Informed search algorithms

Chapter 4

1

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

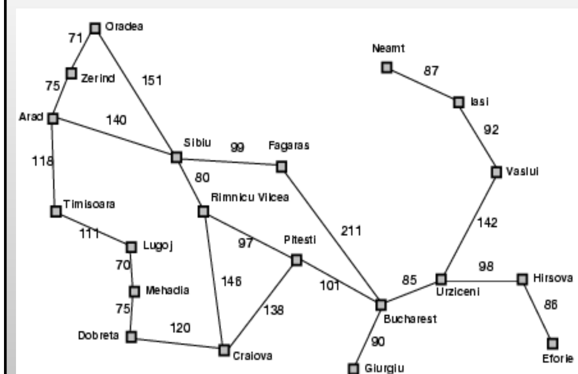
2

Best-first search

- Idea: use an evaluation function *f(n)* for each node
  - estimate of "desirability"
  - → Expand most desirable unexpanded node
- <u>Implementation</u>:
  Order the nodes in fringe in decreasing order of desirability (Priority queue)
- Special cases:
  - greedy best-first search
  - A* search

3

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

4

Greedy best-first search

- Evaluation function
  - $f(n) = h(n)$ (heuristic)

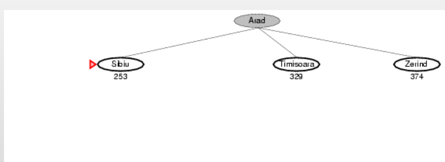
    = estimate of cost from n to *goal*
- e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal

5

Greedy best-first search example

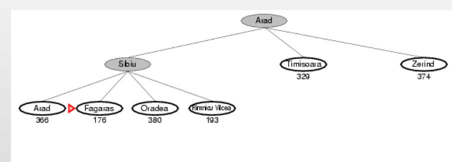
6

Greedy best-first search example



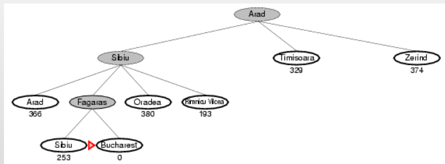
7

Greedy best-first search example



8

Greedy best-first search example



9

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

10

A* search

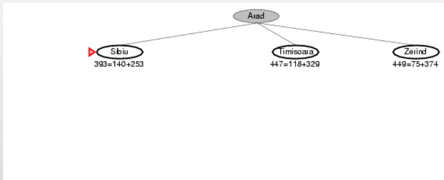
- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n to goal

11

A* search example

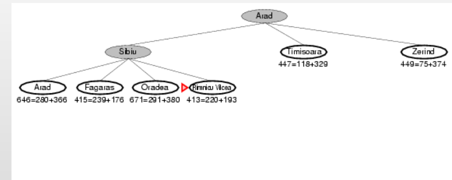
12

A* search example



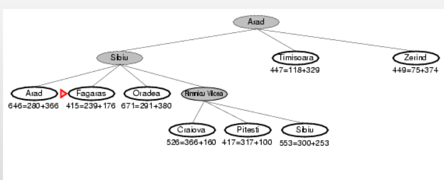
13

A* search example



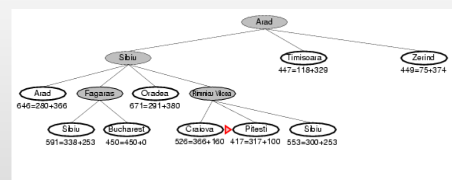
14

A* search example



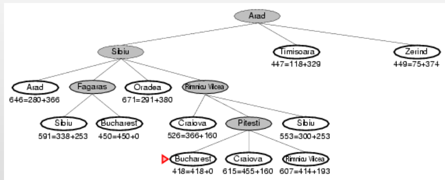
15

A* search example



16

A* search example



17

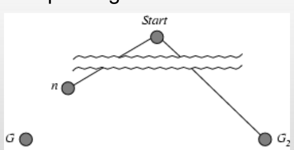
Admissible heuristics

- A heuristic *h(n)* is admissible if for every node *n*, $h(n) \le h^*(n)$, 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: $h_{SLD}(n)$ (never overestimates the actual road distance)
- Theorem: If *h(n)* is admissible, A* using `TREE-SEARCH` is optimal

18

Optimality of A* (proof)

- Suppose some suboptimal goal G_2 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*.

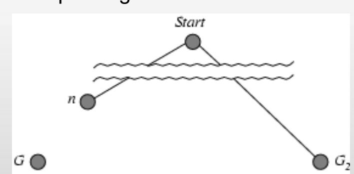


- $f(G_2) = g(G_2)$ — since $h(G_2) = 0$
- $g(G_2) > g(G)$ — since G_2 is suboptimal
- $f(G) = g(G)$ — since $h(G) = 0$
- $f(G_2) > f(G)$ — from above

19

Optimality of A* (proof)

- Suppose some suboptimal goal G_2 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*.



