Planning and Scheduling: Informed Search

Prof. Dr.-Ing. Gerhard K. Kraetzschmar

Hochschule
Bonn-Rhein-Sieg

b-it Bonn-Aachen International Center for Information Technology

Acknowledgements

- These slides refer to Chapter 4 of the textbook:
  S. Russell and P. Norvig:
  Artificial Intelligence: A Modern Approach
  Prentice Hall, 2003, 2nd Edition (or more recent edition)
- These slides are an adaptation of slides by Min-Yen Kan
- The contributions of these authors are gratefully acknowledged.

Outline

- Best-first search
- Greedy best-first search
- A* search
- Heuristics
- Local search algorithms
- Hill-climbing search
- Simulated annealing search
- Local beam search
- Genetic algorithms

Review: Tree Search

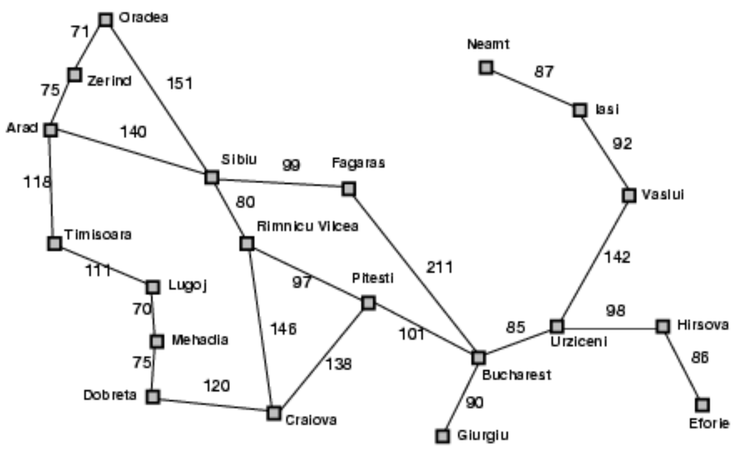
```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
      if the node contains a goal state then return the corresponding solution
      else expand the node and add the resulting nodes to the search tree
```

- A search strategy is defined by picking the order of node expansion

Best-First Search

- Idea: use an evaluation function f(n) for each node
  - Estimate of "desirability"
  - Expand most desirable unexpanded node
- Implementation:
  - Order the nodes in fringe in decreasing order of desirability
- Special cases:
  - Greedy Best-First Search
  - A* Search

Romania With Step Costs In km



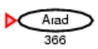
Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

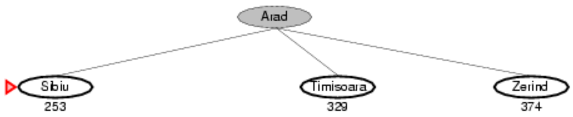
Greedy Best-First Search

- Evaluation function f(n) = h(n) (heuristic)
  - = estimate of cost from n to goal
- For example:
  - $h_{SLD}(n)$ = straight-line distance from n to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal

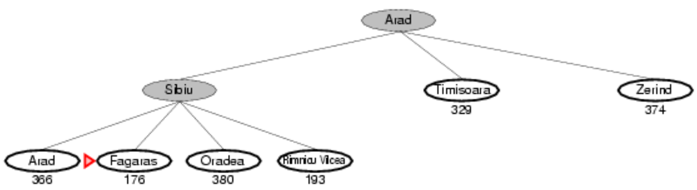
Greedy Best-First Search Example



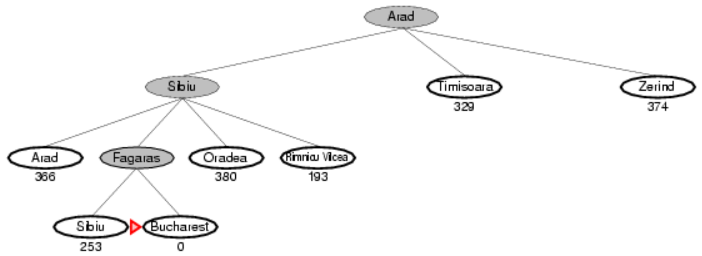
Greedy Best-First Search Example



Greedy Best-First Search Example



Greedy Best-First Search Example



Properties of Greedy Best-First Search

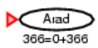
- Complete? No
  - can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt → …
- Time? $O(b^m)$
  - but a good heuristic can give dramatic improvement
- Space? $O(b^m)$
  - keeps all nodes in memory
- Optimal? No

