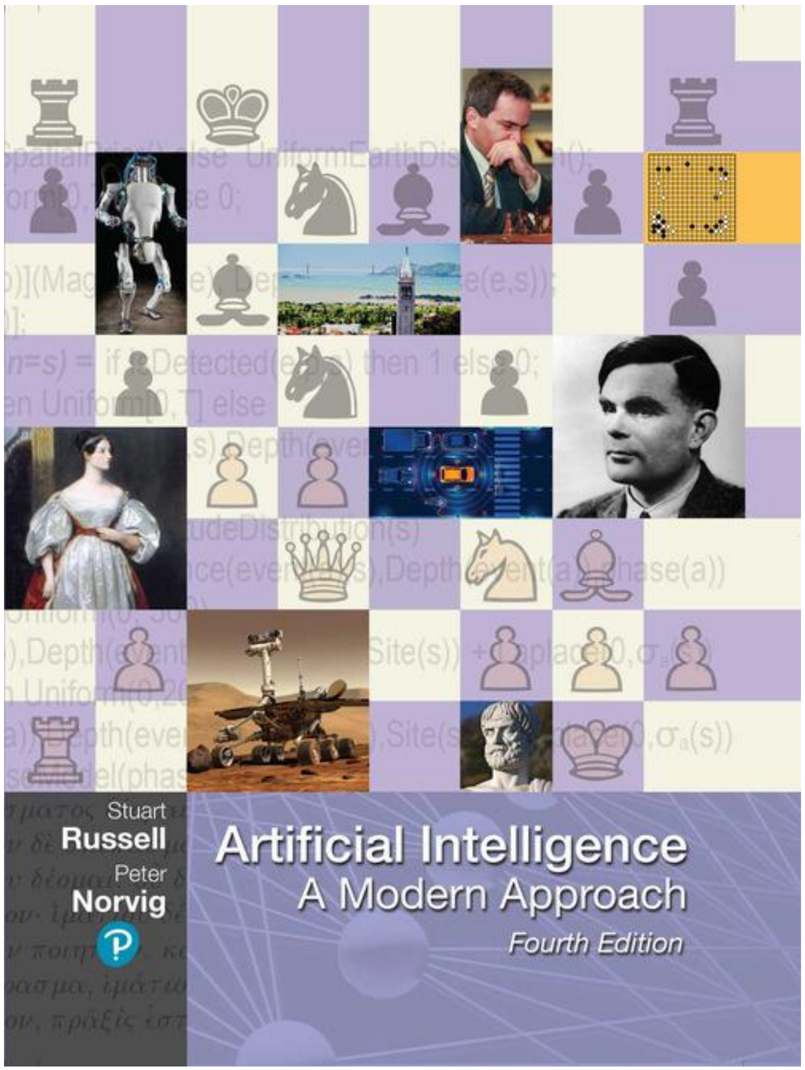
Artificial Intelligence: A Modern Approach

Fourth Edition



Chapter 3

Solving Problems By Searching

Pearson

Outline

- Problem-solving agents
- Example Problems
- Problem formulation
- Search Algorithms
- Uninformed Search Strategies
- Informed (Heuristic) Search Strategies
- Heuristic Functions

Introduction

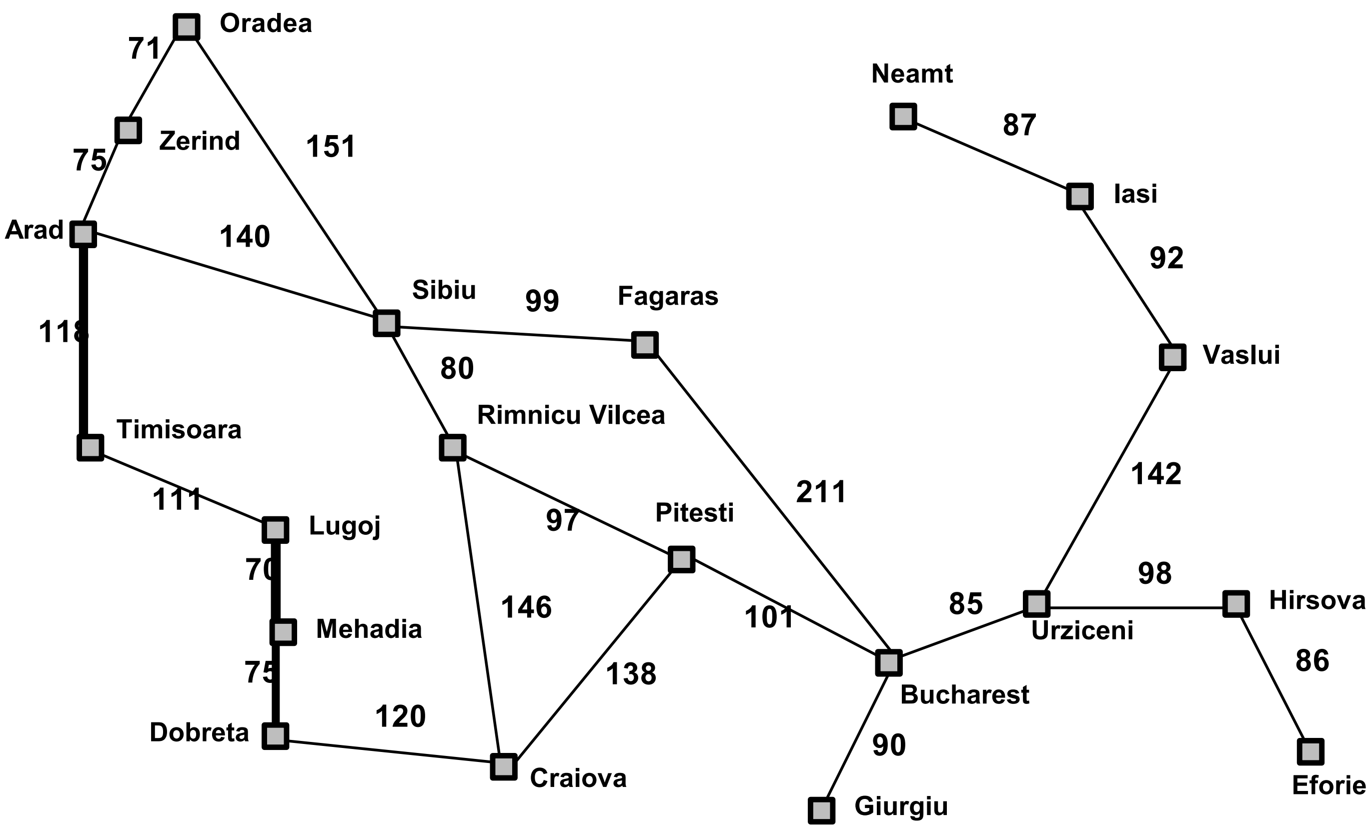
- This section shows how an informed search strategy—one that uses domain-specific hints about the location of goals—can find solutions more efficiently than an uninformed strategy.

