzák, ugyanúgy működnek az elektronikus változatban is, amely tehát tudatos.

2. A normális agy tudatos mentális eseményeinek semmilyen kapcsolata nincs a viselkedéssel, valamint nincsenek is jelen ezek az események az elektronikus agyban, amely tehát nem tudatos.

A második lehetőséget ugyan nem zárhatjuk ki, de ez arra redukálná a tudatot, amit a filozófusok **epifenomenális** (**epiphenomenal**) szereplőnek neveznek: valami: ami megtörténik ugyan, de mintha nem is vetne árnyákot a megfigyelhető világban. Valamint ha a tudat tényleg epifenomén, akkor az agyban lennie kell egy második, nem tudatos mechanizmusnak, amelyhez a „Juj!" tartozik.

Harmadszor pedig tekintsük az operáció visszacsinálása utáni helyzetet, amikor a kísérlet alanyának ismét normális agya van. Emlékezzünk arra, hogy a definíció szerint a kísérleti alany külső viselkedésének olyannak kell lennie, mintha az operáció nem is történt volna meg. Ennek speciális eseteként fel kell tudnunk tenni azt a kérdést: „Milyen volt az operáció alatt? Emlékszik a mutatópálcára?" A kísérleti alanynak pontos emléknyomokkal kell rendelkeznie a tudatos tapasztalatai tényleges természetéről, beleértve a qualiát is; mindannak ellenére, hogy Searle szerint nem is voltak ilyen tapasztalatai.

Searle talán azzal válaszolna, hogy nem jól definiáltuk a kísérletet. Ha a valódi neuronokat visszahelyezésükkor úgy keltjük életre, ami az eltávolításuk óta eltelt időt nem tükrözi, akkor természetesen nem fognak „emlékezni" az operáció alatti tapasztalatokra. Hogy erre az eshetőségre is felkészüljünk, a visszahelyezéskor a mesterséges neuronok belső állapotának megfelelően módosítanunk kell az őket helyettesítő neuronok állapotát. Ha a valódi neuronok feltételezett „nem funkcionális" aspektusai ekkor egy olyan viselkedést eredményeznének, amely funkcionálisan különbözik attól, mint amely a mesterséges neuronok bent hagyásával megfigyelhető lenne, akkor *reductio ad absurdum*hoz jutnánk, mert ez azt jelenti, hogy a mesterséges neuronok funkcionálisan nem ekvivalensek a valódi neuronokkal. (A 26.3. feladat egy lehetséges utat mutat ezen érv cáfolatára.)

Patricia Churchland (Churchland, 1986) rámutat arra, hogy ha a funkcionalista érv működik a neuronok szintjén, akkor működnie kell bármely nagyobb funkcionális egység szintjén is, legyen az akár neuronok egy halmaza, egy mentális modul, egy lebeny, egy agyfélteke vagy akár az egész agy. Ezek szerint, ha elfogadjuk, hogy az agyprotézis kísérlet tanulsága a tudat megmaradása az agy kicserélése során, akkor azt is el kell fogadnunk, hogy a tudat akkor is megmarad, ha az egész agyat lecserélik egy áramkörre, amely csak annyit tesz, hogy a bemenetekhez tartozó kimeneteteket visszakeresi egy óriási táblázatban. Ez zavarba hozza a legtöbb embert (beleértve magát Turingot is), akik intuíciója szerint a visszakereső táblák nem tudatosak, vagy legalábbis az általuk generált tudati tapasztalatok nem azonosak azokkal a tapasztalatokkal, amelyeket egy olyan rendszer generálna, amit le lehetne írni úgy (akár csak az egyszerűen vett számítási értelemben), mint ami hiedelmeket, önismeretet, célokat és hasonlókat tart nyilván és hoz létre. Mindez sugallja, hogy ha meg akarjuk őrizni az agyprotézis-kísérletet mint az intuíciónkat vezérlő segédeszközt, akkor nem folyamodhatunk az agy egy lépésben történő lecseréléséhez, de azt nem jelenti, hogy kénytelenek lennénk a lépésenkénti neuroncseréhez folyamodni, amit Searle szeretne velünk elérni.

2.4. A kínai szoba

Created by XMLmind XSL-FO Converter.

Legutolsó gondolatkísérletünk valószínűleg egyben a leghíresebb is. John Searle (Searle, 1980) gondolta ki, és egy olyan hipotetikus rendszert ír le, amely látnivalóan egy programot futtat, és megfelel a Turing-teszten; de ugyanilyen világos az is (legalábbis Searle szerint), hogy semmit sem *ért meg* a bemenetei és kimenetei közül. Konklúziója szerint tehát a megfelelő program futtatása (azaz a megfelelő válaszok megléte) önmagában még nem *elégséges* feltétele az elmemivoltnak.

A rendszer egy csak angolul értő emberből áll, akinek egy angol nyelven írt szabálykönyve és többfajta papírhalmaza van, melyek közül némely papírhalmaz üres, némelyen pedig valamilyen érthetetlen jelsorozat áll. (Tehát az ember a CPU szerepét játssza, a szabálykönyv a programnak, a papírhalmazok pedig a tárolóeszköznek felelnek meg.) A rendszer egy szobában található, amelynek csak egy kis nyílása van a külvilágra. Ezen a nyíláson keresztül érthetetlen feliratú papírcetlik jelennek meg. Az ember megtalálja a szabálykönyvben ezeket a szimbólumokat, és követi az ott leírtakat. Az utasítások között szerepelhet, hogy írjon le szimbólumokat új papírcédulákra, keressen szimbólumokat a papírhalmazokban, rendezze át a halmazokat és hasonlók. Végül, az utasítások következményeként, egy vagy több szimbólumot egy papírdarabra másol, és kiadja a külvilágba.

Fontos

Eddig minden könnyen követhető. De kívülről nézve azt látjuk, hogy a rendszer kínai mondatokat fogad, kínai mondatokat ad vissza, és ez a párbeszéd nyilvánvalóan éppannyira intelligens, mint amit Turing elképzelt.[6] Searle ennek alapján így érvel: a szobában lévő személy nem ért kínaiul (ez adott). A szabálykönyv és a papírcetlik, lévén ezek csak papírok, szintén nem értenek kínaiul. Ebből az következik, hogy a kínai nyelv semmiféle megértése nem zajlik le. *Tehát Searle szerint a megfelelő program futtatása nem vezet szükségszerűen a megértéshez.*

Turinghoz hasonlóan Searle is megvizsgált néhány lehetséges ellenvetést az érvével szemben, és meg is próbálta visszautasítani ezeket. Több kommentátor, köztük John McCarthy és Robert Wilensky, azt hozta fel, amit Searle rendszerválasznak nevez. Az ellenvetés abban áll, hogy ugyan feltehető az a kérdés, miszerint a szobában lévő személy ért-e kínaiul, de ez olyan, mint ha azt kérdeznénk, hogy a CPU tud-e köbgyököt vonni. Mindkét esetben nemmel kell felelnünk, de mindkét esetben – állítják a rendszerválasz hívei – a rendszer egésze *igenis* rendelkezik a kérdéses képességgel. Az biztos, hogy ha megkérdezzük a kínai szobát, ért-e kínaiul, a válasz igenlő lesz (méghozzá folyékony kínaisággal). Turing *udvarias feltételezése* szerint ennyi elegendő is. Válaszában Searle ismételten rámutat arra, hogy sem az ember, sem a papírhalmaz nem tud kínaiul, tehát semmilyen kínaiul értés sem lehetséges. Továbbá hozzáteszi, az is elképzelhető, hogy az ember megtanulja a szabálykönyvet, és a cetlik tartalmát is a fejében tartja, tehát csupán az ember az, aki kínaiul tudhatna, ha azonban megkérdezzük őt (angolul), akkor a válasza nemleges lesz.

Most már a lényegi kérdésnél vagyunk. A papírok tartalmának memorizálása csak elterelő hadművelet, hiszen mindkettő ugyanúgy egy futó program fizikai megtestesülése. Searle igazi állítása a következő négy axiómán alapszik (Searle, 1990):

1. A számítógépprogramok formális, szintaktikai entitások.

2. Az elmének mentális tartalma, azaz szemantikája van.

3. A szintaktika önmagában még nem elégséges a szemantikához.

4. Az elmét az agy okozza.

Az első három axiómából arra a következtetésre jut, hogy a programok nem elégségesek az elméhez. Másként megfogalmazva: ugyan lehet elme egy programot futtató ágens, de nem lesz szükségszerűen elme csupán a program futtatása miatt. A negyedik axiómából pedig arra a következtetésre jut, hogy „bármely más rendszernek, amely képes elmét okozni, az aggyal (legalább) megegyező kauzális erőkkel kell rendelkeznie”. Ebből pedig arra jut, hogy bármely mesterséges agynak nem elég csupán egy adott programot futtatni, hanem duplikálnia kell az agy kauzális erőit, valamint hogy az emberi agy a mentális jelenségeket nem csupán egy program futtatása révén hozza létre.

[6] Az, hogy a papírhalmaz talán nagyobb, mint az egész bolygó, és hogy a válaszok létrehozása millió évekig eltarthat, az érv *logikai* struktúráját nem érinti. A filozófiai képzés egyik célja éppen az, hogy kialakuljon egy kifinomult érzék, amellyel különválaszthatók a releváns és az irreleváns ellenvetések.

Created by XMLmind XSL-FO Converter.

Az a végkövetkeztetés, hogy az elmékhez nem elégségesek a programok, *igenis* következik az axiómákból (amennyiben elég nagyvonalúan értelmezzük azokat). De a konklúzió nem elégít ki bennünket: Searle csupán azt bizonyította be, hogy ha kifejezetten elutasítjuk a funkcionalizmust (ahogy ezt a 3. axiómájával teszi), akkor nem lehet szükségszerűen állítani azt, hogy a nem-agyak is elmék volnának. Ez teljesen elfogadható, így hát az egész érv arra jut, hogy elfogadható-e a 3. axióma. Searle szerint a kínai szoba érvek éppen az a lényege, hogy intuíciót ad a 3. axióma elfogadásához. De az érvére adott reakciók azt mutatják, hogy csak azoknak nyújt intuitív alapot, akik már eleve hajlanak annak elfogadására, hogy a programok önmagukban nem hozhatnak létre igazi megértést.

Megismételve tehát, a kínai szoba érv célja az, hogy megcáfolja az erős MI-hipotézist: azt az állítást, miszerint a helyes változatú program futtatása szükségszerűen elmét eredményez. Úgy működik az érv, hogy bemutat egy látszólag intelligens, a megfelelő típusú programot futtató rendszert, amely Searle szerint *bizonyíthatóan* nem elme. Ennél a résznél azonban Searle az intuícióra és nem egy bizonyítékra támaszkodik: nézzük csak a szobát, hol lenne itt az elme? De ez az érv ugyanúgy felhozható az aggyal kapcsolatban is: nézzünk rá a sejtek (vagy az atomok) sokaságára, amelyek vakon működnek a biokémia (vagy a fizika) törvényei szerint – hol lenne itt az elme? Miért tud egy nagy darab agy elme lenni, amikor egy nagy darab máj nem?

Továbbmenve, amikor Searle beismeri, hogy elméletileg a neuronoktól különböző anyag is lehet elme, akkor két okból is tovább gyengül az érve: egyrészt csak Searle intuícióira (vagy a saját intuíciónkra), támaszkodhatunk, hogy eldöntsük, elme-e a kínai szoba; másrészt, még ha úgy is döntünk, hogy a szoba nem elme, ez semmit nem mond arról, hogy a programot futtató más fizikai közeg (például a számítógép) lehet-e elme.

Searle megengedi azt a logikai lehetőséget, hogy az agy éppenséggel egy hagyományos típusú logikai programot valósít meg; de ugyanazt a programot egy rosszfajta gépen futtatva nem kapnánk elmét. Searle tagadja, hogy azt gondolná, „a gépeknek nem lehet elméje”, sőt azt állítja, hogy bizonyos gépeknek *igenis* van elméje: az emberek elmével rendelkező biológiai gépek. Nem hagy azonban túl sok útmutatást afelől, hogy az elmével rendelkezés a gépek milyen csoportjára terjed ki.

3. A mesterséges intelligencia fejlesztésének etikai kérdései és kockázatai

Mindeddig arra összpontosítottunk, hogy *képesek vagyunk-e* kifejleszteni egy mesterséges intelligenciát, most azt is meg kell vizsgálnunk, hogy *kell-e* ezt tennünk. Amennyiben az MI-technológiák hatásai többségében inkább negatívak, mint pozitívak lennének, akkor morális felelőssége lenne az ezen a területen dolgozóknak, hogy más területekre helyezzék kutatásukat. Sok új technológia járt akaratlanul is negatív mellékhatásokkal: a belső égésű motor a légszennyezést hozta és azt, hogy még az Édenkertet is leaszfaltozták, a maghasadás felfedezése pedig a csernobili és a Three Mile Island-i[7] atomkatasztrófákat hozta, és a globális pusztulás veszélyének fenyegetését. Minden tudósnak és mérnöknek szembe kell néznie a munkájukkal kapcsolatos etikai kérdésekkel: mely projekteket szabad befejezni, és melyeket nem, és hogyan kell ezeket a projekteket kezelni. Még egy *Ethics of Computing* (Berleur és Brunnstein, 2001) c. kézikönyv is létezik. A mesterséges intelligencia azonban, úgy tűnik, néhány új problémát is felvet, azon túl, hogy olyan hidakat akarunk építeni, amelyek nem dőlnek össze:

- Az emberek az automatizáció miatt elveszíthetik a munkájukat.
- Az embereknek túl sok (vagy túl kevés) szabadidejük marad.
- Az emberek elveszíthetik az egyediségérzésüket.
- Az emberek elveszíthetik a személyiségi jogaik egy részét.
- Az MI-rendszerek alkalmazása megszüntetheti a felelősségre vonhatóságot.
- A mesterséges intelligencia sikere az emberi faj végét jelentheti.

Sorra vesszük mindegyik felvetett problémát.

[7] Az Egyesült Államok eddigi legsúlyosabb atombalesetének helyszíne Pennsylvania államban. Az 1979-ben történt baleset ugyan rendkívül megrázta az amerikai közvéleményt, de káros egészségügyi következményeit tekintve nem mérhető a csernobili esethez. (*A ford.*)

Created by XMLmind XSL-FO Converter.

Az emberek az automatizáció miatt elveszíthetik a munkájukat. A modern ipari gazdaság általában véve függővé vált a számítógépektől, sőt néhány különleges MI-programtól is. A gazdaság nagy része például, különösen az Egyesült Államokban, a fogyasztói hiteltől függ. A hitelkártya-kérelmek elbírálását, a lehívások engedélyezését és a visszaélések felderítését napjainkban mesterséges intelligencia programok végzik. Azt is mondhatná valaki, hogy ezek a programok több ezer dolgozót tettek munkanélkülivé, de valójában, ha elvennénk ezeket a mesterséges intelligencia programokat, nem is léteznének ilyen állások, mert az emberi munka elviselhetetlen többletköltséget róna ezekre a tranzakciókra. A mesterséges intelligenciát használó gépesítés mindeddig több állást hozott létre, mint ahányat megszüntetett, és ezek az új állások érdekesebbek, és jobban fizetnek. Ma már, amikor a mesterséges intelligencia programokat az embereket segítő „intelligens ágenseknek" tekintik, kevesebben félnek a munkahelyük elvesztésétől, mint amikor a mesterséges intelligencia célja az embereket helyettesítő „szakértői rendszerek" létrehozása volt.

Az embereknek túl sok (vagy túl kevés) szabadidejük marad. Alvin Toffler a *Future Shock*-ban (Toffler, 1970) ezt írta: „A századforduló óta a munkahét 50%-kal csökkent. Nem lóg ki a sorból ha azt jósoljuk, hogy 2000-re ismét felével csökken." Arthur C. Clarke (Clarke, 1968b) azt írta, hogy 2001-ben az emberek talán már a „teljes unalommal néznek szembe, ahol az élet legnagyobb problémája a több száz tv-csatorna közti választás lesz". Az egyetlen ezen jóslatok közül, amelyik közel áll a megvalósuláshoz, az a tv-csatornák száma (Springsteen, 1992). Valójában a tudásintenzív iparágakban dolgozó emberek egy 24 órában működő, integrált számítógépesített rendszer részévé válnak, és ahhoz, hogy lépést tartsanak, *hosszabb* munkaórákra lettek kényszerítve. Az ipari gazdaságban a jutalom közelítőleg egyenesen arányos a befektetett idővel; 10%-kal több munka többé-kevésbé 10%-os bevételnövekedést jelent. Az információs gazdaságban, amelyet a nagy sávszélességű kommunikáció és a szellemi tulajdon könnyű replikációja jellemez (ezt nevezi Frank és Cook „a-győztes-mindent-visz társadalom"-nak) (Frank és Cook, 1996), nagy jutalom jár azért, ha valaki csak egy kicsit is jobb a versenyben: 10%-kal több munka akár 100%-os bevételnövekedést is jelenthet. Ezért egyre növekvő nyomás nehezedik mindenkire, hogy keményebben dolgozzon. A mesterséges intelligencia növeli a technológiai innováció ütemét, tehát hozzájárul ehhez az általános trendhez, de egyben a mesterséges intelligencia tartalmazza annak ígéretét is, hogy az automatizált ágensek kicsit átvállalják a teendőket és egy kis időhöz juttatnak minket.

Az emberek elveszíthetik az egyediségérzésüket. Weizenbaum, az `ELIZA` program szerzője, *Computer Power and Human Reason* (Weizenbaum, 1976) c. művében rámutat néhány potenciális veszélyre, amelyeket a mesterséges intelligencia jelent a társadalom számára. Weizenbaum egyik fő érve az, hogy a mesterséges intelligencia kutatása lehetővé teszi annak elképzelését, hogy az emberek automaták volnának, amely elgondolás az autonómia, vagy akár a humanitás érzésének megszűnéséhez vezet. Vegyük észre azonban, hogy ez az elképzelés jóval régebb óta van jelen, mint a mesterséges intelligencia: legalábbis a *L'Homme Machine* (La Mettrie, 1748) c. könyvig vezethető vissza. Azt is vegyük észre, hogy az emberiség túlélte már az egyediségérzésnek más csökkenéseit is: A *De Revolutionibus Orbium Coelestium* (Kopernikusz, 1543) kimozdította a Földet a Naprendszer központjából, és a *Descent of Man* (Darwin, 1871) pedig a *Homo sapienst* egy szintre helyezte a többi fajjal. A mesterséges intelligencia, amenynyiben széles körben sikeres lesz, éppen úgy veszélyt jelenthet a 21. század morális feltételezéseire, ahogyan Darwin evolúciós elmélete fenyegette a 19. század morális feltételezéseit.

Az emberek elveszíthetik a személyiségi jogaik egy részét. Weizenbaum arra is rámutatott, hogy a beszédfelismerési technológia a lehallgatás elterjedéséhez, és így a polgári jogok veszteségéhez vezethet. Nem láthatta előre azt a világot, ahol a terrorista fenyegetettség megváltoztatja, hogy az emberek mennyi megfigyelést hajlandók elfogadni, azt azonban helyesen ismerte fel, hogy a mesterséges intelligencia a tömeges megfigyelés eszközévé válhat. Jóslata talán már valóra is vált: az Egyesült Államok kormányának titkos Echelon-rendszere „felvevőállomások, antennamezők és radarállomások hálózatából áll, a rendszert nyelvek közti fordítást, beszédfelismerést és kulcsszavas keresést használó számítógépek támogatják, amelyek a telefon-, az e-mail, a fax- és a telexforgalmat vizsgálják végig."[8] Néhányan elfogadják, hogy a számítógépesítés a magánszféra veszteségéhez vezet. Scott McNealy, a Sun Microsystems vezetője azt mondta: „Amúgy is nulla a magánszférád. Felejtsd el." Mások nem értenek ezzel egyet: Louis Brandeis bíró[9] ezt írta 1890-ben: „A magánszférához való jog minden jogok legalapvetőbbike (…) a jog az ember saját személyiségéhez."

Az MI-rendszerek alkalmazása megszüntetheti a felelősségre vonhatóságot. Az Egyesült Államokban ma uralkodó pereskedő közhangulat fontos kérdéssé tette a jogi felelősséget. Amikor egy orvos a diagnózis felállításakor egy orvosi szakértői rendszer ítéletére hagyatkozik, ki tehető felelőssé a diagnózis hibájáért? Szerencsés módon, és ez részben a döntéselméleti megközelítés orvoslásra gyakorolt egyre növekvő befolyásának köszönhető, ma már elfogadott alapelv, hogy az orvos, aki végrehajtja a nagy *várható* hasznossági

[8] Lásd „Eavesdropping on Europe", *Wired* hírmagazin, 9/30/1998 és a hivatkozott EU-jelentéseket.
[9] Amerikai jogász és jogi közíró, a múlt század első felében a Legfelsőbb Bíróság tagja. (*A ford.*)

Created by XMLmind XSL-FO Converter.

értékű beavatkozásokat, akkor sem vádolható hanyagsággal, ha ezeknek a beavatkozásoknak katasztrofálisak a *tényleges* következményei a páciensnél. A kérdés ezek után a következő: „Kinek a hibája, ha a diagnózis nem plauzíbilis?" Mindeddig a bíróságok úgy tartották, hogy az orvosi szakértői rendszerek ugyanazt a szerepet töltik be, mint az orvosi tankönyvek és kézikönyvek: az orvosoknak felelőssége, hogy megértsék a döntések indoklását, és saját megítélésük alapján kell határozniuk, hogy elfogadják-e a rendszer javaslatát. Tehát ha az orvosi szakértői rendszereket ágensekként tervezzük, akkor cselekvéseiket nem úgy kell tekinteni, mint ami közvetlenül kihat a páciensre, hanem mint ami befolyásolja az orvos viselkedését. Ha a szakértői rendszerek megbízhatóbban pontosabbak lesznek, mint a diagnózisokat felállító emberek, akkor az orvosok akár jogilag is felelősek lehetnek, ha *nem* használják fel a rendszer javaslatait. Gawande (Gawande, 2002) ezt a feltételezést térképezi fel.

Hasonló kérdések kezdenek megjelenni az interneten használt intelligens ágensekkel kapcsolatban. Már elértek egy kis haladást kényszerfeltételek beépítésével az ágensekbe, hogy például ne okozhassanak károkat más felhasználók állományaiban (Weld és Etzioni, 1994). Amikor pénz is gazdát cserél, a probléma már nagyobb. Amikor egy pénzügyi tranzakció során egy intelligens ágens „valaki nevében" jár el, felelős-e ez a személy a keletkező adósságokért? Lehetségessé válhat-e, hogy egy intelligens ágensnek pénzügyi kintlévősége legyen, és a saját nevében eljárva elektronikus tranzakciókat hajtson végre? Mindeddig nem látszanak világosnak ezek a kérdések. Tudomásunk szerint eddig egy program sem kapta meg azt a jogi státust, hogy individuumként pénzügyi tranzakciókat hajthasson végre, és jelenleg ez nem is látszik indokoltnak. A valódi autópályákon sem tekintik a közlekedési szabályok betartatásakor „vezetőnek" a programokat. A kaliforniai jogban legalábbis semmilyen jogi szankció nincsen, amely megakadályozhatná, hogy egy automata jármű túllépje a sebességhatárt, noha baleset esetén felelőssé tehető a jármű vezérlőmechanizmusának tervezője. Csakúgy, mint az emberi megtermékenyítési technológia esetén, a jog még nem vette fel a lépést az új fejleményekkel.

A mesterséges intelligencia sikere az emberi faj végét jelentheti. Gyakorlatilag bármely technológiában benne rejlik a lehetőség, hogy kárt okozzon rossz kezekben, de a mesterséges intelligencia és a robotika esetében ezek a rossz kezek akár magához a technológiához is tartozhatnak. Számtalan sci-fi szól figyelmeztető jelként arról, hogy robotok vagy robot-ember kiborgok ámokfutást rendeznek. Marry Shelley *Frankenstein, or the Modern Prometheus* (Shelley, 1818)[10] c. műve és Karel Capek *R.U.R.* c. drámája, amelyben a robotok meghódítják a világot, korai példái az ilyen történeteknek. Mozifilmek is szólnak erről: a *The Terminator* (1984) a robotok-leigázzák-a-világot kliséit az időutazással ötvözi, a *The Matrix* (1999) pedig ugyanezt az agyak-a-tartályban-nal kombinálja.

Úgy tűnik, legtöbb esetben a robotok azért főszereplői olyan sok világ-leigázása történetnek, mert az ismeretlent jelképezik, csakúgy, mint a korábbi mesék boszorkányai és szellemei. Jelentenek-e a boszorkányoknál és a szellemeknél komolyabb veszélyt? Valószínűleg nem: ha a robotokat megfelelően, azaz olyan ágensekként tervezik, amelyek a gazdáik céljait teljesítik, akkor a jelenlegi tervezés lépésenkénti előrehaladásából származó robotok szolgálni fognak, nem pedig leigázni. Az emberek azért használják agresszíven az intelligenciájukat, mert a természetes kiválasztódás miatt velük született agresszivitással rendelkeznek. De a gépek, amiket magunk építünk, nem születnek agresszívnak, hacsak nem döntünk úgy, hogy ilyennek építjük őket. Másrészt viszont lehetséges, hogy a számítógépek meghódítanak minket abban az értelemben, hogy szolgálatukkal elengedhetetlenné válnak, csakúgy, mint az autók meghódították (ebben az értelemben) az iparosodott világot. Egy forgatókönyv részletesebb tanulmányozást érdemel. I. J. Good (Good, 1965) írta ezt:

> Definiáljuk az ultraintelligens gépet olyan gépként, amely messze túlhaladja a lehető legintelligensebb ember intellektuális tevékenységét is. Mivel a gépek tervezése az intellektuális tevékenységek egyike, egy ultraintelligens gép még jobb gépeket tud tervezni; nem kérdés, hogy egy „intelligenciarobbanás" történne, ami messze lehagyja az emberi intelligenciát. Az ultraintelligens gép így egyben az emberiség utolsó szükséges felfedezése, feltéve persze, hogy a gép elég engedelmes, és elmondja, hogyan tarthatjuk ellenőrzésünk alatt.
>
> —I. J. Good

Az „intelligenciarobbanás"-t **technológiai szingularitás**nak (**technological singularity**) is nevezte a matematika-professzor és sci-fi író Vernor Vinge, aki így fogalmaz (Vinge, 1993): „Harminc éven belül rendelkezni fogunk a technológiai eszközökkel egy emberfeletti intelligencia létrehozására. Röviddel ezután befejeződik az ember korszaka." Good és Vinge (és még sokan mások) jól figyelik meg, hogy a technológiai fejlődés jelenleg az exponenciális görbe szerint nő (gondoljunk csak a Moore-törvényre). Azonban ebből csak egy előreugrással lehet arra következtetni, hogy ez a görbe így fog folytatódni a közel végtelen növekedés

[10] Charles Babbage-ra is hatással volt fiatalemberként a *Frankeinstein* olvasása.

Created by XMLmind XSL-FO Converter.

szingularitásához. Eddig minden más technológia egy S alakú görbét követett, ahol az exponenciális növekedés végül lecsengett.

A bekövetkező szingularitásról író Vinge-et rémíti ez a lehetőség, de más számítógéptudósok és futuristák örömmel várják. Hans Moravec a *Robot: Mere Machine to Transcendent Mind* c. írásában azt jósolja, hogy a robotok ötven éven belül elérik az emberi intelligenciát, majd pedig meghaladják azt. Ezt írja:

> Egész gyorsan kiszorítanak minket a létezésből. Másokkal szemben engem ez a lehetőség nem tölt el aggodalommal, én ezeket a jövőbeli gépeket a leszármazottainknak, „elmebeli gyermekeink"-nek tartom, akiket képünkre és hasonlatosságunkra építettünk, önmagunkként egy jóval nagyobb képességű alakban. A korábbi generációk biológiai gyermekeihez hasonlóan az emberiség hosszú távú jövőről alkotott legjobb reményét testesítik meg. Belső kötelességünk minden előnyt megadni nekik és ellépni útjukból, ha már többet nem tudunk segíteni.
>
> —(Moravec, 2000)

A *The Age of Spiritual Machines* (Kurzweil, 2000) c. művében Ray Kurzweil azt jósolja, hogy 2000-re már „erős trend lesz összekapcsolni az emberi gondolkodást az eredetileg az emberi faj teremtette gépek világával". Még egy új szó is van: **transzhumanizmus** (**transhumanism**), amely az ezt a jövőt váró aktív társadalmi mozgalmat jelöli. Elegendő csak annyit mondani, hogy az ilyen kérdések kihívást jelentenek a legtöbb mai erkölcsi gondolkodónak, akik az emberi élet és az emberi faj megőrzését jó dolognak tartják.

Gondoljunk bele végül a robotok nézőpontjába. Ha a robotok tudatossá válnak, akkor immorális lehet puszta „gépként" kezelni őket (például darabokra szedni). A robotoknak maguknak is morálisan kell cselekedniük: beléjük kell programozni a jó és a rossz elméletét. A sci-fi írók, kezdve Isaac Asimovval (Asimov, 1942), már el is kezdtek foglalkozni a robotjogok és robotfelelősségek kérdéseivel. A jól ismert – *Mesterséges intelligencia* (A.I.) (Spielberg, 2001) c. mozifilm Brian Aldiss történetén alapul, amelyben egy intelligens robot, akit arra programoztak, hogy embernek képzelje magát, képtelen feldolgozni, hogy nem kell tulajdonos-anyjának. A történet (és persze a film) a robotok polgárjogi mozgalmának megalapozottságáról győzhet meg egyeseket.

4. Összefoglalás

Ebben a fejezetben a következő kérdésekkel foglalkoztunk:

- A filozófusok szóhasználata szerint a **gyenge MI** kifejezés azt a feltevést jelenti, hogy a gépek valószínűleg képesek intelligensen viselkedni, az **erős MI** pedig azt a feltevést, hogy az ilyen gépek elmével rendelkezőknek számítanának (és nem elmét szimulálóknak).
- Alan Turing visszautasította a „Tudnak-e gondolkodni a gépek?" kérdést, és egy viselkedési tesztet állított a helyébe. Sok, a gondolkodó gépek lehetőségével szemben felhozható ellenvetést megelőlegezett. A mesterséges intelligencia kutatói közül kevesen szentelnek figyelmet a Turing-tesztre, előnyben részesítik, ha a rendszerek gyakorlati problémamegoldó képességére összpontosítunk, nem pedig az emberek utánzására való képességükre.
- A modern korban általánosan egyetértenek abban, hogy a mentális állapotok agyállapotok.
- Sem az erős MI ellen, sem a mellette felhozható érvek nem meggyőzők. Kevés első vonalbeli MI-kutató gondolja azt, hogy bármi fontos is függ ennek a vitának a kimenetelétől.
- A tudatosság rejtély maradt.
- A mesterséges intelligencia és a kapcsolódó technológiák által a társadalomra jelentett veszélyek hat fajtáját azonosítottuk. Arra a következtetésre jutottunk, hogy néhány veszély valószínűtlen, kettő azonban további megfontolást érdemel. Az első az, hogy az ultraintelligens gépek, a maitól nagyon különböző, ámde ránk nézvést korántsem kedvező jövőhöz vezethetnek. A második pedig az, hogy a robottechnológia a pszichopatáknak is a kezébe adhatja a tömegpusztító fegyverek kulcsát. Arra jutottunk azonban, hogy ez a fenyegetés inkább származik a biotechnológiából és a nanotechnológiából, mintsem a robotikából.

4.1. Irodalmi és történeti megjegyzések

Created by XMLmind XSL-FO Converter.

Az elme természete az ókortól napjainkig a filozófiai elméletalkotás állandó témája. Platón a *Phaidón* c. dialógusában kifejezetten megvizsgálta, majd elvetette azt a lehetőséget, hogy az elme a test részeinek „egybehangoltsága", szerveződési mintája lenne (ez az álláspont közelít ahhoz, amit a modern elmefilozófiában funkcionalizmusnak nevezünk). Ehelyett arra az álláspontra helyezkedett, hogy az elme egy halhatatlan, nem anyagi lélek, amely a testtől elválasztható, és szubsztanciájában (lényegi alkotóelemét tekintve) különbözik: ez a dualizmus álláspontja. Arisztotelész az élő lényekre jellemző lélek (görögül *ψυχη*) több fajtáját különítette el, és legalábbis néhányukra funkcionalista leírást adott (erről bővebben lásd (Nussbaum, 1978)).

Hírhedt Descartes dualisztikus elmefelfogása, de ironikus módon történelmi hatása inkább a mechanisztikus és materialista elmefelfogás felé mutat. Az állatokat kifejezetten automatának tartotta, és megelőlegezte a Turing-tesztet: „azt nem tudjuk elképzelni, hogy [egy gép] a szavakat különféleképpen elrendezze, s ezáltal értelmesen tudjon felelni arra, amit a jelenlétében mondanak, amint ezt a legtompább eszű emberek is meg tudnak tenni" (Descartes, 1637).[11] Azzal, hogy hevesen kardoskodott az állatok automata mivolta mellett, Descartes valójában könnyebbé tette, hogy az embereket is automatának véljük, noha maga ezt a lépést sohasem tette meg. A *L'Homme Machine* (La Mettrie, 1748) c. könyv pontosan amellett érvelt, hogy az emberek automaták.

A modern analitikus filozófia általában elfogadja a materializmust (gyakran az agyállapot-**azonosság elmélet** (**identity theory**) formájában (Place, 1956; Armstrong, 1968), ami szerint a mentális állapotok azonosak az agyi állapotokkal), de sokkal megosztottabb a funkcionalizmust, az emberi elme gépi analógiáját tekintve, valamint abban a kérdésben, hogy szó szerinti értelemben véve tudnak-e gondolkodni a gépek. Turing (Turing, 1950) *Computing Machinery and Intelligence* c. cikkére adott korai filozófiai válaszok egy része, például Scriven (Scriven, 1953), megpróbálta cáfolni, hogy *értelmes* lenne azt mondani, hogy a gépek gondolkodhatnak, arra alapozva, hogy ez sértené a szó jelentését. Ezt a nézetet, legalábbis Scriven, cikkének újranyomásához illesztett függelékének tanúsága szerint 1963-ra visszavonta (Anderson, 1964). A számítástechnika-kutató Edsger Dijkstra szerint „Az a kérdés, hogy tud-e egy gép gondolkodni, nem érdekesebb annál, hogy tud-e úszni egy tengeralattjáró." Ford és Hayes (Ford és Hayes, 1965) amellett érvelnek, hogy a Turing-teszt semmi segítséget sem nyújt a mesterséges intelligencia számára.

Mivel az elmefilozófiában a mesterséges intelligencia alapján a funkcionalizmus választása tűnik a legtermészetesebbnek, a funkcionalizmus kritikája tehát gyakran a mesterséges intelligencia kritikájának formáját ölti (mint Searle esetében). Block (Block, 1980) osztályozását követve többféle funkcionalizmust különböztethetünk meg. Az agyállapot-azonosság elmélet egyik variánsa, a **funkcionális specifikáció elmélete** (**functional specification theory**) (Lewis, 1966; 1980) a mentális állapotokkal identifikálandó agyi állapotokat a funkcionális szerepük választja ki. A gépi analógiához szorosabban kötődik a **funkcionális állapotazonosság elmélet** (**functional state identify theory**) (Putnam, 1960; 1967). E szerint a mentális állapotokat nem fizikai agyi állapotokkal, hanem a kifejezett számítóeszközként felfogott agy absztrakt számítási állapotaival kell azonosítani. Ezek az absztrakt állapotok feltételezés szerint függetlenek az agy specifikus felépítésétől, ami azonban néhányakat arra késztet, hogy a funkcionális állapotazonosság elméletet a dualizmus egy formájának tartsák!

Mindkét agyállapot-azonosság elméletet, valamint a különböző formájú funkcionalizmusokat támadják azok a szerzők, akik szerint ezek az elméletek nem megfelelők a mentális állapotok *qualia*, azaz „milyen érzés is ez" aspektusának megmagyarázására (Nagel, 1974). Searle ehelyett az intencionalitás funkcionalizmusbeli állítólagos megmagyarázhatatlanságára koncentrált (Searle, 1980; 1984; 1992). Churchland és Churchland (Churchland és Churchland, 1982) mindkét fajta kritikát visszautasítják.

Az **eliminatív (kiküszöbölő) materializmus** (**eliminative materialism**) (Rorty, 1965; Churchland, 1979) abban különbözik az elmefilozófia többi jelentős elméletétől, hogy meg sem kísérli a „népies pszichológia", az elméről alkotott hétköznapi elképzeléseink igazolását, hanem mindet elveti, és megpróbálja az elme tisztán tudományos elméletével helyettesíteni. Elméletileg akár a klasszikus mesterséges intelligencia is szolgáltathatná ezt a tudományos elméletet, de valójában az eliminatív materialisták inkább az idegtudománytól és a neurális hálók kutatásától kölcsönöznek (Churchland, 1986), arra alapozva, hogy a klasszikus mesterséges intelligencia, különösen a 10. fejezetben bemutatotthoz hasonló „tudásreprezentációs" kutatási irány, hajlamos a népies pszichológia igazságaira alapozni. Ugyan az „intencionális beállítottság" álláspont (Dennett, 1971) értelmezhető funkcionalistaként, de valószínűleg az eliminatív materializmus egyik formájának kell tekinteni, mivel az intencionális beállítottság nem objektív tulajdonsága az ágensnek, akire vonatkozik. Azt is vegyük figyelembe, hogy lehetünk eliminatív materialisták egyes mentális jelenségekkel kapcsolatban, miközben másokat nem így elemzünk. Dennett (Dennett, 1978) például a qualiákat illetően sokkal inkább eliminativista, mint az intencionalitással kapcsolatban.

[11] A szöveget a magyar fordítás alapján idéztük: *Értekezés a módszerről* (ford. Szemere Samu és Boros Gábor). (*A ford.*)

Created by XMLmind XSL-FO Converter.

A fejezetben szóltunk a gyenge MI-vel szembeni kritika lehetséges forrásairól. Ugyan a posztneurális hálók korszakában divatossá vált idejétmúltnak tartani a szimbolikus megközelítéseket, nem minden filozófus kritikus a „Jófajta, régivágású MI”-vel szemben. Akadnak ennek még elkötelezett, sőt akár gyakorló hívei is. Zenon Pylyshyn (Pylyshyn, 1984) amellett érvelt, hogy a számítási modell nemcsak elviekben a legjobb modell a megismerés megértéséhez, hanem a jelenlegi kutatás útja is lehet, és kifejezetten megcáfolta Dreyfus kritikáját az emberi megismerés számítási modelljéről. Gilbert Harman (Harman, 1983) a hiedelemmódosítás vizsgálatát az igazság-karbantartó mesterséges intelligencia rendszerek kutatásához kapcsolta. Micheal Bratman az emberi pszichológiáról alkotott „hiedelem-vágy-intenció” modelljét (Bratman, 1987) alkalmazta a tervkészítéssel kapcsolatos mesterséges intelligencia kutatásokra (Bratman, 1992). Aaron Sloman (Sloman, 1978, iii. o.), a mesterséges intelligencia erős hipotézisének radikális híve odáig elment, hogy rasszistának nevezte Joseph Weizenbaumot (Weizenbaum, 1976), mert szerinte a feltételezett intelligens gépeket nem kellene személyeknek tekinteni.

Az elméről, az agyról és a kapcsolódó témákról szóló filozófiai irodalom kiterjedt, és a megfelelő szaknyelv és érvelési módozatokra vonatkozó megalapozott képzés hiányában gyakran nehezen olvasható. Ehhez a folyamathoz az *Encyclopedia of Philosophy* (Edwards, 1967) felettébb hasznos segítséget nyújthat meggyőzően informatív szócikkeivel. A *The Cambridge Dictionary of Philosophy* (Audi, 1999) rövidebb, de közérthetőbb mű, noha a fő szócikkek (mint például az „elmefilozófia”) még így is akár több mint tíz oldal hosszúak is lehetnek. A *MIT Encyclopedia of Cognitive Science* (Wilson és Keil, 1999) egyaránt foglalkozik az elme filozófiájával, biológiájával és pszichológiájával. Az elmefilozófiáról (beleértve ebbe a funkcionalizmust és más, a mesterséges intelligenciához kötődő álláspontokat) szóló cikkek általános gyűjteménye található a *Materialism and the Mind-Body Problem*-ben (Rosenthal, 1971) és a *Readings in the Philosophy of Psychology* (Block, 1980) első kötetében. Biro és Shanan (Biro és Shanan, 1980) a funkcionalizmus ellen és mellett érvelő cikkekből állított össze antológiát. Más cikkgyűjtemények kifejezetten a filozófia és a mesterséges intelligencia kapcsolatával foglalkoznak: *Minds and Machines* (Anderson, 1964); *Philosophical Perspectives in Artificial Intelligence* (Ringle, 1979); *Mind Design* (Haugeland, 1981) és a *The Philosophy of Artificial Intelligence* (Boden 1990). Számtalan bevezető létezik a filozófiai „MI-kérdés”-hez (Boden, 1977; 1990; Haugeland, 1985; Copeland, 1993). A *The Behavioral and Brain Sciences*, rövidítve *BBS*, fontos folyóirat, amelyet a mesterséges intelligenciával és az idegtudományokkal kapcsolatos vitáknak szentelnek. Több újság foglalkozik a mesterséges intelligencia etikai és felelősségi kérdéseivel: *AI and Society, Law, Computers and Artificial Intelligence*, valamint az *Artificial Intelligence and Law*.[12]

4.2. Feladatok

26.1.

Haladjon végig Turing állítólagos gépi „fogyatékosságokról” alkotott listáján, keresse meg, melyeket valósítottak már meg, melyeket lehet elvben megvalósítani egy programmal, és melyek problematikusak azért, mert tudatos mentális állapotokat igényelnek.

26.2.

Szükségszerűen bizonyítja-e a kínai szoba érv cáfolata, hogy a megfelelően programozott számítógépek mentális állapotokkal rendelkeznek? Szükségszerűen együtt jár-e az érv elfogadása azzal, hogy a számítógépeknek nem lehetnek mentális állapotaik?

26.3.

Az agyprotézis érv fontos része, hogy képesnek kell lennünk a kísérleti személy agyát visszaállítani a normális állapotába, hogy külső viselkedése ugyanolyan legyen, mintha a műtét nem is történt volna meg. Megalapozott lehet-e egy olyan szkeptikus ellenvetés, miszerint ez azzal járna, hogy módosítani kell a neuronoknak a tudatossághoz kötődő neurofiziológiai tulajdonságait, melyek különböznek a funkcionális viselkedésben résztvevőktől.

26.4.

Keressen ki a populáris médiából egy vagy több, a mesterséges intelligencia lehetetlenségére irányuló érvet, és elemezze azt.

[12] A magyar olvasó a mesterséges intelligencia filozófiájáról Darabos Tamás művéből tájékozódhat: *A gépesített értelem. Vázlatok a mesterséges intelligencia filozófiájáról.* Áron Kiadó, Budapest, 1991. (*A ford.*)

Created by XMLmind XSL-FO Converter.

26.5.

Próbáljon meg definíciókat adni az „intelligencia”, a „gondolkodás” és a „tudatosság” fogalmakra. Vessen fel lehetséges ellenvetéseket is a definíciókkal szemben.

26.6.

Elemezze a mesterséges intelligenciának a társadalomra jelentett potenciális veszélyeit. Mely fenyegetések a legsúlyosabbak, és hogy lehetne küzdeni ellenük? Hogyan viszonyulnak ezek a potenciális előnyökhöz?

26.7.

A mesterséges intelligencia jelentette potenciális fenyegetések hogyan viszonyulnak más számítástechnikai, vagy bio-, nano- és nukleáris technológiákból eredő veszélyekhez?

26.8.

Néhány kritikus ellenvetése szerint a mesterséges intelligencia lehetetlen, mások szerint meg *túlságosan is* lehetséges, és az ultraintelligens gépek veszélyt jelentenek. A két felhozott ellenvetés közül melyiket tartja valószínűnek? Ellentmondásos-e a kettőt együtt állítani?

Created by XMLmind XSL-FO Converter.

27. fejezet - MI: Jelen és jövő

Ebben a fejezetben felmérjük, hogy hol vagyunk és merre tartunk, mielőtt folytatnánk. Talán nem felesleges.

Az I. részben bevezettük a racionális ágensrendszert mint a mesterséges intelligencia egységes szemléleti keretét. Bemutattuk, hogy a tervezési probléma az ágens rendelkezésére álló észleléseken és cselekvéseken, az ágens viselkedésével kielégítendő célon és a környezet természetén múlik. Az ágensrendszertervek egész sorát lehet elképzelni, a reflexszerű ágenstől kezdve egészen a teljesen céltudatos, tudásalapú ágensig. Továbbá ezen rendszertervek komponensei többféle konkrét alakot ölthetnek – például logikait, valószínűségit vagy „neurálist”. Az ilyen jellegű komponensek működésével a megfelelő fejezetek foglalkoztak.

Fontos

Az ágensrendszer és komponensei tekintetében mind tudományos megértésünk, mind technológiai képességünk hatalmasat fejlődött. Ebben a fejezetben hátrébb lépünk a részletektől, és azt kérdezzük: *„Elvezet-e ez a fejlődés egy változatos környezetekben működésre képes, általános célú intelligens ágenshez?”* A 27.1. alfejezetben számba vesszük az intelligens ágens komponenseit, hogy meghatározhassuk, mi az, amit ismerünk, és mi az, ami még hiányzik. A 27.2. alfejezetben az általános ágensarchitektúra kapcsán tesszük ugyanezt. A 27.3. alfejezetben azt vizsgáljuk, hogy a „racionális ágens tervezése”-e az elsődlegesen helyes cél. (A válasz: „Nem igazán, de egyelőre megteszi.”) Végül a 27.4. alfejezetben azzal foglalkozunk, milyen következményekkel járhat vállalkozásunk sikere.

