Last update: February 2, 2010

Informed search algorithms

CMSC 421: Chapter 4, Sections 1–2

Motivation

- In Chapter 3 we were talked about trial-and-error search
- In the worst case, most searches take exponential time (unless P=NP)
- Can sometimes do much better on the average, using *heuristic* techniques

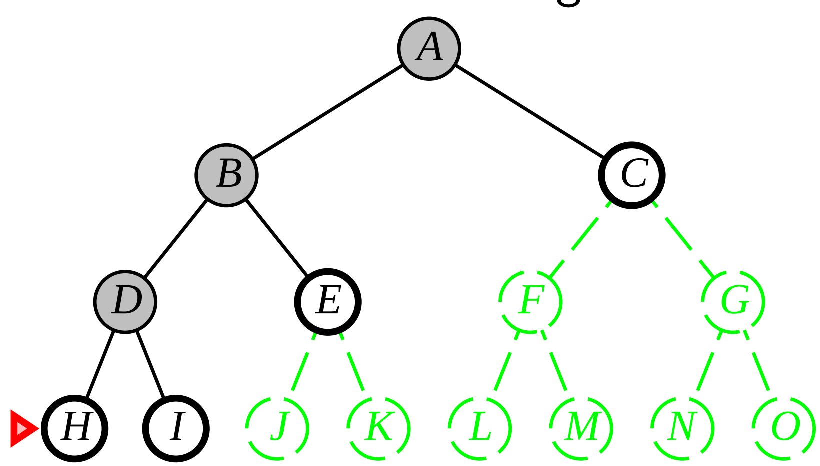
Heuristic:

- Rule of thumb, simplification, or educated guess
- Reduces the search for solutions in domains that are difficult and poorly understood.
- Depending on what heuristic you use, you won't necessarily find an optimal solution, or even a solution at all.

Heuristic tree search

function TREE-SEARCH(*problem*) **returns** a solution, or failure
  fringe ← a list containing MAKE-NODE(INITIAL-STATE[*problem*])
  loop do
    if *fringe* is empty **then return** failure
    node ← REMOVE-FRONT(*fringe*)
    if GOAL-TEST[*problem*] applied to STATE(*node*) succeeds **return** *node*
    fringe ← INSERTALL(EXPAND(*node*, *problem*), *fringe*)

fringe = a list of the nodes that have been generated but not expanded:



Heuristic tree search

```
function TREE-SEARCH( problem) returns a solution, or failure
    fringe ← a list containing MAKE-NODE(INITIAL-STATE[problem])
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem] applied to STATE(node) succeeds return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

Heuristic choice in search algorithms: **what node to expand next**

Use an *evaluation function* $f(n)$ for each node n

– estimate of "desirability"

$\Rightarrow$ Expand most desirable unexpanded node

INSERTALL keeps *fringe* sorted in decreasing order of desirability,

i.e., if $fringe = \langle s_1, s_2, \ldots, s_k \rangle$

then $f(s_1) \leq f(s_2) \leq \ldots \leq f(s_k)$

Thus REMOVE-FRONT always gets a most-desirable node

Heuristic graph search

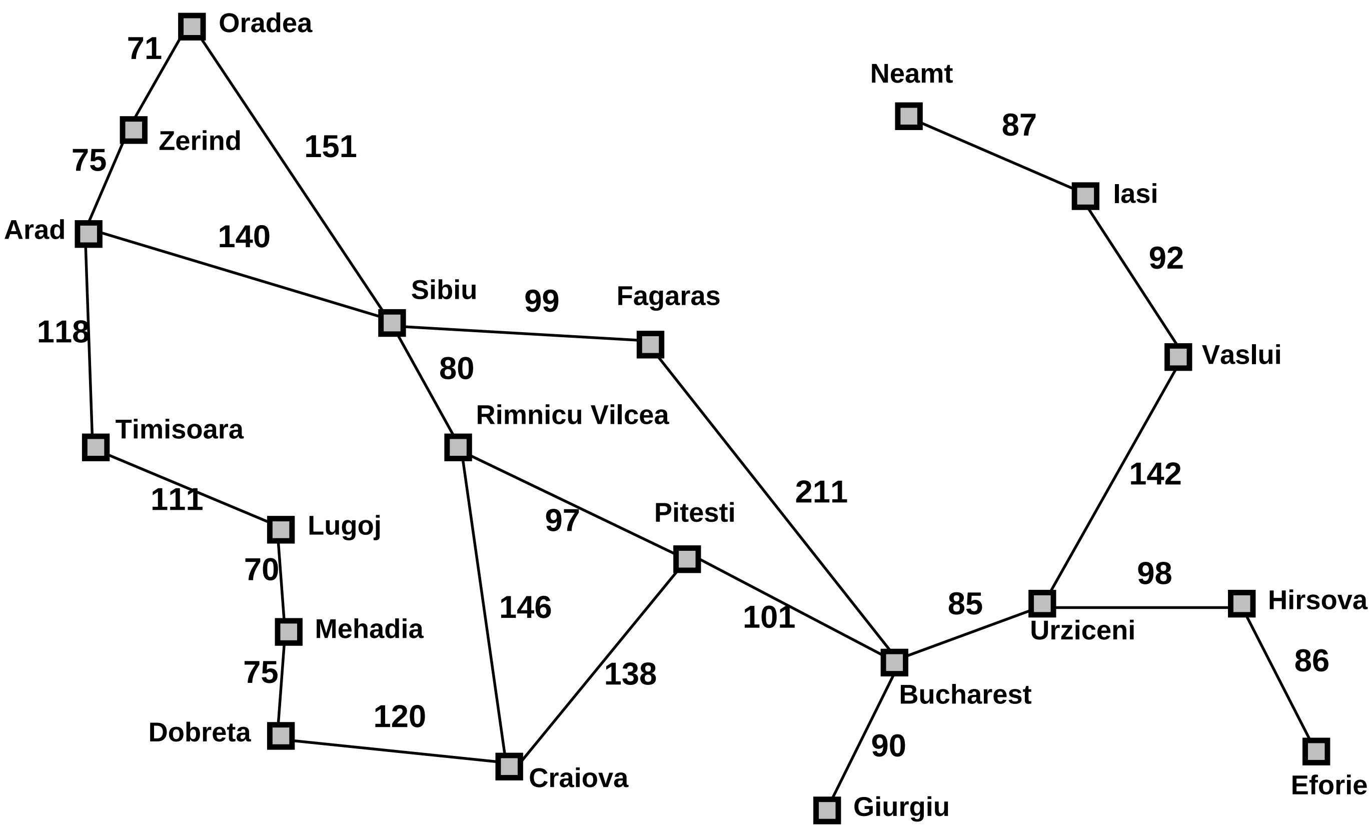
```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
    end
```

Same as for TREE-SEARCH:

Use INSERTALL to keep *fringe* sorted in decreasing order of desirability

Romania with step costs in km