- The hints come in the form of a heuristic function, denoted h(n)

h(n) = estimated cost of the cheapest path from the state at node n to the goal

- For example, in route-finding problems, we can estimate the distance from the current state to a goal by computing the straight-line distance on the map between the two points.

Romania with step costs in km



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy search

- **Greedy best-first search** is a form of best-first search that expands first the node with the lowest value—the node that appears to be closest to the goal—on the grounds that this is likely to lead to a solution quickly.

- So, the evaluation function f(n) = h(n)

- Notice that the values of h_{sld} cannot be computed from the problem description itself (that is, the ACTIONS and RESULT functions).

- It takes a certain amount of **world knowledge** to know that h_{sld} is **correlated** with actual road distances and is, therefore, a useful **heuristic**.

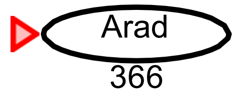
Greedy search

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

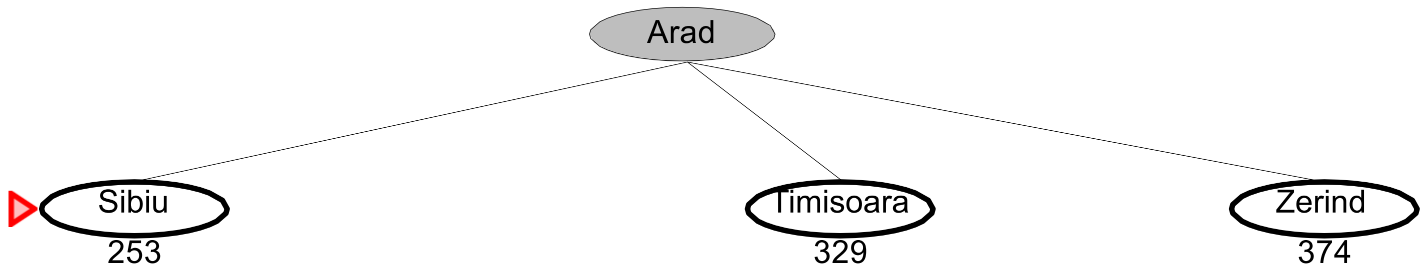
Greedy search expands the node that appears to be closest to goal

Greedy search example



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

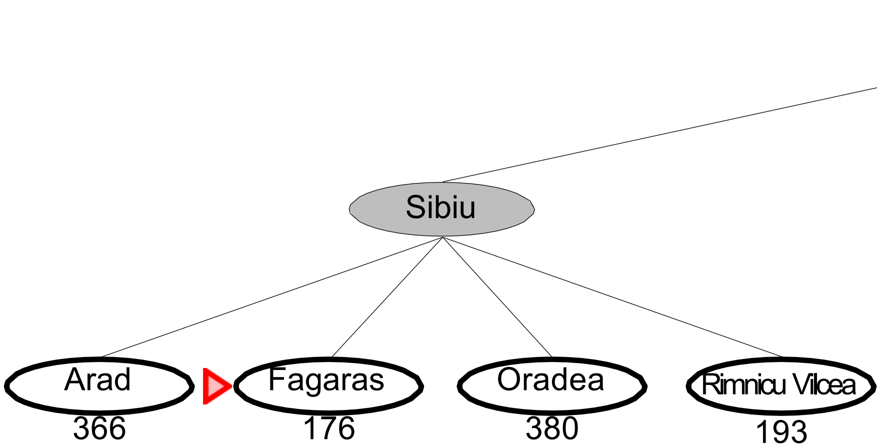
Greedy search example



Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

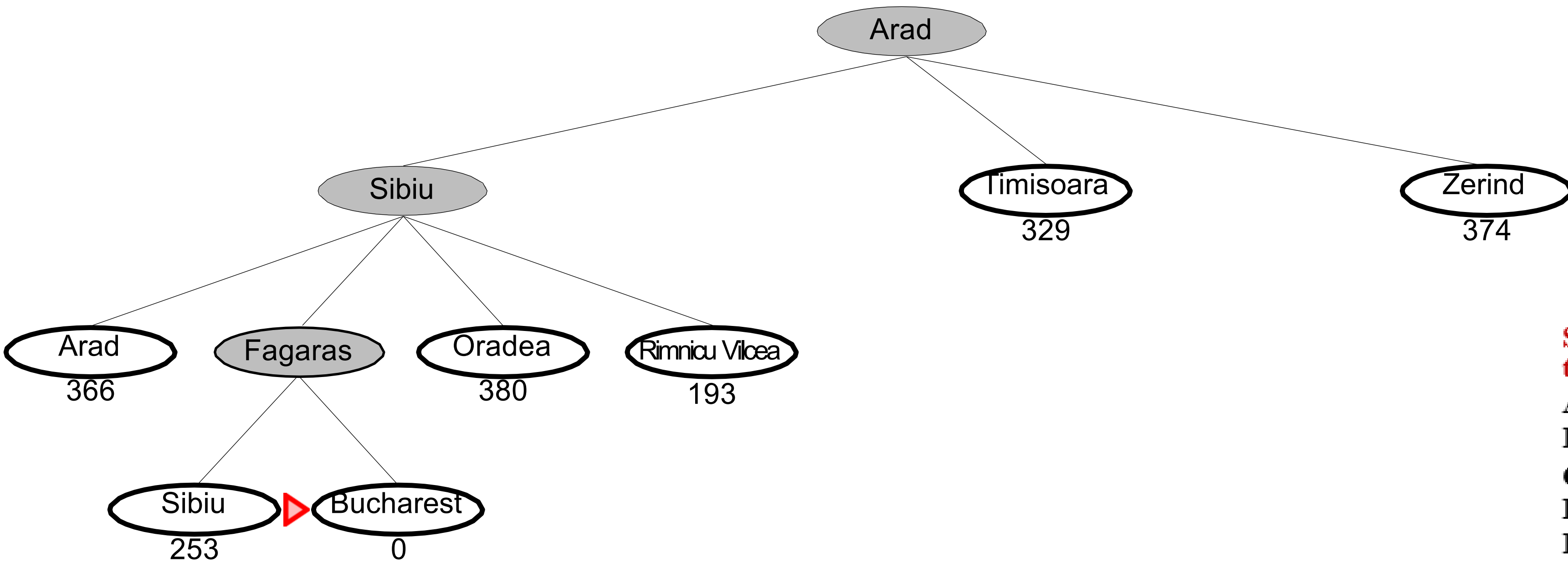
Greedy search example



Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy search example



Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy search

- For this particular problem, greedy best-first search using h_{sld} finds a solution without ever expanding a node that is not on the solution path.