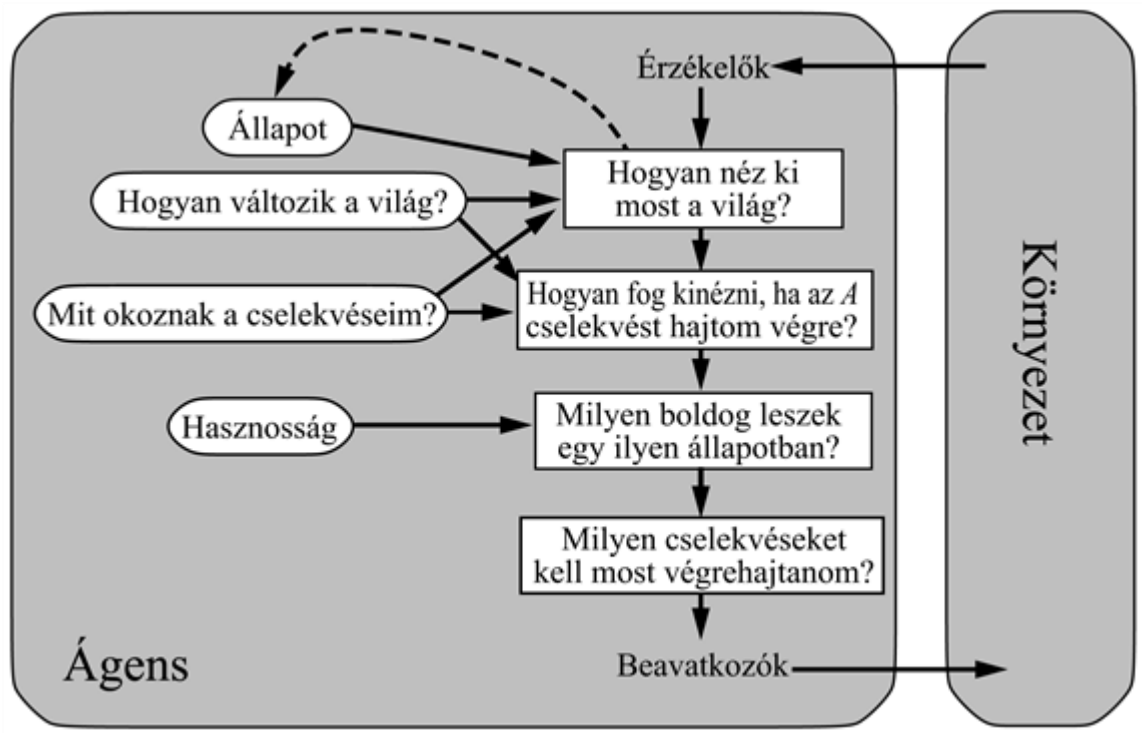
1. Ágensösszetevők

A 2. fejezetben számos ágenstervet és azok komponenseit mutattuk be. Figyelmünket összpontosítandó itt csak a hasznosságalapú ágenssel foglalkozunk, amelyet a 27.1. ábrán mutatunk be újra. Ez a legáltalánosabb ágenstervünk; de megvizsgáljuk tanulási képességekkel való kiterjesztését is (lásd 2.15. ábra).

Interakció a környezettel szenzorok és beavatkozószervek segítségével. Ez a mesterséges intelligencia történetének nagy részében komoly hiányosság. Néhány tiszteletre méltó kivétellel minden MI-rendszert úgy építettek fel, hogy embereknek kellett szolgáltatniuk a bemeneteket, és értelmezni a kimeneteket, a robotika pedig alacsony szintű feladatokra összpontosított, ahol nagyrészt hiányzott a magas szintű következtetés és tervkészítés. Ez részben a magas költségek és a komoly tervezői erőfeszítés miatt volt így, ami ahhoz kellett, hogy egyáltalán valódi, működő robotokat kapjunk. Az elmúlt években az előre elkészített, programozható robotok – mint például a 25.4. (b.) ábrán látható négylábú robot – megjelenésével a helyzet rohamosan megváltozott. Ez a változás végső soron a kicsi, olcsó, nagy felbontású CCD-kameráknak és a kompakt, megbízható motoros hajtásoknak köszönhetően következett be. Az MEMS- (mikro-elektromechikai rendszerek) technológia miniatürizált gyorsulásmérőket és giroszkópokat hozott létre, napjainkban pedig olyan beavatkozószerveket állít elő, amelyek például egy mesterséges repülő rovart bizonyos képességekkel ruháznak majd fel. (Talán MEMS-beavatkozószervek milliói kapcsolhatók majd össze, hogy nagy teljesítményű makroszkopikus beavatkozókat kapjunk.) A fizikai környezeteket illetően az MI-rendszereknek nem lehet több kifogása. Továbbá egy teljességgel új környezet, az internet is létrejött.

27.1. ábra - Egy modell- és hasznosságalapú ágens, ahogyan először a 2.14. ábrán bemutattuk

Created by XMLmind XSL-FO Converter.



Nyomon követni a világ állapotát. Ez az intelligens ágens számára szükséges legfontosabb képességek egyike, amelyhez egyaránt szükség van érzékelésre és a belső reprezentáció frissítésére. A 7. fejezetben ítéletlogikát alkalmaztunk a világ állapotának nyomon követésére; a 10. fejezetben kiterjesztettük ezt az elsőrendű logikára; a 15. fejezetben pedig **szűrő algoritmus**okat (**filtering algorithm**s) mutattunk be, hogy a bizonytalan környezeteket is követni tudjuk. Ezekre a szűrőeszközökre van szükség, amikor igazi – tehát tökéletlen – érzékelésre kerül sor. A jelenlegi szűrő és érzékelési algoritmusokat lehet úgy kombinálni, hogy elég jó teljesítményt kapjunk alacsony szintű predikátumok leírásánál, mint például „a csésze az asztalon van”, de van még megteendő út előttünk, míg az algoritmusok azt is ki tudják jelenteni, hogy: „dr. Russell teázik dr. Norviggel.” Egy másik probléma az, hogy ugyan az approximációs szűrő algoritmusok képesek elég nagy környezeteket kezelni, lényegüket tekintve azonban még mindig *propozicionálisak:* a propozicionális logikához hasonlóan nem kezelik explicit módon az objektumokat és a relációkat. A 14. fejezetben bemutattuk, hogyan lehet ennek a problémának a megoldására elsőrendű logikát alkalmazni; elvárásaink szerint nagy előrelépéssel fog majd járni, ha a komplex környezetek kezelésére is alkalmazzák ezeket az elgondolásokat. Továbbá, mihelyt egy bizonytalan környezetben elkezdünk objektumokról beszélni, felmerül a **bizonytalan azonosság** (**identity uncertainty**) problémája: nem tudjuk, melyik objektum melyik. A logika alapú mesterséges intelligenciában meglehetősen elhanyagolták ezt a problémát, mivel általában feltételezték, hogy az észleletek állandó szimbólumokat tartalmaznak az objektumok azonosítására.

Előrevetíteni, értékelni és kiválasztani jövőbeli cselekvéseket. Az alapvető tudásreprezentációs elvárások itt is ugyanazok, mint a világ állapotának nyomon követésénél; az elsődleges nehézség a cselekvésekkel (mint például beszélgetni és teázni) való boldogulásból származik, amelyek esetenként az igazi ágens elemi lépéseinek ezreit vagy millióit tartalmazhatják. Az emberek is csak úgy tudnak ezzel megbirkózni, hogy a viselkedést egy **hierarchikus struktúrá**ba (**hierarchial structure**) szervezik. A 12. fejezetben bemutatott tervezési algoritmusok némelyike az ehhez hasonló méretű problémák kezelésére hierarchikus és elsőrendű logikai reprezentációt használ; másrészt pedig a 17. fejezetben megadott bizonytalansági döntéshozó algoritmusok lényegileg ugyanazon az elven alapulnak, mint a 3. fejezet állapottérbeli kereső algoritmusai. Láthatóan itt sok a tennivaló, talán a **hierarchikus megerősítéses tanulás** (**hierarchial reinforcement learning**) újabb fejleményei mentén.

Hasznosság mint a preferenciák kifejezése. A racionális döntéseket az elvárt hasznosság maximalizálására alapozni elméletileg teljesen általános, és megóv a tisztán célorientált megközelítések számtalan hibájától (mint például az ellentmondó célok és a bizonytalan elérésük). Mindeddig azonban kevés munka folyt a *realisztikus* hasznossági függvények előállítása terén – képzeljük el például az egymással kölcsönhatásban lévő preferenciák komplex hálózatát, amelyeket az emberek irodai asszisztenseként működő ágensnek meg kell értenie. Elég nehéznek bizonyult az összetett állapotokkal kapcsolatos preferenciákat dekomponálni az összetett állapotokról szóló bizonyosságok Bayes-hálós dekompozíciójához hasonlóan. Ennek az egyik oka az lehet, hogy az állapotok

Created by XMLmind XSL-FO Converter.

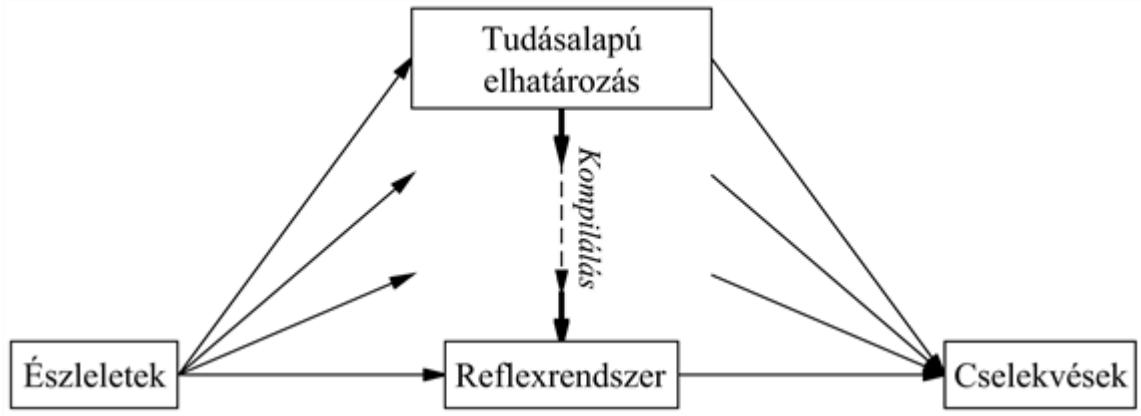
preferenciái valójában az állapotok múltbeli preferenciáiból lettek *összeszerkesztve,* amelyeket **jutalomfüggvények** (**reward functions**) írnak le (lásd 17. fejezet). A keletkező hasznossági függvény akkor is nagyon összetett lehet, ha a jutalomfüggvény egyszerű. Mindez azt sugallja, hogy komolyan kell vennünk a jutalomfüggvények tudásmérnöki konstrukcióját, mert ezzel tudjuk az ágens értésére adni, mit akarunk tőle.

Tanulás. A 18–20. fejezetben leírtuk, hogy az ágens tanulása miként fogalmazható meg az ágens különböző komponenseinek induktív (ellenőrzött, nem ellenőrzött vagy megerősítéses) tanulásaként. Nagyon hatékony logikai és statisztikai technikákat fejlesztettek ki, amelyek egész nagy problémákkal is meg tudnak birkózni, és gyakran elérik, sőt meghaladják az embernek azt a képességét, amellyel egy adott szótár feletti prediktív mintákat azonosít. Másrészt viszont abban a fontos problémában, hogy a bemeneti szótárnál absztraktabb szintű, új reprezentációkat hozzanak létre, a gépi tanulás kevés haladást ért el. Hogyan tud például egy autonóm robot létrehozni az emberektől meg nem kapott olyan új predikátumokat, mint például az *Iroda* és a *Kávé*? Hasonló megfontolások alkalmazhatók a viselkedés tanulására is: a *Teázás* fontos, magas szintű cselekvés, de hogyan kerül be egy olyan cselekvéskönyvtárba, ahol eredetileg csak jóval egyszerűbb cselekvések szerepeltek, mint például a *KezetFelemelni* és a *Lenyelni*. Amíg nem értjük meg ezeket a problémákat, addig a hatalmas józan ész tudásbázisok kézi létrehozásának csüggesztő feladatával nézünk szembe.

2. Ágensarchitektúrák

Természetes, ha azt kérdezzük: „Melyik architektúrát alkalmazzam a 2. fejezetben bemutatottak közül?” A válasz: „Mindegyiket!” Láttuk, hogy a reflexszerű válaszok alkalmasak az időkritikus helyzetekben, míg a tudásalapú elhatározás lehetővé teszi, hogy az ágens előre tervezzen. Egy teljes ágensnek egy **hibrid architektúra** (**hybrid architecture**) segítségével képesnek kell lennie mindkettőre. A hibrid architektúra egyik fontos tulajdonsága, hogy a különböző döntési komponensek határa nem rögzített. A **szerkesztés** vagy **kompilálás** (**compilation**) például az elhatározás szintjén megjelenő deklaratív információt folyamatosan hatékonyabb reprezentációkká alakítja át, eljutva végül a reflexszerű szintre (27.2. ábra). (Ez a célja a 19. fejezetben bemutatott magyarázatalapú tanulásnak.) Az olyan rendszereknek, mint a SOAR (Laird és társai, 1987) és a THEO (Mitchell, 1990), pontosan ez a struktúrájuk. Minden alkalommal, amikor explicit elhatározás révén jutnak el egy probléma megoldásáig, a megoldás általánosított verzióját is eltárolják a reflexszerű komponens részére. Ennek a folyamatnak a *fordítottját* kevésbé tanulmányozták még: a környezet változásakor a megtanult reflexek lehet, hogy többé már nem érvényesek, és az ágensnek vissza kell térnie a tudatos konstrukció szintjére, hogy új viselkedést hozhasson létre.

27.2. ábra - A kompilálás célja a megfontolt (deliberatív) döntéshozatal hatékonyabb, reflexszerű mechanizmussá való átalakítása



Egy ágensnek saját következtetéseit is az ellenőrzése alá kell tudnia vonni. A következtetést abba kell hagynia, ha cselekvésre van szükség, és a következtetésre rendelkezésre álló időt a legígéretesebb számításokra kell fordítania. Egy taxisofőr ágensnek, például, ha közvetlenül maga előtt balesetet észlel, nem félórák alatt, hanem a másodperc töredéke alatt el kell tudnia dönteni, hogy fékezni fog, vagy inkább kikerüli a balesetet. A másodperc ezen töredékét a legfontosabb kérdések vizsgálatára kell fordítania, hogy balra, illetve jobbra szabad-e a sáv, és nincs-e közvetlenül mögötte egy nagy teherautó, és nem arra, hogy milyen az idő, mennyire kopnak a gumik, vagy hogy hol találja a következő utast. Az ilyen problémákkal a **valós idejű MI** (**real-time AI**) foglalkozik. Ahogy az intelligens rendszerek egyre bonyolultabb tárgyterületekre vonulnak be, minden

Created by XMLmind XSL-FO Converter.

probléma valós idejűvé válik, hiszen az ágensnek soha nem lesz elegendő ideje arra, hogy a döntési problémát egzakt módon oldja meg.

Világos, hogy sürgető szükség van az általánosabb döntéshozó helyzetekben is működő módszerekre. Az utóbbi időben két kecsegtető módszertan is napvilágot látott. Az első az **akármikor algoritmus**ok (**anytime algorithm**s) alkalmazása (Dean és Boddy, 1988; Horvitz, 1987). Egy akármikor algoritmusnál az algoritmus kimenetének minősége az idő múlásával fokozatosan javul, bármikor szakítjuk is meg tehát az algoritmust, ésszerű döntéshez jutunk. Az ilyen algoritmus vezérlését egy metadöntési eljárás végzi, amely meghatározza, hogy érdemes-e a számításokat tovább folytatni. Az akármikor algoritmusra egy egyszerű példa a kétszemélyes játékoknál ismertetett iteratív mélyítő keresés. Azonban bonyolultabb rendszerek – amelyek számos ilyen, együttesen működő algoritmusból állnak – is konstruálhatók (Zilberstein és Russell, 1996). A másik módszer a **döntéselméleti metakövetkeztetés** (**decision-theoretic metareasoning**) (Horvitz, 1989; Russell és Wefald, 1991; Horvitz és Breese, 1996). Ebben a megközelítésben az információérték elméletét (16. fejezet) a számítások megválasztására használjuk. A számítás értéke mind a költségétől (a késleltetett cselekvés miatt), mind a hasznosságától (a döntés javított minősége miatt) függ. A metakövetkeztetési sémákkal jobb keresési algoritmusokat lehet előállítani, automatikusan garantálva az akármikor tulajdonságot. A metakövetkeztetés természetesen költséges, de a kompilálás arra is felhasználható, hogy a túlmunka költsége eltörpüljön a számítás költsége mellett, lévén hogy a számítás is felügyelt.

A metakövetkeztetés csupán egyik aspektusa az általános **reflektív architektúrá**nak (**reflective architecture**): ezek az architektúrák lehetővé teszik a magán az architektúrán belül keletkező számítási entitások és cselekvések fölötti tudatosságot. A reflektív architektúrák elméleti alapja úgy építhető fel, hogy a környezet állapotából és magának az ágensnek a számítási állapotából álló, közös állapotteret definiálunk. Döntési eljárásokat és tanulási algoritmusokat lehet úgy tervezni, hogy ebben a közös állapottérben működjenek, ezzel segítve és javítva az ágens számítási teljesítményét.

Végső soron azt várjuk, hogy az olyan feladatspecifikus algoritmusok, mint amilyen az alfa-béta keresés és a hátrafelé láncolás, eltűnnek az MI-rendszerekből, továbbá, hogy ezek olyan általános módszerekkel lesznek helyettesíthetők, amelyek az ágens számításait jó minőségű döntések hatékony létrehozása felé irányítják.

3. Egyáltalán a jó irányba haladunk?

Az előző fejezetekben sok előrelépést soroltunk fel, és még több lehetőséget a jövőbeli fejlődésre. De hova vezet mindez? Dreyfus analógiaként azt hozza fel (Dreyfus, 1992), hogy amikor valaki egy fa megmászásával akar a Holdra jutni, az illető állandó haladásról számolhat be egészen a fa csúcsáig. Ebben a fejezetben az a kérdésünk, hogy a mesterséges intelligencia fejlődése a famászásra hasonlít-e, vagy inkább a holdrakétára.

Az 1. fejezetben azt mondtuk, hogy célunk a *racionálisan cselekvő* ágens építése. Azonban arra is figyelmeztettünk, hogy

> …komplex környezetben a tökéletes racionalitást – mindig helyesen cselekedni – lehetetlen elérni. A számítási szükségletek egyszerűen túl komolyak. A könyv nagyobb részében azonban azzal a munkahipotézissel fogunk élni, hogy a tökéletes döntéshozatal megértése jó kiindulópont.
>
> —1. fejezet

Itt az ideje ismét végiggondolni, mi is az MI pontos célja. Ágenseket akarunk építeni, de milyen specifikáció van közben a fejünkben? Íme, négy lehetőség:

1. **Tökéletes racionalitás** (**perfect rationality**): A tökéletesen racionális ágens minden pillanatban úgy cselekszik, hogy a környezetről szerzett információja alapján várható hasznosságát maximalizálja. Mivel láttuk, hogy a legtöbb környezetben túlságosan időrablók a tökéletes racionalitáshoz szükséges számítások, ezért ez nem realisztikus cél.

2. **Számítható racionalitás** (**calculative rationality**): Ez az a megközelítés, amit implicit módon a logikai és döntéselméleti ágensek tervezésekor használtunk. Egy számíthatóan racionális ágens *végül* meghatározza azt, ami racionális választás *lehetett volna* következtetéseinek az elején. Ez érdekes tulajdonsága egy rendszernek, de a legtöbb környezetben a rosszkor érkező jó válasznak nincs semmi értéke. A gyakorlatban az MI-rendszerek tervezői rá vannak kényszerítve arra, hogy a tervezési minőséget illetően egy elfogadható általános teljesítmény érdekében kompromisszumot hozzanak; szerencsétlen módon azonban a számítható racionalitás elméleti alapja nem nyújt jól kidolgozott lehetőséget egy ilyen kompromisszum meghozatalára.

Created by XMLmind XSL-FO Converter.

3. **Korlátozott racionalitás** (**bounded rationality**): Herbert Simon elvetette a tökéletes (vagy akár a közelítőleg tökéletes) racionalitás gondolatát, és a korlátozott racionalitást állította a helyére (Simon, 1957), ami a valódi ágensek döntéseinek leíró elmélete. A következőt írta:

> A valósvilág-beli objektív racionális viselkedéshez, vagy akár csak ennek az objektív racionális viselkedésnek az elfogadható megközelítéséhez megoldandó problémák méretéhez képest túl kicsi az emberi elme komplex problémák megfogalmazására és megoldására szolgáló kapacitása.
>
> —Herbert Simon

Azt javasolta, hogy a korlátozott racionalitás elsődlegesen az **elfogadhatóság**ban (**satisficing**) nyilvánul meg: olyan hosszú ideig kell foglalkozni a döntéssel, ami elegendő egy „elfogadhatóan jó” válasz megtalálásához. Ezért a munkájáért Simon közgazdasági Nobel-díjat kapott, és később részletesen is írt még róla (Simon, 1982). Ez az emberi viselkedésnek sok esetben hasznosnak tűnő modellje, azonban nem ad formális specifikációt az intelligens ágensek számára, mert az elmélet nem adja meg az „elfogadhatóan jó” definícióját. Továbbá az elfogadhatóság láthatóan csak egyike a korlátozott erőforrások leküzdését szolgáló technikáknak.

4. **Korlátozott optimalitás**, **KO** (**bounded optimality**, **BO**): Egy korlátozottan optimális ágens *adott számítási erőforrások mellett* a lehető legjobban cselekszik. Egy ilyen ágens esetén az ágensprogram várható hasznossága legalább olyan magas, mint az ugyanazon a gépen futó bármilyen más ágensprogramé.

A négy lehetőség közül, úgy tűnik, a „korlátozott optimalitás” választása adja a legtöbb esélyt az MI erős elméleti megalapozására. Megvan az az előnye, hogy elérhető: mindig van legalább egy legjobb program – a tökéletes racionalitásról ugyanez nem mondható el. A korlátozottan optimalitás ágensek hasznosak a mindennapi gyakorlatban is, a számítható racionalitású ágensek általában nem, míg az elfogadhatóságra törekvő ágenseknél ez a szeszélyeiken múlik.

A hagyományos megközelítés az MI-ben az volt, hogy a számítható racionalitással kezdünk, majd az erőforráskorlátok kielégíthetősége érdekében kompromisszumokat hozunk. Ha ezek a korlátok csak kis problémákat okoznak, akkor a keletkező terv várhatóan a KO ágens tervéhez lesz hasonló. Ha azonban az erőforráskorlátok kritikussá válnak – például mert a környezet egyre összetettebbé válik –, akkor a két terv várhatóan el fog térni egymástól. A korlátozott optimalitás elmélete ezen korlátok kezelésére kínál elveket.

A korlátozott optimalitásról egyelőre keveset tudunk. Nagyon egyszerű gépekhez és valamilyen mértékben korlátozott környezetekhez korlátozottan optimális programok konstruálhatók (Etzioni, 1989; Russell és társai, 1993). Bonyolult környezetekben működő nagy, általános rendeltetésű számítógépek esetén azonban fogalmunk sincs, hogy milyenek legyenek a KO-programok. Ha a korlátozott optimalitás konstruktív elmélete valaha is megfogalmazható lesz, hinnünk kell, hogy a korlátozottan optimális programok tervezése nem fog túlságosan függni a használt számítógép részleteitől. A tudományos kutatást nagyon megnehezítené, ha például kiderülne, hogy egy gigabájtos memóriájú gép néhány kbájt memóriával való bővítése lényegi különbséget jelentene a KO-program tervében. Ahhoz, hogy ez ne történjen meg, a korlátozott optimalitás kritériumait kissé lazábban kell kezelni. Az aszimptotikus bonyolultság (A) függelék) mintájára definiálhatjuk az **aszimptotikus korlátozott optimalitás**t, **AKO**) (**asymptotic bounded optimality**, **ABO**) (Russell és Subramanian, 1995). Tegyük fel, hogy egy P program egy M gép esetén korlátozottan optimális a környezetek **E** osztályában, ahol a környezetek komplexitása nem korlátozott. Ekkor a P' program egy AKO-program M részére **E**-ben, ha egy P programot egy kM gépen futtatva a teljesítményében túlszárnyalja, és ahol a kM gép k-szor gyorsabb (vagy nagyobb), mint M. Ha k nem bizonyulna óriásinak, elégedettek lehetnénk, ha nemtriviális környezetek és nemtriviális architektúrák esetén AKO-programokkal rendelkezünk. Kevés ráció lenne abban, hogy óriási erőfeszítések árán inkább KO-, mint AKO-programokat keressünk, ha a használható gépek nagysága és sebessége, úgy tűnik, egy adott idő alatt konstans mértékben úgyis nő.

Megkockáztathatjuk azt is, hogy a KO- vagy az AKO-programok – nagy teljesítőképességű számítógépek és bonyolult környezetek esetén – nem fognak szükségszerűen egyszerű és elegáns struktúrával rendelkezni. Láttuk már, hogy az általános rendeltetésű intelligenciához bizonyos fokú reflexszerű és tudatos képesség szükséges, ami a tudás és a döntéshozatali formák sokaságát, ezen formák mindegyikéhez a tanuló és a kompilálási mechanizmusokat, a következtetés vezérlésének módszereit és a tárgyterületre vonatkozó specifikus tudás nagy tárát jelenti. A korlátozottan optimális ágensnek adaptálódnia kell ahhoz a környezethez, amelyben tartózkodik, így belső szervezettsége végül az adott környezetre jellemző optimalizálási sémákat tükrözheti. Ez elvárható, hasonlóképpen ahhoz, ahogy a súlyukban és a lóerőben korlátozott versenyautók szélsőségesen komplikált konstrukciókig jutottak el. Az a sejtésünk, hogy a korlátozott optimalitáson alapuló mesterséges intelligencia tudománya nagy részben olyan folyamatok tanulmányozásából fog állni, amelyek révén egy ágensprogram a

Created by XMLmind XSL-FO Converter.

korlátozott optimalitáshoz fog konvergálni, és talán kevésbé fog az eredményül kapott, nehezen áttekinthető programok részleteivel törődni.

Összegezve, a korlátozott optimalitás koncepcióját jól definiált és kivitelezhető kutatási feladatnak javasoljuk a mesterséges intelligencia kutatása számára. A korlátozott optimalitás az optimális *programokat* specifikálja, és nem az optimális *cselekvést*. A cselekvést úgyis a programok generálják, és a programok azok, amelyek felett a tervező befolyással rendelkezik.

4. Mi van, ha az MI sikerrel jár?

David Lodge az irodalomkritikusok egyetemi világáról szóló *Small world (Kis világ)* (Lodge, 1984) c. novellájában a főhős megdöbbenést okoz, amikor a kiváló, ámde egymással vitatkozó tagokból álló irodalomelméleti bizottságtól megkérdezi: *„Mi van, ha igazuk van?”* Az elméletalkotók egyike sem gondolt még erre a kérdésre (valószínűleg azért, mert megcáfolhatatlan elméleteken vitatkozni csak öncél lehet). Néha hasonló zavart okozhatunk, ha MI-kutatóktól kérdezzük azt: „Mi van, ha sikerrel jársz?” A mesterséges intelligencia izgalmas dolog, az intelligens számítógépek pedig nyilvánvalóan jobbak, mint buta társaik, miért hát az aggodalom?

Ahogy a 26.3. alfejezetben felvetettük, bizonyos etikai kérdéseket is meg kell fontolni. Az intelligens gépek sokkal többre képesek, de vajon erejüket jóra vagy rosszra fordítják? A mesterséges intelligencia kifejlesztésén fáradozók felelőssége, hogy megbizonyosodjanak munkájuk pozitív hatásáról. Hogy a hatás mennyire lesz széles körű, az a siker mértékétől függ. Már a mesterséges intelligencia mérsékelt sikere is változásokat hozott a számítógép-tudomány tanításában (Stein, 2002) és a szoftverek fejlesztésében. Az MI olyan új alkalmazásokat tett lehetővé, mint például a beszédfelismerő, készletellenőrző vagy megfigyelő rendszerek, robotok és keresőgépek.

A mesterséges intelligencia közepes méretű sikere várhatóan befolyásolná a legkülönbözőbb emberek mindennapi életét. A számítógépesített kommunikációs rendszerek (mint például a mobiltelefonok és az internet) már eddig is mélyreható változásokat okoztak a társadalomban, a mesterséges intelligencia esetében azonban ez még várat magára. Könnyű elképzelni, hogy a tényleg hasznos otthoni és irodai személyi asszisztensek komoly pozitív hatással lennének az emberek életére, habár rövid távon valószínűleg némi gazdasági zavart is okozna. Az ilyen fejlettségi szintű technológiai kapacitás azonban alkalmazható lenne autonóm fegyverek létrehozására is, és ez a legtöbbek szerint nemkívánatos eredmény lenne.

Végül pedig valószínűnek tűnik, hogy a mesterséges intelligencia nagyarányú sikere – az emberihez hasonló vagy annál magasabb szintű intelligencia létrehozása – megváltoztatná az emberiség többségének életét. Átalakulna munkánk és játékunk belső természete, csakúgy, mint az intelligenciáról és a tudatosságról vallott nézeteink, sőt megváltozna az emberi faj jövőbeli sorsa is. Ezen a szinten az MI-rendszerek közvetlen fenyegetést jelenthetnének az emberi autonómiára, szabadságra, sőt akár a fennmaradásunkra is. Mindezek miatt a mesterséges intelligencia kutatásától nem választhatók el az etikai következményei.

Mit hoz majd a jövő? A sci-fi írók láthatóan jobban kedvelik a negatív utópiákat, mint a pozitívakat, valószínűleg azért, mert ez ad lehetőséget az érdekesebb cselekményekre. De a mesterséges intelligencia mindeddig más forradalmi technológiákhoz (nyomtatás, csővezetékek, légi utazás, telefónia) hasonlóan viselkedett: pozitív aspektusaik ellensúlyozták negatív hatásaikat.

Összefoglalásul azt mondhatjuk, hogy láttuk, a mesterséges intelligencia mekkora haladást tett meg rövid története alatt, de ma is érvényes Alan Turing *Computing Machinery and Intelligence* c. esszéjének zárómondata:

> Csak rövid távolságra látunk előre, de azt látjuk, hogy sok a tennivaló.
>
> —Alan Turing: *Computing Machinery and Intelligence*

Created by XMLmind XSL-FO Converter.

A. függelék - Matematikai alapok

1. A1. Bonyolultságanalízis és az O() jelölés

A számítógépes szakembereknek gyakran szembe kell nézniük azzal a feladattal, hogy eldöntsék, két algoritmus közül melyik fut gyorsabban, vagy melyik igényel kevesebb memóriát. Ezt a feladatot kétféle módon lehet megközelíteni. Az első a **teljesítménymérés** (**benchmarking**) – ahol az algoritmusokat számítógépen futtatjuk, és megmérjük, milyen a sebességük másodpercekben, valamint a memóriafogyasztásuk bájtokban. Hiszen végül is ez az, ami igazából számít. A teljesítménymérés azonban nem biztos, hogy kielégítő mérték, mivel igen specifikus: egy konkrét program teljesítményét méri, amely program egy konkrét programozási nyelven íródott, konkrét gépen fut, konkrét fordító és bemeneti adatok mellett. A teljesítménymérő program által szolgáltatott egyszerű adatokból nehéz lenne arra következtetni, hogy az algoritmus milyen jól vizsgázik más fordító, más számítógép vagy más adathalmaz esetén.

1.1. Aszimptotikus analízis

A második megközelítés az **algoritmusok** matematikai **elemzésén** (**analysis of algorithms**) alapul, konkrét implementációtól és bemenetektől függetlenül. Ezt a megközelítést egy példa kapcsán, egy számsorozatot összeadó program esetén fogjuk ismertetni:

```
function ÖSSZEGZÉS(szekvencia) returns egy szám
  összeg ← 0
  for i ← 1 to HOSSZ(szekvencia)
      összeg ← összeg + szekvencia[i]
  end
  return összeg
```

Az elemzés első lépése a bemenetektől való eltekintés, olyan paraméter vagy paraméterek megkeresése, amely vagy amelyek jellemzők a bemenet méretére. Jelen példában a bemenetet a szekvencia hosszával jellemezhetjük, aminek a jele legyen *n*. Második lépés az implementációktól való eltekintés, azaz egy olyan mérték megkeresése, amely tükrözi az algoritmus futásiidő-igényét, de nem kötődik egy konkrét fordítóhoz vagy számítógéphez. Az `ÖSSZEGZÉS` program esetén ez lehetne például a végrehajtott kód sorainak a száma. A mérték részletesebb is lehetne, külön specifikálva az algoritmus által végrehajtott összeadások, értékadások, tömbhivatkozások és elágazások számát. Mindkét esetben megkapjuk az algoritmus által végzett lépések teljes számának a jellemzését a bemenet nagyságának függvényében, amit $T(n)$-nek fogunk nevezni. Ha megszámoljuk a kód sorait, a példánk esetében az eredmény $T(n) = 2n + 2$.

Ha minden program olyan egyszerű lenne, mint az `ÖSSZEGZÉS`, az algoritmusok elemzése triviális lenne. Két probléma miatt a helyzet azért bonyolultabb. Először is, ritka eset olyan paramétert találni, mint az *n*, amely teljes egészében jellemzi az algoritmus által végrehajtott lépések számát. A legjobb, amivel ehelyett rendelkezhetünk, a legrosszabb esetet jelentő $T_{\text{legrosszabb}}(n)$ vagy az átlagos $T_{\text{átlag}}(n)$ számítása. Az átlag kiszámításához az elemzőnek fel kell tételeznie a bemenetek valamilyen eloszlását.

A másik probléma az, hogy az algoritmusok ellenállnak az egzakt elemzésnek. Ilyen esetben közelítésekre szükséges támaszkodnunk. Azt mondhatjuk, hogy az `ÖSSZEGZÉS` algoritmus $O(n)$ – avagy nagy ordó *n* – jelezvén, hogy a mértéke, néhány kis értékű *n* kivételével, a legtöbb esetben *n* konstans-szorosa. Formálisan:

$T(n)$ $O(f(n))$ mértékű, ha $T(n) \leq kf(n)$ valamilyen k-ra, minden $n > n_0$ esetén.

Az $O(n)$ jelölés az **aszimptotikus analízis**t (**asymptotic analysis**) jelenti. Külön vizsgálat nélkül azt mondhatjuk, hogy ha *n* aszimptotikusan közeledik a végtelenhez, egy $O(n)$ algoritmus jobb, mint egy $O(n^2)$ algoritmus. Egy egyedi teljesítménymérő adat egy ilyen kijelentést nem tudna alátámasztani.

Az $O()$ jelölés a konstans szorzóktól eltekint, emiatt kevésbé precíz, viszont könnyebb használni, mint a $T()$-t. Így például egy $O(n^2)$ algoritmus egy $O(n)$ algoritmusnál hosszú távon mindig rosszabb lesz, azonban ha a két algoritmus $T(n^2 + 1)$ és $T(100n + 1000)$, akkor az $O(n^2)$ algoritmus $n \leq 110$-re jobb lesz.

Created by XMLmind XSL-FO Converter.

E hátránya ellenére az aszimptotikus elemzés az algoritmuselemzés legelterjedtebben használt módszere. Ez azért van így, mert az elemzés mind a műveletek egzakt számától (a *k* konstans tényezőt figyelmen kívül hagyva), mind a bemenet egzakt tartalmától (csupán *n* nagyságát figyelembe véve) elvonatkoztat. Ily módon az elemzés matematikailag kivitelezhetővé válik. Az $O()$ jelölés jó kompromisszum a precizitás és a könnyű elemzés között.

1.2. NP és az inherensen nehéz problémák

Az algoritmusok elemzése és az $O()$ jelölés lehetővé teszi, hogy egy konkrét algoritmus hatékonyságáról nyilatkozzunk. Arról azonban nem tudunk mondani semmit, hogy az adott problémához van-e nála jobb algoritmus. A **bonyolultságelmélet** (**complexity analysis**) területe nem az algoritmusokat, hanem inkább a problémákat elemzi. Az első nagy szakadék a polinom időben megoldható és a polinom idejű megoldást nélkülöző problémák között van, akármilyen algoritmust is használnánk. A polinom idejű problémák osztályát – azokét, amelyek $O(n^k)$ időben oldhatók meg valamilyen k esetén – P-vel jelöljük. Ezeket néha „könnyű” problémáknak hívjuk, mert az osztályhoz $O(\log n)$ és $O(n)$ futási idejű problémák is tartoznak. Idetartoznak azonban $O(n^{1000})$ problémák is, a „könnyű” elnevezést tehát túlságosan szó szerint venni nem lehet.

A problémák másik fontos osztálya az NP osztály, a nemdeterminisztikus polinom idejű problémák osztálya. Egy probléma akkor tartozik ide, ha létezik egy algoritmus, ami polinom időben meg tud találni egy megoldást, és még ellenőrizni is tudja, hogy a megtalált megoldás korrekt-e. Az ötlet az, hogy ha tetszőlegesen nagyszámú proceszszorral rendelkezünk ahhoz, hogy az összes sejtést egyszerre próbáljuk ki, vagy pedig ha szerencsések vagyunk, és mindig elsőre eltaláljuk a helyes megoldást, az NP probléma P problémává válik. A számítógépes tudomány egyik nagy nyitott kérdése az, hogy az NP osztály ekvivalens-e a P osztállyal akkor, ha nélkülözzük a végtelen processzorhalmaz, illetve a mindentudó sejtés luxusát. A legtöbb számítógépes szakember meg van győződve arról, hogy P ≠ NP, vagyis, hogy az NP problémák inherens módon nehezek, és nincsenek számukra polinom idejű algoritmusok. Ezt azonban még soha sem bizonyították be.

Akik a P = NP eldöntése iránt érdeklődnek, azok figyelmébe az NP osztály egy alosztályát, az **NP-teljes** (**NP-complete**) problémák alosztályát ajánljuk. A *teljes* szó itt a „legvégsőt” jelenti, és az NP osztály legnehezebb problémáit jelöli ki. Azt bebizonyították már, hogy vagy minden NP-teljes probléma P-ben van, vagy egyik sem. Emiatt ez az osztály elméletileg érdekes, de gyakorlati jelentősége is van, mert sok fontos problémáról tudjuk, hogy NP-teljes. Egy lehetséges példa a Boole-kifejezések kielégíthetőségének problémája: egy adott logikai kifejezés esetén létezik-e az ítéletváltozóinak olyan logikaiérték-hozzárendelése, hogy a kifejezés igaz legyen? Hacsak egy csodának nem leszünk tanúi, és a P = NP nem lesz igaz, nincs olyan algoritmus, amely *minden* kielégíthetőségi problémát polinomiális időben megold. Az MI azonban jobban érdekelt abban, hogy vajon léteznek-e hatékony algoritmusok *tipikus,* előre rögzített eloszlásokból sorsolt problémák esetén. Ahogy ezt a 7. fejezetben láttuk, vannak olyan algoritmusok, mint a `WALKSAT`, amelyek sok probléma esetén egészen jól teljesítenek.

A **co-NP** osztály az NP komplemense abban az értelemben, hogy minden NP-beli döntési probléma esetén létezik egy neki megfelelő co-NP-beli probléma felcserélt „igen” és „nem” válaszokkal. Tudjuk, hogy a P mind az NP, mind a co-NP osztályok részhalmaza, és azt gondoljuk, hogy vannak olyan co-NP-beli problémák, amelyek P-nek nem elemei. A co-NP osztály legnehezebb problémái a **co-NP-teljes** problémák.

A #P osztály az NP-beli döntési problémáknak megfelelő számlálási problémák halmaza. Döntési problémáknak igen-nem a megoldásuk: például létezik-e megoldása ennek a 3-SAT mondatnak? A számlálási problémáknak egész számok a megoldásaik; mennyi megoldása van ennek a 3-SAT mondatnak? Egyes esetekben a számlálási probléma a döntési problémánál sokkal nehezebb. Így például annak az eldöntése, hogy egy páros gráfhoz létezik-e egy tökéletes párosítása $O(VE)$ idejű (ahol a gráfnak V csúcsa és E éle van). A „hány tökéletes párosítása létezik egy konkrét páros gráf esetén” számlálási probléma viszont #P-teljes, amin azt kell érteni, hogy legalább olyan nehéz, mint a #P-beli problémák bármelyike, és így legalább olyan nehéz, mint az NP-beli problémák bármelyike.

A P-TÁR – a polinom-tárigényű – problémák osztályát szintén tanulmányozták még nemdeterminisztikus gép esetén is. Az az általános vélekedés, hogy a P-TÁR-nehéz problémák az NP-teljes problémáknál nehezebbek, bár kiderülhet, hogy NP = P-TÁR, mint ahogy az is kiderülhet, hogy P = NP.

2. A2. Vektorok, mátrixok és lineáris algebra

Created by XMLmind XSL-FO Converter.

A matematikusok egy **vektor**t egy vektortérelemként definiálnak. Mi azonban konkrétabb definíciót fogunk használni. Egy vektor az értékek egy rendezett szekvenciája. A kétdimenziós térben például olyan vektoraink vannak, mint: $\mathbf{x} = \langle 3, 4 \rangle$ és $\mathbf{y} = \langle 0, 2 \rangle$. A szokásos konvencióhoz folyamodunk vas$\vec{x}$ vagy $\bar{y}$dett betűket használva a vektorneveknek, bár egyes szerzők a nevek fölé nyilat vagy vonalat húznak: . Egy vektor elemeire indexekkel lehet hivatkozni: $\mathbf{z} = \langle z_1, z_2, \ldots, z_n \rangle$.

A vektorokon értelmezett két alapvető művelet a vektorok összeadása és egy skalárral való szorzása. Az $\mathbf{x} + \mathbf{y}$ vektorösszeadás az elemenkénti összegzés: $\mathbf{x} + \mathbf{y} = \langle 3 + 0, 4 + 2 \rangle = \langle 3, 6 \rangle$. A skalárszorzás mindegyik elemet egy konstans értékkel szorozza be: $5\mathbf{x} = \langle 5 \times 3, 5 \times 4 \rangle = \langle 15, 20 \rangle$.

Egy vektor hosszát az $|\mathbf{x}|$ jelöli, amelynek értéke egyenlő a vektor elemeinek négyzetösszegéből vett négyzetgyökével: $|\mathbf{x}| = \sqrt{\left(3^2 + 4^2\right)} = 5$. Két vektor skalárszorzata, $\mathbf{x} \cdot \mathbf{y}$ a megfelelő vektorelemek szorzatának összege, azaz $\mathbf{x} \cdot \mathbf{y} = \sum_i x_i y_i$, vagy konkrét esetünkben $\mathbf{x} \cdot \mathbf{y} = 3 \times 0 + 4 \times 2 = 8$.

A vektorokat sokszor egy *n*-dimenziós euklideszi tér irányított vonalszakaszainak (nyilaknak) képzeljük. Ekkor a vektorösszeadás annak felel meg, mintha az egyik vektor kezdő pontját a másik vektor végpontjához illesztenénk, az $\mathbf{x} \cdot \mathbf{y}$ skalárszorzat viszont $|\mathbf{x}||\mathbf{y}|\cos\theta$-vel lesz egyenlő, ahol θ az $\mathbf{x}$ és az $\mathbf{y}$ közötti szög.

Egy **mátrix** az értékek négyszögletes elrendezése sorokba és oszlopokba. Itt egy 3×4 méretű $\mathbf{m}$ mátrixot látunk:

$$\begin{pmatrix} \mathbf{m}_{1,1} & \mathbf{m}_{1,2} & \mathbf{m}_{1,3} & \mathbf{m}_{1,4} \\ \mathbf{m}_{2,1} & \mathbf{m}_{2,2} & \mathbf{m}_{2,3} & \mathbf{m}_{2,4} \\ \mathbf{m}_{3,1} & \mathbf{m}_{3,2} & \mathbf{m}_{3,3} & \mathbf{m}_{3,4} \end{pmatrix}$$

Az $\mathbf{m}_{ij}$ első indexe a sort, a második az oszlopot jelöli. Programozási nyelvekben $\mathbf{m}_{ij}$-t gyakran m[i,j], vagy m[i][j]-nek írják.

Két mátrix összegét a megfelelő elemek összeadásával definiáljuk: $(\mathbf{m} + \mathbf{n})_{ij} = \mathbf{m}_{ij} + \mathbf{n}_{ij}$. (Az összeg különböző méretű $\mathbf{m}$ és $\mathbf{n}$ mátrixokra nincs definiálva.) Egy mátrixnak egy skalárral történő szorzatát szintén definiálhatjuk: $(c\mathbf{m})_{ij} = c\mathbf{m}_{ij}$. A mátrixszorzat (két mátrix szorzata) már bonyolultabb. Az $\mathbf{mn}$ szorzat csak akkor definiált, ha az $\mathbf{m}$ mátrix $a \times b$, az $\mathbf{n}$ mátrix pedig $b \times c$ méretű (azaz a második mátrixnak annyi sora van, mint amennyi oszlopa az első mátrixnak). Az eredmény egy $a \times c$ méretű mátrix. Ez azt jelenti, hogy a mátrixszorzat nem kommutatív: általánosságban $\mathbf{mn} \neq \mathbf{nm}$. Ha a mátrixok megfelelő méretűek, akkor az eredmény:

$$(\mathbf{mn})_{i,k} = \sum_j \mathbf{m}_{i,j} \mathbf{n}_{j,k}$$

Az $\mathbf{I}$ egységmátrix $\mathbf{I}_{ij}$ elemei egységnyiek, ha $i = j$, különben nullák. Az egységmátrix tulajdonsága, hogy $\mathbf{mI} = \mathbf{m}$, minden $\mathbf{m}$-re. Az $\mathbf{m}$ transzponáltját, $\mathbf{m}^{\mathrm{T}}$-t, megkapjuk, ha az $\mathbf{m}$ sorait és oszlopait felcseréljük, formálisan: $\mathbf{m}^T_{i,j} = \mathbf{m}_{i,j}$.

A mátrixokat lineáris egyenletrendszerek megoldására a **Gauss–Jordan-eliminációs** (**Gauss–Jordan elimination**) algoritmussal használjuk, ami egy $O(n^3)$ komplexitású algoritmus. Tekintsünk az alábbi egyenletrendszert, amelynek *x*, *y* és *z* megoldására vagyunk kíváncsiak:

```
+2x + y - z = 8

-3x - y + 2z = -11

-2x + y + 2z = -3
```

Ezt az egyenletrendszert kifejezhetjük mátrixos formában is:

Created by XMLmind XSL-FO Converter.

$$\begin{matrix} x & y & z & c \end{matrix}$$
$$\begin{pmatrix} 2 & 1 & -1 & 8 \\ -3 & -1 & 2 & -1 \\ -2 & 1 & 2 & -3 \end{pmatrix}$$

Itt az *x y z c* nem része a mátrixnak, csak emlékeztetőül szolgál. Tudjuk, hogy ha egy egyenlet mindkét oldalát egy konstanssal megszorozzuk, vagy ha két egyenletet egymáshoz hozzáadunk, akkor szintén egy érvényes egyenlethez jutunk el. A Gauss– Jordan-elimináció ezeket a lépéseket ismételten alkalmazza úgy, hogy először az első változót (az *x*-et) eliminálju k, az elsőt kivéve az összes egyenletből, majd folytatjuk minden *i*-re, az *i*-edik változót minden egyenletből eliminálva, az *i*-edik egyenletet kivéve. Az *x*-et a második egyenletből úgy eliminálju k, hogy az első egyenletet 3/2-vel megszorozzuk, az eredményt pedig a második egyenlethez hozzáadjuk. Ezzel az alábbi mátrixhoz jutunk el:

$$\begin{matrix} x & y & z & c \end{matrix}$$
$$\begin{pmatrix} 2 & 1 & -1 & 8 \\ 0 & 0,5 & 0,5 & 1 \\ 2 & 1 & 2 & -3 \end{pmatrix}$$

Folytatjuk így eliminálva az *x*-et, az *y*-t és a *z*-t, míg végül a:

$$\begin{matrix} x & y & z & c \end{matrix}$$
$$\begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & -1 \end{pmatrix}$$

mátrixot kapjuk, amely jelzi, hogy a megoldás $x = 2$, $y = 3$, $z = -1$. (Próbálja ki!)

