Artificial Intelligence

Informed search *AIMA* Ch. 3.5 and 3.6

Lecture 4

Outline

- ◇ Greedy search
- ◇ A* search
- ◇ Admissible heuristic
- ◇ Optimality of A* search
- ◇ Consistent heuristic
- ◇ Relaxed problems

Revision: Tree search

function TREE-SEARCH(*problem, frontier*) **returns** a solution, or failure
  INSERT(ROOT-NODE(*problem*.INITIAL-STATE), *frontier*)
  while not EMPTY?(*frontier*) **do**
    node ← REMOVE(*frontier*)
    if *problem*.GOAL-TEST applied to *node*.STATE succeeds **return** *node*
    for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**
      INSERT(CHILD-NODE(*problem*, *node*, *action*), *frontier*)
  return failure

A strategy is defined by choosing the order of node expansion

Best-first search

What if we have problem-specific knowledge beyond the problem definition?

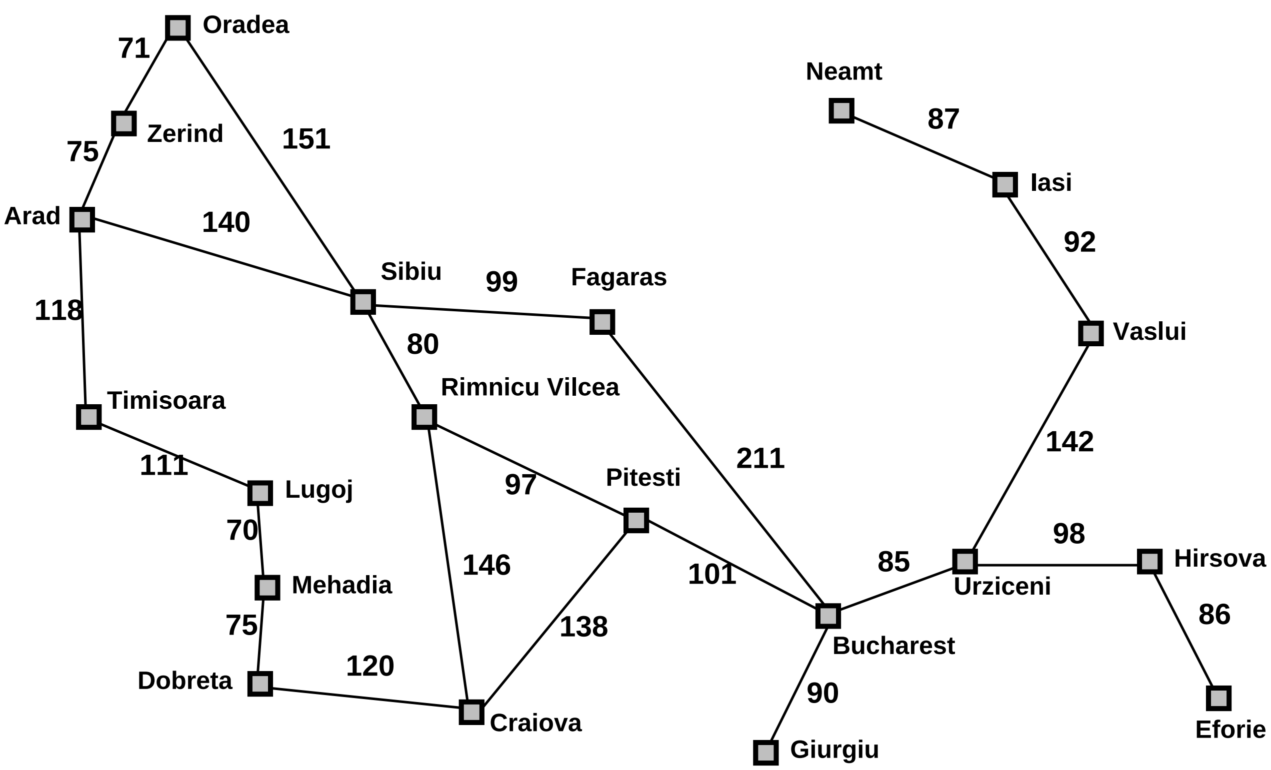
Idea: use an evaluation function $f(n)$ for each node
– estimate of "desirability"

$\Rightarrow$ Expand most desirable unexpanded node

Implementation: *frontier* is a queue sorted in decreasing order of desirability

Special cases:
– greedy search
– A* search
$\Rightarrow$ Obtained by different desirability notions