- The solution it found does not have optimal cost, however ☹

- The path via Sibiu and Fagaras to Bucharest is 32 miles longer than the path through Rimnicu Vilcea and Pitesti.

- This is why the algorithm is called **"greedy"**—on each iteration it tries to get as close to a goal as it can, but greediness can lead to worse results than being careful.

Greedy search

- Greedy best-first graph search is complete in finite state spaces, but not in infinite ones.

- With a good heuristic function, the complexity can be reduced substantially, on certain problems reaching **O(bm)**

Properties of greedy search

Complete??

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

- The most common informed search algorithm is A* search (pronounced “A-star search”),

- It is a **best-first search** that uses the evaluation function

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

  where:
  $g(n)$ is the path cost from the initial state to node n and
  $h(n)$ is the estimated cost of the shortest path from n to a goal state,
  so we have

  f(n) = estimated cost of the best path that <u>continues</u> from n to goal

A* search

Idea: avoid expanding paths that are already expensive

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

$g(n)$ = real/actual 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

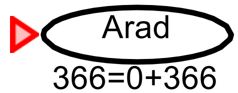
A* search uses an **admissible** heuristic

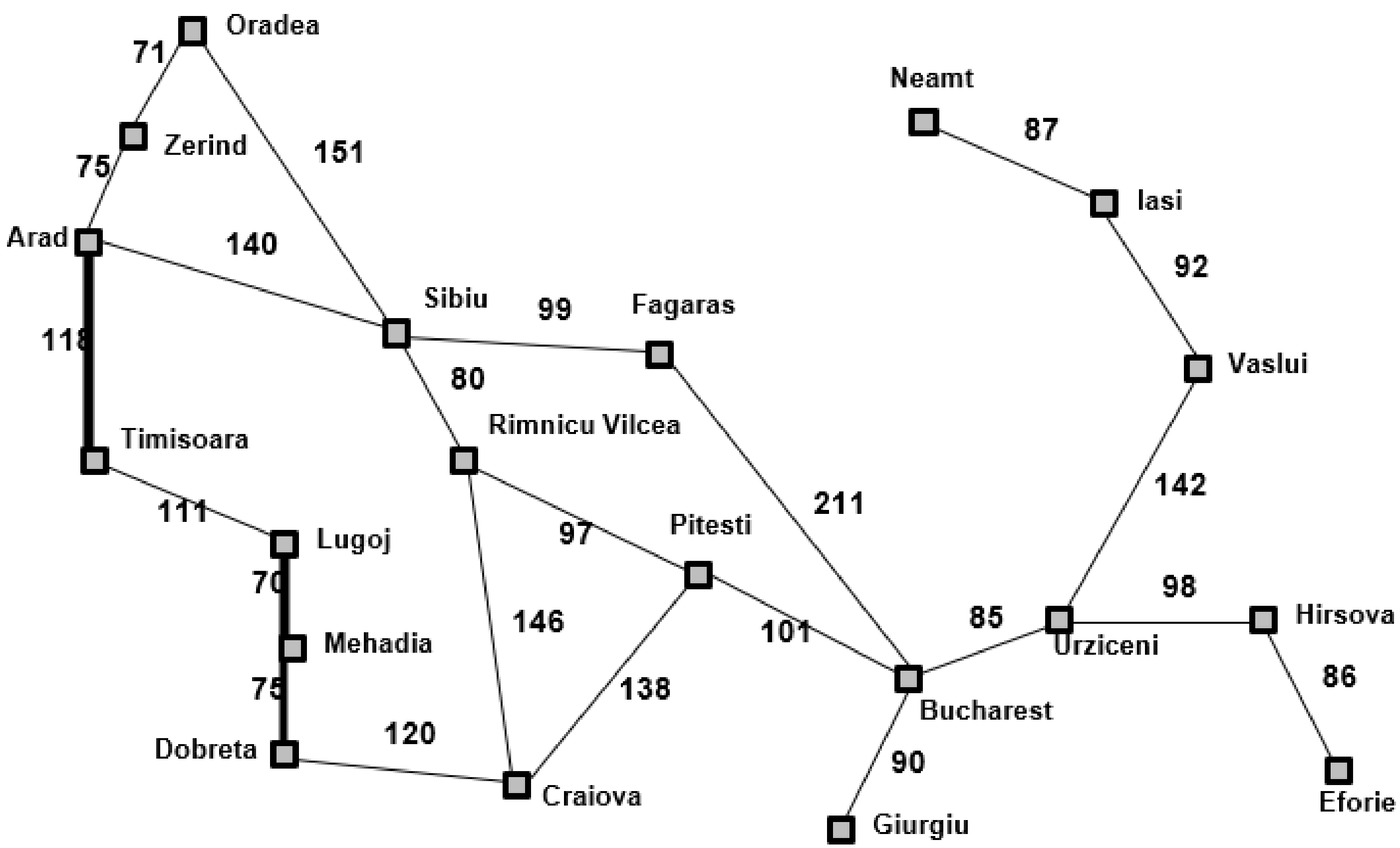
i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost from n.
(Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G.)

