

Informed search algorithms

(Based on slides by Oren Etzioni,
Stuart Russell)

Outline

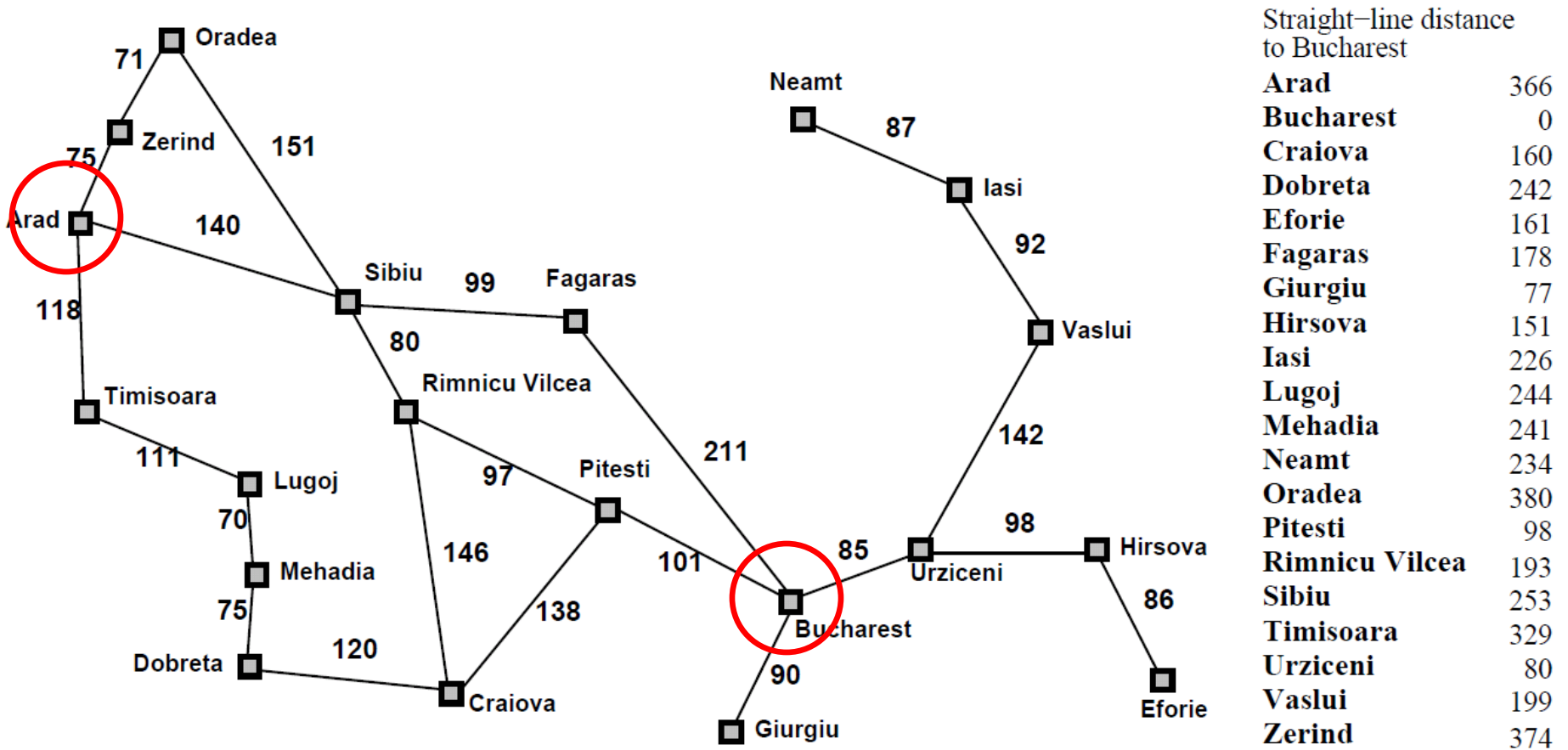
- Greedy best-first search
- A^* search
- Heuristics
- Local search algorithms
- Hill-climbing search
- Simulated annealing search
- Local beam search
- Genetic algorithms

