Informed Search and Exploration

Chapter 4

Outline

- Best-first search
- A* search
- Heuristics
- IDA* search
- Hill-climbing
- Simulated annealing

Review: Tree search

function TREE-SEARCH (*problem, fringe*)
returns a solution, or failure

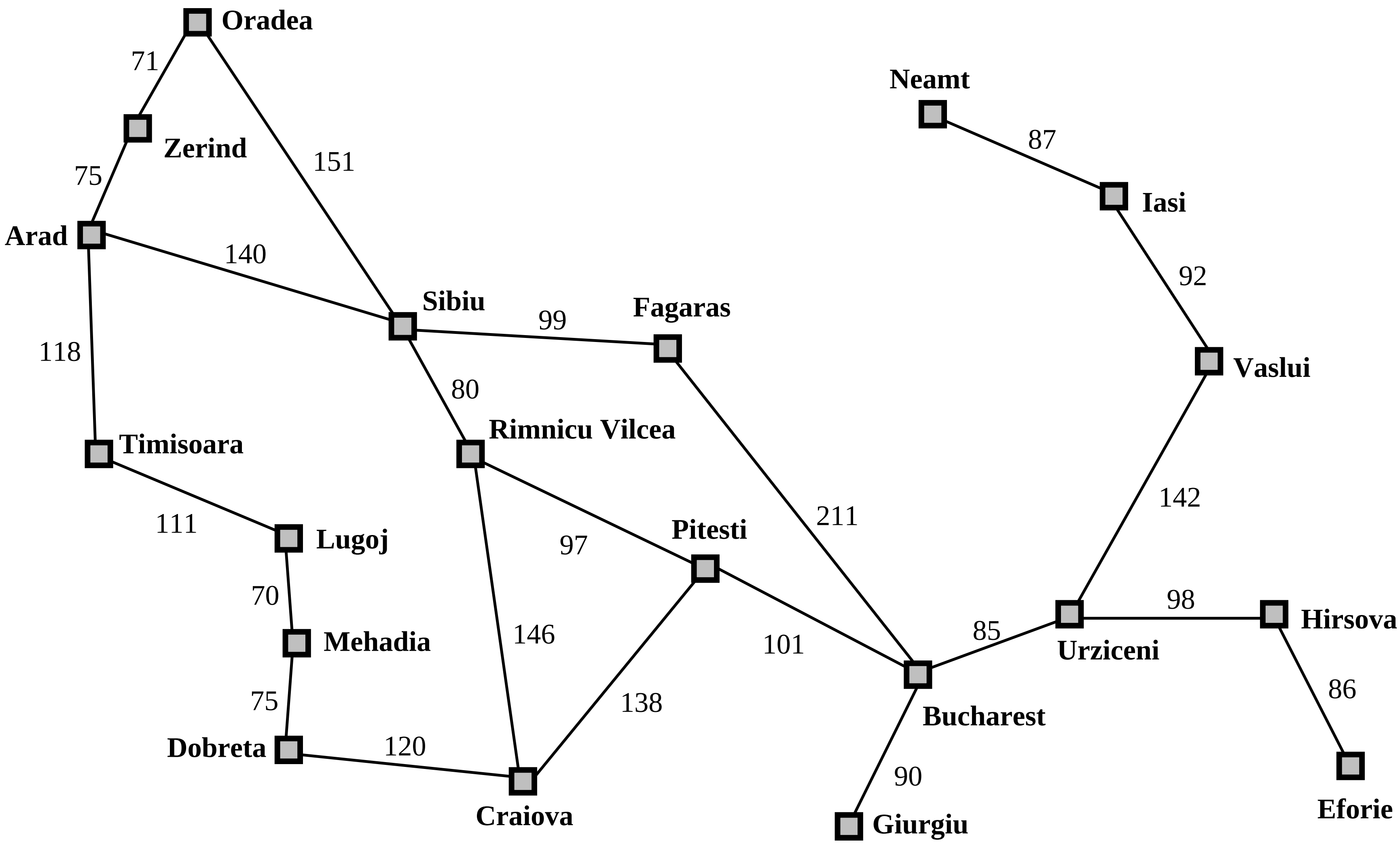
```
fringe ← INSERT(MAKE-NODE(INITIAL-STATE [problem]),fringe)
loop do
  if EMPTY?(fringe) then return failure
  node ← REMOVE-FIRST(fringe)
  if GOAL-TEST[problem] applied to STATE[node] succeeds
    then return SOLUTION(node)
  fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
```

A strategy is defined by picking the *order of node expansion*

Best-first search

- Idea: use an *evaluation function* for each node
  – estimate of "desirability"
- $\Rightarrow$ 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