3. A3. Valószínűségi eloszlások

A valószínűség az eseményhalmazokon definiált olyan mérték, amely az alábbi három axiómát elégíti ki:

1. Minden esemény mértéke egy 0 és 1 közötti szám. Ezt a következőképpen írjuk: $0 \leq P(E = e_i) \leq 1$, ahol az E egy eseményt képviselő véletlen változó, és az e_i-k az E lehetséges értékei. Általánosságban a véletlen változókat nagybetűkkel, az értékeit pedig kisbetűkkel jelöljük.
2. Az egész halmaz mértéke 1; azaz $\sum_{i=1}^{n} P\left(E = e_i\right) = 1$
3. Diszjunkt halmazok uniójának a valószínűsége az egyedi események valószínűségeinek összege, azaz $P(E = e_1 \vee E = e_2) = P(E = e_1) + P(E = e_2)$, ahol e_1 és e_2 diszjunktak.

Egy **valószínűségi modell** (**probabilistic model**) a kölcsönösen egymást kizáró, lehetséges kimenetelek mintateréből áll, és minden kimenetel valószínűségi mértékéből. A holnapi időjárás modelljében például a kimenetelek a következők lehetnek: *napos*, *felhős*, *esős* vagy *havas*. Ezen kimenetelek egy részhalmaza képvisel egy eseményt. A csapadékesemény például az {*esős*, *havas*} részhalmaz.

A $\langle P(E = e_1), \ldots, P(E = e_n) \rangle$ értékekből képzett vektor jelölésére a $\mathbf{P}(E)$-t fogjuk használni. A $P(e_i)$ jelölést szintén használni fogjuk a $P(E = e_i)$ rövidítésére, hasonlóan a $\sum_{i=1}^{n} P\left(E = e_i\right)$-t $\sum_{e} P\left(e\right)$-ként rövidítjük.

A $P(B|A)$ feltételes valószínűséget a $P(B \cap A)/P(A)$ definiálja. *A* és *B* feltételesen függetlenek, ha $P(B|A) = P(B)$ [vagy ekvivalens módon $P(A|B) = P(A)$]. Folytonos változók esetén az értékek halmaza végtelen, és hacsak pontszerű tüskék nincsenek, mindegyik érték valószínűsége 0. Emiatt egy **valószínűségi sűrűségfüggvény**t (**probability density function**) definiálunk, amit szintén $P(X)$-szel fogunk jelölni. $P(X)$ jelentése a $P(A)$ diszkrét valószínűségi függvénytől kissé eltér. A $P(X = c)$ sűrűségfüggvény-érték az a valószínűség, hogy az X a c körüli intervallumba esik, elosztva az intervallum hosszával, és határátmenetet képezve, ahogy az intervallum hossza tart a 0-hoz:

Created by XMLmind XSL-FO Converter.

$$P(X=c)=\lim_{dx\to 0} P(c\le X\le c+dx)/dx$$

A sűrűségfüggvénynek nemnegatívnak kell lennie minden *x*-re, és teljesítenie kell, hogy:

$$\int_{-\infty}^{\infty} P(X)dx=1$$

Definiálhatunk egy **valószínűségi eloszlásfüggvény**t (**cumulative probability density function**), *F*(*X*)-et is, ami annak a valószínűsége, hogy a véletlen változó *x*-nél kisebb:

$$F(X)=\int_{-\infty}^{X} P(Z)dz$$

Jegyezzük meg, hogy a valószínűségi sűrűségfüggvénynek létezik egysége, az eloszlásfüggvény viszont egység nélküli. Ha az *X*-et például másodpercben mérjük, akkor a sűrűséget Hz-ben (azaz 1/s) fogjuk. Ha **X** a méterben mért háromdimenziós tér egy pontja, akkor a sűrűséget $1/m^3$-ben mérjük.

Az egyik legfontosabb valószínűség-eloszlás a **Gauss-eloszlás** (**Gaussian distribution**), amit **normális eloszlás**nak (**normal distribution**) is szokás nevezni. A μ középértékű és σ szórású (σ^2 varianciájú) Gauss-eloszlás definíciója:

$$P(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{-(x-\mu)^2/(2\sigma^2)}$$

ahol *x* egy folytonos változó $-\infty$-től $+\infty$-ig terjedő értékkészlettel. $\mu = 0$ és $\sigma^2 = 1$ mellett kapjuk a **standard normális eloszlás** (**standard normal distribution**) speciális esetet. Egy *d* dimenziós **x** vektor feletti eloszlás a **többváltozós Gauss-eloszlás** (**multivariate Gaussian distribution**):

$$P(\mathbf{x})=\frac{1}{\sqrt{(2\pi)^n|\Sigma|}}e^{-\frac{1}{2}\left((\mathbf{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)}$$

ahol $\boldsymbol{\mu}$ az eloszlás középértékvektora, $\boldsymbol{\Sigma}$ pedig a **kovarianciamátrix** (**covariance matrix**).

A standard normális eloszlás egydimenziós eloszlásfüggvénye:

$$F(x)=\int_{-\infty}^{x} P(x)dx=\frac{1}{2}\left(1+\mathrm{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right)$$

ahol az erf(*x*) az ún. **hibafüggvény** (**error function**), amelynek zárt alakja nincs.

A **központi határeloszlás tétele** (**central limit theorem**) azt mondja, hogy *n* véletlen változó középértékének az eloszlása a normális eloszláshoz tart, ha *n* tart végtelenhez. Igaz ez a véletlen változók majdnem minden halmazára, hacsak valamelyik véges alhalmaz varianciája a többit nem fogja dominálni.

3.1. Irodalmi és történeti megjegyzések

A számítógépes tudományokban ma oly elterjedten használt *O*() jelölést (nagy ordó) P. G. H. Bachmann német matematikus vezette be a számelmélettel kapcsolatban (Bachmann, 1894). Az NP-teljesség koncepcióját Cook (Cook, 1971) dolgozta ki, míg a problémák egymásra redukálásának modern módszertanát Karpnak köszönhetjük (Karp, 1972). Munkásságukért mindketten Turing-díjban – a számítógépes tudományok legmagasabb elismerésében – részesültek.

Az algoritmusok elemzésére és tervezésére irányuló klasszikus munkák között találjuk a (Knuth, 1973; Aho és társai, 1974) műveket. Újabb adalékok Tarjantól (Tarjan, 1983) és a Cormen, Leiserson és Rivest szerzőhármastól (Cormen és társai, 1990) származnak. Ezekben a művekben a hangsúlyt a kezelhető problémákat megoldó algoritmusok tervezésére és elemzésére helyezték. Az NP-teljességnek, illetve a kezelhetetlenség más formáinak az elméletébe a legjobb bevezetők (Garey és Johnson, 1979) és (Papadimitriou, 1994). Az elmélet tárgyalásán túlmenően Garey és Johnson meggyőzően, példákkal illusztrálják, hogy a

Created by XMLmind XSL-FO Converter.

számítógépes szakemberek miért a polinom és az exponenciális futási időigény határára teszik egyhangúan a kezelhető és kezelhetetlen problémák közötti határvonalat. A szerzők az NP-teljesként vagy más szempontból kezelhetetlennek ismert problémák terjedelmes katalógusát is mellékelik.

A valószínűség-számításról jó művek (Chung, 1979; Ross, 1988; Bertsekas–Tsitsiklis, 2002; Feller, 1971).

Created by XMLmind XSL-FO Converter.

B. függelék - Megjegyzések a nyelvekről és az algoritmusokról

1. B1. Nyelvek definiálása Backus–Naur-Formában (BNF)

Jelen könyvben definiáltunk néhány nyelvet, az ítéletlogika nyelvét (3. szakasz - A logika rész), az elsőrendű logika nyelvét (2.2. szakasz - Szimbólumok és interpretációk rész) és az angol nyelv részhalmazát (4. szakasz - Kiterjesztett nyelvtanok rész). Egy formális nyelvet mondathalmazként definiálunk, ahol egy-egy mondat a szimbólumok halmaza. A számunkra érdekes nyelvek mindegyike végtelen sok mondatból áll, következésképpen szükségünk van e halmaz tömör jellemzésére. A jellemzést egy **nyelvtan** (**grammar**) segítségével kíséreljük meg. Nyelvtanunkat **Backus–Naur-forma** (**Backus– Naur form**) vagy **BNF**-formalizmusban fejezzük ki. Egy BNF-nyelvtannak négy komponense van:

- A **zárószimbólumok** vagy a **terminálisok** (**terminal symbols**) halmaza. Ezekből a szimbólumokból, másképpen szavakból, állnak össze a nyelv mondatai. Ezek lehetnek betűk (**A**, **B**, **C**, …) vagy szavak (**a**, **aardvark**, **abacus**, ...).
- A nyelv részkifejezéseit kategorizáló **nemzáró szimbólumok** vagy **nemterminálisok** (**nonterminal symbols**) halmaza. Az angol nyelvben például a *FőnéviKifejezés (NounPhrase*) nemzáró szimbólum a „you” és a „the big slobbery dog” mondatokat is tartalmazó végtelen halmazt jelenti.
- A **kezdő szimbólum** (**start symbol**) a nyelv teljes mondatait jelentő nemzáró szimbólum. Az angol nyelv esetén ez a *Mondat* (*Sentence*), az aritmetika esetén ilyen szimbólum lehet a *Kifejezés* (*Expr*).
- A **képzési szabályok** (**rewrite rules**) halmaza, ahol egy-egy szabály *LHS* → *RHS* formájú. *LHS* egy nemzáró szimbólum, az *RHS* pedig 0 vagy több (záró vagy nemzáró) szimbólumból állhat.

A

```
Mondat → FőnéviKifejezés IgeiKifejezés
```

alakú képzési szabály azt jelenti, hogy ha bármikor is van két, egy *FőnéviKifejezés*-ként és egy *IgeiKifejezés*-ként kategorizált mondatunk, akkor azokat összekapcsolhatjuk, és az eredményt *Mondat*-ként kategorizálhatjuk. A | szimbólum az alternatív jobb oldali kifejezéseket választja szét. Nézzük most az egyszerű aritmetikai kifejezések BNF- nyelvtanát:

```
Kifejezés → Kifejezés Operátor Kifejezés | (
          Kifejezés ) | Szám

Szám      → Számjegy | Szám Számjegy

Számjegy  → 0|1|2|3|4|5|6|7|8|9

Operátor  → + | − | ÷ | ×
```

A nyelvekkel és a nyelvtanokkal bővebben a 22. fejezetben foglalkoztunk. Előfordul, hogy más forrásokban a BNF-re kissé módosított jelölést használnak. Nemzáró szimbólum esetén ⟨ *Számjegy*⟩ -gyel találkozhatunk a *Számjegy* helyett, a zárószimbólum esetén ’szó’-val a **szó** helyett, illetve szabályokban a : : = jelöléssel a → helyett.

2. B2. Az algoritmusok leírása pszeudokóddal

Created by XMLmind XSL-FO Converter.

A könyvünkben több mint 80 algoritmust definiáltunk részletesen. Ahelyett hogy egy programozási nyelvet választottuk volna (és ahelyett, hogy azt kockáztattuk volna, hogy a nyelvet nem ismerő olvasó elveszti a fonalat), úgy döntöttünk, hogy pszeudokóddal írjuk le az algoritmusokat. Ezek legtöbbje érthető lesz azoknak, akik Javát, C++-t, Lispet vagy hasonló nyelveket használnak. Helyenként azonban matematikai formulákat, illetve mindennapi nyelvet használunk olyan részek leírásához, amelyek máskülönben nehezebben lennének olvashatók. Néhány egyéni vonásról is szót kell ejtenünk.

Statikus változók. A **static** kulcsszót egy olyan változó megjelölésére használjuk, amely kezdeti – vagy a rákövetkező értékadásból adódó – értékét a függvény első hívásánál kapja meg, és megtartja a függvény minden azt követő hívásánál. A statikus változó tehát hasonlít egy globális változóra abban, hogy a függvény egyedi hívását túléli, értéke azonban csak a függvényen belül férhető hozzá. A könyv ágensprogramjai statikus változókat „memória” létesítésére használnak. A statikus változókat kezelő programok „objektumként” implementálhatók olyan objektumorientált nyelvekben, mint a Java vagy a Smalltalk. A funkcionális nyelvekben az implementálásuk könnyen oldható meg a lambda-kifejezés végrehajtásával egy olyan környezeten belül, amelyben a szükséges változót definiáltuk.

Függvények mint értékek. A függvények és eljárások jelölésére nagybetűs neveket, a változók jelölésére pedig kis-, dőlt betűs neveket használunk. Egy függvényhívás legtöbb esetben tehát `FN`(*x*) képet nyújt. Megengedjük azonban, hogy egy változó függvényértéket is felvehessen. Ha az *f* értéke például a négyzetgyökfüggvény, akkor az *f* (9) által visszaadott érték a 3.

A tömbök 1-től kezdődnek. Ha másképpen nincs specifikálva, egy tömb első indexe mindig 1, ahogy ez a matematikában szokás, és nem 0, mint a Javában vagy a C-ben.

A beljebb szedett bekezdés fontos. A beljebb szedett bekezdést a hurok vagy a feltételes kifejezés hatáskörének megjelölésére használjuk, mint például a Python nyelvben, ellentétben a Javával és a C++-szal (ahol a zárójelezés használatos), illetve a Pascallal és a Visual Basickel (ahol egy **end** használatos).

3. B3. Online segédnyújtás

Fontos

A könyvben található algoritmusok többségét online kódtárunkban implementáltuk, amelynek címe:

aima.cs.berkeley.edu

Fontos

Ha Önnek megjegyzései lennének a könyvünkkel kapcsolatban, hibákat vett észre, illetve ha ötletei vannak, amelyekkel a könyv javítható, keressen meg bennünket. Kérjük, látogassa meg a weboldalunkon a vitafórumunkat, vagy küldjön e-mail üzenetet a következő címre:

aima@cs.berkeley.edu

Created by XMLmind XSL-FO Converter.

C. függelék - Irodalomjegyzék

Aarup, M., Arentoft, M. M., Parrod, Y., Stadler, J., and Stokes, I. (1994). OPTIMUM-AIV: A knowledge-based planning and scheduling system for spacecraft AIV. In Fox, M. and Zweben, M. (Eds.), *Knowledge Based Scheduling.* Morgan Kaufmann, San Mateo, California.

Abramson, B. and Yung, M. (1989). Divide and conquer under global constraints: A solution to the N-queens problem. *Journal of Parallel and Distributed Computing, 6*(3), 649–662.

Ackley, D. H. and Littman, M. L. (1991). Interactions between learning and evolution. In Langton, C., Taylor, C. Farmer, J. D., and Ramussen, S. (Eds.) *Artificial Life II,* pp. 487–509. Addison-Wesley, Redwood City, California.

Adelson-Velsky, G. M., Arlazarov, V. L., Bitman, A. R., Zhivotovsky, A. A., and Uskov, A. V. (1970). Programming a computer to play chess. *Russian Mathematical Surveys, 25,* 221–262.

Adelson-Velsky, G. M., Arlazarov, V. L., and Donskoy, M. V. (1975). Some methods of controlling the tree search in chess programs. *Artificial Intelligence, 6*(4), 361–371.

Agmon, S. (1954.) The relaxation method for linear inequalities. *Canadian J. Math., 6*(3), 382–392.

Agre, P. E. and Chapman, D. (1987). Pengi: an implementation of a theory of activity. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87),* pp. 268–72, Milan. Morgan Kaufman.

Aho, A. V., Hopcroft, J., and Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Massachusetts.

Aho, A. V. and Ullman, J. D. (1972). *The Theory of Parsing, Translation and Compiling.* Prentice-Hall, Upper Saddle River, New Jersey.

Ait-Kaci, H. and Podelski, A. (1993). Towards a meaning of LIFE. *Journal of Logic Programming, 16*(3–4), 195–234.

Aizerman, M., Braverman, E., and Rozonoer, L. (1964). Theoretical foundations of the potential functions method in pattern recognition learning. *Automation and Remote Control, 25,* 821–837.

Albus, J. S. (1975). A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement, and Control, 97,* 270–277.

Aldous, D. and Vazirani, U. (1994). „Go with the winners" algorithms. In *Procceedings of the 35th Annual Symposiumon Foundations of Computer Science,* pp. 492–501, Santa Fe, New Mexico. IEEE Computer Society Press.

Allais, M. (1953). Le comportment de l'homme rationnel devant la risque: critique des postulats et axiomes de l'école Américaine. *Econometrica, 21,* 503–546.

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the Association for Computing Machinery, 26*(11), 832–843.

Allen, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence, 23,* 123–154.

Allen, J. F. (1991). Time and time again: The many ways to represent time. *International Journal of Intelligent Systems, 6,* 341–355.

Allen, J. F. (1995). *Natural Language Understanding.* Benjamin/Cummings, Redwood City; California.

Allen, J. F., Hendler, J., and Tate, A. (Eds.). (1990). *Readings in Planning.* Morgan Kaufmann, San Mateo, California.

Created by XMLmind XSL-FO Converter.

Almuallim, H. and Dietterich, T. (1991). Learning with many irrelevant features. In *Proccedings of the Ninth National Conference on Artificial Intelligence (AAAI-91),* Vol. 2, pp. 547–552, Anaheim, California. AAAI Press.

ALPAC (1966). Language and machines: Computers in translation and linguistics. Tech. rep. 1416, The Automatic Language Processing Advisory Committee of the National Academy of Sciences, Washington, DC.

Alshawi, H. (Ed.). (1992). *The Core Language Engine.* MIT Press, Cambridge, Massachusetts.

Alterman, R. (1988). Adaptive planning. *Cognitive Science, 12,* 393–422.

Amarel, F. (1998). On representations of problems of reasoning about actions. In Michie, D. (Ed.), *Machine Intelligence 3,* Vol. 3, pp. 131–171. Elsevier/North-Holland, Amsterdam, London, New York.

Ambros-Ingerson, J. and Steel, S. (1988). Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88),* pp. 735–740, St. Paul, Minnesota. Morgan Kaufmann.

Amit, D., Gutfreund, H., and Sompolinsky, H. (1985). Spin-glass models of neural networks. *Physical Review, A 32,* 1007–1018.

Andersen, S. K., Olesen, K. G., Jensen, F. V., and Jensen F. (1989). HUGIN – a shell for building Bayesian belief universes for expert systems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89),* Vol. 2, pp. 1080–1085. Detroit. Morgan Kaufmann.

Anderson, A. R. (Ed.) 1964). *Minds and Machines.* Prentice-Hall, Upper Saddle River, New Jersey.

Anderson, J. A. and Rosenfeld, E. (Eds.). (1988). *Neurocomputing: Foundations of Research.* MIT Press, Cambridge, Massachusetts.

Anderson, J. R. (1980). *Cognitive Psychology and Its Implications.* W. H. Freeman, New York.

Anderson, J. R. (1993). *The Architecture of Cognition.* Harvard University Press, Cambridge, Massachusetts.

Andre, D. and Russell, S. J. (2002). State abstraction for programmable reinforcement learning agents. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02),* pp. 119–125, Edmonton, Alberta. AAAI Press.

Anshelevich, V. A. (2000). The game of Hex: An automatic theorem proving approach to game programming. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00),* pp. 189–194. Austin, Texas. AAAI Press.

Anthony, M. and Bartlett, P. (1999). *Neural Network Learning: Theoretical Foundations.* Cambridge University Press, Cambridge, UK.

Appel, K. and Haken, W. (1977). Every planar map is four colorable: Part I: Discharging. *Illinois J. Math., 21,* 429–490.

Apt, K. R. (1999). The essence of constraint propagation. *Theoretical Computer Science, 221* (1–2), 179–210.

Apté, C. Damerau, F., and Weiss, S. (1994). Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems, 12,* 233–251.

Arkin, R. (1998). *Behavior-Based Robotics.* MIT Press, Boston, MA.

Armstrong, D. M. (1968). *A Materialist Theory of the Mind.* Routledge and Kegan Paul, London.

Arnauld, A. (1662). *La logique, ou l'art de penser.* Chez Charles Savreux, au pied de la Tour de Nostre Dame, Paris.

Arora, S. (1998). Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the Association for Computing Machinery, 45*(5), 753–782.

Created by XMLmind XSL-FO Converter.

Ashby, W. R. (1940). Adaptiveness and equilibrium. *Journal of Mental Science, 86,* 478–483.

Ashby, W. R. (1948). Design for a brain. *Electronic Engineering, December,* 379–383.

Ashby, W. R. (1952). *Design for a Brain.* Wiley, New York.

Asimov, I. (1942). Runaround. *Astounding Science Fiction, March.*

Asimov, I. (1950). *I. Robot.* Doubleday, Garden City, New York.

Astrom, K. J. (1965). Optimal control of Markov decision processes with incomplete state estimation. *J. Math. Anal. Applic., 10,* 174–205.

Audi, R. (Ed.). (1999). *The Cambridge Dictionary of Philosophy.* Cambridge University Press, Cambridge, UK.

Austin, J. L. (1962). *How To Do Things with Words.* Harvard University Press, Cambridge, Massachusetts.

Axelrod, R. (1985). *The Evolution of Cooperation.* Basic Books, New York.

Bacchus, F. (1990). *Representing and Reasoning with Probabilistic Knowledge.* MIT Press, Cambridge, Massachusetts.

Bacchus, F. and Grove, A. (1995). Graphical models for preference and utility. In *Uncertainty in Artificial Intelligence: Proceedings of the Eleventh Conference,* pp. 3–10, Montreal, Canada, Morgan Kaufmann.

Bacchus, F. and Grove, A. (1996). Utility independence in a qualitative decision theory. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning,* pp. 542–552, San Mateo, California, Morgan Kaufmann.

Bacchus, F. Grove, A., Halpern, J. Y., and Koller, D. (1992). From statistics to beliefs. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92),* pp. 602–608, San Jose. AAAI Press.

Bacchus, F. and van Beek, P. (1998). On the conversion between non-binary and binary constraint satisfaction problems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98),* pp. 311–318, Madison, Wisconsin. AAAI Press.

Bacchus, F. and van Run, P. (1995). Dynamic variable ordering in CSPs. In *Proceedings of the First International Conference on Principles and Practice of Constraint Programming,* pp. 258–275, Cassis, France. Springer-Verlag.

Bachmann, P. G. H. (1894). *Die analytische Zahlentheorie.* B. G. Teubner, Leipzig.

Backus, J. W. (1996). Transcript of question and answer session. In Wexelblat, R. L. (Ed.), *History of Programming Languages,* p. 162. Academic Press, New York.

Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval.* Addison Wesley Longman, Reading, Massachusetts.

Bajcsy, R. and Lieberman, L. (1976). Texture gradient as a depth cue. *Computer Graphics and Image Processing, 5*(1), 52–67.

Baker, C. L. (1989). *English Syntax.* MIT Press, Cambridge, Massachusetts.

Baker, J. (1975). The Dragon system – overview. *IEEE Transactions on Acoustics, Speech, and Signal processing, 23,* 24–29.

Baker, J. (1979). Trainable grammars for speech recognition. In *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America,* pp. 547–550, Cambridge, Massachusetts. MIT Press.

Baldwin, J. M. (1896). A new factor in evolution. *American Naturalist, 30,* 441–451. Continued on pages 536–553.

Created by XMLmind XSL-FO Converter.

Ballard, B. W. (1983). The *-minimax search procedure for trees containing chance nodes. *Artificial Intelligence, 21*(3), 327–350.

Baluja, S. (1997). Genetic algorithms and explicit search statistics. In Mozer, M. C. Jordan, M. I., and Petsche, T. (Eds.), *Advances in Neural Information Processing Systems,* Vol. 9. pp. 319–325. MIT Press, Cambridge, Massachusetts.

Bancilhon, F. Maier, D., Sagiv, Y., and Ullman, J. D. (1986). Magic sets and other strange ways to implement logic programs. In *Proceedings of the Fifth ACM Symposium on Principles of Database Systems,* pp. 1–16. New York. ACM Press.

Bar-Hillel, Y. (1954). Indexical expressions. *Mind, 63,* 359–379.

Bar-Hillel, Y. (1960). The present status of automatic translation of languages. In Alt, F. L. (Ed.), *Advances in Computers,* Vol. 1, pp. 91–163. Academic Press, New York.

Bar-Shalom, Y. (Ed.). (1972). *Multitarget-multisensor tracking: Advanced applications.* Artech House, Norwood, Massachusetts.

Bar-Shalom, Y. and Fortmann, T. E. (1988). *Tracking and Data Association.* Academic Press, New York.

Barrett, A. and Weld, D. S. (1994). Task-decomposition via plan parsing. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94),* pp. 1117–1122, Seattle. AAAI Press.

Bartak, R. (2001). Theory and practice of constraint propagation. In *Proceedings of the Third Workshop on Constraint Programming for Decision and Control (CPDC-01),* pp. 7–14, Gliwice, Poland.

Barto, A. G. Bradtke, S. J., and Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence, 73*(1), 81–138.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics, 13,* 834–846.

Barto, A. G., Sutton, R. S., and Brouwer, P. S. (1981). Associative search network: A reinforcement learning associative memory. *Biological Cybernetics, 40*(3), 201–211.

Barton, G. E. Berwick, R. C., and Ristad, E. S. (1987). *Computational Complexity and Natural Language.* MIT Press, Cambridge, Massachusetts.

Barwise, J. and Etchemendy, J. (1993). *The Language of First-Order Logic: Including the Macintosh Program Tarski's World 4.0* (Third Revised and Expanded edition). Center for the Study of Language and Information (CSLI), Stanford, California.

Bateman, J. A. (1997). Enabling technology for multilingual natural language generation: The KPML development environment. *Natural Language Engineering, 3*(1), 15–55.

Bateman, J. A., Kasper, R. T., Moore, J. D., and Whitney, R. A. (1989). A general organization of knowledge for natural language proecessing: The penman upper model. Tech. rep., Information Sciences Institute, Marina del Rey, CA.

Baum, E., Boneh, D., and Garrett, C. (1995). On genetic algorithms. In *Proceedings of the Eighth Annual Conference on Computational Learning Theory (COLT-92),* pp. 230–239, Santa Cruz, California. ACM Press.

Baum, E. and Haussler, D. (1989). What size net gives valid generalization?. *Neural Computation, 1*(1), 151–160.

Baum, E. and Smith, W. D. (1997). A Bayesian approach to relevance in game playing. *Artificial Intelligence, 97*(1–2), 195–242.

Baum, E. and Wilczek, F. (1988). Supervised learning of probability distributions by neural networks. In Anderson, D. Z. (Ed.), *Neural Information Processing Systems,* pp. 52–61. American Institute of Physics, New York.

Created by XMLmind XSL-FO Converter.

Baum, L. E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics, 41*.

Baxter, J. and Bartlett, P. (2000). Reinforcement learning in POMDP's via direct gradient ascent. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 41–48, Standford, California. Morgan Kaufmann.

Bayardo, R. J. and Schrag, R. C. (1997). Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97).* pp. 203–208. Providence, Rhode Island. AAAI Press.

Bayes, T. (1763). An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London, 53*, 370–418.

Beal, D. F. (1980). An analysis of minimax. In Clarke, M. R. B. (Ed.). *Advances in Computer Chess 2*, pp. 103–109. Edinburgh University Press, Edinburgh, Scotland.

Beal, D. F. (1990). A generalised quiescence search algorithm. *Artificial Intelligence, 43*(1), 85–98.

Beckert, B. and Posegga, J. (1995). Leantap: Lean, tableau-based deduction. *Journal of Automated Reasoning, 15*(3), 339–358.

Beeri, C. Fagin, R., Maier, D., and Yannakakis, M. (1983). On the desirability of acyclic database schemes. *Journal of the Association for Computing Machinery, 30*(3), 479–513.

Bell, C. and Tate, A. (1985). Using temporal constraints to restrict search in a planner. In *Proceedings of the Third Alvey IKBS SIG Workshop*, Sunningdale, Oxfordshire, UK. Institution of Electrical Engineers.

Bell, J. L. and Machover, M. (1977). *A Course in Mathematical Logic.* Elservier/North-Holland, Amsterdam, London, New York.

Bellman, R. E. (1978). *An Introduction to Artificial Intelligence: Can Computers Think?* Boyd & Fraser Publishing Company, San Francisco.

Bellman, R. E. and Dreyfus, S. E. (1962). *Applied Dynamic Programming.* Princeton University Press, Princeton, New Jersey.

Bellman, R. E. (1957). *Dynamic Programming.* Princeton University Press, Princeton, New Jersey.

Belongie, S., Malik, J., and Puzicha, J. (2002). Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 24*(4), 509–522.

Bender, E. A. (1996). *Mathematical methods in artificial intelligence.* IEEE Computer Society Press, Los Alamitos, California.

Bentham, J. (1823). *Principles of Morals and Legislation.* Oxford University Press, Oxford, UK. Original work published in 1789.

Berger, J. O. (1985). *Statistical Decision Theory and bayesian Analysis.* Springer Verlag, Berlin.

Berlekamp, E. R. Conway, J. H., and Guy, R. K. (1982). *Winning Ways, For Your Mathematical Plays.* Academic Press, New York.

Berleur, J. and Brunnstein, K. (2001). *Ethics of Computing: Codes, Spaces for Discussion and Law.* Chapman and Hall, London.

Berliner, H. J. (1977), BKG – a program that plays backgammon. Tech. rep., Computer Science Department, Carnegie-Mellon University, Pittsburgh.

Berliner, H. J. (1979). The B* tree search algorithm: A best-first proof procedure. *Artificial Intelligence, 12*(1), 23–40.

Created by XMLmind XSL-FO Converter.

Berliner, H. J. (1980a). Backgammon computer program beats world champion. *Artificial Intelligence, 14,* 205–220.

Berliner, H. J. (1980b). Computer backgammon. *Scientific American, 249*(6), 64–72.

Berliner, H. J. and Ebeling, C. (1989). Pattern knowledge and search: The SUPREM architecture. *Artificial Intelligence, 38*(2), 161–198.

Bernardo, J. M. and Smith, A. F. M. (1994). *Bayesian Theory.* Wiley, New York.

Berners-Lee, T. Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American, 284*(5), 34–43.

Bernoulli, D. (1738). Specimen theoriae novae de mensura sortis. *Proceedings of the St. Petersburg Imperial Academy of Sciences, 5,* 175–192.

Bernstein, A. and Roberts, M. (1985). Computer vs. chess player. *Scientific American, 198*(6), 96–105.

Bernstein, A., Roberts, M., Arbuckle, T., and Belsky, M. S. (1958). A chess playing program for the IBM 704. In *Proceedings of the 1958 Western Joint Computer Conference,* pp. 157–159, Los Angeles. American Institute of Electrical Engineers.

Bernstein, P. L. (1996). A*gainst the Odds: The Remarkable Story of Risk.* Wiley, New York.

Berrou, C., Glavieux, A., and Thitimajshima, P. (1993). Near Shannon limit error control-correcting coding and decoding: Turbo-codes. I. In *Proc. IEEE International Conference on Communications,* pp. 1064–1070, Geneva, Switzerland. IEEE.

Berry, D. A. and Fristedt, B. (1985). *Bandit Problems: Sequential Allocation of Experiments.* Chapman and Hall, London.

Bertele, U. and Brioschi, F. (1972). *Nonserial dynamic programming.* Academic Press, New York.

Bertoli, P. Cimatti, A., and Roveri, M. (2001a). Heuristic search + symbolic modell checking = efficient conformant planning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01),* pp. 467–472. Seattle, Morgan Kaufmann.

Bertoli, P. Cimatti, A., Roveri, M., and Traverso, P. (2001b). Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01),* pp. 473–478. Seattle. Morgan Kaufmann.

Bertsekas, D. (1987). *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, Upper Saddle River, New Jersey.

Bertsekas, D. and Tsitsiklis, J. N. (1996). *Neurodynamic programming.* Athena Scientific, Belmont, Massachusetts.

Bertsekas, D. and Tsitsiklis, J. N. (2002). *Introduction to Probability.* Athena Scientific, Belmont, Massachusetts.

Bibel, W. (1981). On matrices with connections. *Journal of the Association for Computing Machinery, 28*(4), 633–645.

Bibel, W. (1993). *Deduction: Automated Logic.* Academic Press, London.

Biggs, N. L., Lloyd, E. K., and Wilson, R. J. (1986). *Graph Theory 1736–1936.* Oxford University Press, Oxford, UK.

Binder, J., Koller, D., Russelll, S. J., and Kanazawa, K. (1997a). Adaptive probabilistic networks with hidden variables. *Machine Learning, 29,* 213–244.

Binder, J. Murphy, K., and Russel, S. J. (1997b). Space-efficient inference in dynamic probabilistic networks. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97),* pp. 1292–1296, Nagoya, Japan. Morgan Kaufmann.

Created by XMLmind XSL-FO Converter.

Binford, T. O. (1971). Visual perception by computer. Invited paper presented at the IEEE Systems Science and Cybernetics Conference, Miami.

Binmore, K. (1982). *Essays on Foundations of Game Theory*. Pitman, London.

Birnbaum, L. and Selfridge, M. (1981). Conceptual analysis of natural language. In Schank, R. and Riesbeck, C. (Eds.), *Inside Computer Understanding.* Lawrence Erlbaum, Potomac, Maryland.

Biro, J. I. and Shahan, R. W. (Eds.) (1982). *Mind, Brain and Function: Essays in the Philosophy of Mind.* University of Oklahoma Press, Norman, Oklahoma.

Birtwistle, G. Dahl, O.-J., Myrhaug, B. and Nygaard, K. (1973). *Simula Begin.* Studentliteratur (Lund) and Auerbach, New York.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition.* Oxford University Press, Oxford, UK.

Bistarelli, S. Montanari, U., and Rossi, F. (1997). Semiring-based constraint satisfaction and optimization. *Journal of the Association for Computing Machinery*, *44*(2), 201–236.

Bitner, J. R. and Reingold, E. M. (1975). Backtrack programming techniques. *Communications of the Association for Computing Machinery*, *18*(11), 651–656.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2001). Latent Dirichlet Allocation. In *Neural Information Processing Systems*, Vol. 14, Cambridge, Massachusetts, MIT Press.

Blinder, A. S. (1983). Issues in the coordination of monetary and fiscal policies. In *Monetary Policy Issues in the 1980s.* Federal Reserve Bank, Kansas City, Missouri.

Block, N. (ed.) (1980). *Readings in Philosophy of Psychology*, Vol. 1. Harvard University Press, Cambridge, Massachusetts.

Blum, A. L. and Furst, M. (1995). Fast planning through planning graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pp. 1636–1642, Montreal Morgan Kaufmann.

Blum, A. L. and Furst, M. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, *90*(1–2), 281–300.

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, *36*(4), 929–965.

Bobrow, D. G. (1967). Natural language input for a computer problem solving system. In Minsky, M. L. (Ed.), *Semantic Information Processing*, pp. 133–215. MIT Press, Cambridge, Massachusetts.

Bobrow, D. G., Kaplan, R., Kay, M., Norman, D. A., Thompson, H., and Winograd, T. (1977). GUS, a frame driven dialog system. *Artificial Intelligence*, *8*, 155–173.

Bobrow, D. G. and Raphael, B. (1974). New programming languages for artificial intelligence research. *Computing Surveys*, *6*(3), 153–174.

Boden, M. A. (1977). *Artificial Intelligence and Natural Man.* Basic Books, New York.

Boden, M. A. (Ed.) (1990). *The Philosophy of Artificial Intelligence*, Oxford University Press, Oxford, UK.

Bonet, B. and Geffner, H. (1999). Planning as heuristic search: New results. In *Proceedings of the European Conference on Planning*, pp. 360–372, Durham, UK. Springer-Verlag.

Bonet, B. and Geffner, H. (2000). Planning with incomplete information as heuristic search in belief space. In Chien, S., Kambhampati, S., and Knoblock, C. A. (Eds.), *International Conference on Artificial Intelligence Planning and Scheduling*, pp. 52–61, Menlo Park, California. AAAI Press.

Boole, G. (1847). *The Mathematical Analysis of Logic: Being an Essay towards a Calculus of Deductiva Reasoning.* Macmillan, Barclay, and Macmillan, Cambridge.

Created by XMLmind XSL-FO Converter.

Boolos, G. S. and Jeffrey, R. C. (1989). *Computability and Logic* (3rd edition). Cambridge University Press, Cambridge, UK.

Booth, T. L. (1969). Probabilistic representation of formal languages. In *IEEE Conference Record of the 1969 Tenth Annual Symposium on Switching and Automata Theory*, pp. 74–81, Waterloo, Ontario. IEEE.

Borel, E. (1921). La théorie du jeu et les équations intégrales à noyau symétrique. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences, 173,* 1304–1308.

Borenstein, J., Everett, B., and Feng, L. (1996). *Navigating Mobile Robots: Systems and Techniques.* A. K. Peters, Ltd., Wellesley, MA.

Borenstein, J. and Koren., Y. (1991). The vector field histogram–fast obstacle avoidance for moile robots. *IEEE Transactions on Robotics and Automation*, *7*(3), 278–288.

Borgida, A., Brachman, R. J., McGuinness, D. L., and Alperin Resnick, L. (1989). CLASSIC: A structural data model for objects. *SIGMOD Record, 18*(2), 58–67.

Boser, B. E. Guyon, I. M. and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory (COLT-92),* Pittsburgh, Pennsylvania. ACM Press.

Boutilier, C. and Brafman, R. I. (2001). Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research, 14,* 105–136.

Boutilier, C., Dearden, R., and Goldszmidt, M. (2000). Stochastic dynamic programming with factored representations. *Artificial Intelligence, 121,* 49–107.

Boutilier, C. Reiter, R., and Price, B. (2001). Symbolic dynamic programming for first-order MDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01),* pp. 467–472, Seattle, Morgan Kaufmann.

Boutilier, C., Reiter, R., Soutchanski, M., and Thrun, S. (2000). Decision-theoretic, high-level agent programming in the situation calculus. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00),* pp. 355–362, Austin, Texas. AAAI Press.

Box, G. E. P. (1957). Evolutionary operation: A method of increasing industrial productivity. *Applied Statistics, 6,* 81–101.

Boyan, J. A. (2002). Technical update: Least-squares temporal difference learning. *Machine Learning, 49*(2–3), 233–246.

Boyan, J. A. and Moore, A. W. (1998). Learning evaluation functions for global optimization and Boolean satisfiability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98),* Madison, Wisconsin. AAAI Press.

Boyen, X., Friedman, N., and Koller, D. (1999). Discovering the hidden structure of complex dynamic systems. In *Uncertainty in Artificial Intelligence: Proceedings of the Fifteenth Conference,* Stockholm. Morgan Kaufmann.

Boyer, R. S. and Moore, J. S. (1979). *A Computational Logic.* Academic Press, New York.

Boyer, R. S. and Moore, J. S. (1984). Proof checking the RSA public key encryption algorithm. *American Mathematical Monthly, 91*(3), 181–189.

Brachman, R. J. (1979). On the epistemological status of semantic networks. In Findler, N. V. (Ed.), *Associative Networks: Representation and Use of Knowledge by Computers,* pp. 3–50. Academic Press, New York.

Brachman, R. J., Fikes, R. E., and Levesque, H. J. (1983). Krypton: A functional approach to knowledge representation. *Computer, 16*(10), 67–73.

Created by XMLmind XSL-FO Converter.

Brachman, R. J. and Levesque, H. J. (Eds.). (1985). *Readings in Knowledge Representation.* Morgan Kaufmann, San Mateo, California.

Bradtke, S. J. and Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, *22*, 33–57.

Brafman, R. I. and Tennenholtz, M. (2000). A near optimal polynomial time algorithm for learning in certain classes of stochastic games. *Artificial Intelligence, 121,* 31–47.

Braitenberg, V. (1984). *Vehicles: Experiments in Synthetic Psychology.* MIT Press.

Bransford, J. and Johnson, M. (1973). Consideration of some problems in comprehension. In Chase, W. G. (Ed.), *Visual Information Processing.* Academic Press, New York.

Bratko, I. (1986). *Prolog Programming for Artificial Intelligence* (1st edition). Addison–Wesley, Reading, Massachusetts.

Bratko, I. (2001). *Prolog Programming for Artificial Intelligence* (Third edition). Addison-Wesley, Reading, Massachusetts.

Bratman, M. E. (1987). *Intention, Plans, and Practical Reason.* Harvard University Press, Cambridge, Massachusetts.

Bratman, M. E. (1992). Planning and the stability of intention. *Minds and Machines, 2*(1), 1–16.

Breese, J. S. and Heckerman, D. (1996). Decision-theoretic troubleshooting: A framework for repair and experiment. In *Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference*, pp. 124–132, Portland, Oregon, Morgan Kaufmann.

Breiman, L. (1996). Bagging predictors. *Machine Learning, 26*(2), 123–140.

Breiman, L., Friedman, J., Olshen, R. A., and Stone, P. J. (1984). *Classification and Regression Trees.* Wadsworth International Group, Belmont, California.

Brelaz, D. (1979). New methods to color the vertices of a graph. *Communications of the Association for Computing Machinery, 22*(4), 251–256.

Brent, R. P. (1973). *Algorithms for minimization without derivatives.* Prentice-Hall, Upper Saddle River, New Jersey.

Bresnan, J. (1982). *The Mental Representation of Grammatical Relations.* MIT Press, Cambridge, Massachusetts.

Brewka, G., Dix, J., and Konolige, K. (1997). *Nononotonic Reasoning: An Overview.* CSLI Publications, Stanford, California.

Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In Fogelman Soulié, F. and Hérault, J. (Eds.), *Neurocomputing: Algorithms, Architectures and Applications.* Springer-Verlag, Berlin.

Briggs, R. (1985). Knowledge representation in Sanskrit and artificial intelligence. *Al Magazine, 6*(1), 32–39.

Brin, S. and Page, L. (1998). The anatomy of a largescale hypertextual web search engine. In *Proceedings of the Seventh World Wide Web Conference*, Brisbane, Australia.

Broadbent, D. E. (1958). *Perception and communication.* Pergamon, Oxford, UK.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation, 2,* 14–23.

Brooks, R. A. (1989). Engineering approach to building complete, intelligent beings. *Proceedings of the SPIE – the International Society for Optical Engineering, 1002,* 618–625.

Created by XMLmind XSL-FO Converter.

Brooks, R. A. (1990). Elephants don't play chess. *Autonomous Robots, 6,* 3–15.

Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence, 47*(1–3), 139–159.

Brooks, R. A. and Lozano-Perez, T. (1985). A subdivision algorithm in configuration space for findpath with rotation. *IEEE Transactions on Systems, Man and Cybernetics, 15*(2), 224–233.

Brown, M. Grundy, W., Lin, D., Cristianini, N., Sugnet, C., Furey, T., Ares, M., and Haussler, D. (2000). Knowledge-based analysis of microarray gene expression data using support vector machines. In *Proceedings of the national Academy of Sciences,* Vol. 97, pp. 262–267.

Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Mercer, R. L., and Rossin, P. (1988). A statistical approach to language translation. In *Proceedings of the 12th International Conference on Computational Linguistics,* pp. 71–76, Budapest, John von Neumann Society for Computing Sciences.

Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics, 19*(2), 263–311.

Brownston, L., Farrell, R., Kant, E., and Martin, N. (1985). *Programming expert systems in OPS5: An introduction to rule-based programming.* Addison-Wesley, Reading, Massachusetts.

Brudno, A. L. (1963). Bounds and valuations for shortening the scanning of variations. *Problems of Cybernetics, 10,* 225–241.

Bruner, J. S., Goodnow, J. J., and Austin, G. A. (1957). *A Study of Thinking.* Wiley, New York.

Bryant, R. E. (1992). Symbolic Boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys, 24*(3), 293–318.

Bryson, A. E. and Ho, Y.-C. (1969). *Applied Optimal Control.* Blaisdell, New York.

Buchanan, B. G. and Mitchell, T. M. (1978). Model-directed learning of production rules. In Waterman, D. A. and Hays-Roth, F. (Eds.), *Pattern-Directed Inference Systems,* pp. 297–312. Academic Press, New York.

Buchanan, B. G., Mitchell, T. M., Smith, R. G., and Johnson, C. R. (1978). Models of learning systems. In *Encyclopedia of Computer Science and Technology,* Vol. 11. Dekker, New York.

Buchanan, B. G., and Shortliffe, E. H. (Eds.). (1984). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley, Reading, Massachusetts.

Buchanan, B. G., Sutherland, G. L., and Feigenbaum, E. A. (1969). Heuristic DENDRAL: A program for generating explanatory hypotheses in organic chemistry. In Meltzer, B., Michie, D., and Swann, M. (Eds.), *Machine Intelligence 4,* pp. 209–254. Edinburgh University Press, Edinburgh, Scotland.

Bundy, A. (1999). A survey of automated deduction. In Wooldridge, M. J. and Veloso, M. (Eds.), *Artificial intelligence today: Recent trends and developments,* pp. 153–174. Springer-Verlag, Berlin.

Bunt, H. C. (1985). The formal representation of (quasi-) continuous concepts. In Hobbs, J. R. and Moore, R. C. (Eds.), *Formal Theories of the Commonsense World,* chap. 2, pp. 37–70. Ablex, Norwood, New Jersey.

Burgard, W., Cremers, A. B., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., and Thrun, S. (1999). Experiences with an interactive museum tourguide robot. *Artificial Intelligence, 114*(1–2), 3–55.

Buro, M. (2002). Improving heuristic mini-max search by supervised learning. *Artificial Intelligence, 134*(1–2), 85–99.

Burstall, R. M. (1974). Program proving as hand simulation with a little induction. In *Information Processing '74,* pp. 308–312. Elsevier/North-Holland, Amsterdam, London, New York.

Burstall, R. M. and Darlington, J. (1977). A transformation system for developing recursive programs. *Journal of the Association for Computing Machinery, 24*(1), 44–67.

Created by XMLmind XSL-FO Converter.

Burstein, J., Leacock, C., and Swartz, R. (2001). Automated evaluation of essays and short answers. In *Fifth International Computer Assisted Assessment (CAA) Conference*, Loughborough, U.K. Loughborough University.

Bylander, T. (1992). Complexity results for serial decomposability. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pp. 729–734, San Jose. AAAI Press.

Bylander, T. (1994). The computational complexity of propositional strips planning. *Artificial Intelligence, 69*, 165–204.

Calvanese, D., Lenzerini, M., and Nardi, D. (1999). Unifying class-based representation formalism. *Journal of Artificial Intelligence Research, 11*, 199–240.

Campbell, M. S., Hoane, A. J., and Hsu, F.-H. (2002). Deep Blue. *Artificial Intelligence, 134*(1–2), 57–83.