- $f(G_2) > f(G)$ — from above
- $h(n) \le h^*(n)$ — since h is admissible
- $g(n) + h(n) \le g(n) + h^*(n)$
- $f(n) \le f(G)$

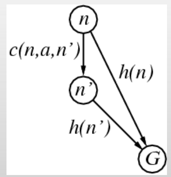
Hence $f(G_2) > f(n)$, and A* will never select G_2 for expansion

20

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

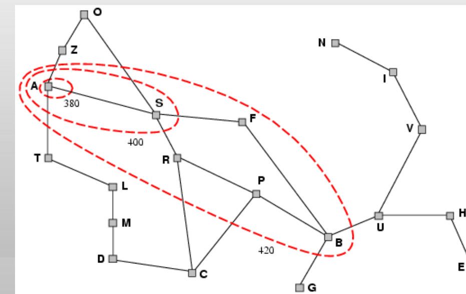
$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n,a,n') + h(n') \\ &\ge g(n) + h(n) \\ &= f(n) \end{aligned}$$

- i.e., $f(n)$ is non-decreasing along any path.
- Theorem: If $h(n)$ is consistent, A* using `GRAPH-SEARCH` is optimal

21

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 $f_i < f_{i+1}$



22

Properties of A*

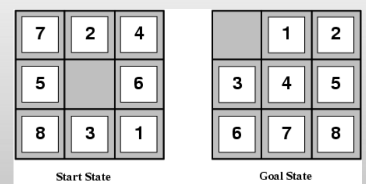
- Complete? Yes (unless there are infinitely many nodes with f ≤ $f(G)$)
- Time? Exponential
- Space? Keeps all nodes in memory
- Optimal? Yes

23

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance
  (i.e., no. of squares from desired location of each tile)



- $h_1(S)$ = ?
- $h_2(S)$ = ?

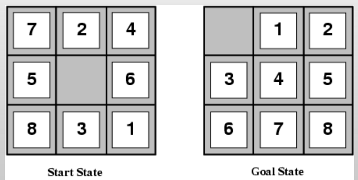
24

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

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



- $h_1(S)$ = ? 8
- $h_2(S)$ = ? 3+1+2+2+2+3+3+2 = 18

25

Dominance

- If $h_2(n) \geq h_1(n)$ for all *n* (both admissible)
- then h_2 dominates h_1
- h_2 is better for search
- Typical search costs (average number of nodes expanded):
- *d=12* IDS = 3,644,035 nodes
  $A^*(h_1)$ = 227 nodes
  $A^*(h_2)$ = 73 nodes
- *d=24* IDS = too many nodes
  $A^*(h_1)$ = 39,135 nodes
  $A^*(h_2)$ = 1,641 nodes

26

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 $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution

27

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

28

Example: *n*-queens

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

29

Hill-climbing search

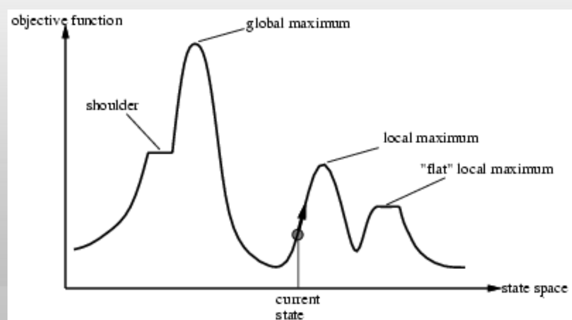
- "Like climbing Everest in thick fog with amnesia"

```
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
```

30

Hill-climbing search

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



31

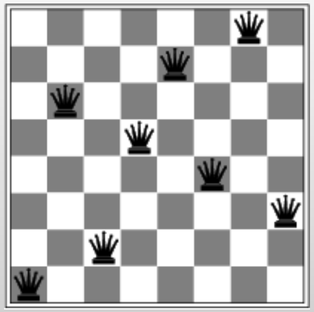
Hill-climbing search: 8-queens problem

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♛	13	16	13	16
♛	14	17	15	♛	14	16	16
17	♛	16	18	15	♛	15	♛
18	14	♛	15	15	14	♛	16
14	14	13	17	12	14	12	18

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

32

Hill-climbing search: 8-queens problem



- A local minimum with *h = 1*

33

Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency

```
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 ← 1 to ∞ do
      T ← schedule[t]
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE[next] − VALUE[current]
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(ΔE/T)
```

34

Simulated Annealing

- Metropolis - 1953:
  simulation of cooling of material in a heath bath;
  - A solid material is heated past its melting point and then cooled back into a solid state (annealing).
  - The final structure depends on how the cooling is performed
  - slow cooling → large crystal (low energy)
  - fast cooling → imperfections (high energy)
- According to thermodynamics: at temperature T, the probability of an increase in energy of ΔE is:

$p(\Delta E) = e^{-\Delta E/kT}$ k is the Boltzmann constant

35

35

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.

36

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.

37