Recall that we want to get from Arad to Bucharest:



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

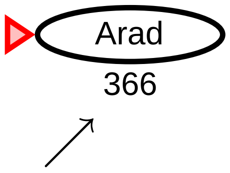
Heuristic function $h(n)$ = estimate of cost from n to the closest goal

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

Greedy search uses $f(n) = h(n)$,
i.e., keeps $fringe$ ordered in increasing value of h

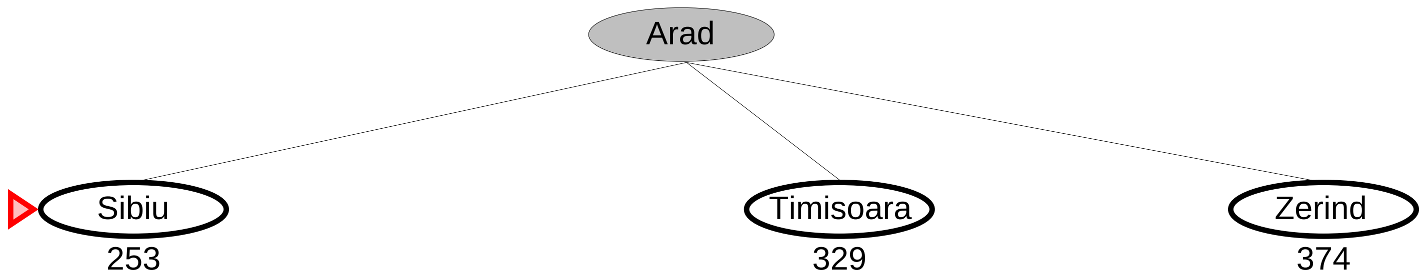
hence always expands whatever node **appears** to be closest to a goal

Greedy search example

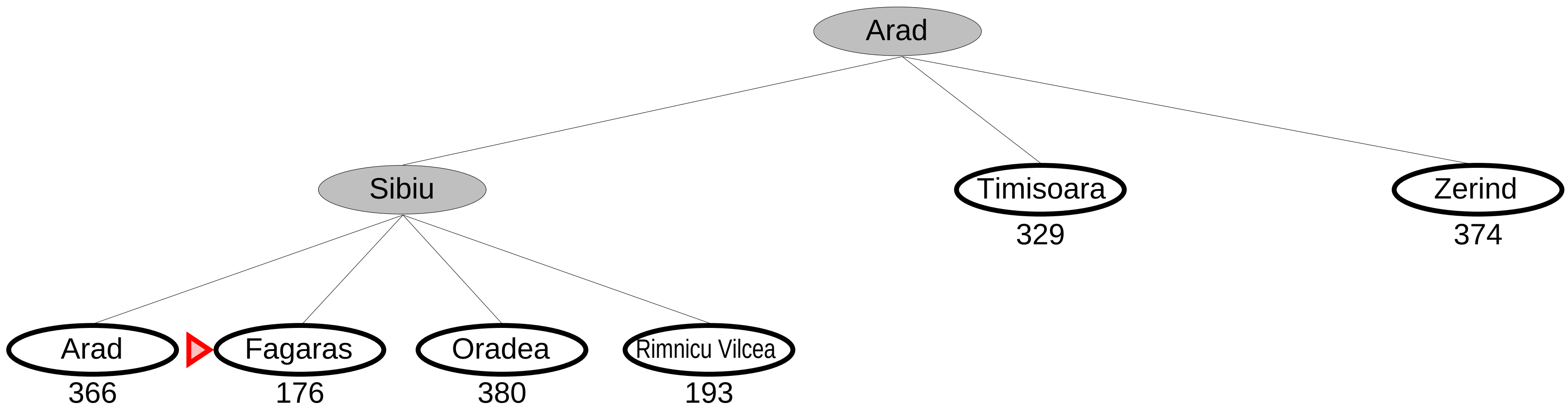


straight-line distance to Bucharest

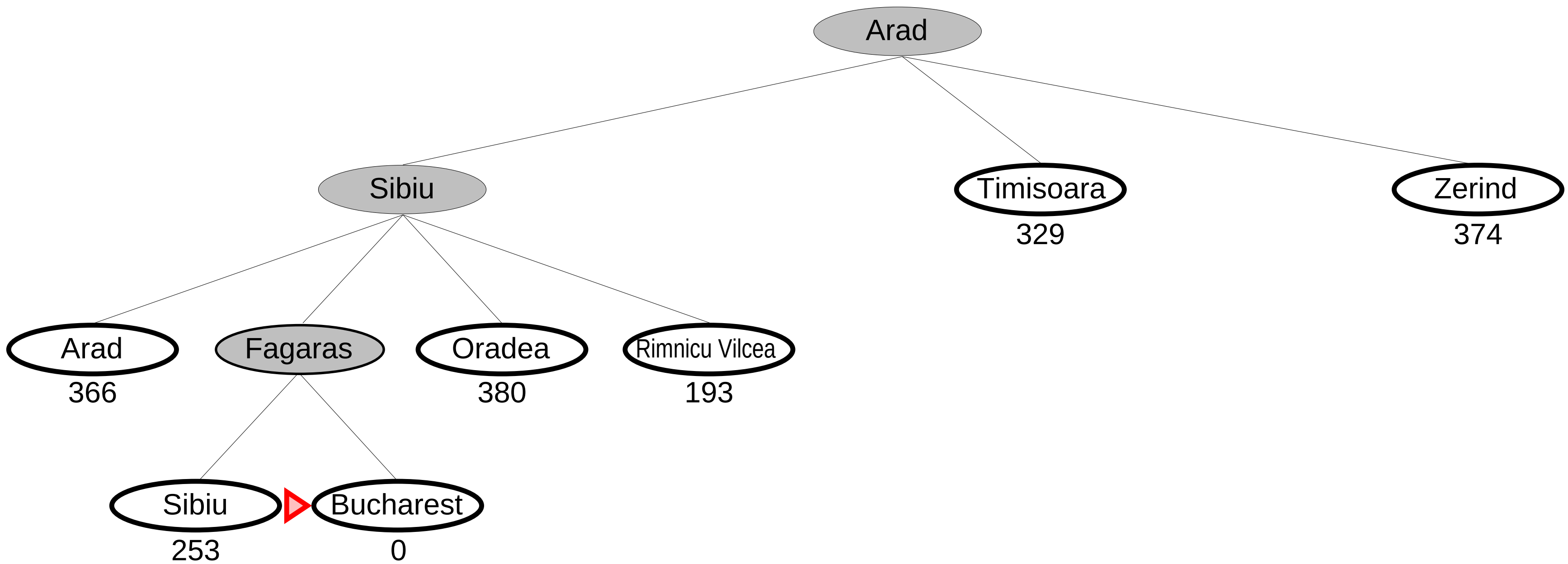
Greedy search example



Greedy search example



Greedy search example



Properties of greedy search

Complete?

Properties of greedy search

Complete? No. Can get stuck in loops:

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

Complete in finite space with repeated-state checking

Time?

Properties of greedy search

Complete? No. Can get stuck in loops:

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

Complete in finite space with repeated-state checking

Time? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?

Properties of greedy search

Complete? No. Can get stuck in loops:

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

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?

Properties of greedy search

Complete? No. Can get stuck in loops:

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

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

Problem with terminology:

Greedy search is not the same as an ordinary **greedy algorithm**.

An ordinary greedy algorithm doesn't remember all of $fringe$.
It remembers only the current path, and never backtracks. Hence:

- ♢ Repeated-state checking cannot make it complete
- ♢ It runs in time $O(l)$ if it finds a solution of length l

A* tree 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

Optimality requirement for A* tree search:

A* needs an *admissible* heuristic, i.e., $0 \leq h(n) \leq h^*(n)$
where $h^*(n)$ is the **true** cost from n.
(Thus $h(G) = 0$ for any goal G.)