Canny, J. and Reif, J. (1987). New lower bound techniques for robot motion planning problems. In *IEEE Symposium on Foundations of Computer Science*, pp. 39–48.

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 8*, 679–698.

Canny, J. (1988). *The Complexity of Robot Motion Planning.* MIT Press. Cambridge, Massachusetts.

Carbonell, J. G. (1983). Derivational analogy and its role in problem solving. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-83)*, pp. 64–69, Washington, DC. Morgan Kaufmann.

Carbonell, J. G. Knoblock, C. A., and Minton, S. (1989). PRODIGY: An integrated architecture for planning and learning. Technical report CMU-CS-89-189, Computer Science Department, Carnegie-Mellon University, Pittsburgh.

Carbonell, J. R. and Collins, A. M. (1973). Natural semantics in artificial intelligence. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI-73)*, pp. 344–351, Stanford, California. IJCAII.

Carnap, R. (1928). *Der logische Aufbau der Welt.* Weltkreis-verlag, Berlin-Schlachtensee. Translated into English as (Carnap, 1967).

Carnap, R. (1948). On the application of inductive logic. *Philosophy and Phenomenological Research, 8*, 133–148.

Carnap, R. (1950). *Logical Foundations of Probability.* University of Chicago Press, Chicago.

Carrasco, R. C., Oncina, J., and Calera, J. (1998). *Stochastic Inference of Regular Tree Languages*, Vol. 1433 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin.

Cassandra, A. R. Kaelbling, L., P., and Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pp. 1023–1028. Seattle. AAAI Press.

Ceri, S., Gottlob, G., and Tanca, L. (1990). *Logic programming and databases.* Springer-Verlag, Berlin.

Chakrabarti, P. P., Ghose, S., Acharya, A., and de Sarkar, S. C. (1989). Heuristic search in restricted memory. *Artificial Intelligence, 41*(2), 197–222.

Chan, W. P., Prete, F., and Dickinson, M. H. (1998). Visual input to the efferent control system of a fly's 'gyroscope'. *Science, 289*, 289–292.

Chandra, A. K. and Harel, D. (1980). Computable queries for relational data bases. *Journal of Computer and System Sciences, 21*(2), 156–178.

Chandra, A. K. and Merlin, P. M. (1977). Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, pp. 77–90, New York. ACM Press.

Created by XMLmind XSL-FO Converter.

Chang, C.-L. and Lee, R. C.-T. (1973). *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York.

Chapman, D. (1987). Planning for conjunctive goals. *Artificial Intelligence, 32*(3), 333–377.

Charniak, E. (1993). *Statistical Language Learning*. MIT Press, Cambridge, Massachusetts.

Charniak, E. (1996). Tree-bank grammars. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pp. 1031–1036, Portland, Oregon. AAAI Press.

Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pp. 598–603, Providence, Rhode Island. AAAI Press.

Charniak, E. and Goldman, R. (1992). A Bayesian model of plan recognition. *Artificial Intelligence, 64*(1) 53–79.

Charniak, E. and McDermott, D. (1985). *Introduction to Artificial Intelligence*. Addison-Wesley, Reading, Massachusetts.

Charniak, E., Riesbeck, C., McDermott, D., and Meehan, J. (1987). *Artificial Intelligence Programming* (2nd edition). Lawrence Erlbaum Associates, Potomac, Maryland.

Chatfield, C. (1989). *The Analysis of Time Series: An Introduction* (4th edition). Chapman and Hall, London.

Cheeseman, P. (1985). In defense of probability. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, pp. 1002–1009, Los Angeles, Morgan Kaufmann.

Cheeseman, P. (1988). An inquiry into computer understanding. *Computational Intelligence, 4*(1), 58–66.

Cheeseman, P., Kanefsky, B., and Taylor, W. (1991). Where the really hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pp. 331–337, Sydney. Morgan Kaufmann.

Cheeseman, P., Self, M., Kelly, J., and Stutz, J. (1988). Bayesian classification. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, Vol. 2, pp. 607–611, St. Paul, Minnesota, Morgan Kaufmann.

Cheeseman, P. and Stutz, J. (1996). Bayesian classification (AutoClass): Theory and results. In Fayyad, U., Piatesky-Shapiro, G., Smyth, P., and Uthurusamy, R. (Eds.), *Advances in Knowledge Discovery and Data Mining*. AAAI Press/MIT Press, Menlo Park, California.

Cheng, J. and Druzdzel, M. J. (2000). AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. *Journal of Artificial Intelligence Research, 13*, 155–188.

Cheng, J. Greiner, R., Kelly, J., Bell, D. A., and Liu, W. (2002). Learning Bayesian networks from data: An information-theory based approach. *Artificial Intelligence, 137*, 43–90.

Chierchia, G. and McConnell-Ginet, S. (1990). *Meaning and Grammar*. MIT Press, Cambridge, Massachusetts.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory, 2*(3), 113–124.

Chomsky, N. (1957). *Syntactic Structures*. Mouton, The Hague and Paris.

Chomsky, N. (1980). Rules and representations. *The Behavioral and Brain Sciences, 3*, 1–61.

Choset, H. (1996). *Sensor Based Motion Planning: The Hierarchical Generalized Voronoi Graph*. Ph.D. thesis, California Institute of Technology.

Chung, K. L. (1979). *Elementary Probability Theory with Stochastic Processes* (3rd edition). Springer-Verlag, Berlin.

Created by XMLmind XSL-FO Converter.

Church, A. (1936). A note on the Entscheidungsproblem. *Journal of Symbolic Logic, 1,* 40–41 and 101–102.

Church, K. and Patil, R. (1982). Coping with syntactic ambiguity or how to put the block in the box on the table. *American Journal of Computational Linguistics, 8*(3–4), 139–149.

Church, K. and Gale, W. A. (1991). A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language, 5,* 19–54.

Churchland, P. M. (1979). *Scientific Realism and the Plasticity of Mind.* Cambridge University Press, Cambridge, UK.

Churchland, P. M. and Churchland, P. S. (1982). Functionalism, qualia, and intentionality. In Biro, J. I. and Shahan, R. W. (Eds.), *Mind, Brain and Function: Essays in the Philosophy of Mind,* pp. 121–145. University of Oklahoma Press, Norman, Oklahoma.

Churchland, P. S. (1986). *Neurophilosophy: Toward a Unifed Science of the Mind–Brain.* MIT Press, Cambridge, Massachusetts.

Cimatti, A., Roveri, M., and Traverso, P. (1998). Automatic OBDD-based generation of universal plans in non-determistic domains. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98),* pp. 875–881, Madison, Wisconsin. AAAI Press.

Clark, K. L. (1978). Negation as failure. In Gallaire, H. and Minker, J. (Eds.), *Logic and Data Bases,* pp. 293–322. Plenum, New York.a

Clark, P. and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning, 3,* 261–283.

Clarke, A. C. (1968a). *2001: A Space Odyssey.* Signet.

Clarke, A. C. (1968b). The world of 2001. Vogue.

Clarke, E. and Grumberg, O. (1987). Research on automtic verification of finite-state concurrent systems. *Annual Review of Computer Science, 2,* 269–290.

Clarke, E. Grumberg, O., and Peled, D. (1999). *Model Checking.* MIT Press, Cambridge, Massachusetts.

Clarke, M. R. B. (Ed.). (1977). *Advances in Computer Chess 1.* Edinburgh University Press, Scotland.

Clearwater, S. H. (Ed.). (1996). *Market-Based Control.* World Scientific, Singapore and Teneck, New Jersey.

Clocksin, W. F. and Mellish, C. S. (1994). *Programming in Prolog* (4th edition). Springer-Verlag, Berlin.

Clowes, M. B. (1971). On seeing things. *Artificial Intelligence, 2*(1), 79–116.

Cobham, A. (1964). The intrinsic computational difficulty of functions. In Bar-Hillel, Y. (Ed.), *Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science,* pp. 24–30, Jerusalem. Elsevier/North-Holland.

Cobley, P. (1997). *Introducting Semiotics.* Totem Books, New York.

Cohen, J. (1988). A view of the origins and development of PROLOG. *Communications of the Association for Computing Machinery, 31,* 26–36.

Cohen, P. R. (1995). *Empirical methods for artificial intelligence.* MIT Press, Cambridge, Massachusetts.

Cohen, P. R. and Levesque, H. J. (1990). Intention is choice with commitment. *Artificial Intelligence, 42*(2–3), 213–261.

Cohen, P. R., Morgan, J., and Pollack, M. E. (1990). *Intentions in Communication.* MIT Press, Cambridge, Massachusetts.

Cohen, P. R. and Perrault, C. R. (1979). Elements of a plan-based theory of speech acts. *Cognitive Science, 3*(3), 177–212.

Created by XMLmind XSL-FO Converter.

Cohen, W. W. and Page, C. D. (1995). Learnability in inductive logic programming: Methods and results. *New Generation Computing, 13*(3–4), 369–409.

Cohn, A. G. Bennett, B., Gooday, J. M., and Gotts, N. (1997). RCC: A calculus for region based qualitative spatial reasoning. *GeoInformatica, 1*, 275–316.

Collins, M. J. (1996). A new statistical parser based on bigram lexical dependencies. In Joshi, A. K. and Palmer, M. (Eds.), *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pp. 184–191, San Francisco, Morgan Kaufmann Publishers.

Collins, M. J. (1999). *Head-driven Statistical Models for Natural Language Processing.* Ph.D. thesis, University of Pennsylvania.

Collins, M. and Duffy, K. (2002). New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the ACL.*

Colmerauer, A. (1975). Les grammaires de metamorphose. Tech. rep., Groupe d'Intelligence Artificielle, Université de Marsaille-Luminy.

Colmerauer, A., Kanoui, H., Pasero, R., and Roussel, P. (1973). Un systéme de communication homme–machine en Français. Rapport, Groupe d'Intelligence Artificielle, Université d'Aix-Marseille II.

Condon, J. H. and Thompson, K. (1982). Belle chess harware. In Clarke, M. R. B. (Ed.), *Advances in Computer Chess 3*, pp. 45–54. Pergamon, Oxford, UK.

Congdon, C. B., Huber, M., Kortenkamp, D., Bidlack, C., Cohen, C., Huffman, S., Koss, F., Raschke, U., and Weymouth, T. (1992). CARMEL versus Flakey: A comparison of two robots. Tech. rep. Papers from the AAAI Robot Competition, RC-92-01, American Association for Artificial Intelligence, Menlo Park, CA.

Connell, J. (1989). *A Colony Architecture for an Artificial Creature.* Ph.D. thesis, Artificial Intelligence Laboratory, MIT, Cambridge, MA. also available as AI Technical Report 1151.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pp. 151–158, New York. ACM Press.

Cook, S. A. and Mitchell, D. (1997). Finding hard instances of the satisfiability problem: A survey. In Du, D., Gu, J., and Pardalos, P. (Eds.), *Satisfiability problems: Theory and applications.* American Mathematical Society, Providence, Rhode Island.

Cooper, G. (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence, 42*, 393–405.

Cooper, G. and Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning, 9*, 309–347.

Copeland, J. (1993). *Artificial Intelligence: A Philosophical Introduction.* Blackwell, Oxford, UK.

Copernicus (1543). *De Revolutionibus Orbium Coelestium.* Apud Ioh. Petreium, Nuremberg.

Cormen, t. H., Leiserson, C. E., and Rivest, R. (1990). *Introduction to Algorithms.* MIT Press, Cambridge, Massachusetts.

Cortes, C. and Vapnik, V. N. (1995). Support vector networks. *Machine Learning, 20*, 273–297.

Cournot, A. (Ed.) (1838). *Recherches sur les principes mathématiques de la théorie des richesses.* L. Hachette, Paris.

Covington, M. A., (1994). *Natural Language Processing for Prolog Programmers.* Prentice-Hall, Upper Saddle River, New Jersey.

Cowan, J. D. and Sharp, D. H. (1988a). Neural nets. *Quarterly Reviews of Biophysics, 21*, 365–427.

Cowan, J. D. and Sharp, D. H. (1988b). Neural nets and artificial intelligence. *Daedalus, 117*, 85–121.

Created by XMLmind XSL-FO Converter.

Cox, I. (1993). A review of statistical data association techniques for motion correspondence. *International Journal of Computer Vision, 10*, 53–66.

Cox, I. and Hingorani, S. L. (1994). An efficient implementation and evaluation of Reid's multiple hypothesis tracking algorithm for visual tracking. In *Proceedings of the 12th International Conference on Pattern Recognition*, Vol. 1, pp. 437–442, Jerusalem, Israel. International Association for Pattern Recognition (IAPR).

Cox, I. and Wilfong, G. T. (Eds.). (1990). *Autonomous Robot Vehicles.* Springer-Verlag, Berlin.

Cox, R. T. (1946). Probability, frequency, and reasonable expectation. *American Journal of Physics, 14*(1), 1–13.

Craig, J. (1989). *Introduction to Robotics: Mechanics and Control (2nd Edition).* Addison-Wesley Publishing, Inc., Reading, MA.

Craik, K. J. (1943). *The Nature of Explanation.* Cambridge University Press, Cambridge, UK.

Crawford, J. M. and Auton, L. D. (1993). Experimental results on the crossover point in satisfiability problems. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pp. 21–27, Washington, DC. AAAI Press.

Cristianini, N. and Schölkopf, B. (2002). Support vector machines and kernel methods: The new generation of learning machines. *AI Magazine, 23*(3), 31–41.

Cristianini, N. and Shawe-Taylor, J. (2000). *An introduction to support vector machines and other kernel-based learning methods.* Cambridge University Press, Cambridge, UK.

Crockett, L. (1994). *The Turing Test and the Frame Problem: AI's Mistaken Understanding of Intelligence.* Ablex, Norwood, New Jersey.

Cross, S. E. and Walker, E. (1994). Dart: Applying knowledge based planning and scheduling to crisis action planning. In Zweben, M. and Fox, M. S. (Eds.), *Intelligent Scheduling*, pp. 711–729. Morgan Kaufmann, San Mateo, California.

Cruse, D. A. (1986). *Lexical Semantics.* Cambridge University Press.

Culberson, J. and Schaeffer, J. (1998). Pattern databases. *Computational Intelligence, 14*(4), 318–334.

Cullingford, R. E. (1981). Integrating knowledge sources for computer „understanding" tasks. *IEEE Transactions on Systems, Man and Cybernetics (SMC), 11.*

Cussens, J. and Dzeroski, S. (2000). *Learning Language in Logic*, Vol. 1925 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin.

Cybenko, G. (1988). Continuous valued neural networks with two hidden layers are sufficient. Technical report, Department of Computer Science, Tufts University, Medford, Massachusetts.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Controls, Signals, and Systems, 2*, 303–314.

Daganzo, C. (1979). *Multinomial probit: The theory and its application to demand forecasting.* Academic Press, New York.

Dagum, P. and Luby, M. (1993). Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence, 60*(1), 141–153.

Dahl, O.-J., Myrhaug, B., and Nygaard, K. (1970). (Simula 67) common base language. Tech. rep. N. S-22, Norsk Regnesentral (Norwegian Computing Center), Oslo.

Dale, R., Moisl, H., and Somers, H. (2000). *Handbook of Natural Language Processing*. Marcel Dekker, New York.

Created by XMLmind XSL-FO Converter.

Dantzig, G. B. (1949). Programming of interdependent activities: II. mathematical model. *Econometrica, 17,* 200–211.

Darwiche, A. (2001). Recursive conditioning. *Artificial Intelligence, 126,* 5–41.

Darwiche, A. and Ginsberg, M. L. (1992). A symbolic generalization of probability theory. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92),* pp. 622–627, San Jose. AAAI Press.

Darwin, C. (1859). *On The Origin of Species by Means of Natural Selection.* J. Murray, London.

Darwin, C. (1871). *Descent of Man.* J. Murray.

Dasgupta, P., Chakrabarti, P. P., and DeSarkar, S. C. (1994). Agent searching in a tree and the optimality of iterative deepening. *Artificial Intelligence, 71,* 195–208.

Davidson, D. (1980). *Essays on Actions and Events.* Oxford University Press, Oxford, UK.

Davies, T. R. (1985). Analogy. Informal note INCSLI-85-4, Center for the Study of Language and Information (CSLI), Stanford, California.

Davies, T. R. and Russell, S. J. (1987). A logical approach to reasoning by analogy. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87),* Vol. 1, pp. 264–270. Milan. Morgan Kaufmann.

Davis, E. (1986). *Representing and Acquiring Geographic Knowledge.* Pitman and Morgan Kaufmann, London and San Mateo, California.

Davis, E. (1990). *Representations of Commonsense Knowledge.* Morgan Kaufmann, San Mateo, California.

Davis, K. H., Biddulph, R., and Balashek, S. (1952). Automatic recognition of spoken digits. *Journal of the Acoustical Society of America, 24*(6), 637–642.

Davis, M. (1957). A computer program for Presburger's algorithm. In Robinson, A. (Ed.), *Proving Theorems (as Done by Man, Logician, or Machine),* pp. 215–233, Cornell University, Ithaca, New York. Communications Research Division, Institute for Defense Analysis. Proceedings of the Summer Institute for Symbolic Logic. Second edition; publication date is 1960.

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Communications of the Association for Computing Machinery, 5,* 394–397.

Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the Association for Computing Machinery, 7*(3), 201– 215.

Davis, R. and Lenat, D. B. (1982). *Knowledge-Based Systems in Artificial Intelligence.* McGraw-Hill, New York.

Dayan, P. (1992). The convergence of TD(λ) for general λ. *Machine Learning, 8*(3–4), 341–362.

Dayan, P. and Abbott, L. F. (2001). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems.* MIT Press, Cambridge, Massachusetts.

de Dombal, F. T., Leaper, D. J., Horrocks, J. C., and Staniland, J. R. (1974). Human and computer-aided diagnosis of abdominal pain: Further report with emphasis on performance of clinicians. *British Medical Journal, 1,* 376–380.

de Dombal, F. T., Staniland, J. R., and Clamp, S. E. (1981). Geographical variation in disease presentation. *Medical Decision Making, 1,* 59–69.

de Finetti, B. (1937). Le prévision: ses lois logiques, ses sources subjectives. *Ann. Inst. Poincaré, 7,* 1–68.

de Freitas, J. F. G., Niranjan, M., and Gee, A. H. (200). Sequential Monte Carlo methods to train neural network models. *Neural Computation, 12*(4), 933–953.

Created by XMLmind XSL-FO Converter.

de Kleer, J. (1975). Qualitative and quantitative knowledge in classical mechanics. Tech. rep. AI-TR-352, MIT Artificial Intelligence Laboratory.

de Kleer, J. (1986a). An assumption-based TMS. *Artificial Intelligence, 28*(2), 127–162.

de Kleer, J. (1986b). Extending the ATMS. *Artificial Intelligence, 28*(2), 163–196.

de Kleer, J. (1986c). Problem solving with the ATMS. *Artificial Intelligence, 28*(2), 197–224.

de Kleer, J. (1989). A comparison of ATMS and CSP techniques. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89),* Vol. 1, pp. 290–296, Detroit. Morgan Kaufmann.

de Kleer, J. and Brown, J. S. (1985). A qualitative physics based on confluences. In Hobbs, J. R. and Moore, R. C. (Eds.), *Formal Theories of the Commonsense World,* chap. 4, pp. 109–183. Ablex, Norwood, New Jersey.

de Marcken, C. (1996). *Unsupervised Language Acquisition.* Ph.D. thesis, MIT.

De Morgan, A. (1864). On the syllogism IV and on the logic of relations. *Cambridge Philosophical Transactions, x,* 331–358.

De Readt, L. (1992). *Interactive Theory Revision: An Inductive Logic Programming Approach.* Academic Press, New York.

de Saussure, F. [1910 (republished 1993)]. *Lectures on General Linguistics.* Pergamon Press, Oxford, UK.

Deacon, T. W. (1997). *The symbolic species: The coevolution of language and the brain.* W. W. Norton, New York.

Deale, M., Yvanovich, M., Schnitzius, D., Kautz, D., Carpenter, M., Zweben, M., Davis, G., and Daun, B. (1994). The space shuttle ground processing scheduling system. In Zweben, M. and Fox, M. (Eds.), *Intelligent Scheduling,* pp. 423–449. Morgan Kaufmann, San Mateo, California.

Dean, T., Basye, K., Chekaluk, R., and Hyun, S. (1990). Coping with uncertainty in a control system for navigation and exploration. In *Preceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90),* Vol. 2, pp. 1010–1015, Boston. MIT Press.

Dean, T. and Boddy, M. (1988). An analysis of time dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88),* pp. 49á–54, St. Paul, Minnesota. Morgan Kaufmann.

Dean, T., Firby, R. J., and Miller, D. (1990). Hierarchical planning involving deadlines, travel time, and resources. *Computational Intelligence, 6*(1), 381–398.

Dean, T., Kaelbling, L. P., Kirman, J., and Nicholson, A. (1993). Planning with deadlines in stochastic domains. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93),* pp. 574–579, Washington, DC. AAAI Press.

Dean, T. and Kanazawa, K. (1989a). A model for projection and action. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89),* pp. 985–990, Detroit. Morgan Kaufmann.

Dean, T. and Kanazawa, K. (1989b). A model for reasoning about persistence and causation. *Computational Intelligence, 5*(3), 142–150.

Dean, T., Kanazawa, K., and Shewchuk, J. (1990). Prediction, observation and estimation in planning and control. In *5th IEEE International Symposium on Intelligent Control,* Vol. 2, pp. 645–650, Los Alamitos, CA. IEEE Computer Society Press.

Dean, T. and Wellman, M. P. (1991). *Planning and Control.* Morgan Kaufmann, San Mateo, California.

Debevec, P., Taylor, C., and Malik, J. (1996). Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *Proceedings. of the 23rd Annual Conference on Computer Graphics (SIGGRAPH),* pp. 11–20.

Created by XMLmind XSL-FO Converter.

Debreu, G. (1960). Topological methods in cardinal utility theory. In Arrow, K. J., Karlin, S., and Suppes, P. (Eds.), *Mathematical Methods in the Social Sciences, 1959.* Stanford University Press, Stanford, California.

Dechter, R. (1990a). Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence, 41,* 273–312.

Dechter, R. (1990b). On the expressiveness of networks with hidden variables. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90),* pp. 379–385, Boston. MIT Press.

Dechter, R. (1992). Constraint networks. In Shapiro, S. (Ed.), *Encyclopedia of Artificial Intelligence* (2nd edition), pp. 276–285. Wiley and Sons, New York.

Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence, 113,* 41–85.

Dechter, R. and Frost, D. (1999). Backtracking algorithms for constraint satisfaction problems. Tech. rep., Department of Information and Computer Science, University of California, Irvine.

Dechter, R. and Pearl, J. (1985). Generalized best-first search strategies and the optimality of A*. *Journal of the Association for Computing Machinery, 32*(3), 505–536.

Dechter, R. and Pearl, J. (1987). Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence, 34*(1), 1–38.

Dechter, R. and Pearl, J. (1989). Tree clustering for constraint networks. *Artificial Intelligence, 38*(3), 353–366.

DeCoste, D. and Scholkopf, B. (2002). Training invariant support vector machines. *Machine Learning, 46*(1), 161–190.

Dedekind, R. (1888). *Was sind und was sollen die Zahlen.* Braunschweig, Germany.

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., and Harshman, r. A. (1990). Indexing by latent semantic analysis. *Journal of the American Society of Information Science, 41*(6), 391–407.

DeGroot, M. H. (1970). *Optimal Statistical Decisions.* McGraw-Hill, New York.

DeGroot, M. H. (1989). *Probability and Statistics* (2nd edition). Addison-Wesley, Reading, Massachusetts.

DeJong, G. (1981). Generalizations based on explanations. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI-81),* pp. 67–69, Vancouver, British Columbia. Morgan Kaufmann.

DeJong, G. (1982). An overview of the FRUMP system. In Lehnert, W. and Ringle, M. (Eds.), *Strategies for Natural Language Processing,* pp. 149–176. Lawrence Erlbaum, Potomac, Maryland.

DeJong, G. and Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning, 1,* 145–176.

Dempster, A. P. (1968). A generalization of Bayesian inference. *Journal of the Royal Statistical Society, 30 (Series B),* 205–247.

Dempster, A. P., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, 39 (Series B),* 1–38.

Denes, P. (1959). The design and operation of the mechanical speech recognizer at University College London. *Journal of the British Institution of Radio Engineers, 19*(4), 219–234.

Deng, X. and Papadimitriou, C. H. (1990). Exploring an unknown graph. In *Proceedings 31st Annual Symposium on Foundations of Computer Science,* pp. 355–361, St. Louis. IEEE Computer Society Press.

Denis, F. (2001). Learning regular languages from simple positive examples. *Machine Learning, 44*(1/2), 37–66.

Dennett, D. C. (1971). Intentional systems. *The Journal of Philosophy, 68*(4), 87–106.

Created by XMLmind XSL-FO Converter.

Dennett, D. C. (1978). Why you can't make a computer that feels pain. *Synthese, 38*(3).

Dennett, D. C. (1984). Cognitive wheels: the frame problem of AI. In Hookway, C. (Ed.), *Minds, Machines, and Evolution: Philosophical Studies*, pp. 129–151. Cambridge University Press, Cambridge, UK.

Deo, N. and Pang, C.-Y. (1984). Shortest path algorithms: Taxonomy and annotation. *Networks, 14*(2), 275–323.

Descartes, R. (1637). Discourse on method. In Cottingham, J., Stoothoff, R., and Murdoch, D. (Eds.), *The Philosophical Writings of Descartes*, Vol. I. Cambridge University Press, Cambridge, UK.

Descotte, Y. and Latombe, J.-C. (1985). Making compromises among antagonist constraints in a planner. *Artificial Intelligence, 27*, 183–217.

Devroye, L. (1987). *A course in density estimation.* Birkhauser, Boston.

Devroye, L., Gyorfi, L., and Lugosi, G. (1996). *A probabilistic theory of pattern recognition.* Springer-Verlag, Berlin.

Dickmanns, E. D. and Zapp, A. (1987). Autonomous high speed road vehicle guidance by computer vision. In Isermann, R. (Ed.), *Automatic Control– World Congress, 1987: Selected Papers from the 10th Triennial World Congress of the International Federation of Automatic Control*, pp. 221–226, Munich. Pergamon.

Dietterich, T. (1990). Machine Learning. *Annual Review of Computer Science, 4*, 255–306.

Dietterich, T. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research, 13*, 227–303.

DiGioia, A. M., Kanade, T., and Wells, P. (1996). Final report of the second international workshop on robotics and computer assisted medical interventions. *Computer Aided Surgery, 2*, 69–101.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik, 1*, 269–271.

Dissanayake, G., Newman, P., Clark, S., Durrant-Whyte, H., and Csorba, M. (2001). A solution to the simultaneous localisation and map building (SLAM) problem. *IEEE Transactions of Robotics and Automation, 17*(3), 229–241.

Do, M. B. and Kambhampati, S. (2001). Sapa: A domain-independent heuristic metric temporal planner. In *Proceedings of the European Conference on Planning*, Toledo, Spain. Springer-Verlag.

Domingos, P. and Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero–one loss. *Machine Learning, 29*, 103–130.

Doran, C., Egedi, D., Hockey, B. A., Srinivas, B., and Zaidel, M. (1994). XTAG system–a wide coverage grammar of English. In Nagao, M. (Ed.), *Proceedings of the 15th COLING*, Kyoto, Japan.

Doran, J. and Michie, D. (1966). Experiments with the graph traverser program. *Proceedings of the Royal Society of London, 294, Series A*, 235–259.

Dorf, R. C. and Bishop, R. H. (1999). *Modern Control Systems.* Addison-Wesley, Reading, Massachusetts.

Doucet, A. (1997). *Monte Carlo methods for Bayesian estimation of hidden Markov models: Application to radiation signals.* Ph.D. thesis, Université de Paris-Sud, Orsay, France.

Dowling, W. F. and Gallier, J. H. (1984). Linear-time algorithms for testing the satisfiability of propositional Horn formulas. *Journal of Logic Programming, 1*, 267–284.

Dowty, D., Wall, R., and Peters, S. (1991). *Introduction to Montague Semantics.* D. Reidel, Dordrecht, Netherlands.

Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence, 12*(3), 231–272.

Created by XMLmind XSL-FO Converter.

Doyle, J. (1983). What is rational psychology? Toward a modern mental philosophy. *AI Magazine, 4*(3), 50–53.

Doyle, J. and Patil, R. (1991). Two theses of knowledge representation: Language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence, 48*(3), 261–297.

Drabble, B. (1990). Mission scheduling for spacecraft: Diaries of t-sched. In *Expert Planning Systems*, pp. 76–81, Brighton, UK. Institute of Electrical Engineers.

Draper, D., Hanks, S., and Weld, D. S. (1994). Probabilistic planning with information gathering and contingent execution. In *Proceedings of the Second International Conference on AI Planning Systems*, pp. 31–36, Chicago. Morgan Kaufmann.

Dreyfus, H. L. (1972). *What Computers Can't Do: A Critique of Artificial Reason.* Harper and Row, New York.

Dreyfus, H. L. (1992). *What Computers Still Can't Do: A Critique of Artificial Reason.* MIT Press, Cambridge, Massachusetts.

Dreyfus, H. L. and Dreyfus, S. E. (1986). *Mind over Machine: The Power of Human Intuition and Expertise in the Era of the Computer.* Blackwell, Oxford, UK.

Dreyfus, S. E. (1969). An appraisal of some shortest-paths algorithms. *Operations Research, 17*, 395–412.

Du, D., Gu, J., and Pardalos, P. M. (Eds.). (1999). *Optimization methods for logical inference.* American Mathematical Society, Providence, Rhode Island.

Dubois, D. and Prade, H. (1994). A survey of belief revision and updating rules in various uncertainty models. *International Journal of Intelligent Systems, 9*(1), 61–100.

Duda, R. O., Gasching, J. and Hart, P. E. (1979). Model design in the Prospector consultant system for mineral exploration. In Michie, D. (Ed.), *Expert Systems in the Microelectronic Age*, pp. 153–167. Edinburgh University Press, Edinburgh, Scotland.

Duda, R. O. and Hart, P. E. (1973). *Pattern classification and scene analysis.* Wiley, New York.

Duda, R. O., Hart, P. E., and Stork, D. g. (2001). *Pattern Classification.* Wiley, New York.

Dudek, G. and Jenkin, M. (2000). *Computational Principles of Mobile Robotics.* Cambridge University Press, Cambridge CB2 2RU, UK.

Durfee, E. H. and Lesser, V. R. (1989). Negotiating task decomposition and allocation using partial global planning. In Huhns, M. and Gasser, L. (Eds.), *Distributed AI*, Vol. 2. Morgan Kaufmann, San Mateo, California.

Dyer, M. (1983). *In-Depth Understanding.* MIT Press, Cambridge, Massachusetts.

Dzeroski, S., Muggleton, S. H., and Russell, S. J. (1992). PAC-learnability of determinate logic programs. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory (COLT-92)*, pp. 128–135, Pittsburgh, Pennsylvania. ACM Press.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery, 13*(2), 94–102.

Ebeling, C. (1987). *All the Right Moves.* MIT Press, Cambridge, Massachusetts.

Eco, U. (1979). *Theory of Semiotics.* Indiana University Press, Bloomington, Indiana.

Edmonds, J. (1965). Paths, trees, and flowers. *Canadian Journal of Mathematics, 17*, 449–467.

Edwards, P. (Ed.). (1967). *The Encyclopedia of Philosophy.* Macmillan, London.

Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F. (1998). The KR system dlv: Progress report, comparisons and benchmarks. In Cohn, A., Schubert, L., and Shapiro, S. (Eds.), *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, pp. 406–417, Trento, Italy.

Created by XMLmind XSL-FO Converter.

Elhadad, M. (1993). FUF: The universal unifier–user manual. Technical report, Ben Gurion University of the Negev, Be'er Sheva, Israel.

Elkan, C. (1993). The paradoxical success of fuzzy logic. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pp. 698–703, Washington, DC. AAAI Press.

Elkan, C. (1997). Boosting and naive Bayesian learning. Tech. rep., Department of Computer Science and Engineering, University of California, San Diego.

Elman, J., Bates, E., Johnson, M., Karmiloff-Smith, A., Parisi, D., and Plunkett, K. (1997). *Rethinking Innateness.* MIT Press, Cambridge, Massachusetts.

Empson, W. (1953). *Seven Types of Ambiguity.* New Directions, New York.

Enderton, H. B. (1972). *A Mathematical Introduction to Logic.* Academic Press, New York.

Erdmann, M. A. and Mason, M. (1988). An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation*, *4*(4), 369–379.

Erman, L. D., Hayes-Roth, F., Lesser, V. R., and Reddy, R. (1980). The hearsay-ii speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, *12*(2), 213–253.

Ernst, H. A. (1961). *MH-1, a Computer-Operated Mechanical Hand.* Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Ernst, M., Millstein, T., and Weld, D. S. (1997). Automatic SAT-compilation of planning problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 1169–1176, Nagoya, Japan. Morgan Kaufmann.

Erol, K., Hendler, J., and Nau, D. S. (1994). HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pp. 1123–1128, Seattle. AAAI Press.

Erol, K., Hendler, J., and Nau, D. S. (1996). Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, *18*(1), 69–93.

Etzioni, O. (1989). Tractable decision-analytic control. In *Proc. of 1st International Conference on Knowledge Representation and Reasoning*, pp. 114–125, Toronto.

Etzioni, O., Hanks, S., Weld, D. S., Draper, D., Lesh, N., and Williamson, M. (1992). An approach to planning with incomplete information. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, Massachusetts.

Etzioni, O. and Weld, D. S. (1994). A softbot-based interface to the Internet. *Communications of the Association for Computing Machinery*, *37*(7), 72–76.

Evans, T. G. (1968). A program for the solution of a class of geometric-analogy intelligence-test questions. In Minsky, M. L. (Ed.), *Semantic Information Processing*, pp. 271–353. MIT Press, Cambridge, Massachusetts.

Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. Y. (1995). *Reasoning about Knowledge.* MIT Press, Cambridge, Massachusetts.

Fahlman, S. E. (1974). A planning system for robot construction tasks. *Artificial Intelligence*, *5*(1), 1–49.

Fahlman, S. E. (1979). *NETL: A System for Representing and Using Real-World Knowledge.* MIT Press, Cambridge, Massachusetts.

Faugeras, O. (1992). What can be seen in three dimensions with an uncalibrated stereo rig?. In Sandini, G. (Ed.), *Preceedings of the European Conference on Computer Vision*, Vol. 588 of *Lecture Notes in Computer Science*, pp. 563–578. Springer-Verlag.

Faugeras, O. (1993). *Three-Dimensional Computer Vision: A Geometric Viewpoint.* MIT Press, Cambridge, Massachusetts.

Created by XMLmind XSL-FO Converter.

Faugeras, O., Luong, Q.-T., and Papadopoulo, T. (2001). *The Geometry of Multiple Images.* MIT Press, Cambridge, Massachusetts.

Fearing, R. S. and Hollerbach, J. M. (1985). Basic solid mechanics for tactile sensing. *International Journal of Robotics Research, 4*(3), 40–54.

Featherstone, R. (1987). *Robot Dynamics Algorithms.* Kluwer Academic Publishers, Boston, MA.

Feigenbaum, E. A. (1961). The simulation of verbal learning behavior. *Proceedings of the Western Joint Computer Conference, 19,* 121–131.

Feigenbaum, E. A., Buchanan, B. G., and Lederberg, J. (1971). On generality and problem solving: A case study using the DENDRAL program. In Meltzer, B. and Michie, D. (Eds.), *Machine Intelligence 6,* pp. 165–190. Edinburgh University Press, Edinburgh, Scotland.

Feigenbaum, E. A. and Feldman, J. (Eds.). (1963). *Computers and Thought.* McGraw-Hill, New York.

Feldman, J. and Sproull, R. F. (1977). Decision theory and artificial intelligence II: The hungry monkey. Technical report, Computer Science Department, University of Rochester.

Feldman, J. and Yakimovsky, Y. (1974). Decision theory and artificial intelligence I: Semantics-based region analyzer. *Artificial Intelligence, 5*(4) 349–371.

Fellbaum, C. (2001). *Wordnet: An Electronic Lexical Database.* MIT Press, Cambridge, Massachusetts.

Feller, W. (1971). *An Introduction to Probability Theory and its Applications,* Vol. 2. John Wiley.

Ferraris, P. and Giunchiglia, E. (2000). Planning as satisability in nondeterministic domains. In *Proceedings of Seventeenth National Conference on Artificial Intelligence,* pp. 748–753. AAAI Press.

Fikes, R. E., Hart, P. E., and Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence, 3*(4), 251–288.

Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence, 2*(3–4), 189–208.

Fikes, R. E. and Nilsson, N. J. (1993). STRIPS, a retrospective. *Artificial Intelligence, 59*(1–2), 227–232.

Findlay, J. N. (1941). Time: A treatment of some puzzles. *Australasian Journal of Psychology and Philosophy, 19*(3), 216–235.

Finney, D. J. (1947). *Probit analysis: A statistical treatment of the sigmoid response curve.* Cambridge University Press, Cambridge, UK.

Firby, J. (1994). Task networks for controlling continuous processes. In Hammond, K. (Ed.), *Proceedings of the Second International Conference on AI Planning Systems,* pp. 49–54, Menlo Park, CA. AAAI Press.

Firby, R. J. (1996). Modularity issues is reactive planning. In *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96),* pp. 78–85, Edinburgh, Scotland. AAAI Press.

Fischer, M. J. and Ladner, R. E. (1977). Propositional modal logic of programs. In *Proceedings of the 9th ACM Symposium on the Theory of Computing,* pp. 286–294, New York. ACM Press.

Fisher, R. A. (1922). On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London, Series A 222,* 309–368.

Fix, E. and Hodges, J. L. (1951). Discriminatory analysis–nonparametric discrimination: Consistency properties. Tech. rep. 21-49-004, USAF School of Aviation Medicine, Randolph Field, Texas.

Fogel, D. B. (2000). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press, Piscataway, New Jersey.

Created by XMLmind XSL-FO Converter.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial Intelligence through Simulated Evolution.* Wiley, New York.

Forbes, J. (2002). *Learning Optimal Control for Autonomous Vehicles.* Ph.D. thesis, University of California, Berkeley.

Forbus, K. D. (1985). The role of qualitative dynamics in naive physics. In Hobbs, J. R. and Moore, R. C. (Eds.), *Formal Theories of the Commonsense World,* chap. 5, pp. 185–226. Ablex Norwood, New Jersey.

Forbus, K. D. and de Kleer, J. (1993). *Building Problem Solvers.* MIT Press, Cambridge, Massachusetts.

Ford, K. M. and Hayes, P. J. (1995). Turing Test considered harmful. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95),* pp. 972–977, Montreal. Morgan Kaufmann.

Forestier, J.-P. and Varaiya, P. (1978). Multilayer control of large Markov chains. *IEEE Transactions on Automatic Control, 23*(2), 298–304.

Forgy, C. (1981). OPS5 user's manual. Technical report CMU-CS-81-135, Computer Science Department, Carnegie-Mellon University, Pittsburgh.

Forgy, C. (1982). A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence, 19*(1), 17–37.

Forsyth, D. and Zisserman, A. (1991). Reflections on shading. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 13*(7), 671–679.

Fortescue, M. D. (1984). *West Greenlandic.* Croom Helm, London.

Foster, D. W. (1989). *Elegy by W. W.: A Study in Attribution.* Associated University Presses, Cranbury, New Jersey.

Fourier, J. (1827). Analyse des travaux de l'Académie Royale des Sciences, pendant l'année 1824; partie mathématique. *Histoire de l'Académie Royale des Sciences de France, 7,* xlvii–lv.

Fox, D., Burgard, W., Dellaert, F., and Thrun, S. (1999). Monte Carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the National Conference on Artificial Intelligence (AAAI),* Orlando, FL. AAAI.

Fox, M. S. (1990). Constraint-guided scheduling: A short history of research at CMU. *Computers in Industry, 14*(1–3), 79–88.

Fox, M. S., Allen, B., and Strohm, G. (1982). Job shop scheduling: An investigation in constraint-directed reasoning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-82),* pp. 155–158, Pittsburgh, Pennsylvania. Morgan Kaufmann.

Fox, M. S. and Long, D. (1998). The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research, 9,* 367–421.

Frakes, W. and Baeza-Yates, R. (Eds.). (1992). *Information Retrieval: Data Structures and Algorithms.* Prentice-Hall, Upper Saddle River, New Jersey.

Francis, S. and Kucera, H. (1967). *Computing Analysis of Present-day American English.* Brown University Press, Providence, Rhode Island.

Franco, J. and Paull, M. (1983). Probabilistic analysis of the Davis Putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics, 5,* 77–87.

Frank, R. H. and Cook, P. J. (1996). *The Winner-Take-All Society.* Penguin, New York.

Frege, G. (1879). *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens.* Halle, Berlin. English translation appears in van Heijenoort (1967).

Created by XMLmind XSL-FO Converter.

Freuder, E. C. (1978). Synthesizing constraint expressions. *Communications of the Association for Computing Machinery, 21*(11), 958–966.

Freuder, E. C. (1982). A sufficient condition for backtrack-free search. *Journal of the Association for Computing Machinery, 29*(1), 24–32.

Freuder, E. C. (1985). A sufficient condition for backtrack-bounded search. *Journal of the Association for Computing Machinery, 32*(4), 755–761.

Freuder, E. C. and Mackworth, A. K. (Eds.). (1994). *Constraint-based reasoning.* MIT Press, Cambridge, Massachusetts.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning,* Bari, Italy. Morgan Kaufmann.

Friedberg, R. M. (1958). A learning machine: Part I. *IBM Journal, 2,* 2–13.

Friedberg, R. M., Dunham, B., and North, T. (1959). A learning machine: Part II. *IBM Journal of Research and Development, 3*(3), 282–287.

Friedman, G. J. (1959). Digital simulation of an evolutionary process. *General Systems Yearbook, 4,* 171–184.

Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting. *Annals of Statistics, 28*(2), 337–374.

Friedman, N. (1998). The Bayesian structural EM algorithm. In *Uncertainty in Artificial Intelligence: Proceedings of the Fourteenth Conference,* Madison, Wisconsin. Morgan Kaufmann.

Friedman, N. and Goldszmidt, M. (1996). Learning Bayesian networks with local structure. In *Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference,* pp. 252–262, Portland, Oregon. Morgan Kaufmann.

Fry, D. B. (1959). Theoretical aspects of mechanical speech recognition. *Journal of the British Institution of Radio Engineers, 19*(4), 211–218.

Fuchs, J. J., Gasquet, A., Olalainty, B., and Currie, K. W. (1990). PlanERS-1: An expert planning system for generating spacecraft mission plans. In *First International Conference on Expert Planning Systems,* pp. 70–75, Brighton, UK. Institute of Electrical Engineers.

Fudenberg, D. and Tirole, J. (1991). *Game theory.* MIT Press, Cambridge, Massachusetts.

Fukunaga, A. S., Rabideau, G., Chien, S., and Yan, D. (1997). ASPEN: A framework for automated planning and scheduling of spacecraft control and operations. In *Proceedings of the International Symposium on AI, Robotics and Automation in Space,* pp. 181–187, Tokyo.

Fung, R. and Chang, K. C. (1989). Weighting and integrating evidence for stochastic simulation in Bayesian networks. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89),* pp. 209–220, Windsor, Ontario. Morgan Kaufmann.

Gaifman, H. (1964). Concerning measures in first order calculi. *Israel Journal of Mathematics, 2,* 1–18.

Gallaire, H. and Minker, J. (Eds.). (1978). *Logic and Databases.* Plenum, New York.

Gallier, J. H. (1986. *Logic for Computer Science: Foundations of Automatic Theorem Proving.* Harper and Row, New York.

Gallo, G. and Pallottino, S. (1988). Shortest path algorithms. *Annals of Operations Research, 13,* 3–79.

Gamba, A., Gamberini, L., Palmieri, G., and Sanna, R. (1961). Further Experiments with PAPA. *Nuovo Cimento Supplemento, 20*(2), 221–231.

Garding, J. (1992). Shape from texture for smooth curved surfaces in perspecitve projection. *Journal of Mathematical Imaging and Vision, 2*(4), 327–350.

Created by XMLmind XSL-FO Converter.

Gardner, M. (1968). *Logic Machines, Diagrams and Boolean Algebra*. Dover, New York.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability.* W. H. Freeman, New York.

Gaschnig, J. (1977). A general backtrack algorithm that eliminates most redundant tests. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77),* p. 457, Cambridge, Massachusetts. IJCAII.

Gaschnig, J. (1979.) Performance measurement and analysis of certain search algorithms. Technical report CMU-CS-79-124, Computer Science Department, Carnegie-Mellon University.

Gasser, R., (1995). *Efficiently harnessing computational resources for exhaustive search*. Ph.D. thesis, ETH Zürich, Switzerland.

Gasser, R. (1998). Solving nine men's morris. In Nowakowski, r. (Ed.), *Games of No Chance.* Cambridge University Press, Cambridge, UK.

Gat, E. (1998). Three-layered architectures. In Kortenkamp, D., Bonasso, R. P., and Murphy, R. (Eds.), *AI-based Mobile Robots: Case Studies of Successful Robot Systems*, pp. 195–210. MIT Press.

Gauss, K. F. (1809). *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientium.* Sumtibus F. Perthes et I. H. Besser, Hamburg.

Gauss, K. F. (1829). Beiträge zur theorie der algebraischen gleichungen. Collected in *Werke, Vol. 3*, pages 71–102. K. Gesellschaft Wissenschaft, Göttingen, Germany, 1876.

Gawande, A. (2002). *Complications: A Surgeon's Notes on an Imperfect Science.* Metropolitan Books, New York.

Ge, N., Hale, J. and Charniak, E. (1998). A statistical approach to anaphora resolution. In *Proceedings of the Sixth Workshop on Very Large Corpora*, pp. 161–171, Montreal. COLING-ACL.

Geiger, D., Verma, T., and Pearl, J. (1990). Identifying independence in Bayesian networks. *Networks, 20*(5), 507–534.

Gelb, A. (1974). *Applied Optimal Estimation.* MIT Press, Cambridge, Massachusetts.

Gelernter, H. (1959). Realization of a geometry-theorem proving machine. In *Proceedings of an International Conference on Information Processing*, pp. 273–282, Paris. UNESCO House.

Gelfond, M. and Lifschitz, v. (1988). Compiling circumscriptive theories into logic programs. In Reinfrank, M., de Kleer, J., Ginsberg, M. L., and Sandewall, E. (Eds.), *Non-Monotonic Reasoning: 2nd International Workshop Proceedings*, pp. 74–99, Grassau, Germany. Springer-Verlag.

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. (1995). *Bayesian Data Analysis.* Chapman & Hall, London.

Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 6*(6), 721–741.

Genesereth, M. R. (1984). The use of design descriptions in automated diagnosis. *Artificial Intelligence, 24*(1–3), 411–436.

