

Outline

- Best-first search
 - Greedy best-first search
 - A* search
 - Heuristics
- Local search algorithms
 - Hill-climbing search
 - Beam search
 - Simulated annealing search
 - Genetic algorithms
- Constraint Satisfaction Problems

Motivation

- Uninformed search algorithms are too inefficient
 - they expand far too many unpromising paths
- Example:
 - 8-puzzle

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- Average solution depth = 22
- Breadth-first search to depth 22 has to expand about 3.1×10^{10} nodes

→ try to be more clever with what nodes to expand

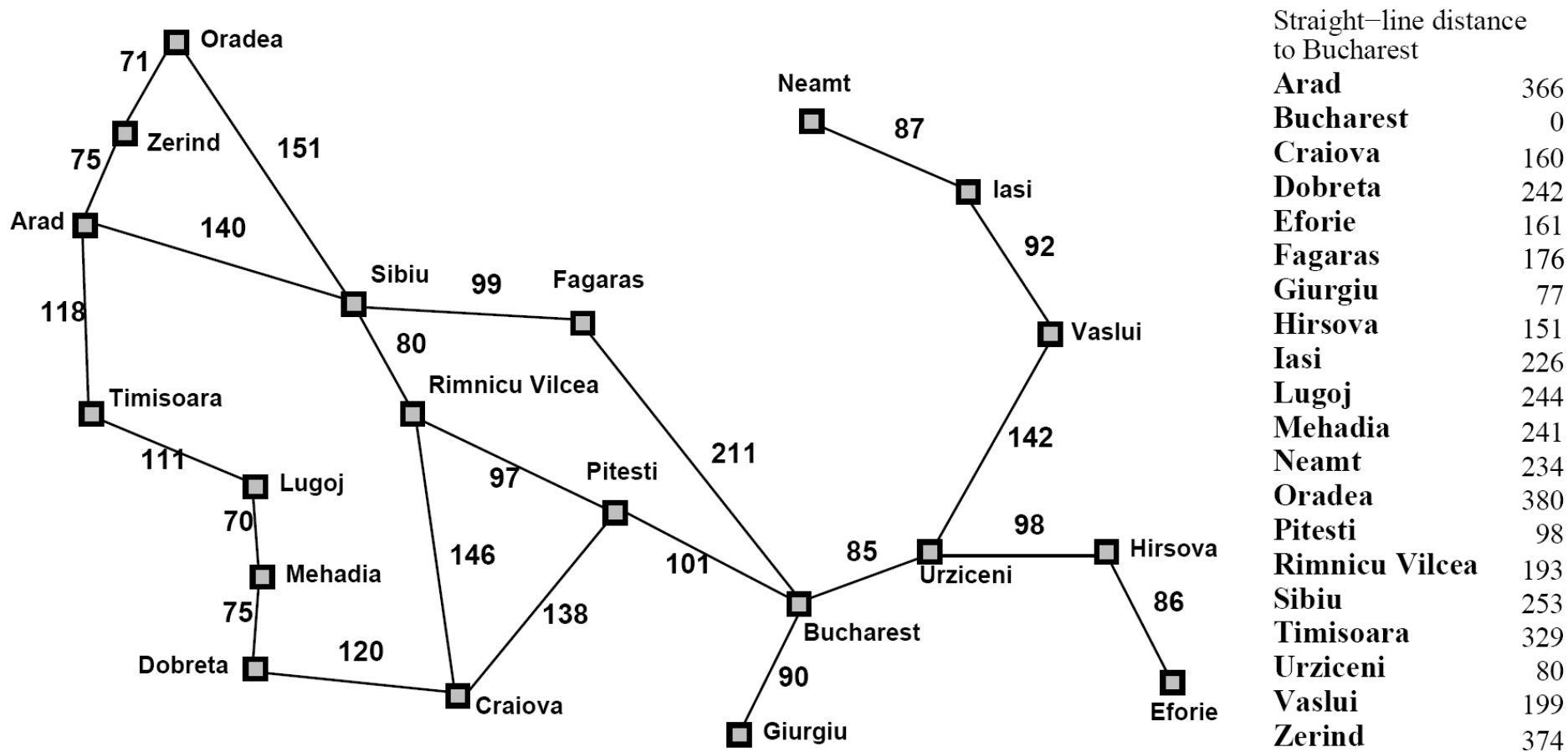
Best-First Search

- Recall
 - Search strategies are characterized by the order in which they expand the nodes of the search tree
 - Uninformed tree-search algorithms sort the nodes by problem-independent methods (e.g., recency)
- Basic Idea of Best-First Search
 - use an **evaluation function** $f(n)$ for each node
 - estimate of the "desirability" of the node's state
 - expand most desirable unexpanded node
- Implementation
 - use Game-Tree-Search algorithm
 - order the nodes in fringe in decreasing order of desirability
- Algorithms
 - Greedy best-first search
 - A* search

Heuristic

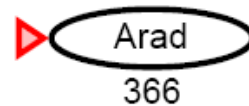
- Greek "heurisko" (εὕρισκω) → "I find"
 - cf. also „Eureka!“
- informally denotes a „rule of thumb“
 - i.e., knowledge that may be helpful in solving a problem
 - note that heuristics may also go wrong!
- In tree-search algorithms, a heuristic denotes a function that estimates the remaining costs until the goal is reached
- Example:
 - straight-line distances may be a good approximation for the true distances on a map of Romania
 - and are easy to obtain (ruler on the map)
 - but cannot be obtained directly from the distances on the map