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

Theorem: If the optimality requirement is satisfied, then A* tree search never returns a non-optimal solution

A* tree search

Completeness requirement for A* tree search:

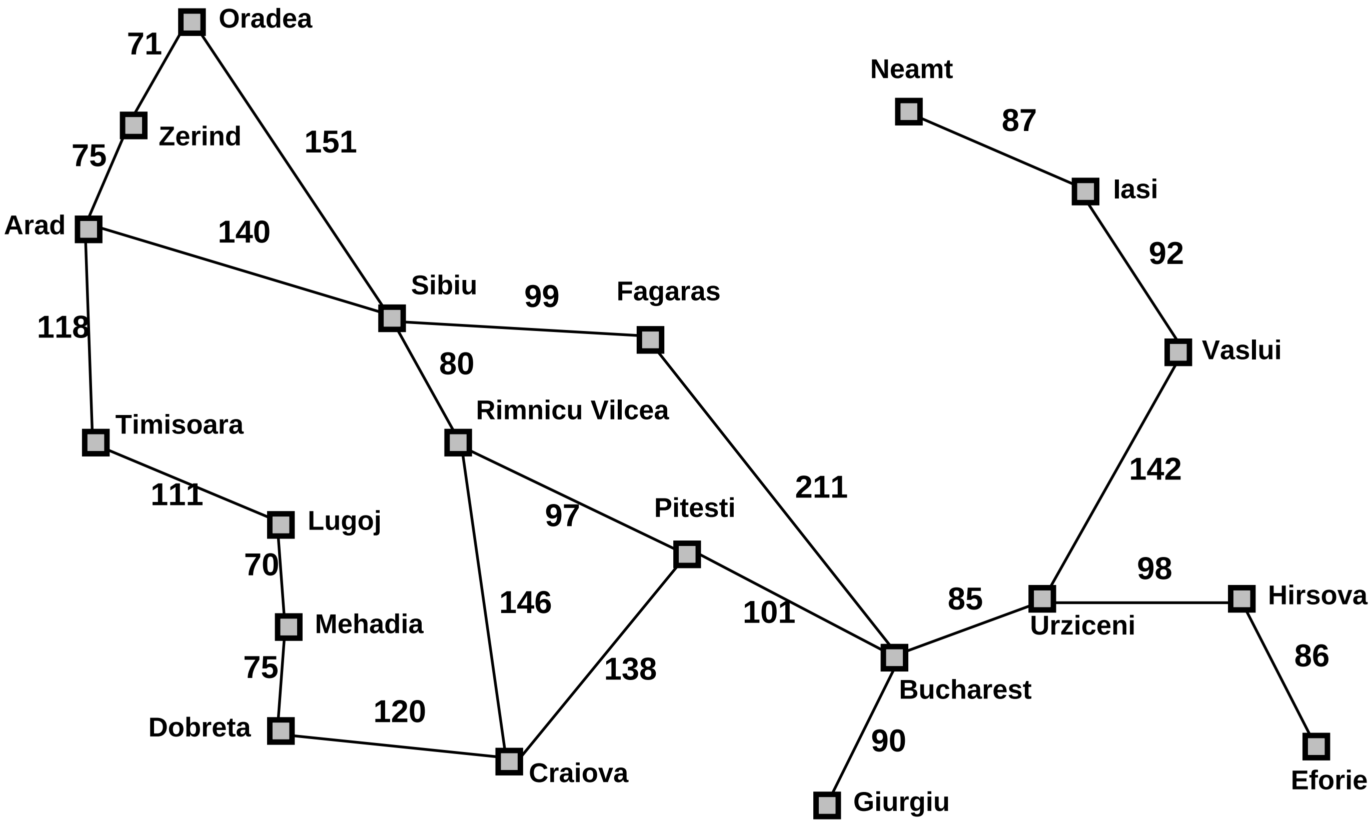
No infinite path has a finite cost

Theorem: On any solvable problem that satisfies the completeness requirement, A* tree search returns a solution.

Corollary: If the optimality requirement also is satisfied, then A* tree search returns an optimal solution.

Romania with step costs in km

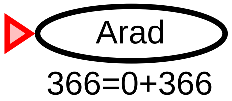
Recall that we want to get from Arad to Bucharest:



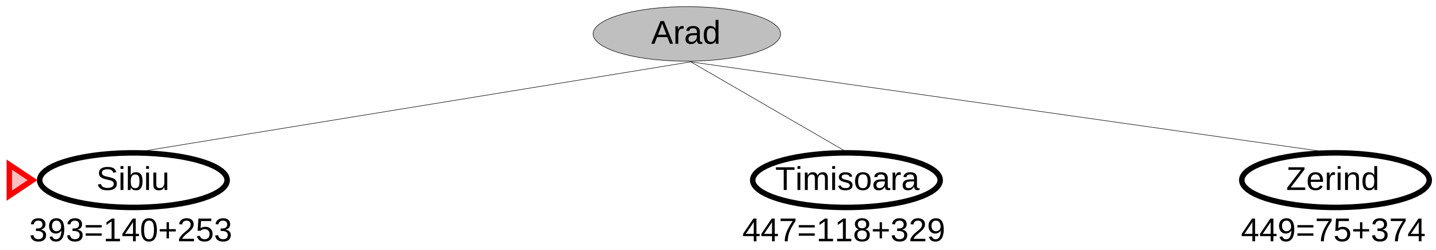
Straight−line distance to Bucharest

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

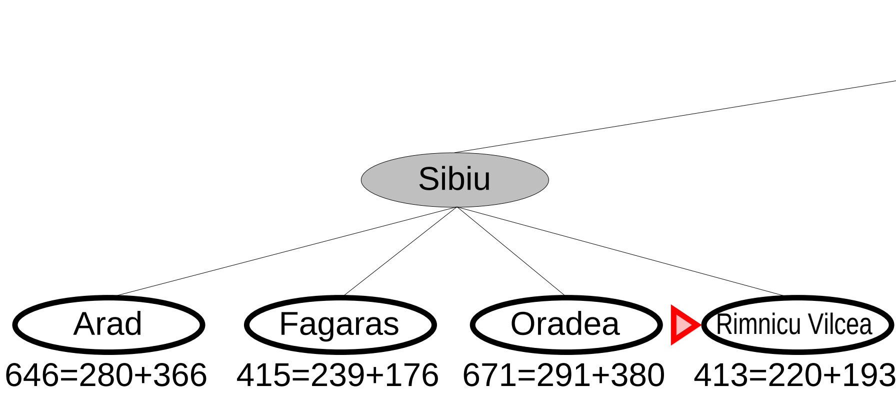
A* tree search



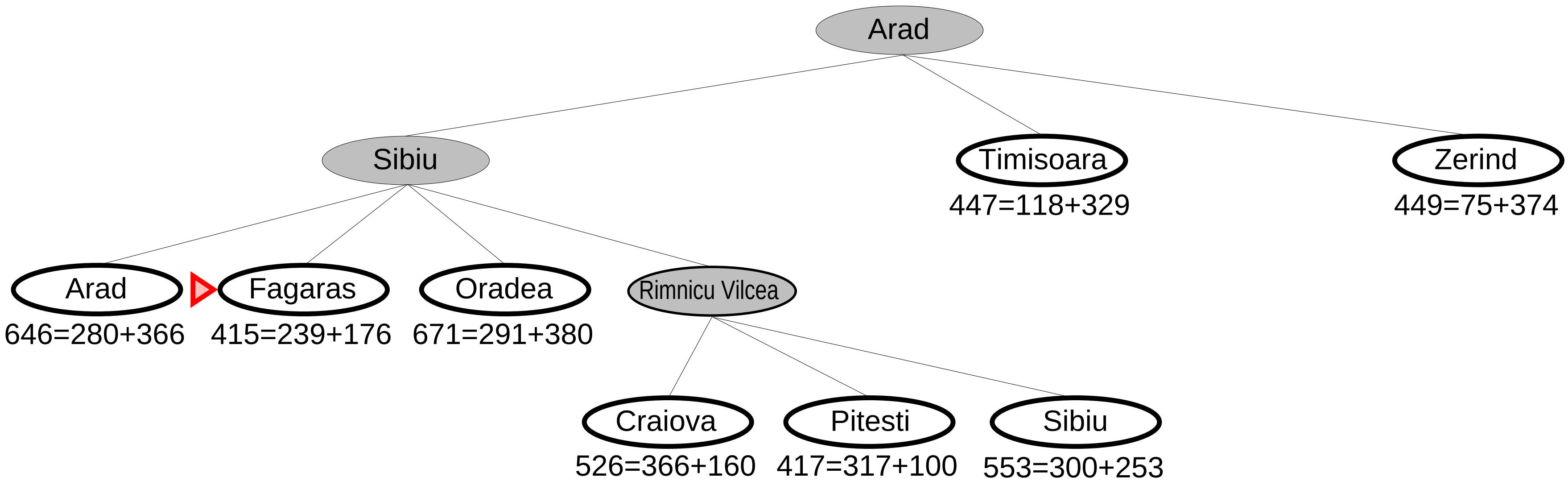
A* tree search



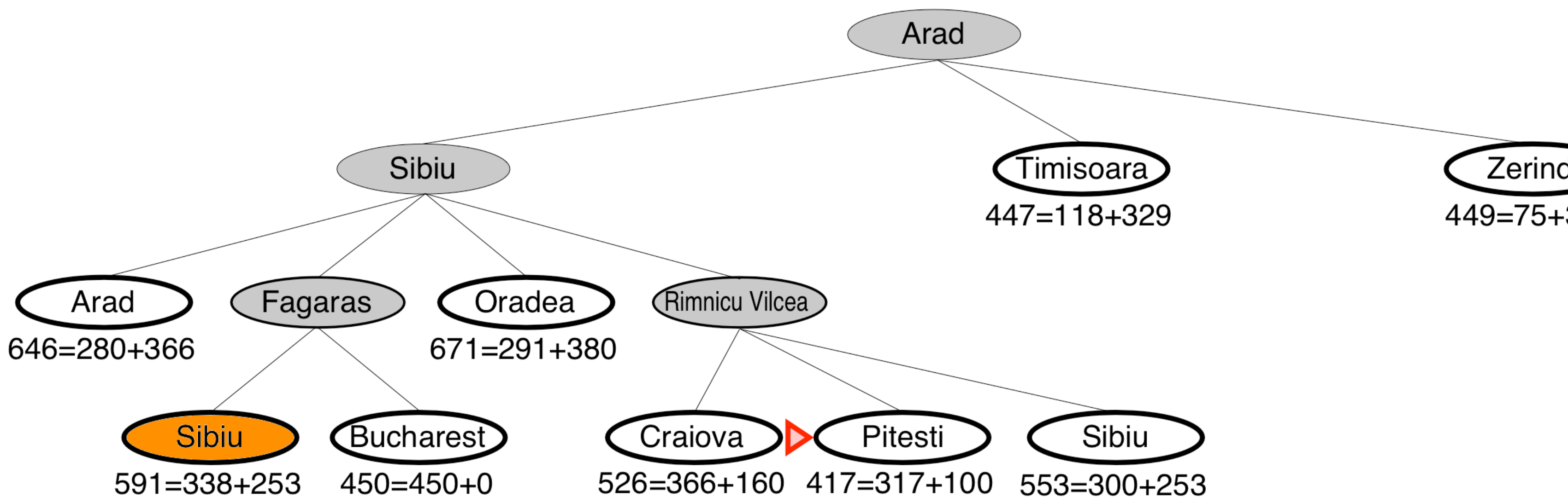
A* tree search



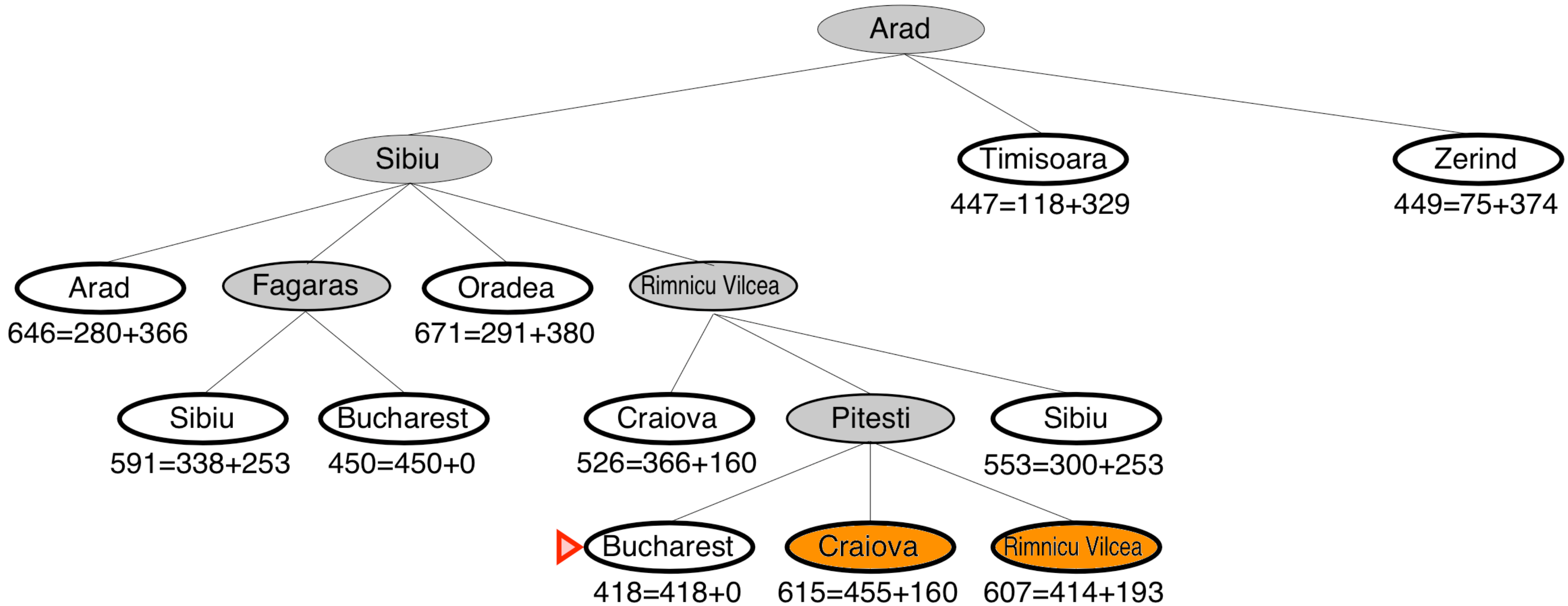
A* tree search



A* tree search

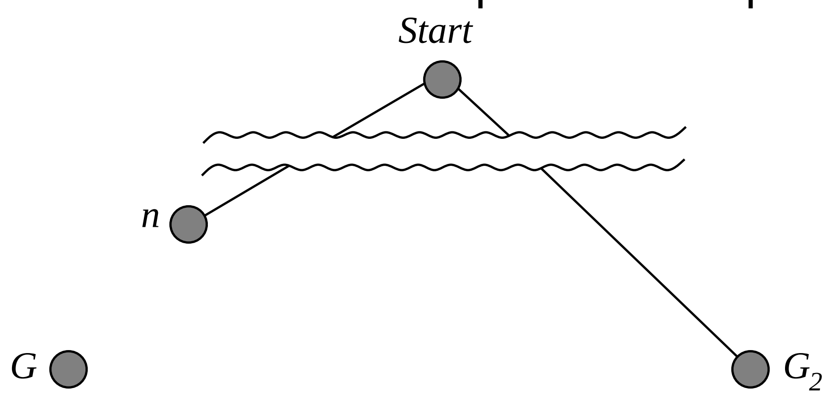


A* tree search



Optimality of A*

Suppose some suboptimal goal G_2 has been generated and is in $fringe$. 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* graph search

A* can also be used with Graph-Search.