Best-first search

- A search strategy is defined by picking the **order of node expansion**
- Idea: use an **evaluation function** $f(n)$ for each node
 - estimate of "desirability"

→ Expand most desirable unexpanded node
- Implementation:
Order the nodes in fringe in decreasing order of desirability
- Special cases:
 - greedy best-first search
 - A* search

Romania with step costs in km



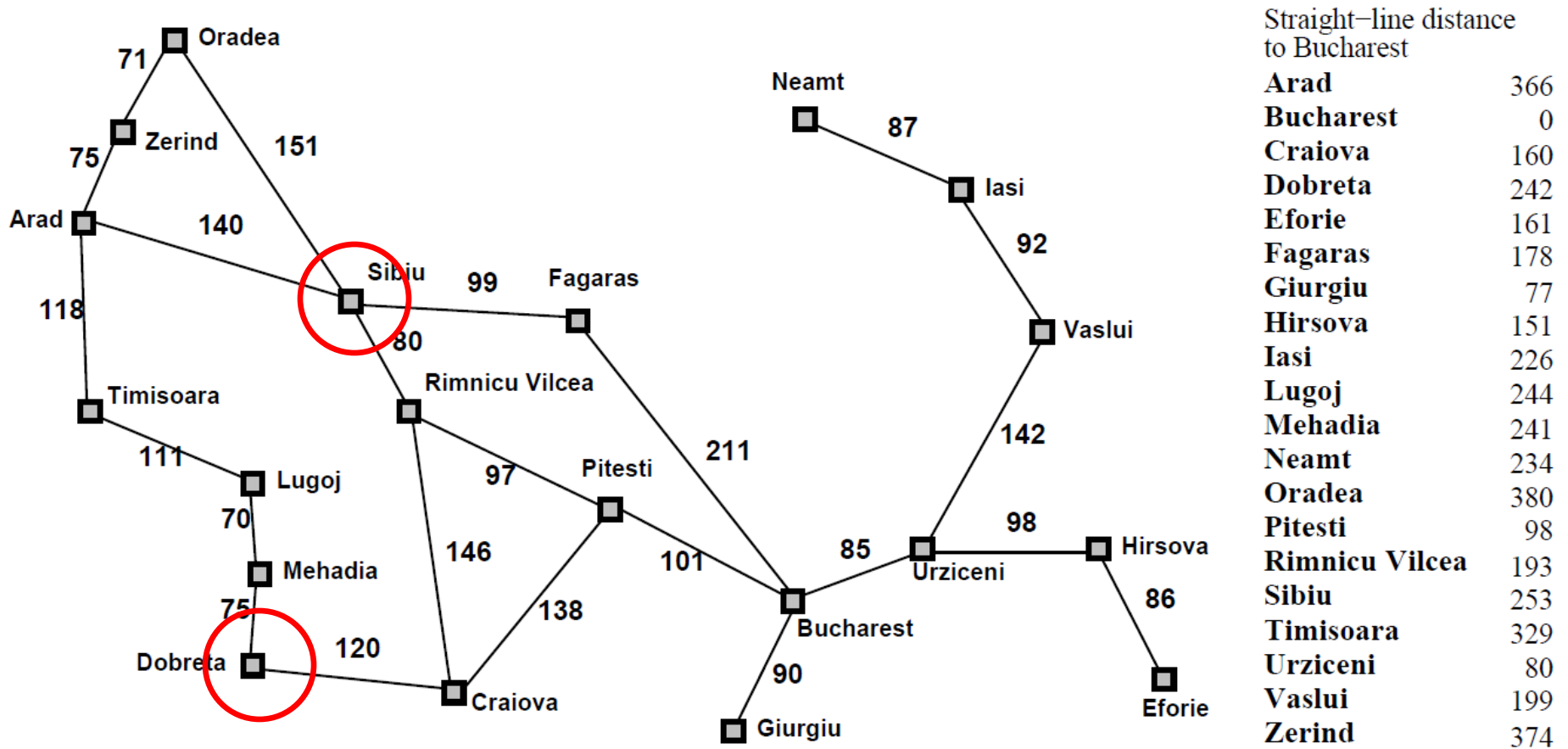
Greedy best-first search

- Evaluation function $f(n) = h(n)$ (**h**euristic)
= 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

