Introduction to Artificial Intelligence

Informed Search

Bernhard Beckert

UNIVERSITÄT KOBLENZ-LANDAU

Wintersemester 2003/2004

Outline

- **Best-first search**
- **A* search**
- **Heuristics**
- **Hill climbing**
- **Iterative improvement algorithms**

Review: Tree search

```
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-FIRST(fringe)
      if GOAL-TEST[problem] applied to STATE(node) succeeds then
         return node
      else
         fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
   end
```

Strategy

Defines the order of node expansion

Best-first search

Idea

Use an evaluation function for each node (estimate of "desirability")

Expand most desirable unexpanded node

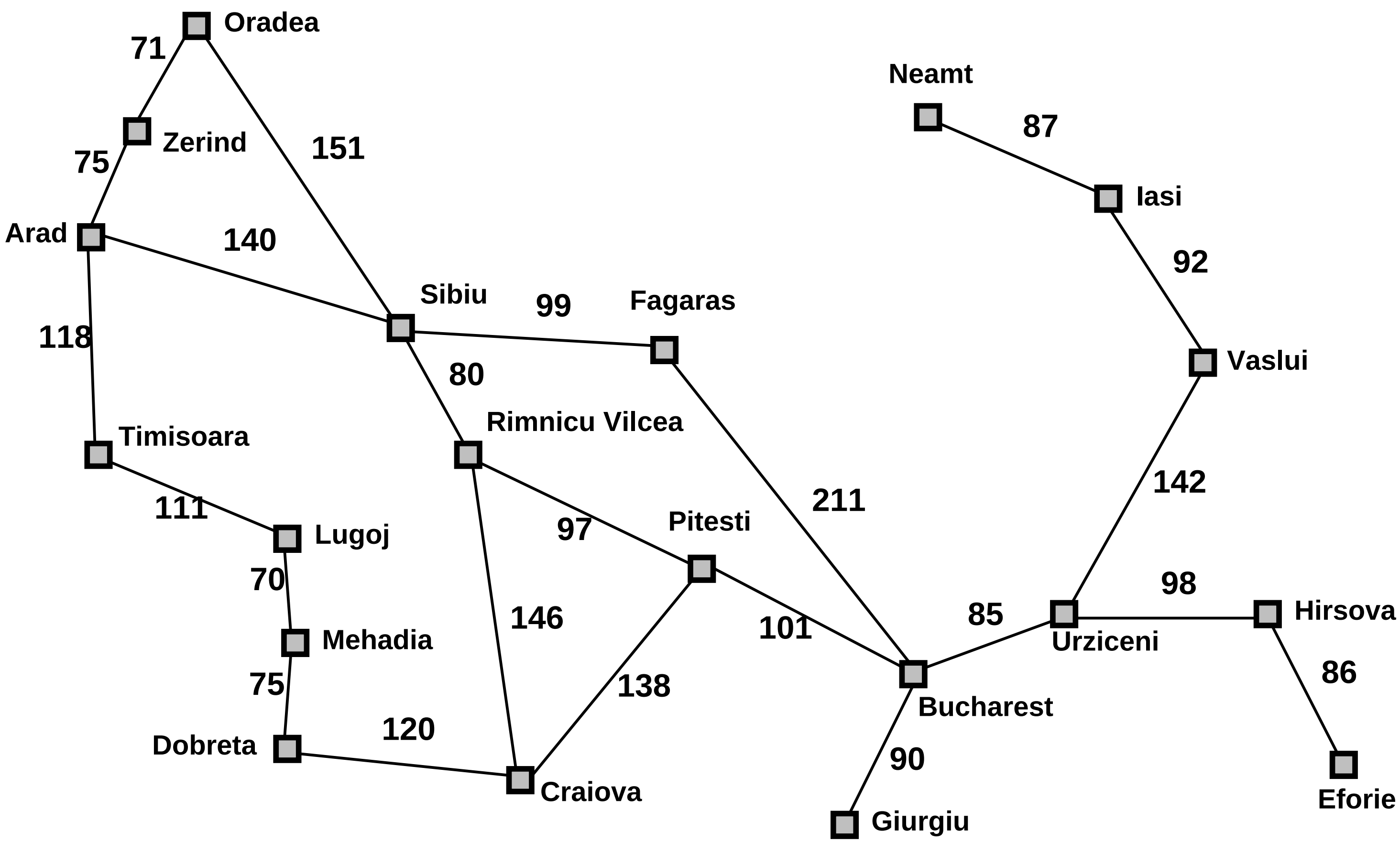
Implementation

***fringe* is a queue sorted in decreasing order of desirability**

Special cases

- **Greedy search**
- **A* search**

Romania with step costs in km



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

Heuristic

Evaluation function

$$h(n) \quad = \quad \textbf{estimate of cost from } n \textbf{ to } goal$$

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

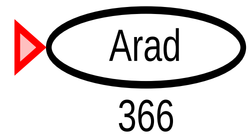
Example

$$h_{\mathbf{SLD}}(n) \quad = \quad \textbf{straight-line distance from } n \textbf{ to Bucharest}$$

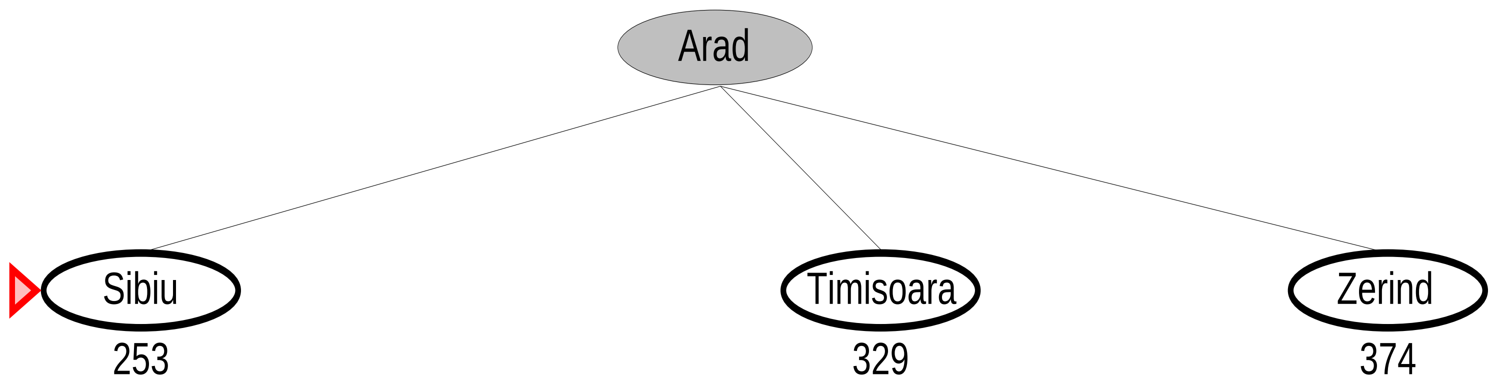
Note

Unlike uniform-cost search the node evaluation function has nothing to do with the nodes explored so far