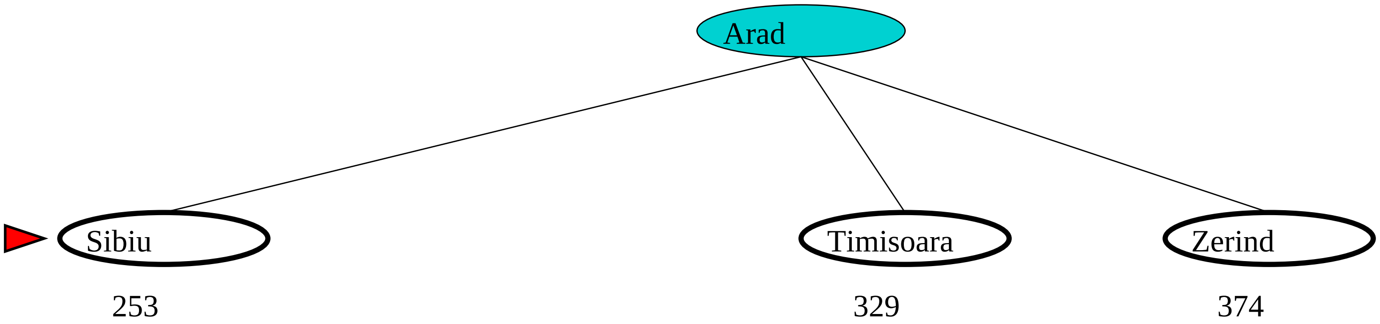


Greedy search

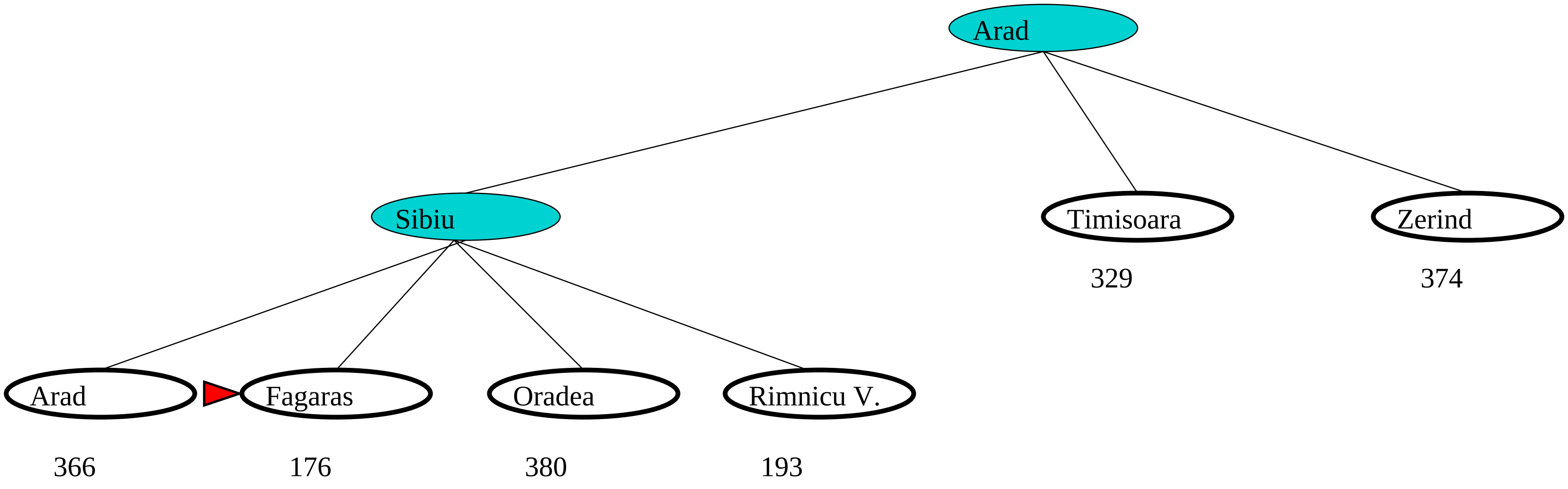
- Evaluation function $h(n)$ (*h*euristic) = estimate of cost from n to the closest goal
- E.g., $h_{\mathrm{SLD}}(n)$ = straight-line distance from n to Bucharest
- Greedy search expands the node that *appears* to be closest to goal

Greedy search example

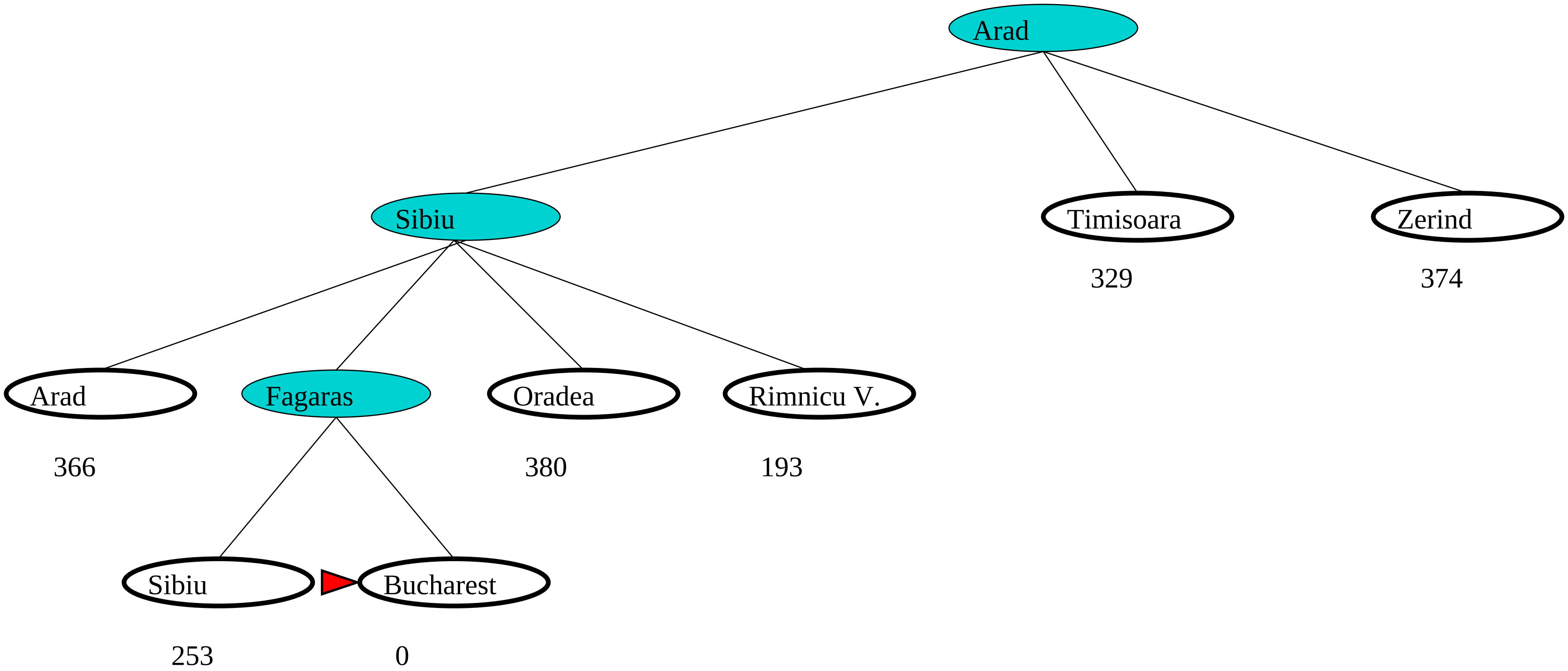
After expanding Arad



After expanding Sibiu



After expanding Fagaras



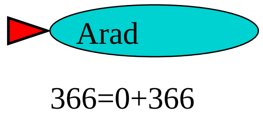
Properties of greedy search

- Complete No — can get stuck in loops, e.g.,
  Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$
  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