Properties of greedy best-first search

- Complete?
- No – can get stuck in loops, e.g., lasi $\rightarrow$ Neamt $\rightarrow$ lasi $\rightarrow$ Neamt $\rightarrow$
- Time?
- $O(b^m)$, but a good heuristic can give dramatic improvement
- Space?
- $O(b^m)$ -- keeps all nodes in memory
- Optimal?
- No

Romania with step costs in km



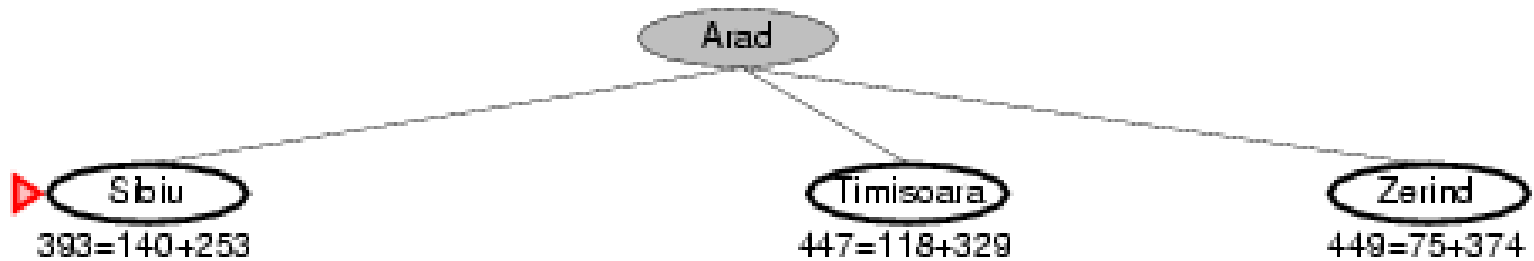
A* search

- 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

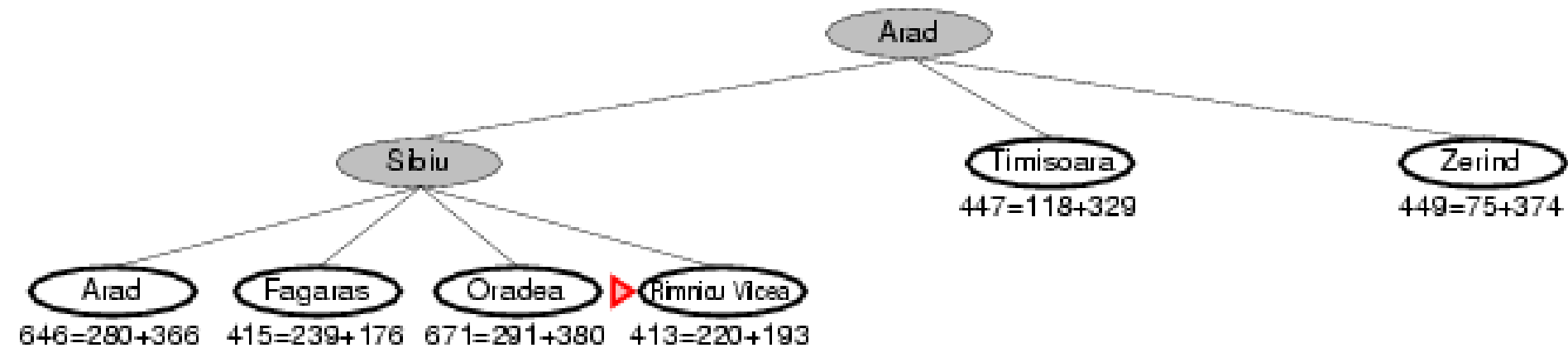
A* search example



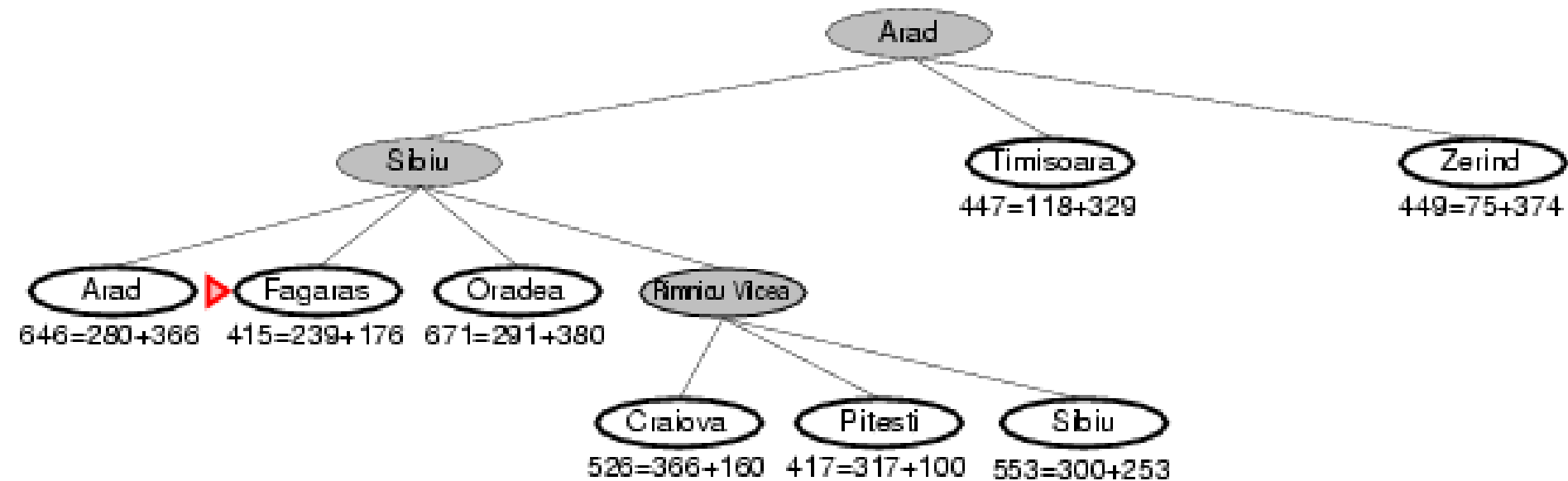
A* search example