Romania Example: Straight-line Distances



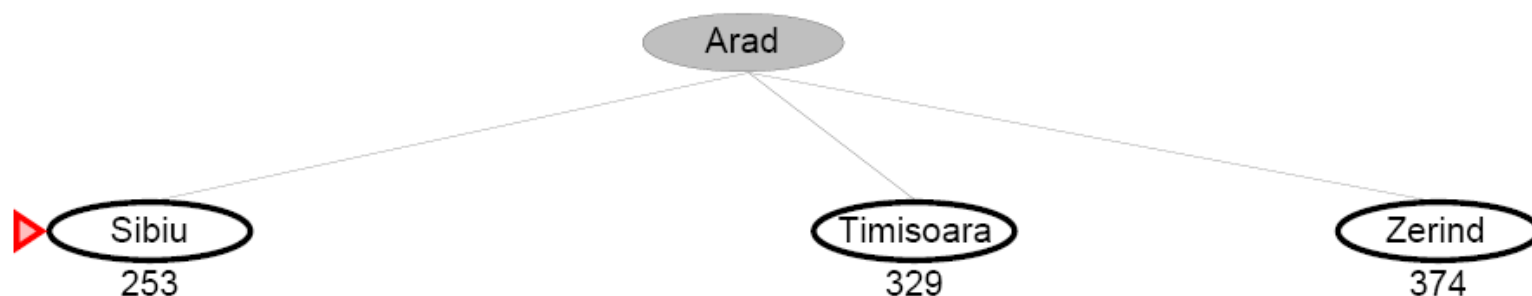
Greedy Best-First Search

- Evaluation function $f(n) = h(n)$ (heuristic)
 - estimates the cost from node 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
 - according to evaluation function
- **Example:**



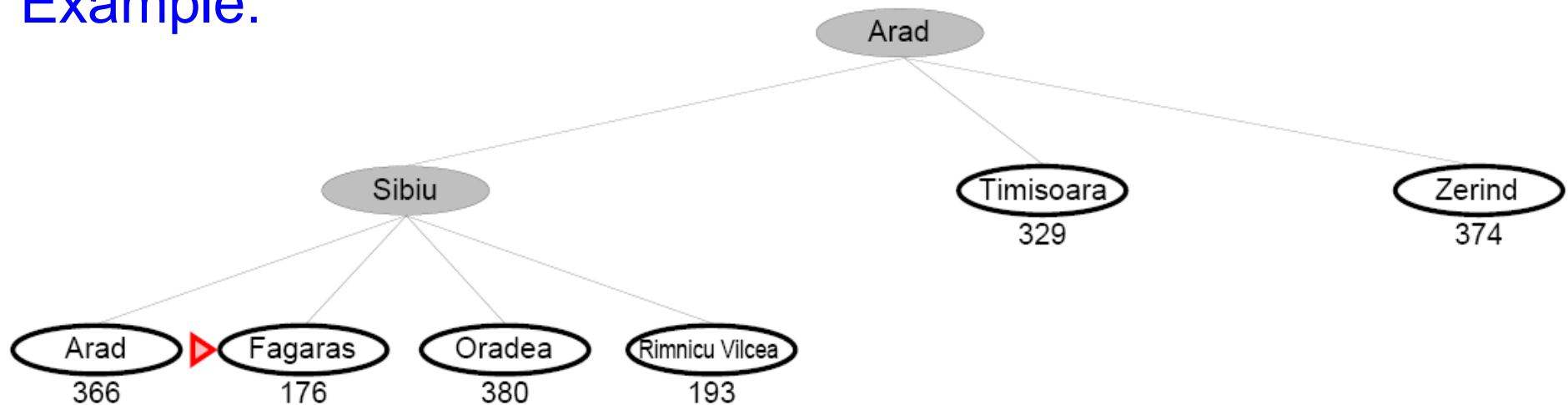
Greedy Best-First Search

- Evaluation function $f(n) = h(n)$ (heuristic)
 - estimates the cost from node 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
 - according to evaluation function
- **Example:**



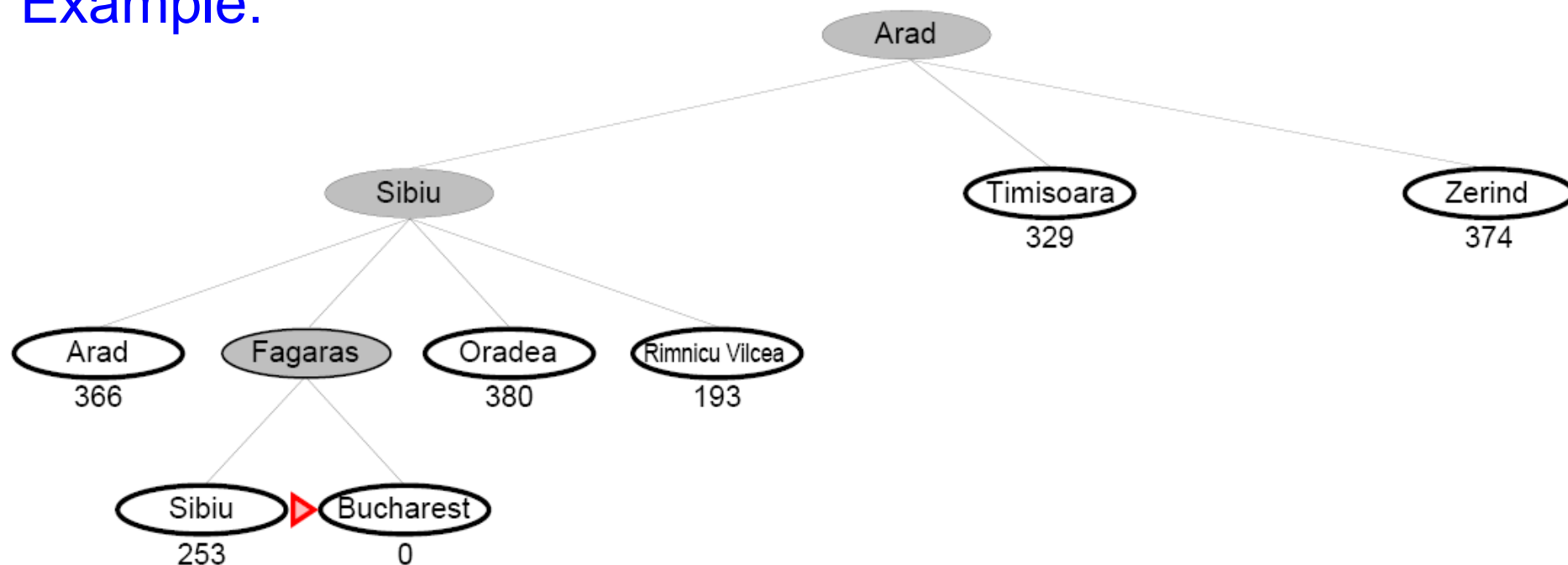
Greedy Best-First Search

- Evaluation function $f(n) = h(n)$ (heuristic)
 - estimates the cost from node 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
 - according to evaluation function
- Example:**