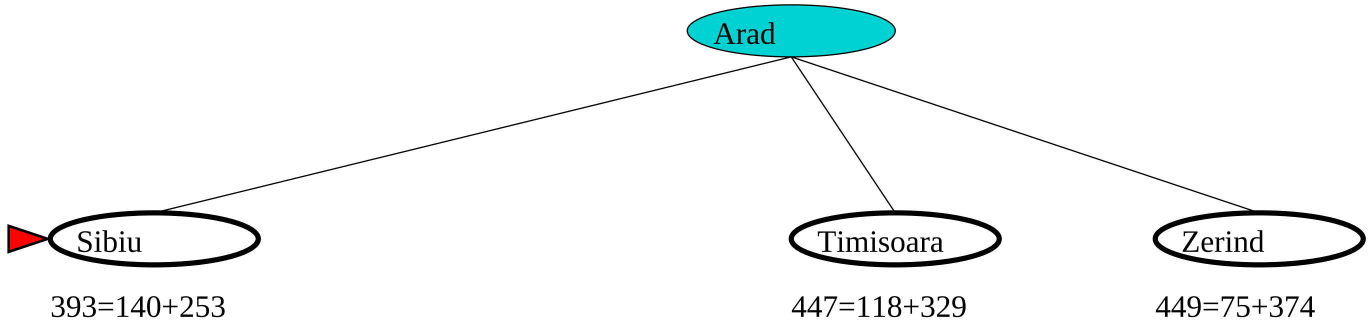
A* search

- 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 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.

