Introduction to Artificial Intelligence

Informed search algorithms

Chapter 4, Sections 1–2, 4

Dieter Fox

Outline

◇ Best-first search

◇ A* search

◇ Heuristics

◇ Hill-climbing

◇ Simulated annealing

Review: General search

```
function GENERAL-SEARCH(problem, QUEUING-FN) returns a solution, or failure
  nodes ← MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[problem]))
  loop do
     if nodes is empty then return failure
     node ← REMOVE-FRONT(nodes)
     if GOAL-TEST[problem] applied to STATE(node) succeeds then return node
     nodes ← QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))
  end
```

A strategy is defined by picking the *order of node expansion*

Best-first search

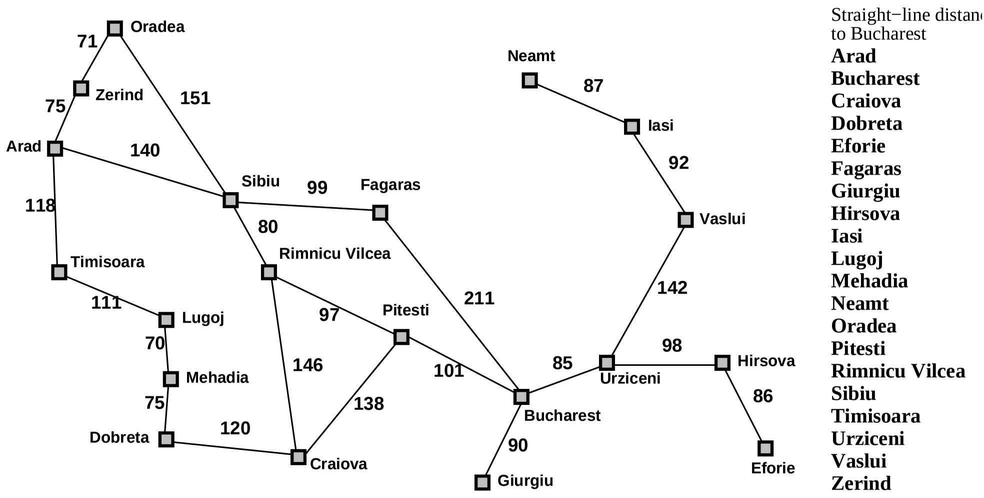
Idea: use an *evaluation function* for each node
– estimate of “desirability”

⇒ Expand most desirable unexpanded node

Implementation:
QUEUEINGFN = insert successors in decreasing order of desirability

Special cases:
greedy search
A* search

Romania with step costs in km



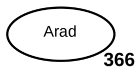
Greedy search

Evaluation function $h(n)$ (**heuristic**)
= estimate of cost from n to $goal$

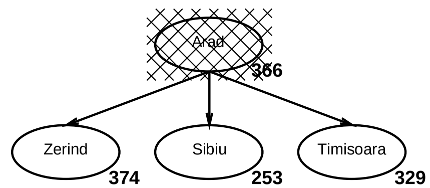
E.g., $h_{\mathrm{SLD}}(n)$ = straight-line distance from n to Bucharest

