



# Informed search I.

Artificial intelligence  
Kristóf Karacs  
PPKE-ITK



## Recap


- What is intelligence?
- Agent model
- Problem solving by search
  - Non-informed search strategies
  - Informed search strategies

# Program

- **Problem solving by search**
- Search including other agents
- Machine learning
- Logic and inference
- Search in logic representation, planning
- Inference in case of constraints
- Bayesian networks
- Fuzzy logic

# Outline

- Best-first search
- What information is available?
- Heuristic, heuristic function
- Strategies: UCS, greedy, A\*
- Properties of heuristics
- Designing heuristics
- Comparing search algorithms
- Further informed search strategies
  - IDA\*, RBFS, SMA\*



## Search and AI

- Search methods are ubiquitous in AI systems
- An autonomous robot uses search
  - to decide which actions to take and which sensing operations to perform,
  - to quickly anticipate collision,
  - to plan trajectories,
  - to interpret large numerical datasets provided by sensors into compact symbolic representations,
  - to diagnose why something did not happen as expected,
  - etc...
- Many searches may occur concurrently and sequentially



## Applications

- Search plays a key role in many applications
  - Route finding: airline travel, networks
  - Package/mail distribution
  - Pipe routing, VLSI routing
  - Comparison and classification of protein folds
  - Pharmaceutical drug design
  - Design of protein-like molecules
  - Video games

## Assumptions in basic search

- World is
  - static
  - discretizable
  - observable
- Actions are deterministic
- In many real world problems these assumptions do not hold →

Extended search techniques are required

## Search strategy

- The **fringe** is the set of all search nodes not yet expanded
- The fringe is implemented as a priority queue
  - `insert( $n$ ,  $Q$ )`
  - `remove( $Q$ )`
- The ordering of the nodes in the queue defines the search strategy

## Revisiting states

- Most search strategies have two versions
  - States *may be revisited*
  - States *may not be revisited*
- Implementations
  - Flag for each state
  - Visited list
- Not appropriate for all strategies

## Best-first search

- Which node is good?
- $f()$  : evaluation function (typically cost function)
- Selection criteria: minimal value of  $f()$
- Note: “Best” does not guarantee optimality of the solution path

## Properties of best-first search

- If the state space is *infinite*, then in general the search is *not complete*
- If the state space is *finite* and revisited states are *not discarded*, then in general the search is *not complete*
- If the state space is *finite* and revisited states are *discarded*, then the search is *complete*, but in general it is *not optimal*

## Search algorithm

- `insert(initial-node, Q)`
- **Cycle**
  - If `Q` is empty then return failure
  - `n ← remove(Q)`
  - `s ← state(n)`
  - If `is-goal(s)` then return `s` and/or path
  - For every state `s'` in `succ(s)`
    - Create a node `n'` as a successor of `n`
    - `insert(n', Q)`

## Using information in search

- Intelligence: situation evaluation
- Cost of an action
  - Distance in route planning
  - Power consumption
- Path cost
  - $g()$  : Sum of all action costs in the path

## Defeating exponential blow up

- Decreasing the number of actions in a given state (policy function)
- Decreasing search depth (value function)
- Monte Carlo tree search...

## Heuristic searches

- Heuristic = Rule of thumb
  - Different from heuristic measures
  - Influences the node to expand
- Values that can help
  - Path cost  $g()$
  - Heuristic measures  $h()$

## Heuristic Function

- $h(\text{node})$ 
  - Estimates path cost of reaching the solution
  - Independent of the actual search tree
  - $h(\text{goal state}) = 0$
- Methods to derive a heuristic function
  - Mathematically
  - By introspection
  - Inspection of particular searches
  - Computer programs (e.g.: Absolver)
- Example: straight line distance

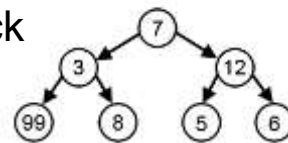


## Uniform Cost Search (non-informed)

- ~BFS
- Expands node with smallest  $g()$   
(ignores heuristic measures)
- Finds a solution with least cost
  - Condition: action costs must be positive
- Optimal and complete
- Can be very slow

## Greedy Search

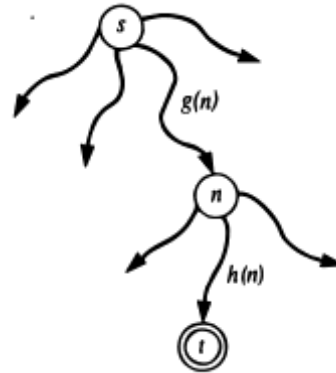
- Expand node with smallest  $h()$   
(ignores path cost)
- If in a dead-end then backtrack