Optimality requirement is same as before:

- ♢ The heuristic must be admissible

There are two completeness requirements:

- ♢ One is the same as before: no infinite path has a finite cost
- ♢ The other is something that Russell & Norvig don't mention:

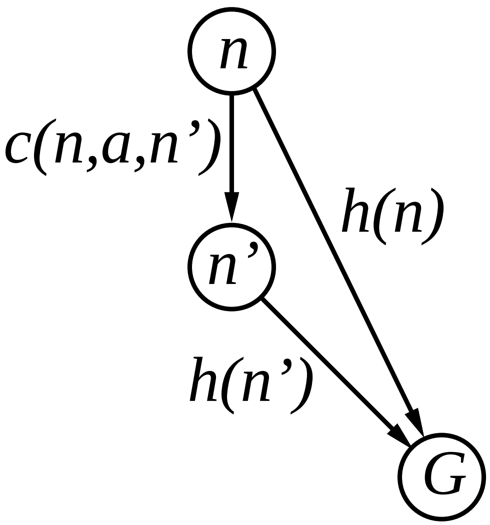
  Either the heuristic needs to be *consistent*,
  or else we need to modify the Graph-Search algorithm

Consistent heuristics

Consistency is analogous to the *triangle inequality* from Euclidian geometry

A heuristic is *consistent* if

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



If h is consistent, then for every child n' of n,

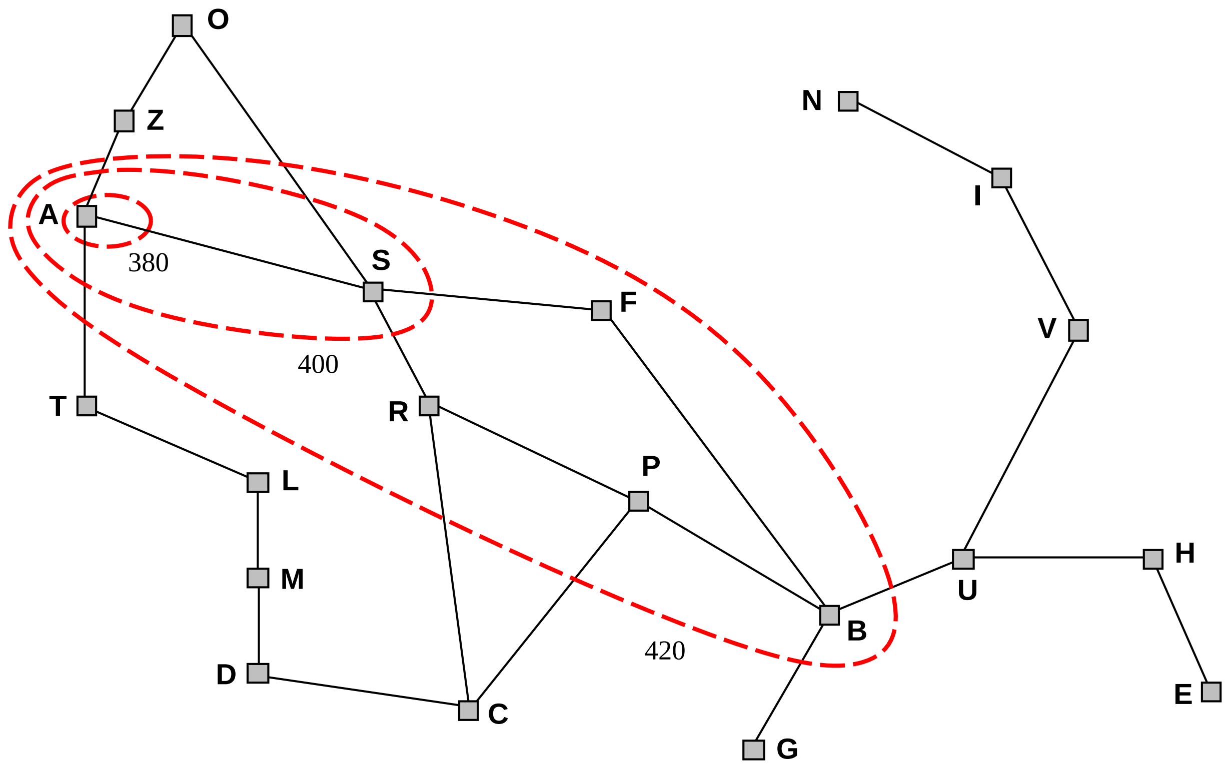
$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

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

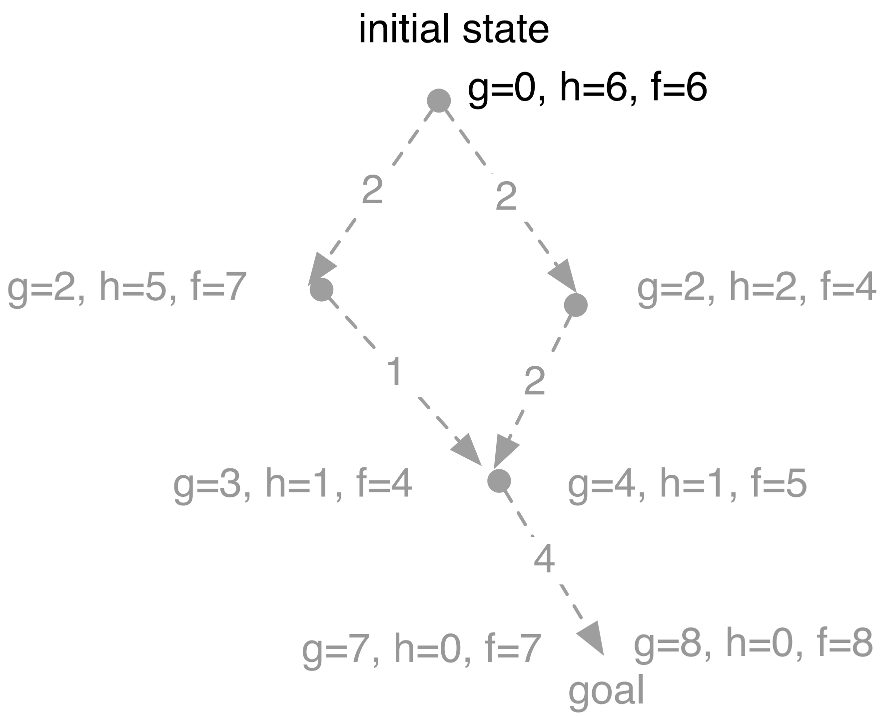
Behavior of A* with consistent heuristic

If h is consistent, then 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 $f_i < f_{i+1}$



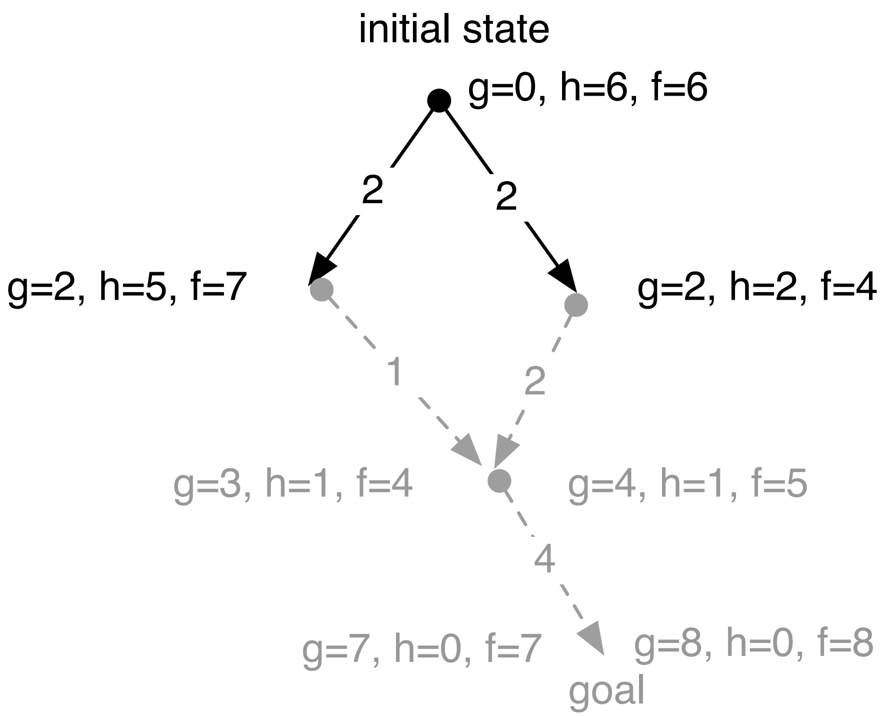
Behavior of A* with inconsistent heuristic



If h is inconsistent, then

- ◇ As we go along a path, f may sometimes decrease
- ◇ A* doesn't always expand nodes in order of increasing f value, because A* may find lower-cost paths to nodes it has already expanded
- ◇ A* will need to re-expand these nodes
- ◇ Problem: Graph-Search won't re-expand them

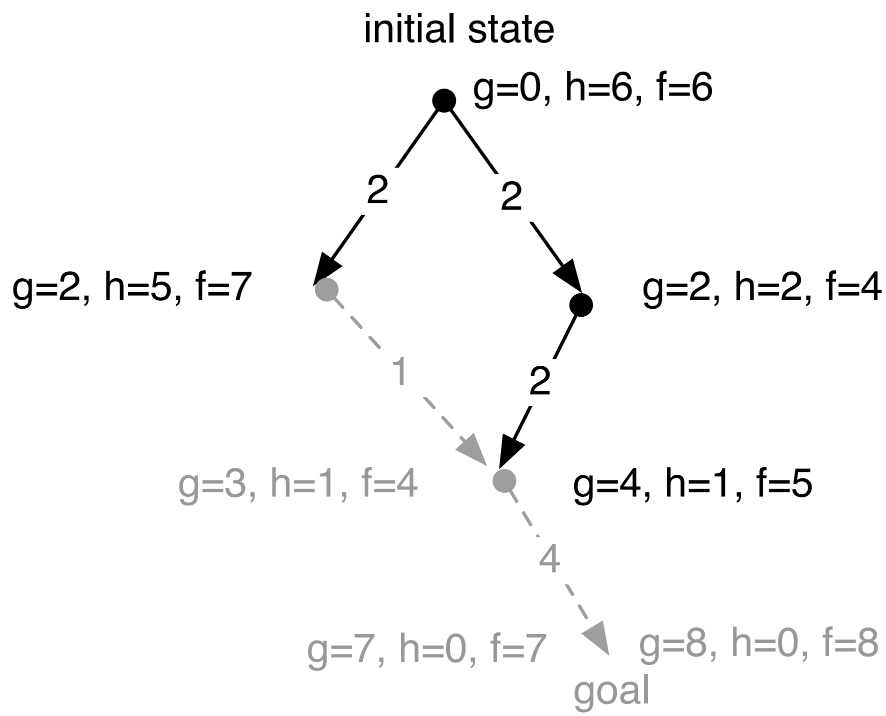
Behavior of A* with inconsistent heuristic



If h is inconsistent, then

- ◇ As we go along a path, f may sometimes decrease
- ◇ A* doesn't always expand nodes in order of increasing f value, because A* may find lower-cost paths to nodes it has already expanded
- ◇ A* will need to re-expand these nodes
- ◇ Problem: Graph-Search won't re-expand them

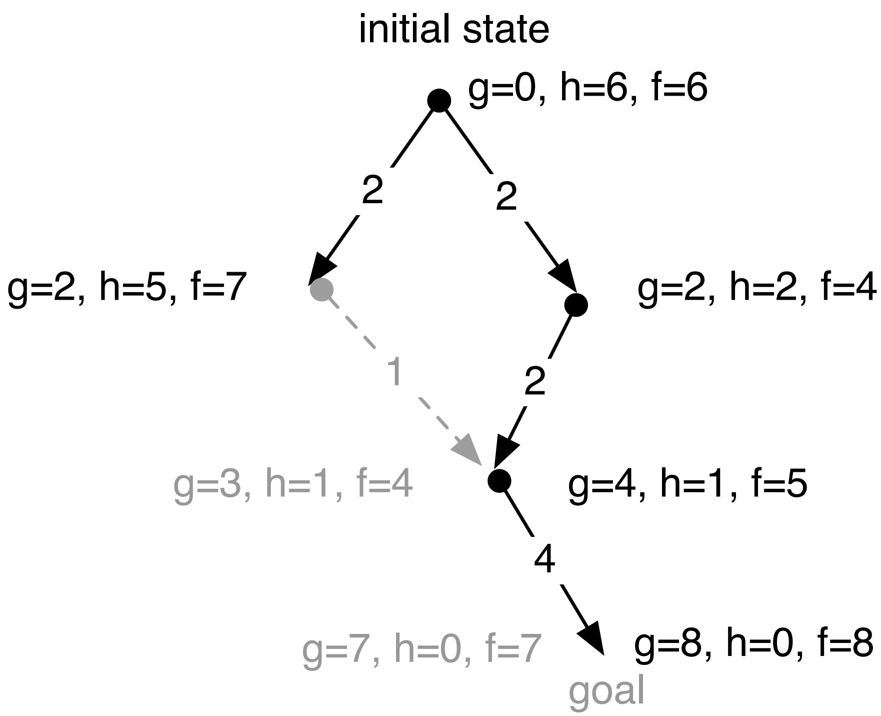
Behavior of A* with inconsistent heuristic



If h is inconsistent, then

- ◇ As we go along a path, f may sometimes decrease
- ◇ A* doesn't always expand nodes in order of increasing f value, because A* may find lower-cost paths to nodes it has already expanded