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

Evaluation function $h(n)$ (**h**euristic)

$=$ estimated cost of getting from node n to the closest goal

E.g., $h_{\text{SLD}}(n)$ = straight-line distance from n to Bucharest

Typically, $h(n)$ depends only on the state of n, and not on its parents

– e.g., the straight-line distance from n to Bucharest does not depend on how we got to n

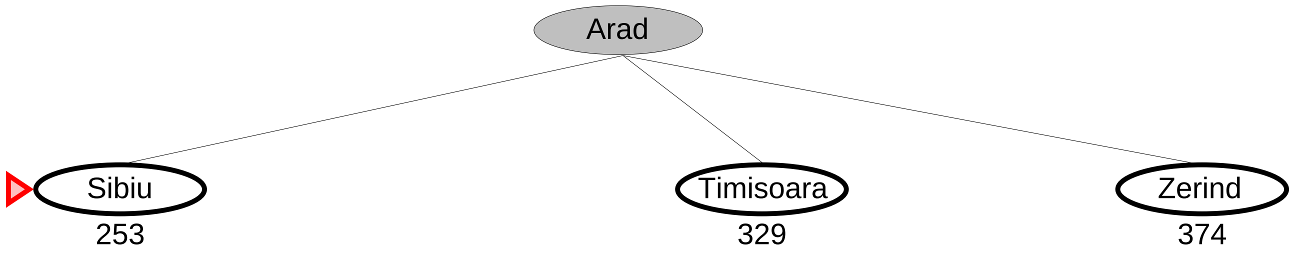
Greedy search expands the node that **appears** to be closest to goal

◇ Resembles DFS in the way it prefers to follow a single path.

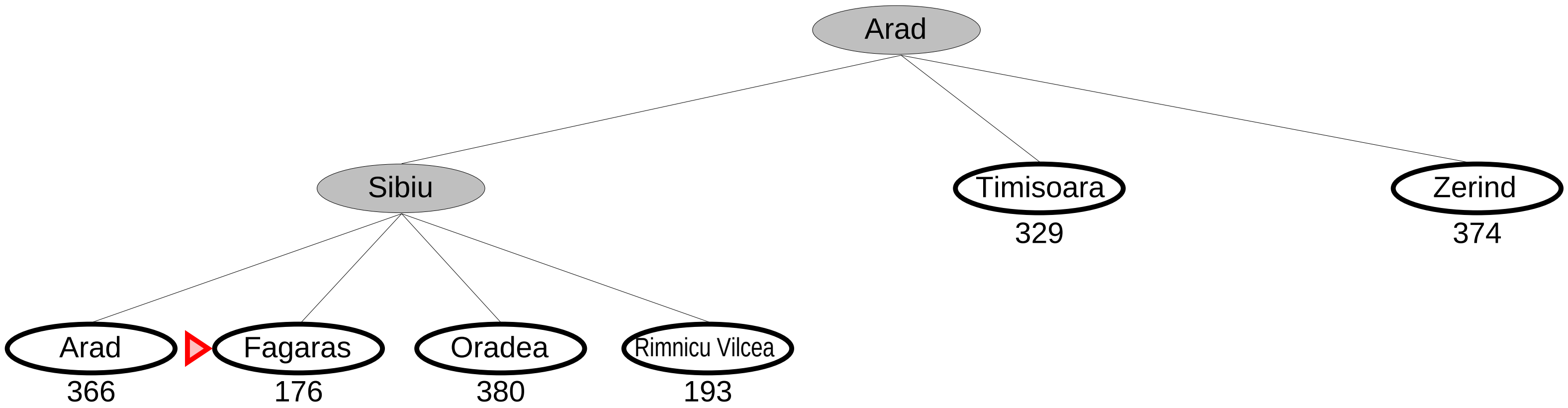
◇ Can go down an infinite path if we don't detect repeated states.

