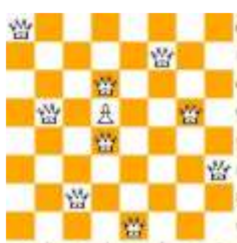
Chapter 3:
Informed Search and Exploration

Dr. Daisy Tang



Informed Search

- Definition:
  - Use problem-specific knowledge beyond the definition of the problem itself
  - Can find solutions more efficiently
- Best-first search
  - Greedy best-first search
  - A*
- Heuristics

Best-First Search

- Idea: use an evaluation function $f(n)$ for each node
  - estimate of "desirability"
  - Expand most desirable unexpanded node
- Implementation: use a data structure that maintains the frontier in a decreasing order of desirability
- Is it really the best?

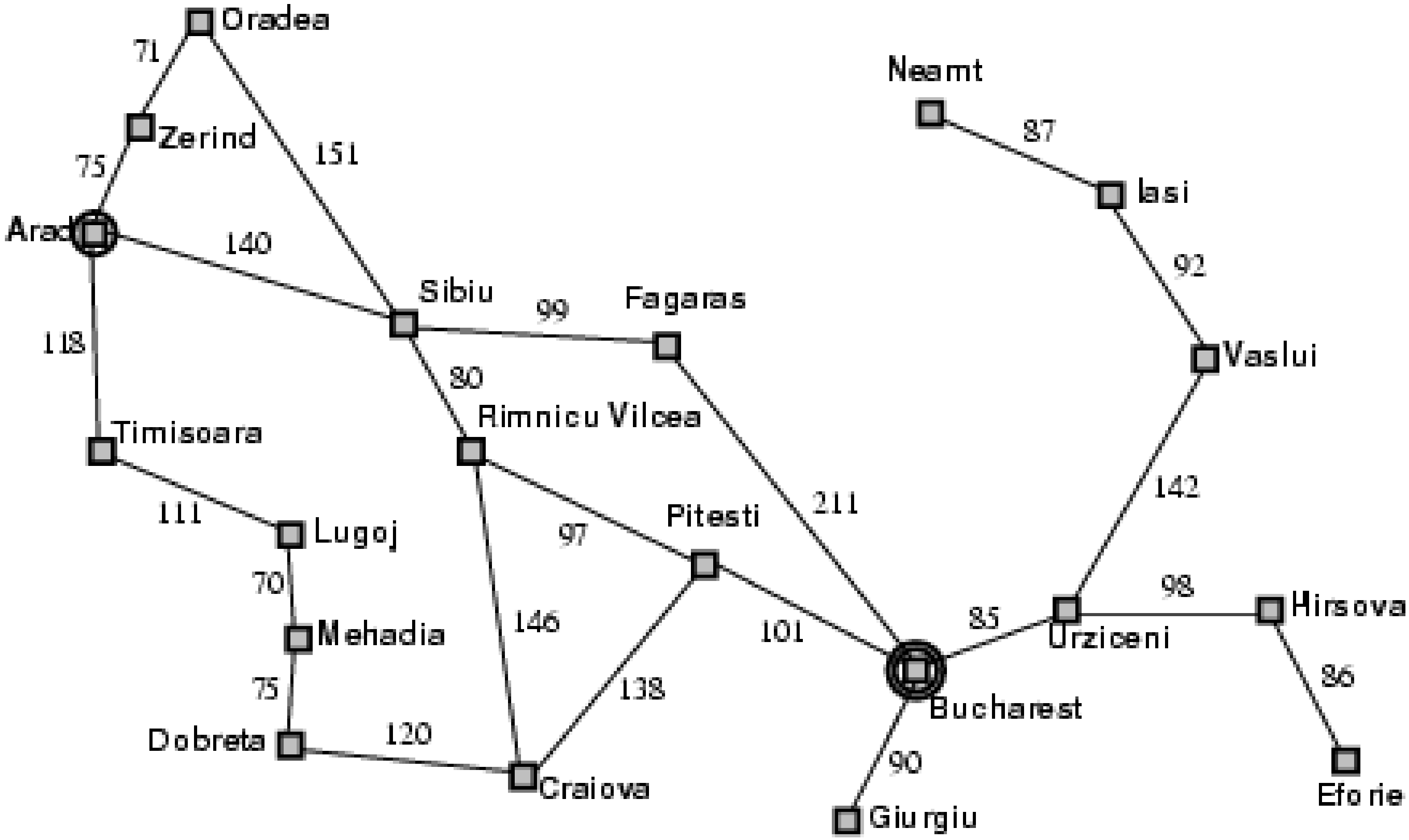
- Special cases: uniform-cost (Dijkstra’s algorithm), greedy search, A* search