- ◇ A* will need to re-expand these nodes
- ◇ Problem: Graph-Search won't re-expand them

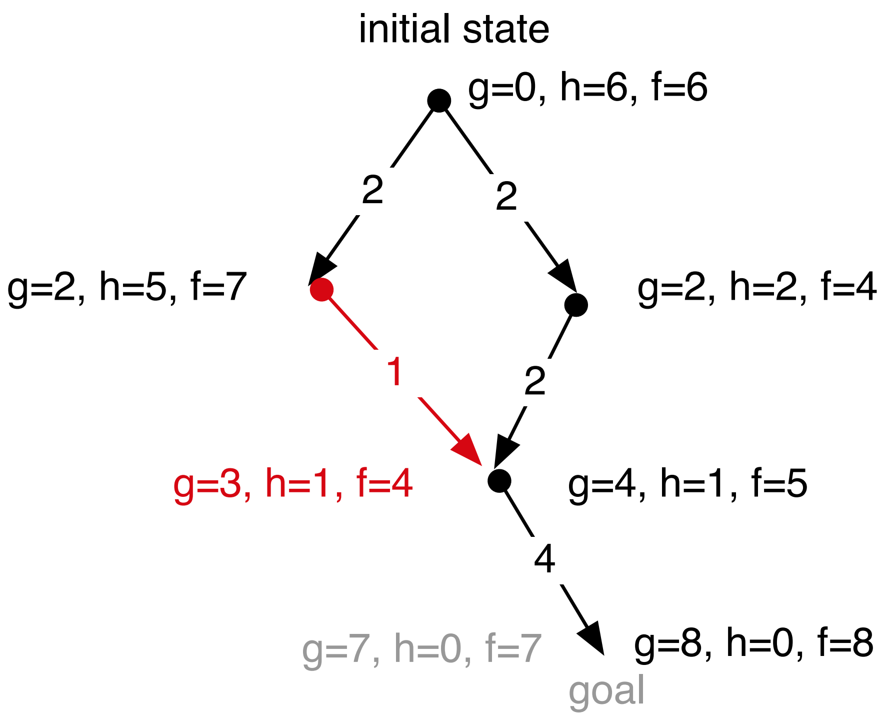
Behavior of A* with inconsistent heuristic



If h is inconsistent, then

- As we go along a path, f may sometimes decrease
- A* doesn't always expand nodes in order of increasing f value, because A* may find lower-cost paths to nodes it has already expanded
- A* will need to re-expand these nodes
- Problem: Graph-Search won't re-expand them

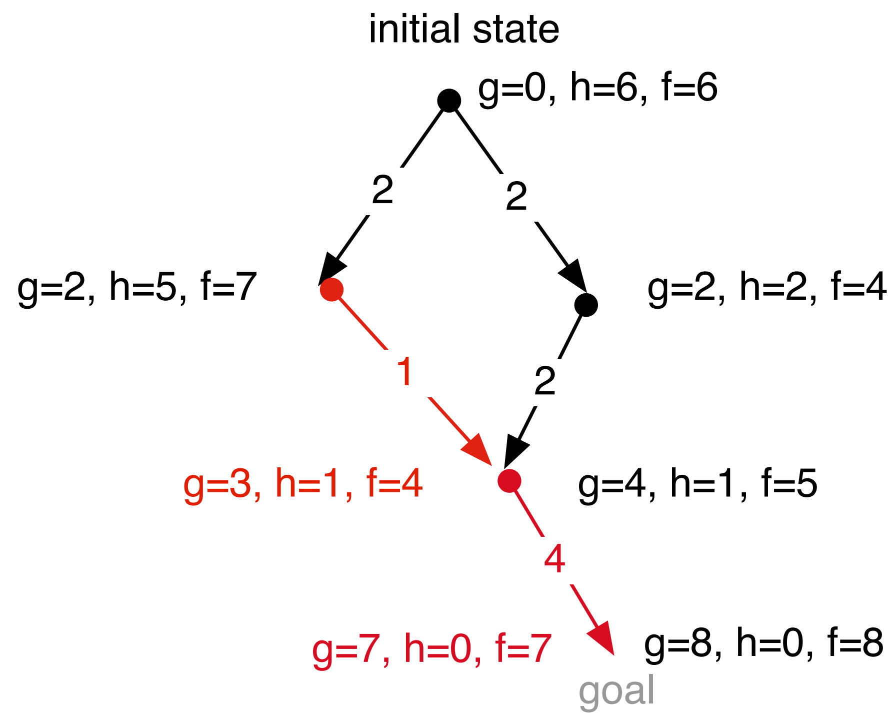
Behavior of A* with inconsistent heuristic



If h is inconsistent, then

- ♢ As we go along a path, f may sometimes decrease
- ♢ A* doesn't always expand nodes in order of increasing f value, because A* may find lower-cost paths to nodes it has already expanded
- ♢ A* will need to re-expand these nodes
- ♢ Problem: Graph-Search won't re-expand them

Behavior of A* with inconsistent heuristic



If h is inconsistent, then

- ◇ As we go along a path, f may sometimes decrease
- ◇ A* doesn't always expand nodes in order of increasing f value, because A* may find lower-cost paths to nodes it has already expanded
- ◇ A* will need to re-expand these nodes
- ◇ Problem: GRAPH-SEARCH won't re-expand them

A* for graphs

- Re-expands a node if it find a better path to the node
- Finds optimal solutions even if the heuristic is inconsistent

function A*(*problem*) **returns** a solution, or failure

    closed ← an empty set
    fringe ← a list containing MAKE-NODE(INITIAL-STATE[*problem*])
    loop do
        if *fringe* is empty **then return** failure
        node ← REMOVE-FRONT(*fringe*)
        if GOAL-TEST[*problem*] applied to STATE(*node*) succeeds **return** *node*
        insert *node* into *closed*
        for each node $n \in$ EXPAND(*node*, *problem*) **do**
            if there is a node $m \in$ *closed* $\cup$ *fringe* such that
                STATE(m) = STATE(n) and $f(m) \leq f(n)$
            then do nothing
            else
                insert n into *fringe* after the last node m such that $f(m) \leq f(n)$
    end

Properties of A*

Complete?