A* Search

- Idea:
  - Avoid expanding paths that are already expensive

- Evaluation function f(n) = g(n) + h(n)
  - g(n) = cost so far to reach n
  - h(n) = estimated cost from n to goal
  - f(n) = estimated total cost of path through n to goal

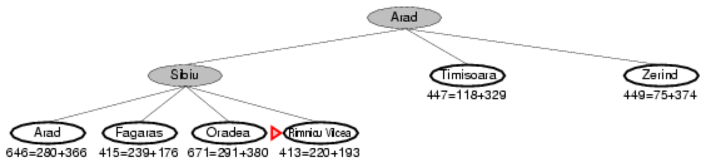
A* Search Example



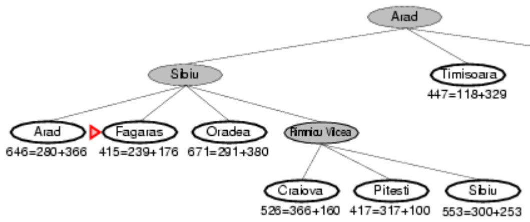
Hochschule
Bonn-Rhein-Sieg
© 2009 Gerhard K. Kraetzschmar

A* Search Example

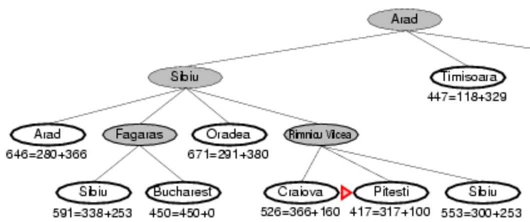
A* Search Example



A* Search Example

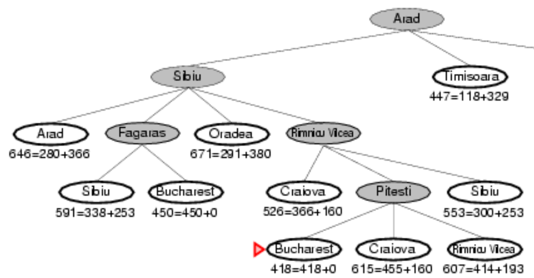


A* Search Example



Hochschule Bonn-Rhein-Sieg
© 2009 Gerhard K. Kraetzschmar

A* Search Example

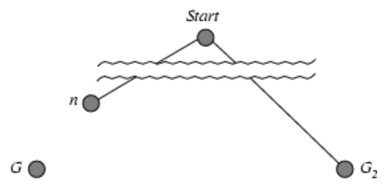


Admissible Heuristics

- A heuristic h(n) is admissible if for every node n, h(n) ≤ h*(n), where h*(n) is the true cost to reach the goal state from n.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic.
- Example: $h_{SLD}(n)$ (SLD = Straight-Line Distance) (never overestimates the actual road distance)
- Theorem: If h(n) is admissible, A* using **TREE-SEARCH** is optimal

Optimality of A* (Proof)

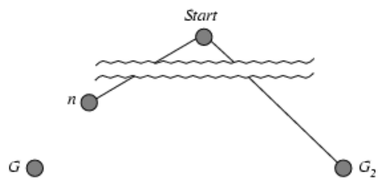
- Suppose some suboptimal goal G2 has been generated and is in the fringe. Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G.



- f(G2) = g(G2) since h(G2) = 0
- g(G2) > g(G) since G2 is suboptimal
- f(G) = g(G) since h(G) = 0
- f(G2) > f(G) from above

Optimality of A* (Proof)

- Suppose some suboptimal goal G2 has been generated and is in the fringe. Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G.



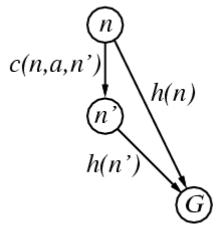
- f(G2) > f(G) from above
- h(n) ≤ h*(n) since h is admissible
- g(n) + h(n) ≤ g(n) + h*(n)
- f(n) ≤ f(G)

- Hence f(G2) > f(n), and A* will never select G2 for expansion

Consistent Heuristics

- A heuristic is consistent
  if for every node n, every successor n' of n generated by any action a,