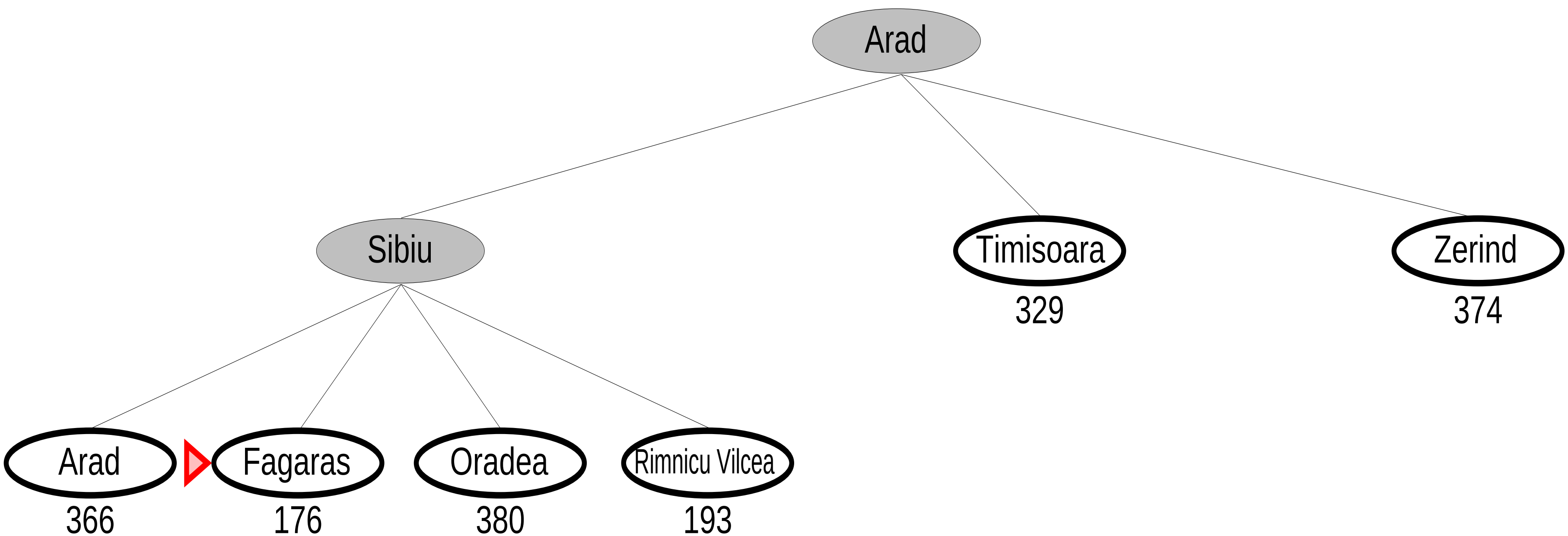
Greedy search: Example Romania



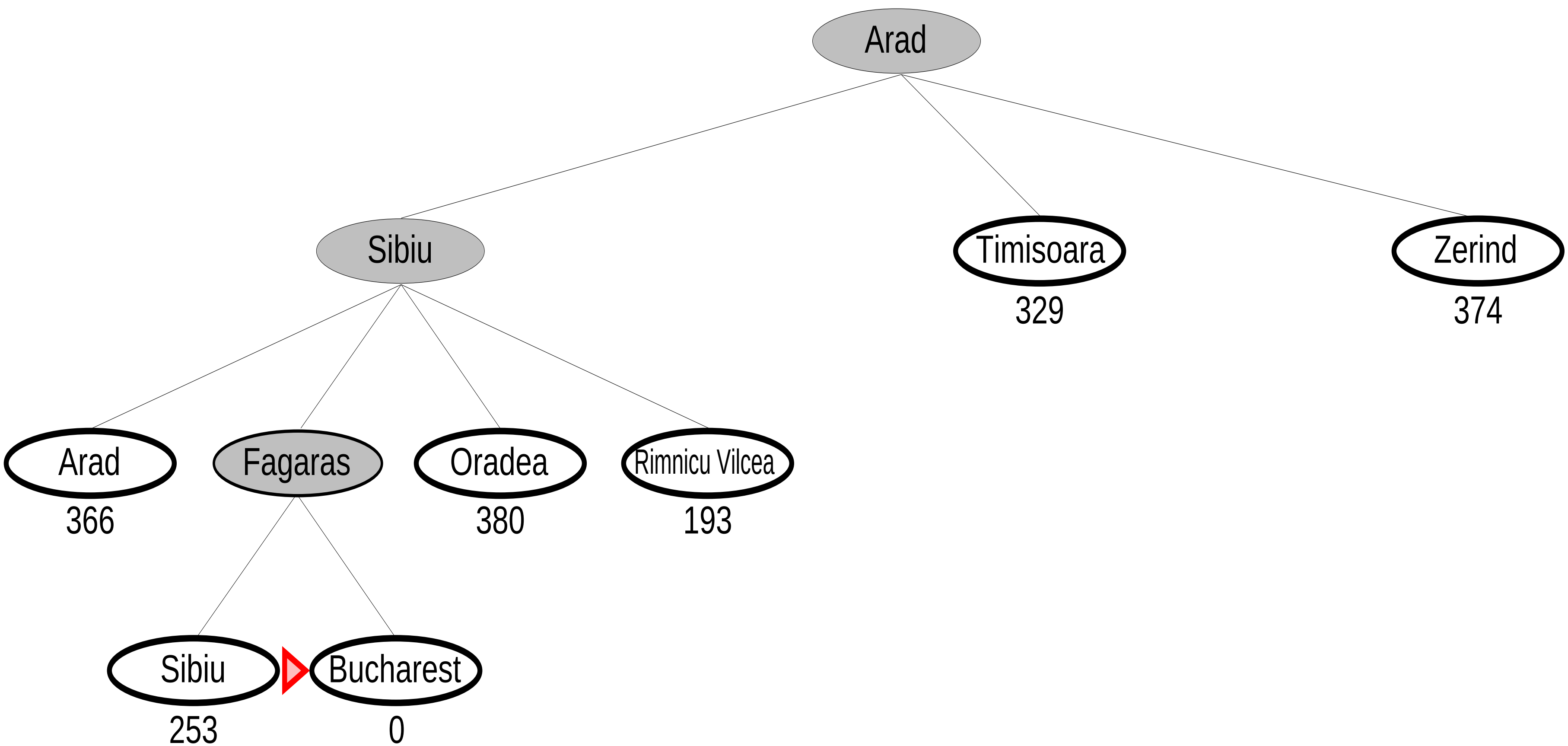
Greedy search: Example Romania



Greedy search: Example Romania



Greedy search: Example Romania



Greedy search: Properties

Complete

Time

Space

Optimal

Greedy search: Properties

Complete No

Can get stuck in loops

Example: Iasi to Oradea

Iasi $\longrightarrow$ Neamt $\longrightarrow$ Iasi $\longrightarrow$ Neamt $\longrightarrow \cdots$