- Problems
  - Blind alley effect: estimates can be wrong, leading to superfluous curves
  - May lead to non-optimal solution ( $h()$  is only an estimate of path cost to the goal)

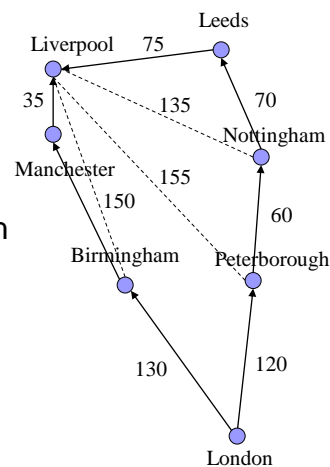
# A\* search

- Combines
  - uniform cost search and
  - greedy search
- $f(n)$  estimates the cost of the best path through  $n$
- $f(n) = g(n) + h(n)$
- Hart, Nilsson and Raphael, 1968



## Example: route finding

- $g(n)$  = distance from London
- $h(n)$  = straight line distance to Liverpool
- $f(n) = g(n) + h(n)$
- 1<sup>st</sup> round: Birmingham, Peterborough
  - $f(\text{Peterborough}) = 120 + 155 = 275$
  - $f(\text{Birmingham}) = 130 + 150 = 280$
- Expands Peterborough
- Returns to Birmingham in the next step, because  $120+60+135 > 130+150$



## Properties of heuristics

- A heuristic  $h(n)$  is **admissible** if it never overestimates the path cost from node  $n$  to the goal node, i.e.  $0 \leq h(n) \leq h^*(n)$ .
- A heuristic  $h(n)$  is **consistent** (**monotone**) if, for every node  $n$  and every successor  $n'$  of  $n$  generated by any action  $a$ 
$$h(n) \leq c(n, a, n') + h(n').$$
- $h_1$  **dominates**  $h_2$  if  $h_1(n) \geq h_2(n)$ .

## Completeness theorem

- A\* always finds an optimal solution path (even for non-admissible heuristics) if there are finitely many nodes with  $f(n) \leq f^*$ ,  $f^*$  being the cost of the optimal path. This is guaranteed if
  - all action costs  $\geq \epsilon$ , for some fixed  $\epsilon > 0$ , and
  - the branching degree of all nodes are finite
- Proof
  - Let  $f^*$  be the cost of the optimal path
  - All nodes with  $f(n) < f^*$  will get expanded
  - Some further nodes with  $f(n) = f^*$  may get expanded

## Optimality theorems

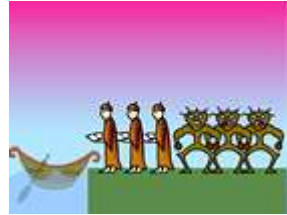
- If  $h(n)$  is admissible, then  $A^*$  is optimal with no visited list
- If  $h(n)$  is consistent, then  $A^*$  is optimal using a visited list
- If  $h(n)$  is admissible, then  $A^*$  is optimally efficient: with any given heuristic no other search strategy expands fewer nodes

## Dominance theorem

- If  $h$  and  $h'$  are heuristic functions and  $h$  dominates  $h'$ , then any node expanded by an  $A^*$  search using  $h$  is also expanded by  $A^*$  using  $h'$
- Thus using a dominant heuristic will result in fewer expanded nodes

## Missionaries and cannibals

- Three missionaries and three cannibals must cross a river, using a boat that can carry at most two.
- Find a sequence of operations that ensures that cannibals never outnumber missionaries on either side of the river!



## Designing heuristics

- Good heuristics can be hard to find
  - Often they are implicit in the problem, such as the Euclidean distance heuristic for route-finding
  - They may be found by relaxing some constraint in the problem
    - 8-puzzle, 15-puzzle: allow two tiles to occupy the same square
    - Missionaries: don't worry about missionaries getting eaten
- Good heuristics can be hard to compute
  - Overall goal: minimizing the total time
  - (Avg. time of computing the heuristic value + node expansion) \* (total no. of nodes expanded during search)
  - Trade off between the branching factor and heuristic complexity