- A key component is a heuristic function $h(n)$:
  - $h(n)$ = estimated cost of the **cheapest path** from node n to a goal node
  - $h(n) = 0$ if n is the goal
  - $h(n)$ could be general or problem-specific

Best First Search Algorithm

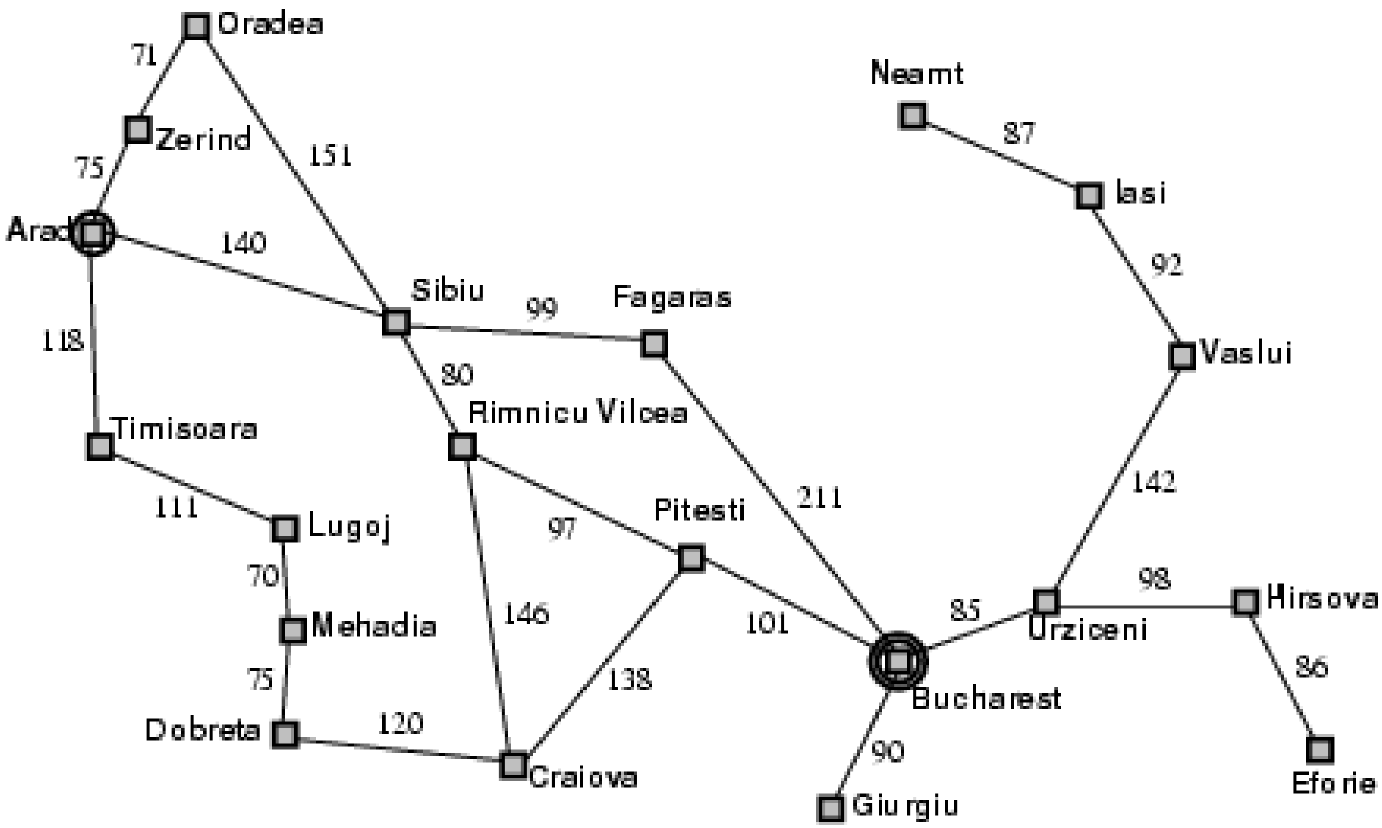
1. initialize the Q with the starting state (node)
2. while Q is not empty, do
    1) assign the first element of Q to N
    2) if N is the goal, return SUCCESS
    3) remove N from Q
    4) add the children of N to Q
    5) sort the entire Q by $f(n)$
3. return FAILURE