Greedy search expands the node that *appears* to be closest to goal

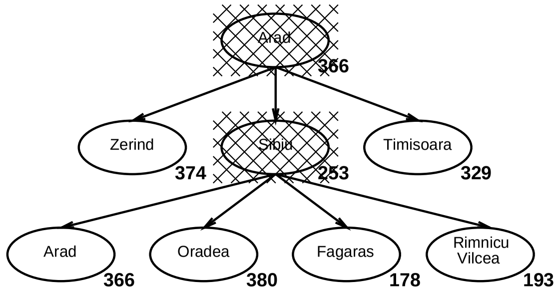
Greedy search example



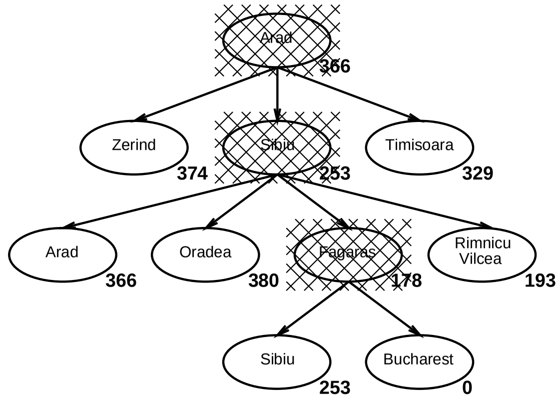
Greedy search example



Greedy search example



Greedy search example



Properties of greedy search

Complete??

Time??

Space??

Optimal??

Properties of greedy search

Complete: No–can get stuck in loops, e.g.,
Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$
Complete in finite space with repeated-state checking

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

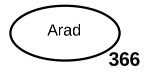
$g(n)$ = cost so far to reach n
$h(n)$ = estimated cost to goal from n
$f(n)$ = estimated total cost of path through n to goal

A* search uses an *admissible* heuristic
i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the *true* cost from n.

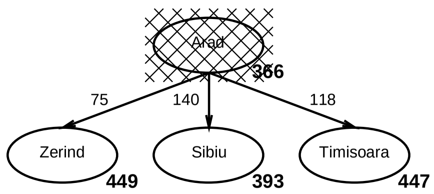
E.g., $h_{\mathrm{SLD}}(n)$ never overestimates the actual road distance