## Comparing search algorithms

- Effective branching factor (ebf):  $b^*$ 
  - Branching rate of a search tree, in which each node has the same number of outgoing edges (BFS)
- Calculation
  - $d$  : depth of solution
  - $N$  : number of nodes expanded
$$N = 1 + b^* + b^{*2} + b^{*3} + \dots + b^{*d} = \frac{b^{*d+1} - 1}{b^* - 1}$$
  - Solve for  $b^*$

## Example: Effective Branching Factor

- Suppose
  - $N = 15$  steps
  - $d = 4$
- Solve:  $\frac{b^{*4+1} - 1}{b^* - 1} = 15$
- Result:  $b^* = 1.57$

## Numerical comparison

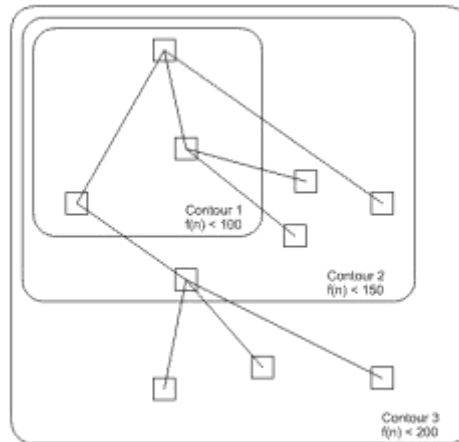
- The shortest solution for the missionaries-and-cannibals problem takes 12 steps

Search strategy	Number of steps	Effective branching factor
BFS	24,464	2.21
A* search, $h_1(x)$ = number of people still on the left bank of the river	1,202	1.67
A* search, $h_2(x)$ : relaxes the requirement that cannibals not outnumber missionaries	40	1.18

## Iterative-Deepening A\* search

- Bottleneck of A\* is *memory* (not time)
  - All visited nodes have to be recorded
- Iterative deepening like in IDS
  - Define contours based on evaluation function
  - Iteratively increase the limit
  - At each iteration use a cutoff value equal to the smallest  $f(n)$  of any node that exceeded the limit in the previous iteration

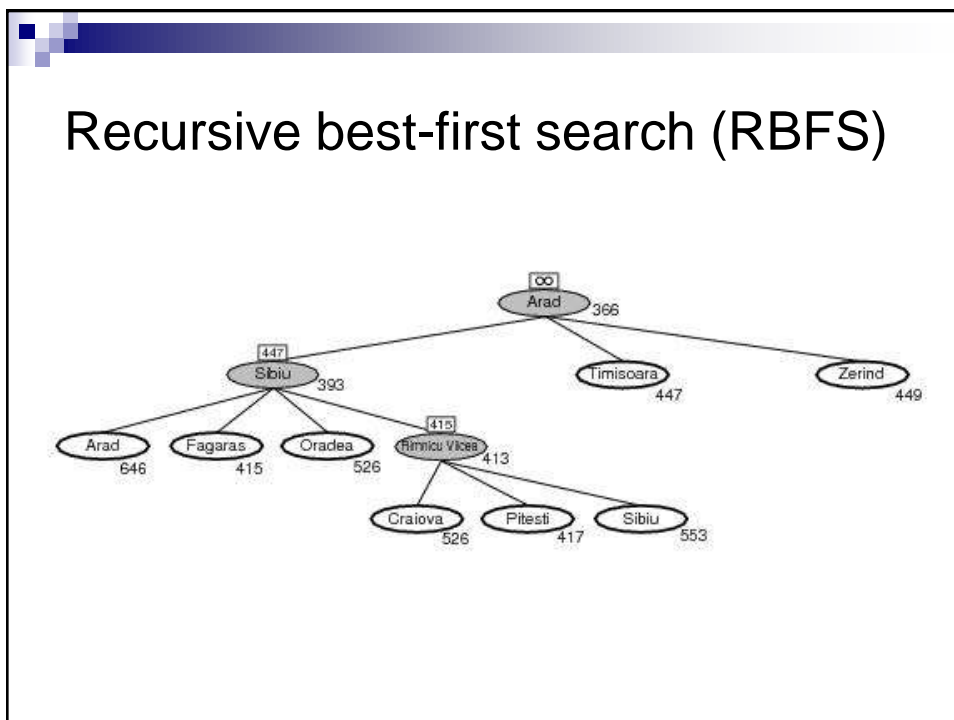
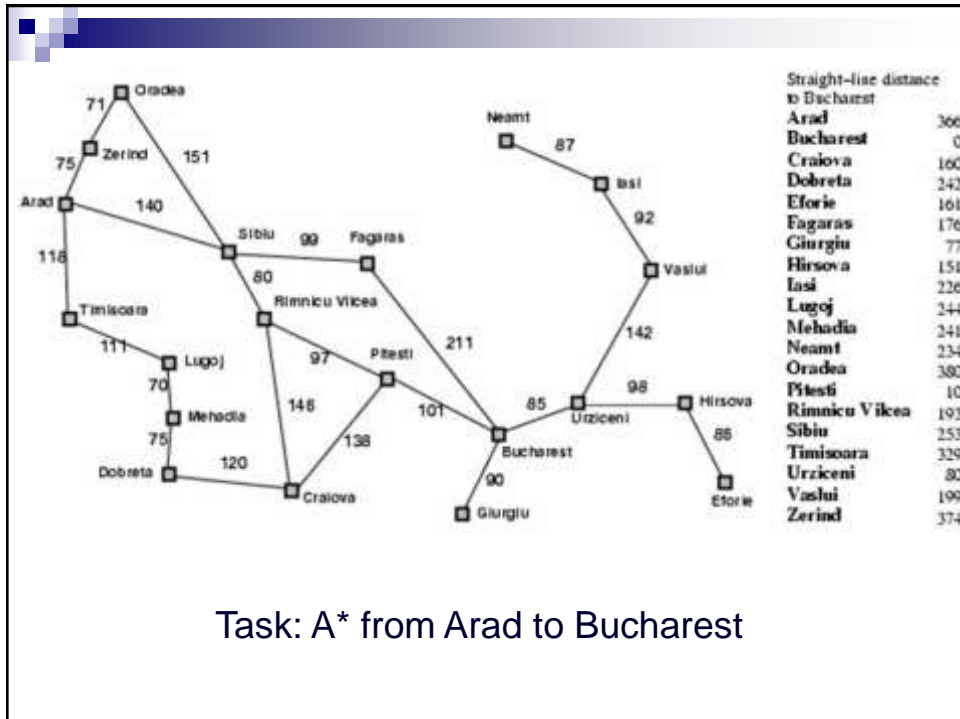
## IDA\* search - contours