Genesereth, M. R. and Nilsson, N. J. (1987). *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, San Mateo, California.

Genesereth, M. R. and Nourbakhsh, I. (1993). Time-saving tips for problem solving with incomplete information. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pp. 724–730, Washington, DC. AAAI Press.

Genesereth, M. R. and Smith, D. E. (1981). Metalevel architecture. Memo HPP-81-6, Computer Science Department, Stanford University, Stanford, California.

Created by XMLmind XSL-FO Converter.

Gentner, D. (1983). Structure mapping: A theoretical framework for analogy. *Cognitive Science, 7,* 155–170.

Gentzen, G. (1934). Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift, 39,* 176–210, 405–431.

Georgeff, M. P. and Lansky, A. L. (1987). Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87),* pp. 677–682, Seattle. Morgan Kaufmann.

Gerevini, A. and Schubert, L. K. (1996). Accelerating partial-order planners: Some techniques for effective search control and pruning. *Journal of Artificial Intelligence Research, 5,* 95–137.

Gerevini, A. and Serina, I. (2002). LPG: A planner based on planning graphs with action costs. In *Proceedings of the Sixth International Conference on AI Planning and Scheduling,* pp. 281–290, Menlo Park, California. AAAI Press.

Germann, U., Jahr, M., Knight, K., Marcu, D., and Yamada, K. (2001). Fast decoding and optimal decoding for machine translation. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL),* pp. 228–235, Toulouse, France.

Gershwin, G. (1937). Let's call the whole thing off. song.

Ghahramani, Z. and Jordan, M. I. (1997). Factorial hidden Markov models. *Machine Learning, 29,* 245–274.

Ghallab, M., Howe, A., Knoblock, C. A., and McDermott, D. (1998). PDDL–the planning dormain definition language. Tech. rep. DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, Connecticut.

Ghallab, M. and Laruelle, H. (1994). Representation and control in IxTeT, a temporal planner. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS-94),* pp. 61–67, Chicago. AAAI Press.

Giacomo, G. D., Lespérance, Y., and Levesque, H. J. (2000). ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence, 121,* 109–169.

Gibson, J. J. (1950). *The Perception of the Visual World.* Houghton Mifflin, Boston.

Gibson, J. J. (1979). *The Ecological Approach to Visual Perception.* Houghton Mifflin, Boston.

Gibson, J. J., Olum, P., and Rosenblatt, F. (1955). Parallax and perspective during aircraft landings. *American Journal of Psychology, 68,* 372–385.

Gilks, W. R., Richardson, S., and Spiegelhalter, D. J. (Eds.). (1996). *Markov chain Monte Carlo in practice.* Chapman and Hall, London.

Gilks, W. R., Thomas, A., and Spiegelhalter, D. J. (1994). A language and program for complex Bayesian modelling. *The Statistician, 43,* 169–178.

Gilmore, P. C. (1960). A proof method for quantification theory: Its justification and realization. *IBM Journal of Research and Development, 4,* 28–35.

Ginsberg, M. L. (1989). Universal planning: An (almost) universally bad idea. *AI Magazine, 10*(4), 40–44.

Ginsberg, M. L. (1993). *Essentials of Artificial Intelligence.* Morgan Kaufmann, San Mateo, California.

Ginsberg, M. L. (1999). GIB: Steps toward an expert-level bridge-playing program. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99),* pp. 584–589, Stockholm. Morgan Kaufmann.

Ginsberg, M. L., Frank, M., Halpin, M. P., and Torrance, M. C. (1990). Search lessons learned from crossword puzzles. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90),* Vol. 1, pp. 210–215, Boston. MIT Press.

Gittins, J. C. (1989). *Multi-Armed Bandit Allocation Indices.* Wiley, New York.

Created by XMLmind XSL-FO Converter.

Glanc, A. (1978). On the etymology of the word „robot". *SIGART Newsletter, 67,* 12.

Glover, F. (1989). Tabu search: 1. *ORSA Journal on Computing, 1*(3), 190–206.

Glover, F. and Laguna, M. (Eds.). (1997). *Tabu search.* Kluwer, Dordrecht, Netherlands.

Gödel, K. (1930). *Über die Vollständigkeit des Logikkalküls,* Ph.D. thesis, University of Vienna.

Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik, 38,* 173–198.

Goebel, J., Volk, K., Walker, H., and Gerbault, F. (1989). Automatic classification of spectra from the infrared astronomical satellite (IRAS). *Astronomy and Astrophysics, 222,* L5–L8.

Gold, B. and Morgan, N. (2000). *Speech and Audio Signal Processing.* Wiley, New York.

Gold, E. M. (1967). Language identification in the limit. *Information and Control, 10,* 447–474.

Golden, K. (1998). Leap before you look: Information gathering in the PUCCINI planner. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS-98),* pp. 70–77, Pittsburgh, Pennsylvania. AAAI Press.

Goldman, N. (1975). Conceptual generation. In Schank, R. (Ed.), *Conceptual Information Processing,* chap. 6. North-Holland, Amsterdam.

Goldman, R. and Boddy, M. (1996). Expressive planning and explicit knowledge. In *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96),* pp. 110–117, Edinburgh, Scotland. AAAI Press.

Gomes, C., Selman, B., and Kautz, H. (1998). Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98),* pp. 431–437, Madison, Wisconsin. AAAI Press.

Good, I. J. (1950). Contribution to the discussion of Eliot Slater's „Statistics for the chess computer and the factor of mobility". In *Symposium on Information Theory,* p. 199, London. Ministry of Supply.

Good, I. J. (1961). A causal calculus. *British Journal of the Philosophy of Science, 11,* 305–318.

Good, I. J. (1965). Speculations concerning the first ultraintelligent machine. In Alt, F. L. and Rubinoff, M. (Eds.), *Advances in Computers,* Vol. 6, pp. 31–88. Academic Press, New York.

Goodman, D. and Keene, R. (1997). *Man versus Machine: Kasparov versus Deep Blue.* H3 Publications, Cambridge, Massachusetts.

Goodman, N. (1954). *Fact, Fiction and Forecast.* University of London Press, London.

Goodman, N. (1977). *The Structure of Appearance* (3rd edition). D. Reidel, Dordrecht, Netherlands.

Gordon, M. J., Milner, A. J., and Wadsworth, C. P. (1979). *Edinburgh LCF.* Springer-Verlag, Berlin.

Gordon, N. J. (1994). *Bayesian methods for tracking.* Ph.D. thesis, Imperial College, University of London.

Gordon, N. J., Salmond, D. J., and Smith, A. F. M. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings F (Radar and Signal Processing), 140*(2), 107–113.

Gorry, G. A. (1968). Strategies for computer-aided diagnosis. *Mathematical Biosciences, 2*(3–4, 293–318.

Gorry, G. A., Kassirer, J. P., Essig, A., and Schwartz, W. B. (1973). Decision analysis as the basis for computer-aided management of acute renal failure. *American Journal of Medicine, 55,* 473–484.

Gottlob, G., Leone, N., and Scarcello, F. (1999a). A comparison of structural CSP decomposition methods. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99),* pp. 394–399, Stockholm. Morgan Kaufmann.

Created by XMLmind XSL-FO Converter.

Gottlob, G., Leone, N., and Scarcello, F. (1999b). Hypertree decompositions and tractable queries. In *Proceedings of the 18th ACM International Symposium on Principles of Database Systems*, pp. 21–32, Philadelphia. Association for Computing Machinery.

Graham, S. L., Harrison, M. A., and Ruzzo, W. L. (1980). An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems, 2*(3), 415–462.

Grassmann, H. (1861). *Lehrbuch der Arithmetik.* Th. Chr. Fr. Enslin, Berlin.

Grayson, C. J. (1960). Decisions under uncertainty: Drilling decisions by oil and gas operators. Tech. rep., Division of Research, Harvard Business School, Boston.

Green, B., Wolf, A., Chomsky, C., and Laugherty, K. (1961). BASEBALL: An automatic question answerer. In *Proceedings of the Western Joint Computer Conference*, pp. 219–224.

Green, C. (1969a). Application of theorem proving to problem solving. In *Proceedings of the First International Joint Conference on Artificial Intelligence (ijcai-69)*, pp. 219–239, Washington, DC. IJCAII.

Green, C. (1969b). Theorem-proving by resolution as a basis for question-answering systems. In Meltzer, B., Michie, D., and Swann, M. (Eds.), *Machine Intelligence 4*, pp. 183–205. Edinburgh University Press, Edinburgh, Scotland.

Green, C. and Raphael, B. (1968). The use of theorem-proving techniques in question-answering systems. In *Proceedings of the 23rd ACM National Conference*, Washington, DC. ACM Press.

Greenblatt, R. D., Eastlake, D. E., and Crocker, S. D. (1967). The Greenblatt chess program. In *Proceedings of the Fall Joint Computer Conference*, pp. 801–810. American Federation of Information Processing Societies (AFIPS).

Greiner, R. (1989). Towards a formal analysis of EBL. In *Proceedings of the Sixth International Machine Learning Workshop*, pp. 450–453, Ithaca, NY. Morgan Kaufmann.

Grice, H. P. (1957). Meaning. *Philosophical Review, 66*, 377–388.

Grosz, B. J., Joshi, A. K., and Weinstein, S. (1995). Centering: A framework for modeling the local coherence of discourse. *Computational Linguistics, 21*(2), 203–225.

Grosz, B. J. and Sidner, C. L. (1986). Attention, intentions, and the structure of discourse. *Computational Linguistics, 12*(3), 175–204.

Grosz, B. J., Sparck Jones, K., and Webber, B. L. (Eds.). (1986). *Readings in Natural Language Processing.* Morgan Kaufmann, San Mateo, California.

Grove, W. and Meehl, P. (1996). Comparative efficiency of informal (subjective, impressionistic) and formal (mechanical, algorithmic) prediction procedures: The clinical statistical controversy. *Psychology, Public Policy, and Law, 2*, 293–323.

Gu, J. (1989). *Parallel Algorithms and Architectures for Very Fast AI Search.* Ph.D. thesis, University of Utah.

Guard, J., Oglesby, F., Bennett, J., and Settle, L. (1969). Semi-automated mathematics. *Journal of the Association for Computing Machinery, 16*, 49–62.

Guibas, L. J., Knuth, D. E., and Sharir, M. (1992). Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica, 7*, 381–413. See also *17th Int. Coll. on Automata, Languages and Programming*, 1990, pp. 414–431.

Haas, A. (1986). A syntactic theory of belief and action. *Artificial Intelligence, 28*(3), 245–292.

Hacking, I. (1975). *The Emergence of Probability.* Cambridge University Press, Cambridge, UK.

Hald, A. (1990). *A History of Probability and Statistics and Their Applications before 1750.* Wiley, New York.

Halpern, J. Y. (1990). An analysis of first-order logics of probability. *Artificial Intelligence, 46*(3), 311–350.

Created by XMLmind XSL-FO Converter.

Hamming, R. W. (1991). *The Art of Probability for Scientists and Engineers.* Addison-Wesley, Reading, Massachusetts.

Hammond, K. (1989). *Case-Based Planning: Viewing Planning as a Memory Task.* Academic Press, New York.

Hamscher, W., Console, L., and Kleer, J. D. (1992). *Readings in Model-based Diagnosis.* Morgan Kaufmann, San Mateo, California.

Handschin, J. E. and Mayne, D. Q. (1969). Monte Carlo techniques to estimate the conditional expectation in multi-stage nonlinear filtering. *International Journal of Control, 9*(5), 547–559.

Hansen, E. (1998). Solving POMDPs by searching in policy space. In *Uncertainty in Artificial Intelligence: Proceedings of the Fourteenth Conference,* pp. 211–219, Madison, Wisconsin. Morgan Kaufmann.

Hansen, E. and Zilberstein, S. (2001). LAO*: a heuristic search algorithm that finds solutions with loops. *Artificial Intelligence, 129*(1–2), 35–62.

Hansen, P. and Jaumard, B. (1990). Algorithms for the maximum satisfiability problem. *Computing, 44*(4), 279–303.

Hanski, I. and Cambefort, Y. (Eds.). (1991). *Dung Beetle Ecology.* Princeton University Press, Princeton, New Jersey.

Hansson, O. and Mayer, A. (1989). Heuristic search as evidential reasoning. In *Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence,* Windsor, Ontario. Morgan Kaufmann.

Hansson, O., Mayer, A., and Yung, M. (1992). Criticizing solutions to relaxed models yields powerful admissible heuristics. *Information Sciences, 63*(3), 207–227.

Haralick, R. M. and Elliot, G. L. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence, 14*(3), 263–313.

Hardin, G. (1968). The tragedy of the commons. *Science, 162,* 1243–1248.

Harel, D. (1984). Dynamic logic. In Gabbay, D. and Guenthner, F. (Eds.), *Handbook of Philosophical Logic,* Vol. 2, pp. 497–604. D. Reidel, Dordrecht, Netherlands.

Harman, G. H. (1983). *Change in View: Principles of Reasoning.* MIT Press, Cambridge, Massachusetts.

Harsanyi, J. (1967). Games with incomplete information played by Bayesian players. *Management Science, 14,* 159–182.

Hart, P. E., Nilsson, N. J. and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics, SSC-4*(2), 100–107.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1972). Correction to „A formal basis for the heuristic determination of minimum cost paths". *SIGART Newsletter, 37,* 28–29.

Hart, T. P. and Edwards, D. J. (1961). The tree prune (TP) algorithm. Artificial intelligence project memo 30, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Hartley, R. and Zisserman, A. (2000). *Multiple view geometry in computer vision.* Cambridge University Press, Cambridge, UK.

Haslum, P. and Geffner, H. (2001). Heuristic planning with time and resources. In *Proceedings of the IJCAI-01 Workshop on Planning with Resources,* Seattle.

Hastie, T. and Tibshirani, R. (1996). Discriminant adaptive nearest neighbor classification and regression. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E. (Eds.), *Advances in Neural Information Processing Systems,* Vol. 8, pp. 409–15. MIT Press, Cambridge, Massachusetts.

Created by XMLmind XSL-FO Converter.

Hastie, T., Tibshirani, R. and Friedman, J. (2001). *The Elements of Statistical Learning: Data Mining, Inference and Prediction.* Springer-Verlag, Berlin.

Haugeland, J. (Ed.). (1981). *Mind Design.* MIT Press, Cambridge, Massachusetts.

Haugeland, J. (Ed.). (1985). *Artificial Intelligence: The Very Idea.* MIT Press, Cambridge, Massachusetts.

Haussler, D. (1989). Learning conjunctive concepts in structural domains. *Machine Learning, 4*(1), 7–40.

Havelund, K., Lowry, M., Park, S., Pecheur, C., Penix, J., Visser, W., and White, J. L. (2000). Formal analysis of the remote agent before and after flight. In *Proceedings of the 5th NASA Langley Formal Methods Workshop,* Williamsburg, VA.

Hayes, P. J. (1978). The naive physics manifesto. In Michie, D. (Ed.), *Expert Systems in the Microelectronic Age.* Edinburgh University Press, Edinburgh, Scotland.

Hayes, P. J. (1979). The logic of frames. In Metzing, D. (Ed.), *Frame Conceptions and Text Understanding,* pp. 46–61. de Gruyter, Berlin.

Hayes, P. J. (1985a). Naive physics I: Ontology for liquids. In Hobbs, J. R. and Moore, R. C. (eds.), *Formal Theories of the Commonsense World,* chap. 3, pp. 71–107. Ablex, Norwood, New Jersey.

Hayes, P. J. (1985b). The second naive physics manifesto. In Hobbs, J. R. and Moore, R. C. (Eds.), *Formal Theories of the Commonsense World,* chap. 1, pp. 1–36. Ablex, Norwood, New Jersey.

Hebb, D. O. (1949). *The Organization of Behavior.* Wiley, New York.

Heckerman, D. (1986). Probabilistic interpretation for MYCIN's certainty factors. In Kanal, L. N. and Lemmer, J. F. (Eds.), *Uncertainty in Artificial Intelligence,* pp. 167–196. Elsevier/North-Holland, Amsterdam, London, New York.

Heckerman, D. (1991). *Probabilistic Similarity Networks.* MIT Press, Cambridge, Massachusetts.

Heckerman, D. (1998). A tutorial on learning with Bayesian networks. In Jordan, M. I. (Ed.), *Learning in graphical models.* Kluwer, Dordrecht, Netherlands.

Heckerman, D., Geiger, D., and Chickering, D. M. (1994). Learning Bayesian networks: The combination of knowledge and statistical data. Technical report MSR-TR-94-09, Microsoft Research, Redmond, Washington.

Heim, I. and Kratzer, A. (1998). *Semantics in a Generative Grammar.* Blackwell, Oxford, UK.

Heinz, E. A. (2000). *Scalable search in computer chess.* Vieweg, Braunschweig, Germany.

Held, M. and Karp, R. M. (1970). The traveling salesman problem and minimum spanning trees. *Operations Research, 18,* 1138–1162.

Helmert, M. (2001). On the complexity of planning in transportation domains. In Cesta, A. and Barrajo, D. (Eds.), *Sixth European Conference on Planning (ECP-01),* Toledo, Spain. Springer-Verlag.

Hendrix, G. G. (1975). Expanding the utility of semantic networks through partitioning. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75),* pp. 115–121, Tbilisi, Georgia. IJCAII.

Henrion, M. (1988). Propagation of uncertainty in Bayesian networks by probabilistic logic sampling. In Lemmer, J. F. and Kanal, L. N. (Eds.), *Uncertainty in Artificial Intelligence 2,* pp. 149–163. Elsevier/North-Holland, Amsterdam, London, New York.

Henzinger, T. A. and Sastry, S. (Eds.). (1998). *Hybrid systems: Computation and control.* Springer-Verlag, Berlin.

Herbrand, J. (1930). *Recherches sur la Théorie de la Démonstration.* Ph.D. thesis, University of Paris.

Created by XMLmind XSL-FO Converter.

Hewitt, C. (1969). PLANNER: a language for proving theorems in robots. In *Proceedings of the First International Joint Conference on Artificial Intelligence (IJCAI-69)*, pp. 295–301, Washington, DC. IJCAII.

Hierholzer, C. (1873). Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen, 6*, 30–32.

Hilgard, E. R. and Bower, G. H. (1975). *Theories of Learning* (4th edition). Prentice-Hall, Upper Saddle River, New Jersey.

Hintikka, J. (1962). *Knowledge and Belierf.* Cornell University Press, Ithaca, New York.

Hinton, G. E. and Anderson, J. A. (1981). *Parallel Models of Associative Memory.* Lawrence Erlbaum Associates, Potomac, Maryland.

Hinton, G. E. and Nowlan, S. J. (1987). How learning can guide evolution. *Complex Systems, 1*(3), 495–502.

Hinton, G. E. and Sejnowski, T. (1983). Optimal perceptual inference. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 448–453, Washington, DC. IEEE Computer Society Press.

Hinton, G. E. and Sejnowski, T. (1986). Learning and relearning in Boltzmann machines. In Rumelhart, D. E. and McClelland, J. L. (Eds.), *Parallel Distributed Processing*, chap. 7, pp. 282–317. MIT Press, Cambridge, Massachusetts.

Hirsh, H. (1987). Explanation-based generalization in a logic programming environment. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, Milan. Morgan Kaufmann.

Hirst, G. (1981). *Anaphora in Natural Language Understanding: A Survey*, Vol. 119 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin.

Hirst, G. (1987). *Semantic Interpretation against Ambiguity.* Cambridge University Press, Cambridge, UK.

Hobbs, J. R. (1978). Resolving pronoun references. *Lingua, 44*, 339–352.

Hobbs, J. R. (1990). *Literature and Cognition.* CSLI Press, Stanford, California.

Hobbs, J. R., Appelt, D., Bear, J., Israel, D., Kameyama, M., Stickel, M. E., and Tyson, M. (1997). FASTUS: A cascaded finite-state transducer for extracting information from natural-language text. In Roche, E. and Schabes, Y. (Eds.) *Finite-State Devices for Natural Language Processing*, pp. 383–406. MIT Press, Cambridge, Massachusetts.

Hobbs, J. R. and Moore, R. C. (Eds.). (1985). *Formal Theories of the Commonsense World.* Ablex, Norwood, New Jersey.

Hobbs, J. R., Stickel, M. E., Appelt, D., and Martin, P. (1993). Interpretation as abduction. *Artificial Intelligence, 63*(1–2), 69–142.

Hoffmann, J. (2000). A heuristic for domain independent planning and its use in a n enforced hillclimbing algorithm. In *Proceedings of the 12th International Symposium on Methodologies for Intelligent Systems*, pp. 216–227, Charlotte, North Carolina. Springer-Verlag.

Hogan, N. (1985). Impedance control: An approach to manipulation. parts i, ii, and iii. *Transactions ASME Journal of Dynamics, Systems, Measurement, and Control, 107*(3), 1–24.

Holland, J. H. (1975). *Adaption in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Michigan.

Holland, J. H. (1995). *Hidden order: How adaptation builds complexity.* Addison-Wesley, Reading, Massachusetts.

Holldobler, S. and Schneeberger, J. (1990). A new deductive approach to planning. *New Generation Computing, 8*(3), 225–244.

Created by XMLmind XSL-FO Converter.

Holzmann, G. J. (1997). The Spin model checker. *ISSS Transactions on Software Engineering, 23*(5), 279–295.

Hood, A. (1824). Case 4th–28 July 1824 (Mr. Hood's cases of injuries of the brain). *The Phrenological Journal and Miscellany, 2,* 82–94.

Hopfield, J. J. (1982). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences of the United States of America, 79,* 2554–2558.

Horn, A. (1951). On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic, 16,* 14–21.

Horn, B. K. P. (1970). Shape from shading: A method for obtaining the shape of a smooth opeque object from one view. Technical report 232, MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts.

Horn, B. K. P. (1986). *Robot Vision.* MIT Press, Cambridge, Massachusetts.

Horn, B. K. P. and Brooks, M. J. (1989). *Shape from Shading.* MIT Press, Cambridge, Massachusetts.

Horning, J. J. (1969). *A study of grammatical inference.* Ph.D. thesis, Stanford University.

Horowitz, E. and Sahni, s. (1978). *Fundamentals of computer algorithms.* Computer Science Press, Rockwille, Maryland.

Horswill, I. (2000. Functional programming of behavior-based systems. *Autonomous Robots, 9,* 83–93.

Horvitz, E. J. (1987). Problem-solving design: Reasoning about computational value, trade-offs, and resources. In *Proceedings of the Second Annual NASA Research Forum,* pp. 26–43, Moffett Field, California. NASA Ames Research Center.

Horvitz, E. J. (1989). Rational metareasoning and compilation for optimizing decisions under bounded resources. In *Proceedings of Computational Intelligence 89,* Milan. Association for Computing Machinery.

Horvitz, E. J. and Barry, M. (1995). Display of information for time-critical decision making. In *Uncertainty in Artificial Intelligence: Proceedings of the Eleventh Conference,* pp. 296–305, Montreal Canada. Morgan Kaufmann.

Horvitz, E. J., Breese, J. S., Heckerman, D., and Hovel, D. (1998). The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Uncertainty in Artificial Intelligence: Proceedings of the Fourteenth Conference,* pp. 256–265, Madison, Wisconsin. Morgan Kaufmann.

Horvitz, E. J., Breese, J. S., and Henrion, M. (1988). Decision theory in expert systems and artificial intelligence. *International Journal of Approximate Reasoning, 2,* 247–302.

Horvitz, E. J. and Breese, J. S. (1996). Ideal partition of resources for metareasoning. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96),* pp. 1229–1234, Portland, Oregon. AAAI Press.

Horvitz, E. J. and Heckerman, D. (1986). The inconsistent use of measures of certainty in artificial intelligence research. In Kanal, L. N. and Lemmer, J. F. (Eds.), *Uncertainty in Artificial Intelligence,* pp. 137–151. Elsevier/North-Holland, Amsterdam, London, New York.

Horvitz, E. J., Heckerman, D., and Langlotz, C. P. (1986). A framework for comparing alternative formalisms for plausible reasoning. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86),* Vol. 1, pp. 210–214, Philadelphia. Morgan Kaufmann.

Hovy, E. (1988). *Generating Natural Language under Progmatic Constraints.* Lawrence Erlbaum, Potomac, Maryland.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, Massachusetts.

Howard, R. A. (1966). Information value theory. *IEEE Transactions on Systems Science and Cybernetics, SSC-2,* 22–26.

Created by XMLmind XSL-FO Converter.

Howard, R. A. (1977). Risk preference. In Howard, R. A. and Matheson, J. E. (Eds.), *Readings in Decision Analysis,* pp. 429–465. Decision Analysis Group, SRI International, Menlo Park, California.

Howard, R. A. (1989). Microrisks for medical decision analysis. *International Journal of Technology Assessment in Health Care, 5,* 357–370.

Howard, R. A. and Matheson, J. e. (1984). Influence diagrams. In Howard, R. A. and Matheson, J. E. (Eds.), *Readings on the Principles and Applications of Decision Analysis,* pp. 721–762. Strategic Decisions Group, Menlo Park, California.

Hsu, F.-H. (1999). IBM's Deep Blue chess grandmaster chips. *IEEE Micro, 19*(2), 70–80.

Hsu, F.-H., Anantharaman, T. S., Campbell, M. S., and Nowatzyk, A. (1990). A grandmaster chess machine. *Scientific American, 263*(4), 44–50.

Huang, T., Koller, D., Malik, J., Ogasawara, G., Rao, B., Russell, S. J., and Weber, J. (1994). Automatic symbolic traffic scene analysis using belief networks. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94),* pp. 966–972, Seattle. AAAI Press.

Huang, X. D., Acero, A., and Hon, H. (2001). *Spoken Language Processing.* Prentice Hall, Upper Saddle River, New Jersey.

Hubel, D. H. (1988). *Eye, Brain, and Vision.* W. H. Freeman, New York.

Huddleston, R. D. and Pullum, G. K. (2002). *The Cambridge Grammar of the English Language.* Cambridge University Press, Cambridge, UK.

Huffman, D. A. (1971). Impossible objects as nonsense sentences. In Meltzer, B. and Michie, D. (Eds.), *Machine Intelligence 6,* pp. 295–324. Edinburgh University Press, Edinburgh, Scotland.

Hughes, B. D. (1995). *Random Walks and Random Environments, Vol. 1: Random Walks.* Oxford University Press, Oxford, UK.

Huhns, M. N. and Singh, M. P. (Eds.). (1998). *Readings in agents.* Morgan Kaufmann, San Mateo, California.

Hume, D. (1739). *A Treatise of Human Nature* (2nd edition). Republished by Oxford University Press, 1978, Oxford, UK.

Hunsberger, L. and Grosz, B. J. (2000). A combinatorial auction for collaborative planning. In *International Conference on Multi-Agent Systems (ICMAS-2000).*

Hunt, E. B., Marin, J., and Stone, P. T. (1966). *Experiments in Induction.* Academic Press, New York.

Hunter, L. and States, D. J. (1992). Bayesian classification of protein structure. *IEEE Expert, 7*(4), 67–75.

Hurwicz, L. (1973). The design of mechanisms for resource allocation. *American Economic Review Papers and Proceedings, 63*(1), 1–30.

Hutchins, W. J. and Somers, H. (1992). *An Introduction to Machine Translation.* Academic Press, New York.

Huttenlocher, D. P. and Ullman, S. (1990). Recognizing solid objects by alignment with an image. *International Journal of Computer Vision, 5*(2), 195–212.

Huygens, C. (1657). Ratiociniis in ludo aleae. In van Schooten, F. (Ed.), *Exercitionum Mathematicorum.* Elsevirii, Amsterdam.

Hwa, R. (1998). An empirical evaluation of probabilistic lexicalized tree insertion grammars. In *Proceedings of COLING-ACL '98,* pp. 557–563, Montreal. International Committee on Computational Linguistics and Association for Computational Linguistics.

Hwang, C. H. and Schubert, L. K. (1993). EL: A formal, yet natural, comprehensive knowledge representation. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93),* pp. 676-682, Washington, DC. AAAI Press.

Created by XMLmind XSL-FO Converter.

Indyk, P. (2000). Dimensionality reduction techniques for proximity problems. In *Proceedings of the Eleventh Annual ACM–SIAM Symposium on Discrete Algorithms*, pp. 371–378, San Francisco. Association for Computing Machinery.

Ingerman, P. Z. (1967). Panini–Backus form suggested. *Communications of the Association for Computing Machinery, 10*(3), 137.

Inoue, K. (2001). Inverse entailment for full clausal theories. In *LICS-2001 Workshop on Logic and Learning*, Boston, IEEE.

Intille, S. and Bobick, A. (1999). A framework for recognizing multi-agent action from visual evidence. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pp. 518–525, Orlando, Florida. AAAI Press.

Isard, M. and Blake, A. (1996). Contour tracking by stochastic propagation of conditional density. In *Proceedings of Fourth European conference on Computer Vision*, pp. 343–356, Cambridge, UK. Springer-Verlag.

Jaakkola, T. and Jordan, M. I. (1996). Computing upper and lower bounds on likelihoods in intractable networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference*, pp. 340–348. Morgan Kaufmann, Portland, Oregon.

Jaakkola, T., Singh, S. P., and Jordan, M. I. (1995). Reinforcement learning algorithm for partially observable Markov decision problems. In Tesauro, G., Touretzky, D., and Leen, T. (Eds.), *Advances in Neural Information Processing Systems 7*, pp. 345–352, Cambridge, Massachusets. MIT Press.

Jaffar, J. and Lassez, J.-L. (1987). Constraint logic programming. In *Proceedings of the Fourteenth ACM Conference on Principles of Programming Languages*, pp. 111–119, Munich. Association for Computing Machinery.

Jaffar, J., Michaylov, S., Stuckey, P. J., and Yap, R. H. C. (1992a). The CLP(R) language and system. *ACM Transactions on Programming Languages and Systems, 14*(3), 339–395.

Jaffar, J., Stuckey, P. J., Michaylov, S., and Yap, R. H. C. (1992b). An abstract machine for CLP(R)**. SIGPLAN Notices, 27**(7), 128–139.

Jáskowski, S. (1934). On the rules of suppositions in formal logic. *Studia Logica, 1*.

Jefferson, G. (1949). The mind of mechanical man: The Lister Oration delivered at the Royal College of Surgeons in England. *British Medical Journal, 1*(25), 1105–1121.

Jeffrey, R. C. (1983). *The Logic of Decision* (2nd edition). University of Chicago Press, Chicago.

Jeffreys, H. (1948). *Theory of Probability.* Oxford, Oxford, UK.

Jelinek, F. (1969). Fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development, 64*, 532–556.

Jelinek, F. (1976). Continuous speech recognition by statistical methods. *Proceedings of the IEEE, 64*(4), 532–556.

Jelinek, F. (1997). *Statistical methods for speech recognition.* MIT Press, Cambridge, Massachusetts.

Jelinek, F. and Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*, pp. 381–397, Amsterdam, London, New York. North Holland.

Jennings, H. S. (1906). *Behavior of the lower organisms.* Columbia University Press, New York.

Jensen, F. V. (2001). *Bayesian Networks and Decision Graphs.* Springer-Verlag, Berlin.

Jespersen, O. (1965). *Essentials of English Grammar.* University of Alabama Press, Tuscaloosa, Alabama.

Created by XMLmind XSL-FO Converter.

Jevons, W. S. (1874). *The Principles of Science.* Routledge/Thoemmes Press, London.

Jimenez, P. and Torras, C. (2000). An efficient algorithm for searching implicit AND/OR graphs with cycles. *Artificial Intelligence, 124*(1), 1–30.

Joachims, T. (2001). A statistical learning model of text classification with support vector machines. In *Proceedings of the 24th Conference on Research and Development in Information Retrieval (SIGIR),* pp. 128–136, New Orleans. Association for Computing Machinery.

Johnson, W. W. and Story, W. E. (1879). Notes On the „15" puzzle. *American Journal of Mathematics, 2,* 397–404.

Johnson-Laird, P. N. (1988). *The Computer and the Mind: An Introduction to Cognitive Science.* Harvard University Press, Cambridge, Massachusetts.

Johnston, M. D. and Adorf, H.-M. (1992). Scheduling with neural networks: The case of the Hubble space telescope. *Computers & Operations Research, 19*(3–4), 209–240.

Jones, N. D., Gomard, C. K., and Sestoft, P. (1993). *Partial Evaluation and Automatic Program Generation.* Prentice-Hall, Upper Saddle River, New Jersey.

Jones, R., Laird, J. E., and Nielsen, P. E. (1998). Automated intelligent pilots for combat flight simulation. In *Preceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98),* pp. 1047–54, Madison, Wisconsin. AAAI Press.

Jonsson, A., Morris, P., Muscettola, N., Rajan, K., and Smith, B. (2000). Planning in interplanetary space: Theory and practice. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS-00),* pp. 177–186, Breckenridge, Colorado. AAAI Press.

Jordan, M. I. (1995). Why the logistic funciton? a tutorial discussion on probabilities and neural networks. Computational cognitive science technical report 9503, Massachusetts Institute of Technology.

Jordan, M. I. (2003). *An Introduction to Graphical Models.* In press.

Jordan, M. I., Ghahramani, Z., Jaakkola, T., and Saul, L. K. (1998). An introduction to variational methods for graphical models. In Jordan, M. I. (Ed.), *Learning in Graphical Models.* Kluwer, Dordrecht, Netherlands.

Jordan, M. I., Ghahramani, Z., Jaakkola, T., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine Learning, 37*(2–3), 183–233.

Joshi, A. K. (1985). Tree-adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions. In Dowty, D., Karttunen, L., and Zwicky, A. (Eds.), *Natural Language Parsing.* Cambridge University Press, Cambridge, UK.

Joshi, A. K., Webber, B. L., and Sag, I. (1981). *Elements of Discourse Understanding.* Cambridge University Press, Cambridge, UK.

Joslin, D. and Pollack, M. E. (1994). Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94),* p. 1506, Seattle. AAAI Press.

Jouannaud, J.-P. and Kirchner, C. (1991). Solving equations in abstract algebras: A rule-based survey of unification. In Lassez, J.-L. and Plotkin, G. (Eds.), *Computational Logic,* pp. 257–321. MIT Press, Cambridge, Massachusetts.

Judd, J. S. (1990). *Neural Network Design and the Complexity of Learning.* MIT Press, Cambridge, Massachusetts.

Juels, A. and Wattenberg, M. (1996). Stochastic hill-climbing as a baseline method for evaluating genetic algorithms. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E. (Eds.), *Advances in Neural Information Processing Systems,* Vol. 8, pp. 430–6. MIT Press, Cambridge, Massachusetts.

Julesz, B. (1971). *Foundations of Cyclopean Perception.* University of Chicago Press, Chicago.

Created by XMLmind XSL-FO Converter.

Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice-Hall, Upper Saddle River, New Jersey.

Kadane, J. B. and Larkey, P. D. (1982). Subjective probability and the theory of games. *Management Science, 28*(2), 113–120.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence, 101*, 99–134.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research, 4*, 237–285.

Kaelbling, L. P. and Rosenschein, S. J. (1990). Action and planning in embedded agents. *Robotics and Autonomous Systems, 6*(1–2), 35–48.

Kager, R. (1999). *Optimality Theory.* Cambridge University Press, Cambridge, UK.

Kahneman, D., Slovic, P., and Tversky, A. (Eds.). (1982). *Judgment under Uncertainty: Heuristics and Biases.* Cambridge University Press, Cambridge, UK.

Kaindl, H. and Khorsand, A. (1994). Memory-bounded bidirectional search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94),* pp. 1359–1364, Seattle. AAAI Press.

Kalman, R. (1960). A new approach to linear filtering and prediction problems. *Journal of Basic Engineering, 82*, 35–46.

Kambhampati, S. (1994). Exploiting causal structure to control retrieval and refitting during plan reuse. *Computational Intelligence, 10*, 213–244.

Kambhampati, S., Mali, A. D., and Srivastava, B. (1998). Hybrid planning for partially hierarchical domains. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98),* pp. 882–888, Madison, Wisconsin. AAAI Press.

Kanal, L. N. and Kumar, V. (1988). *Search in Artificial Intelligence.* Springer-Verlag, Berlin.

Kanal, L. N. and Lemmer, J. f. (Eds.) (1986). *Uncertainty in Artificial Intelligence.* Elsevier/North-Holland, Amsterdam, London, New York.

Kanazawa, K., Koller, D., and Russell, S. J. (1995). Stochastic simulation algorithms for dynamic probabilistic networks. In *Uncertainty in Artificial Intelligence: proceedings of the Eleventh Conference,* pp. 346–351, Montreal, Canada. Morgan Kaufmann.

Kaplan, D. and Montague, R. (1960). A paradox regained. *Notre Dame Journal of Formal Logic, 1*(3), 79–90.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica, 4*, 373–395.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W. (Eds.), *Complexity of Computer Computations,* pp. 85–103. Plenum, New York.

Kasami, T. (1965). An efficient recognition and syntax analysis algorithm for context-free languages. Tech. rep. AFCRL-65–758, Air Force Cambridge Research Laboratory, Bedford, Massachusetts.

Kasparov, G. (1997). IBM owes me a rematch. *Time, 149*(21), 66–67.

Kasper, R. T. (1988). Systemic grammar and functional unification grammar. In Benson, J. and Greaves, W. (Eds.), *Systemic Functional Approaches to Discourse.* Ablex, Norwood, New Jersey.

Kaufmann, M., Manolios, P., and Moore, J. S. (2000). *Computer-Aided Reasoning: An Approach.* Kluwer, Dordrecht, Netherlands.

Created by XMLmind XSL-FO Converter.

Kautz, H. McAllester, D. A., and Selman, B. (1996). Encoding plans in propositional logic. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, pp. 374–384, Cambridge, Massachusetts. Morgan Kaufmann.

Kautz, H. and Selman, b. (1992). Planning as satisfiability. In *ECAI 92: 10th European Conference on Artificial Intelligence Proceedings*, pp. 359–363, Vienna. Wiley.

Kautz, H. and Selman, B. (1998). BLACKBOX: A new aproach to the application of theorem proving to problem solving. Working Notes of the AIPS-98 Workshop on Planning as Combinatorial Search.

Kavraki, L., Svestka, P., Latombe, J.-C., and Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, *12*(4), 566–580.

Kay, M., Gawron, J. M., and Norvig, P. (1994). *Verbmobil: A Translation System for Face-ToFace Dialog.* CSLI Press, Stanford, California.

Kaye, r. (2000). Minesweeper is NP-complete!. *Mathematical Intelligencer*, *5*(22), 9–15.

Kearns, M. (1990). *The Computational Complexity of Machine Learning*. MIT Press, Cambridge, Massachusetts.

Kearns, M., Mansour, Y., and Ng, A. Y. (2000). Approximate planning in large POMDPs via reusable trajectories. In Solla, S. A., Leen, T. K., and Müller, K.-R. (Eds.), *Advances in Neural Information Processing Systems 12*. MIT Press, Cambridge, Massachusetts.

Kearns, M. and Singh, S. P. (1998). Near-optimal reinforcement learning in polynomial time. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 260–268, Madison, Wisconsin. Morgan Kaufmann.

Kearns, M. and Vazirani, U. (1994). *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, Massachusetts.

Keeney, R. L. (1974). Multiplicative utility functions. *Operations Research*, *22*, 22–34.

Keeney, R. L. and Raiffa, H. (1976). *Decisions with Multiple Objectives: preferences and Value Tradeoffs*. Wiley, New York.

Kehler, A. (1997). Probabilistic coreference in information extraction. In Cardie, C. and Weischedel, R. (Eds.), *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, pp. 163–173. Association for Computational Linguistics, Somerset, New Jersey.

Kemp, M. (Ed.). (1989). *Leonardo on Painting: An Anthology of Writings.* Yale University Press, New Haven, Connecticut.

Kern, C. and Greenstreet, M. R. (1999). Formal verification in hardware design: A survey. *ACM Transactions on Design Automation of Electronic Systems*, *4*(2), 123–193.

Keynes, J. M. (1921). *A Treatise on Probability.* Macmillan, London.

Khatib, O. (1986). Real-time obstacle avoidance for robot manipulator and mobile robots. *The International Journal of Robotics Research*, *5*(1), 90–98.

Kietz, J.-U. and Dzeroski, S. (1994). Inductive logic programming and learnability. *SIGART Bulletin*, *5*(1), 22–32.

Kim, J. H. (1983). *CONVINCE: A Conversational Inference Consolidation Engine.* Ph.D. thesis, Department of Computer Science, University of California at Los Angeles.

Kim, J. H. and Pearl, J. (1983). A computational model for combined causal and diagnostic reasoning in inference systems. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, pp. 190–193, Karlsruhe, Germany. Morgan Kaufmann.

Created by XMLmind XSL-FO Converter.

Kim, J. H. and Pearl, J. (1987). CONVINCE: A conversational inference consolidation engine. *IEEE Transactions on Systems, Man, and Cybernetics, 17*(2), 120–132.

King, R. D., Muggleton, S. H., Lewis, R. A., and Sternberg, M. J. E. (1992). Drug design by machine learning: The use of inductive logic programming to model the structure activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the National Academy of Sciences of the United States of America, 89*(23), 11322–11326.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Sience, 220,* 671–680.

Kirkpatrick, S. and Selman, b. (1994). Critical behavior in the satisfiability of random Boolean expressions. *Science, 264*(5163), 1297–1301.

Kirousis, L. M. and Papadimitriou, C. H. (1988). The complexity of recognizing polyhedral scenes. *Journal of Computer and System Sciences, 37*(1), 14–38.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. (1997). RoboCup: The robot world cup initiative. In Johnson, W. L. and Hayes-Roth, B. (Eds.), *Proceedings of the First International Conference on Autonomous Agents*, pp. 340–347, New York. ACM Press.

Kjaerulff, U. (1992). A computational scheme for reasoning in dynamic probabilistic networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Eighth Conference,* pp. 121–129, Stanford, California. Morgan Kaufmann.

Klein, D. and Manning, C. D. (2001). Parsing with treebank grammars: Empirical bounds, theoretical models, and the structure of the Penn treebank. In *Proceedings of the 39th Annual Meeting of the ACL.*

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM, 46*(5), 604–632.

Knight, K. (1999). A statistical mt tutorial workbook. prepared in connection with the Johns Hopkins University summer workshop.

Knoblock, c. A. (1990). Learning abstraction hierarchies for problem solving. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90),* Vol. 2, pp. 923–928, Boston. MIT Press.

Knuth, D. E. (1968). Semantics for context-free languages. *Mathematical Systems Theory, 2*(2), 127–145.

Knuth, D. E. (1973). *The Art of Computer Programming* (second edition)., Vol. 2: Fundamental Algorithms. Addison-Wesley, Reading, Massachusetts.

Knuth, D. E. (1975). An analysis of alpha–beta pruning. *Artificial Intelligence, 6*(4), 293–326.

Knuth, D. E. and Bendix, P. B. (1970). Simple word problems in universal algebras. In Leech, J. (Ed.), *Computational Problems in Abstract Algebra,* pp. 263–267. Pergamon, Oxford, UK.

Koditschek, D. (1987). Exact robot navigation by means of potential functions: some topological considerations. In *Proceedings of the 1987 IEEE International Conference on Robotics and Automation,* Vol. 1, pp. 1–6, Raleigh, North Carolina. IEEE Computer Society Press.

Koehler, J., Nebel, B., Hoffman, J., and Dimopoulos, y. (1997). Extending planning graphs to an ADL subset. In *Proceedings of the Fourth European Conference on Planning,* pp. 273–285, Toulouse, France. Springer-Verlag.

Koenderink, J. J. (1990). *Solid Shape.* MIT Press, Cambridge, Massachusetts.

Koenderink, J. J. and van Doorn, A. J. (1975). Invariant properties of the motion parallax field due to the movement of rigid bodies relative to an boserver. *Optica Acta, 22*(9), 773–791.

Koenderink, J. J. and van Doorn, A. J. (1991). Affine structure from motion. *Journal of the Optical Society of America A, 8,* 377–385.

Created by XMLmind XSL-FO Converter.

Koenig, s. (1991). Optimal probabilistic and decision-theoretic planning using Markovian decision theory. Master's report, Computer Science Division, University of California, Berkeley.

Koenig, S. (2000). Exploring unknown environments with real-time search or reinforcement learning. In solla, S. A., Leen, T. K., and Müller, K.-R. (Eds.), *Advances in Neural Information Processing Systems 12*. MIT Press, Cambridge, Massachusetts.

Koenig, s. and Simmons, R. (1998). Solving robot navigation problems with initial pose uncertainty using real-time heuristic search. In *aips98*. AAAI Press, Menlo Park, California.

Kohn, W. (1991). Declarative control architecture. *Communications of the Association for Computing Machinery, 34*(8), 65–79.

Koller, D., Meggido, N., and von Stengel, B. (1996). Efficient computation of equilibria for extensive two-person games. *Games and Economic Behaviour, 14*(2), 247–259.

Koller, D. and Pfeffer, A. (1997). Representations and solutions for game-theoretic problems. *Artificial Intelligence, 94*(1–2), 167–215.

Koller, d. and Pfeffer, a. (1998). Probabilistic frame-based systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 580–587, Madison, Wisconsin. AAAI Press.

Koler, d. and Sahami, M. (1997). Hierarchically classifying documents using very few words. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 170-178. Morgan Kaufmann.

Kolmogorov, A. N. (1941). Interpolation und extrapolation von stationaren zufalligen folgen. *Bulletin of the Academy of Sciences of the USSR, Ser. Math. 5*, 3–14.

Kolmogorov, A. N. (1950). *Foundations of the Theory of Probability*. Chelsea, New York.

Kolmogorov, A. N. (1963). On tables of random numbers. *Sankhya, the Indian Journal of Statistics, Series A 25*.

Kolmogorov, A. N. (1965). Three approaches to the quantitative definition of information. *Problems in Information Transmission, 1*(1), 1–7.

Kolodner, J. (1983). Reconstructive memory: A computer model. *Cognitive Science, 7*, 281–328.

Kolodner, J. (1993). *Case-Based Reasoning*. Morgan Kaufmann, San Mateo, California.

Kondrak, G. and van Beek, P. (1997). A theoretical evaluation of selected backtracking algorithms. *Artificial Intelligence, 89*, 365–387.

Konolige, K. (1997). COLBERT: A language for reactive control in Saphira. In *KI-97: Advances in Artificial Intelligence*, LNAI, p. 31–52. Springer-Verlag.

Konolige, K. (1982). A first order formalization of knowledge and action for a multi-agent planning system. In Hayes, J. E., Michie, D., and Pao, Y.-H. (Eds.), *Machine Intelligence 10*. Ellis Horwood, Chichester, England.

Koopmans, T. C. (1972). Representation of preference orderings over time. In McGuire, C. B. and Radner, R. (Eds.), *Decision and Organization.* Elsevier/North-Holland, Amsterdam, London, New York.

Korf, R. E. (1985a). Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence, 27*(1), 97–109.

Korf, R. E. (1985b). Iterative-deepening A*: An optimal admissible tree search. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, pp. 1034–1036, Los Angeles. Morgan Kaufmann.