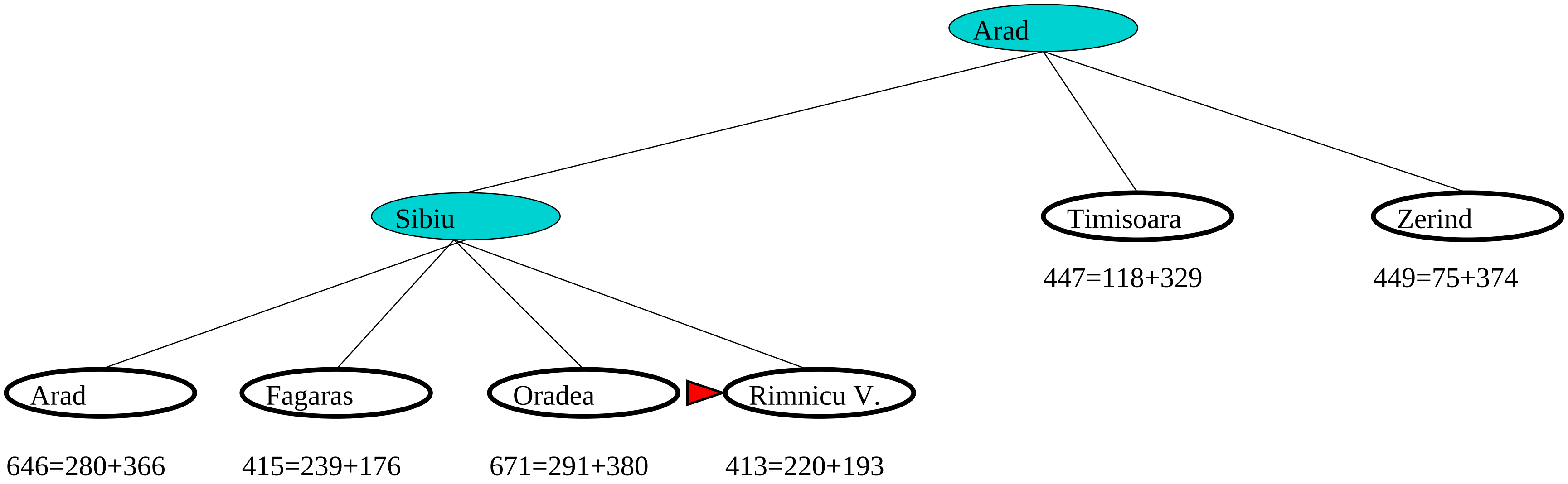
A* search example



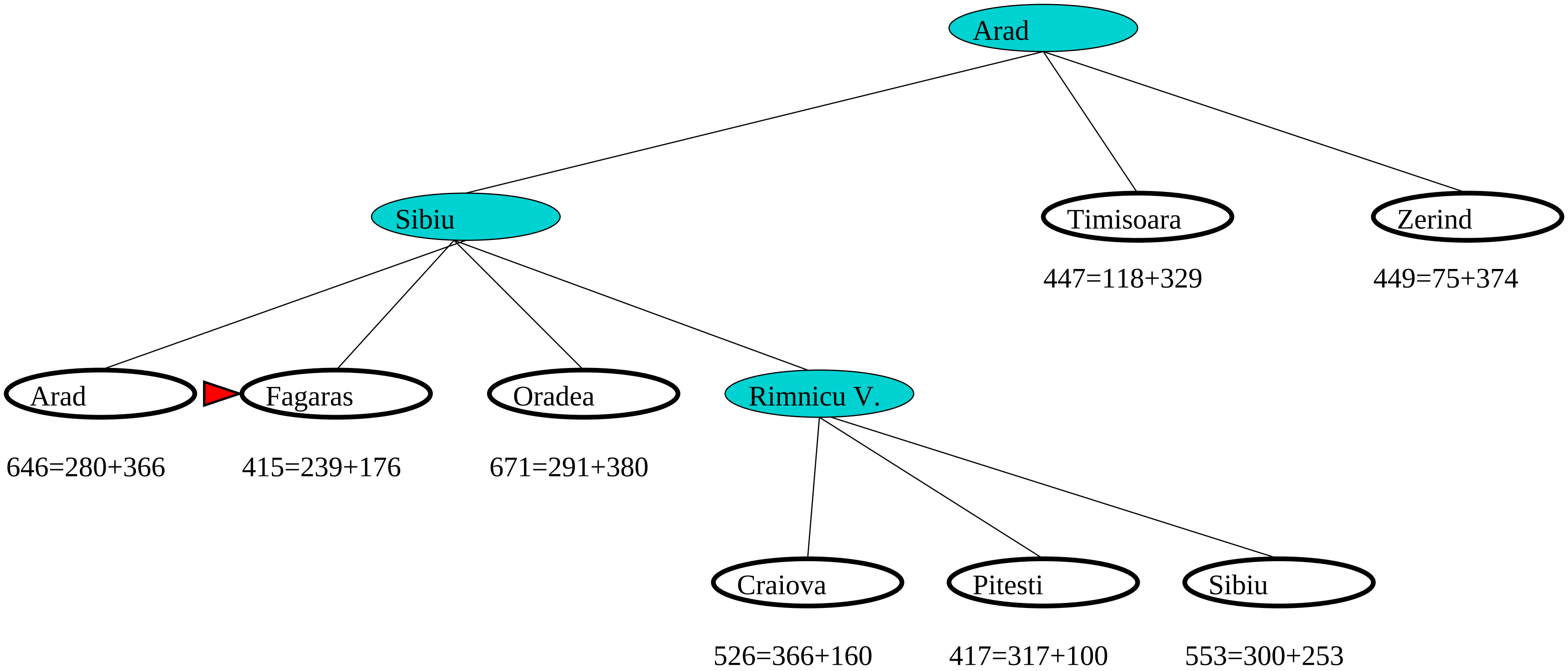
After expanding Arad



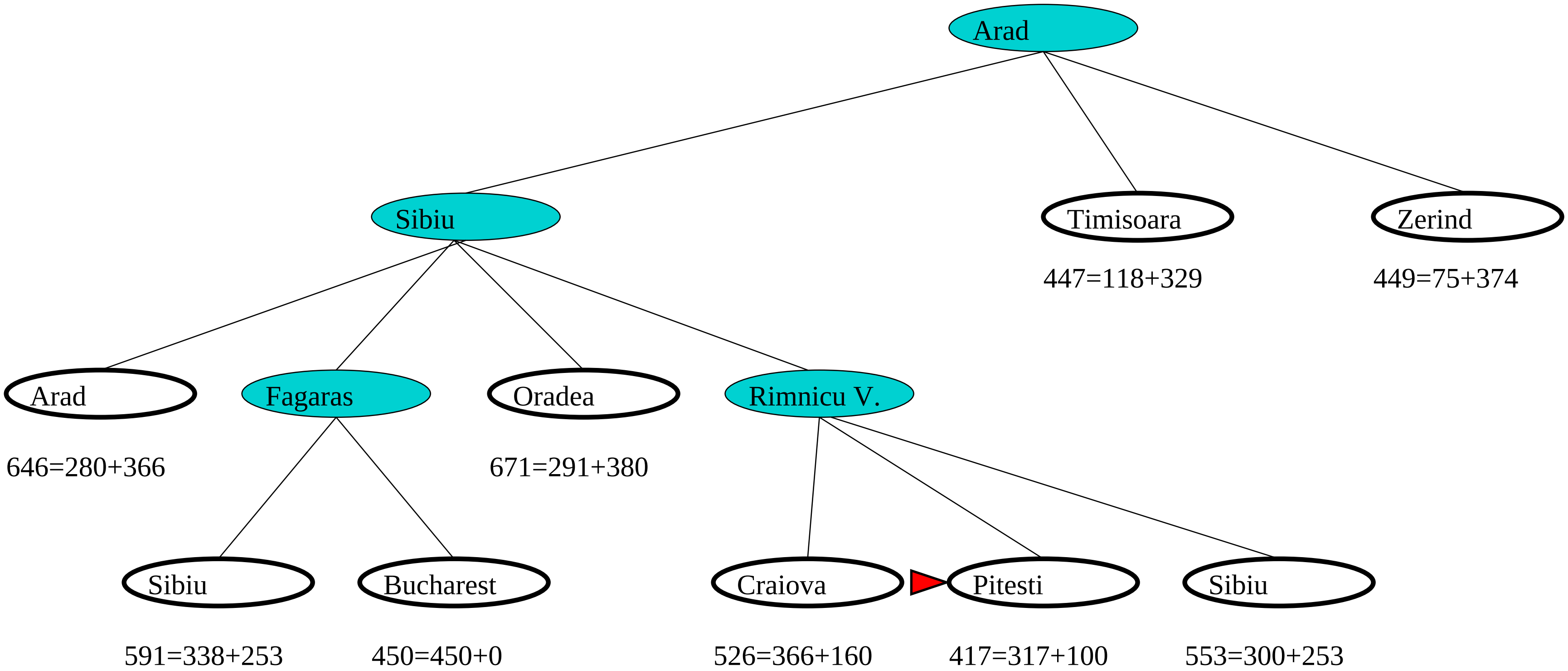
After expanding Sibiu



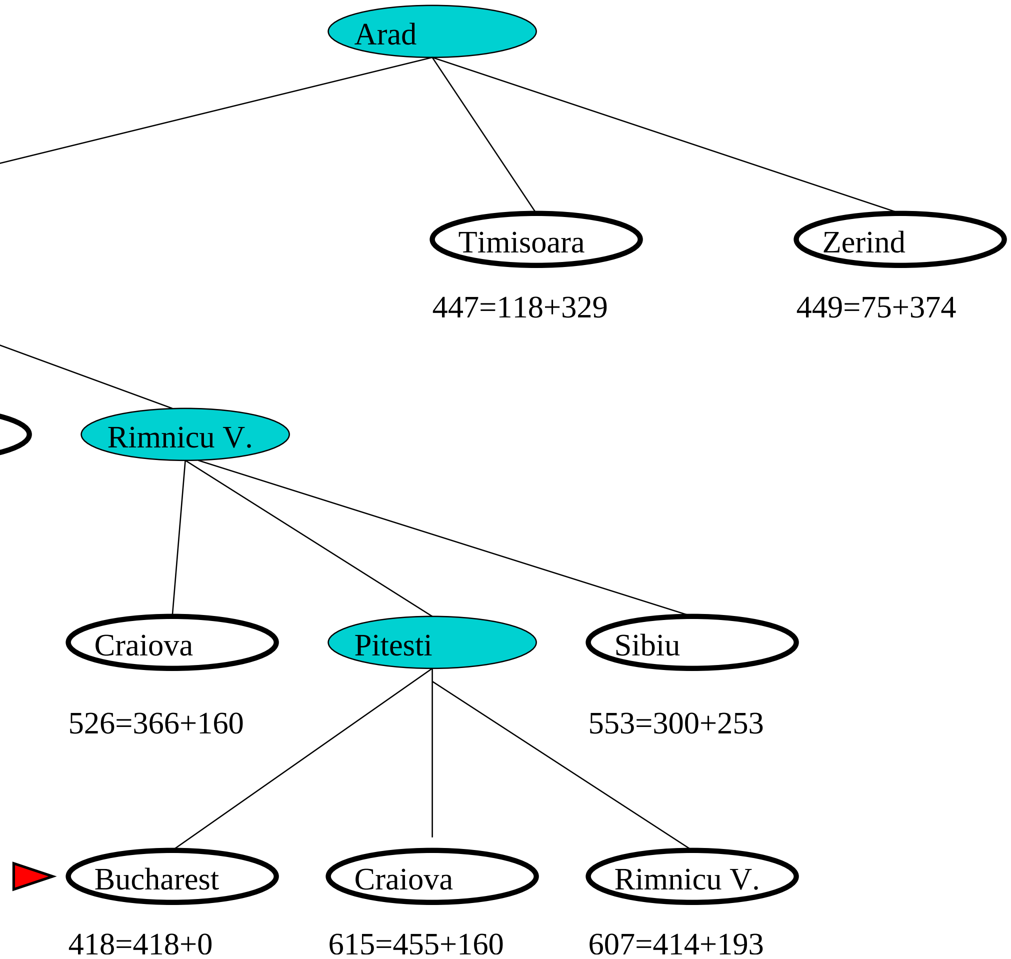
After expanding Rimnicu Vilcea



After expanding Fagaras



After expanding Pitesti

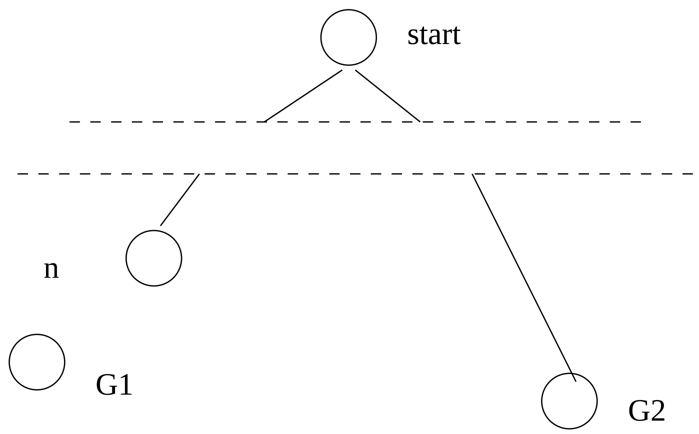


Optimality of A* (standard proof)