Properties of A*

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

Time?

Properties of A*

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

Time? $O(\text{entire state space})$ in worst case, $O(d)$ in best case

Space?

Properties of A*

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

Time? $O($entire state space$)$ in worst case, $O(d)$ in best case

Space? Keeps all nodes in memory

Finds optimal solutions?

Properties of A*

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

Time? $O($entire state space$)$ in worst case, $O(d)$ in best case

Space? Keeps all nodes in memory

Finds optimal solutions? Yes

Additional properties:

A* expands all nodes in $fringe$ that have $f(n) < C^*$
A* expands some nodes with $f(n) = C^*$

If f is *consistent*, A* expands no nodes with $f(n) > C^*$

How to create admissible heuristics

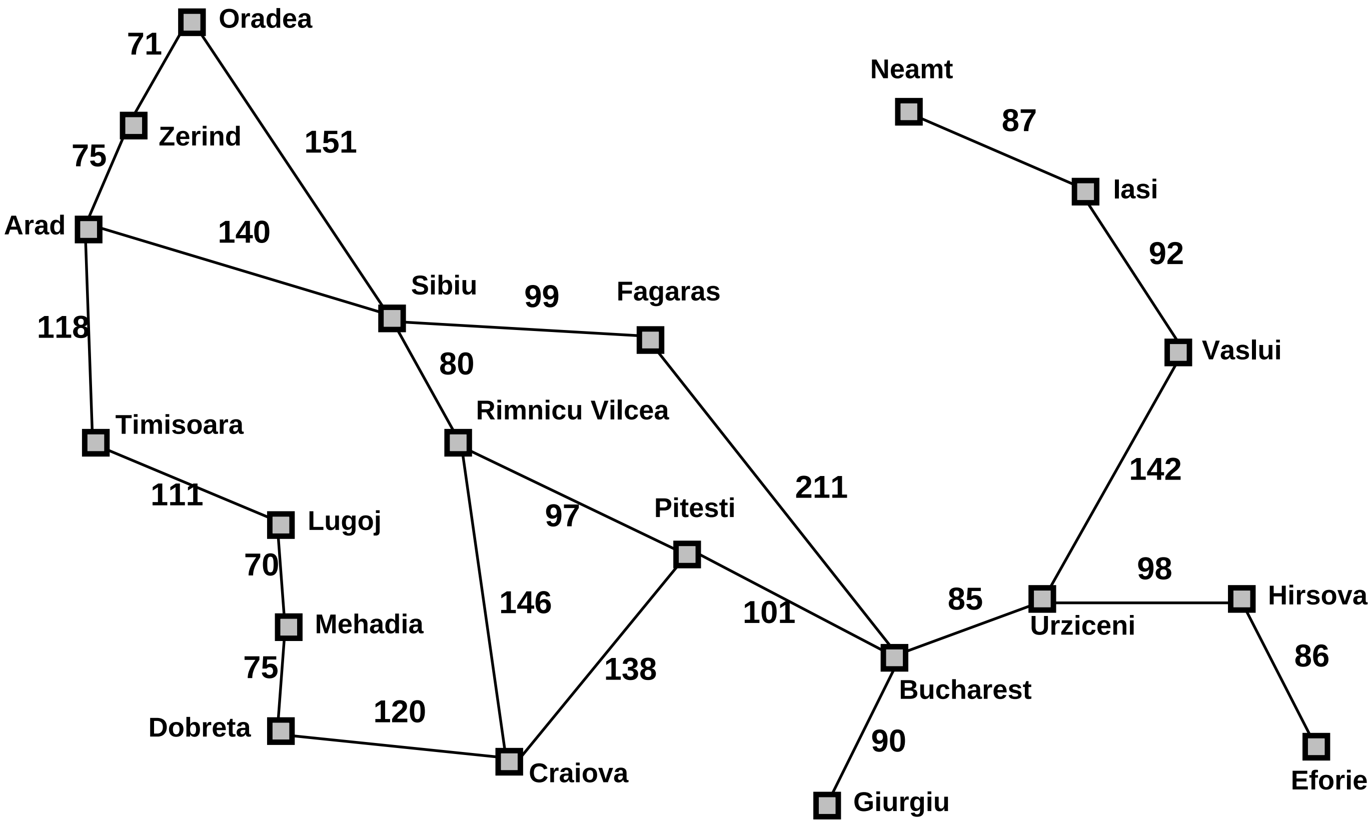
- Suppose P is a problem we're trying to solve
  Let $h^*(s)$ = minimum cost of solution path

- Let P' be a **relaxation** of P
  Remove some constraints on what constitutes a solution

- Every solution path in P is also a solution path in P'
  P' may have additional solution paths that aren't solution paths in P

- Suppose we can find optimal solutions to P' quickly
  Let $h(s)$ = minimum cost of solution path **in** P'
  Then $h(s) \leq h^*(s)$, i.e., h is an admissible heuristic for P

Example: 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

$h(\text{at } c) =$ cost of a straight line from city c to Bucharest

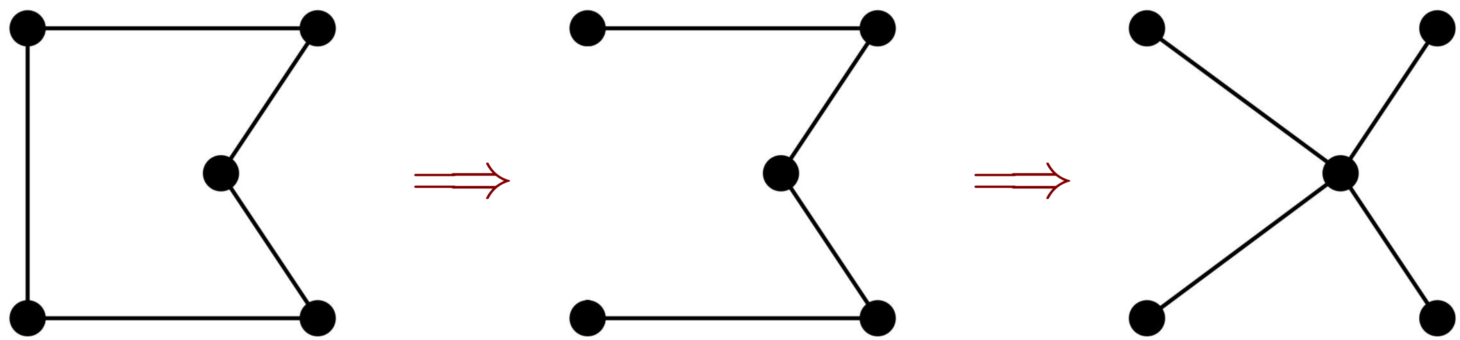
We relaxed the problem to allow paths that are straight lines

Example: TSP

Well-known example: *traveling salesperson problem* (TSP)

♢ Given a *complete* graph (edges between all pairs of nodes)

♢ Find a least-cost *tour* (simple cycle that visits each city exactly once)



Relax the problem twice:

(1) Let {solutions} include paths that visit all cities

(2) Let {solutions} include trees

Minimum spanning tree can be computed in $O(n^2)$

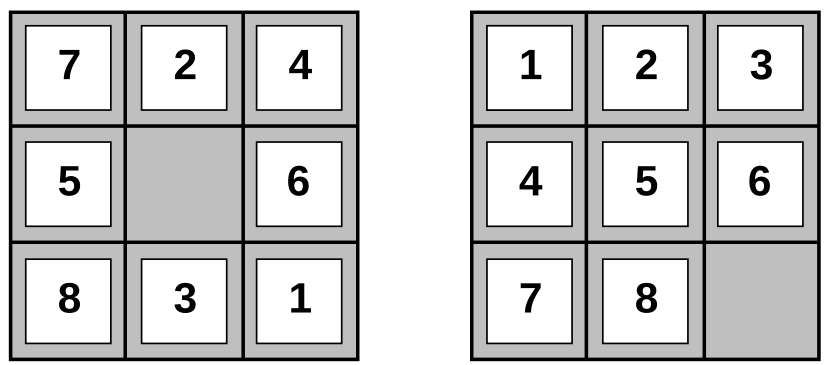
$\Rightarrow$ lower bound on the least-cost path that visits all cities

$\Rightarrow$ lower bound on the least-cost tour

Example: the 8-puzzle

Relaxation 1: allow a tile to move to any other square
regardless of whether the square is adjacent
regardless of whether there's another tile there already

This gives us $h_1(n)$ = number of misplaced tiles



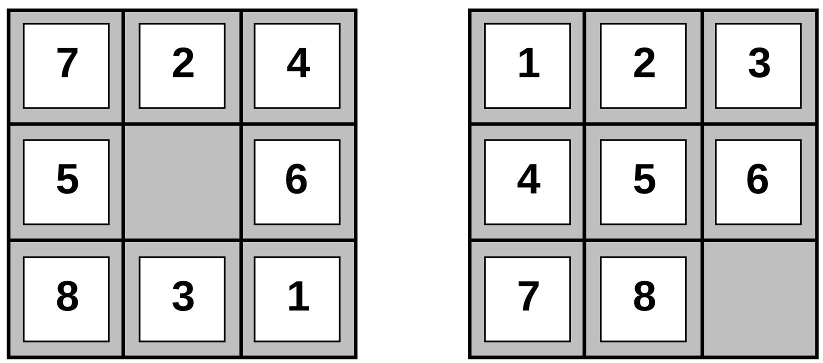
Start State   Goal State

$h_1(S)$ =?

Example: the 8-puzzle

Relaxation 1: allow a tile to move to any other square