Complete in finite space with repeated-state checking

Time

Space

Optimal

Greedy search: Properties

Complete	**No**
	Can get stuck in loops
	Example: Iasi to Oradea
	Iasi $\longrightarrow$ Neamt $\longrightarrow$ Iasi $\longrightarrow$ Neamt $\longrightarrow \cdots$
	Complete in finite space with repeated-state checking
Time	$O(b^m)$
Space	
Optimal	

Greedy search: Properties

Complete **No**

Can get stuck in loops

Example: Iasi to Oradea

Iasi $\longrightarrow$ Neamt $\longrightarrow$ Iasi $\longrightarrow$ Neamt $\longrightarrow \cdots$

Complete in finite space with repeated-state checking

Time $O(b^m)$

Space $O(b^m)$

Optimal

Greedy search: Properties

Complete No

Can get stuck in loops

Example: Iasi to Oradea

Iasi $\longrightarrow$ Neamt $\longrightarrow$ Iasi $\longrightarrow$ Neamt $\longrightarrow \cdots$

Complete in finite space with repeated-state checking

Time $O(b^m)$

Space $O(b^m)$

Optimal No

Greedy search: Properties

Complete No

Can get stuck in loops

Example: Iasi to Oradea

Iasi $\longrightarrow$ Neamt $\longrightarrow$ Iasi $\longrightarrow$ Neamt $\longrightarrow \cdots$

Complete in finite space with repeated-state checking

Time $O(b^m)$

Space $O(b^m)$

Optimal No

Note

Worst-case time same as depth-first search,

Worst-case space same as breadth-first

But a good heuristic can give dramatic improvement

A* search

Idea

Avoid expanding paths that are already expensive

Evaluation function

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

where

$$\begin{aligned} g(n) &\quad = \quad \textbf{cost so far to reach } n \\ h(n) &\quad = \quad \textbf{estimated cost to goal from } n \\ f(n) &\quad = \quad \textbf{estimated total cost of path through } n \textbf{ to goal} \end{aligned}$$

A* search: Admissibility

Admissibility of heuristic

$h(n)$ **is admissible if**

$$h(n) \leq h^*(n) \qquad \text{for all } n$$

where $h^*(n)$ is the true cost from n to goal

A* search: Admissibility

Admissibility of heuristic

$h(n)$ **is admissible if**

$$h(n) \leq h^*(n) \qquad \textbf{for all } n$$

where $h^*(n)$ **is the true cost from** n **to goal**

Also required

$$h(n) \geq 0 \qquad \textbf{for all } n$$

In particular: $h(G) = 0$ **for goal** G

A* search: Admissibility

Admissibility of heuristic

$h(n)$ **is admissible if**

$$h(n) \leq h^*(n) \qquad \textbf{for all } n$$

where $h^*(n)$ is the true cost from n to goal

Also required

$$h(n) \geq 0 \qquad \textbf{for all } n$$

In particular: $h(G) = 0$ for goal G

Example

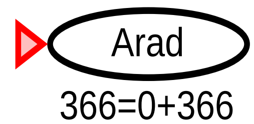
Straight-line distance never overestimates the actual road distance