Greedy Best-First Search

- Evaluation function $f(n) = h(n)$ (heuristic)
 - estimates the cost from node 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
 - according to evaluation function
- Example:**

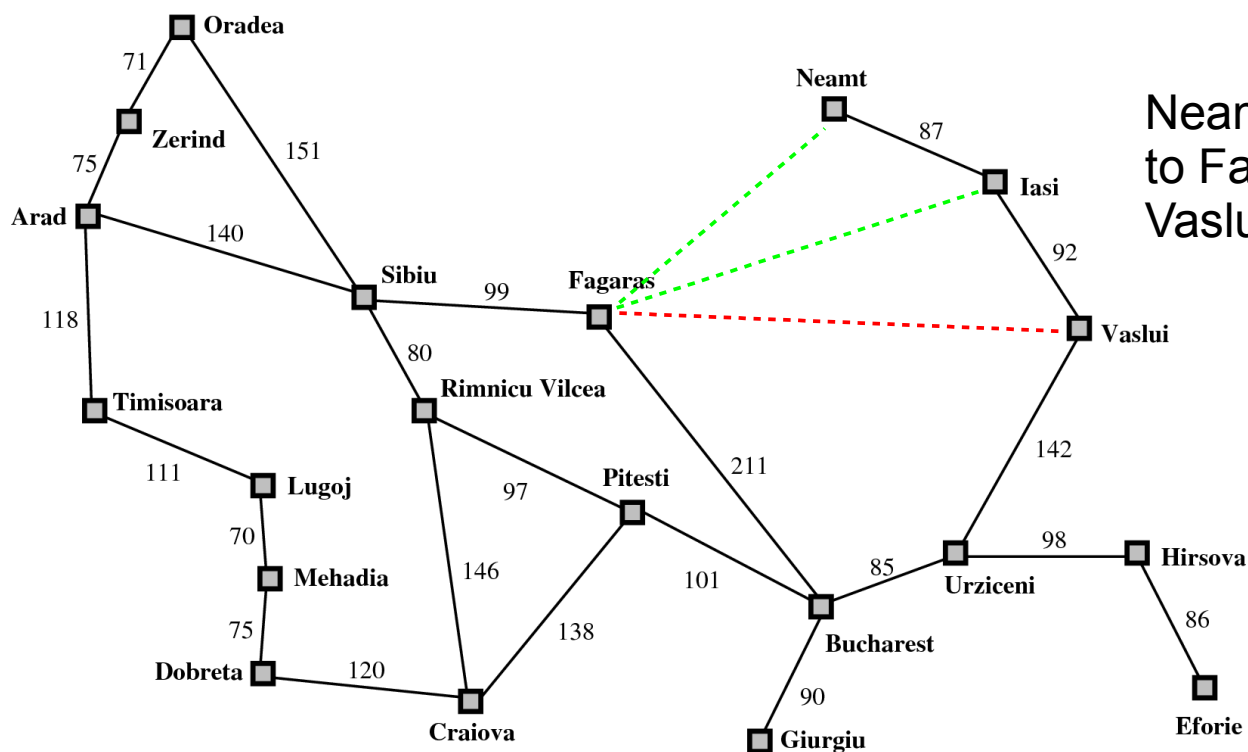


Properties of Greedy Best-First Search

Completeness

- No – can get stuck in loops
- Example: We want to get from Iasi to Fagaras
 - Iasi → Neamt → Iasi → Neamt → ...

Note:
These two are **different** search nodes referring to the same state!



Neamt is closer to Fagaras than Vaslui

Properties of Greedy Best-First Search

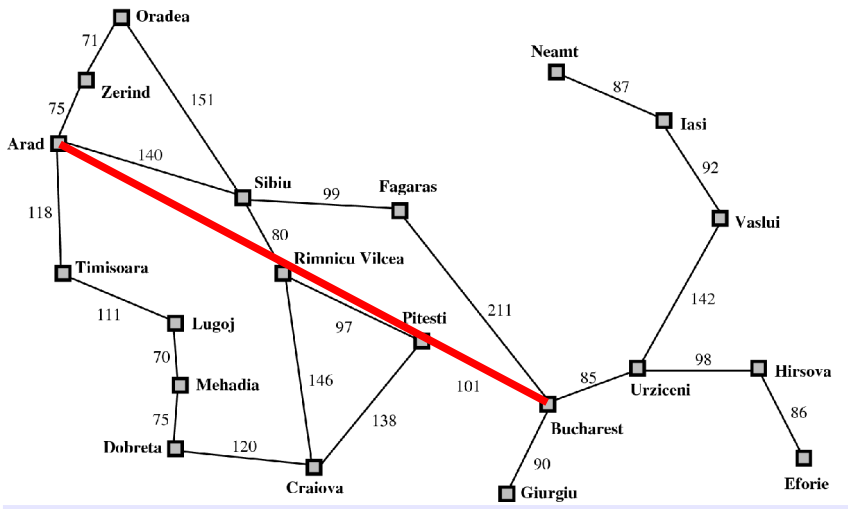
- **Completeness**
 - No – can get stuck in loops
 - can be fixed with careful checking for duplicate states
 - **complete** in finite state space with repeated-state checking
- **Time Complexity**
 - $O(b^m)$, like depth-first search
 - but a good heuristic can give dramatic improvement
 - optimal case: best choice in each step → only d steps
 - a good heuristic improves chances for encountering optimal case
- **Space Complexity**
 - has to keep all nodes in memory → same as time complexity
- **Optimality**
 - **No**
 - Example:
 - solution Arad → Sibiu → Fagaras → Bucharest is not optimal