Recall Romania Map Example



What's a proper heuristic that measures cheapest path from current node to goal node?

Romania Map with Costs



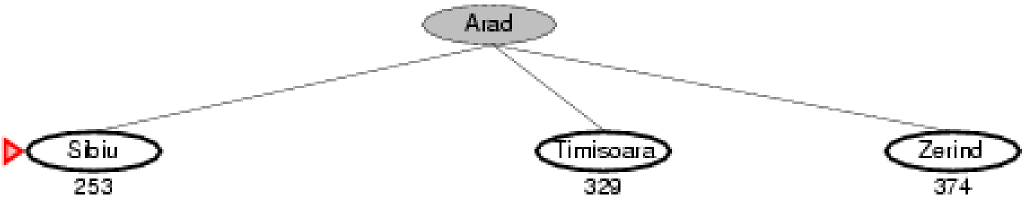
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Best-First Search

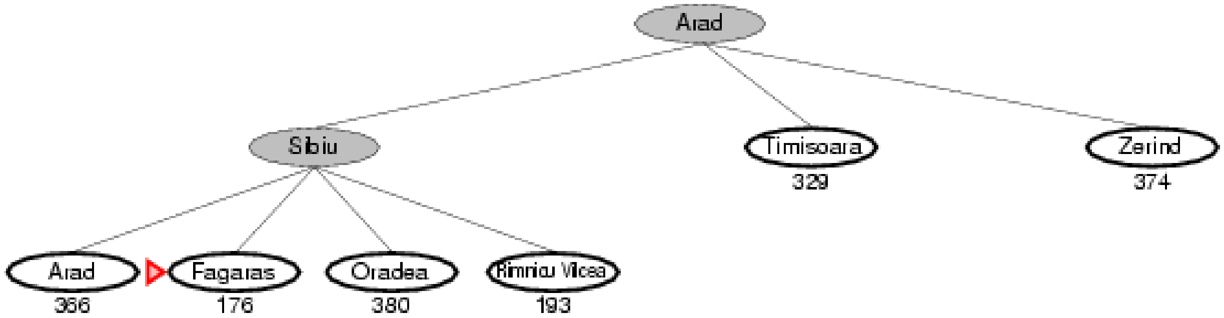
- Evaluation function: $f(n) = h(n)$
  - estimate the cost from n to goal
- h_{SLD} = straight line distance from n to Bucharest