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



- If h is consistent, we have

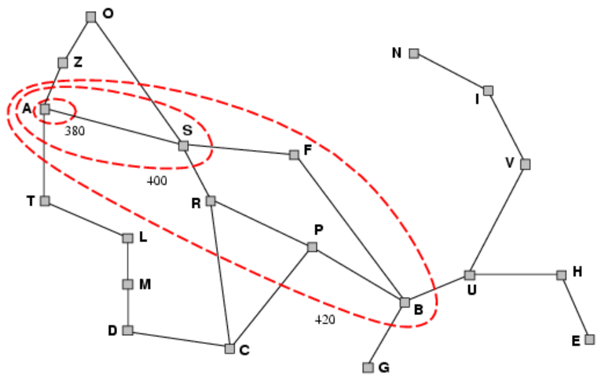
  f(n') = g(n') + h(n')
  = g(n) + c(n,a,n') + h(n')
  ≥ g(n) + h(n)
  = f(n)

  i.e., f(n) is non-decreasing along any path.

- Theorem: If h(n) is consistent, A* using **`GRAPH-SEARCH`** is optimal

Optimality of A*

- A* expands nodes in order of increasing f value
- Gradually adds "f-contours" of nodes
- Contour i has all nodes with $f = f_i$, where $f_i < f_{i+1}$



Properties of A*

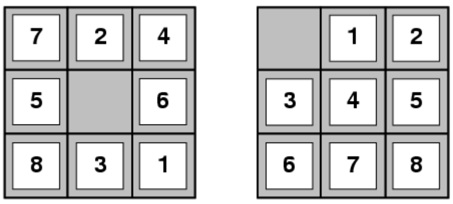
- Complete? Yes
  - (unless there are infinitely many nodes with f ≤ f(G))
- Time? Exponential
- Space? Keeps all nodes in memory
- Optimal? Yes

Admissible Heuristics

E.g., for the 8-puzzle:

- h1(n) = number of misplaced tiles
- h2(n) = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



Start State Goal State

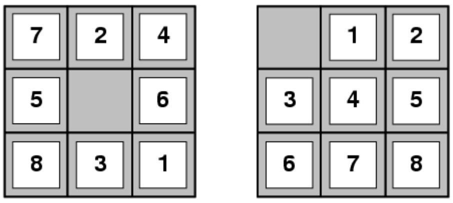
- h1(S) = ?
- h2(S) = ?

Admissible Heuristics

E.g., for the 8-puzzle:

- h1(n) = number of misplaced tiles
- h2(n) = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



Start State Goal State

- h1(S) = ? 8
- h2(S) = ? 3+1+2+2+2+3+3+2 = 18

Dominance

- If h2(n) ≥ h1(n) for all n (both admissible), then h2 **dominates** h1
- If h2 dominates h1, h2 is better for search
- Typical search costs (average number of nodes expanded):
  - d=12 IDS = 3,644,035 nodes
    A*(h1) = 227 nodes
    A*(h2) = 73 nodes
  - d=24 IDS = too many nodes
    A*(h1) = 39,135 nodes
    A*(h2) = 1,641 nodes

Relaxed Problems

- A problem with fewer restrictions on the actions is called a relaxed problem
- **The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem**
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then h1(n) gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square,
  then h2(n) gives the shortest solution

Local Search Algorithms

- In many optimization problems, the path to the goal is irrelevant;
  the goal state itself is the solution
- State space = set of "complete" configurations
- Find configuration satisfying constraints, e.g., n-queens
- In such cases, we can use local search algorithms
- Keep a single "current" state, try to improve it

Example: n-Queens

- Put n queens on an n × n board
  with no two queens on the same row, column, or diagonal

Hill-Climbing Search

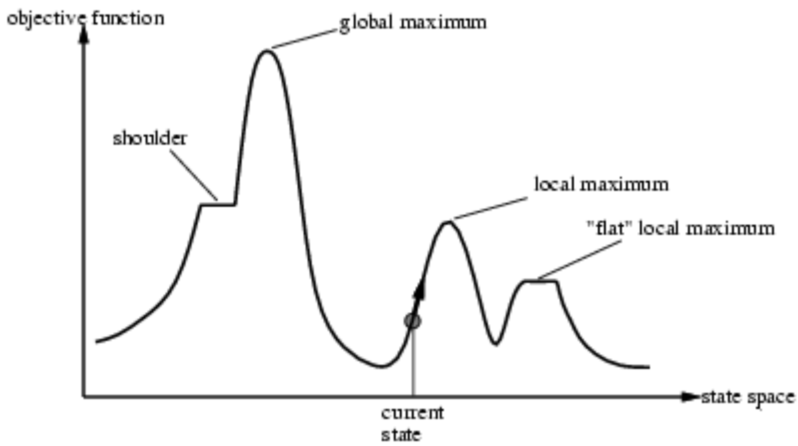
- "Like climbing Mt Everest in thick fog with amnesia"