Greedy search example

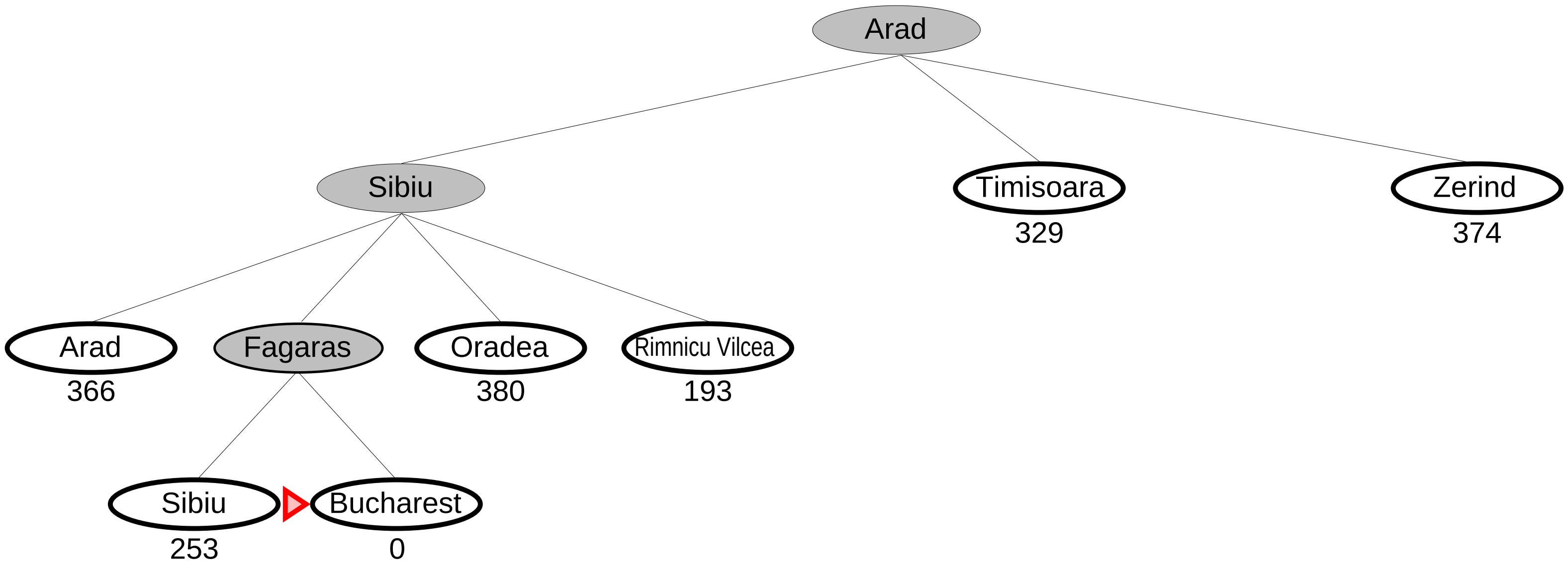
Greedy search example



Greedy search example



Greedy search example



Properties of greedy search

Complete?? No: can get stuck in loops
– e.g., with Oradea as goal:
Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow \ldots$
– Complete in finite state spaces 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

A* evaluation function: $f(n) = g(n) + h(n)$

$g(n)$ = cost of the path from the start node to n
  – depends only on path cost, not on the heuristic
$h(n)$ = estimated cost of getting from node n to the closest goal
$f(n)$ = estimated total cost of a path through n to a goal

Admissible heuristic

Let $h^*(n)$ be the **true** cost of getting from n to the closest goal

Heuristic $h(n)$ is admissible if $0 \leq h(n) \leq h^*(n)$ for each node n
(Note: these two conditions imply $h(n_G) = 0$ for each goal node n_G)

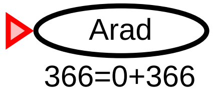
$\Rightarrow$ I.e., $h(n)$ is admissible if it never overestimates the cost of getting from n to the closest goal

E.g., $h_{\mathrm{SLD}}(n)$ never overestimates the actual road distance

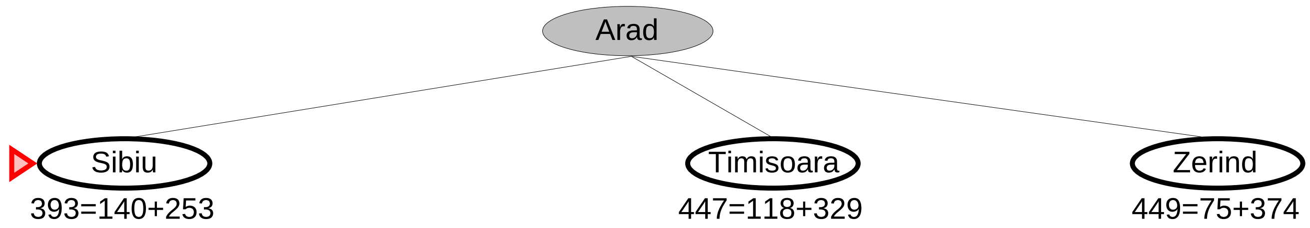
Theorem: A* tree search with an admissible heuristic is optimal

$\Rightarrow$ A* tree search with admissible heuristic is used throughout AI!