Example: Arad to Bucharest

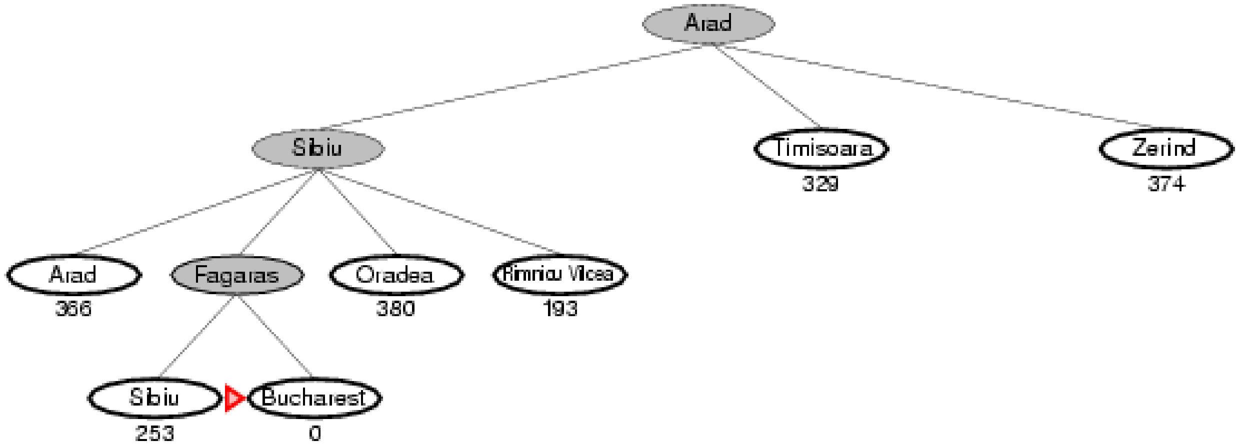
Example: Arad to Bucharest



Example: Arad to Bucharest



Example: Arad to Bucharest



Analysis of Greedy Best-First

- Complete?
  - From Iasi to Fagaras
  - 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

In-Class Exercise #3.8

- Draw the search tree generated by Greedy search implemented with tree-search algorithm to find a path from T (Timisoara) to B (Bucharest)

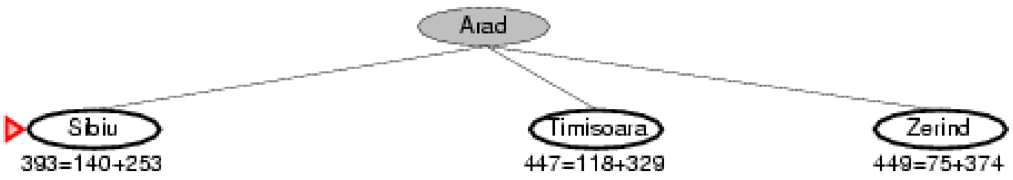
A*: Minimizing Total Est. Cost

- Idea: avoid expanding paths that are already expensive

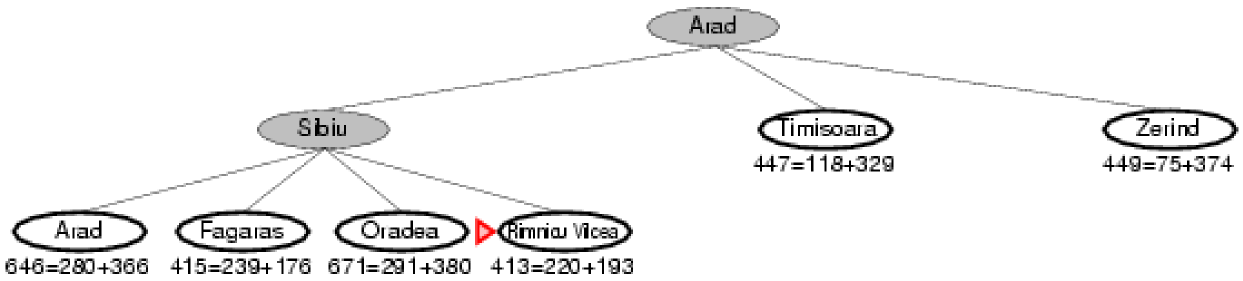
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ = cost so far to reach n
  - $h(n)$ = estimated cost from n to goal
  - $f(n)$ = estimated total cost of path through n to goal