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum
  inputs: *problem*, a problem
  local variables: *current*, a node
        neighbor, a node

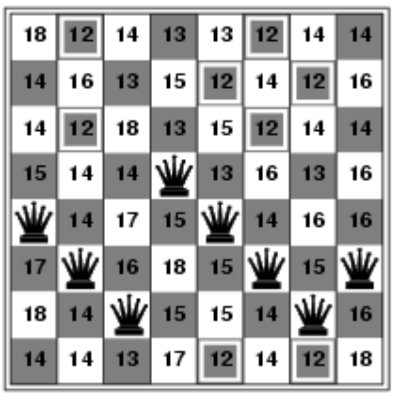
  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  loop do
    neighbor ← a highest-valued successor of *current*
    if VALUE[neighbor] ≤ VALUE[current] **then return** STATE[*current*]
    current ← *neighbor*

Hill-Climbing Search

- Problem: depending on the initial state, we can get stuck in local maxima

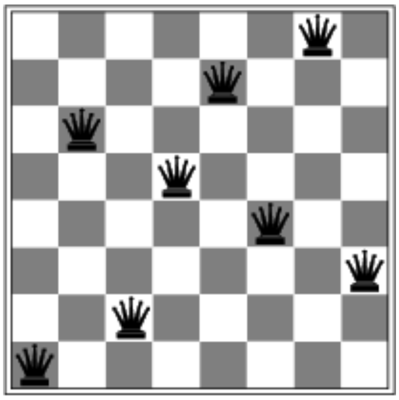


Hill-Climbing Search: 8-Queens Problem



- h = number of pairs of queens that are attacking each other, either directly or indirectly
- h = 17 for the above state

Hill-Climbing Search: 8-Queens Problem



- A local minimum with *h = 1*

Simulated Annealing Search

- Idea:
  - Escape local maxima by allowing some "bad" moves
  - But gradually decrease the frequency of "bad" moves

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
      schedule, a mapping from time to "temperature"
  local variables: *current*, a node
      next, a node
      T, a "temperature" controlling prob. of downward steps

  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  for $t \leftarrow 1$ **to** ∞ **do**
    $T \leftarrow$ *schedule*[t]
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

Properties of Simulated Annealing Search

- One can prove:
  - If T decreases slowly enough,
    then simulated annealing search will find a global optimum with probability approaching 1

- Widely used in VLSI layout, airline scheduling, etc.

Local Beam Search

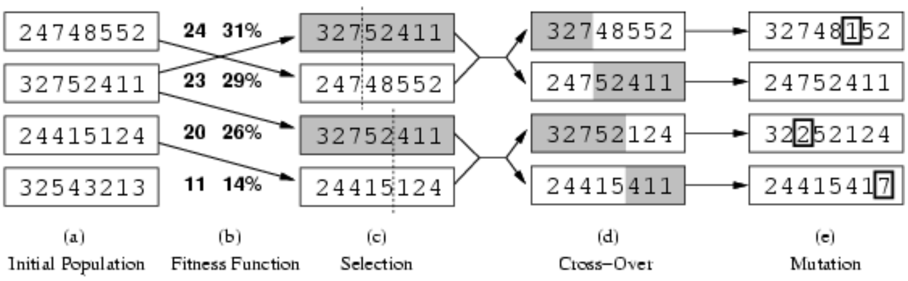
- Keep track of k states rather than just one
- Start with k randomly generated states
- At each iteration, all the successors of all k states are generated
- If any one is a goal state, stop;
  else select the k best successors from the complete list and repeat.

Hochschule
Bonn-Rhein-Sieg
© 2009 Gerhard K. Kraetzschmar

Genetic Algorithms

- A successor state is generated by combining two parent states
- Start with k randomly generated states (a population)
- A state is represented as a string over a finite alphabet
  (often a string of 0s and 1s)
- Evaluation function (fitness function).
  Higher values for better states.
- Produce the next generation of states by
  - Selection,
  - Crossover, and
  - Mutation

Genetic Algorithms



- Fitness function:
  - number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
  - 24/(24+23+20+11) = 31%
  - 23/(24+23+20+11) = 29% etc

Genetic Algorithms