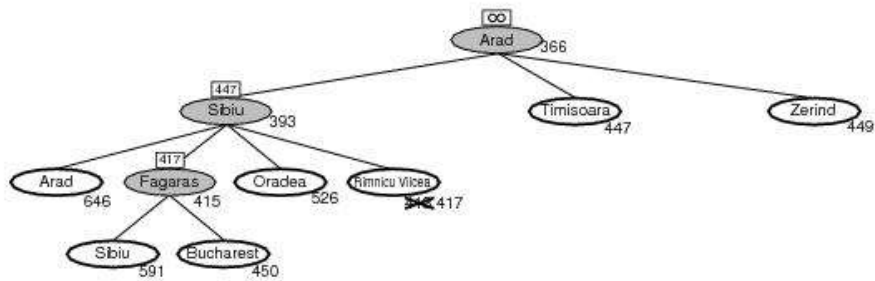
## Recursive best-first search (RBFS)

- Best-first (using  $f()$ )
  - Stores search tree and the best alternative solution for each expanded node
  - If there is a better alternative among the nodes visited earlier  $\rightarrow$  forget the current subtree and continue there
  - When recursion unwinds, replace the  $f()$  value of a node with best  $f()$  value of its children
- Requires linear space

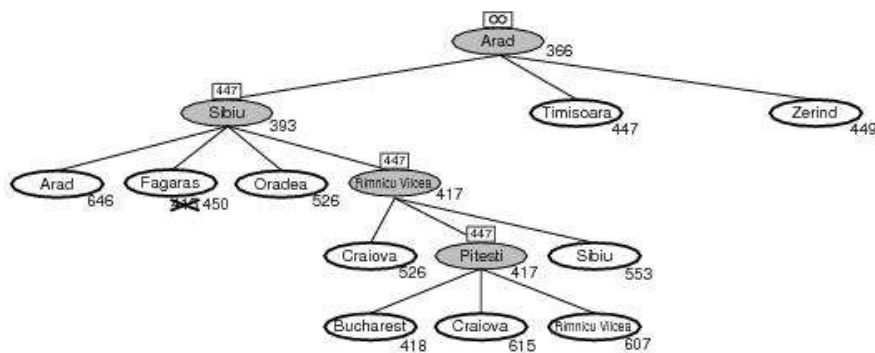




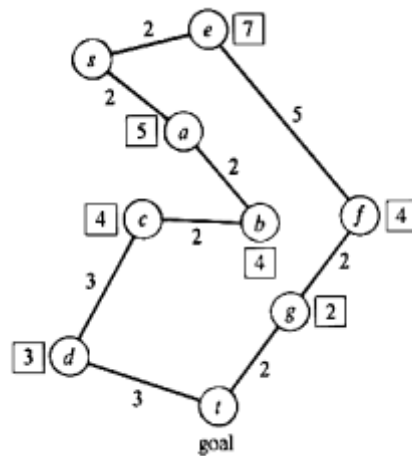
## Recursive best-first search (RBFS)