A* Search Example

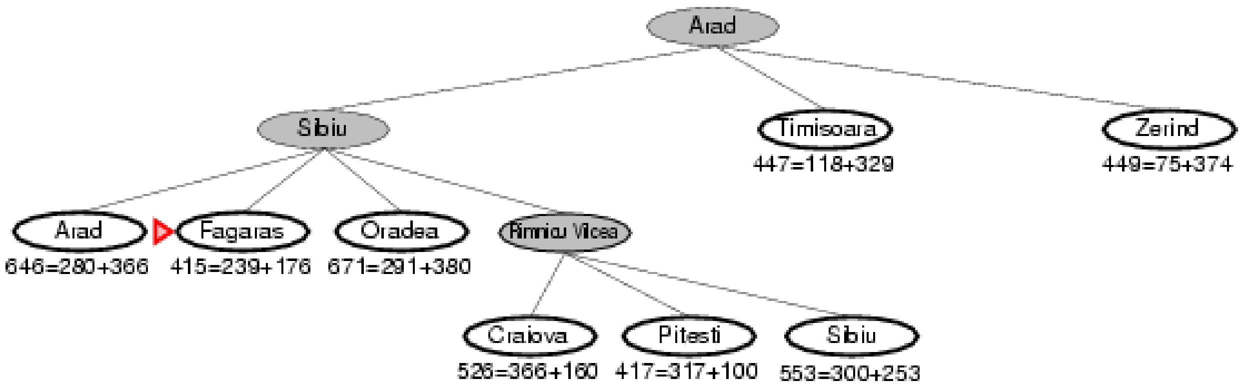
A* Search Example



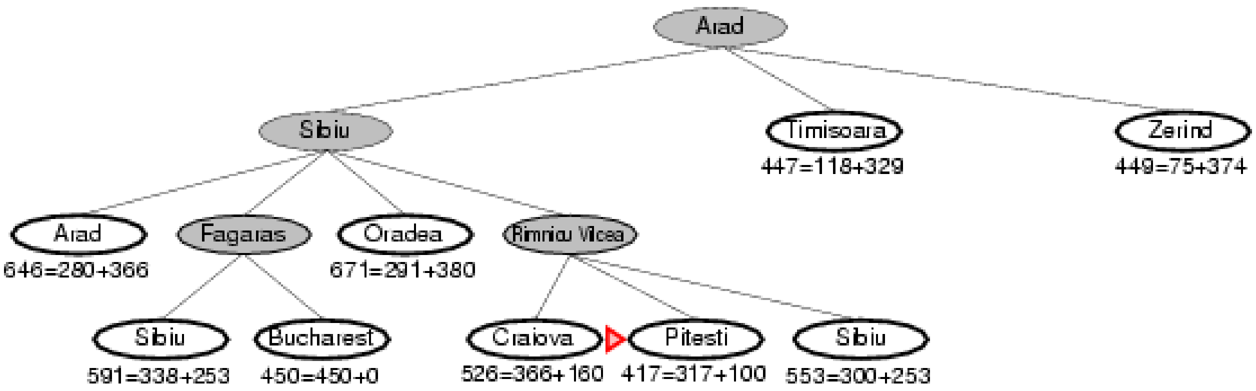
A* Search Example



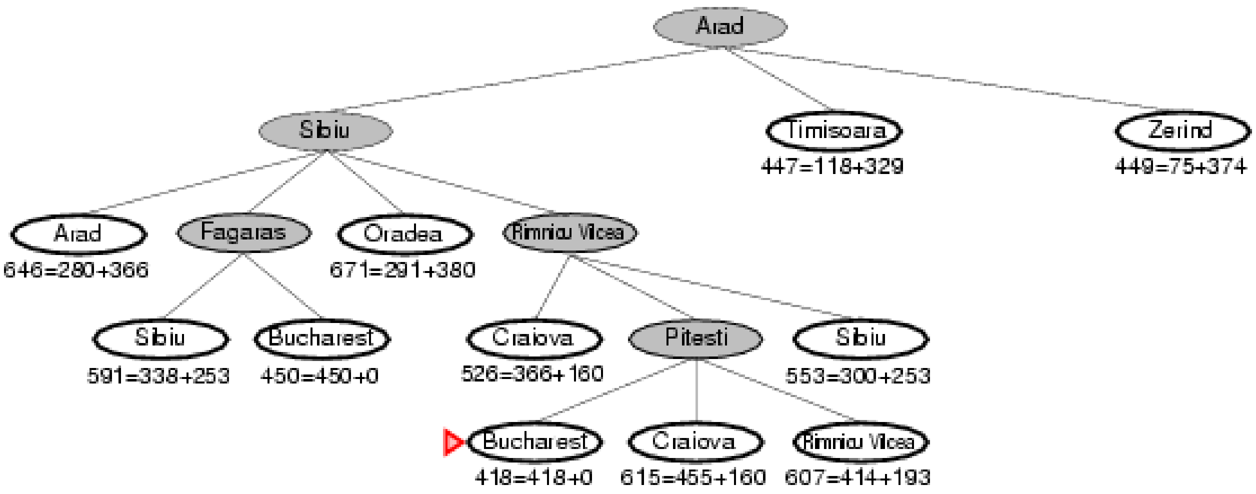
A* Search Example



A* Search Example



A* Search Example



In-Class Exercise #3.9

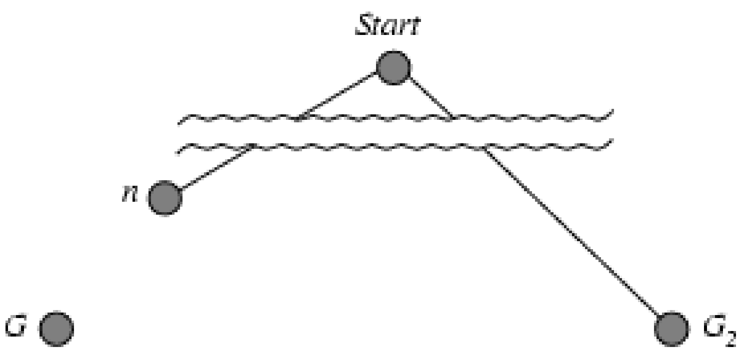
- Draw the search tree generated by applying A* and graph-search to find a path from Lugoj to Bucharest using the straight-line distance heuristic. Show the f-cost f(n) for each node.