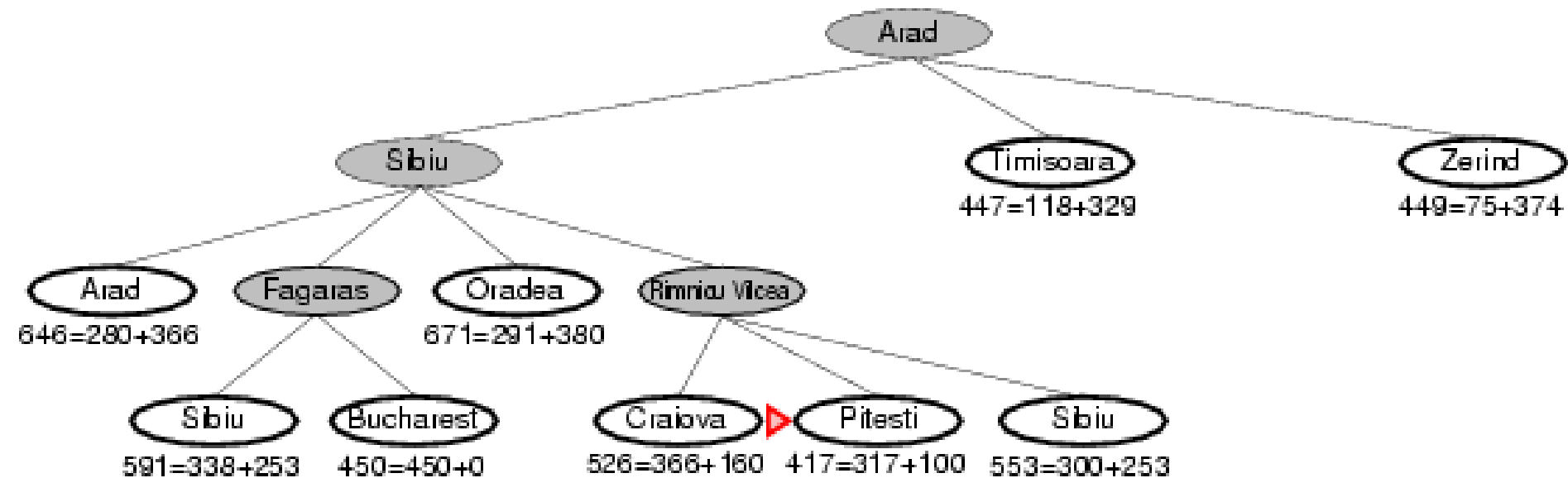
A* search example



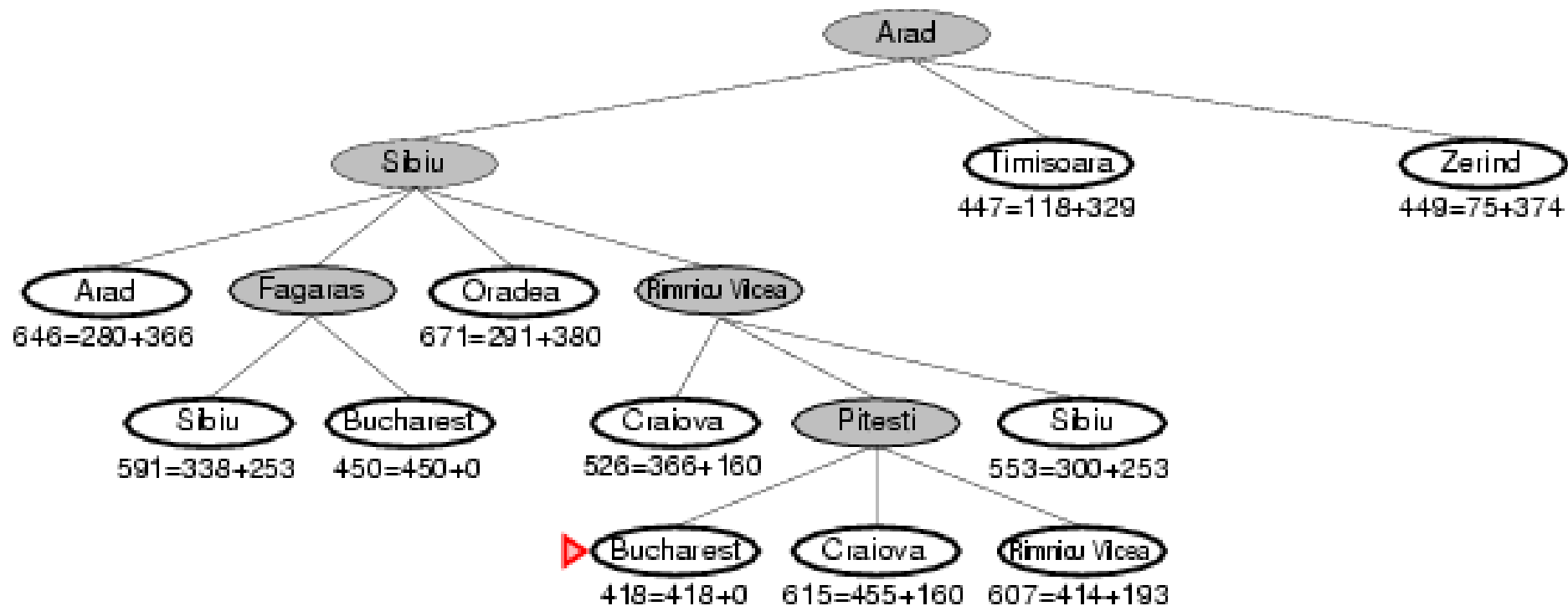
A* search example



A* search example



A* search example



Admissible heuristics

- 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

Properties of A*

- Complete?