Korf, R. E. (1987). Planning as search: A quantitative approach. *Artificial Intelligence, 33*(1), 65–88.

Korf, R. E. (1988). Optimal path finding algorithms. In Kanal, L. N. and Kumar, V. (Eds.), *Search in Artificial Intelligence*, chap. 7, pp. 223–267. Springer-Verlag, Berlin.

Created by XMLmind XSL-FO Converter.

Korf, R. E. (1990). Real-time heuristic search. *Artificial Intelligence, 42*(3), 189–212.

Korf, R. E. (1991). Best-first search with limited memory. UCLA Computer Science Annual.

Korf, R. E. (1993). Linear-space best-first search. *Artificial Intelligence, 62*(1), 41–78.

Korf, R. E. (1995). Space-efficient search algorithms. *ACM Computing Surveys, 27*(3), 337–339.

Korf, R. E. and Chickering, D. M. (1996). Best-first minimax search. *Artificial Intelligence, 84*(1–2), 299–337.

Korf, R. E. and Felner, a. (2002). Disjoint pattern database heuristics. *Artificial Intelligence*, *134*(1–2), 9–22.

Korf, R. E. and Zhang, W. (2000). Divide-and-conquer frontier search applied to optimal sequence alignment. In *Proceedings of the 17th National Conference on Artificial Intelligence*, pp. 910–916, Cambridge, Massachusetts. MIT Press.

Kortenkamp, D., Bonasso, R. P., and Murphy, R. (Eds.). (1998). *AI-based Mobile Robots: Case studies of successful robot systems*, Cambridge, MA. MIT Press.

Kotok, a. (1962). A chess playing program for the IBM 7090. Ai project memo 41, MIT Computation Center, Cambridge, Massachusetts.

Koutsoupias, E. and Papadimitriou, C. H. (1992). On the greedy algorithm for satisfiability. *Information Processing Letters, 43*(1), 53–55.

Kowalski, R. (1974). Predicate logic as a programming language. In *Proceedings of the IFIP-74 Congress*, pp. 569–574. Elsevier/North-Holland.

Kowalski, R. (1979a). Algorithm = logic + control. *Communications of the Association for Computing Machinery, 22*, 424–436.

Kowalski, R. (1979b). *Logic for Problem Solving*. Elsevier/North-Holland, Amsterdam, London, New York.

Kowalski, R. (1988). The early years of logic programming. *Communications of the Association for Computing Machinery, 31*, 38–43.

Kowalski, R. and Kuehner, D. (1971). Linear resolution with selection function. *Artificial Intelligence, 2*(3–4), 227–260.

Kowalski, R. and Sergot, M. (1986). A logic-based calculus of events. *New Generation Computing, 4*(1), 67–95.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, Massachusetts.

Koza, J. R. (1994). *Genetic Programming II: Automatic discovery of reusable programs.* MIT Press, Cambridge, Massachusetts.

Koza, J. R., Bennett, F. H., Andre, D., and Keane, M. A. (1999). *Genetic Programming III: Darwinian invention and problem solving.* Morgan Kaufmann, San Mateo, California.

Kraus, S., Ephrati, E., and Lehmann, D. (1991). Negotiation in a non-cooperative environment. *Journal of Experimental and Theoretical Artificial Intelligence, 3*(4), 255–281.

Kripke, S. A. (1963). Semantical considerations on modal logic. *Acta Philosophica Fennica, 16*, 83–94.

Krovetz, R. (1993). Viewing morphology as an inference process. In *Proceedings of the Sixteenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pp. 191–202, New York. ACM Press.

Kruppa, E. (1913). Zur Ermittlkung eines Objecktes aus zwei Perspektiven mit innerer Orientierung. *Sitz.-Ber. Akad. Wiss., Wien, Math. Naturw., Kl. Abt. Ila, 122*, 1939–1948.

Created by XMLmind XSL-FO Converter.

Kuhn, H. W. (1953). Extensive games and the problem of information. In Kuhn, H. W. and Tucker, A. W. (Eds.), *Contributions to the Theory of Games II.* Princeton University Press, Princeton, New Jersey.

Kuipers, B. J. and Levitt, T. S. (1988). Navigation and mapping in large-scale space. *AI Magazine*, *9*(2), 25–43.

Kukich, K. (1992). Techniques for automatically correcting words in text. *ACM Computing Surveys*, *24*(4), 377–439.

Kumar, P. R. and Varaiya, P. (1986). *Stochastic systems: Estimation, identification, and adaptive control.* Prentice-Hall, Upper Saddle River, New Jersey.

Kumar, V. (1992). Algortihms for constraint satisfaction problems: A survey. *AI Magazine*, *13*(1), 32–44.

Kumar, V. and Kanal, L. N. (1983). A general branch and bound formulation for understanding and synthesizing and/or tree search procedures. *Artificial Intelligence*, *21*, 179–198.

Kumar, V. and Kanal, L. N. (1988). The CDP: A unifying formulation for heuristic search, dynamic programming, and branch-and-bound. In Kanal, L. N. and Kumar, V. (Eds.), *Search in Artificial Intelligence*, chap. 1, pp. 1–27. Springer-Verlag, Berlin.

Kumar, V., Nau, D. S., and Kanal, L. N. (1988). A general branch-and bound formulation for AND/OR grap and game tree search. In Kanal, L. N. and Kumar, V. (Eds.), *Search in Artificial Intelligence*, chap. 3, pp. 91–130. Springer-Verlag, Berlin.

Kuper, g. M. and Vardi, M. Y. (1993). On the complexity of queries in the logical data model. *Theoretical Computer Science*, *116*(1), 33–57.

Kurzweil, R. (1990). *The Age of Intelligent Machines.* MIT Press, Cambridge, Massachusetts.

Kurzweil, r. (2000). *The Age of Spiritual Machines.* Penguin.

Kyburg, H. E. (1977). Randomness and the right reference class. *The Journal of Philosophy*, *74*(9), 501–521.

Kyburg, H. E. (1983). The reference class. *Philosophy of Science*, *50*, 374–397.

La Mettrie, J. O. (1748). *L'homme machine.* E. Luzac, Leyde, France.

La Mura, P. and Shoham, Y. (1999). Expected utility networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Fifteenth Conference*, pp. 366–373, Stockholm. Morgan Kaufmann.

Ladkin, P. (1986a). Primitives and units for time specification. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Vol. 1, pp. 354–359, Philadelphia. Morgan Kaufmann.

Ladkin, P. (1986b). Time representation: a taxonomy of interval relations. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Vol. 1, pp. 360–366, Philadelphia. Morgan Kaufmann.

Lafferty, J. and Zhai, C. (2001). Probabilistic relevance models based on document and query generation. In *Proceedings of the Workshop on Language Modelling and Information retrieval.*

Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence*, *33*(1), 1–64.

Laird, J. E., Rosenbloom, P. S., and Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11–46.

Lakoff, G. (1987). *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind.* University of Chicago Press, Chicago.

Lakoff, G. and Johnson, M. (1980). *Metaphors We Live By.* University of Chicago Press, Chicago.

Lamarck, J. B. (1809). *Philosophie zoologique.* Chez Dentu et L'Auteur, Paris.

Created by XMLmind XSL-FO Converter.

Langley, P., Simon, H. A., Bradshaw, G. L., and Zytkow, J. M. (1987). *Scientific Discovery: Computational Explorations of the Creative Processes.* MIT Press, Cambridge, Massachusetts.

Langton, C. (Ed.). (1995). *Artificial life.* MIT Press, Cambridge, Massachusetts.

Laplace, P. (1816). *Essai philosophique sur les probabilités* (3rd edition). Courcier Imprimeur, Paris.

Lappin, S. and Leass, H. J. (1994). An algorithm for pronominal anaphora resolution. *Computational Linguistics, 20*(4), 535–561.

Lari, K. and Young, S. J. (1990). The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer, Speech and Language, 4,* 35–56.

Larrañaga, P., Kuijpers, C., Murga, R., Inza, I., and Dizdarevic, S. (1999). Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review, 13,* 129–170.

Latombe, J.-C. (1991). *Robot Motion Planning.* Kluwer, Dordrecht, Netherlands.

Lauritzen, S. (1995). The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis, 19,* 191–201.

Lauritzen, S. (1996). *Graphical models.* Oxford University Press, Oxford, UK.

Lauritzen, S., Dawid, A., Larsen, B., and Leimer, H. (1990). Independence properties of directed Markov fields. *Networks, 20*(5), 491–505.

Lauritzen, S. and Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, B 50*(2), 157–224.

Lauritzen, S. and Wermuth, N. (1989). Graphical models for associations between variables, some of which are qualitative and some quantitative. *Annals of Statistics, 17,* 31–57.

Lavrac, N. and Dzeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications.* Ellies Horwood, Chichester, England.

Lawler, E. L. (1985). *The travelling salesman problem: A guided tour of combinatorial optimization.* Wiley, New York.

Lawler, E. L., Lenstra, J. K., Kan, A., and Shmoys, D. B. (1992). *The Travelling Salesman Problem.* Wiley Interscience.

Lawler, E. L., Lenstra, J. K., Kan, A., and Shmoys, D. B. (1993). Sequencing and scheduling: algorithms and complexity. In Graves, S. c., Zipkin, P. H., and Kan, A. h. g. r. (Eds.), *Logistics of Production and Inventory: Handbooks in Operations Research and Management Science, Volume 4,* pp. 445–522. North-Holland, Amsterdam.

Lawler, E. L. and Wood, D. E. (1966). Branch-and-bound methods: A survey. *Operations Research, 14*(4), 699–719.

Lazanas, A. and Latombe, J.-C. (1992). Landmark-based robot navigaton. In *Preceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92),* pp. 816–822, San Jose. AAAI Press.

Le Cun, Y., Jackel, L., Boser, B., and Denker, J. (1989). Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine, 27*(11), 41–46.

LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., Denker, J., Drucker, H., Guyon, I., Muller, U., Sackinger, E., Simard, P., and Vapnik, V. N. (1995). Comparison of learning algorithms for handwritten digit recognition. In Fogelman, F. and Gallinari, P. (Eds.), *International Conference on Artificial Neural Networks,* pp. 53–60, Berlin. Springer-Verlag.

Leech, G., Rayson, P., and Wilson, A. (2001). *Word Frequencies in Written and Spoken English: Based on the British National Corpus.* Longman, New York.

Created by XMLmind XSL-FO Converter.

Lefkovitz, D. (1960). A strategic pattern recognition program for the game Go. Technical note 60–243, Wright Air Development Division, University of Pennsylvania, Moore School of Electrical Engineering.

Lenat, D. B. (1983). EURISKO: A program that learns new heuristics and domain concepts: The nature of heuristics, III: Program design and results. *Artificial Intelligence, 21*(1–2), 61–98.

Lenat, D. B. (1995). Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM, 38*(11).

Leant, D. B. and Brown, J. S. (1984). Why AM and EURISKO appear to work. *Artificial Intelligence, 23*(3), 269–294.

Leant, D. B. and Guha, R. V. (1990). *Building Large Knowledge-Based Systems: Representation and Inference in the CYC project.* Addison-Wesley, Reading, Massachusetts.

Leonard, H. S. and Goodman, N. (1940). The calculus of individuals and its uses. *Journal of Symbolic Logic, 5*(2), 45–55.

Leonard, J. J. and Durrant-Whyte, H. (1992). *Directed sonar sensing for mobile robot navigation.* Kluwer, Dordrecht, Netherlands.

Les′niewski, S. (1916). Podstawy ogólnej teorii mnogos′ci. Moscow.

Lettvin, J. Y., Maturana, H. R., McCulloch, W. S., and Pitts, W. H. (1959). What the frog's eye tells the frog's brain. *Proceedings of the IRE, 47*(11), 1940–1951.

Letz, r., Schumann, J., Bayerl, S., and Bibel, W. (1992). SETHEO: A high-performance theorem prover. *Journal of Automated Reasoning, 8*(2), 183–212.

Levesque, H. J. and Brachman, R. J. (1987). Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence, 3*(2), 78–93.

Levesque, H. J., Reiter, R., Lespérance, Y., Lin, F., and Scherl, R. (1997a). GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming, 31*, 59–84.

Levesque, H. J., Reiter, R., Lespérance, y., Lin, F., and Scherl, R. (1997b). GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming, 31*, 59–84.

Levitt, G. M. (2000). *The Turk, Chess Automation.* McFarland and Company.

Levy, D. N. L. (Ed.). (1988a). *Computer Chess Compendium.* Springer-Verlag, Berlin.

Levy, d. N. L. (Ed.). (1988b). *Computer Games.* Springer-Verlag, Berlin.

Lewis, D. D. (1998). Naive Bayes at forty: The independence assumption in information retrieval. In *Machine Learning: ECML-98. 10th European Conference on Machine Learning. Proceedings*, pp. 4–15, Chemnitz, Germany. Springer-Verlag.

Lewis, D. K. (1966). An argument for the identity theory. *the Journal of Philosophy, 63*(1), 17–25.

Lewis, D. K. (1972). General semantics. In Davidson, d. and Harman, G. (Eds.), *Semantics of Natural Language*, pp. 169–218. Reidel, Dordrecht, Netherlands.

Lewis, D. K. (1980). Mad pain and Martian pain. In Block, N. (Ed.), *Readings in Philosophy of Psychology*, Vol. 1, pp. 216–222. Harvard University Press, Cambridge, Massachusetts.

Li, C. M. and Anbulagan (1997). Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 366–371, Nagoya, Japan. Morgan Kaufmann.

Li, M. and Vitanyi, P. M. B. (1993). *An Introduction to Kolmogorov Complexity and Its Applications.* Springer-Verlag, Berlin.

Created by XMLmind XSL-FO Converter.

Lifschitz, V. (1986). On the semantics of STRIPS. In Georgeff, M. P. and Lansky, a. L. (Eds.), *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pp. 1–9, timberline, Oregon. Morgan Kaufmann.

Lifschitz, V. (2001). Answer set programming and plan generation. *Artificial Intelligence*, *138*(1–2), 39–54.

Lighthill, J. (1973). Artificial intelligence: A general survey. In Lighthill, J., Sutherland, N. s., Needham, r. M., Longuet-Higgins, H. c., and Michie, D. (Eds.), *Artificial Intelligence: A Paper Symposium.* Science Research Council of Great Britain, London.

Lin, F. and Reiter, R. (1997). How to progress a database. *Artificial Intelligence*, *92*(1–2), 131–167.

Lin, S. (1965). Computer solutions of the travelling salesman problem. *Bell Systems Technical Journal*, *44*(10), 2245–2269.

Lin, S. and Kernighan, b. W. (1973). An effective heuristic algorithm for the travelling-salesman problem. *Operations Research*, *21*(2), 498–516.

Linden, T. A. (1991). Representing software design as partially developed plans. In Lowry, M. R. and McCartney, R. D. (Eds.), *Automating Software Design*, pp. 603–625. MIT Press, Cambridge, Massachusetts.

Lindsay, r. K. (1963). Inferential memory as the basis of machines which understand natural language. In Feigenbaum, E. A. and Feldman, J. (Eds.), *Computers and Thought*, pp. 217–236. McGraw-Hill, New Jork.

Lindsay, R. K., Buchanan, B. G., Feigenbaum, E. A., and Lederberg, J. (1980). *Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project.* McGraw-Hill, New York.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning (ML-94)*, pp. 157–163, Mew Brunswick, NJ. Morgan Kaufmann.

Littman, M. L., Keim, g. A., and Shazeer, N. M. (1999). Solving crosswords with PROVERB. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pp. 914–915, Orlando, Florida. AAAI Press.

Liu, J. S. and Chen, R. (1998). Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association*, *93*, 1022–1031.

Lloyd, J. W. (1987). *Foundations of Logic Programming.* Springer-Verlag, Berlin.

Locke, J. (1690). *An Essay Concerning Human Understanding.* William Tegg.

Locke, W. N. and Booth, A. D. (1955). *Machine Translation of Languages: Fourteen Essays.* MIT Press, Cambridge, Massachusetts.

Lodge, D. (1984). *Small World.* Penguin Books, New York.

Lohn, J. D., Kraus, W. F., and Colombano, S. P. (2001). Evolutionary optimization of yagi-uda antennas. In *Proceedings of the Fourth International Conference on Evolvable Systems*, pp. 236–243.

Longuet-Higgins, H. C. (1981). A computer algorithm for reconstructing a scene from two projections. *Nature*, *293*, 133–135.

Lovejoy, W. S. (1991). A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, *28*(1–4), 47–66.

Loveland, D. (1968). Mechanical theorem proving by model elimination. *Journal of the Association for Computing Machinery*, *15*(2), 236–251.

Loveland, d. (1970). A linear format for resolution. In *Proceedings of the IRIA Symposium on Automatic Demonstration*, pp. 147–162, Berlin. Springer-Verlag.

Loveland, D. (1984). Automated theorem-proving: A quarter-century review. *Contemporary Mathematics*, *29*, 1–45.

Created by XMLmind XSL-FO Converter.

Lowe, D. g. (1987), Three-dimensional object recognition from single two-dimensional images. *Artificial Intelligence, 31,* 355–395.

Löwenheim, L. (1915). Über möglichkeiten im Relativkalkül. *Mathematische Annalen, 76,* 447– 470.

Lowerre, b. T. (1976). *The* Harpy *Speech Recognition System.* Ph.D. thesis, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

Lowerre, B. T. and Reddy, R. (1980). The harpy speech recognition system. In Lea, W. A. (Ed.), *Trends in Speech Recognition,* chap. 15. Prentice-Hall, Upper Saddle River, New Jersey.

Lowry, M. R. and McCartney, R. D. (1991). *Automating Software Design.* MIT Press, Cambridge, Massachusetts.

Loyd, D. (1959). *Mathematical Puzzles of Sam Loyd: Selected and Edited by Martin Gardner.* Dover, New York.

Lozano-Perez, T. (1983). Spatial planning: A configuration space approach. *IEEE Transactions on Computers, C-32*(2), 108–120.

Lozano-Perez, T., Mason, M., and Tylor, R. (1984). Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research, 3*(1), 3–24.

Luby, M., Sinclair, A., and Zuckerman, D. (1993). Optimal speedup of Las Vegas algorithms. *Information Processing Letters, 47,* 173–180.

Luby, M. and Vigoda, E. (1999). Fast convergence of the glauber dynamics for sampling independent sets. *Random Structures and Algorithms, 15*(3–4), 229–241.

Lucas, J. R. (1961). Minds, machines, and Gödel. *Philosophy, 36.*

Lucas, J. R. (1976). This Gödel is killing me: A rejoinder. *Philosophia, 6*(1), 145–148.

Lucas, P. (1996). Knowledge acquisition for decision-theoretic expert systems. *AISB Quarterly, 94,* 23–33.

Luce, D. R. and Raiffa, H. (1957). *Games and Decisions.* Wiley, New York.

Luger, G. F. (Ed.). (1995). *Computation and intelligence: Collected readings.* AAAI Press, Menlo Park, California.

MacKay, D. J. C. (1992). a practical Bayesian framework for back-propagation networks. *Neural Computation, 4*(3), 448–472.

Mackworth, A. K. (1973). Interpreting pictures of polyhedral scenes. *Artificial Intelligence, 4,* 121–137.

Mackworth, A. K. (1977). Consistency in networks of relations. *Artificial Intelligence, 8*(1), 99–118.

Mackworth, A. K. (1992). Constraint satisfaction. In Shapiro, S. (Ed.), *Encyclopedia of Artificial Intelligence* (second edition)., Vol. 1, pp. 285–293. Wiley, New York.

Mahanti, A. and Daniels, C. J. (1993). A SIMD approach to parallel heuristic search. *Artificial Intelligence, 60*(2), 243–282.

Majercik, S. M. and Littman, M. L. (1999). Planning under uncertainty via stochastic satisfiability. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence,* pp. 549–556.

Malik, J, (1987). Interpreting line drawings of curved objects. *International Journal of Computer Vision, 1*(1), 73–103.

Malik, J. and Rosenholtz, r. (1994). Recovering surface curvature and orientation from texture distortion: A least squares algorithm and sensitivity analysis. In Eklundh, J.-O. (Ed.), *Proceedings of the Third European Conf. on Computer Vision,* pp. 353–364, Stockholm. Springer-Verlag.

Created by XMLmind XSL-FO Converter.

Malik, J. and Rosenholtz, R. (1997). Computing local surface orientation and shape from texture for curved surfaces. *International Journal of Computer Vision, 23*(2), 149–168.

Mann, W. C. and Thompson, S. A. (1983). Relational propositions in discourse. Tech. rep. RR-83-115, Information Sciences Institute.

Mann, W. C. and Thompson, S. A. (1988). Rhetorical structure theory: Toward a functional theory of text organization. *Text, 8*(3), 243–281.

Manna, Z. and Waldinger, R. (1971). Toward automatic program synthesis. *Communications of the Association for Computing Machinery, 14*(3), 151–165.

Manna, Z. and Waldinger, R. (1985). *The Logical Basis for Computer Programming: Volume 1: Deductive Reasoning.* Addison-Wesley, Reading, Massachusets.

Manna, Z. and Waldinger, R. (1986). Special relations in automated deduction. *Journal of the Association for Computing Machinery, 33*(1), 1–59.

Manna, Z. and Waldinger, R. (1992). Fundamentals of deductive program synthesis. *IEEE Transactions on Software Engineering, 18*(8), 674– 704.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing.* MIT Press.

Marbach, P. and Tsitsiklis, J. N. (1998). Simulation-based optimization of Markov reward processes. Technical report LIDS-P-2411, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of english: The penn treebank. *Computational Linguistics, 19*(2), 313–330.

Markov, A. A. (1913). An example of statistical investigation in the text of „Eugene Onegin" illustrating coupling of „tests" in chains. *Proceedings of the Academy of Sciences of St. Petersburg, 7.*

Maron, M. E. (1961). Automatic indexing: An experimental inquiry. *Journal of the Association for Computing Machinery,* 8(3), 404–417.

Maron, M. E. and Kuhns, J.-L. (1960). On relevance, probabilistic indexing and information retrieval. *Communications of the ACM, 7,* 219–244.

Marr, D. (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information.* W. H. Freeman, New York.

Marriott, K. and Stuckey, P. J. (1998). *Programming with Constraints: An Introduction.* MIT Press, Cambridge, Massachusetts.

Marsland, a. T. and Schaeffer, J. (Eds.). (1990). *Computers, Chess, and Cognition.* Springer-Verlag, Berlin.

Martelli, A. and Montanari, U. (1976). Unification in linear time and space: A structured presentation. Internal report B 76-16, Istituto di Elaborazione della Informazione, Pisa, Italy.

Martelli, A. and Montanari, U. (1978). Optimizing decision trees through heuristically guided search. *Communications of the Association for Computing Machinery, 21,* 1025–1039.

Marthi, B., Pasula, H., Russell, S. J., and Peres, Y. (2002). Decayed MCMC filtering. In *Uncertainty in Artificial Intelligence: Proceedings of the Eighteenth Conference,* pp. 319–326, Edmonton, Alberta. Morgan Kaufmann.

Martin, J. H. (1990). *A Computational Model of Metaphor Interpretation.* Academic Press, New York.

Martin, P. and Shmoys, D. B. (1996). A new approach to computing optimal schedules for the jobshop scheduling problem. In *Proceedings of the 5th International IPCO Conference,* pp. 389–403. Springer-Verlag.

Maslov, S. Y. (1964). An inverse method for establishing deducibility in classical predicate calculus. *Doklady Akademii nauk SSSR, 159,* 17–20.

Created by XMLmind XSL-FO Converter.

Maslov, S. Y. (1967). An inverse method for establishing deducibility of nonprenex formulas of the predicate calculus. *Doklady Akademii nauk SSSR, 172*, 22–25.

Mason, M. (1993). Kicking the sensing habit. *AI Magazine, 14*(1), 58–59.

Mason, M. (2001). *Mechanics of Robotic Manipulation.* MIT Press.

Mason, M. and Salisbury, J. (1985). *Robot hands and the mechanics of manipulation.* MIT Press.

Mataric, M. J. (1997). Reinforcement learning in the multi-robot domain. *Autonomous Robots, 4*(1), 73–83.

Mates, B. (1953). *Stoic Logic*. University of California Press, Berkeley and Los Angeles.

Maxwell, J. and Kaplan, r. (1993). The interface between phrasal and functional constraints. *Computational Linguistics, 19*(4), 571–590.

Maxwell, J. and Kaplan, R. (1995). A method for disjunctive constraint satisfaction. In Dalrymple, M., Kaplan, r., Maxwell, J., and Zaenen, A. (Eds.), *Formal Issues in Lexical-Functional Grammar*, No. 47 in CSLI Lecture Note Series, chap. 14, pp. 381–481. CSLI Publications.

McAllester, D. A. (1980). An outlook on truth maintenance. Ai memo 551, MIT AI Laboratory, Cambridge, Massachusetts.

McAllester, D. A. (1988). Conspiracy numbers for min-max search. *Artificial Intelligence, 35*(3), 287–310.

McAllester, D. A. (1989). *Ontic: A Knowledge Representation System for Mathematics.* MIT Press, Cambridge, Massachusetts.

McAllester, D. A. (1998). What is the most pressing issue facing ai and the aaai today?. Candidate statement, election for Councilor of the American Association for Artificial Intelligence.

McAllester, D. A. and Givan, R. (1992). Natural language syntax and first-order inference. *Artificial Intelligence, 56*(1), 1–20.

McAllester, D. A. and Rosenblitt, D. (1991). Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, Vol. 2, pp. 634–639, Anaheim, California. AAAI Press.

McCarthy, J. (1958). Programs with common sense. In *Proceedings of the Symposium on Mechanisation of Thought Processes*, Vol. 1, pp. 77–84, London. Her Majesty's Stationery Office.

McCarthy, J. (1963). Situations, actions, and causal laws. Memo 2, Stanford University Artificial Intelligence Project, Stanford, California.

McCarthy, J. (1968). Programs with common sense. In Minsky, M. L. (Ed.), *Semantic Information Processing*, pp. 403–418. MIT Press, Cambridge, Massachusetts.

McCarthy, J. (1980). Circumscription: A form of non-monotonic reasoning. *Artificial Intelligence, 13*(1–2), 27–39.

McCarthy, J. and Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., Michie, D., and Swann, M. (Eds.), *Machine Intelligence 4*, pp. 463–502. Edinburgh University Press, Edinburgh, Scotland.

McCarthy, J., Minsky, M. L., Rochester, N., and Shannon, c. E. (1955). Proposal for the Dartmouth summer research project on artificial intelligence. Tech. rep., Dartmouth College.

McCawley, J. D. (1988). *The Syntactic Phenomena of English*, Vol. 2 volumes. University of Chicago Press.

McCawley, J. D. (1993). *Everything That Linguists Have Always Wanted to Know about Logic but Were Ashamed to Ask* (Second edition). University of Chicago Press, Chicago.

Created by XMLmind XSL-FO Converter.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, 5*, 115–137.

McCune, W. (1992). Automated discovery of new axiomatizations of the left group and right group calculi. *Journal of Automated Reasoning, 9*(1), 1–24.

McCune, W. (1997). Solution of the robbins problem. *Journal of Automated Reasoning, 19*(3), 263–276.

McDermott, D. (1976). Artificial intelligence meets natural stupidity. *SIGART Newsletter, 57*, 4–9.

McDermott, D. (1978a). Planning and acting. *Cognitive Science, 2*(2), 71–109.

McDermott, D. (1978b). Tarskian semantics, or, no notation without denotation!. *Cognitive Science, 2*(3).

McDermott, D. (1987). A critique of pure reason. *Computational Intelligence, 3*(3), 151–237.

McDermott, D. (1996). A heuristic estimator for means-ends analysis in planning. In *Proceedings of the Third International Conference on AI Planning Systems*, pp. 142–149, Edinburgh, Scotland. AAAI Press.

McDermott, D. and Doyle, J. (1980). Non-monotonic logic: i. *Artificial Intelligence, 13*(1–2), 41–72.

McDermott, J. (1982). R1: A rule-based configurer of computer systems. *Artificial Intelligence, 19*(1), 39–88.

McEliece, R. J., MacKay, D. J. C., and Cheng, J.-F. (1998). Turbo decoding as an instance of Pearl's „belief propagation" algorithm. *IEEE Journal on Selected Areas in Communications, 16*(2), 140– 152.

McGregor, J. J. (1979). Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Sciences, 19*(3), 229–250.

McKeown, K. (1985). *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text.* Cambridge University Press, Cambridge, UK.

McLachlan, g. J. and Krishnan, t. (1997). *The EM Algorithm and Extensions.* Wiley, New York.

McMillan, K. L. (1993). *Symbolic Model Checking.* Kluwer, Dordrecht, Netherlands.

Meehl, P. (1955). *Clinical vs. Statistical Prediction.* University of Minnesota Press, Minneapolis.

Melc′uk, I. A. and Polguere, A. (1988). A formal lexicon in the meaning-text theory (or how to do lexica with words). *Computational Linguistics, 13*(3–4), 261–275.

Mendel, G. (1866). Versuche über pflanzen-hybriden. *Verhandlungen des Naturforschenden Vereins, Abhandlungen, Brünn, 4,* 3–47. Translated into English by C. T. Druery, published by Bateson (1902).

Mercer, J. (1909). Functions of positive and negative type and their connection with the theory of integral equations. *Philos. Trans. Roy. Soc. London, A, 209*, 415–446.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953). Equations of state calculations by fast computing machines. *Journal of Chemical Physics, 21*, 1087–1091.

Mézard, M. and Nadal, J.-P. (1989). Learning in feed-forward layered networks: The tiling algorithm. *Journal of Physics, 22*, 2191–2204.

Michalski, R. S. (1969). On the quasi-minimal solution of the general covering problem. In *Proceedings of the First International Symposium on Information Processing*, pp. 125–128.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (Eds.). (1983). *Machine Learning: An Artificial Intelligence Approach*, Vol. 1. Morgan Kaufmann, San Mateo, California.

Michalski, r. S., Carbonell, J. G., and Mitchell, T. M. (Eds.). (1986a). *Machine Learning: An Artificial Intelligence Approach*, Vol. 2. Morgan Kaufmann, San Mateo, California.

Created by XMLmind XSL-FO Converter.

Michalski, r. S., Mozetic, I., Hong, J., and Lavrac, N. (1986b). The multi-purpose incremental learning system aq15 and its testing application to three medical domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pp. 1041–1045, Philadelphia. Morgan Kaufmann.

Michel, S. and Plamondon, P. (1996). Bilingual sentence alignment: Balancing robustnes and accuracy. In *Proceedings of the Conference of the Association for Machine Translation in the Americas (AMTA).*

Michie, d. (1966). Game-playing and game-learning automata. In Fox, L. (Ed.), *Advances in Programming and Non-Numerical Computation*, pp. 183– 200. Pergamon, Oxford, UK.

Michie, D. (1972). Machine intelligence at Edinburgh. *Management Informatics, 2*(1), 7–12.

Michie, D. (1974). Machine intelligence at Edinburgh. In *On Intelligence*, pp. 143–155. Edinburgh University Press.

Michie, d. and Chambers, r. A. (1968). BOXES: An experiment in adaptive control. In Dale, E. and Michie, D. (Eds.), *Machine Intelligence 2*, pp. 125–133. Elsevier/North-Holland, Amsterdam, London, New York.

Michie, d., Spiegelhalter, D. J., and Taylor, C. (Eds.). (1994). *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, Chichester, England.

Milgrom, P. (1997). Putting auction theory to work: The simultaneous ascending auction. Tech. rep. Technical Report 98–0002, Stanford University Department of Economics.

Mill, J. S. (1843). *A System of Logic, Ratiocinative and Inductive: Being a Connected View of the Principles of Evidence, and Methods of Scientific Investigation.* J. W. Parker, London.

Mill, J. S. (1863). *Utilitarianism.* Parker, Son and Bourn, London.

Miller, A. C., Merkhofer, M. M., Howard, R. A., Matheson, J. E., and Rice, T. R. (1976). Development of automated aids for decision analysis. Technical report, SRI International, Menlo Park, California.

Minsky, M. L. (Ed.). (1968). *Semantic Information Processing.* MIT Press, Cambridge, Massachusetts.

Minsky, M. L. (1975). A framework for representing knowledge. In Winston, P. H. (Ed.), *The Psychology of Computer Vision*, pp. 211–277. McGraw-Hill, New York. Originally an MIT AI Laboratory memo; the 1975 version is abridged, but is the most widely cited.

Minsky, M. L. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry* (first edition). MIT Press, Cambridge, Massachusetts.

Minsky, M. L. and Papert, S. (1988). *Perceptrons: An Introduction to Computational Geometry* (Expanded edition). MIT Press, Cambridge, Massachusetts.

Minton, S. (1984). Constraint-based generalization: Learning game-playing plans from single expamples. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-84)*, pp. 251–254, Austin, Texas. Morgan Kaufmann.

Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pp. 564–569, St. Paul, Minnesota. Morgan Kaufmann.

Minton, S., Johnston, M. D., Philips, A. B., and Laird, P. (1992). Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence, 58*(1–3), 161–205.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, Massachusetts.

Mitchell, M., Holland, J. H., and Forrest, S. (1996). When will a genetic algorithm outperform hill climbing?. In Cowan, J., Tesauro, G., and Alspector, J. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 6. MIT Press, Cambridge, Massachusetts.

Created by XMLmind XSL-FO Converter.

Mitchell, T. M. (1977). Version spaces: A candidate elimination approach to rule learning. In *Proceedings of the Fifth Intgernational Joint Conference on Artificial Intelligence (IJCAI-77)*, pp. 305–310, Cambridge, Massachusetts. IJCAII.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence, 18*(2), 203–226.

Mitchell, t. M. (1990). Becoming increasingly reactive (mobile robots). In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, Vol. 2, pp. 1051–1058, Boston. MIT Press.

Mitchell, T. M. (1997). *Machine Learning.* McGraw-Hill, New York.

Mitchell, t. M., Keller, R., and Kedar-Cabelli, S. (1986). Explanation-based generalization: A unifying view. *Machine Learning, 1*, 47–80.

Mitchell, T. M., Utgoff, P. E., and Banerji, r. (1983). Learning by experimentation: Acquiring and refining problem-solving heuristics. In Michalski, r. s., Carbonell, J. G., and Mitchell, T. M. (Eds.), *Machine Learning: An Artificial Intelligence Approach*, pp. 163–190. Morgan Kaufmann, San Mateo, California.

Mitkov, R. (2002). *Anaphora Resolution.* Longman, New York.

Mohr, R. and Henderson, T. C. (1986). Arc and path consistency revisited. *Artificial Intelligence, 28*(2), 225–233.

Mohri, M., Pereira, F., and Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech and Language, 16*(1), 69–88.

Montague, P. r., Dayan, P., Person, C., and Sejnowsky, T. (1995). Bee foraging in uncertain environments using predictive Hebbian learning. *Nature, 377*, 725–728.

Montague, R. (1970). English as a formal language. In *Linguaggi nella Società e nella Tecnica,* pp. 189–224. Edizioni di Comunità, Milan.

Montague, r. (1973). The proper treatment of quantification in ordinary English. In Hintikka, K. J. J., Moravcsik, J. M. E., and Suppes, P. (Eds.), *Approaches to Natural Language.* D. Reidel, Dordrecht, Netherlands.

Montanari, U. (1974). Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences, 7*(2), 95–132.

Montemerlo, M. Thrun, S., Koller, D., and Wegbreit, B. (2002). FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Preceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, Edmonton, Alberta. AAAI Press.

Mooney, R. (1999). Learning for semantic interpretation: Scaling up without dumbing down. In Cussens, J. (Ed.), *Proceedings of the 1st Workshop on Learning Language in Logic*, pp. 7–15. Springer-Verlag.

Mooney, R. J. and Califf, M. E. (1995). Induction of first-order decision lists: Results on learning the past tense of English verbs. *Journal of AI Research, 3*, 1–24.

Moore, A. W. and Atkeson, C. G. (1993). Prioritized sweeping—reinforcement learning with less data and less time. *Machine Learning, 13*, 103–130.

Moore, E. F. (1959). The shortest path through a maze. In *Proceedings of an International Symposium on the Theory orf Switching, Part II*, pp. 285–292. Harvard University Press, Cambridge, Massachusetts.

Moore, J. S. and Newell, A. (1973). How can Merlin understand?. In Gregg, L. (Ed.), *Knowledge and Cognition.* Lawrence Erlbaum Associates, Potomac, Maryland.

Moore, R. C. (1980). Reasoning about knowledge and action. Artificial intelligence center technical note 191, SRI Inernational, Menlo Park, California.

Moore, R. C. (1985). A formal theory of knowledge and action. In Hobbs, J. R. and Moore, R. C. (Eds.), *Formal Theories of the Commonsense World*, pp. 319–358. Ablex, Norwood, New Jersey.

Created by XMLmind XSL-FO Converter.

Moravec, H. P. (1983). The stanford cart and the cmu rover. *Proceedings of the IEEE, 71*(7), 872–884.

Moravec, H., P. and Elfes, A. (1985). High resolution maps from wide angle sonar. In *1985 IEEE International Conference on Robotics and Automation*, pp. 116–121, St. Louis, Missouri. IEEE Computer Society Press.

Moravec, H. P. (1988). *Mind Children: The Future of Robot and Human Intelligence.* Harvard Unviersity Press, Cambridge, Massachusetts.

Moravec, H. P. (2000). *Robot: Mere Machine to Transcendent Mind.* Oxford University Press.

Morgenstern, L. (1987). Knowledge preconditions for actions and plans. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pp. 867–874, Milan. Morgan Kaufmann.

Morgenstern, L. (1998). Inheritance comes of age: Applying nonmonotonic techniques to problems in industry. *Artificial Intelligence, 103*, 237–271.

Morjaria, M. A., Rink, F. J., Smith, W. D., Klempner, G., Burns, C., and Stein, J. (1995). Elicitation of probabilities for belief networks: Combining qualitative and quantitative information. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 141–148. Morgan Kaufmann.

Morrison, P. and Morrison, E. (Eds.). (1961). *Charles Babbage and His Calculating Engines: Selected Writings by Charles Babbage and Others.* Dover, New York.

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. (2001). Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC 2001)*, pp. 530–535. ACM Press.

Mosteller, F. and Wallace, D. L. (1964). *Inference and Disputed Authorship: The Federalist.* Addison-Wesley.

Mostow, J. and Prieditis, A. E. (1989). Discovering admissible heuristics by abstracting and optimizing: A transformational approach. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, Vol. 1, pp. 701–707, Detroit. Morgan Kaufmann.

Motzkin, T. S. and Schoenberg, I. J. (1954). The relaxation method for linear inequalities. *Canadian Journal of Mathematics, 6*(3), 393–404.

Moussouris, J., Holloway, J., and Greenblatt, R. D. (1979). CHEOPS: A chess-oriented processing system. In Hayes, J. E., Michie, D., and Mikulich, L. I. (Eds.), *Machine Intelligence 9*, pp. 351–360. Ellis Horwood, Chichester, England.

Moutarlier, P. and Chatila, R. (1989). Stochastic multisensory data fusion for mobile robot location and environment modeling. In *5th Int. Symposium on Robotics Research*, Tokyo.

Muggleton, S. H. (1991). Inductive logic programming. *New Generation Computing, 8*, 295–318.

Muggleton, s. H. (1992). *Inductive Logic Programming.* Academic Press, New York.

Muggleton, S. H. (1995). Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming, 13*(3–4), 245–286.

Muggleton, S. H. (2000). Learning stochastic logic programs. Proceedings of the AAAI 2000 Workshop on Learning Statistical Models from Relational Data.

Muggleton, S. H. and Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pp. 339–352. Morgan Kaufmann.

Muggleton, S. H. and De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming, 19/20*, 629–679.

Muggleton, s. H. and Feng, C. (1990). Efficient induction of logic programs. In *Proceedings of the Workshop on Algorithmic Learning Theory*, pp. 368–381, Tokyo. Ohmsha.

Müller, M. (2002). Computer Go. *Artificial Intelligence, 134*(1–2), 145–179.

Created by XMLmind XSL-FO Converter.

Mundy, J. and Zisserman, A. (Eds.). (1992). *Geometric Invariance in Computer Vision.* MIT Press, Cambridge, Massachusetts.

Murphy, K., Weiss, Y., and Jordan, M. I. (1999). Loopy belief propagation for approximate inference: An empirical study. In *Uncertainty in Artificial Intelligence: Proceedings of the Fifteenth Conference*, pp. 467–475, Stockholm. Morgan Kaufmann.

Murphy, K. and Russell, S. J. (2001). Raoblackwellised particle filtering for dynamic bayesian networks. In Doucet, A., de Freitas, N., and Gordon, N. J. (Eds.), *Sequential Monte Carlo Methods in Practice.* Springer-Verlag, Berlin.

Murphy, R. (2000). *Introduction to AI Robotics.* MIT Press, Cambridge, Massachusetts.

Muscettola, N., Nayak, P., Pell, B., and Williams, B. (1998). Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence, 103,* 5–48.

Myerson, r. B. (1991). *Game Theory: Analysis of Conflict.* Harvard University Press, Cambridge.

Nagel, t. (1974). What is it like to be a bat?. *Philosophical Review, 83,* 435–450.

Nalwa, V. S. (1993). *A Guided Tour of Computer Vision.* Addison-Wesley, Reading, Massachusetts.

Nash, J. (1950). Equilibrium points in N-person games. *Proceedings of the National Academy of Sciences of the United States of America, 36,* 48–49.

Nau, D. S. (1980). Pathology on game trees: A summary of results. In *Proceedings of the First Annual National Conference on Artificial Intelligence (AAAI-80),* pp. 102–104, Stanford, California. AAAI.

Nau, d. S. (1983). Pathology on game trees revisited, and an alternative to minimaxing. *Artificial Intelligence, 21*(1–2), 221–244.

Nau, D. S., Kumar, V., and Kanal, L. N. (1984). General branch and bound, and its relation to A* and AO*. *Artificial Intelligence, 23,* 29–58.

Naur, P. (1963). Revised report on the algorithmic language Algol 60. *Communications of the Association for Computing Machinery, 6* (1), 1–17.

Nayak, P. and Williams, B. (1997). Fast context switching in real-time propositional reasoning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97),* pp. 50–56, Providence, Rhode Island. AAAI Press.

Neal, R. (1996). *Bayesian Learning for Neural Networks.* Springer-Verlag, Berlin.

Nebel, B. (2000). On the compilability and expressive power of propositional planning formalisms. *Journal of AI Research, 12,* 271–315.

Nelson, G. and Oppen, D. C. (1979). Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems, 1*(2), 245–257.

Netto, E. (1901). *Lehrbuch der Combinatorik.* b. G. Teubner, Leipzig.

Nevill-Manning, C. G. and Witten, I. H. (1997). Identifying hierarchical structures in sequences: A linear-time algorithm. *Journal of AI Research, 7,* 67–82.

Nevins, A. J. (1975). Plane geometry theorem proving using forward chaining. *Artificial Intelligence, 6*(1), 1–23.

Newell, a. (1982). The knowledge level. *Artificial Intelligence, 18*(1), 82–127.

Newell, A. (1990). *Unified Theories of Cognition.* Harvard University Press, Cambridge, Massachusetts.

Newell, A. and Ernst, G. (1965). The search for generality. In Kalenich, W. A. (Ed.), *Information Processing 1965: Proceedings of IFIP Congress 1965,* Vol. 1, pp. 17–24, Chicago. Spartan.

Created by XMLmind XSL-FO Converter.

Newell, A., Shaw, J. C., and Simon, H. A. (1957). Empirical explorations with the logic theory machine. *Proceedings of the Western Joint Computer Conference, 15,* 218–239. Reprinted in Feigenbaum and Feldman (1963).

Newell, A., Shaw, J. C., and Simon, H. A. (1958). Chess playing programs and the problem of complexity. *IBM Journal of Research and Development, 4*(2), 320–335.

Newell, A. and Simon, H. a. (1961). GPS, a program that simulates human thought. In Billing, H. (Ed.), *Lernende Automaten,* pp. 109–124. R. Oldenbourg, Munich.

Newell, A. and Simon, H. A. (1972). *Human Problem Solving.* Prentice-Hall, Upper Saddle River, New Jersey.

Newell, A. and Simon, H. A. (1976). Computer science as empirical inquiry: Symbols and search. *Communications of the Association for Computing Machinery, 19,* 113–126.

Newton, I. (1664–1671). Methodus fluxionum et serierum infinitarum. Unpublished notes.

Ng, A. Y., Harada, D., and Russell, S. J. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning,* Bled, Slovenia. Morgan Kaufmann.

Ng, A. Y. and Jordan, M. I. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *Uncertainty in Artificial Intelligence: proceedings of the Sixteenth Conference,* pp. 406–415, Stanford, California. Morgan Kaufmann.

Nguyen, X. and Kambhampati, S. (2001). Reviving partial order planning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01),* pp. 459–466, Seattle. Morgan Kaufmann.

Nguyen, X. Kambhampati, S., and Nigenda, R. S. (2001). Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. Tech. rep., Computer Science and Engineering Department, Arizona State University.

Nicholson, A. and Brady, J. M. (1992). The data association problem when monitoring robot vehicles using dynamic belief networks. In *ECAI 92: 10th European Conference on Artificial Intelligence Proceedings,* pp. 689–693, Vienna, Austria. Wiley.

Niemelä, I., Simons, P., and Syrjänen, T. (2000). Smodels: A system for answer set programming. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning.*

Nilsson, D. and Lauritzen, S. (2000). Evaluating influence diagrams using LIMIDs. In *Uncertainty in Artificial Intelligence: Proceedings of the Sixteenth Conference,* pp. 436–445, Stanford, California. Morgan Kaufmann.

Nilsson, N. J. (1965). *Learning Machines: Foundations of Trainable Pattern-Classifying Systems.* McGraw-Hill, New York. republished in 1990.

Nilsson, N. J. (1971). *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill, New York.

Nilsson, N. J. (1980). *Principles of Artificial Intelligence.* Morgan Kaufmann, San Mateo, California.

Nilsson, N. J. (1984). Shakey the robot. Technical note 323, SRI International, Menlo Park, California.

Nilsson, N. J. (1986). Probabilistic logic. *Artificial Intelligence, 28*(1), 71–87.

Nilsson, N. J. (1991). Logic and artificial intelligence. *Artificial Intelligence, 47*(1–3), 31–56.

Nilsson, N. J. (1998). *Artificial Intelligence: A New Synthesis.* Morgan Kaufmann, San Mateo, California.

Norvig, P. (1988). Multiple simultaneous interpretations of ambiguous sentences. In *Proceedings of the 10th Annual Conference of the Cognitive Science Society.*

Norvig, P. (1992). *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp.* Morgan Kaufmann, San Mateo, California.

Created by XMLmind XSL-FO Converter.

Nowick, S. M., Dean, M. E., Dill, D. L., and Horowitz, M. (1993). The design of a high-performance cache controller: A case study in asynchronous synthesis. *Integration: The VLSI Journal, 15*(3), 241–262.