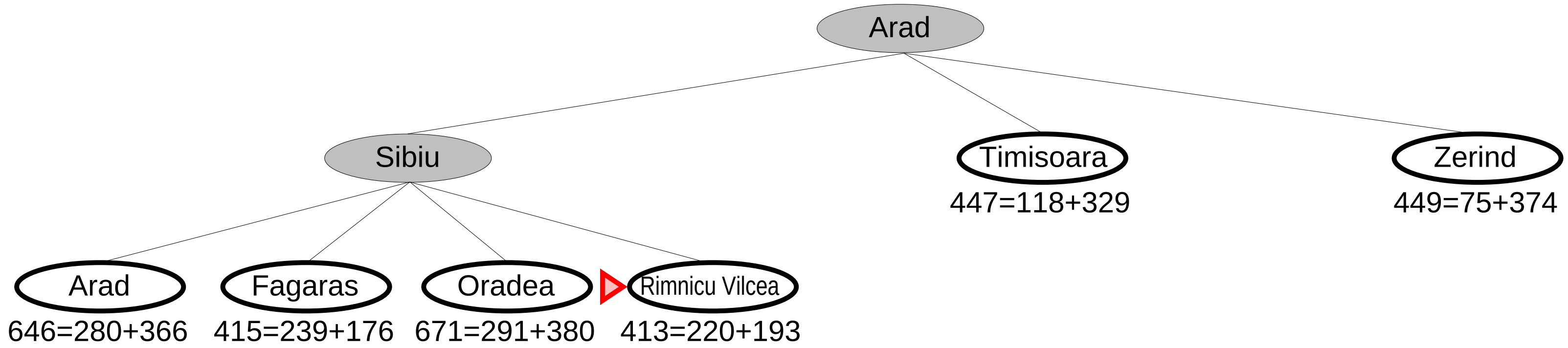
A* tree search example



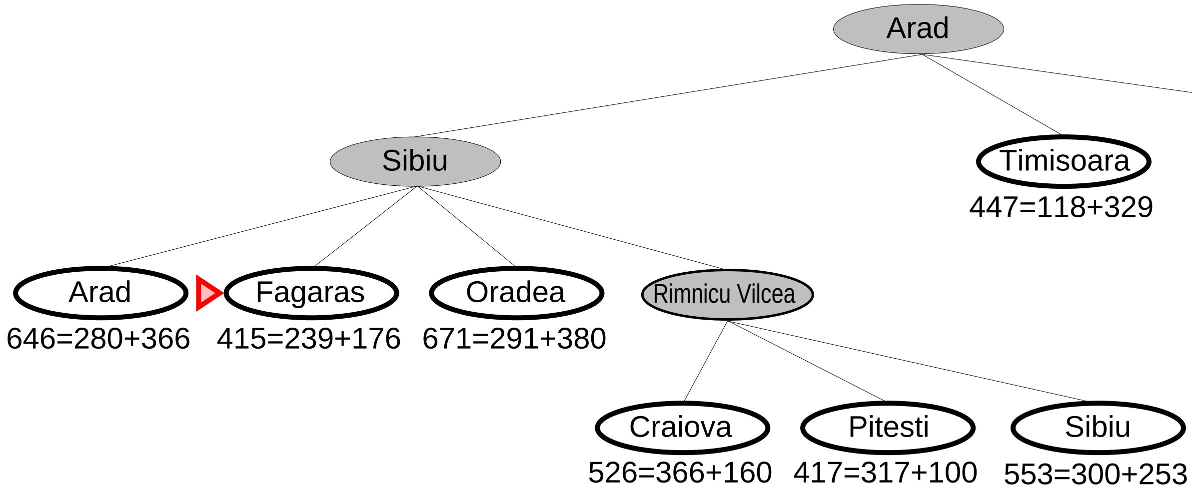
A* tree search example



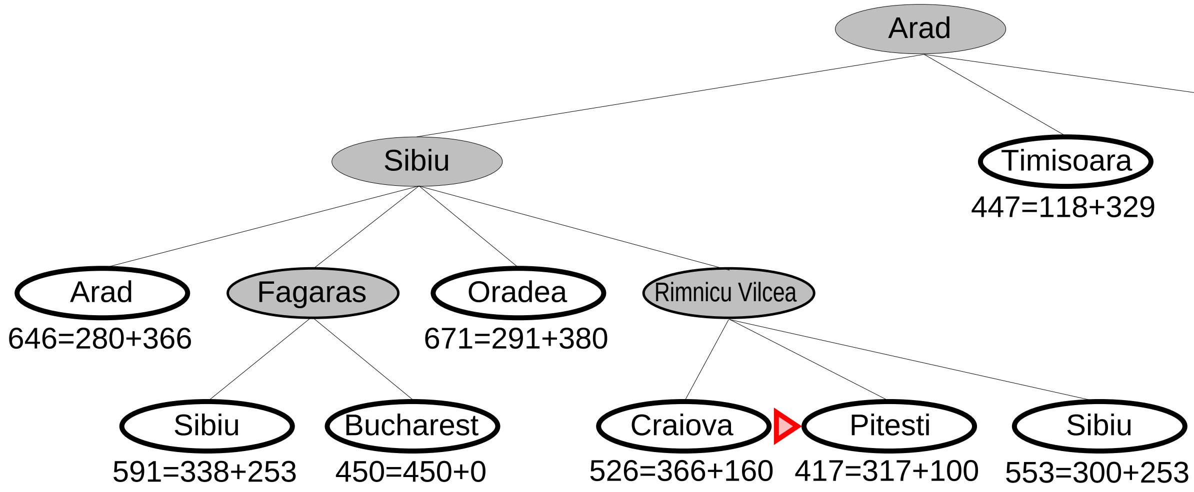
A* tree search example



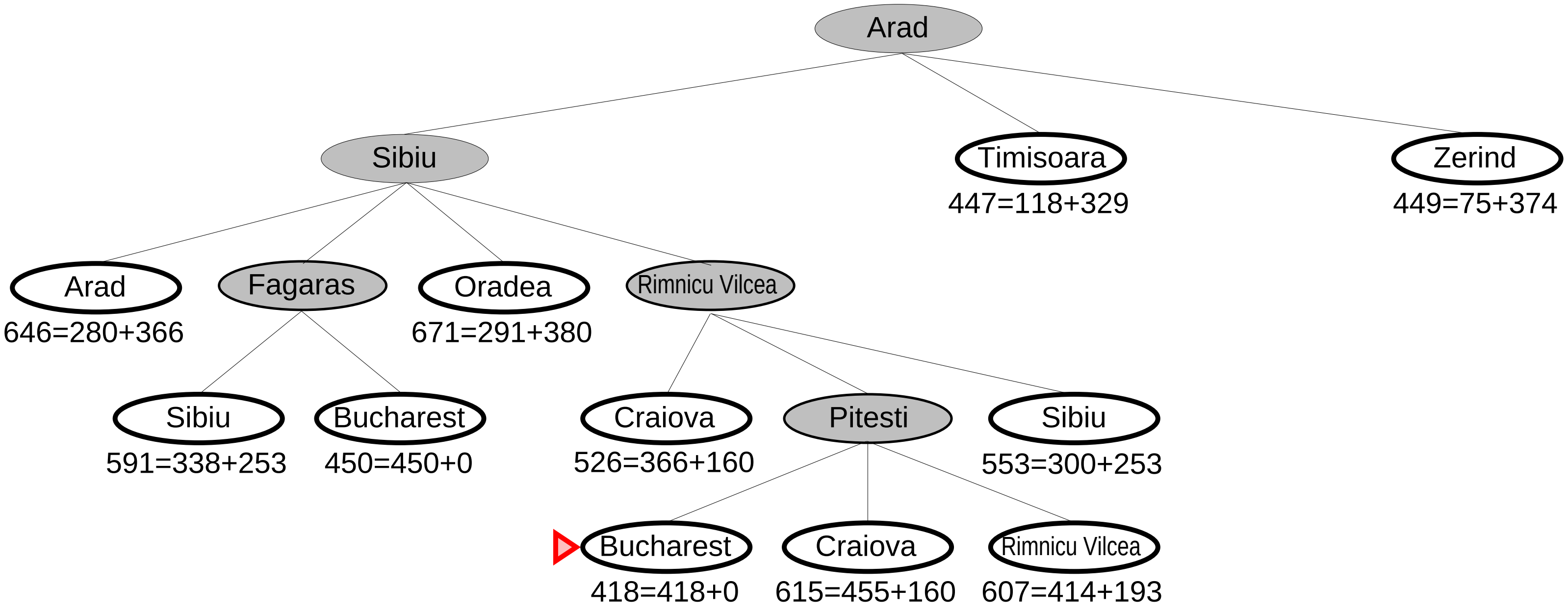
A^* tree search example



A* tree search example



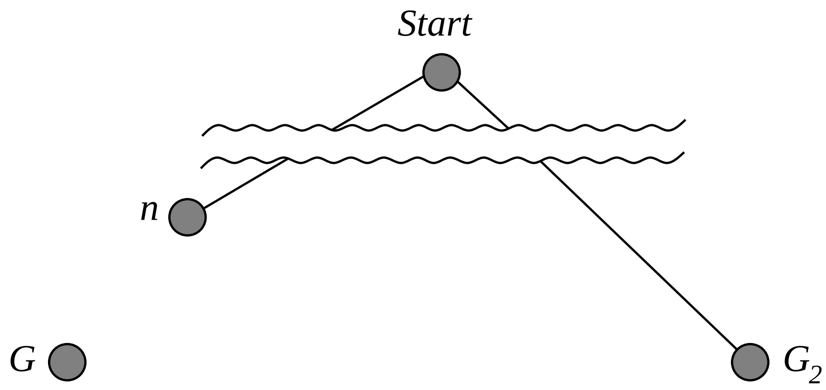
A* tree search example



Optimality of A* tree search

Theorem: A* tree search with an admissible heuristic is optimal

Proof. Assume that a suboptimal goal node G_2 is in the queue, and consider an arbitrary unexpanded node n on a shortest path to an optimal goal G.



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