Theorem: A* search is optimal

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.

Optimality of A* (standard proof)



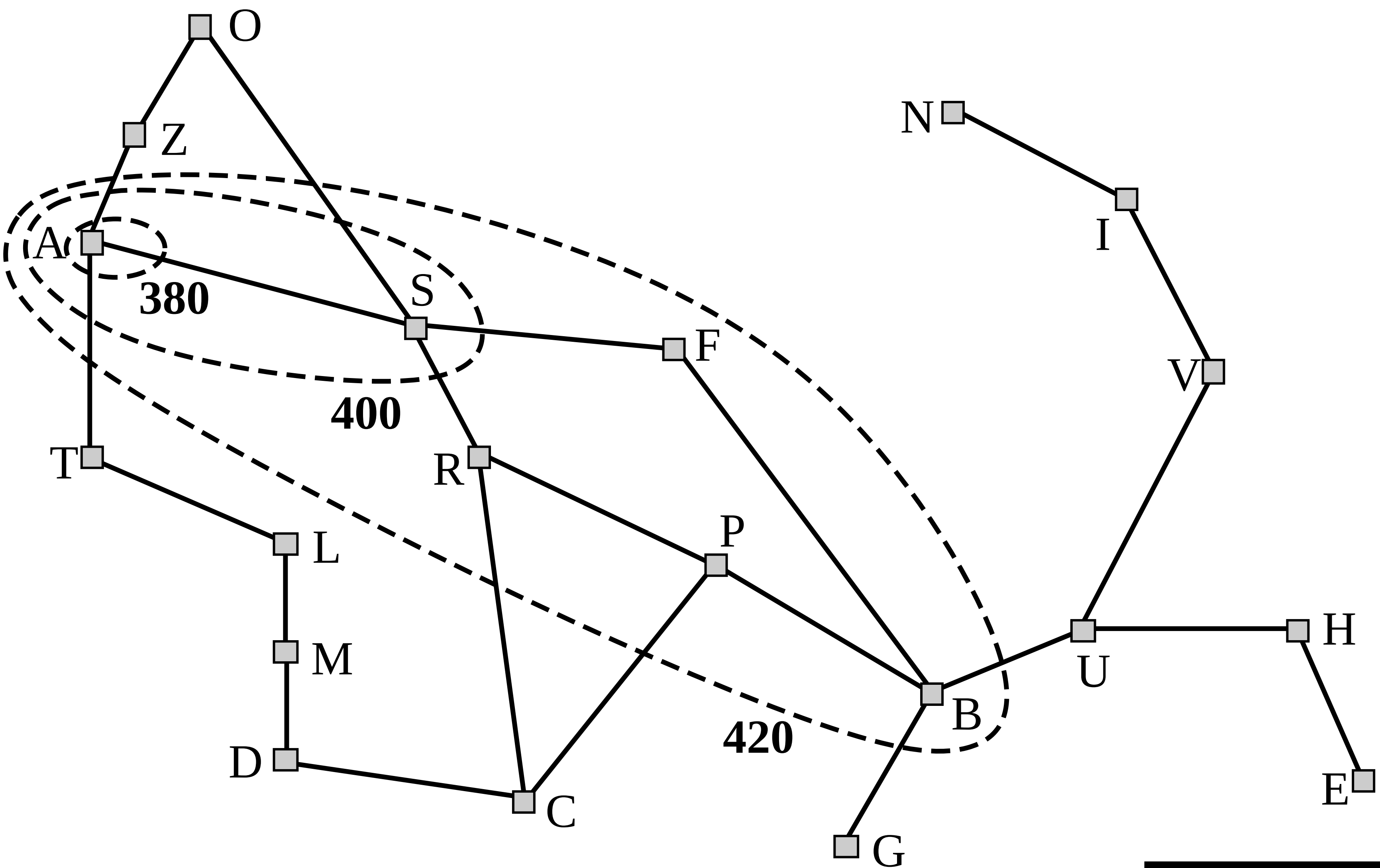
$$\begin{aligned} f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\ &> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\ &\geq f(n) && \text{since } h \text{ is admissible} \end{aligned}$$

Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

Optimality of A* (more intuitive)

- 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 $f_i < f_{i+1}$
- Note: with uniform-cost search (A* search with h(n)=0) the bands are "circular";
with a more accurate heuristic, the bands will stretch toward the goal and become more narrowly focused around the optimal path

F-contours



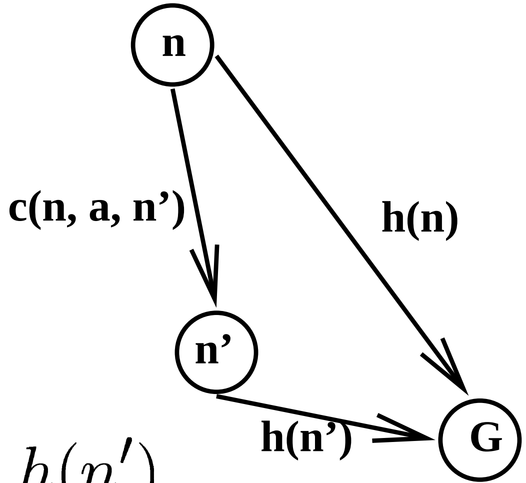
Properties of A*

- Complete Yes, unless there are infinitely many nodes with $f \leq f(G)$
- Time Exponential in (relative error in $h \times$ length of solution)
- 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 $f(n) > C^*$

Proof of lemma: Consistency

A heuristic is *consistent* if

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



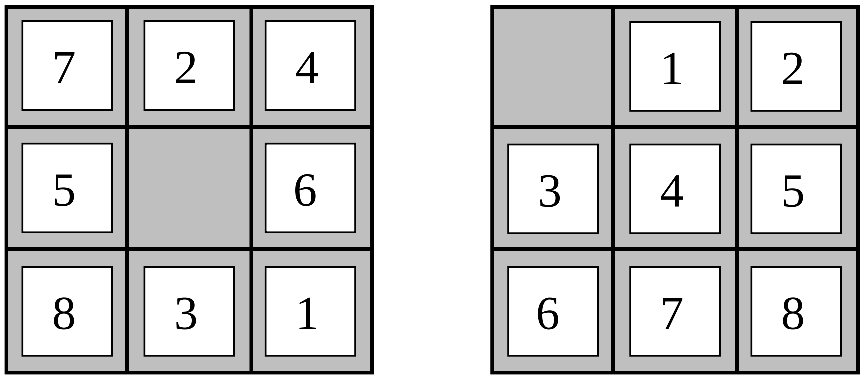
If h is consistent, we have

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

I.e., $f(n)$ is nondecreasing along any path.

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)



Start State Goal State

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

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:

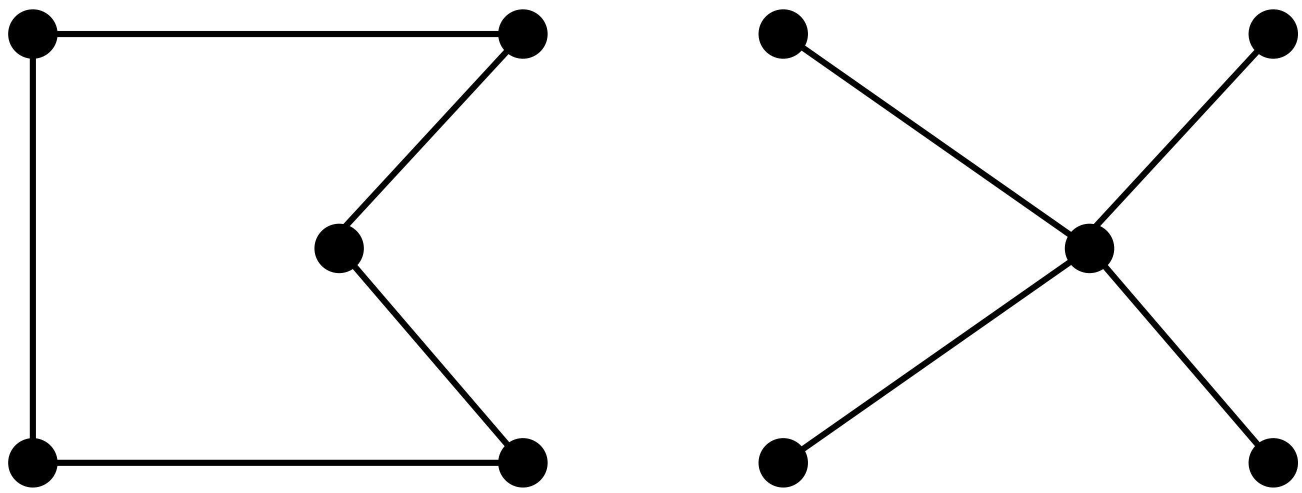
$d = 14$	IDS = 3,473,941 nodes
	A$^*(h_1)$ = 539 nodes
	A$^*(h_2)$ = 113 nodes
$d = 24$	IDS $\approx$ 54,000,000,000 nodes
	A$^*(h_1)$ = 39,135 nodes
	A$^*(h_2)$ = 1,641 nodes