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

Theorem: A* search is optimal

A^* search example

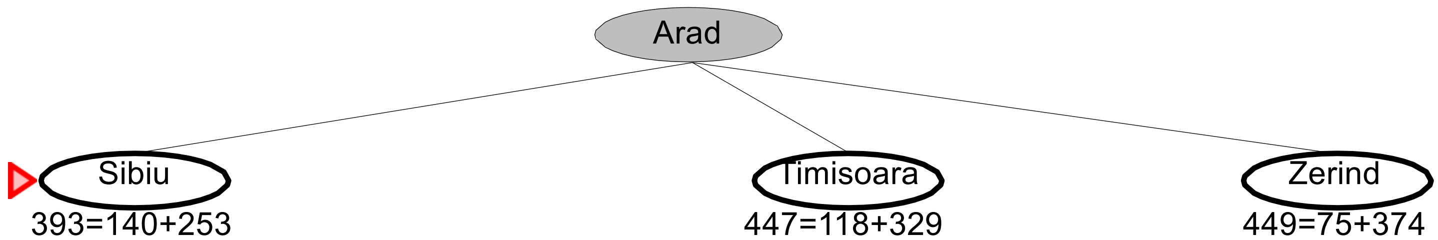


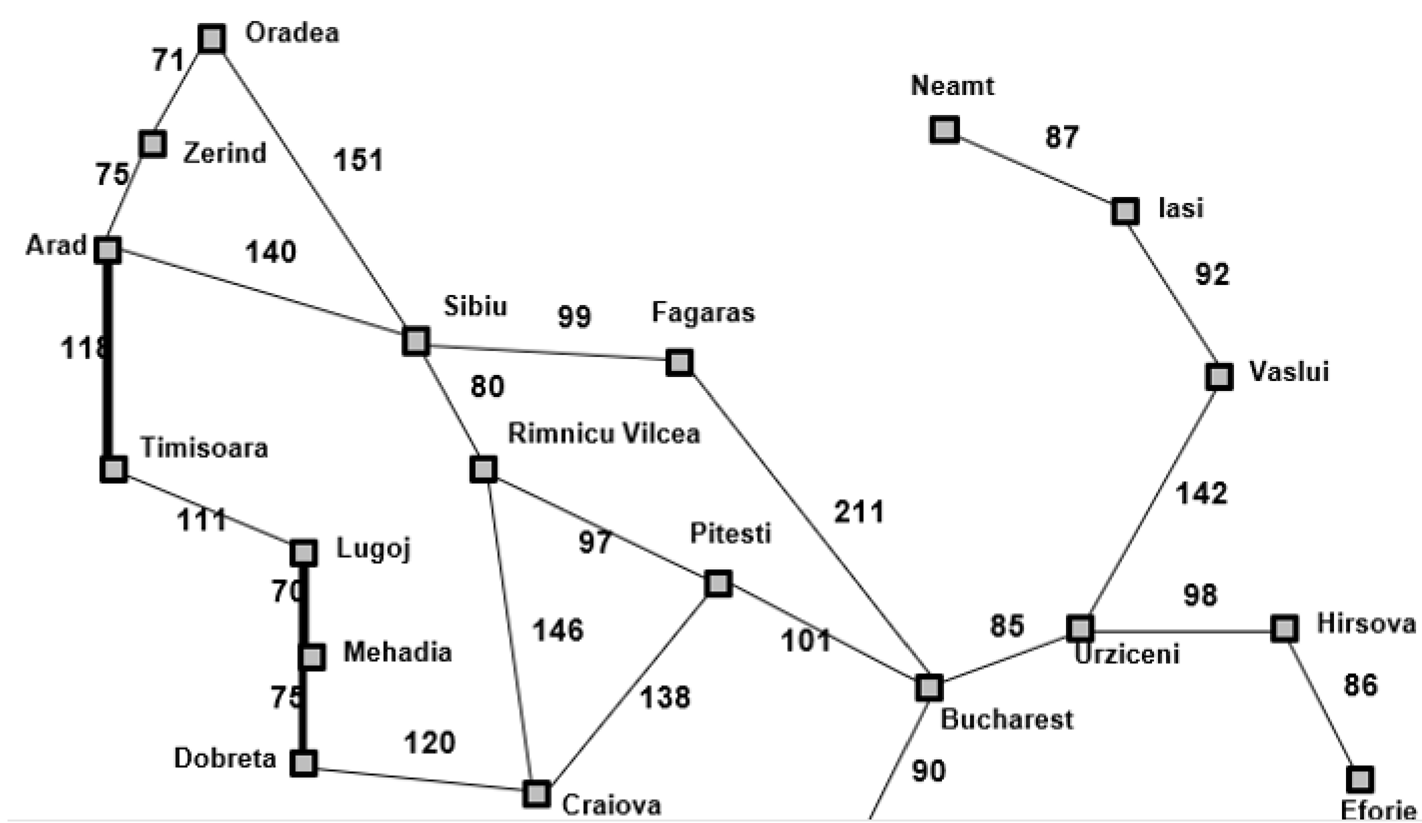


Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

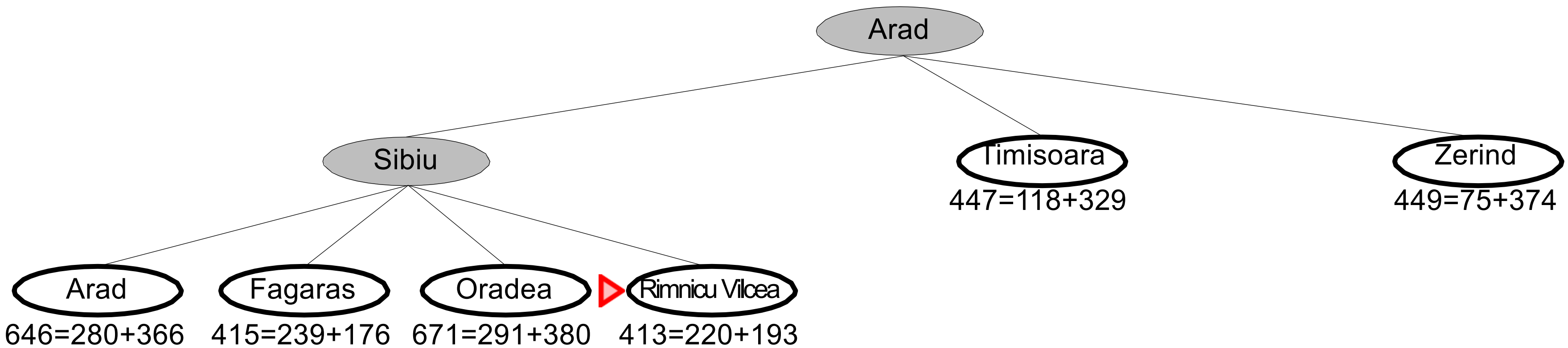


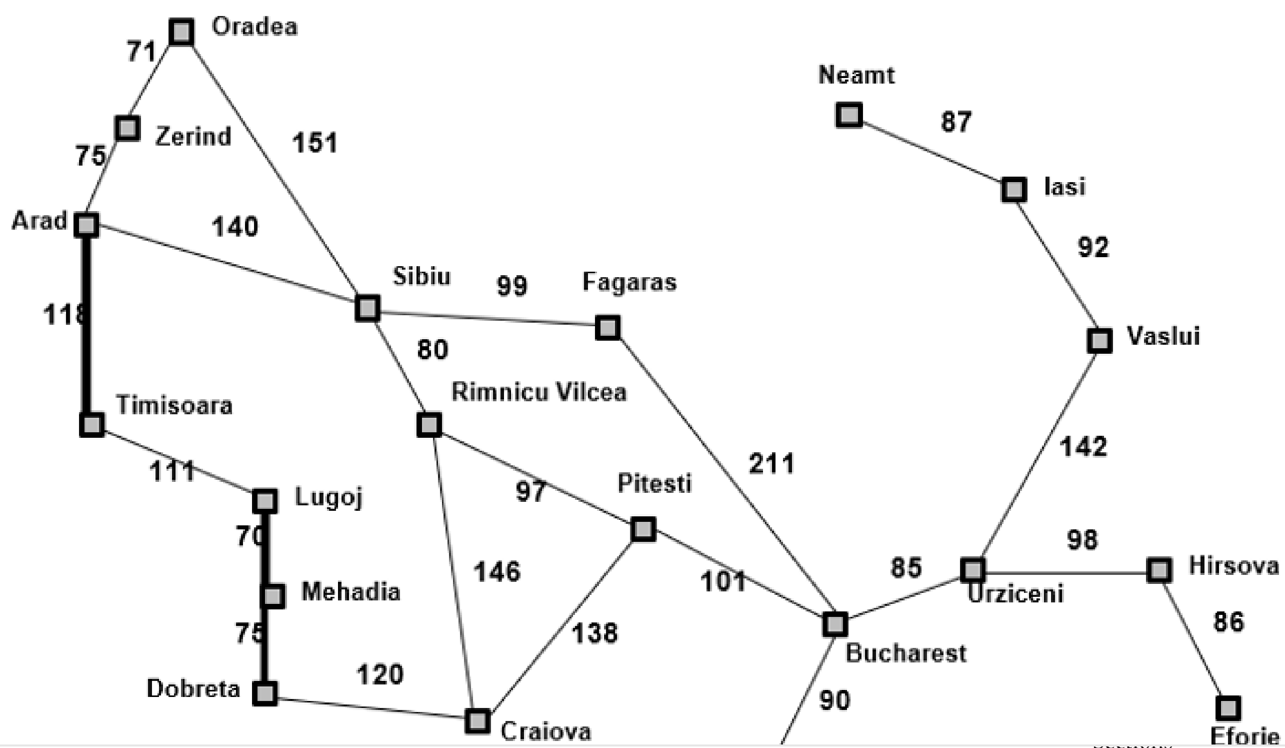


Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