A* Search

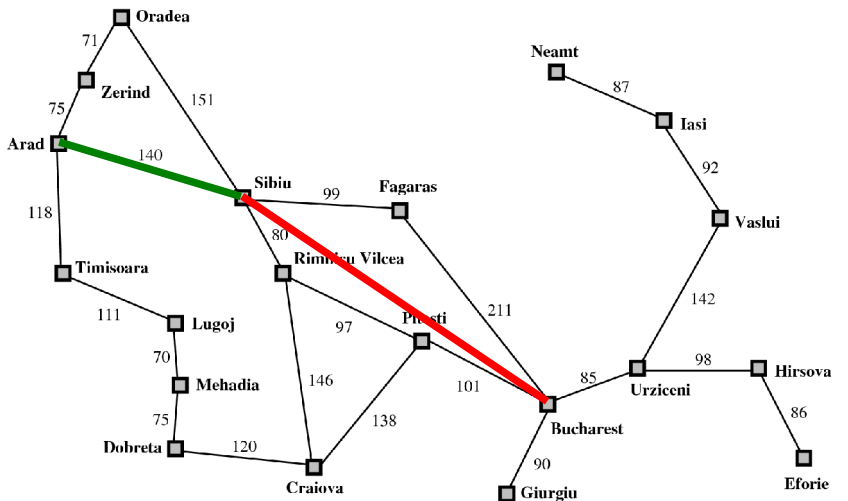
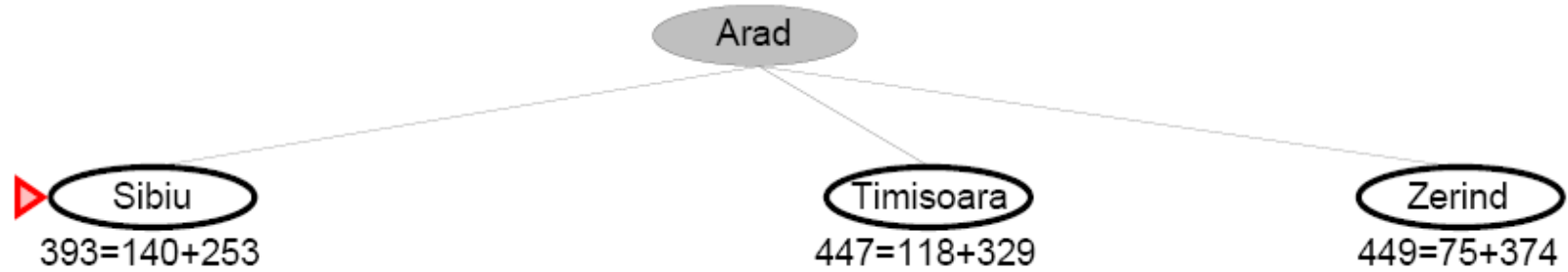
- Best-known form of best-first search
- Basic idea:
 - avoid expanding paths that are already expensive
→ evaluate complete path cost not only remaining costs
- Evaluation function: $f(n) = g(n) + h(n)$
 - $g(n)$ = cost so far to reach node n
 - $h(n)$ = estimated cost to get from n to goal
 - $f(n)$ = estimated cost of path to goal via n

A* Search Example

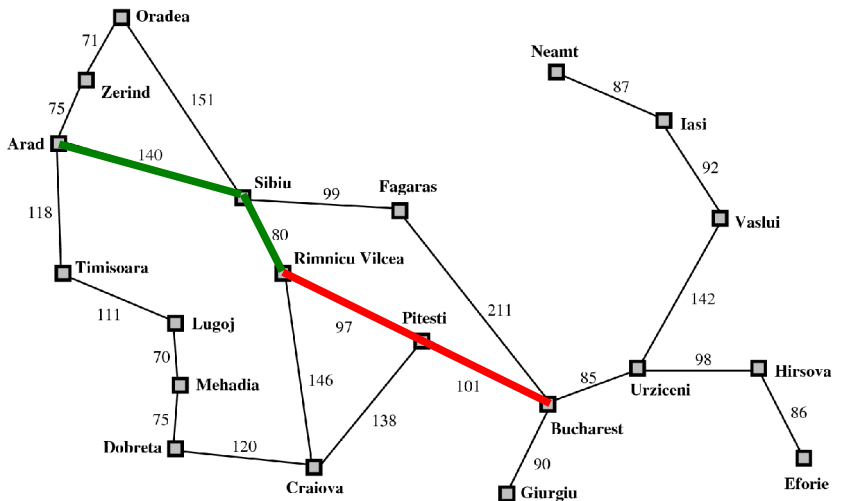
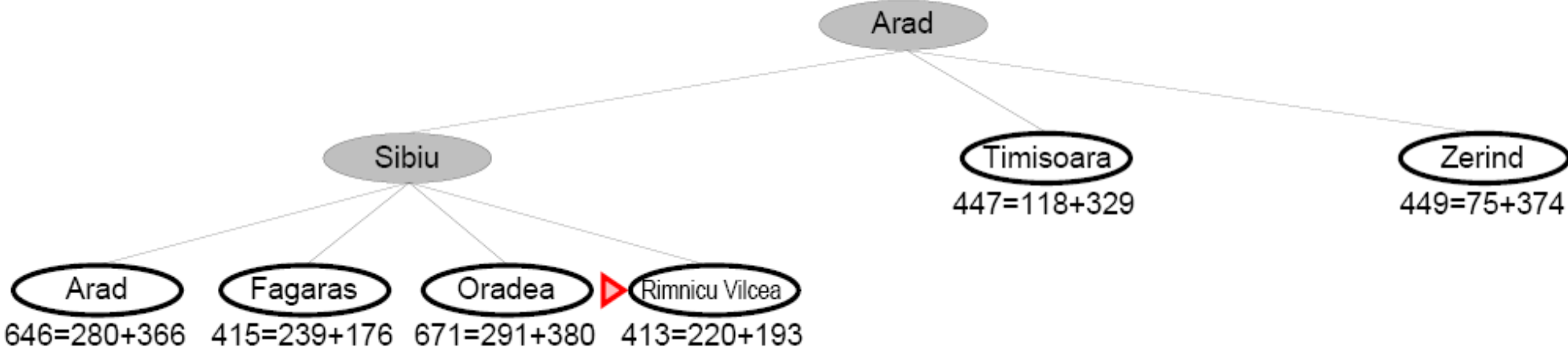
▶ Arad
366=0+366