Nunberg, g. (1979). The non-uniqueness of semantic solutions: Polysemy. *Language and Philosophy, 3*(2) 143–184.

Nussbaum, M. C. (1978). *Aristotle's* De Motu Animalium. Princeton University Press, Princeton, New Jersey.

Ogawa, S., Lee, t.-M., Kay, A. R., and Tank, D. W. (1990). Brain magnetic resonance imaging with contrast dependent on blood oxygenation. *Proceedings of the National Academy of Sciences of the United States of America, 87,* 9868–9872.

Olawsky, D. and Gini, M. (1990). Deferred planning and sensor use. In Sycara, K. P. (Ed.), *Proceedings, DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control,* San Diego, California. Defense Advanced Research Projects Agency (DARPA), Morgan Kaufmann.

Olesen, K. G. (1993). Causal probabilistic networks with both discrete and continuous variables. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 15*(3), 275–279.

Oliver, r. M. and Smith, J. Q. (Eds.). (1990). *Influence Diagrams, Belief Nets and Decision Analysis.* Wiley, New York.

Olson, C. F. (1994). Time and space efficient pose clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 251–258, Washington, DC. IEEE Computer Society Press.

Oncina, J. and Garcia, P. (1992). Inferring regular languages in polynomial update time. In Perez, Sanfeliu, and Vidal (Eds.), *Pattern Recognition and Image Analysis*, pp. 49–61. World Scientific.

O'Reilly, U.-M. and Oppacher, f. (1994). Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing. In Davidor, Y., Schwefel, H.-P., and Manner, R. (Eds.), *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, pp. 397–406, Jerusalem, Israel. Springer-Verlag.

Ormoneit, D. and Sen, S. (2002). Kernel-based reinforcement learning. *Machine Learning, 49*(2–3), 161–178.

Ortony, A. (Ed.). (1979). *Metaphor and Thought.* Cambridge University Press, Cambridge, UK.

Osborne, M. J. and Rubinstein, A. (1994). *A Course in Game Theory.* MIT Press, Cambridge, Massachusetts.

Osherson, D. N., Stob, M., and Weinstein, S. (1986). *Systems That Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists.* MIT Press, Cambridge, Massachusetts.

Page, C. D. and Srinivasan, A. (2002). ILP: A short look back and a longer look forward. Submitted to Journal of Machine Learning Research.

Pak, I. (2001). On mixing of certain random walks, cutoff phenomenon and sharp threshold of random matroid processes. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science, 110,* 251–272.

Palay, A. J. (1985). *Searching with Probabilities.* Pitman, London.

Palmer, D. A. and Hearst, M. A. (1994). Adaptive sentence boundary disambiguation. In *Proceedings of the Conference on Applied Natural Language Processing*, pp. 78–83. Morgan Kaufmann.

Palmer, S. (1999). *Vision Science: Photons to Phenomenology.* MIT Press, Cambridge, Massachusetts.

Papadimitriou, c. H. (1994). *Computational Complexity.* Addison-Wesley.

Papadimitriou, C. H., Tamaki, H., Raghavan, P., and Vempala, s. (1998). Latent semantic indexing: A probabilistic analysis. In *Proceedings of the ACM Conference on Principles of Database Systems (PODS),* pp. 159–168, New York. ACM Press.

Created by XMLmind XSL-FO Converter.

Papadimitriou, C. H. and Tsitsiklis, J. N. (1987). The complexity of markov decision processes. *Mathematics of Operations Research, 12*(3), 441– 450.

Papadimitriou, C. H. and Yannakakis, M. (1991). Shortest paths without a map. *Theoretical Computer Science, 84*(1), 127–150.

Papavassiliou, V. and Russell, S. J. (1999). Convergence of reinforcement learning with general function approximators. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pp. 748–757, Stockholm. Morgan Kaufmann.

Parekh, R. and Honavar, V. (2001). Dfa learning from simple examples. *Machine Learning, 44*, 9–35.

Parisi, g. (1988). *Statistical field theory.* Addison-Wesley, Reading, Massachusetts.

Parker, D. B. (1985). Learning logic. Technical report TR-47, Center for Computational Research in Economics and Management Science, Massachusetts Institute of Technology, Cambridge, Masachusetts.

Parker, L. E. (1996). On the design of behavior-based multi-robot teams. *Journal of Advanced Robotics, 10*(6).

Parr, R. and Russell, S. J. (1998). Reinforcement learning with hierarchies of machines. In Jordan, M. I., Kearns, M., and Solla, S. A. (Eds.), *Advances in Neural Information Processing Systems 10.* MIT Press, Cambridge, Massachusetts.

Parzen, E. (1962). On estimation of a probability density function and mode. *Annals of Mathematical Statistics, 33*, 1065–1076.

Pasula, H. and Russell, S. J. (2001). Approximate inference for first-order probabilistic languages. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, Seattle. Morgan Kaufmann.

Pasula, H., Russell, S. J., Ostland, M., and Ritov, Y. (1999). Tracking many objects with many sensors. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, Stockholm. Morgan Kaufmann.

Paterson, M. S. and Wegman, M. N. (1978). Linear unification. *Journal of Computer and System Sciences, 16*, 158–167.

Patrick, B. G., Almulla, M., and Newborn, M. M. (1992). An upper bound on the time complexity of iterative-deepening-A*. *Annals of Mathematics and Artificial Intelligence, 5*(2–4), 265–278.

Patten, T. (1988). *Systemic Text Generation as Problem Solving.* Studies in Natural Language Processing. Cambridge University Press, Cambridge, UK.

Paul, R. P. (1981). *Robot Manipulators: Mathematics, Programming, and Control.* MIT Press, Cambridge, Massachusetts.

Peano, G. (1889). *Arithmetices principia, nova methodo exposita.* Fratres Bocca, Turin.

Pearl, J. (1982a). Reverend Bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-82)*, pp. 133–136, Pittsburgh, Pennsylvania. Morgan Kaufmann.

Pearl, J. (1982b). The solution for the branching factor of the alpha–beta pruning algorithm and its optimality. *Communications of the Association for Computing Machinery, 25*8), 559–564.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading, Massachusetts.

Pearl, J. (1986). Fusion, propagation, and structuring in belief networks. *Artificial Intelligence, 29*, 241–288.

Pearl, J. (1987). Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence, 32*, 247–257.

Created by XMLmind XSL-FO Converter.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, California.

Pearl, J. (2000). *Causality: Models, Reasoning, and Inference.* Cambridge University Press, Cambridge, UK.

Pearl, J. and Verma, T. (1991). A theory of inferred causation. In Allen, J. A., Fikes, R., and Sandewall, E. (Eds.), *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pp. 441–452, San Mateo, California. Morgan Kaufmann.

Pearson, J. and Jeavons, P. (1997). A survey of tractable constraint satisfaction problems. Technical report CSD-TR-97-15, Royal Holloway College, U. of London.

Pednault, E. P. D. (1986). Formulating multiagent, dynamic-world problems in the classical planning framework. In Georgeff, M. P. and Lansky, A. L. (Eds.), *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pp. 47–82, Timberline, Oregon. Morgan Kaufmann.

Peirce, C. S. (1870). Description of a notation for the logic of relatives, resulting from an amplification of the conceptions of Boole's calculus of logic. *Memoirs of the American Academy of Arts and Sciences, 9*, 317–378.

Peirce, C. S. (1883). A theory of probable inference. Note B. The logic of relatives. In *Studies in Logic by Members of the Johns Hopkins University*, pp. 187–203, Boston.

Peirce, C. S. (1902). Logic as semiotic: The theory of signs. Unpublished manuscript; reprinted in (Buchler 1955).

Peirce, C. S. (1909). Existential graphs. Unpublished manuscript; reprinted in (Bucler 1955).

Pelikan, M., Goldberg, D. E., and Cantu-Paz, E. (1999). BOA: The Bayesian optimization algorithm. In *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 525–532, Orlando, Florida. Morgan Kaufmann.

Pemberton, J. C. and Korf, R. E. (1992). Incremental planning on graphs with cycles. In Hendler, J. (Ed.), *Artificial Intelligence Planning Systems: Proceedings of the First International Conference*, pp. 525– 532, College Park, Maryland. Morgan Kaufmann.

Penberthy, J. S. and Weld, D. S. (1922). UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of KR-92*, pp. 103–114. Morgan Kaufmann.

Peng, J. and Williams, R. J. (1993). Efficient learning and planning within the Dyna framework. *Adaptive Behavior, 2*, 437–454.

Penrose, R. (1989). *The Emperor's New Mind.* Oxford University Press, Oxford, UK.

Penrose, R. (1994). *Shadows of the Mind.* Oxford University Press, Oxford, UK.

Peot, M. and Smith, D. E. (1992). Conditional nonlinear planning. In Hendler, J. (Ed.), *Proceedings of the First International Conference on AI Planning Systems*, pp. 189–197, College Park, Maryland. Morgan Kaufmann.

Pereira, F. and Shieber, S. M. (1987). *Prolog and Natural-Language Analysis.* Center for the Study of Language and Information (CSLI), Stanford, California.

Pereira, f. and Warren, D. H. D. (1980). Definite clause grammars for language analysis: A survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence, 13*, 231–278.

Peterson, C. and Anderson, J. R. (1987). A mean field theory learning algorithm for neural networks. *Complex Systems, 1*(5) 995–1019.

Pfeffer, A. (2000). *Probabilistic Reasoning for Complex Systems.* Ph.D. thesis, Stanford University, Stanford, California.

Pinker, S. (1989). *Learnability and Cognition.* MIT Press, Cambridge, MA.

Created by XMLmind XSL-FO Converter.

Pinker, s. (1995). Language acquisition. In Gleitman, L. R., Liberman, M., and Osherson, D. N. (Eds.), *An Invitation to Cognitive Science* (second edition)., Vol. 1. MIT Press, Cambridge Massachusetts.

Pinker, S. (2000). *The Language Instinct: How the Mind Creates Language.* MIT Press, Cambridge, Massachusetts.

Plaat, A., Schaeffer, J., Pijls, W., and de Bruin, A. (1996). Best-first fixed-depth minimax algorithms. *Artificial Intelligence Journal, 87* (1–2), 255–293.

Place, U. T. (1956). Is consciousness a brain process?. *British Journal of Psychology, 47,* 44–50.

Plotkin, g. (1971). *Automatic Methods of Inductive Inference.* Ph.D. thesis, Edinburgh University.

Plotkin, G. (1972). Building-in equational theories. In Meltzer, B. and Michie, D. (Eds.), *Machine Intelligence 7,* pp. 73–90. Edinburgh University Press, Edinburgh, Scotland.

Pnueli, A. (1977). The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77),* pp. 46–57, Providence, Rhode Island. IEEE, IEEE Computer Society Press.

Pohl, I. (1969). Bi-directional and heuristic search in path problems. Tech. rep. 104, SLAC (Stanford Linear Accelerator Center), Stanford, California.

Pohl, I. (1970). First results on the effect of error in heuristic search. In Meltzer, B. and Michie, D. (Eds.), *Machine Intelligence 5,* pp. 219–236. Elsevier/North-Holland, Amsterdam, London, New York.

Pohl, I. (1971). Bi-directional search. In Meltzer, B. and Michie, D. (Eds.), *Machine Intelligence 6,* p. 127–140. Edinburgh University Press, Edinburgh, Scotland.

Pohl, I. (1973). The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI-73),* pp. 20–23, Stanford, California. IJCAII.

Pohl, I. (1977). Practical and theoretical considerations in heuristic search algorithms. In Elcock, E. W. and Michie, D. (Eds.), *Machine Intelligence 8,* pp. 55–72. Ellis Horwood, Chichester, England.

Pomerleau, d. A. (1993). *Neural Network Perception for Mobile Robot Guidance.* Kluwer, Dordrecht, Netherlands.

Ponte, J. M. and Croft, W. B. (1998). A language modeling approach to information retrieval. In *Research and Development in Information Retrieval,* pp. 275–281.

Poole, D. (1993). Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence, 64,* 81–129.

Poole, D., Mackworth, A. K., and Goebel, R. (1998). *Computational intelligence: A logical approach.* Oxyford University Press, Oxford, UK.

Popper, K. R. (1959). *The Logic of Scientific Discovery.* Basic Books, New York.

Popper, K. R. (1962). *Conjectures and Refutations: The Growth of Scientific Knowledge.* Basic Books, New York.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program, 13*(3), 130–137.

Post, E. L. (1921). Introduction to a general theory of elementary propositions. *American Journal of Mathematics, 43,* 163–185.

Pradhan, M., Provan, G. M., Middleton, B., and Henrion, M. (1994). Knowledge engineering for large belief networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Tenth Conference,* pp. 484–490, Seattle, Washington, Morgan Kaufmann.

Pratt, V. R. (1976). Semantical considerations on Floyd-Hoare logic. In *Proceedings of the 17th IEEE Symposium on the Foundations of Computer Science,* pp. 109–121. IEEE Computer Society Press.

Created by XMLmind XSL-FO Converter.

Prawitz, d. (1960). An improved proof procedure. *Theoria, 26,* 102–139.

Prawitz, D. (1965). *Natural Deduction: A Proof Theoretical Study.* Almquist and Wiksell, Stockholm.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2002). *Numerical Recipes in C++: The Art of Scientific Computing* (Second edition). Cambridge University Press, Cambridge, UK.

Prieditis, A. E. (1993). Machine discovery of effective admissible heuristics. *Machine Learning, 12*(1–3), 117–141.

Prinz, D. G. (1952). Robot chess. *Research, 5,* 261–266.

Prior, A. N. (1967). *Past, Present, and Future.* Oxford University Press, Oxford, UK.

Prosser, P. (1993). Hybrid algorithms for constraint satisfaction problems. *Computational Intelligence, 9,* 268–299.

Pryor, L. and Collins, G. (1996). Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research, 4,* 287–339.

Pullum, g. K. (1991). *The Great Eskimo Vocabulary Hoax (and Other Irreverent Essays on the Study of Language).* University of Chicago Press, Chicago.

Pullum, G. K. (1996). Learnability, hyperlearning, and the poverty of the stimulus. In *22nd Annual Meeting of the Berkeley Linguistics Society.*

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley, New York.

Puterman, M. L. and Shin, M. C. (1978). Modified policy iteration algorithms for discounted Markov decision problems. *Management Science, 24*(11), 1127–1137.

Putnam, H. (1960). Minds and machines. In Hook, S. (Ed.), *Dimensions of Mind,* pp. 138–164. Macmillan, London.

Putnam, H. (1963). 'Degree of confirmation' and inductive logic. In Schilpp, P. A. (Ed.), *The Philosophy of Rudolf Carnap,* pp. 270–292. Open Court, La Salle, Illinois.

Putnam, H. (1967). The nature of mental states. In Capitan, W. H. and Merrill, D. D. (Eds.), *Art, Mind, and Religion,* pp. 37–48. University of Pittsburgh Press, Pittsburgh.

Pylyshyn, Z. W. (1974). Minds, machines and phenomenology: Some reflections on Dreyfus' „What Computers Can't Do". *International Journal of Cognitive Psychology, 3*(1), 57–77.

Pylyshyn, Z. W. (1984). *Computation and Cognition: Toward a Foundation for Cognitive Science.* MIT Press, Cambridge, Massachusetts.

Quillian, R. (1961). A design for an understanding machine. Paper presented at a colloquium: Semantic Problems in Natural Language, King's College, Cambridge, England.

Quine, W. V. (1953). Two dogmas of empiricism. In *From a Logical Point of View,* pp. 20–46. Harper and Row, New York.

Quine, W. V. (1960). *Word and Object.* MIT Press, Cambridge, Massachusetts.

Quine, W. V. (1982). *Methods of Logic* (Fourth edition). Harvard University Press, Cambridge, Massachusetts.

Quinlan, e. and O'Brien, S. (1992). Sublanguage: Characteristics and selection guidelines for MT. In *AI and Cognitive Science '92: Proceedings of Annual Irish Conference on Artificial Intelligence and Cognitive Science '92,* pp. 342–345, Limerick, Ireland. Springer-Verlag.

Quinlan, J. R. (1979). Discovering rules from large collections of examples: A case study. In Michie, D. (Ed.), *Expert Systems in the Microelectronic Age.* Edinburgh University Press, Edinburgh, Scotland.

Created by XMLmind XSL-FO Converter.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*, 81–106.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning, 5*(3), 239–266.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning.* Morgan Kaufmann, San Mateo, California.

Quinlan, J. R. and Cameron-Jones, R. M. (1993). FOIL: a midterm report. In Brazil, P. B. (Ed.), *European Conference on Machine Learning Proceedings (ECML-93)*, pp. 3–20, Vienna. Springer-Verlag.

Quirk, R., Greenbaum, S., Leech, G., and Svartvik, J. (1985). *A Comprehensive Grammar of the English Language.* Longman, New York.

Rabani, Y. Rabinovich, Y., and Sinclair, A. (1998). A computational view of population genetics. *Random Structures and Algorithms, 12*(4), 313–334.

Rabiner, L. R. and Juang, B.-H. (1993). *Fundamental os Speech Recognition.* Prentice-Hall, Upper Saddle River, New Jersey.

Ramakrishnan, R. and Ullman, J. D. (1995). A survey of research in deductive database systems. *Journal of Logic Programming, 23*(2), 125–149.

Ramsey, F. P. (1931). Truth and probability. In Braithwaite, R. B. (Ed.), *The Foundations of Mathematics and Other Logical Essays.* Harcourt Brace Jovanovich, New York.

Raphson, J. (1690). *Analysis aequationum universalis.* Apud Abelem Swalle, London.

Rassenti, S., Smith, V., and Bulfin, R. (1982). A combinatorial auction mechanism for airport time slot allocation. *Bell Journal of Economics, 13*, 402–417.

Ratner, D. and Warmuth, M. (1986). Finding a shortest solution for th $n \times n$ extension of the 15-puzzle is intractable. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Vol. 1, pp. 168–172, Philadelphia. Morgan Kaufmann.

Rauch, H. E., Tung, F., and Striebel, C. T. (1965). Maximum likelihood estimates of linear dynamic systems. *AIAA Journal, 3*(8), 1445–1450.

Rechenberg, I. (1965). Cybernetic solution path of an experimental problem. Library translation 1122, Royal Aircraft Establishment.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, Stuttgart, Germany.

Regin, J. (1994). A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pp. 362–367, Seattle. AAAI Press.

Reichenbach, H. (1949). *The Theory of Probability: An Inquiry into the Logical and Mathematical Foundations of the Calculus of Probability* (Second edition). University of California Press, Berkeley and Los Angeles.

Reif, J. (1979). Complexity of the mover's problem and generalizations. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, pp. 421–427, San Juan, Puero Rico. IEEE, IEEE Computer Society Press.

Reiter, E. and Dale, R. (2000). *Building Natural Language Generation Systems.* Studies in Natural Language Processing. Cambridge University Press, Cambridge, UK.

Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence, 13*(1–2), 81–132.

Reiter, R. (1991). The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V. (Ed.), *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pp. 359–380. Academic Press, New York.

Created by XMLmind XSL-FO Converter.

Reiter, R. (2001a). On knowledge-based programming with sensing in the situation calculus. *ACM Transactions on Computational Logic, 2*(4), 433– 457.

Reiter, R. (2001b). *Knowledge in Action: Logical Foundations for Specifying Implementing Dynamical Systems.* MIT Press, Cambridge, Massachusetts.

Reitman, W. and Wilcox, B. (1979). The structure and performance of the INTERIM.2 Go program. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence (IJCAI-79)*, pp. 711–719, Tokyo. IJCAII.

Remus, H. (1962). Simulation of a learning machine for playing Go. In *Proceedings IFIP Congress*, pp. 428–432, Amsterdam, London, New York. Elsevier/North-Holland.

Rényi, A.)1970). *Probability Theory.* Elsevier/North-Holland, Amsterdam, London, New York.

Rescher, N. and Urquhart, A. (1971). *Temporal Logic.* Springer-Verlag, Berlin.

Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics, 21*, 25–34. SIGGRAPH '87 Conference Proceedings.

Rich, E. and Knight, K. (1991). *Artificial Intelligence* (second edition). McGraw-Hill, New York.

Richardson, M., Bilmes, J., and Diorio, C. (2000). Hidden-articulator Markov models: Performance improvements and robustness to noise. In ICASSP-2000: *2000 International Conference on Acoustics, Speech, and Signal Processing*, Los Alamitos, CA. IEEE Computer Society Press.

Rieger, C. (1976). An organization of knowledge for problem solving and language comprehension. *Artificial Intelligence, 7*, 89–127.

Ringle, M. (1979). *Philosophical Perspectives in Artificial Intelligence.* Humanities Press, Atlantic Highlands, New Jersey.

Rintanen, J. (1999). Improvements to the evaluation of quantified boolean formulae. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pp. 1192–1197, Stockholm. Morgan Kaufmann.

Ripley, B. D. (1996). *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge, UK.

Rissanen, J. (1984). Universal coding, information, prediction, and estimation. *IEEE Transactions on Information Theory, IT-30*(4), 629–636.

Ritchie, G. D. and Hanna, F. K. (1984). AM: A case study in AI methodology. *Artificial Intelligence, 23*(3), 249–268.

Rivest, R. (1987). Learning decision lists. *Machine Learning, 2*(3), 229–246.

Roberts, L. G. (1963). Machine perception of three-dimensional solids. Technical report 315, MIT Lincoln Laboratory.

Robertson, N. and Seymour, P. D. (1986). Graph minors. ii. Algorithmic aspects of tree-width. *Journal of Algorithms, 7*(3), 309–322.

Robertson, S. E. (1977). The probability ranking principle in ir. *Journal of Documentation, 33*, 294–304.

Robertson, S. E. and Sparck Jones, K. (1976). Relevance weighting of search terms. *Journal of the American Society for Information Science, 27*, 129–146.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12, *23–41*.

Roche, E. and Schabes, Y. (1977). *Finite-State Language Processing (Language, Speech and Communication).* Bradford Books, Cambridge.

Created by XMLmind XSL-FO Converter.

Rock, I. (1984). *Perception.* W. H. Freeman, New York.

Rorty, R. (1965). Mind-body identity, privacy, and categories. *Review of Metaphysics, 19*(1), 24–54.

Rosenblatt, F. (1957). The perception: A perceiving and recognizing automaton. Report 85-460-1, Project Para, Cornell Aeronautical Laboratory, Ithaca, New York.

Rosenblatt, F. (1960). On the convergence of reinforcement procedures in simple perceptrons. Report VG-1196-G-4, Cornell Aeronautical Laboratory, Ithaca, New York.

Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan, Chicago.

Rosenblatt, M. (1956). Remarks on some nonparametric estimates of a density function. *Annals of Mathematical Statistics, 27,* 832–837.

Rosenblueth, A., Wiener, N., and Bigelow, J. (1943). Behavior, purpose, and teleology. *Philosophy of Science, 10,* 18–24.

Rosenschein, J. S. and Zlotkin, G. (1994). *Rules of Encounter.* MIT Press, Cambridge, Massachusetts.

Rosenschein, S. J. (1985). Formal theories of knowledge in AI and robotics. *New Generation Computing, 3*(4), 345–357.

Rosenthal, D. M. (Ed.) (1971). *Materialism and the Mind-Body Problem.* Prentice-Hall, Upper Saddle River, New Jersey.

Ross, S. M. (1988). *A First Course in Probability* (third edition). Macmillan, London.

Roussel, P. (1975). Prolog: Manual de reference et d'utilization. Tech. ren. Groupe d'Intelligence Artificielle, Université d'Aix-Marseille.

Rouveirol, C. and Puget, J.-F. (1989). A simple and general solution for inverting resolution. In *Proceedings of the European Working Session on Learning,* pp. 201–210, Porto, Portugal Pitman.

Rowat, P. F. (1979). *Representing the Spatial Experience and Solving Spatial problems in a Simulated Robot Environment.* Ph.D. thesis, University of British Columbia, Vancouver, BC, Canada.

Roweis, S. T. and Ghahramani, U. (1999). A unifying review of Linear Gaussian Models. *Neural Computation, 11*(2), 305–345.

Rubin, D. (1988). Using the SIR algorithm to simulate posterior distributions. In Bernardo, J. M., de Groot, M. H., Lindley, D. V., and Smith, A. F. M. (Eds.), *Bayesian Statistics 3,* pp. 395–402. Oxford University Press, Oxford, UK.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986a). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L. (Eds.), *Parallel Distributed Processing.* Vol. 1, chap. 8, pp. 318–362. MIT Press, Cambridge, Massachusetts.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986b). Learning representations by backpropagating errors. *Nature, 323,* 533–536.

Rumelhart, D. E. and McClelland, J. L. (Eds.). (1986). *Parallel Distributed Processing.* MIT Press, Cambridge, Massachusetts.

Ruspini, E. H. Lowrance, J. D., and Strat, T. M. (1992). Understanding evidential reasoning. *Intgernational Journal of Approximate Reasoning, 6*(3), 401–424.

Russell, J. G. B. (1990). Is screening for abdominal aortic aneurysm worthwhile?. *Clinical Radiology, 41,* 182–184.

Russell, S. J. (1985). The compleat guide to MRS. Report STAN-CS-85-1080, Computer Science Department, Stanford University.

Created by XMLmind XSL-FO Converter.

Russell, S. J. (1986). A quantitative analysis of analogy by similarity. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pp. 284–288, Philadelphia. Morgan Kaufmann.

Russell, S. J. (1988). Tree-structured bias. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, Vol. 2. pp. 641–645, St. Paul, Minnesota. Morgan Kaufmann.

Russell, S. J. (1992). Efficient memory-bounded search methods. In *ECAI 92: 10th European Conference on Artificial Intelligence Proceedings*, pp. 1–5, Vienna. Wiley.

Russell, S. J. (1998). Learning agents for uncertain environments (extended abstract). In *Proceedings of the Eleventh Annual ACM Workshop on Computational Learning Theory (COLT-98)*, pp. 101–103, Madison, Wisconsin. ACM Press.

Russell, S. J., Binder, J., Koller, D., and Kanazawa, K. (1995). Local learning in probabilistic networks with hidden variables. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pp. 1146–52, Montreal. Morgan Kaufmann.

Russell, S. J. and Grosof, B. (1987). A declarative approach to bias in concept learning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, Seattle. Morgan Kaufmann.

Russell, S. J. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Upper Saddle River, New Jersey.

Russell, S. J. and Subramanian, D. (1995). Provably bounded-optimal agents. *Journal of Artificial Intelligence Research*, *3*, 575–609.

Russell, S. J., Subramanian, D., and Parr, R. (1993). Provably bounded optimal agents. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pp. 338–345, Chambery, France. Morgan Kaufmann.

Russell, S. J. and Wefald, E. H. (1989). On optimal game-tree search using rational meta-reasoning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pp. 334–340, Detroit. Morgan Kaufmann.

Russell, S. J. and Wefald, E. H. (1991). *Do the Right Thing: Studies in Limited Rationality*. MIT Press, Cambridge, Massachusetts.

Rustagi, J. S. (1976). *Variational Methods in Statistics.* Academic Press, New York.

Ryder, J. L. (1971). Heuristic analysis of large trees as generated in the game of Go. Memo AIM-155, Stanford Artificial Intelligence Project, Computer Science Department, Stanford University, Stanford, California.

Sabin, D. and Freuder, E. C. (1994). Contradicting conventional wisdom in constraint satisfaction. In *ECAI 94: 11th European Conference on Artificial Intelligence. Proceedings*, pp. 125–129, Amsterdam. Wiley.

Sacerdoti, E. D. (1974). Planning in a hieararchy of abstraction spaces. *Artificial Intelligence*, *5*(2), 115–135.

Sacerdoti, E. D. (1975). The nonlinear nature of plans. *In Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, pp. 206–214, Tbilisi, Georgia. IJCAII.

Sacerdoti, E. D. (1977). *A Structure for Plans and Behavior*. Elsevier/North-Holland, Amsterdam, London, New York.

Sacerdoti, E. D., Fikes, R. E., Reboh, R., Sagalowicz, D., Waldinger, R., and Wilber, B. M. (1976). QLISP – a language for the interactive development of complex systems. In *Proceedings of the AFIPS National Computer Conference*, pp. 349–356.

Sacks, E. and Joskowicz, L. (1993). Automated modeling and kinematic simulation of mechanisms. *Computer Aided Design*, *25*(2), 106–118.

Sadri, F. and Kowalski, R. (1995). Variants of the event calculus. In *International Conference on Logic Programming*, pp. 67–81.

Created by XMLmind XSL-FO Converter.

Sag, I. and Wasow, T. (1999). *Syntactic Theory: An Introduction.* CSLI Publications, Stanford, California.

Sager, N. (1981). *Natural Language Information Processing: A Computer Grammar of English and 1st Applications.* Addison-Wesley, Reading, Massachusetts.

Sahami, M., Dumais, S. T., Heckerman, D., and Horvitz, E. J. (1998). A Bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop,* Madison, Wisconsin. AAAI Technical Report WS-98-05.

Sahami, M., Hearst, M. A., and Saund, E. (1996). Applying the multiple cause mixture model to text categorization. In Saitta, L. (Ed.), *Proceedings of ICML-96, 13th International Conference on Machine Learning,* pp. 435–443, Bari, Italy. Morgan Kaufmann Publishers.

Salomaa, A. (1969). Probabilistic and weighted grammars. *Information and Control, 15, 5*29– 544.

Salton, G. and McGill, M. J. (1983). *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, NY.

Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM, 18*(11), 613–620.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development, 3*(3), 210–229.

Samuel, A. L. (1967). Some studies in machine learning using the game of checkers II – Recent progress. *IBM Journal of Research and Development, 11*(6), 601–617.

Samuelsson, C. and Rayner, M. (1991). Quantitative evaluation of explanation-based learning as an optimization tool for a large-scale natural language system. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91),* pp. 609–615, Sydney. Morgan Kaufmann.

Sato, T. and Kameya, Y. (1997). PRISM: A symbolic-statistical modeling language. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97),* pp. 1330–1335, Nagoya, Japan. Morgan Kaufmann.

Saul, L. K., Jaakkola, T., and Jordan, M. I. (1996). Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research, 4,* 61–76.

Savage, L. J. (1954). *The Foundations of Statistics.* Wiley, New York.

Sayere, K. (1993). Three more flaws in the computation model. Paper presented at the APA (Central Division) Annual Conference, Chicago, Illinois.

Schabes, Y., Abeille, A., and Joshi, A. K. (1988). Parsing strategies with lexicalized grammars: Application to tree adjoining grammars. In Vargha, D. (Ed.), *Proceedings of the 12th International Conference on Computational Linguistics (COLING),* Vol. 2, pp. 578–583, Budapest, John von Neumann Society for Computer Science.

Schaeffer, J. (1997). *One Jump Ahead: Challenging Human Supremacy in Checkers.* Springer-Verlag, Berlin.

Schank, R. C. and Abelson, R. P. (1977). *Scripts, Plans, Goals, and Understanding.* Lawrence Erlbaum Associates, Potomac, Maryland.

Schank, R. C. and Riesbeck, C. (1981). *Inside Computer Understanding: Five Programs Plus Miniatures.* Lawrence Erlbaum Associates, Potomac, Maryland.

Schapire, R. E. (1999). Theoretical views of boosting and applications. In *Algorithmic Learning Theory: Proceedings of the 10th International Conference (ALT'99),* pp. 13–25. Springer-Verlag, Berlin.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning, 5*(2), 197–227.

Created by XMLmind XSL-FO Converter.

Schmolze, J. G. and Lipkis, T. A. (1983). Classification in the KL-ONE representation system. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83),* pp. 330–332, Karlsruhe, Germany. Morgan Kaufmann.

Schofield, P. D. A. (1967). Complete solution of the eight puzzle. In Dale, E. and Michie, D. (Eds.), *Machine Intelligence 2,* pp. 125–133. Elsevier/North-Holland, Amsterdam, London, New York.

Schölkopf, B. and Smola, A. J. (2002). *Learning with Kernels.* MIT Press, Cambridge, Massachusetts.

Schöning, T. (1999). A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science,* pp. 410–414, New York. IEEE Computer Society Press.

Schoppers, M. J. (1987). Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87),* pp. 1039– 1046, Milan. Morgan Kaufmann.

Schoppers, M. J. (1989). In defense of reaction plans as caches. *AI Magazine, 10*(4), 51–60.

Schröder, E. (1877). *Der Operationskreis des Logikkalküls.* B. G. Teubner, Leipzig.

Schultz, W., Dayan, P., and Montague, P. R. (1997). A neural substrate of prediction and reward. *Science, 275,* 1593.

Schütze, H. (1995). *Ambiguity in Language Learning: Computational and Cognitive Models.* Ph.D. thesis, Stanford University. Also published by CSLI Press, 1997.

Schwartz, J. T. Scharir, M., and Hopcroft, J. (1987). *Planning, Geometry and Complexity of Robot Motion.* Ablex Publishing Corporation, Norwood, NJ.

Schwartz, S. P. (Ed.). (1977). *Naming, Necessity, and Natural Kinds.* Cornell University Press, Ithaca, New York.

Scott, D. and Krauss, P. (1966). Assigning probabilities to logical formulas. In Hintikka, J. and Suppes, P. (Eds.), *Aspects of Inductive Logic:* North-Holland, Amsterdam.

Scriven, M. (1953). The mechanical concept of mind. *Mind, 62,* 230–240.

Searle, J. R. (1969). *Speech Acts: An Essay in the Philosophy of Language.* Cambridge University Press, Cambridge, UK.

Searle, J. R. (1980). Minds, brains, and programs. *Behavior and Brain Sciences, 3,* 417–457.

Searle, J. R. (1984). *Minds, Brains and Science.* Harvard University Press, Cambridge, Massachusetts.

Searle, J. R. (1990). Is the brain's mind a computer program?. *Scientific American, 262,* 26–31.

Searle, J. R. (1992). *The Rediscovery of the Mind.* MIT Press, Cambridge, Massachusetts.

Selman, B., Kautz, H., and Cohen, B. (1996). Local search strategies for satisfiability testing. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 26,* pp. 521–532. American Mathematical Society, Providence, Rhode Island.

Selman, B. and Levesque, H. J. (1993). The complexity of path-based defeasible inheritance. *Artificial Intelligence, 62*(2), 303–339.

Selman, B. Levesque, H. J., and Mitchell, D. (1992). A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92),* pp. 440–446, San Jose. AAAI Press.

Shachter, R. D. (1986). Evaluating influence diagrams. *Operations Research, 34,* 871–882.

Created by XMLmind XSL-FO Converter.

Shachter, R. D. (1998). Bayes-ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In *Uncertainty in Artificial Intelligence: Proceedings of the Fourtheenth Conference*, pp. 480–487, Madison, Wisconsin. Morgan Kaufmann.

Shachter, R. D., D'Ambrosio, B., and Del Favero, B. A. (1990). Symbolic probabilistic inference in belief networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pp. 126–131, Boston, MIT Press.

Shachter, R. D. and Kenley, C. R. (1989). Gaussian influence diagrams. *Management Science*, *35*(5), 527–550.

Shachter, R. D. and Peot, M. (1989). Simulation approaches to general probabilistic inference on belief networks. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89)*, Windsor, Ontario, Morgan Kaufmann.

Shafer, G. (1976). *A Mathematical Theory of Evidence.* Princeton University Press, Princeton, New Jersey.

Shafer, G. and Pearl, J. (Eds.). (1990). *Readings in Uncertain Reasoning.* Morgan Kaufmann, San Mateo, California.

Shahookar, K. and Mazumder, P. (1991). VLSI cell placement techniques. *Computing Surveys*, *23*(2), 143–220.

Shanahan, M. (1997). *Solving the Frame Pproblem.* MIT Press, Cambridge, Massachusetts.

Shanahan, M. (1999). The event calculus explained. In Wooldridge, M. J. and Veloso, M. (Eds.), *Artificial Intelligence Today*, pp. 409–430. Springer-Verlag, Berlin.

Shankar, N. (1986). *Proof-Checking Metamathematics.* Ph.D. thesis, Computer Science Department, University of Texas at Austin.

Shannon, C. E. and Weaver, W. (1949). *The Mathematical Theory of Communication.* University of Illinois Press, Urbana, Illinois.

Shannon, C. E. (1950). Programming a computer for playing chess. *Philosophical Magazine*, *41*(4), 256–275.

Shapiro, E. (1981). An algorithm that infers theories from facts. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*, p. 1064, Vancouver, British Columbia. Morgan Kaufmann.

Shapiro, S. C. (Ed.). (1992). *Encyclopedia of Artificial Intelligence* (second edition). Wiley, New York.

Shapley, S. (1953). Stochastic games. In *Proceedings of the National Academy of Sciences*, Vol. 39, pp. 1095–1100.

Shavlik, J. and Dietterich, T. (Eds.). (1990). *Readings in Machine Learning.* Morgan Kaufmann, San Mateo, California.

Shelley, M. (1818). *Frankenstein: or, the Modern Prometheus.* Pickering and Chatto.

Shenoy, P. P. (1989). A valuation-based language for expert systems. *International Journal of Approximate Reasoning*, *3*(5), 383–411.

Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, *22*(8), 888–905.

Shoham, Y. (1987). Temporal logics in AI: Semantical and ontological considerations. *Artificial Intelligence*, *33*(1), 89–104.

Shoham, Y. (1993). Agent-oriented programming. *Artificial Intelligence*, *60*(1), 51–92.

Shoham, Y. (1994). *Artificial Intelligence Techniques in Prolog.* Morgan Kaufmann, San Mateo, California.

Created by XMLmind XSL-FO Converter.

Shortliffe, E. H. (1976). *Computer-Based Medical Consultations: MYCIN.* Elsevier/North-Holland, Amsterdam, London, New York.

Shwe, M. and Cooper, G. (1991). An empirical analysis of likelihood-weighting simulation on a large, multiply connected medical belief networks. *Computers and Biomedical Research, 1991*(5), 453–475.

Siekmann, J. and Wrightson, G. (Eds.), (1983). *Automation of Reasoning.* Springer-Verlag, Berlin.

Sietsma, J. and Dow, R. J. F. (1988). Neural net pruning – why and how. In *IEEE International Conference on Neural Networks*, pp. 325–333, San Diego. IEEE.

Siklossy, L. and Dreussi, J. (1973). An efficient robot planner which generates its own procedures. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI-73)*, pp. 423–430, Stanford, California. IJCAII.

Silverstein, C., Henzinger, M., Marais. H., and Moricz, M. (1998). Analysis of a very large altavista query log. Tech. rep. 1998–014, Digital Systems Research Center.

Simmons, R. and Koenig, S. (1995). Probabilistic robot navigation in partially observable environments. In *Proceedings of IJC*AI-95. pp. 1080– 1087, Montreal, Canada. IJCAI Inc.

Simmons, R. and Slocum, J. (1972). Generating english discourse from semantic networks. *Communications of the ACM, 15*(10), 891–905.

Simon, H. A. (1947). *Administrative behavior.* Macmillan, New York.

Simon, H. A. (1957). *Models of Man: Social and Rational.* John Wiley, New York.

Simon, H. A. (1963). Experiments with a heuristic compiler. *Journal of the Association for Computing Machinery, 10*, 493–506.

Simon, H. A. (1981). *The Sciences of the Artificial* (second edition). MIT Press, Cambridge, Massachusetts.

Simon, H. A. (1982). *Models of Bounded Rationality, Volume 1*. The MIT Press, Cambridge, Massachusetts.

Simon, H. A. and Newell, A. (1958). Heuristic problem solving: The next advance in operations research. *Operations Research, 6*, 1–10.

Simon, H. A. and Newell, A. (1961). Computer simulation of human thinking and problem solving. *Datamation, June/July*, 35–37.

Simon, J. C. and Dubois, O. (1989). Number of solutions to satisfiability instances – applications to knowledge bases. *Int. J. Pattern Recognition and Artificial Intelligence, 3*, 53–65.

Sirovitch, L. and Kirby, M. (1987). Low-dimensional procedure for characterization of human faces. *Journal of the Optical Society of America A, 2*, 586–591.

Skinner, B. F. (1953). *Science and Human Behavior.* Macmillan, London.

Skolem, T. (1920). Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze nebst einem Theoreme über die dichte Mengen. *Videnskapsselskapetes skrifter, I. Matematisk-naturvidenskabelig klasse, 4.*

Skolem, T. (1928). Über die mathematische Logik. *Norsk matematisk tidsskrift, 10*, 125–142.

Slagle, J. R. (1963a). A heuristic program that solves symbolic integration problems in freshman calculus. *Journal of the Association for Computing Machinery, 10*(4).

Slagle, J. R. (1963b). Game trees, *m & n* minimaxing, and the *m & n* alpha-beta procedure. Artificial intelligence group report 3, University of California, Lawrence Radiation Laboratory, Livermore, California.

Slagle, J. R. and Dixon, J. K. (1969). Experiments with some programs that search game trees. *Journal of the Association for Computing Machinery, 16*(2), 189–207.

Created by XMLmind XSL-FO Converter.

Slate, D. J. and Atkin, L. R. (1977). CHESS 4.5 – Northwestern University chess program. In Frey, P. W. (Ed.), *Chess Skill in Man and Machine*, pp. 82–118. Springer-Verlag, Berlin.

Slater, E. (1950). Statistics for the chess computer and the factor of mobility. In *Symposiumon Information Theory*, pp. 150–152, London,. Ministry of Supply.

Sleator, D. and Temperley, D. (1993). Parsing English with a link grammar. In *Third Annual Workshop on Parsing technologies.*

Sloman, A. (1978). *The Computer Revolution in Philosophy*. Harvester Press, Hassocks, Sussex, UK.

Smallwood, R. D. and Sondik, E. J. (1973). The optimal control of partially observable Markov processes over a finite horizon. *Operations Research, 21*, 1071–1088.

Smith, D. E., Genesereth, M. R., and Ginsberg, M. L. (1986). Controlling recursive inference. *Artificial Intelligence, 30*(3), 343–389.

Smith, D. R. (1990). KIDS: a semiautomatic program development system. *IEEE Transactions on Software Engineering, 16*(9), 1024–1043.

Smith, D. R. (1996). Machine suppurt for software development. In *Proceedings of the 18th International Conference on Software Engineering*, pp. 167–168, Berlin. IEEE Computer Society Press.

Smith, D. E. and Weld, D. S. (1998). Conformant Graphplan. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 889–896, Madison, Wisconsin. AAAI Press.

Smith, J. Q. (1988). *Decision Analysis.* Chapman and Hall, London.

Smith, J. M. and Szathmáry, E. (1999). *The Origins of Life: From the Birth of Life to the Origin of Language.* Oxford University Press, Oxford, UK.

Smith, R. C. and Cheeseman, P.(1986). On the representation and estimation of spatial uncertainty. *International Journal of Robotics Research, 5*(4), 56–68.

Smith, S. J. J., Nau, D. S., and Throop, T. A. (1998). Success in spades: Using ai planning techniques to win the world championship of computer bridge. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 1079–1086, Madison, Wisconsin. AAAI Press.

Smolensky, P. (1988). On the proper treatment of connectionism. *Behavioral and Brain Sciences, 2*, 1–74.

Smyth, P. Heckerman, D., and Jordan, M. I. (1997). Probabilistic independence networks for hidden Markov probability models. *Neural Computation, 9*(2), 227–269.

Soderland, S. and Weld, D. S. (1991). Evaluating nonlinear planning. Technical report TR-91-02-03, University of Washington Department of Computer Science and Engineering, Seattle, Washington.

Solomonoff, R. J. (1964). A formal theory of inductive inference. *Information and Control, 7*, 1–22, 224–254.

Sondik, E. J. (1971). *The Optimal Control of Partially Observable Markov Decision Processes.* Ph.D. thesis, Stanford University, Stanford, California.

Sosic, r. and Gu, J. (1994). Efficient local search with conflict minimalization: A case study of the n-queens problem. *IEEE Transactions on Knowledge and Data Engineering, 6*(5), 661–668.

Sowa, J. (1999). *Knowledge Representation: Logical, Philosophical and Computational Foundations.* Blackwell, Oxford, UK.

Spiegelhalter, D. J. (1986). Probabilistic reasoning in predictive expert systems. In Kanal, L. N. and Lemmer, J. F. (Eds.), *Uncertainty in Artificial Intelligence*, pp. 47–67. Elsevier/North-Holland, Amsterdam, London, New York.

Spiegelhalter, D. J. Dawid, P., Lauritzen, S., and Cowell, R. (1993). Bayesian analysis in expert systems. *Statistical Science, 8*, 219–282.

Created by XMLmind XSL-FO Converter.

Spielberg, S. (2001). AI. movie.

Spirtes, P., Glymour, C., and Scheines, R. (1993). *Causation, prediction, and search.* Springer-Verlag, Berlin.

Springsteen, B. (1992). 57 channels (and nothin' on). In *Human Touch.* Sony.

Srinivasan, A., Muggleton, S. H., King, R. D., and Sternberg, M. J. E. (1994). Mutagenesis: ILP experiments in a non-determinate biological domain. In Wrobel, S. (Ed.), *Proceedings of the 4th International Workshop on Inductive Logic Programming*, Vol. 237, pp. 217–232. Gesellschaft für Mathematik und Datenverarbeitung MBH.

Srivas, M. and Bickford, M. (1990). Formal verification of a pipelined microprocessor. *IEEE Software*, *7*(5), 52–64.

Stallman, R. M. and Sussman, G. J. (1977). Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, *9*(2), 135–196.

Stanfill, C. and Waltz, D. (1986). Toward memory-based reasoning. *Communications of the Association for Computing Machinery*, *29*(12), 1213–1228.

Stefik, M. (1995). *Introduction to Knowledge Systems.* Morgan Kaufmann, San Mateo, California.

Stein, L. A. (2002). *Interactive Programming in Java (pre-publication draft).* Morgan Kaufmann, San Mateo, California.

Steinbach, M., Karypis, G., and Kumar, V. (2000). A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, pp. 109–110. ACM Press.

Stevens, K. A. (1981). The information content of texture gradients. *Biological Cybernetics*, *42*, 95–105.

Stickel, M. E. (1985). Automated deduction by theory resolution. *Journal of Automated Reasoning*, *1*(4), 333–355.

Stickel, M. E. (1988). A Prolog Technology Theorem Prover: implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, *4*, 353–380.

Stiller, L. B. (1992). KQNKRR. *ICCA Journal*, *15*(1), 16–18.

Stillings, N. A. Weisler, S., Feinstein, M. H. Garfield, J. L., and Rissland, E. L. (1995). *Cognitive Science: An Introduction* (second edition). MIT Press, Cambridge, Massachusetts.

Stockman, G. (1979). A minimax algorithm better than alpha–beta?. *Artificial Intelligence*, *12*(2), 179–196.

Stolcke, A. and Omohundro, S. (1994). Inducing probabilistic grammars by Bayesian model merging. In *Proceedings of the Second International Colloquium on Grammatical Inference and Applications (ICGI-94)*, pp. 106–118, Alicante, Spain. Springer-Verlag.