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

- Time? Exponential

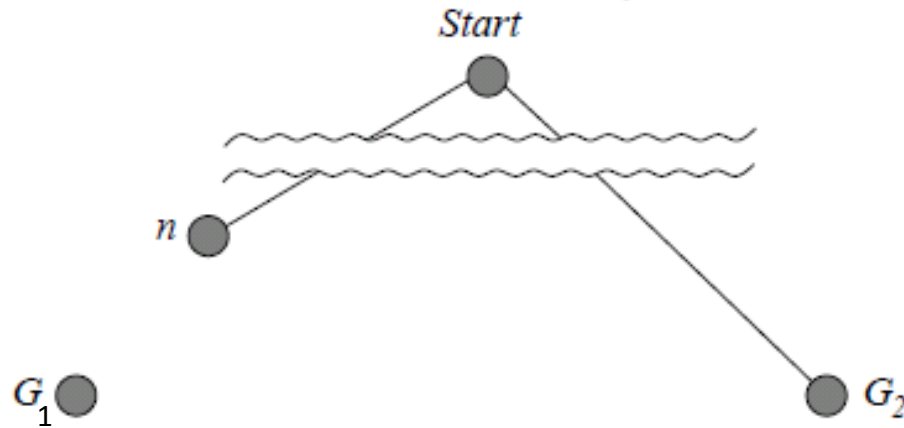
- Space? Keeps all nodes in memory

- Optimal?

Yes

Why optimal? By contradiction

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

A* is “optimally efficient”

- With an admissible heuristic,
 - A* expands *all* nodes with $f(n) < C$
 - A* expands *some* nodes with $f(n) = C$
 - A* expands *no* nodes with $f(n) > C$
- So, except for the variable (usually small) number of nodes with $f(n) = C$,
 - No optimal algorithm using h expands fewer nodes than A*



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)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

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)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

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

Dominance

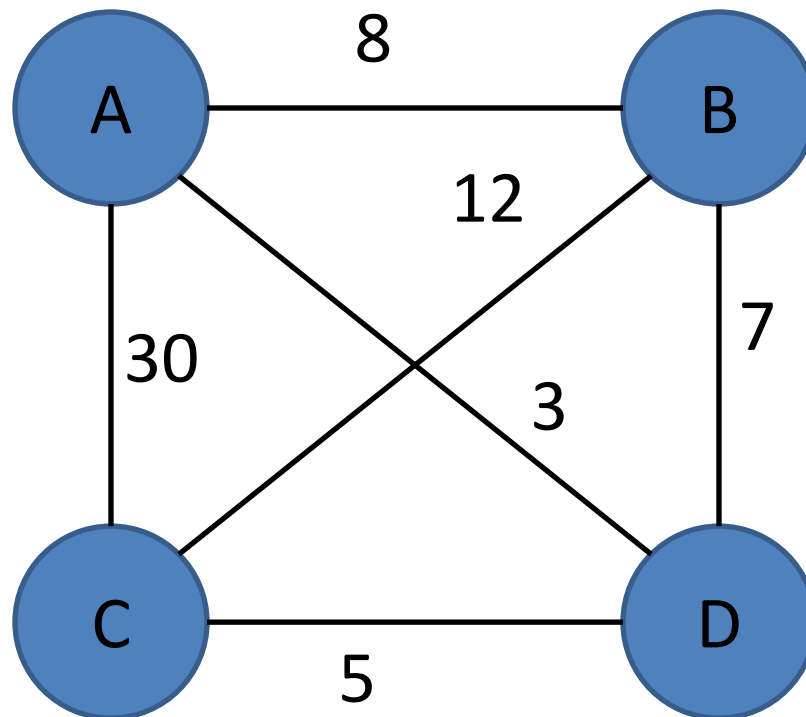
- If $h_2(n) \geq h_1(n)$ for all n (both admissible)
then h_2 **dominates** h_1
- h_2 is at least as good as h_1 for search, and likely better
 - Why?
- Typical search costs (average number of nodes expanded):
 - $d=12$ IDS = 3,644,035 nodes
 $A^*(h_1) = 227$ nodes
 $A^*(h_2) = 73$ nodes
 - $d=24$ IDS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