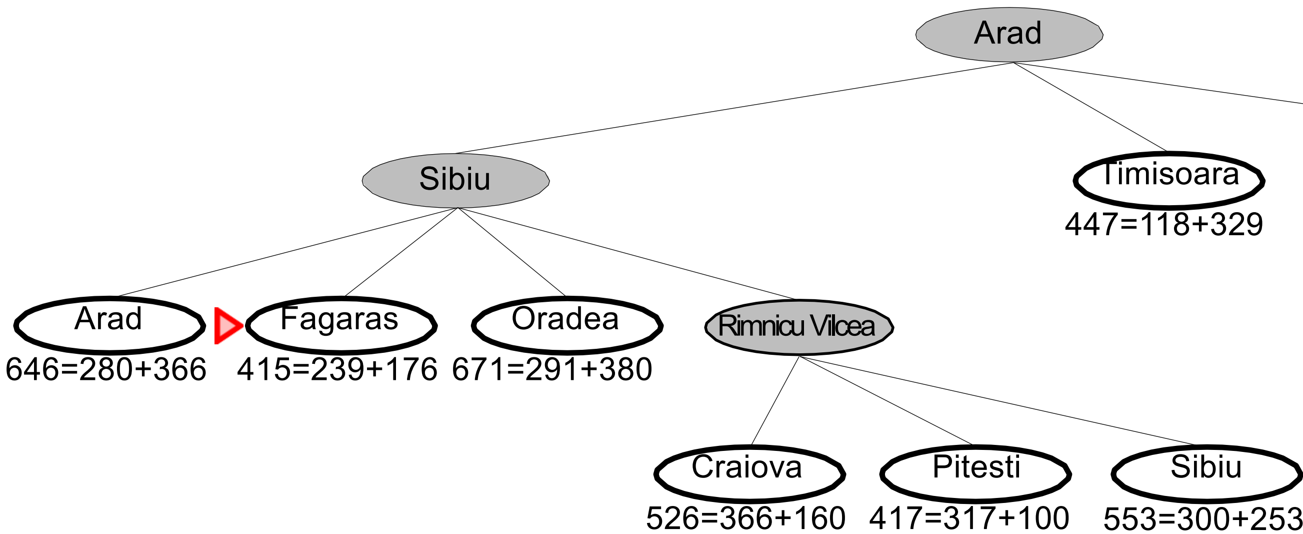


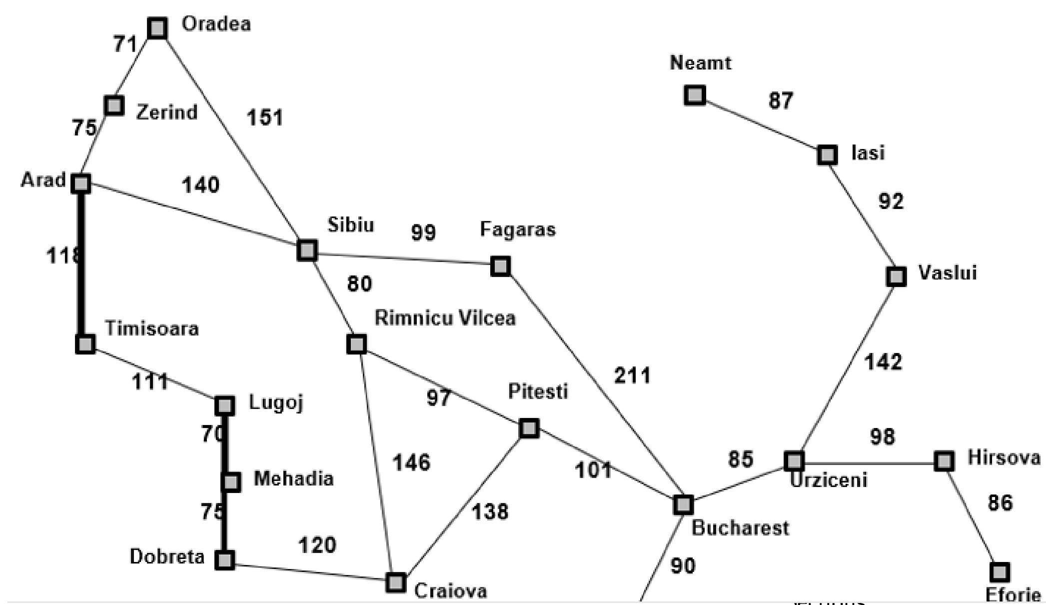


Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

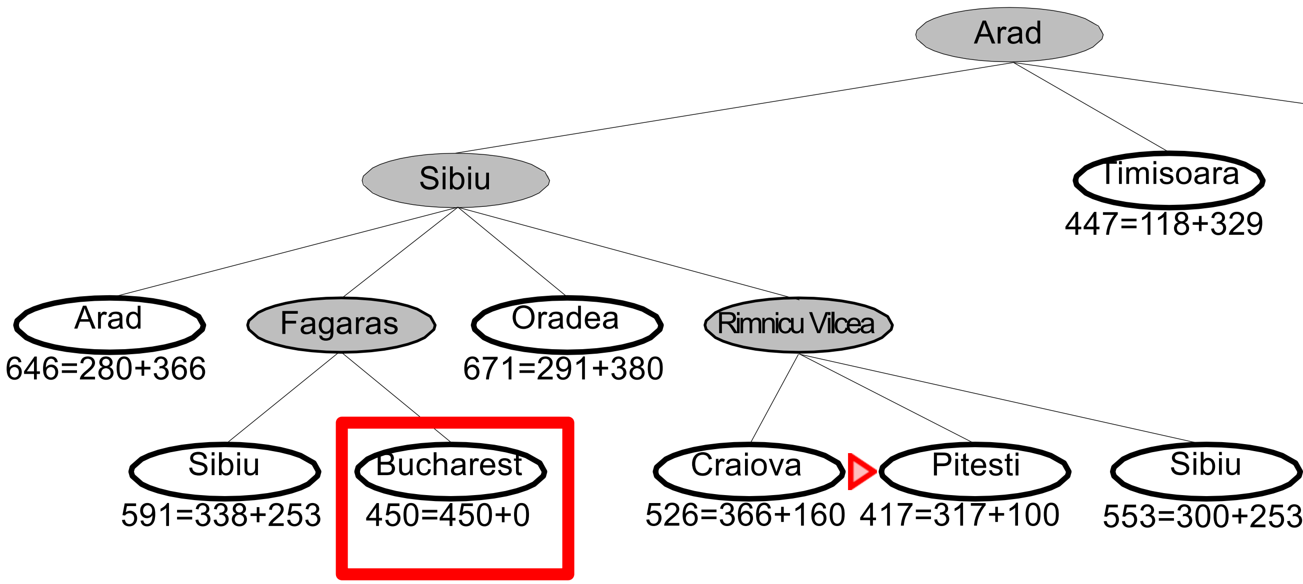


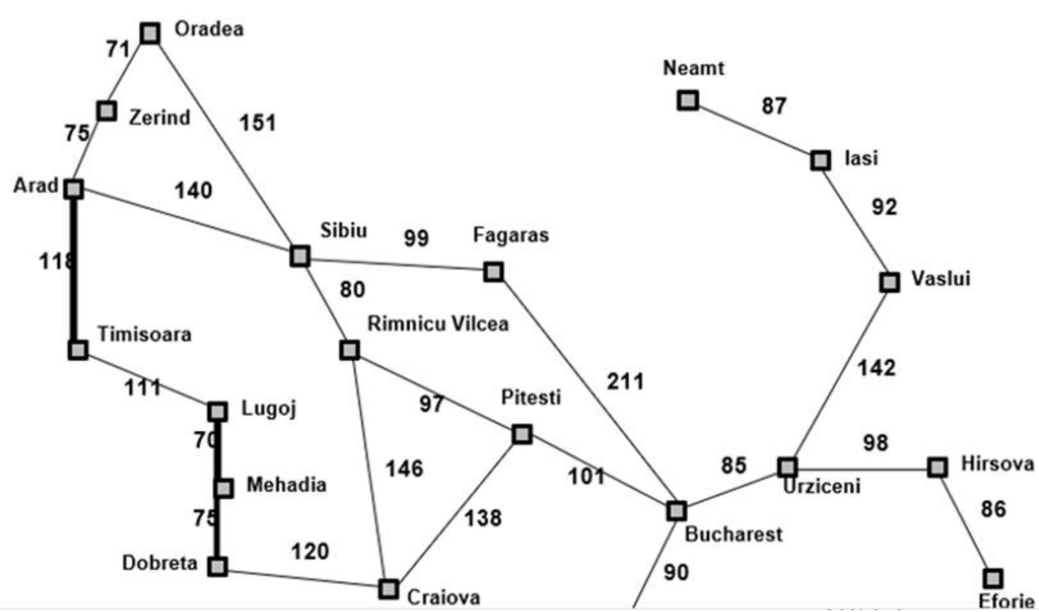


Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

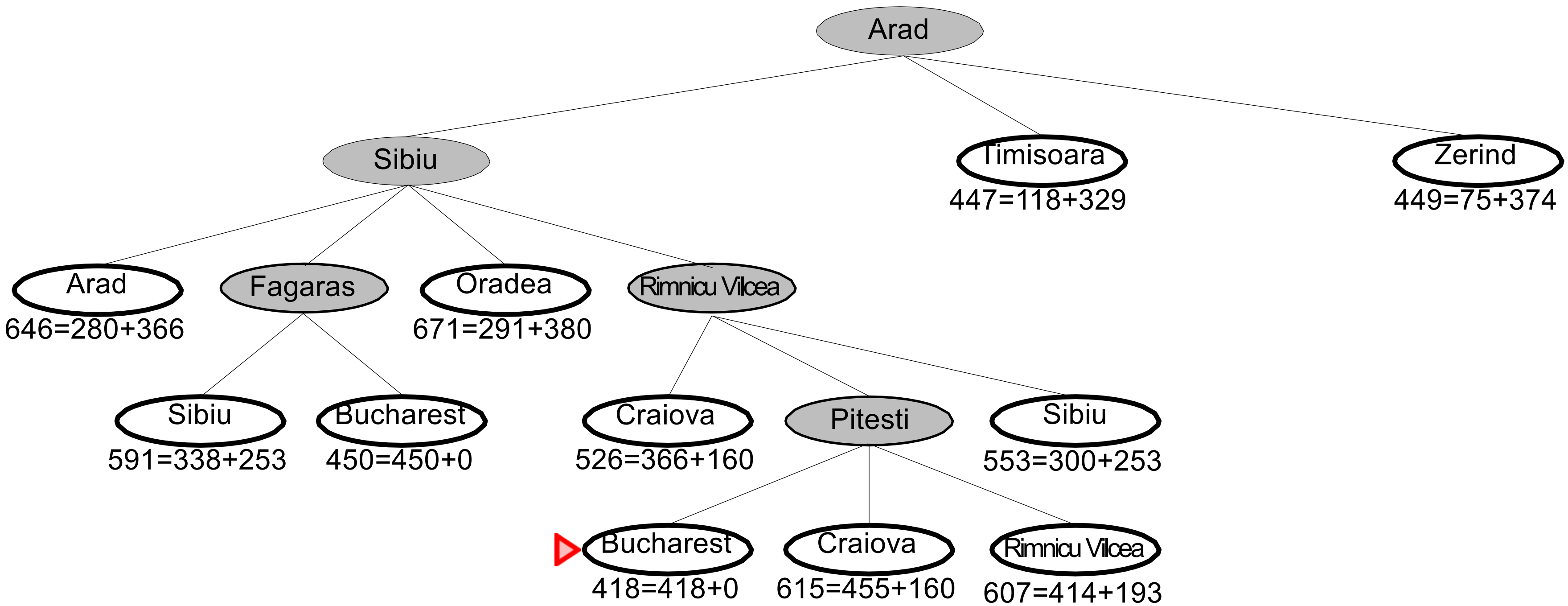




Straight−line distance to Bucharest