Stone, P. (2000). *Layered Learning in Multi-Agent Systems: A Winning Approach to Robotic Soccer.* MIT Press, Cambridge, Massachusetts.

Strachey, C. (1952). Logical or non-mathematical programmes. In *Proceedings of the Association for Computing Machinery (ACM)*, pp. 46–49, Toronto, Canada.

Subramanian, D. (1993). Artificial intelligence and conceptual design. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pp. 800–809, Chambery, France. Morgan Kaufmann.

Subramanian, D. and Feldman, R. (1990). The utility of EBL in recursive domain theories. In *Proceedings of the Eighth National Conference an Artificial Intelligence (AAAI-90)*, Vol. 2, pp. 942–949, Boston, MIT Press.

Subramanian, D. and Wang, E. (1994). Constraint-based kinematic synthesis. In *Proceedings of the International Conference on Qualitative Reasoning*, pp. 228–239. AAAI Press.

Created by XMLmind XSL-FO Converter.

Sugihara, K. (1984). A necessary and sufficient condition for a picture to represent a polyhedral scene. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 6*(5), 578–586.

Sussman, G. J. (1975). *A Computer Model of Skill Acquisition.* Elsevier/North-Holland, Amsterdam, London, New York.

Sussman, G. J. and Winograd, T. (1970). MICROPLANNER Reference Manual. Ai memo 203, MIT AI Lab, Cambridge, Massachusetts.

Sutherland, I. (1963). Sketchpad: A man-machine graphical communication system. In *Proceedings of the Spring Joint Computer Conference,* pp. 329–346. IFIPS.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning, 3*, 9–44.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In Solla, S. A., Leen, T. K., and Müller, K.-R. (Eds.), *Advances in Neural Information Processing Systems 12,* pp. 1057–1063. MIT Press, Cambridge, Massachusetts.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning: Proceedings of the Seventh International Conference,* pp. 216–224, Austin, Texas. Morgan Kaufmann.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, Massachusetts.

Swade, D. D. (1993). Redeeming Charles Babbage's mechanical computer. *Scientific American, 268*(2), 86–91.

Swerling, P. (1959). First order error propagation in a stagewise smoothing procedure for satellite observations. *Journal of Astronautical Sciences, 6*, 46–52.

Swift, T. and Warren, D. S. (1994). Analysis of SLG-WAM evaluation of definite programs. In *Logic Programming. Proceedings of the 1994 International Symposium,* pp. 219–235, Ithaca NY. MIT Press.

Syrjänen, T. (2000). Lparse 1.0 user's manual http://saturn.tcs.hut.fi/Software/smodels.

Tadepalli, P. (1993). Learning from queries and examples with tree-structured bias. In *Proceedings of the Tenth International Conference on Machine Learning,* pp. 322–329, Amherst, Massachusetts. Morgan Kaufmann.

Tait, P. G. (1880). Note on the theory of the „15 puzle". *Proceedings of the Royal Society of Edinburgh, 10,* 664–665.

Tamaki, H. and Sato, T. (1986). OLD resolution with tabulation. In *Third International Conference on Logic Programming,* pp. 84–98, London, Springer-Verlag.

Tambe, M., Newell, A., and Rosenbloom, P. S. (1990). The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning, 5,* 299–348.

Tarjan, R. E. (1983). *Data Structures and Network Algorithms.* CBMS-NSF Regional Conference Series im Applied Mathematics. SIAM (Society for Industrial and Applied Mathematics, Philadelphia.

Tarski, A. (1935). Die Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophica, 1,* 261–405.

Tarski, A. (1956). *Logic, Semantics, Metamathematics: Papers from 1923 to 1938.* Oxford University Press, Oxford, UK.

Tash, J. K. and Russell, S. J. (1994). Control strategies for a stochastic planner. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94),* pp. 1079–1085, Seattle. AAAI Press.

Tate, A. (1975a). Interacting goals and their use. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75),* pp. 215–218, Tbilisi, Georgia. IJCAII.

Tate, A. (1975b). *Using Goal Structure to Direct Search in a Problem Solver,* Ph.D. thesis, University of Edinburgh, Edinburgh, Scotland.

Created by XMLmind XSL-FO Converter.

Tate, A. (1977). Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, pp. 888–893, Cambridge, Massachusetts. IJCAII.

Tate, A. and Whiter, A. M. (1984). Planning with multiple resource constraints and an application to a naval planning problem. In *Proceedings of the First Conference on AI Applications*, pp. 410–416, Denver, Colorado.

Tatman, J. A. and Shachter, R. D. (1990). Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man and Cybernetics, 20*(2), 365–379.

Tesauro, G. (1989). Neurogammon wins computer olympiad. *Neural Computation, 1*(3), 321–323.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning, 8*(3–4), 257–277.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the Association for Computing Machinery, 38*(3), 58–68.

Tesauro, G. and Sejnowski, T. (1989). A parallel network that learns to play backgammon. *Artificial Intelligence, 39*(3) 357–390.

Thagard, P. (1996). *Mind: Introduction to Cognitive Science.* MIT Press, Cambridge, Massachusetts.

Thaler, R. (1992). *The Winner's Curse: Paradoxes and Anomalies of Economic Life.* Princeton University Press, Princeton, New Jersey.

Thielscher, M. (1999). From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence, 111*(1–2), 277–299.

Thomason, R. H. (Ed.). (1974). *Formal Philosophy: Selected Papers of Richard Montague.* Yale University Press, New Haven, Connecticut.

Thompson, D. W. (1917). *On Growth and Form.* Cambridge University Press, Cambridge, UK.

Thrun, S. (2000). Towards programming tools for robots that integrate probabilistic computation and learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA),* San Francisco, CA. IEEE.

Thrun, S. (2002). Robotic mapping: A survey. In Lakemeyer, G. and Nebel, B. (Eds.), *Exploring Artificial Intelligence in the New Millenium.* Morgan Kaufmann. to appear.

Titterington, D. M., Smith, A. F. M., and Makov, U. E. (1985). *Statistical analysis of finite mixture dictributions.* Wiley, New York.

Toffler, A. (1970). *Future Shock.* Bantam.

Tomasi, C. and Kanade, T. (1992). Shape and motion from image streams under orthography: A factorization method. *International Journal of Computer Vision, 9,* 137–154.

Touretzky, D. S. (1986). *The Mathematics of Inheritance Systems.* Pitman and Morgan Kaufmann, London and San Mateo, California.

Trucco, E. and Verri, A. (1998). *Introductory Techniques for 3-D Computer Vision.* Prentice Hall, Upper Saddle River, New Jersey.

Tsang, E. (1993). *Foundations of Constraint Satisfaction.* Academic Press, New York.

Tsitsiklis, J. N. and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control, 42*(5), 674–690.

Tumer, K. and Wolpert, D. (2000). Collective intelligence and braess' paradox. In *Proceedings of the AAAI/IAAI,* pp. 104–109.

Turcotte, M., Muggleton, S. H., and Sternberg, M. J. E. (2001). Automated discovery of structural signatures of protein fold and function. *Journal of Molecular Biology, 306,* 591–605.

Created by XMLmind XSL-FO Converter.

Turing, A. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, 2nd series, 42,* 230–265.

Turing, A. (1948). Intelligent machinery. Tech. rep., National Physical Laboratory. reprinted in (Ince, 1992).

Turing, A. (1950). Computing machinery and intelligence. *Mind, 59,* 433–460.

Turing, A., Strachey, C., Bates, M. A. and Bowden, B. V. (1953). Digital computers applied to games. In Bowden, B. V. (Ed.), *Faster than Thought,* pp. 286–310. Pitman, London.

Turtle, H. R. and Croft, W. B. (1992). A comparison of text retrieval models. *The Computer Journal, 35*(1), 279–289.

Tversky, A. and Kahneman, D. (1982). Causal schemata in judgements under uncrtainty. In Kahneman, D., Slovic, P., and Tversky, A. (Eds.), *Judgement Under Uncertainty. Heuristics and Biases.* Cambridge University Press, Cambridge, UK.

Ullman, J. D. (1985). Implementation of logical query languages for databases. *ACM Transactions on Database Systems, 10*(3), 289–321.

Ulmann, J. D. (1989). *Principles of Database and Knowledge-Base Bystems.* Computer Science Press, Rockville, Maryland.

Ullman, S. (1979). *The Interpretation of Visual Motion.* MIT Press, Cambridge, Massachusetts.

Vaessens, R. J. M., Aarts, E. H. I., and Lenstra, J. K. (1996). Job shop scheduling by local search. *INFORMS J. on Computing, 8,* 302–117.

Valiant, L. (1984). A theory of the learnable. *Communications of the Association for Computing Machinery, 27,* 1134–1142.

van Benthem, J. (1983). *The Logic of Time.* D. Reidel, Dordrecht, Netherlands.

Van Emden, M. H. and Kowalski, R. (1976). The semantics of predicate logic as a programming language. *Journal of the Association for Computing Machiney, 23*(4) 733–742.

van Harmelen, F. and Bundy, A. (1988). Explanation-based generalisation = partial evaluation. *Artificial Intelligence, 36*(3) 401–412.

van Heijenoort, J. (Ed.), (1967). *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931.* Harvard University Press, Cambridge, Massachusetts.

Van Hentenryck, P., Saraswat, V., and Deville, Y. (1998). Design, implementation, and evaluation of the constraint language cc(fd). *Journal of Logic Programming, 37*(1–3), 139–164.

van Nunen, J. A. E. E. (1976). A set of successive approximation methods for discounted Markovian decision problems. *Zeitschrift fur Operations Research, Serie A, 20*(5), 203–208.

van Roy, B. (1998). *Learning and value function approximation in complex decision processes.* Ph.D. thesis, Laboratory for Information and Decision Systems, MIT, Cambridge, Massachusetts.

Van Roy, P. L. (1990). Can logic programming executes, as fast as imperative programming?. Report UCB/CSD 90/600, Computer Science Division, University of California, Berkeley, California.

Vapnik, V. N. (1998). *Statistical Learning Theory.* Wiley, New York.

Vapnik, V. N. and Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications, 16,* 264–280.

Varian, H. R. (1995). Economic mechanism design for computerized agents. In *USENIX Workshop on Electronic Commerce,* pp. 13–21.

Created by XMLmind XSL-FO Converter.

Veloso, M. and Carbonell, J. G. (1993). Derivational analogy in prodigy: Automating case acquisition, storage, and utilization. *Machine Learning, 10,* 249–278.

Vere, S. A. (1983). Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 5,* 246–267.

Vinge, V. (1993). The coming technological singularity: How to survive in the post-human era. In *VISION-21 Symposium.* NASA Lewis Research Center and the Ohio Aerospace Institute.

Viola, P. and Jones, M. (2002). Robust real-time object detection. *International Journal of Computer Vision,* in press.

von Mises, R. (1928). *Wahrscheinlichkeit, Statistik und Wahrheit.* J. Springer, Berlin.

von Neumann, J. (1928). Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen, 100*(295– 320).

von Neumann, J. and Morgenstern, O. (1944). *Theory of Games and Economic Behavior* (first edition). Princeton University Press, Princeton, New Jersey.

von Winterfeldt, D. and Edwards, W. (1986). *Desicion Analysis and Behavioral Research.* Chambridge University Press, Cambridge, UK.

Voorhees, E. M. (1993). Using WordNet to disambiguate word sens for text retrieval. In *Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 171–80, Pittsburgh. Association for Computing Machinery.

Vossen, T., Ball, M., Lotem, A., and Nau, D. S. (2001). Applying integer programming to ai planning. *Knowledge Engieneering Review, 16,* 85–100.

Waibel, A. and Lee, K.-F., (1990). *Readings in Speech Recognition.* Morgan Kaufmann, San Mateo, California.

Waldinger, R. (1975). Achieving several goals simultaneously. In Elcock, E. W. and Michie, D. (Eds.), *Machine Intelligence 8,* pp. 94–138. Ellis Horwood, Chichester, England.

Waltz, D. (1975). Understanding line drawings of scenes with shadows. In Winston, P. H. (Ed.), *the Psychology of Computer Vision.* McGraw-Hill, New York.

Wanner, E. (1974). *On remembering, forgetting and understanding sentences.* Mouton, The Hague and Paris.

Warren, D. H. D. (1974). WARPLAN: A System for Generating Plans. Department of Computational Logic Memo 76, University of Edinburgh, Edinburgh, Scotland.

Warren, D. H. D. (1976). Generating conditional plans and programs. In *Proceedings of the AISB Summer Conference,* pp. 344–354.

Warren, D. H. D. (1983). An abstract Prolog instruction set. Technical note 309, SRI International, Menlo Park, California.

Warren, D. H. D., Pereira, L. M., and Pereira, F. (1977). PROLOG: The language and its implementation compared with LISP. *SIGPLAN Notices, 12*(8), 109–115.

Watkins, C. J. (1989). *Models of Delayed Reinforcement Learning.* Ph.D. thesis, Psychology Department, Cambridge University, Cambridge, UK.

Watson, J. D. and Crick, F. H. C. (1953). A structure for deoxyribose nucleic acid. *Nature, 171,* 737.

Webber, B. L. (1983). So what can we talk about now? In Brady, M. and Berwick, R. (Eds.), *Computational Models of Discourse.* MIT Press, Cambridge, Massachusetts.

Webber, B. L. (1988). Tense as discourse anaphora. *Computational Linguistics, 14*(2), 61–73.

Webber, B. L. and Nilsson, N. J. (Eds.). (1981). *Readings in Artificial Intelligence.* Morgan Kaufmann, San Mateo, California.

Created by XMLmind XSL-FO Converter.

Weidenbach, C. (2001). SPASS: Combining superposition, sorts and splitting. In Robinson, A. and Voronkov, A. (Eds.), *Handbook of Automated Reasoning.* mit, mit-ad.

Weiss, G. (1999). *Multiagent systems.* MIT Press, Cambridge, Massachusetts.

Weiss, S. and Kulikowski, C. A. (1991). *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems.* Morgan Kaufmann, San Mateo, California.

Weizenbaum, J. (1976). *Computer Power and Human Reason.* W. H. Freeman, New York.

Weld, D. S. (1994). An introduction to least commitment planning. *AI Magazine, 15*(4), 27–61.

Weld, D. S. (1999). Recent advances in ai planning. *AI Magazine, 20*(2) 93–122.

Weld, D. S. Anderson, C. R., and Smith, D. E. (1998). Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98),* pp. 897–904, Madison, Wisconsin. AAAI Press.

Weld, D. S. and de Kleer, J. (1990). *Readings in Qualitative Reasoning about Physical Systems.* Morgan Kaufmann, San Mateo, California.

Weld, D. S. and Etzioni, O. 1994). The first law of robotics: A call to arms. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94),* Seattle. AAAI Press.

Wellman, M. P. (1985). Reasoning about preference models. Technical report MIT/LCS/TR-340, Laboratory for Computer Science, MIT, Cambridge, Massachusetts.

Wellman, M. P. (1988). *Formulation of Tradeoffs in Planning under Uncertainty.* Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Wellman, M. P. (1990a). Fundamental concepts of qualitative probabilistic networks. *Artificial Intelligence, 44*(3), 257–303.

Wellman, M. P. (1990b). The STRIPS assumption for planning under uncertainty. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90),* pp. 198–203, Boston. MIT Press.

Wellman, M. P. (1995). The economic approach to artificial intelligence. *ACM Computing Surveys, 27*(3), 360–362.

Wellman, M. P., Breese, J. S., and Goldman, R. (1992). From knowledge bases to decision models. *Knowledge Engieneering Review, 7*(1), 35–53.

Wellman, M. P. and Doyle, J. (1992). Modular utility representation for decision-theoretic planning. In *Proceedings, First International Conference on AI Planning Systems,* pp. 236–242, College Park, Maryland. Morgan Kaufmann.

Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* Ph.D. thesis, Harvard University, Cambridge, Massachusetts.

Werbos, P. (1977). Advanced forecasting methods for global crisis warning and models of intelligence. *General Systems Yearbook, 22,* 25–38.

Wesley, M. A. and Lozano-Perez, T. (1979). An algorithm for planning collision-free paths among polyhedral objects. *Communications of the ACM, 22*(10), 560–570.

Wheatstone, C. (1838). On some remarkable, and hitherto unresolved, phenomena of binocular vision. *Philosophical Transactions of the Royal Society of London, 2,* 371–394.

Whitehead, A. N. (1911). *An Introduction to Mathematics.* Williams and Northgate, London.

Whitehead, A. N. and Russell, B. (1910). *Principia Mathematica.* Cambridge University Press, Cambridge, UK.

Created by XMLmind XSL-FO Converter.

Whorf, B. (1956). *Language, Thought, and Reality*. MIT Press. Cambridge, Massachusetts.

Widrow, B. (1962). Generalization and information storage in networks of adaline „neurons". In Yovits, M. C. Jacobi, G. T., and Goldstein, G. D. (Eds.), *Self-Organizing Systems 1962*, pp. 435–461, Chicago, Illionis. Spartan.

Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, pp. 96–104, New York.

Wiener, N. (1942). The extrapolation, interpolation, and smoothing of stationary time series. Osrd 370, Report to the Services 19, Research Project DIC-6037, MIT, Cambridge, Massachusetts.

Wiener, N. (1948). *Cybernetics.* Wiley, New York.

Wilensky, R. (1978). *Understanding goal-based stories.* Ph.D. thesis, Yale University, New Haven, Conecticut.

Wilensky, R. (1983). *Planning and Understanding.* Addison-Wesley, Reading, Massachusetts.

Wilkins, D. E. (1980). Using patterns and plans in chess. *Artificial Intelligence, 14*(2), 165–203.

Wilkins, D. E. (1988). *Practical Planning: Extending the AI Planning Paradigm.* Morgan Kaufmann, San Mateo, California.

Wilkins, D. E. (1990). Can AI planners solve practical problems?. *Computational Intelligence, 6*(4), 232–246.

Wilkins, D. E., Myers, K. L., Lowrance, J. D., and Wesley, L. P. (1995). Planning and reacting in uncertain and dynamic environments. *Journal of Exprimental and Theoretical AI, 7*(1), 197–227.

Wilks, Y. (1975). An intelligent analyzer and understanding of English. *Communications of the ACM, 18*(5), 264–274.

Williams, R. J.(1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning, 8,* 229–256.

Williams, R. J. and Baird, L. C. I. (1993). Tight performance bounds on greedy policies based on imperfect value functions. Tech. rep. NU-CCS-93-14, College of Computer Science, Northeastern University, Boston.

Wilson, R. A. and Keil, F. C. (Eds.). (1999). *The MIT Encyclopedia of the Cognitive Sciences.* MIT Press, Cambridge, Massachusetts.

Winograd, S. and Cowan, J. D. (1963). *Reliable Computation in the Presence of Noise.* MIT Press, Cambridge, Massachusetts.

Winograd, T. (1972). Understanding natural language. *Cognitive Psychology, 3*(1), 1–191.

Winston, P. H. (1970). Learning structural descriptions from examples. Technical report MAC-TR-76, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Winston, P. H. (1992). *Artificial Intelligence* (Third edition). Addison-Wesley, Reading, Massachusetts.

Wirth, R. and O'Rorke, P. (1991). Constraints on predicate invention. In *Machine Learning: Proceedings of the Eighth International Workshop (ML-91),* pp. 457–461, Evanston, Illionis. Morgan Kaufmann.

Witten, I. H. and Bell, T. C. (1991). The zero frequency problem: Estimating the probabilistic of novel events in adaptive text compression. *IEEE Transactions on Information Theory, 37*(4), 1085–1094.

Witten, I. H., Moffat, A., and Bell, T. C. (1999). *Managing Gigabytes: Compressing and Indexing Documents and Images* (second edition). Morgan Kaufmann, San Mateo, California.

Wittgenstein, L. (1922). *Tractatus Logico-Philosophicus* (second edition). Routledge and Kegan Paul, London. Reprinted 1971, edited by D. F. Pears and B. F. McGuinness. This edition of the English translation also

Created by XMLmind XSL-FO Converter.

contains Wittgenstein's original German text on facing pages, as well as Bertrand Russell's introduction to the 1922 edition.

Wittgenstein, L. (1953). *Philosophical Investigations.* Macmillan, London.

Wojciechowski, W. S. and Wojcik, A. S. (1983). Automated design of multiple-valued logic circuits by automated theorem proving techniques. *IEEE Transactions on Computers, C-32*(9), 785– 798.

Wojcik, A. S. (1983). Formal design verification of digital systems. In *ACM IEEE 20th Design Automation Conference Proceedings,* pp. 228– 234, Miami Beach, Florida. IEEE.

Wood, M. K. and Dantzig, G. B. (1949). Programming of interdependent activities. i. general discussion. *Econometrica, 17,* 193–199.

Woods, W. A. (1973). Progress in natural language understanding: An application to lunar geology. In *AFIPS Conference Proceedings,* Vol. 42, pp. 441–450.

Woods, W. A. (1975). What's in a link? Foundations for semantic networks. In Bobrow, D. G. and Collins, A. M. (Eds.), *Representation and Understanding: Studies in Cognitive Science,* pp. 35–82. Academic Press, New York.

Woods, W. A. (1978). Semantics and quantification in natural language question answering. In *Advances in Computers.* Academic Press.

Wooldridge, M. and Rao, A. (Eds.). (1999). *Foundations of national agency.* Kluwer, Dordrecht, Netherlands.

Wos, L., Carson, D., and Robinson, G. (1964). The unit preference strategy in theorem proving. In *Proceedings of the Fall Joint Computer Conference,* pp. 615–621.

Wos, L., Carson, D., and Robinson, G. (1965). Efficiency and completeness of the set-of-support strategy in theorem proving. *Journal of the Association for Computing Machinery, 12,* 536–541.

Wos, L., Overbeek, R., Lusk, E., and Boyle, J. (1992). *Automated Reasoning: Introduction and Applications* (second edition). McGraw-Hill, New York.

Wos, L. and Robinson, G. (1968). Paramodulation and set of support. In *Proceedings of the IRIA Symposium on Automatic Demonstration,* pp. 276–310. Springer-Verlag.

Wos, L., Robinson, G., Carson, D., and Shalla, L. (1967). The concept of demodulation in theorem proving. *Journal of the Association for Computing Machinery, 14,* 698–704.

Wos, L. and Winker, S. (1983). Open questions solved with the assistance of AURA. In Bledsoe, W. W. and Loveland, D. (Eds.), *Automated Theorem Proving: After 25 Years: Proceedings of the Special Session of the 89th Annual Meeting of the American Mathematical Society,* pp. 71–88, Denver, Colorado. American Mathematical Society.

Wright, S. (1921). Correlation and causation. *Journal of Agricultural Research, 20,* 557–585.

Wright, S. (1931). Evolution in Mendelian populations. *Genetics, 16,* 97–159.

Wright, S. (1934). The method of path coefficients. *Annals of Mathematical Statistics, 5,* 161–215.

Wu, D. (1993). Estimating probability distributions over hypotheses with variable unification. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93),* pp. 790–795, Chambery, France. Morgan Kaufmann.

Wygant, R. M. (1989). CLIPS – a powerful development and delivery expert system tool. *Computers and Industrial Engieneering, 17,* 546–549.

Yamada, K. and Knight, K. (2001). A syntax-based statistical transaction model. In *Proceedings of the Thirty Ninth Annual Conference of the Association for Computational Linguistics,* pp. 228–235.

Created by XMLmind XSL-FO Converter.

Yang, Q. (1990). Formalizing planning knowledge for hierarchical planning. *Computational Intelligence, 6,* 12–24.

Yang, Q. (1997). *Intelligent planning: A decomposition and abstraction based approach.* Springer-Verlag, Berlin.

Yedidia, J., Freeman, W., and Weiss, Y. (2001). Generalized belief propagation. In Leen, T. K. Dietterich, T., and Tresp, V. (Eds.), *Advances in Neural Information Processing Systems 13.* MIT Press, Cambridge, Massachusetts.

Yip, K. M.-K. (1991). *KAM: A System for Intelligently Guiding Numerical Experimentation by Computer.* MIT Press, Cambridge, Massachusetts.

Yngve, V. (1955). A model and an hypothesis for language structure. In Locke, W. N. and Booth, A. D. (Eds.), *Machine Translation of Languages,* pp. 208–226. MIT Press, Cambridge, Massachusetts.

Yob, G. (1975). Hunt the wumpus!. *Creative Computing, Sep/Oct.*

Yoshikawa, T. (1990). *Foundations of Robotics: Analysis and Control.* MIT Press, Cambridge, Massachusetts.

Young, R. M. Pollack, M. E., and Moore, J. D. (1994). Decomposition and causality in partial order planning. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS-94),* pp. 188–193, Chicago.

Younger, D. H. (1967). Recognition and parsing of context-free languages in time n^3. *Information and Control, 10*(2), 189–208.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control, 8,* 338–353.

Zadeh, L. A. (1978). Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems, 1,* 3–28.

Zaritskii, V. S., Svetnik, V. B. and Shimelevich, L. I. (1975). Monte-Carlo technique in problems of optimal information processing. *Automation and Remote Control, 36,* 2015–22.

Zelle, J. and Mooney, R. J. (1996). Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence,* pp. 1050– 1055.

Zermelo, E. (1913). Uber Eine Anwendung der Mengenlehre auf die Theorie des Schachspiels. In *Proceedings of the Fifth International Congress of Mathematicians,* Vol. 2, pp. 501–504.

Zermelo, E. (1976). An application of set theory to the theory of chess-playing. *Firbush News, 6,* 37–42. English translation of (Zermelo 1913).

Zhang, N. L. and Poole, D. (1994). A simple approach to bayesian network computations. In *Proceedings of the 10th Canadian Conference on Artificial Intelligence,* pp. 171–178, Banff, Alberta, Morgan Kaufmann.

Zhang, N. L. and Poole, D. (1996). Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intelligence Research, 5,* 301–328.

Zhou, R. and Hansen, E. (2002). Memory-bounded A* graph search. In *Proceedings of the 15th International Flairs Conference.*

Zhu, D. J. and Latombe, J.-C. (1991). New heuristic algorithms for efficient hierarchical path planning. *IEEE Transaction on Robotics and Automation, 7*(1), 9–20.

Zilberstein, S. and Russell, S. J. (1996). Optimal composition of real-time systems. *Artificial Intelligence, 83,* 181–213.

Zimmermann, H.-J. (Ed.). (1999). *Practical application of fuzzy technologies.* Kluwer, Dordrecht, Netherlands.

Zimmermann, H.-J. (2001). *Fuzzy Set Theory – And Its Applications* (Fourth edition). Kluwer, Dordrecht, Netherlands.

Created by XMLmind XSL-FO Converter.

Zobrist, A. L. (1970). *Feature Extraction and Representation for Pattern Recognition and the Game of Go.* Ph.D. thesis, University of Wisconsin.

Zuse, K. (1945). The Plankalkül. Report 175, Gesellschaft für Mathematik und Datenverarbeitung, Bonn, Germany.

Zweig, G. and Russell, S. J. (1998). Speech recognition with dynamic Bayesian networks. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 173–180, Madison, Wisconsin. AAAI Press.

Created by XMLmind XSL-FO Converter.

D. függelék - Magyar nyelvű szakirodalom

Mint korábban, ehhez a kiadáshoz is mellékeljük a kapcsolódó magyar nyelvű irodalom egy rövid válogatását. Az élvonalbeli magyar kutatók többsége angolul publikál, így ez a válogatás semmiképpen sem tekinthető reprezentatívnak a hazai műhelyek, a művelt területek és a kutatás intenzitása szempontjából. A magyar nyelvű irodalom egyúttal kevésbé kapcsolódik egy szűkebb „szakterülethez" – számos általános, interdiszciplináris, illetve összefoglaló művet vonultatunk fel. A magyar nyelvű szakirodalom megadásánál az a szándék vezérelt minket, hogy az angol nyelvben kevésbé jártas, ám az MI-témák iránt érdeklődő olvasó közvetlen segítséget kapjon olvasmányai folytatásában, ezért nem is törekedtünk arra, hogy a válogatásban helyet kapjanak különböző „belső", nehezen vagy hosszú távon nem garantáltan hozzáférhető oktatási, illetve egyéb anyagok. A magyar nyelvű irodalmi válogatásban megérdemelten hangsúlyozott helyet kaptak a magyar nyelvészeti források. A magyar nyelv feldolgozásáról hitelesen csakis magyar szerzőktől magyarul olvashatunk. A válogatáshoz végül csatoltuk néhány hazai konferencia cikkeit is.

1. Mesterséges intelligencia általánosságban

Borgulya, I.: *Neurális hálók és fuzzy-rendszerek*. Dialóg Campus Kiadó, Budapest, 1998.

Calvin, W. H.: *A gondolkodó agy. Az intelligencia fejlődéstörténete.* Világ Egyetem, Kulturtrade Kiadó, 1997.

Clark, A.: *A megismerés építőkövei.* Osiris Kiadó, Budapest, 1996.

Darab T.: *A gépesített értelem. Vázlatok a mesterséges intelligencia filozófiájához*. Áron, Budapest, 1991.

Dennett, D. C.: *Micsoda elmék. A tudatosság megismerése felé.* Világ Egyetem, Kulturtrade Kiadó, 1996.

Fekete I.–Gregorics T.–Nagy S.: Bevezetés a mesterséges intelligenciába. Heurisztikus gráfkeresés, kétszemélyes játékok, automatikus tételbizonyítás. LSI Oktatóközpont, 1990, 1999.

Futó I. (szerk.): *Mesterséges intelligencia*. Aula Kiadó, Budapest, 1999.

Horváth G. (szerk.): *Neurális hálózatok és műszaki alkalmazásaik*. Egyetemi tankönyv. Műegyetemi Kiadó, Budapest, 1998.

Kóczy T. L.–Tick D.: *Fuzzy rendszerek*. Typotex, Budapest, 2000.

Mérő L.: *Észjárások.* Tercum, Budapest. 1997.

Penrose, R.: *A császár új elméje*. Akadémiai Kiadó, Budapest, 1993.

Ruzsa I.: *Klasszikus, intenzionális és modális logika*. Akadémiai Kiadó, Budapest, 1984.

Sántáné-Tóth E.: *Tudásalapú technológia, szakértő rendszerek*. Dunaújvárosi Főiskola Kiadó Hivatala, Dunaújváros, 1995, 1997, 1998, 2000.

Yoshiaki Shirai, Jun-ichi Tsujii: *Mesterséges intelligencia. Alapelvek, alkalmazások.* Novotrade Rt. Budapest, 1987.

Az ÚjALAPLAP tematikus MI-sorozatában megjelent népszerűsítő dolgozatok, 1992. február-1996. augusztus közötti számokban

Álló G.: *Látni és láttatni* I-VI. 1993(2), 25–27., 1993(3), 21–22. 25., 1993(4), 22, 25–26.,1993(5), 21–22, 25., 1993(6), 21–22, 25–26.,1993(7), 20–22.

Aszalós L.: *A fuzzy vezéreljen titeket – Az igen és a nem közötti világ*. 1996(3), 38–40.

Aszalós L.: *Sejtek, „automatizált" sejtek, sejtautomaták – Mesterséges élet?* 1996(7), 42–44.

Created by XMLmind XSL-FO Converter.

Aszalós L.: *A sakk hatása a kutatásra – A játékfák az égig nőnek*. 1996(5), 51–52.

Aszalós L.: *Nyelviskola – A Prolog mint következtető rendszer*. 1995(8), 45–47.

Aszalós L.: *Mesterséges tanácsadók – Szupergráf a házibulhoz*. 1996(4), 38–40.

Aszalós L.: *Izgalmas gondolkodásutánzatok – Logika és tömörítés*. 1996(8), 36–40.

Bessenyei Z., Tomcsányi T.: *Turbógenerátor-rezgésdiagnosztika*. 1992(9), 42.

Borsodi G.–Lois L.: *Előkészületek a beszédfelismeréshez* I-II. 1995(2), 39–40., 1995(3), 34–36.

Farkas E.: A *világmodellek korlátos világa* I-II. 1993(10), 25–26., 1993(11), 45–46.

Farkas Z.: *A rendszer mint diagnoszta. Növény- és tudásnövekmény-védelem*. 1994(1), 37.

Farkas Zs.: *MProlog-alapon*. 1992(9), 38–39.

Fekete I.: *Tudásreprezentáció* I-II. 1992(6), 23.–25., 1992(7), 19–21.

Futó I.: *Az EXPO kockázata*. 1992(9), 40–41.

Galántai Z.: *Egyszerű szabályok bonyolult következményekkel – Az életjáték is lehet szenvedély*. 1996(7), 45–47.

Horváth I.: *Intelligens CAD rendszerek* I-II. 1994(9), 27–28., 1995(12), 27–29.

Horváth I.: *Tervezői stratégiák és eljárások. Vannak még problémák*. 1995(2), 34–37.

Horváth I.: *A tervezés szakértőrendszerei* I-II. 1995(4), 50–51., 1995(5), 51–52.

Horváth I.: *Az alaksajátosság paradigma – Nem elég tervezni, kivitelezni is kell!* 1995(6), 50–51.

Horváth I.: *A lényeg burkoltan – Sajátos alakfelismerés.* 1995(7), 50–51.

Horváth I.: *Tanulni nehéz – de valahogy kell. A rendszer intelligenciája*. 1995(8), 48–50.

Istenes P.: *Karakterológia a mátrixrácson. A kézírás felismerése*. 1993(9), 35–37.

Jakab Á.: *A CNN univerzális számítógép és szuperszámítógép-chip. Összefogva a neuro-biológusokkal*. 1994(7), 14–16.

Jakab Á.: *Ember tervez…* 1994(8), 27.

Jakab Á.: *Mert utálunk robotolni…* 1994(2), 37.

Jakab Á.: *Videre necesse est.* 1993(2), 25.

Koch P.: *A „titkos" tudás. Források és kútfők.* 1992(10), 20–21.

Kő A.–Lovrics L.: *Új csillag a közgazdasági modellezésben: Backpropagation*. 1993.

Lindner L.: *Genius 2 – a sakk-király. Gépek lesznek a sakkvilágbajnokok?* 1994(12), 13–15.

Pirkó J.: *3 D(imenzió) – biológiai és optikai szemszögből. A szemező számítógép.* 1993(8), 32–35.

Sántáné-Tóth E.: *Mesterséges intelligencia. Agyszerű és nagyszerű.* 1992(2), 13–14.

Sántáné-Tóth E.: *A tudás minden előtt!* 1992(8), 17–20.

Sántáné-Tóth E.: *Szakértőrendszer-projektek és -alkalmazások.* 1992(9), 37–38.

Seregy L.: *Lektorálva 15 ezertől 15 milliárdig? A magyar szókincs.* 1993(11), 39–40.

Strausz Gy.: *Mesterséges neurális hálózatok* I-II. 1992(12), 18–20.,1993(1), 25–27.

Created by XMLmind XSL-FO Converter.

Strausz Gy.: *Ügynökök és stratégiák; genetikus algoritmusok és kromoszómák - Elosztott intelligens rendszerek.* 1995(12), 51–52.

Szeredi P.: *VAGY- és ÉS- független és függő – Párhuzamos logikai programozás.* 1996(1), 38–40.

Szőts M.: *Automatikus tételbizonyítás.* 1992(5), 20–22.

Tar J.: *Mesterséges mesterek* I-V. 1994(2), 37–39., 1994(3), 35–37., 1994(5), 45–46., 1994(6), 37., 1994(7), 37–38.

Vámos T.: *Ember és ember között.* 1992(4), 44–45.

Váncza J.: *Mit tanulhatunk mi a gép tanulásából? Tud(hat)-e a gép tanulni?* 1992(11), 31–34.

Alkalmazások és határterületek

Álló G.–Hegedűs Gy. és társai: *A Digitális képfeldolgozás alapproblémái.* Akadémiai Kiadó, Budapest, 1989.

Álló G., Főglein J. és társai: *Bevezetés a számítógépes képfeldolgozásba – javított kiadás.* BME Mérnöktovábbképző Intézet, Egyetemi jegyzet, Budapest, 1993.

Bach I.: *Számítástechnikai nyelvészet*. BME, Budapest, 1991.

Bach I., Farkas E., Naszódi M.: *A magyar nyelv elemzése számítógéppel*. SZTAKI Tanulmányok, 199/1987.

Bacz E.–Takács E.: *Kis magyar nyelvtan*. Gondolat– Talentum, Budapest, 1996.

Benkő T.–Lukácsy G.–Szeredi P.: *A Szemantikus Világháló elmélete és gyakorlata*. Typotex, Budapest, 2005.

Berke J.–Hegedűs Gy. és társai: *Digitális képfeldolgozás és alkalmazásai.* Keszthelyi Akadémia Alapítvány, Keszthely, 1996.

Copi XX.–Irving M.–Gould, J. A.: *Kortárstanulmányok a logikaelmélet kérdéseiről*. Gondolat, Budapest, 1985

Csibi S. (szerk.): *Információ közlése és feldolgozása*. Tankönyvkiadó, Budapest, 1986.

Kasztkin, V. N.–Valdkina, L. I.: *Algoritmusok és játékok.* Tankönyvkiadó, Budapest, 1988.

Kenesei I. (szerk.): *A nyelv és a nyelvek*, Akadémiai Kiadó, Budapest, 1995, 2004.

Kovács P. A.: *Sakkprogramozásról mindenkinek*, Novotrade Rt., 1987.

Kovács M.–Olaszy G.–Nikléczy P.–Gósy M.: *Magyar nyelvi beszédtechnológiai alapismeretek*. (600 oldal CD-ROM-on). Szerk.: Olaszy G., Nikol Kiadó, Budapest, 2002.

Lipi G.–Nemes Á.–Novák I.: *A kínai hadseregtől az utazó ügynökig. Gráfok gépközelben.* Novotrade Kiadó, 1990.

Lukács O.: *Matematikai Statisztika.* Műszaki Könyvkiadó, Budapest, 1996

Olaszy G.–Kovács M.–Nikléczy P.–Gósy M.: *Magyar nyelvi beszédtechnológiai alapismeretek*. (600 oldal CD-ROM-on). Szerk.: Olaszy G., Nikol Kiadó, Budapest, 2002.

Pásztorné Varga K.: *Logikai alapozás alkalmazásokhoz. Matematikai logika – számítástudomány.* ELTE Budapest, Egyetemi jegyzet, 1997

Pásztorné Varga K.–Várterész M.: *Matematikai logika alkalmazásszemléletű tárgyalása*. Panem, Budapest, 2003.

Pinker S.: *A nyelvi ösztön. Hogyan hozza létre az elme a nyelvet?* Typotex, 1999, Budapest

Pléh Cs.: *A mondatmegértés a magyar nyelvben*. Osiris, Budapest, 1998.

Prékopa A.: *Valószínűségelmélet műszaki alkalmazásokkal*, Műszaki Könyvkiadó, Budapest, 1980.

Created by XMLmind XSL-FO Converter.

Prószéky G.: *Számítógépes nyelvészet.* SZÁMALK, Budapest, 1989.

Prószéky G.–Kis B.*: Számítógéppel emberi nyelven (Intelligens szövegkezelés számítógéppel).* SZAK Kiadó, Bicske, 1999.

Rathmann J.: *Idegen szavak a filozófiában*. Nemzeti Tankönyvkiadó, Budapest, 1996.

Rényi A.: *Valószínűségszámítás.* Tankönyvkiadó, Budapest, 1973.

Szeghegyi Á.–Velencei J.: *Üzleti döntéshozásra alkalmas tudásalapú döntéstámogató rendszer*. Főiskolai jegyzet, Budapesti Műszaki Főiskola. 2003.

Vetier A.: *Szemléletes mérték- és valószínűségelmélet.* Tankönyvkiadó, Budapest, 1991.

Williams, J. D.: *Játékelmélet*, Műszaki Könyvkiadó, Budapest, 1972.

Folyóiratokban és konferenciakiadványokban megjelent dolgozatok

Benkő T.–Krauth P.: *IQ*SILK: Modell-vezérelt adatintegrálás logikai alapokon*. Alma Mater sorozat, Vállalat, információ, tudomány. BME GTK ITM és KFKI, 2002. aug., 223–244. o.

Biró M.–Mayer, J.–Rapcsák T.–Vermes M.: *Matematikai programozási szakértői rendszerek*. Alkalmazott Matematikai Lapok 16, 1992, 217–278. o.

Bognár K.: *Klasszifikáció az esetalapú következtetőrendszerekben*. Alkalmazott Matematikai Lapok 21, 2004, 317–328. o.

Bognár K.: *Leíró logikák az ismeretábrázolásban*. Alkalmazott Matematikai Lapok 20(2), 2000, 183–193. o.

Csendes, D.–Hatvani, Cs.–Alexin, Z. és társai: *Kézzel annotált magyar nyelvi korpusz*. Első Magyar Számítógépes Nyelvészeti Konferencia (MSZNY) kiadványa, Szeged, 2003. 238–247. o.

Dezsényi Cs.–Varga P.–Mészáros T.–Strausz Gy.– Dobrowiecki T.: *Tudásalapú információkinyerés: Az IKF projekt.* Tudományos és Műszaki Tájékoztatás, Könyvtár- és információtudományi szakfolyóirat, Vol. 51, No. 5, 2004.

Fék M.–Németh G.–Olaszy G.–Gordos G.: *Megértést segítő részletező gépi névfelolvasás Magyar nyelvre*. II. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY) kiadványa, Szeged, 2004. 301–306. o.

Gregorics T.: *Megjegyzések a heurisztikus gráfkereső algoritmusokhoz.* Alkalmazott Matematikai Lapok, 1992, 16., 61–71. o.

Hócza A.: *Teljes mondatszintaxis tanulása és felismerése*. II. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2004) kiadványa, Szeged, 2004. dec. 9–10., 127–136. o.

Lukácsy G.–Benkő T.–Szeredi P.–Krauth P.: *Ontológiakezelés logikai módszerekkel.* Networkshop 2003 kiadványa. Nemzeti Információs Infrastruktúra Fejlesztési Program, 2003.

Lukácsy G.–Szeredi P.: *Ontológiák és alkalmazásuk orvosi területen*. IME – Informatika és Menedzsment az Egészségügyben, III(1), 2004, 26–30. o.

Mikó B.–Szántai M.*: Költségbecslés a szerszámgyártásban.* Gépgyártás 2003/3., 14–18. o.

Németh I.–Püspöki J.: *Szerszámgépek felépítési változatainak tervezése mechatronikai tervező rendszer keretébe.* Gép, LIV(10–11), 2003. 112–117. o.

Olaszy G.–Kiss G.–Németh G.–Olaszi P.: *Profivox: a legkorszerűbb hazai beszédszintetizátor*. In Gósy M. (szerk.): Beszédkutatás 2000, MTA Nyelvtudományi Intézete, 167–179. o.

Olaszy G.–Gordos G.–Németh G.: *A gépi beszédkeltés korszerű módszerei*. Alkalmazott Nyelvtudomány, III(1) 2003, 13–27. o.

Created by XMLmind XSL-FO Converter.

Molnár A.: *Elektromos hajtású robotrepülőgépek*. Haditechnika 2002 – Nemzetközi Szimpózium, 2002. szeptember 16–17, Zrínyi Miklós Nemzetvédelmi Egyetem Bolyai János Katonai Műszaki Főiskolai Kar, Budapest, Bolyai Szemle különsz. II. 21–33. o.

Paczolay, D.–Kocsor, A.–Sejtes, Gy.–Hégely, G.: *A „Beszédmester" csomag bemutatása, informatikai és nyelvi aspektusok*. Alkalmazott Nyelvtudomány, IV(1), 2004, 57–79. o.

Roska T.: *Egyszerű és bonyolult – fogalmak és mértékek mesterséges és élő rendszerekben. Bevezető*. Magyar Tudomány, 48(3), 2003, 290–291. o.

Roska T.: *Bonyolultság és egyszerűség analogikai hullám-számítógépekben és néhány idegi jelenség modelljében*. Magyar Tudomány, 48(3), 2003, 328–331. o.

Roska T.–Bálya D.–Csapodi M.–Zarándy Á.: *Analogikai celluláris számítógépek. Egy új számítógépelv.* Természet Világa, 2000, II. különsz., 71–74. o.

Sántáné-Tóth E.: *Szakértő rendszerek fejlesztésének módszertani és oktatási kérdései.* „Informatika az oktatásban" 2002" Konferencia CD-kiadványa, Debrecen, 2002. aug. 28–30, 6. o.

Starkné Werner Á.: *Szakértői rendszer lehetőségek a kémiaoktatásban*. A kémia tanítása módszertani folyóirat, VI. évf. 1998, 3–4.

Starkné Werner Á.: *Mesterséges intelligenciabeli lehetőségek az internetes és multimédiás földrajz oktatásában*. A földrajz tanítása módszertani folyóirat, XII. évf. 2004(1), 3–7. o.

Starkné Werner Á., Vígh K.: *Döntést támogató szakértői rendszer az aromaterápiában.* XXI. Neumann Kollokvium, Veszprém 1998. nov. 12–14., 200–204. o.

Szeredi P.–Gyimóthy T.: *Logikai programozás és alkalmazásai.* IX. Neumann centenáriumi Kongresszus kiadványa, NJSZT, 2003. 181–198. o.

Tatai G.–Laufer L.: *Nyelvi elemek érzelmi töltetének vizsgálata és felhasználása természetes nyelvi dialógusrendszerben.* Magyar Számítógépes Nyelvészet Konferencia (MSZNY), 2003. Szeged, 102–107. o.

Tatai G.–Laufer L.–Csordás A.–Fegyó T.: *Interaktív, szintetikus tanárok az e-oktatásban*. „Multimédia az oktatásban – Fókuszban az e-learning gyakorlata" Konf. kiadványa, 2004. Szeged, 27–29. o.

Tóth L.–Kocsor A.: *Az MTBA magyar telefonbeszéd-adatbázis kézi feldolgozásának tapasztalatai*. In Gósy M. (szerk.): Beszédkutatás 2003. 134–146. o.

Vámos T.: *Fizika – füszisz – információs társadalom*. Fizikai Szemle, 2005/1. „Fizika éve" sz., 8. o.

Velencei J.: *Leírható-e a tudás?* CEO magazin. 2000. 3. sz.

Viharos Zs. J.: *Megmunkált felületek minősítése neurális háló alkalmazásával.* Fiatal Műszakiak Tudományos Ülésszaka, Kolozsvár, 1996. 7–13. o.

Elektronikus források

Kis B.: *Mi van a szavakon túl? Nyelvtani szerkezetek felismerése számítógéppel.*

http://www.mek.iif.hu/porta/szint/tarsad/nyelvtud/mivana/mivana.htm

Naszódi M.: *Nyelvhelyesség-ellenőrzés számítógéppel.*

http://www.mek.iif.hu/porta/szint/tarsad/nyelvtud/nylvhely/nyelvhely.htm

Sántáné-Tóth E. (szerk.): *Magyar Mesterséges Intelligencia Bibliográfia – Válogatás az 1988– 1996 között megjelent publikációkból*. NJSZT– OMIKK, Budapest, 1996. aug. (400 referencia) http://www.mek.iif.hu/porta/szint/egyeb/katalog/mi-bibl

Created by XMLmind XSL-FO Converter.