Admissible Heuristic

- A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$, 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

Optimality of A* -- Proof

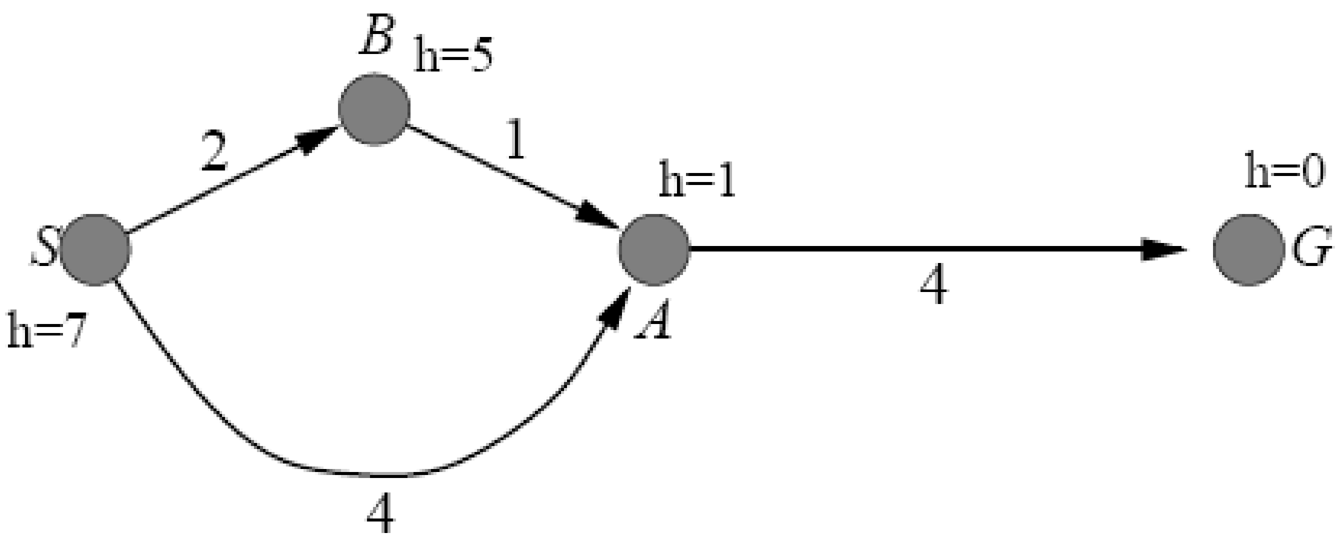
- Suppose some suboptimal goal G_2 has been generated and is in the frontier. Let *n* be an unexpanded node in the frontier such that *n* is on a shortest path to an optimal goal *G*.