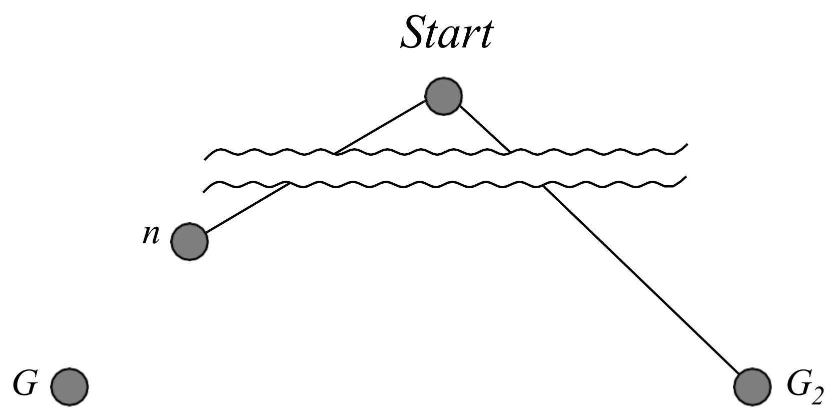
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

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.



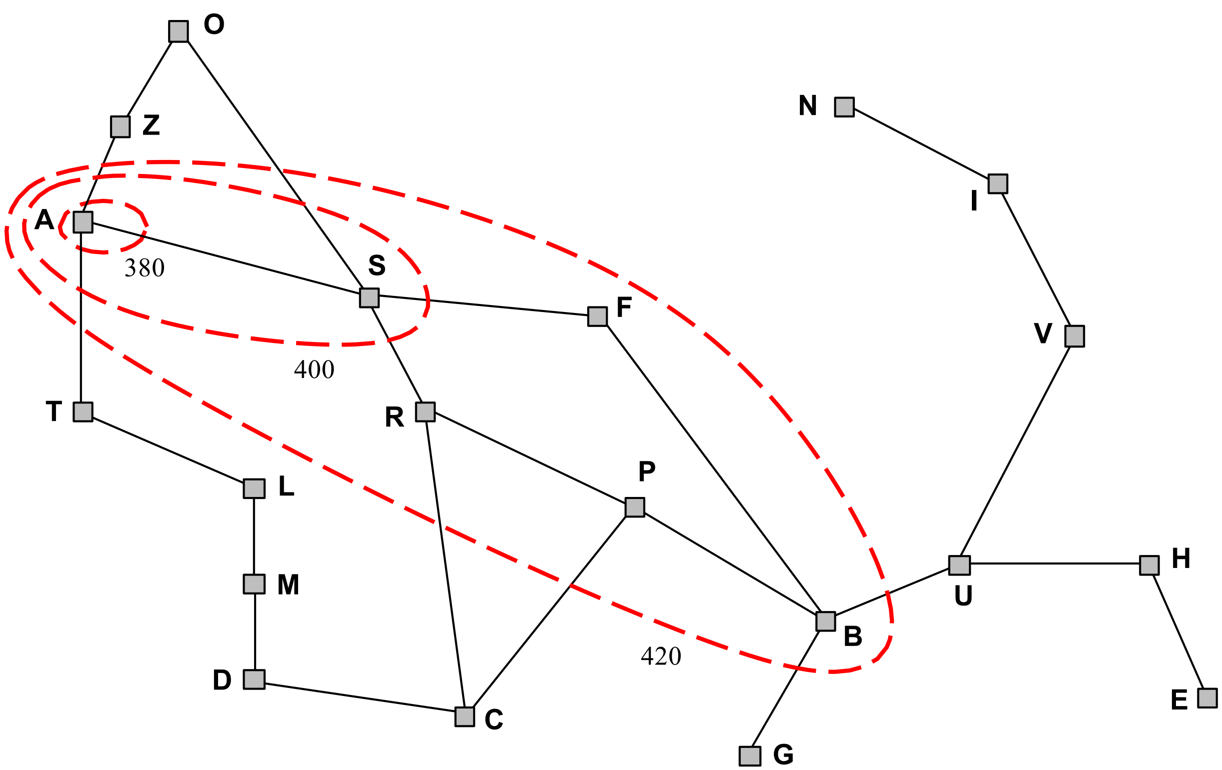
$$
\begin{aligned}
f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\
&> g(G) && \text{since } G_2 \text{ is suboptimal} \\
&\geq f(n) && \text{since } h \text{ is admissible}
\end{aligned}
$$