Theorem: A* search is optimal

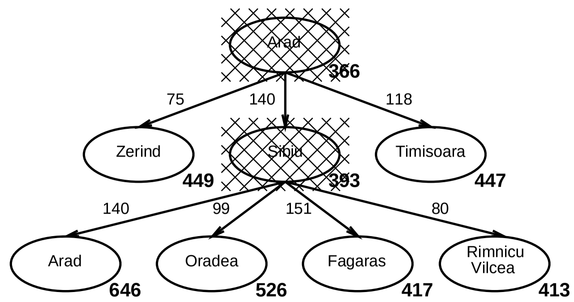
A* search example



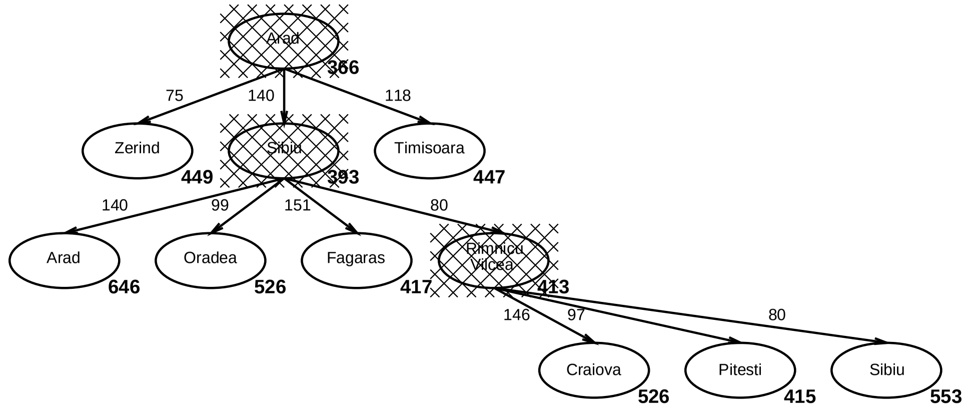
A* search example



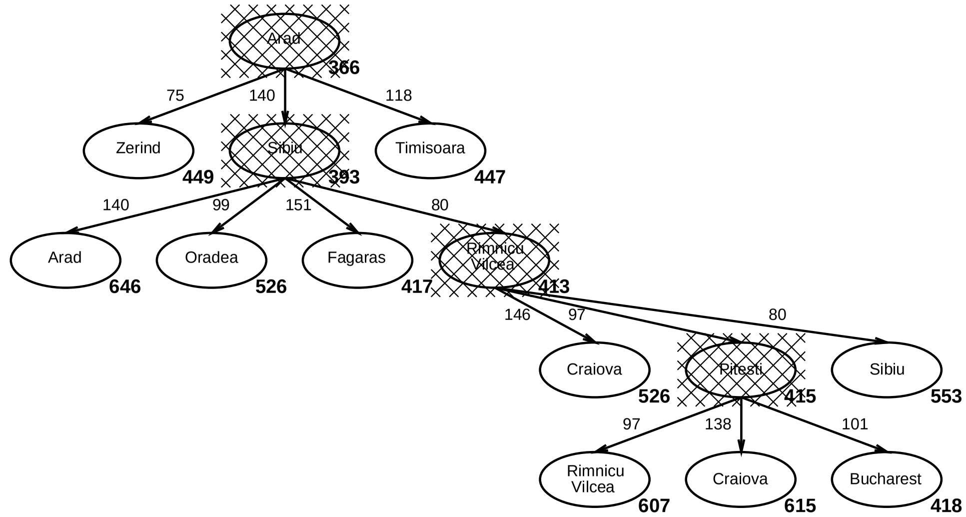
A* search example



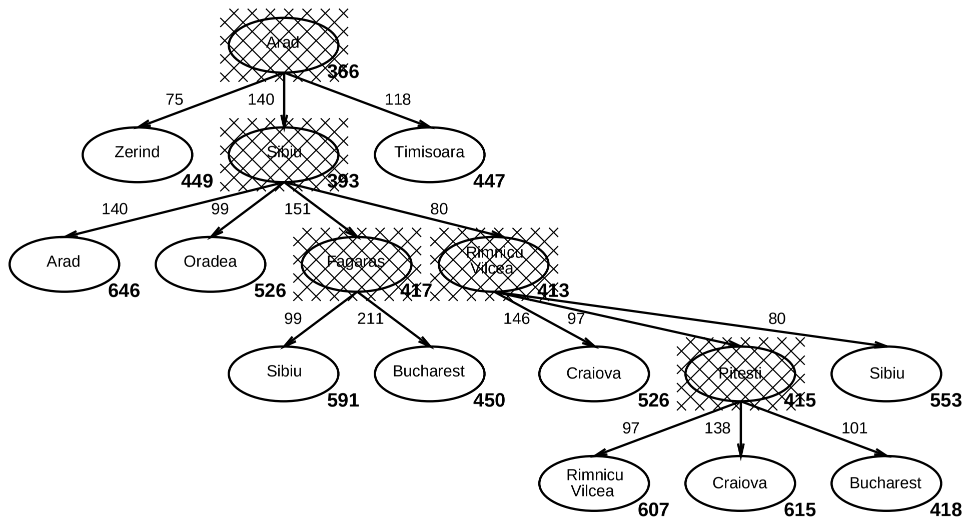
A* search example



A* search example

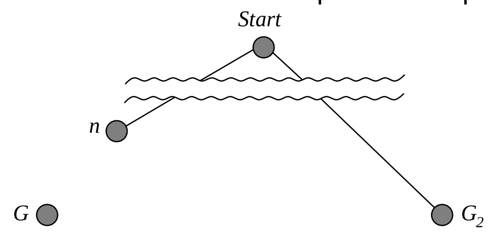


A* search example



Optimality of A*

Suppose some suboptimal goal G_2 has been generated and is in the queue. Let n be an unexpanded node on a shortest path to an optimal goal G_1.



$$
\begin{aligned}
f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\
&> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\
&\geq f(n) && \text{since } h \text{ is admissible}
\end{aligned}
$$

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

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in $h \times$ length of soln.]