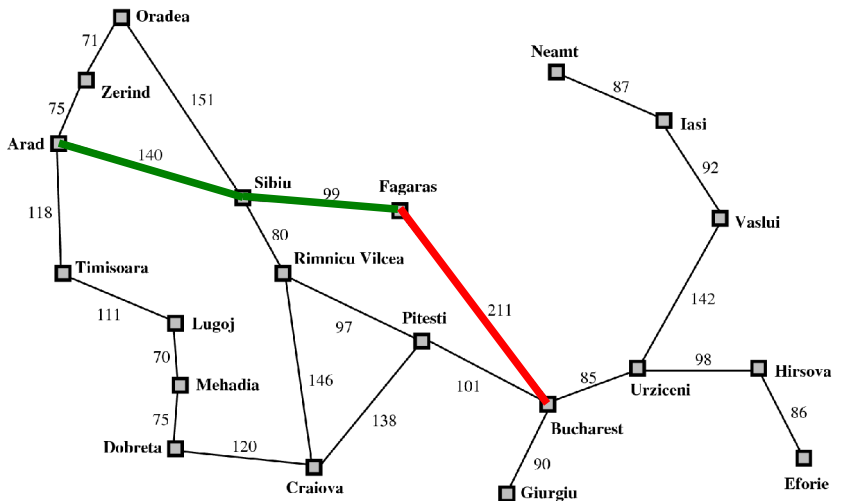
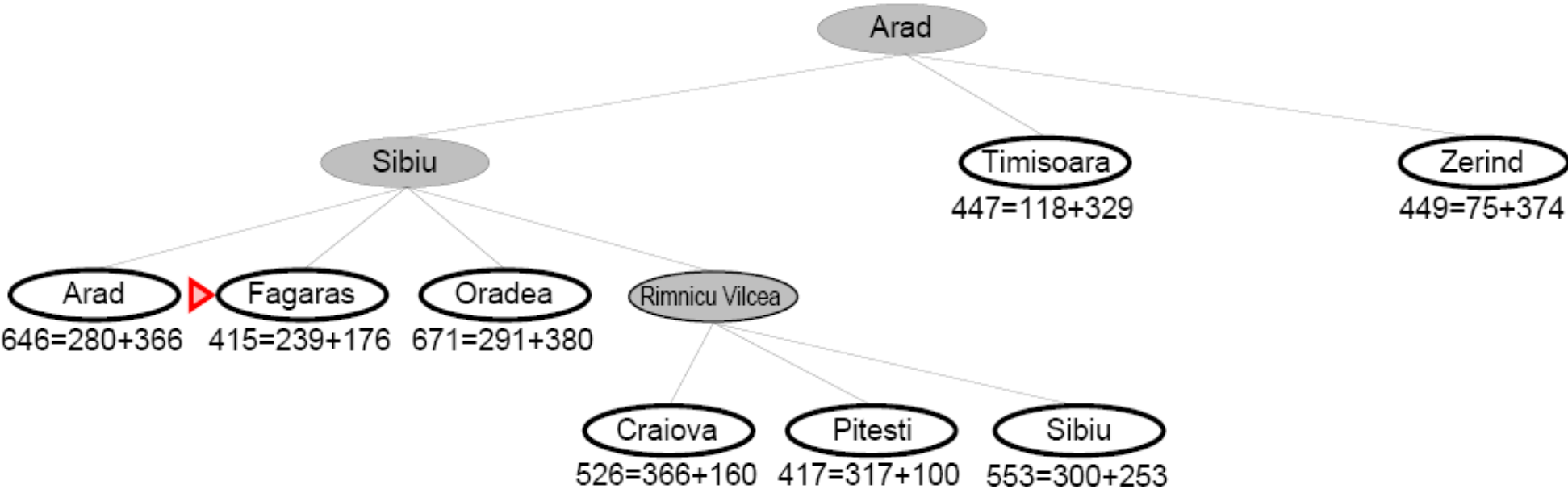
A* Search Example