A* search: Admissibility

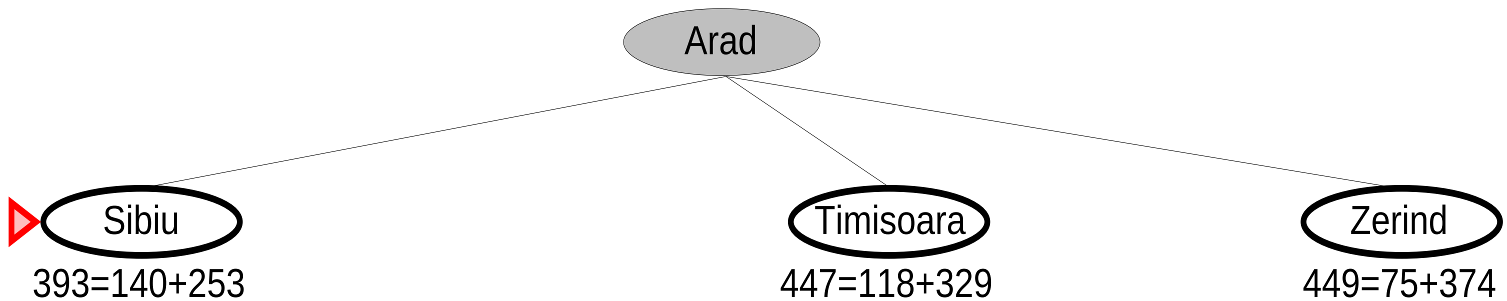
Theorem

A* search with admissible heuristic is optimal

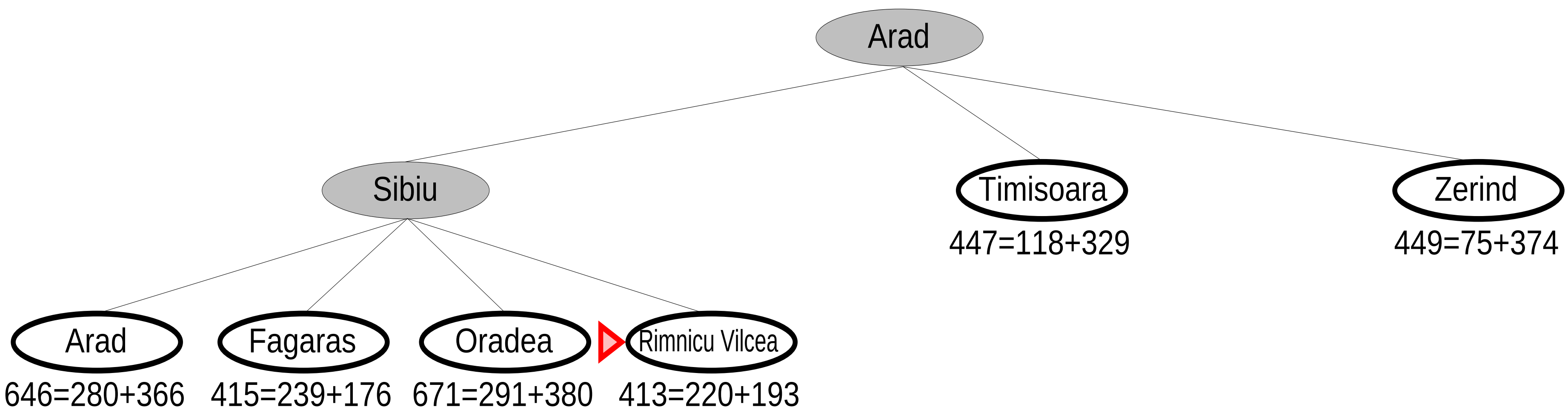
A* search example



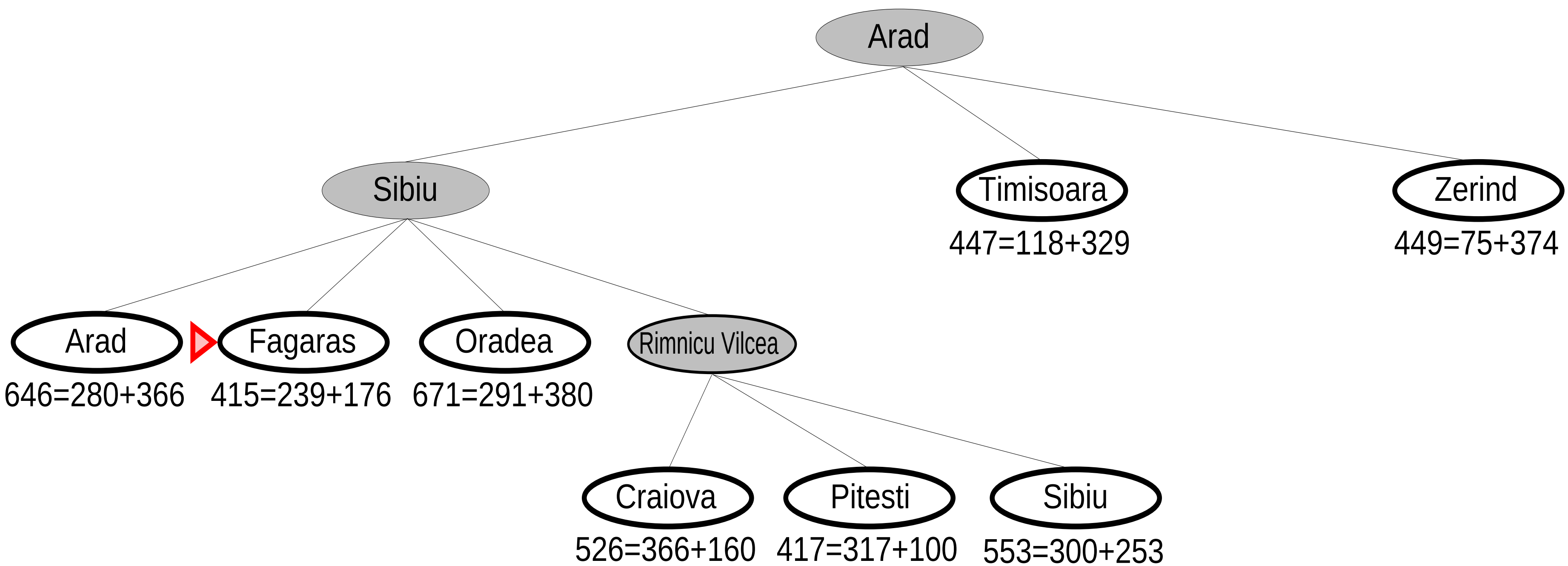
A* search example



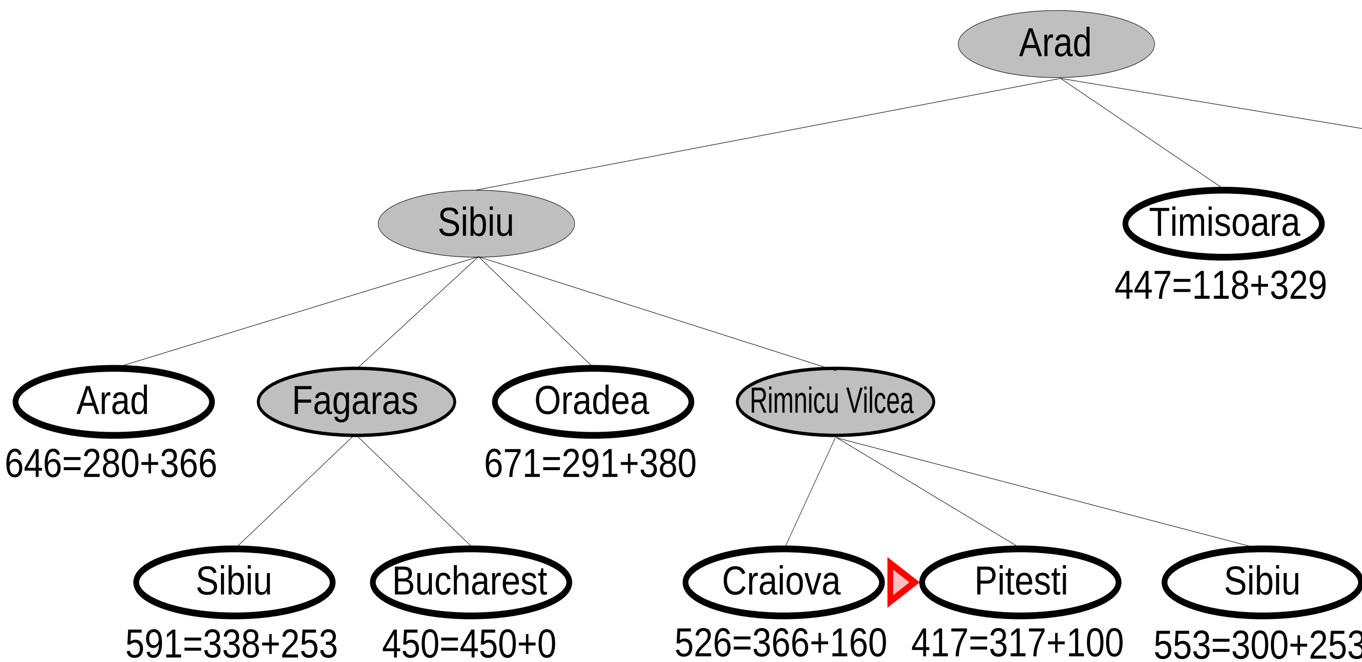
A* search example



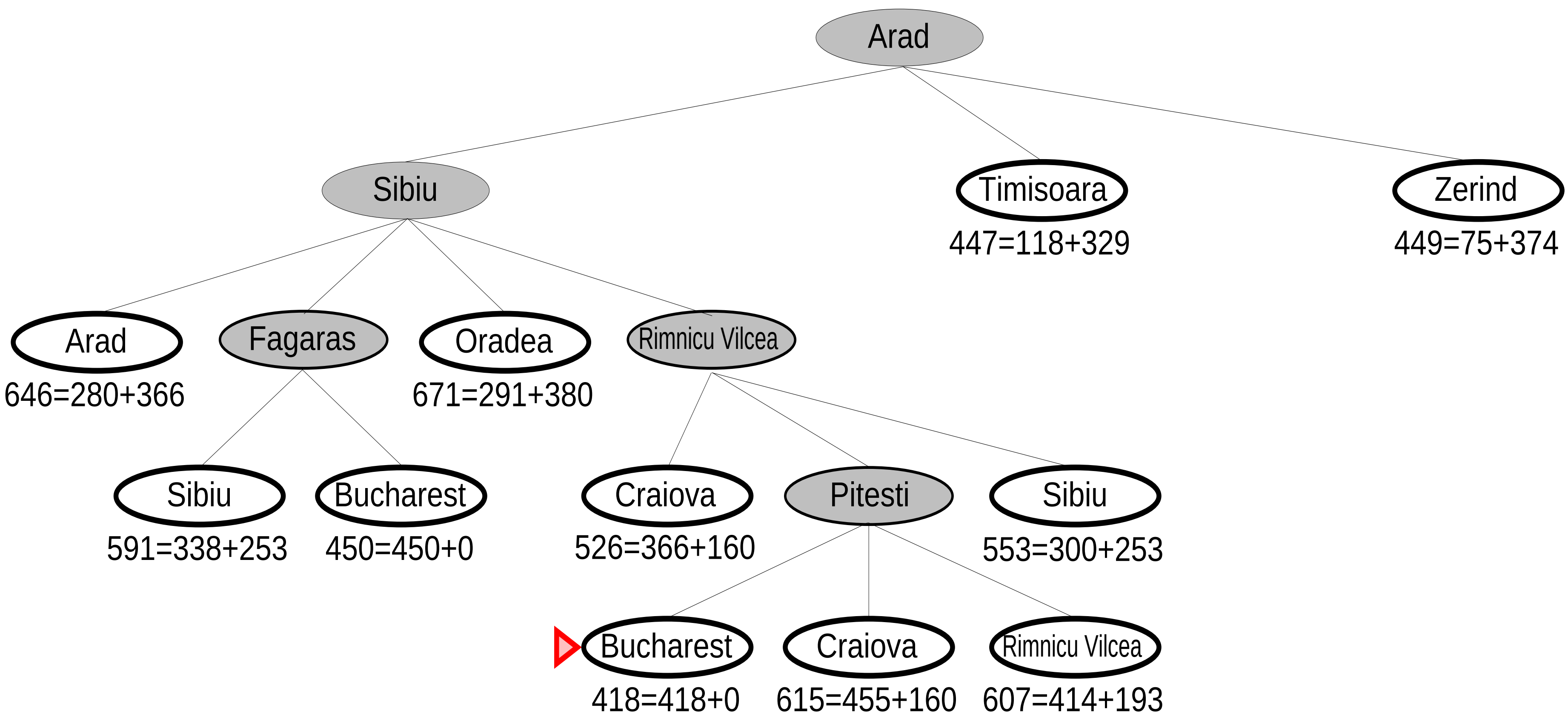
A^* search example



A^* search example

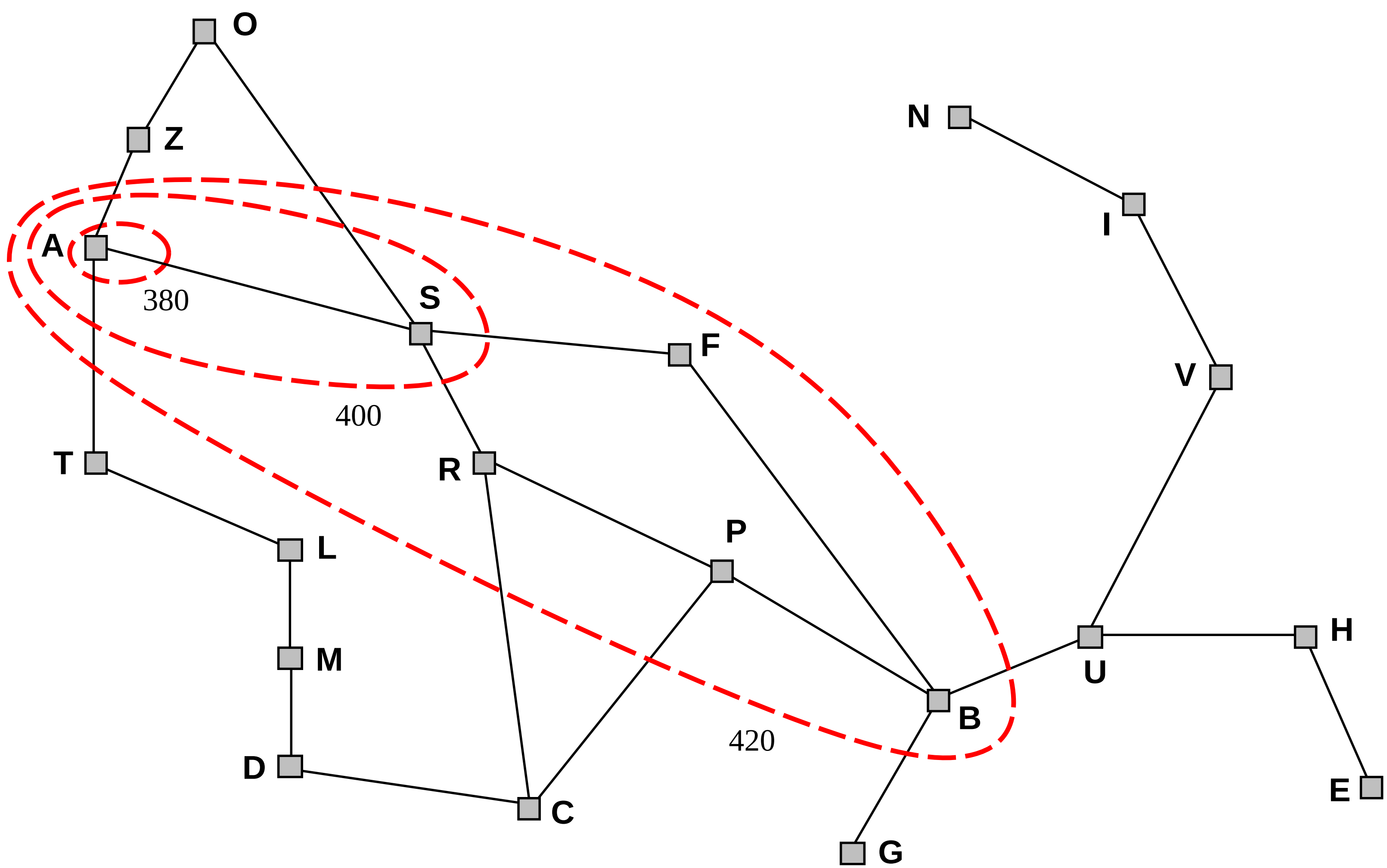


A^* search example

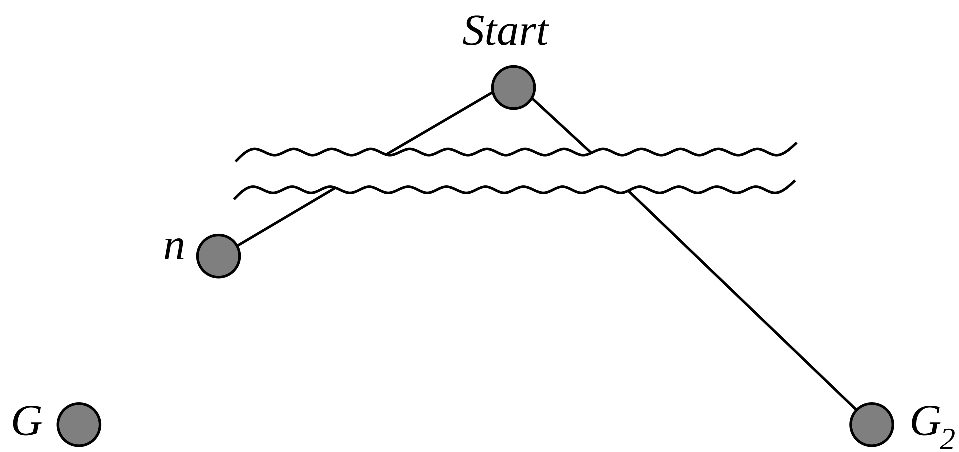


A* search: f-contours

A* gradually adds "f-contours" of nodes



Optimality of A* search: Proof



Suppose a suboptimal goal G_2 has been generated

Let n be an unexpanded node on a shortest path to an optimal goal G