- Assume the optimal cost is C*
- $f(G_2) = g(G_2) + h(G_2) = g(G_2) > C^*$
- $f(n) = g(n) + h(n) <= C^*$
- from the above, we have
  - $f(n) <= C^* < f(G_2)$
- thus G_2 will not be expanded

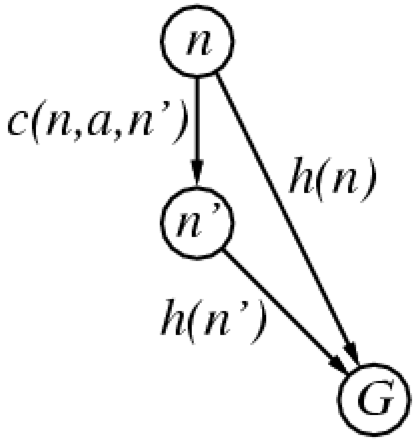
Case-Study

- A* using Graph-Search returns a suboptimal solution with an h(n) function that is admissible.



Consistency Heuristics

- A heuristic is consistent if for every node n, every successor n' of n generated by any action a, $h(n) \le c(n,a,n') + h(n')$
- Triangle inequality



- Every consistent heuristic is also admissible

Proof by induction on the number k of nodes on the shortest path to any goal from n.

K = 1, let n' be the goal node; then $h(n) \le c(n, a, n')$

Assume n' is on the shortest path k steps from the goal and that h(n') is admissible by hypothesis, then:

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

So, h(n) at k+1 steps from the goal is also admissible.

A* With Graph Search

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

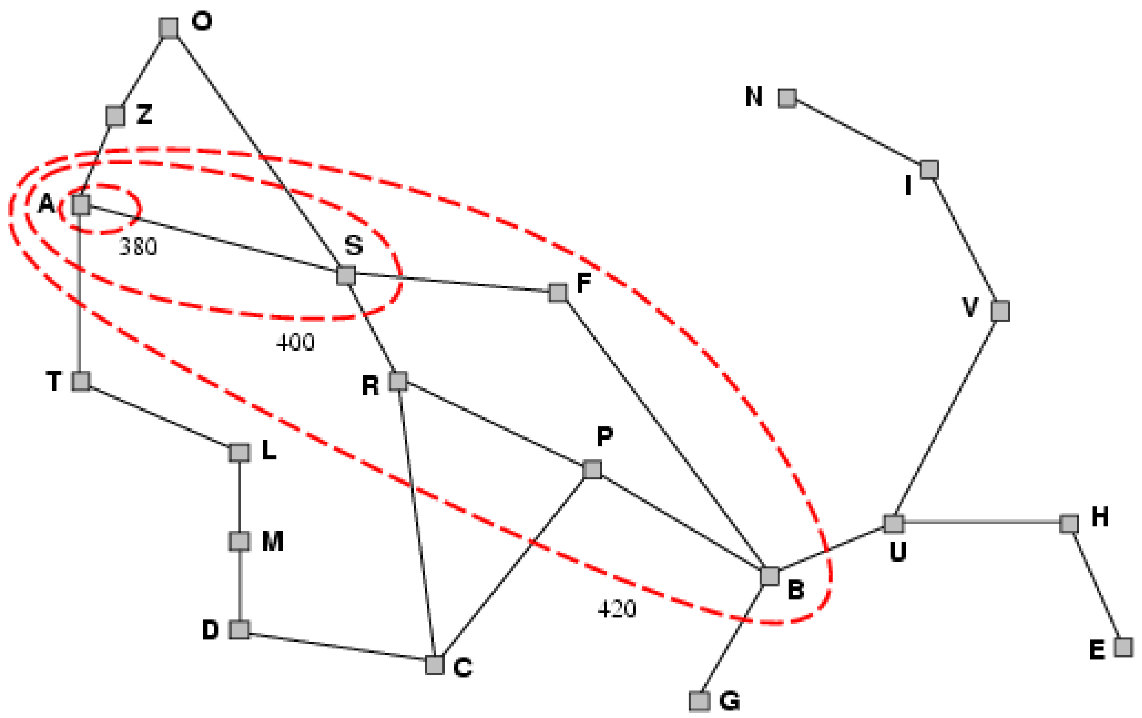
- If h is consistent, and n' is a successor of n, we have

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

- i.e., $f(n)$ is non-decreasing along any path
- Whenever A* selects a node for expansion, the optimal path to that node has been found