Since $f(G_2) > f(n)$, node G_2 will not be selected for expansion before n.

Properties of A* tree search

Complete?? Yes, unless there are infinitely many nodes with $f(n) \leq f(G)$

Time?? Exponential in [relative error in h $\times$ length of solution.]

Space?? Keeps all nodes in memory

Optimal?? Yes

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

◊ Main difficulty with A*: memory requirements

Optimality of A* graph search

Reminder: graph search does not visit the same state twice

A* graph search is not optimal for an arbitrary admissible heuristic: if a suboptimal path to a node n is discovered first, the optimal path discovered later will not be considered

Solution 1: discard the more expensive path to n

Extra bookeeping is messy, but ensures optimality.

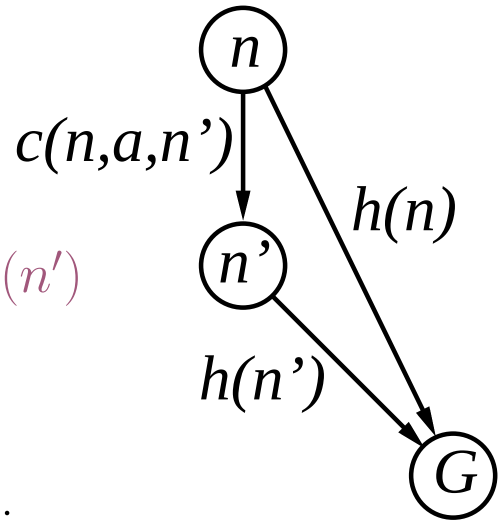
Solution 2: ensure that optimal paths are explored first, e.g., by using a **consistent** heuristic

Enforces a form of triangle inequality (stipulating that each side of a triangle cannot be longer than the sum of the other two).

Consistent heuristic

A heuristic is consistent if

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



If h is consistent, then

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

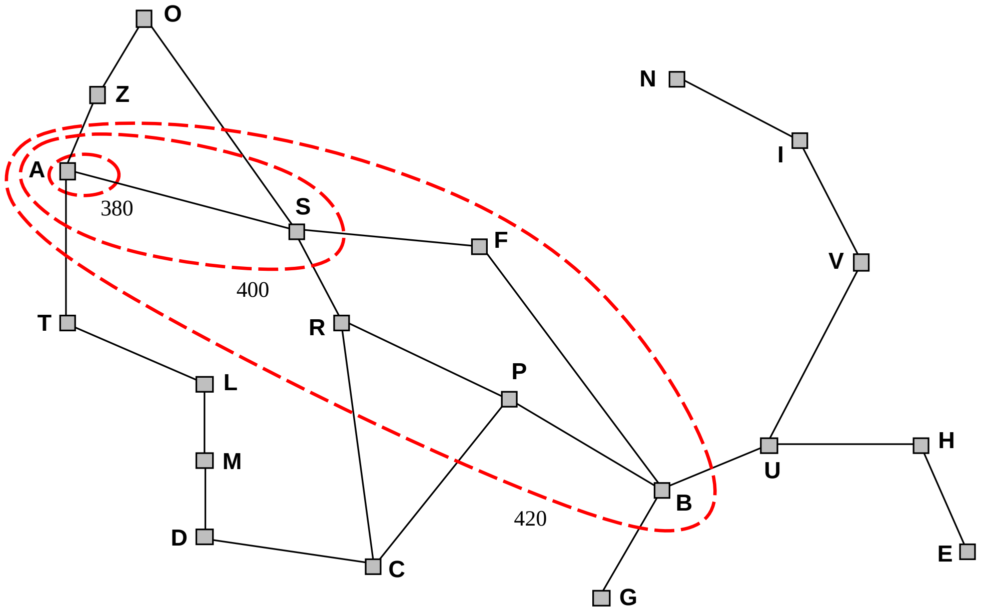
$\Rightarrow$ I.e., along each path, $f(n)$ is **nondecreasing**.

