Today

- Uninformed search: summary
- Informed search,
- Search Heuristics

See Russell and Norvig, Chapters 3,4

Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes*	Yes*	No	Yes, if $l \geq d$	Yes
Time	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes*	No	No	Yes

Here * indicates conditions stated earlier.

Graph search

The state space with actions leading from state to state corresponds naturally to a **graph** rather than a **tree**; the state appears only once in the graph.

There are data structures corresponding to graphs, and graph search algorithms that avoid repetition of states already seen.

The idea is to keep track of nodes that have already been expanded; if search arrives back at such a node, it is ignored in future search.

See Russell and Norvig for details.

Informed search strategies

Informed strategies use heuristic "rule of thumb" ideas to guide search based on some estimation of where the solution is most likely to be found.

We look at some such strategies:

- Best-first search
- A* search
- Heuristics

Reminder: Searching a Tree

function TREE-SEARCH(*problem*, *fringe*) **returns** a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]), *fringe*)
  loop do
    if *fringe* is empty **then return** failure
    node ← REMOVE-FRONT(*fringe*)
    if GOAL-TEST[*problem*] applied to STATE(*node*) succeeds **return** *node*
    fringe ← INSERTALL(EXPAND(*node*, *problem*), *fringe*)

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:
fringe is a queue sorted in decreasing order of desirability

Special cases:
  greedy search
  A* search

Romania with step costs in km

Use a straight line heuristic: distance to goal in straight line.

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

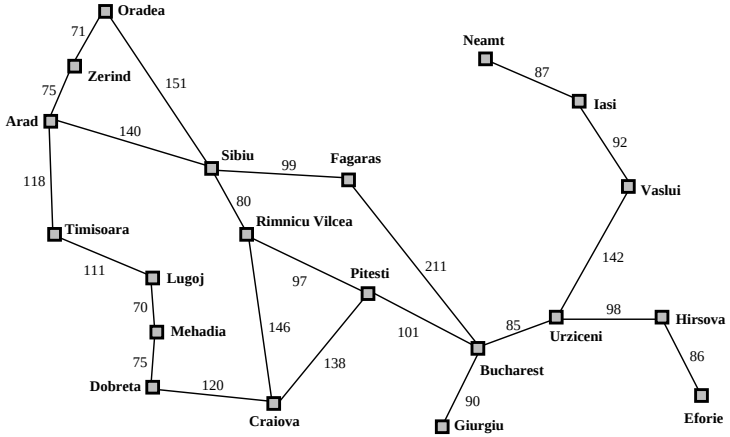
Greedy search

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

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

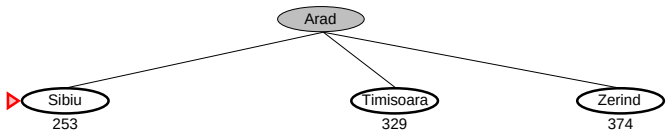
Greedy search expands the node that **appears** to be closest to goal
prefer the action that takes us to the state with the **minimum** heuristic cost.

Example: Romania

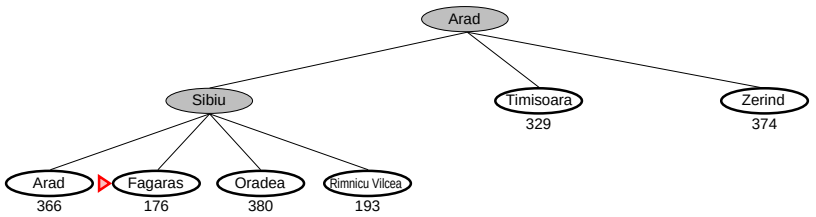


Greedy search example

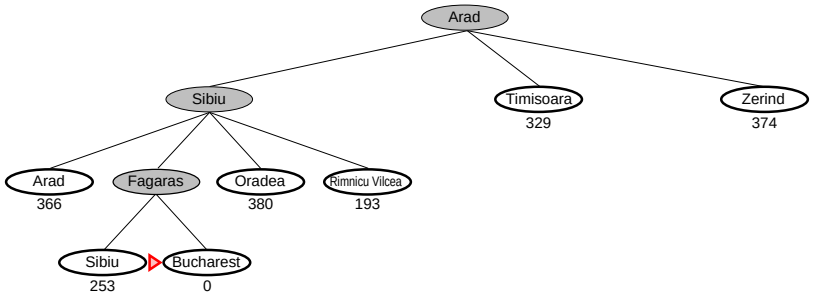
Greedy search example



Greedy search example



Greedy search example



Properties of greedy search

Complete?? No–can get stuck in loops, e.g., with Oradea as goal,
Iasi → Neamt → Iasi → Neamt →
Complete in finite space with repeated-state checking

Time??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,
Iasi → Neamt → Iasi → Neamt →
Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,
Iasi → Neamt → Iasi → Neamt →
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??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,
Iasi → Neamt → Iasi → Neamt →
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 of **cheapest** solution from n.
(Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G.)

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

A* search

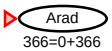
A* search has a very good property:

A* search is **optimal**!

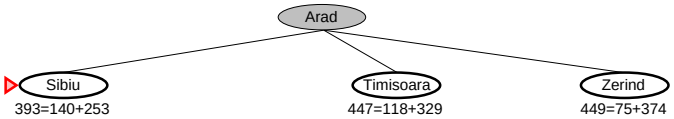
So if there is any solution, A* search is guaranteed to find a least cost solution.

Remember, this needs an **admissible** heuristic.