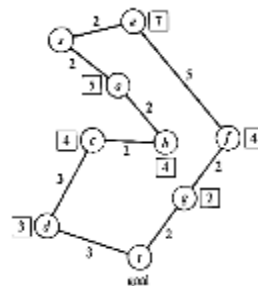
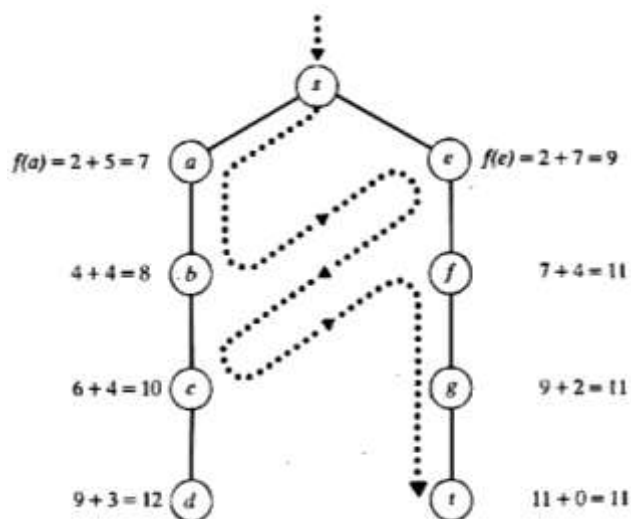
## Recursive best-first search (RBFS)



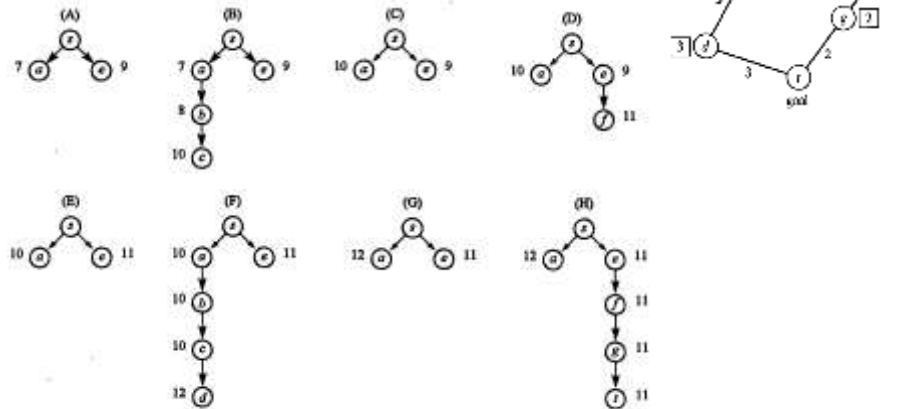
## A\* vs. RBFS example



## A\* search



# RBFS



## RBFS analysis

- More efficient than IDA\* and still optimal
  - Best-first Search based on **next best f() contour**; fewer regeneration of nodes
  - Exploit results of search at a specific f() contour by saving next f() contour associated with a node whose successors have been explored
- Like IDA\* still suffers from excessive node regeneration
- IDA\* and RBFS not good for graphs
  - Can't check for repeated states other than those on current path
- Both are hard to characterize in terms of expected time complexity

## Simplified memory-bounded A\* (SMA\*)

- $B$ : bound on memory
- If memory is full when performing A\*  $\rightarrow$  drop worst leaf node (with lowest  $f()$ ) and back-up the value of the forgotten node to its parent
- If correctly parameterized, SMA\* can solve more complex problems than A\*

## SMA\* analysis

- Complete, if there is any reachable solution
- Optimal, if any optimal solution is reachable
- Problem: if  $B$  is too low  $\rightarrow$  thrashing may occur (among a small set of candidate nodes)



## Summary

- Assumptions and applications
- Best-first search
- Path cost, heuristic function
- Strategies: UCS, greedy, A\*, IDA\*, RBFS, SMA\*
- Admissible, consistent, dominant heuristics
- Designing good heuristics
- Effective branching factor (comparison)