Space?? Keeps all nodes in memory

Optimal?? Yes—cannot expand f_{i+1} until f_i is finished

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)

5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5

Goal State

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

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)

5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5

Goal State

$h_1(S)$ =: 7
$h_2(S)$ =: 2+3+3+2+4+2+0+2 = 18

Dominance

If $h_2(n) \geq h_1(n)$ for all n (both admissible)
then h_2 *dominates* h_1 and is better for search

Typical search costs:

$d = 14$ IDS = 3,473,941 nodes
A*(h_1) = 539 nodes
A*(h_2) = 113 nodes
$d = 24$ IDS = too many nodes
A*(h_1) = 39,135 nodes
A*(h_2) = 1,641 nodes

Relaxed problems

Admissible heuristics can be derived from the *exact* solution cost of a *relaxed* version of the 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

Iterative improvement algorithms

In many optimization problems, *path* is irrelevant; the goal state itself is the solution

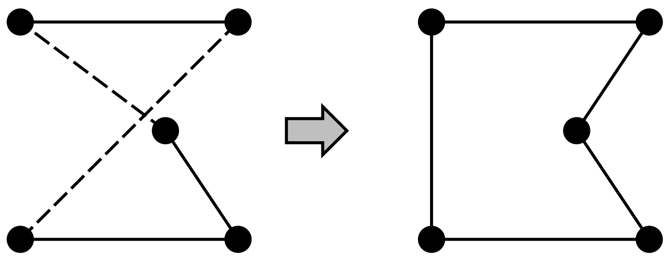
Then state space = set of “complete” configurations;
find *optimal* configuration, e.g., TSP
or, find configuration satisfying constraints, e.g., n-queens

In such cases, can use *iterative improvement* algorithms; keep a single “current” state, try to improve it

Constant space, suitable for online as well as offline search

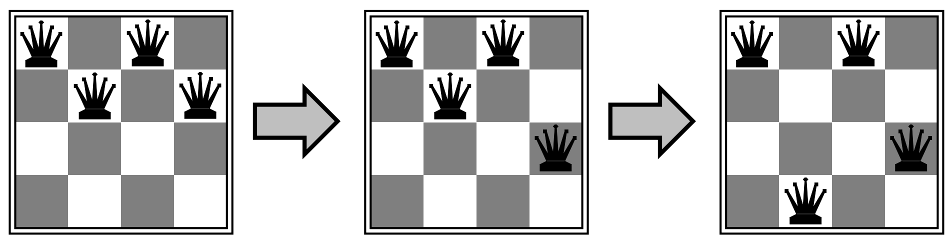
Example: Travelling Salesperson Problem

Find the shortest tour that visits each city exactly once



Example: n-queens

Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal



Hill-climbing (or gradient ascent/descent)

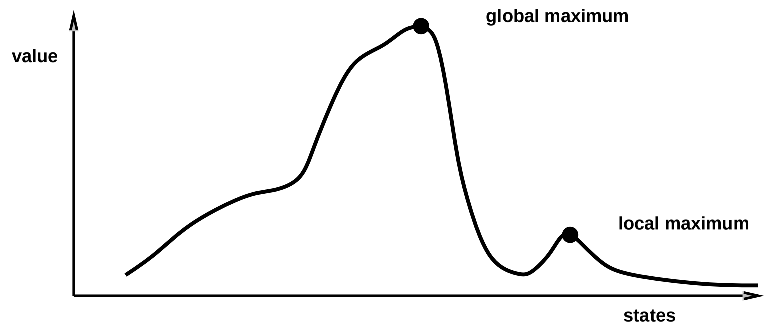
"Like climbing Everest in thick fog with amnesia"

```
function Hill-Climbing(problem) returns a solution state
   inputs: problem, a problem
   local variables: current, a node
                    next, a node

   current ← Make-Node(Initial-State[problem])
   loop do
      next ← a highest-valued successor of current
      if Value[next] < Value[current] then return current
      current ← next
   end
```


Hill-climbing contd.

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



Simulated annealing

Idea: escape local maxima by allowing some "bad" moves
but gradually decrease their size and 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 the probability 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)
```


Properties of simulated annealing

T decreased slowly enough $\Longrightarrow$ always reach best state

Is this necessarily an interesting guarantee??

Devised by Metropolis et al., 1953, for physical process modelling

Widely used in VLSI layout, airline scheduling, etc.