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??

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??

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 $f(n) = C*$

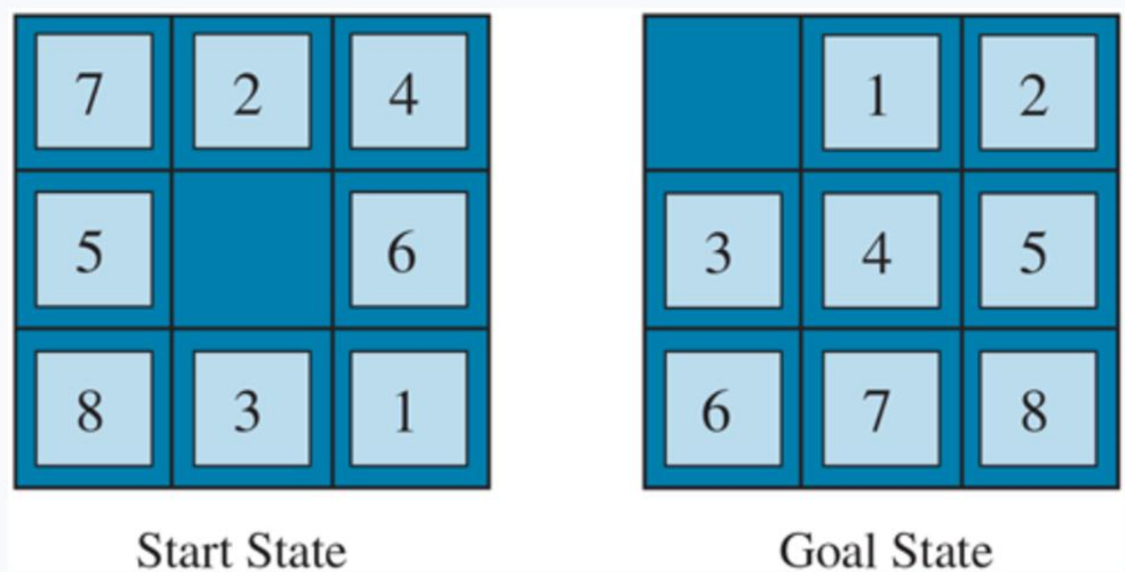
A$*$expands no nodes with $f(n) > C*$

Admissible heuristics

- There are 9!/2 reachable states in an 8-puzzle, so a search could easily keep them all in memory.