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

Relaxed problems (cont'd)

Well-known example: *travelling salesperson problem (TSP)*
Find the shortest tour visiting all cities exactly once



Minimum spanning tree can be computed in $O(n^2)$ and is a lower bound on the shortest (open) tour

Iterative Deepening A* (IDA*)

- Idea: perform iterations of DFS. The cutoff is defined based on the f-cost rather than the depth of a node.
- Each iteration expands all nodes inside the contour for the current f-cost, peeping over the contour to find out where the contour lies.

Iterative Deepening A* (IDA*)

function IDA* (*problem*)
returns a solution sequence

inputs: *problem*, a problem
local variables:
f-limit, the current f-COST limit
root, a node

root $\leftarrow$ MAKE-NODE(INITIAL-STATE[*problem*])
f-limit $\leftarrow$ f-COST(*root*)
loop do
solution, f-limit $\leftarrow$ DFS-CONTOUR(*root, f-limit*)
if *solution* is non-null **then return** *solution*
if *f-limit* = ∞ **then return** failure

Iterative Deepening A* (IDA*)

```
function DFS-Contour (node, f-limit)
returns a solution sequence and a new f-Cost limit

   inputs:      node, a node
      f-limit, the current f-Cost limit
   local variables:
      next-f, the f-Cost limit for the next contour, initally ∞

   if f-Cost[node] > f-limit then return null, f-Cost[node]
   if Goal-Test[problem](State[node]) then return node, f-limit
   for each node s in Successors(node) do
      solution, new-f ← DFS-Contour(s, f-limit)
      if solution is non-null then return solution, f-limit
      next-f ← Min(next-f, new-f)
   return null, next-f
```

Properties of IDA*

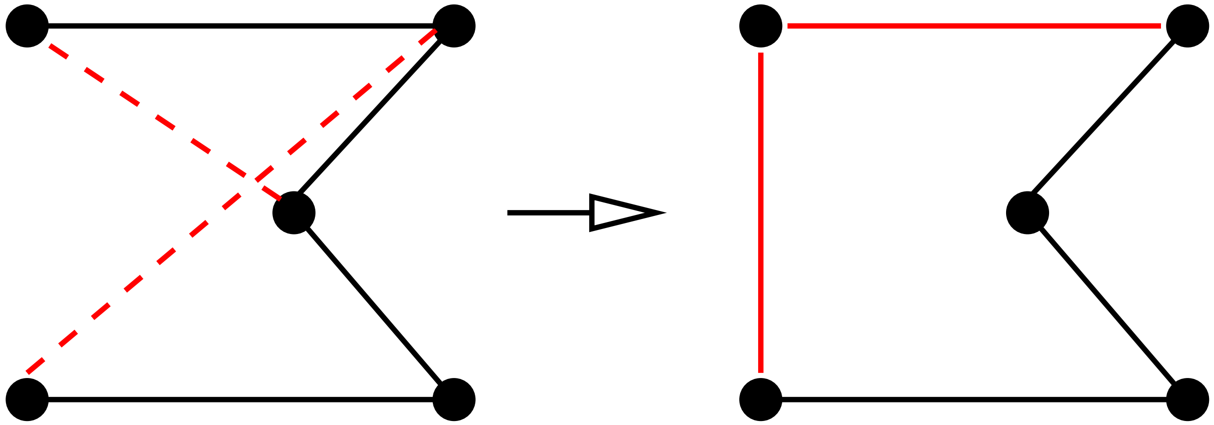
- Complete Yes, similar to A*.
- Time Depends strongly on the number of different values that the heuristic value can take on.
  8-puzzle: few values, good performance
  TSP: the heuristic value is different for every state. Each contour only includes one more state than the previous contour. If A* expands N nodes, IDA* expands 1 + 2 + $\ldots$ + N = $O(N^2)$ nodes.
- Space It is DFS, it only requires space proportional to the longest path it explores. If δ is the smallest operator cost, and f^* is the optimal solution cost, then IDA* will require bf^*/δ nodes.
- Optimal Yes, similar to A*

Iterative improvement algorithms

- In many optimization problems, the path is irrelevant; the goal state itself is the solution
- Then state space = set of “complete” configurations; find optimal configuration, e.g., TSP or, find configuration satisfying constraints, e.g., timetable
- In such cases, can use *iterative improvement* algorithms; keep a single “current” state, try to improve it
- Constant space, suitable for online as well as offline search