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

Thus, A* never selects G_2 for expansion

A* search: Properties

Complete

Time

Space

Optimal

A* search: Properties

Complete **Yes**

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

Time

Space

Optimal

A* search: Properties

Complete Yes

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

Time Exponential in

[relative error in h $\times$ length of solution]

Space

Optimal

A^* search: Properties

Complete Yes

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

Time Exponential in

[relative error in h $\times$ length of solution]

Space Same as time

Optimal

A* search: Properties

Complete Yes

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

Time Exponential in

[relative error in h $\times$ length of solution]

Space Same as time

Optimal Yes

A* search: Properties

Complete Yes

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

Time Exponential in

[relative error in $h \times$ length of solution]

Space Same as time

Optimal Yes

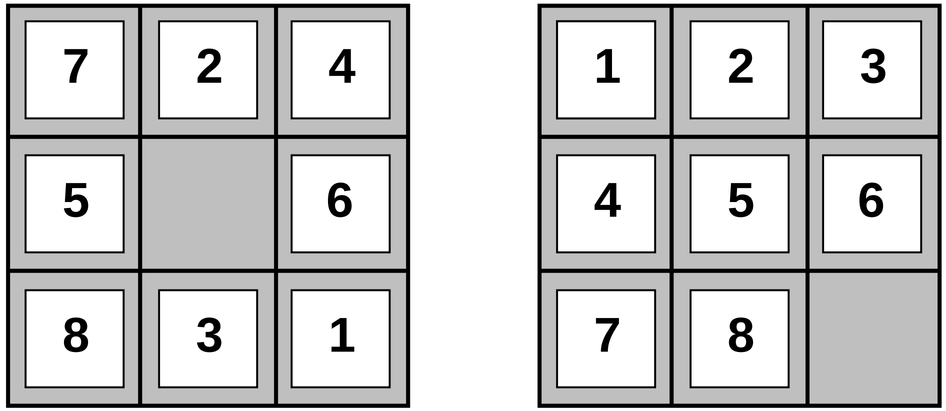
Note

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: Example 8-puzzle



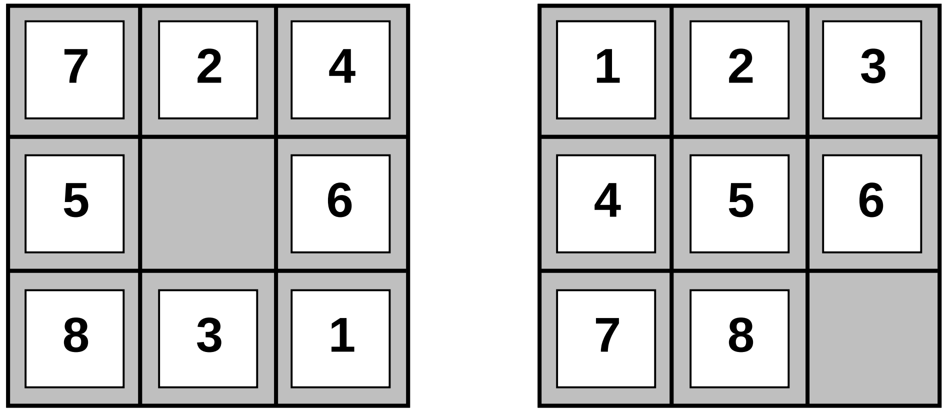
Start State Goal State

Addmissible heuristics

$h_1(n)$ = **number of misplaced tiles**

$h_2(n)$ = **total Manhattan distance**
(i.e., no. of squares from desired location of each tile)