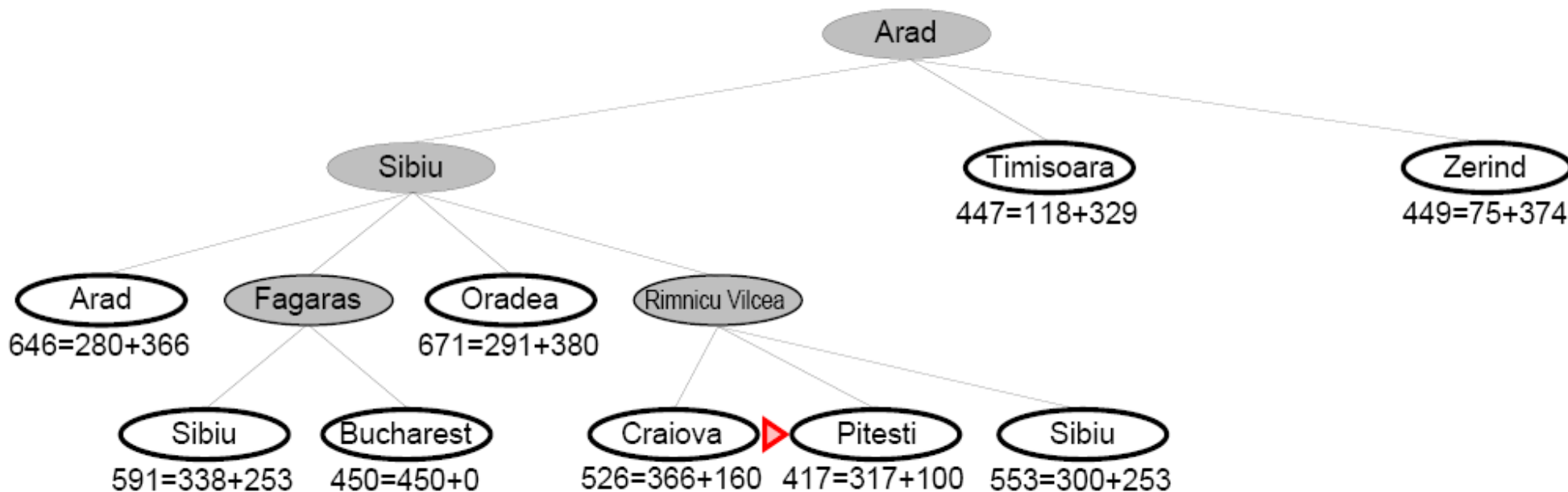
A* Search Example



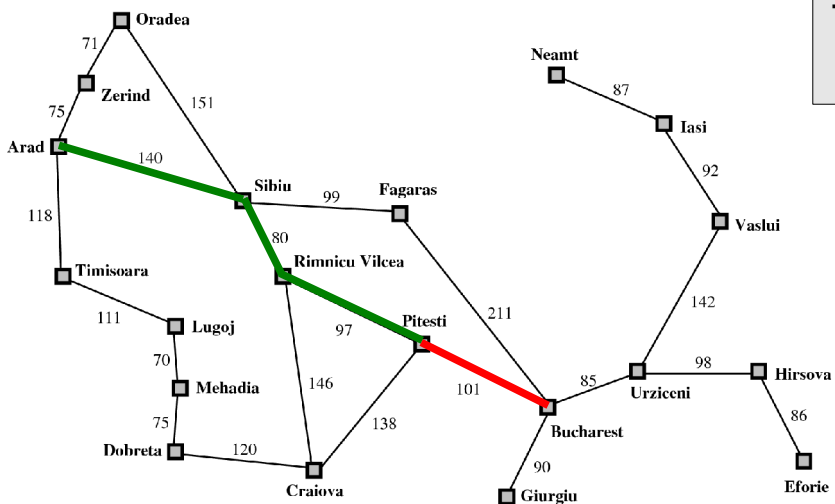
A* Search Example



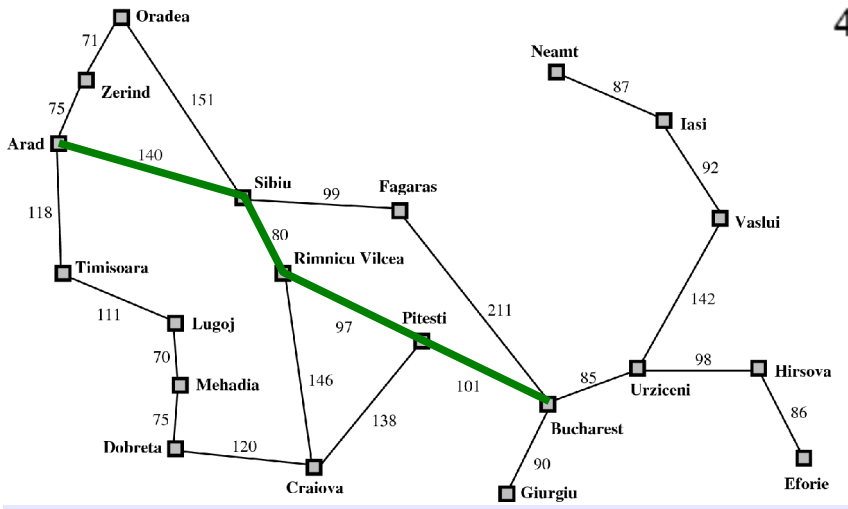
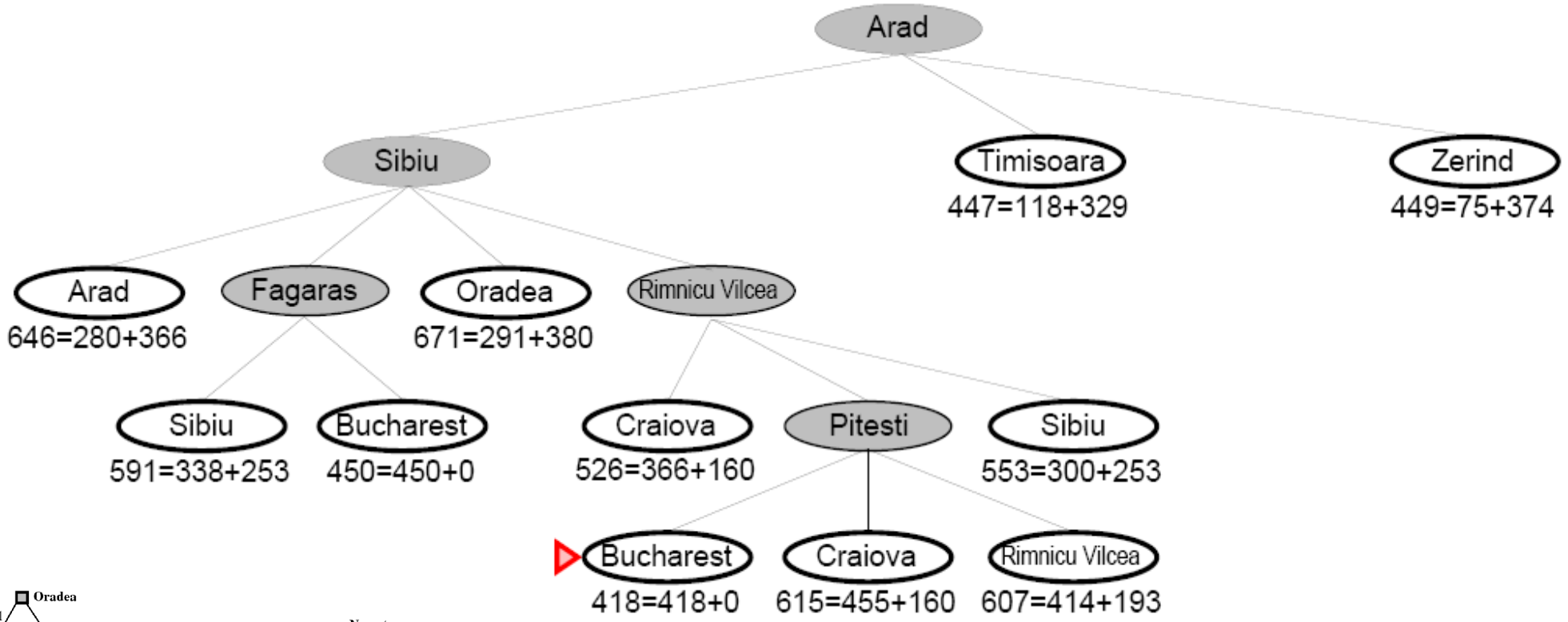
A* Search Example



Note that Pitesti will be expanded even though Bucharest is already in the fringe!
Greedy Search would not do that.