- ◇ Consistent $\Rightarrow$ Admissible (but not the other way around)
- ◇ Graph search with a consistent heuristic is optimal

Optimality of A* graph search

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 $f_i < f_{i+1}$



Dealing with memory requirements: IDA*

IDA* combines iterative deepening with A*:
– instead of depth, cutoff is the f-cost—that is, the value of $g(n) + h(n)$

function IDA*(*problem*) **returns** a goal node, or failure
  root ← Root-Node(*problem*.Initial-State)
  f_limit ← h(*root*)
  loop forever
    goal, *f_limit* ← IDA-DFS(*problem*, *root*, *f_limit*)
    if *goal* ≠ *failure* **return** *goal*
    if *f_limit* = ∞ **return** *failure*

Dealing with memory requirements: IDA*

function IDA-DFS(*problem*, *node*, *f_limit*) **returns** a goal node, or next cost
  f_node ← g(*node*) + h(*node*)
  if *f_node* > *f_limit* **return** (*failure*, *f_node*)
  if *problem*.GOAL-TEST succeeds on *node*.STATE **return** (*node*, *f_limit*)
  next_f_limit ← ∞
  for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**
    successor ← CHILD-NODE(*problem*, *node*, *action*)
    goal, *new_f_limit* ← IDA-DFS(*problem*, *successor*, *f_limit*)
    if *goal* ≠ *failure* **return** (*goal*, *new_f_limit*)
    next_f_limit ← min(*next_f_limit*, *new_f_limit*)
  return (*failure*, *next_f_limit*)

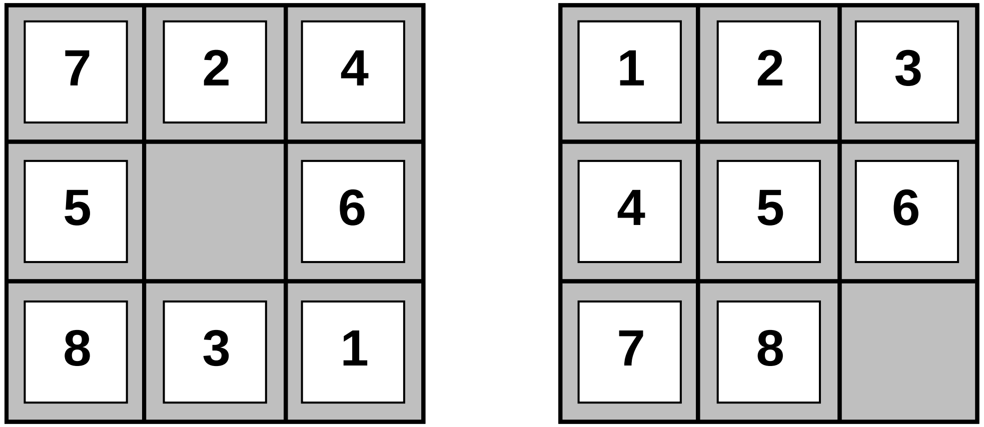
Avoids the overhead of keeping a sorted queue of nodes
Progress can be slow if f-cost increases slowly

Alternatives: recursive best-first search; (simplified) memory-bounded A*

Deriving heuristic functions

For example, for the 8-puzzle:

$h_1(n)$ = the number of misplaced tiles
$h_2(n)$ = the total Manhattan distance
– i.e., the sum over all tiles of the number of moves needed to bring the tile into required position