- But for the 15-puzzle, there are 16!/2 states—over 10 trillion—so to search that space we will need the help of a good **admissible heuristic function**.

- There is a long history of such heuristics for the 15-puzzle; here are two commonly used candidates:



A typical instance of the 8-puzzle. The shortest solution is 26 actions long.

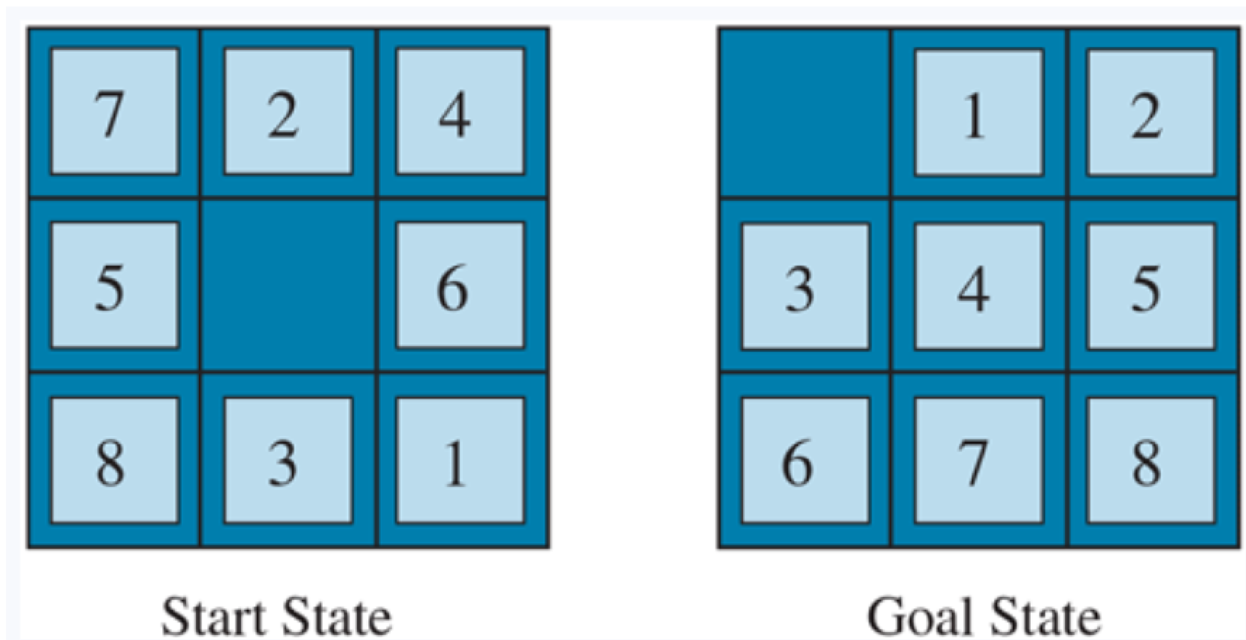
Pearson

Admissible heuristics

E.g., for the 8-puzzle:

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

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



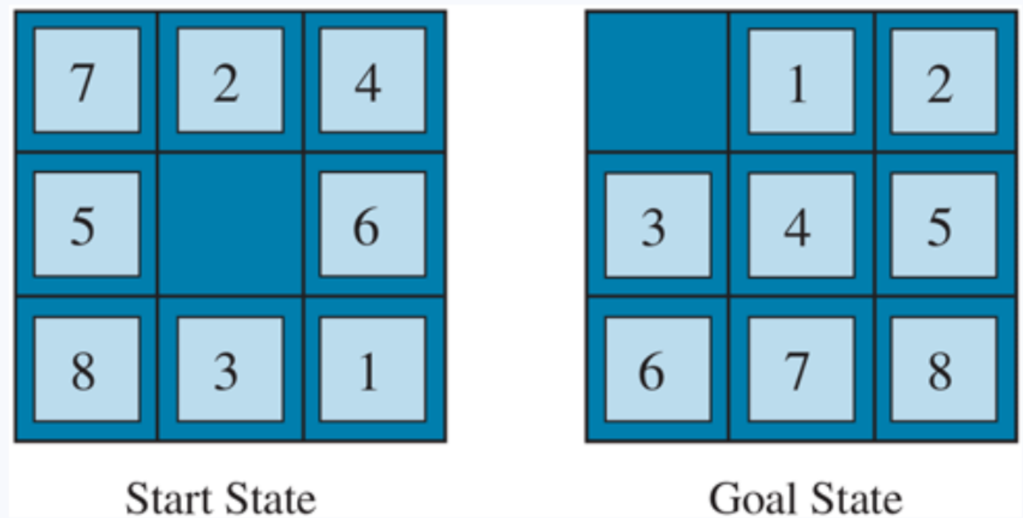
A typical instance of the 8-puzzle. The shortest solution is 26 actions long.

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
$h_2(S)$ =?? 4+0+3+4+1+0+2+1= 15



A typical instance of the 8-puzzle. The shortest solution is 26 actions long.

Admissible heuristics

- H_1 is an admissible heuristic because any tile that is out of place will require at least one move to get it to the right place.

- H_2 is also admissible because all any move can do is move one tile one step closer to the goal.

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 $\approx$ 54,000,000,000 nodes
A$*(h_1)$ = 39,135 nodes
A$*(h_2)$ = 1,641 nodes

Given any admissible heuristics h_a, h_b,

$$h(n) = \max(h_a(n), h_b(n))$$

is also admissible and dominates h_a, h_b

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

A problem consists of five parts: the **initial state**, a set of **actions**, a **transition model** describing the results of those actions, a set of **goal states**, and an **action cost function**.

Uninformed search methods have access only to the **problem definition**. Algorithms build a search tree in an attempt to find a solution.

Informed search methods have access to a **heuristic** function h(n) that estimates the cost of a solution from n.