A* Search Example



Properties of A*

- **Completeness**

- **Yes**

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

- **Time Complexity**

- it can be shown that the number of nodes grows exponentially unless the error of the heuristic $h(n)$ is bounded by the logarithm of the value of the actual path cost $h^*(n)$, i.e.

$$|h(n) - h^*(n)| \leq O(\log h^*(n))$$

- **Space Complexity**

- keeps all nodes in memory

- typically the main problem with A*

- **Optimality**

- ???

- following pages

Admissible Heuristics

A heuristic is **admissible** if it **never overestimates** the cost to reach the goal

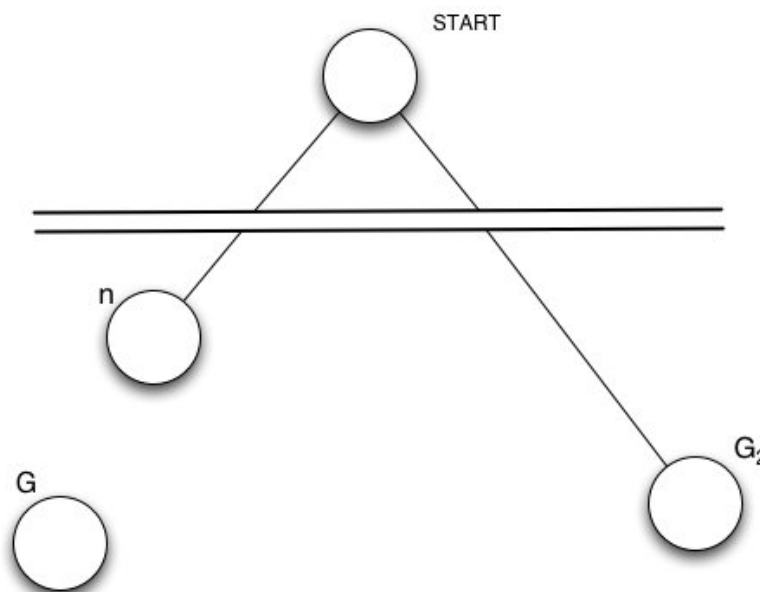
- **Formally:**
 - $h(n) \leq h^*(n)$ if $h^*(n)$ are the true cost from n to goal
- **Example:**
 - Straight-Line Distances h_{SLD} are an admissible heuristics for actual road distances h^*
- **Note:**
 - $h(n) \geq 0$ must also hold, so that $h(goal) = 0$

Theorem

If $h(n)$ is **admissible**, A* using TREE-SEARCH is optimal.

Proof:

Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G with path cost C^* .



Suppose some suboptimal goal G_2 has been generated and is in the fringe.

$$C^* = g(n) + h^*(n)$$

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

$$h(n) \leq h^*(n)$$

because h admissible

$$f(n) \leq C^* < f(G_2)$$

G_2 will never be expanded,
and G will be returned

$g(G_2) > C^*$
because G_2 suboptimal

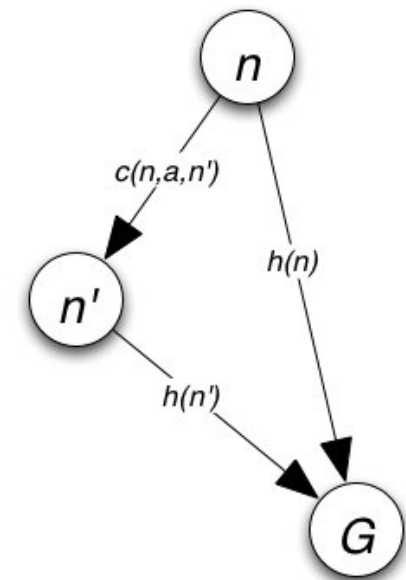
$f(G_2) = g(G_2)$
because $h(G_2) = 0$
(holds for all goal nodes)

Consistent Heuristics

- Graph-Search discards new paths to repeated state even though the new path may be cheaper
 - Previous proof breaks down
- 2 Solutions
 1. Add extra bookkeeping to remove the more expensive path
 2. Ensure that optimal path to any repeated state is always followed first
- Requirement for Solution 2:

A heuristic is **consistent** if for every node n and every successor n' generated by any action a it holds that

$$h(n) \leq c(n, a, n') + h(n')$$



Lemma 1

Every consistent heuristic is admissible.

Proof Sketch:

for all nodes n , in which an action a leads to goal G

$$h(n) \leq c(n, a, G) + h(G) = h^*(n)$$

by induction on the path length from goal, this argument can be extended to all nodes, so that eventually

$$\forall n : h(n) \leq h^*(n)$$

-
- Note:
 - not every admissible heuristic is consistent
 - but most of them are
 - it is hard to find non-consistent admissible heuristics

Lemma 2

If $h(n)$ is **consistent**, then the values of $f(n)$ along any path are **non-decreasing**.

Proof:

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

Theorem

If $h(n)$ is **consistent**, A^* is optimal.

Proof:

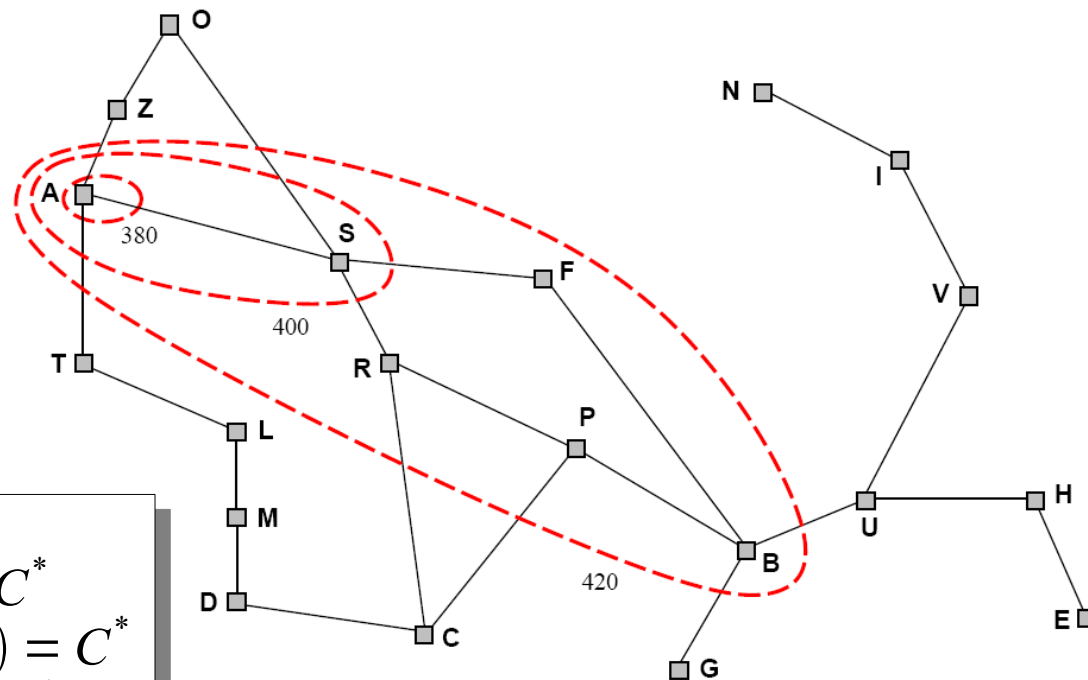
A^* expands nodes in order of increasing f value