Start State **Goal State**

$h_1(S) =$?? 6
$h_2(S) =$?? $4+0+3+3+1+0+2+1 = 14$

Dominance

If $h_2(n) \geq h_1(n)$ for all n and both heuristics are admissible,
then h_2 dominates h_1 and is better for search
– h_2 is then closer to the actual cost $h^*(n)$

Typical search costs (for 8-puzzle):

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

Combining heuristics: given two admissible heuristics h_a and h_b,

$$h(n) = \max(h_a(n), h_b(n))$$

is also admissible, and it dominates both h_a and h_b

Relaxed problems

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

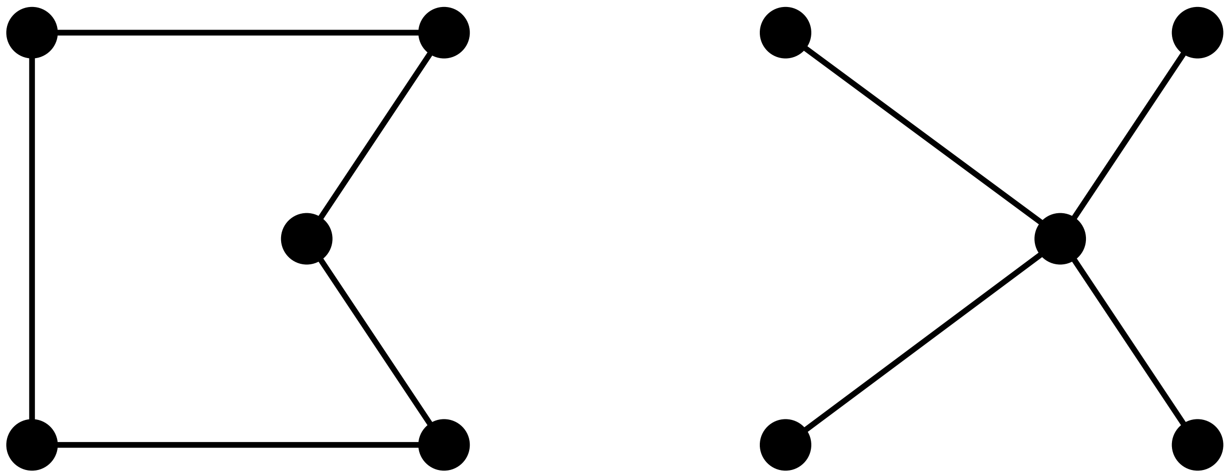
If the rules of the 8-puzzle are relaxed so that a tile can move **anywhere**, then $h_1(n)$ is the cost of the shortest solution

If the rules are relaxed so that a tile can move to **any adjacent square**, then $h_2(n)$ is the cost of 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: Example

Well-known example: traveling salesman 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

Subproblems and pattern databases

Exact cost of a solution to a subproblem gives an admissible heuristic
– e.g., in an 8-puzzle, $h(n)$ might be the **exact cost** of moving tiles 1–4 into correct position (we disregard other tiles)

Subproblems: special kinds of relaxed problems

Exact cost of subproblem solutions is often stored in a pattern database
– we precompute and store the solutions to all possible subproblems
  – can be done by searching backwards from the goal
– $h(n)$ can be obtained via simple lookup

We can add h-values for **disjoint subproblems**

Disjoint subproblems

Subproblems of 8-puzzle:
- h_1: move tiles 1–4 into appropriate position
- h_2: move tiles 5–8 into appropriate position

We want to define h_1 and h_2 so that $h(n) = h_1(n) + h_2(n)$ is admissible

Not disjoint if the cost of a subproblem is the number of total steps
- moving tiles 1–4 into position involves moving tiles 5–8 as well
- $h(n)$ is not admissible as it may count some moves twice

Disjoint if in each subproblem we count only the moves of the target tiles
- in $h_1(n)$ we count only the number of moves of tiles 1–4
- in $h_2(n)$ we count only the number of moves of tiles 5–8

To reach a goal from n we clearly need at least $h_1(n) + h_2(n)$ steps
$\Rightarrow h(n)$ is admissible

Summary

Heuristic functions estimate costs of shortest paths

Good heuristic can dramatically reduce search cost

Greedy best-first search expands n with lowest $h(n)$
– incomplete and not always optimal

A* tree search expands nodes n with lowest $g(n) + h(n)$
– complete and optimal if heuristic is admissible
– also optimally efficient (up to tie-breaks, for forward search)

A* graph search is optimal if heuristic is consistent

Admissible heuristic can be derived as solutions to relaxed problems

Subproblems can be seen as relaxed problems; can precompute solutions in a database