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

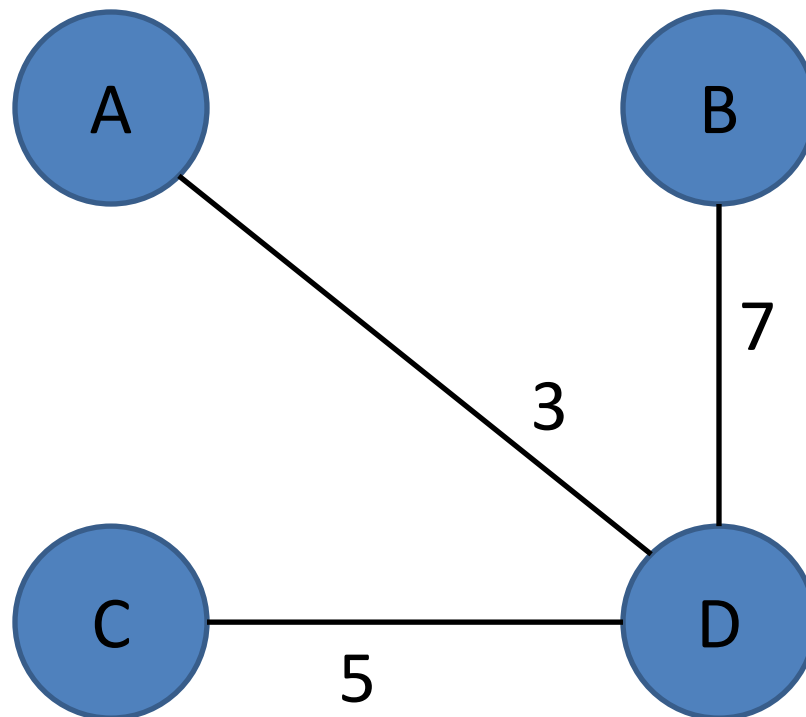
Traveling Salesman Problem

- Goal: find the least-cost cycle in the graph that visits each node exactly once



TSP Relaxed Problem Heuristic

- Relaxed problem: find least-cost *tree* that connects all nodes (minimum spanning tree).
 - $\text{Cost(MST)} \leq \text{Cost(Best Tour} - 1 \text{ edge)} < \text{Cost(Best Tour)}$



Combining Heuristics

- Say we have two heuristics, h_1 and h_2 , and neither dominates the other.
 - What can we do?
- $h_3(n) = \max(h_1(n), h_2(n))$
 - h_3 dominates h_1, h_2

Pattern Databases

*	2	4
*		*
*	3	1

Start State

	1	2
3	4	*
*	*	*

Goal State

- $h(n)$ = cost to get $\{1,2,3,4\}$ in right place
 - Compute once for all possible configurations and store
- Can use multiple sub-problems (e.g., $\{5,6,7,8\}$) and combine with max
 - Or, ignore * moves and *add* disjoint subproblems

Summary of A* Search

- Expands node n with minimum $f(n) = g(n) + h(n)$
= path cost so far + heuristic estimate
- Optimal for *admissible* heuristic $h(n)$
 - I.e. h that underestimates true path cost
- Designing good heuristics is crucial for performance
 - One method: Relaxed problems
- Combining heuristics
 - Take max or add “disjoint” heuristics