Optimality of A*

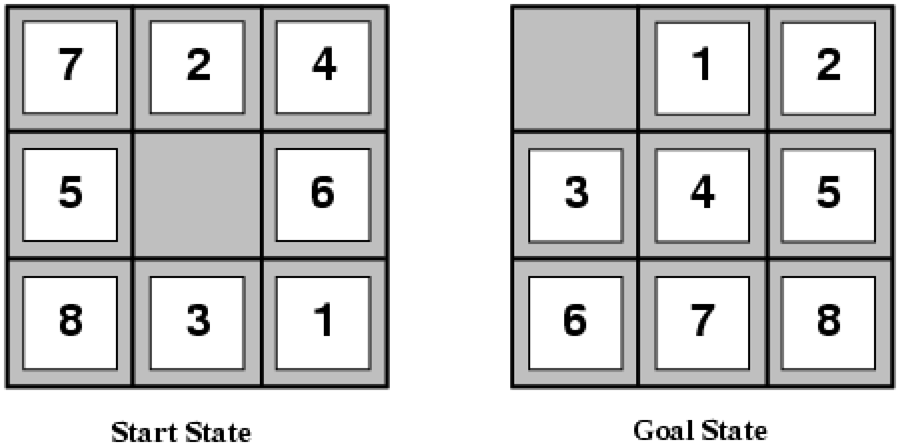
- A* expands nodes in order of increasing *f* value
- Assume C* is the optimal cost
  - A* expands all nodes with f(n) < C*
  - A* might then expand some of the nodes right on the "goal contour" (f(n) = C*) before selecting a goal node
  - Uniform-cost search (h(n) = 0) → bands more circular



Analysis of A*

- Important idea:
  - appropriate $h(n)$ function
  - A* is optimal efficient (no other optimal alg. is guaranteed to expand fewer nodes than A*)
  - pruning while still guaranteeing optimality
- Complete?
  - Yes (unless there are infinitely many nodes with $f \leq f(G)$)
- Optimal?
  - Yes, with finite b and positive path cost
- However, A* is not the answer for all problems
- Time?
  - Exponential in the length of the solution
- Space?
  - Keeps all nodes in memory
  - A* usually runs out of space long before it runs out of time

Heuristic Functions



- Two commonly used functions:
  - $h_1(n)$ = number of misplaced tiles
  - *e.g.*, $h_1(n) = 8$ in the above example
  - $h_2(n)$ = sum of the distances of the tiles from their goal positions, called Manhattan distance
  - *e.g.*, $h_2(n) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$ in the above example

Quality of Heuristic

- Assume # of nodes generated by A* for a problem is **N** and the solution depth is **d**, then **b*** is the effective branching factor that a uniform tree of depth d would have to have. Thus:
  - $N + 1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$

- b* might vary across problem instances, but is fairly constant for sufficiently hard problems

- A well-designed heuristic would have a value of *b** close to 1

The Comparison

	Search Cost			Effective Branching Factor		
d	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Each data point corresponds to 100 instances of the 8-puzzle problem in which the solution depth varies.

Heuristics Domination

- 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
  - A* using h_2 will never expand more nodes than A* using h_1

Inventing Admissible Heuristic

- 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

Inventing Admissible Heuristic

- If a collection of admissible heuristics h_1 … h_m is available, and none of them dominates any of the others, we can choose
  - $h(n) = \max\{h_1(n), \ldots, h_m(n)\}$

- We can also get from sub-problem of a given problem

In-Class Exercise #3.10

- The heuristic path algorithm is a best-first search in which the objective function is $f(n) = (2-w) \times g(n) + w \times h(n)$. For what values of w is the algorithm guaranteed to be optimal? (You may assume that h is admissible.) What kind of search does this perform when $w = 0$? When $w = 1$? When $w = 2$?

In-Class Exercise #3.11

- Prove each of the following statements:
  - Breadth-first search is a special case of uniform-cost search.
  - Breadth-first search, depth-first search, and uniform-cost search are special cases of best-first search.
  - Uniform-cost search is a special case of A* search.

Discussion

- Project 1

Summary

- Applying heuristics to reduce search costs
  - Best-first search: $h(n)$
  - Greedy best-first search: $h(n)$
  - A*: $f(n) = g(n) + h(n)$