Example: Travelling Salesperson Problem

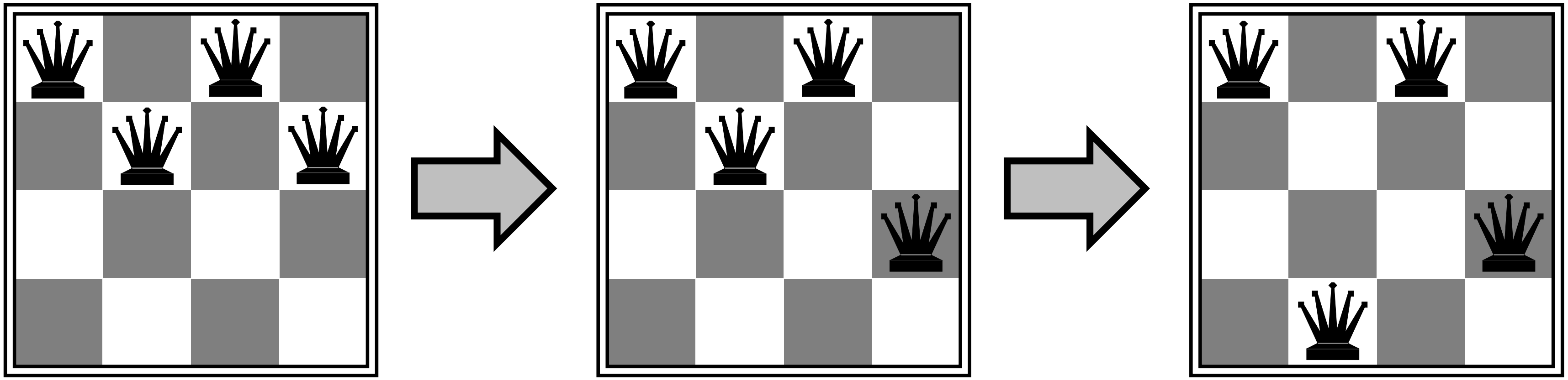
Start with any complete tour, perform pairwise exchanges



Example: n-queens

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

Move a queen to reduce number of conflicts



Hill-climbing (or gradient ascent/descent)

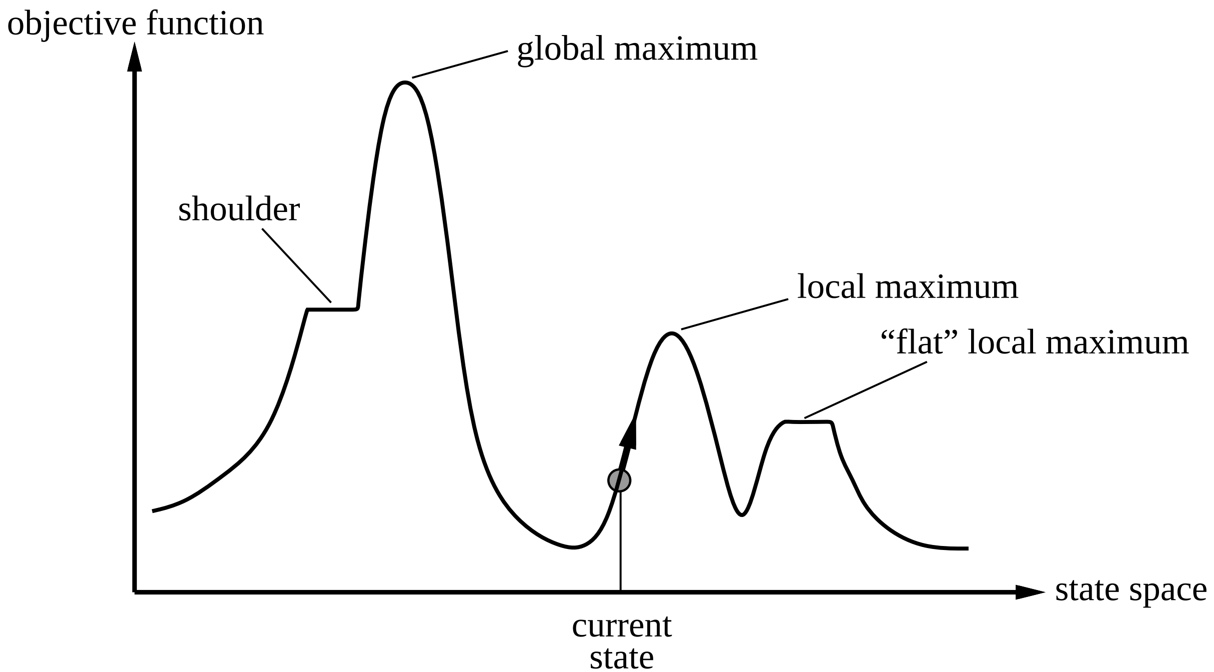
function HILL-CLIMBING (*problem*)
returns a state that is a local maximum

  inputs: *problem*, a problem
  local variables:
    current, a node
    neighbor, a node

  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  loop do
    neighbor ← a highest-valued successor of *current*
    if VALUE[*neighbor*] ≤ VALUE[*current*] **then return** STATE[*current*]
    current ← *neighbor*

Hill-climbing (cont'd)

- "Like climbing Everest in thick fog with amnesia"
- Problem: depending on initial state, can get stuck on local maxima



- In continuous spaces, problems w/ choosing step size, slow convergence

Simulated annealing

function SIMULATED-ANNEALING (*problem, schedule*)
returns a solution state

inputs: *problem*, a problem
schedule, a mapping from time to "temperature"
local variables: *current*, a node
next, a node
T, a "temperature" controlling the probability of downward steps

current $\leftarrow$ MAKE-NODE(INITIAL-STATE[*problem*])
for $t \leftarrow 1$ **to** ∞ **do**
T $\leftarrow$ *schedule*[*t*]
if $T = 0$ **then return** *current*
next $\leftarrow$ a randomly selected successor of *current*
$\Delta E \leftarrow$ VALUE[*next*] - VALUE[*current*]
if $\Delta E > 0$ **then** *current* $\leftarrow$ *next*
else *current* $\leftarrow$ *next* only with probability $e^{\Delta E/T}$

Properties of simulated annealing

- Idea: escape local maxima by allowing some “bad” moves but gradually decrease their size and frequency
- At fixed “temperature” T, state occupation probability reaches Boltzman distribution $p(x) = \alpha e^{\frac{E(x)}{kT}}$
- T decreased slowly enough $\Longrightarrow$ always reach best state
- Is this necessarily an interesting guarantee??
- Devised by Metropolis et al., 1953, for physical process modelling
- Widely used in VLSI layout, airline scheduling, etc.