Admissible heuristics: Example 8-puzzle



Start State Goal State

Addmissible heuristics

$h_1(n)$ = number of misplaced tiles

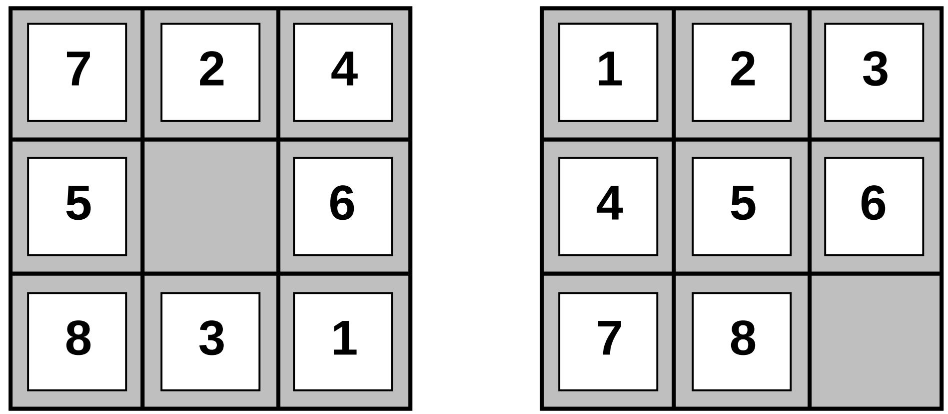
$h_2(n)$ = total M̌anhattan distance
(i.e., no. of squares from desired location of each tile)

In the example

$h_1(S)$ =

$h_2(S)$ =

Admissible heuristics: Example 8-puzzle



Start State Goal State

Addmissible heuristics

$h_1(n)$ = **number of misplaced tiles**

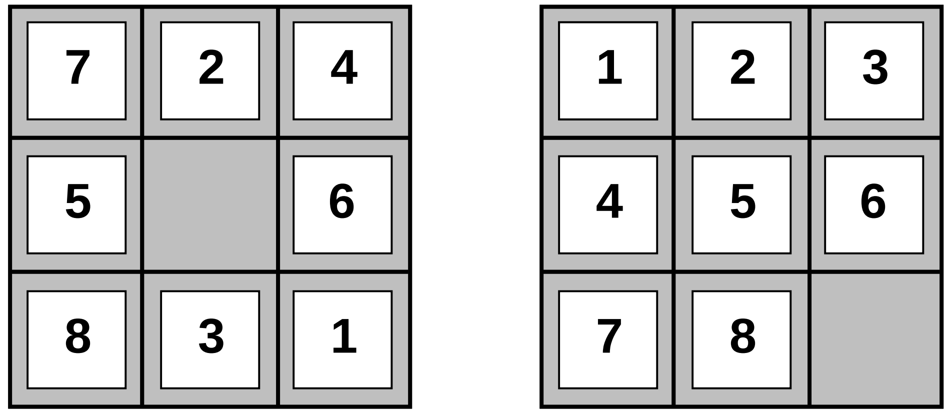
$h_2(n)$ = **total Manhattan distance**
(i.e., no. of squares from desired location of each tile)

In the example

$h_1(S)$ = **6**

$h_2(S)$ =

Admissible heuristics: Example 8-puzzle



Start State Goal State

Addmissible heuristics

$h_1(n)$ = **number of misplaced tiles**

$h_2(n)$ = **total Manhattan distance**
(i.e., no. of squares from desired location of each tile)

In the example

$h_1(S)$ = **6**

$h_2(S)$ = **2 + 0 + 3 + 1 + 0 + 1 + 3 + 4 = 14**

Dominance

Definition

h_1, h_2 **two admissible heuristics**

h_2 **dominates** h_1 **if**

$$h_2(n) \geq h_1(n) \qquad \textbf{for all } n$$

Dominance

Definition

h_1, h_2 **two admissible heuristics**

h_2 **dominates** h_1 **if**

$$h_2(n) \geq h_1(n) \qquad \textbf{for all } n$$

Theorem

If h_2 **dominates** h_1**, then** h_2 **is better for search than** h_1**.**

Dominance: Example 8-puzzle

Typical search costs

$d = 14$	**IDS**	**3,473,941 nodes**
	A$^*(h_1)$	**539 nodes**
	A$^*(h_2)$	**113 nodes**
$d = 24$	**IDS**	**too many nodes**
	A$^*(h_1)$	**39,135 nodes**
	A$^*(h_2)$	**1,641 nodes**

d**:** **depth of first solution**
IDS: **iterative deepening search**

Relaxed problems

Finding good admissible heuristics is an art!

Deriving admissible heuristics

Admissible heuristics can be derived from the *exact* solution cost of a *relaxed* version of the problem

Relaxed problems

Finding good admissible heuristics is an art!

Deriving admissible heuristics

Admissible heuristics can be derived from the *exact* solution cost of a *relaxed* version of the problem

Example

If the rules of the 8-puzzle are relaxed so that a tile can move *anywhere*, then we get heuristic h_1

If the rules are relaxed so that a tile can move to *any adjacent square*, then we get heuristic h_2

Relaxed problems

Finding good admissible heuristics is an art!

Deriving admissible heuristics

Admissible heuristics can be derived from the *exact* solution cost of a *relaxed* version of the problem

Example

If the rules of the 8-puzzle are relaxed so that a tile can move *anywhere*, then we get heuristic h_1

If the rules are relaxed so that a tile can move to *any adjacent square*, then we get heuristic h_2

Key point

The optimal solution cost of a relaxed problem is not greater than the optimal solution cost of the real problem

Hill-climbing (gradient ascent/descent)

Idea

Do not systematically enumerate search space,
rather start anywhere and go to next local optimum.