Contour labelled f_i
contains all nodes
with $f(n) < f_i$

Contours expand gradually
Cannot expand f_{i+1} until f_i is finished.

Eventually

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



Memory-Bounded Heuristic Search

- Space is the main problem with A^*
- Some **solutions** to A^* space problems
(maintaining completeness and optimality)
 - **Iterative-deepening A^* (IDA*)**
 - like iterative deepening
 - cutoff information is the f -cost ($g + h$) instead of depth
 - **Recursive best-first search (RBFS)**
 - recursive algorithm that attempts to mimic standard best-first search with linear space.
 - keeps track of the f -value of the best alternative path available from any ancestor of the current node
 - **(Simple) Memory-bounded A^* ((S)MA*)**
 - drop the worst leaf node when memory is full

Admissible Heuristics: 8-Puzzle

- $h_{\text{MIS}}(n)$ = number of misplaced tiles
 - admissible because each misplaced tile must be moved at least once
- $h_{\text{MAN}}(n)$ = total Manhattan distance
 - i.e., no. of squares from desired location of each tile
 - admissible because this is the minimum distance of each tile to its target square
- **Example:**

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

$$h_{\text{MIS}}(\text{start}) = 8$$

$$h_{\text{MAN}}(\text{start}) = 18$$

$$h^*(\text{start}) = 26$$

Effective Branching Factor

- Evaluation Measure for a search algorithm:
 - assume we searched N nodes and found solution in depth d
 - the effective branching factor b^* is the branching factor of a uniform tree of depth d with $N+1$ nodes, i.e.

$$1 + N = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

- Measure is fairly constant for different instances of sufficiently hard problems
 - Can thus provide a good guide to the heuristic's overall usefulness.
 - A good value of b^* is 1

Efficiency of A* Search

- Comparison of number of nodes searched by A* and Iterative Deepening Search (IDS)
 - average of 100 different 8-puzzles with different solutions
 - Note:** heuristic $h_2 = h_{MAN}$ is always better than $h_1 = h_{MIS}$

d	Suchkosten			Effektiver Verzweigungsfaktor		
	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	3644035	227	73	2,78	1,42	1,24
14	—	539	113	—	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

Dominance

If h_1 and h_2 are admissible, h_2 **dominates** h_1 if $\forall n : h_2(n) \geq h_1(n)$

- if h_2 dominates h_1 it will perform better because it will *always* be closer to the optimal heuristic h^*
- **Example:**
 - h_{MAN} dominates h_{MIS} because if a tile is misplaced, its Manhattan distance is ≥ 1

Theorem: (**Combining admissible heuristics**)

If h_1 and h_2 are two admissible heuristics then

$$h(n) = \max(h_1(n), h_2(n))$$

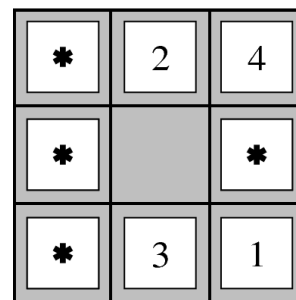
is also admissible and dominates h_1 and h_2

Relaxed Problems

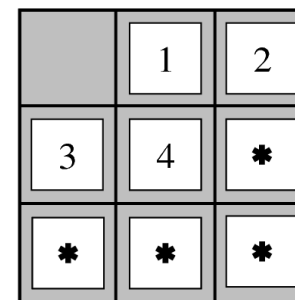
- A problem with fewer restrictions on the actions is called a **relaxed problem**
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- Examples:
 - If the rules of the 8-puzzle are relaxed so that a tile can move **anywhere**, then h_{MIS} gives the shortest solution
 - If the rules are relaxed so that a tile can move to **any adjacent square**, then h_{MAN} gives the shortest solution
- Thus, looking for relaxed problems is a good strategy for **inventing admissible heuristics**.

Pattern Databases

- Admissible heuristics can also be derived from the solution cost of a subproblem of a given problem.
 - This cost is a lower bound on the cost of the real problem.
- Pattern databases store the **exact solution** (length) for every possible **subproblem** instance
 - constructed once for all by searching backwards from the goal and recording every possible pattern
- Example:**
 - store exact solution costs for solving 4 tiles of the 8-puzzle
 - sample pattern:



Start State



Goal State

Learning of Heuristics

- Another way to find a heuristic is through learning from experience
- Experience:
 - states encountered when solving lots of 8-puzzles
 - states are encoded using features, so that similarities between states can be recognized
- Features:
 - for the 8-puzzle, features could, e.g. be
 - the number of misplaced tiles
 - number of pairs of adjacent tiles that are also adjacent in goal
 - ...
- An inductive learning algorithm can then be used to predict costs for other states that arise during search.
- No guarantee that the learned function is admissible!

Summary

- Heuristic functions estimate the costs of shortest paths
- Good heuristics can dramatically reduce search costs
- Greedy best-first search expands node with lowest estimated remaining cost
 - incomplete and not always optimal
- A* search minimizes the path costs so far plus the estimated remaining cost
 - complete and optimal, also **optimally efficient**:
 - no other search algorithm can be more efficient, because they all have search the nodes with $f(n) < C^*$
 - otherwise it could miss a solution
- Admissible search heuristics can be derived from exact solutions of reduced problems
 - problems with less constraints
 - subproblems of the original problem