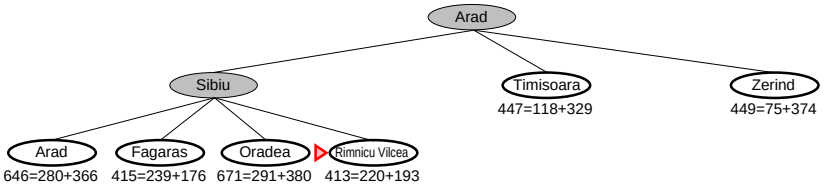
A* search example



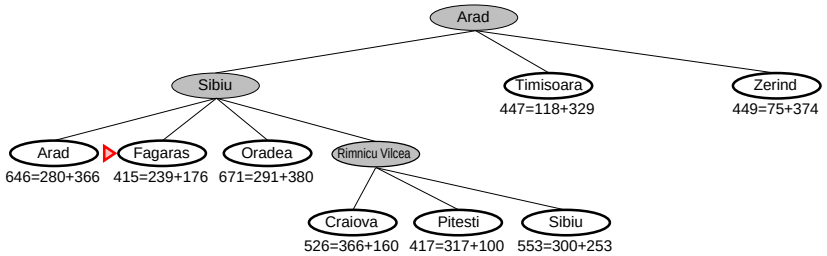
A* search example



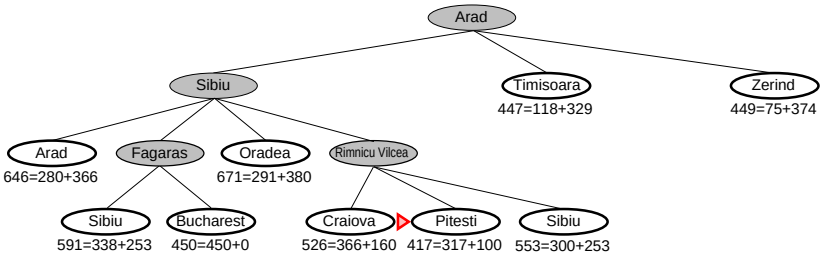
A* search example



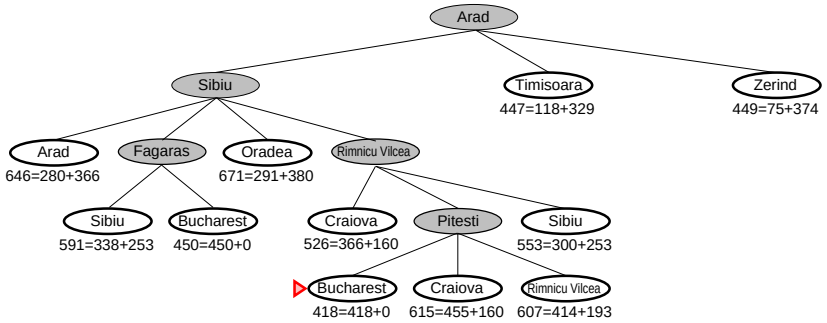
A* search example



A* search example

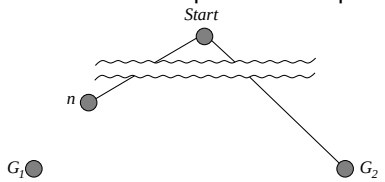


A* search example



Optimality of A* (standard proof)

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.



Then:

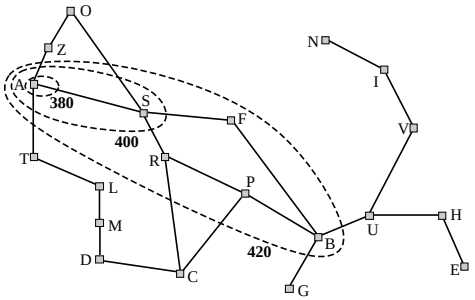
$$\begin{aligned} f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\ &> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\ &\geq f(n) && \text{since } h \text{ is admissible} \end{aligned}$$

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

Optimality of A* (more useful)

Lemma: A* expands nodes in order of increasing f value

Gradually adds "f-contours" of nodes (cf. breadth-first adds layers)
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 $f \leq f(G)$

Time??

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??

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

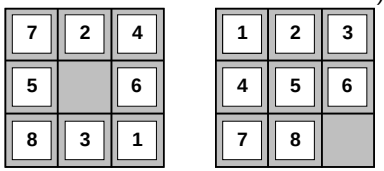
Optimal?? Yes—cannot expand f_{i+1} until f_i is finished

A* expands all nodes with $f(n) < C^*$
A* expands some nodes with $f(n) = C^*$
A* expands no nodes with $f(n) > C^*$

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)

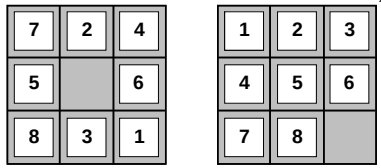


$h_1(S)$ =??
$h_2(S)$ =??

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)



$h_1(S)$ =?? 6
$h_2(S)$ =?? 4+0+3+3+1+0+2+1 = 14

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 for solution at length d:

$d = 14$	IDS = 3,473,941 nodes
	$A^*(h_1)$ = 539 nodes
	$A^*(h_2)$ = 113 nodes
$d = 24$	IDS $\approx$ 54,000,000,000 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

Key point: the optimal solution cost of a relaxed problem
is no greater than the optimal solution cost of the real problem

Summary

Work with an evaluation function for each node

- Greedy search algorithms – go for most desirable node
- A^* search using an admissible heuristic function
- A^* is optimal search strategy
- Heuristics for the eight puzzle