Depth-first search with heuristic and w/o memory

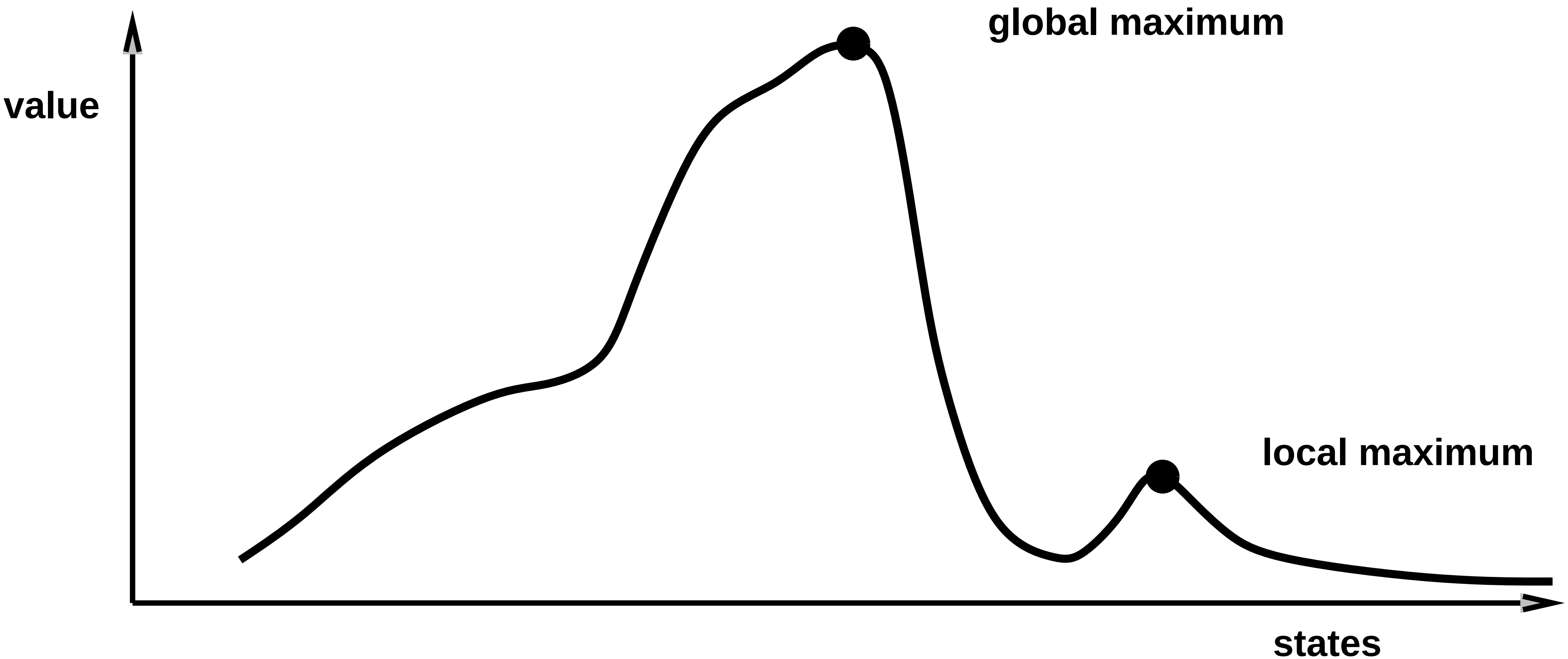
```
function HILL-CLIMBING(problem) returns a state that is a local minimum
   inputs: problem     /* a problem */
   local variables: current     /* a node */
                    neighbour     /* a node */

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      neighbour ← a highest-valued successor of current
      if VALUE[neighbour] < VALUE[current] then return STATE[current]
      current ← neighbour
   end
```

Hill-climbing

Problem

Depending on initial state, can get stuck on local maxima



Hill-climbing: Improvement

Idea

Escape local maxima by allowing some “bad” or random moves.

Various flavors

- **local search**
- **simulated annealing**

Properties

All are incomplete, non-optimal

Sometimes performing well in practice, if (optimal) solutions are dense

Iterative improvement algorithms

Idea

In many search problems, the path is irrelevant;
the goal state itself is the solution

Iterative improvement algorithms

Idea

**In many search problems, the path is irrelevant;
the goal state itself is the solution**

Then

State space = set of “complete” configurations

Iterative improvement

Keep a single “current” state, try to improve it

Similar to depth-first

Advantage

Constant space

Iterative improvement algorithms: Example TSP

Travelling Salesman Problem

Find shortest trip through set of cities
such that each city is visited exactly once

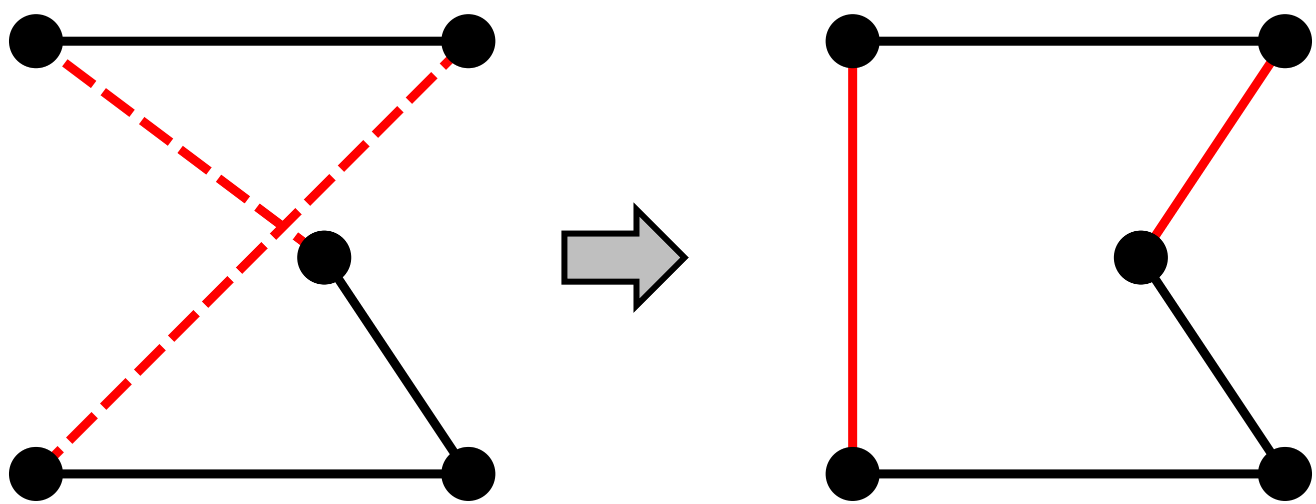
Iterative improvement algorithms: Example TSP

Travelling Salesman Problem

Find shortest trip through set of cities such that each city is visited exactly once

Idea for iterative improvement

Start with any complete tour, perform pairwise exchanges



Iterative improvement algorithms: Example n-queens

n-queens problem

Put n queens on $n \times n$ board
with no two queens in the same row, columns, or diagonal

Iterative improvement algorithms: Example n-queens

n-queens problem

Put n queens on $n \times n$ board
with no two queens in the same row, columns, or diagonal

Idea for iterative improvement

Move a queen to reduce number of conflicts