regardless of whether the square is adjacent
regardless of whether there's another tile there already

This gives us $h_1(n)$ = number of misplaced tiles



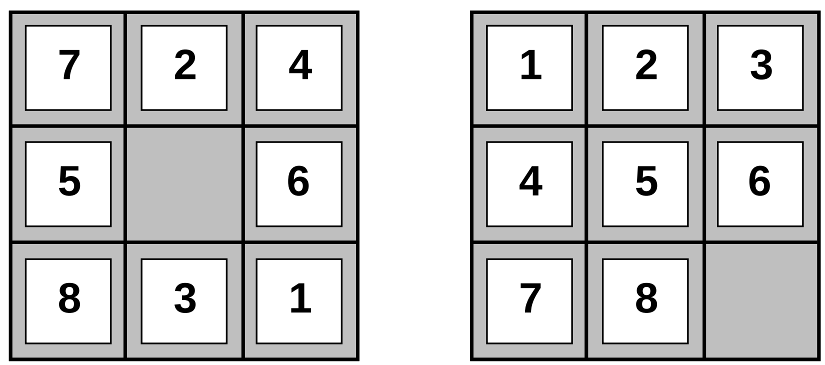
Start State Goal State

$h_1(S)$ =? 6

Example: the 8-puzzle

Relaxation 2: allow a tile to move to any adjacent square, regardless of whether there's another tile there already

This gives us $h_2(n)$ = total *Manhattan* distance



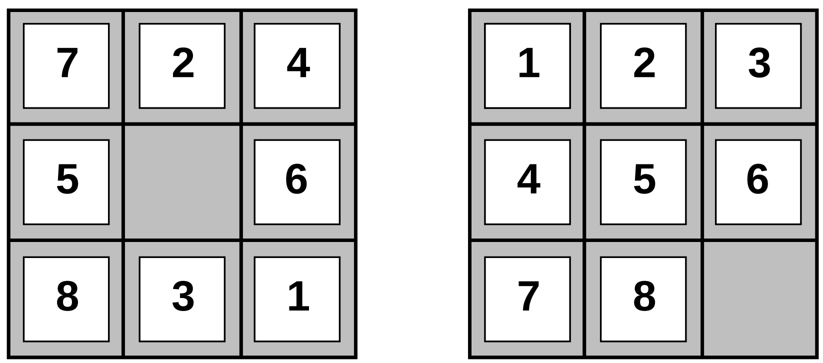
Start State **Goal State**

$h_2(S)$ =?

Example: the 8-puzzle

Relaxation 2: allow a tile to move to any adjacent square, regardless of whether there's another tile there already

This gives us $h_2(n)$ = total *Manhattan* distance



Start State Goal State

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

Dominance

$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total *Manhattan* distance

Notice that $h_1(n) \leq h_2(n) \leq h^*(n)$ for all n,
i.e., h_2 *dominates* h_1,

h_2's estimate of h^* is never worse than h_1's, and is often better than h_1's

Hence h_2 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

One way to get dominance

If h_a are h_b admissible heuristic functions, then
$h(n) = \max(h_a(n), h_b(n))$ is admissible and dominates h_a, h_b

Iterative-Deepening A*

```
function IDA*(problem) returns a solution
    inputs: problem, a problem

    f_0 ← h(initial state)
    for i ← 0 to ∞ do
        result ← COST-LIMITED-SEARCH(problem, f_i)
        if result is a solution then return result
        else f_{i+1} ← result
    end

function COST-LIMITED-SEARCH(problem, fmax) returns solution or number

    depth-first search, backtracking at every node n such that f(n) > fmax
    if the search finds a solution then
        return the solution
    else
        return min{f(n) | the search backtracked at n}
```

Properties of IDA*

Complete??

Properties of IDA*

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

Time?

Properties of IDA*

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

Time? Like A* *if* $f(n)$ is an integer and the number of nodes with $f(n) \leq k$ grows exponentially with k

Space?

Properties of IDA*

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

Time? Like A* *if* $f(n)$ is an integer and the number of nodes with $f(n) \leq k$ grows exponentially with k

Space? $O(bd)$

Optimal?

Properties of IDA*

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

Time? Like A* *if* $f(n)$ is an integer and the number of nodes with $f(n) \leq k$ grows exponentially with k

Space? $O(bd)$

Optimal? Yes

With consistent heuristic:
IDA* cannot expand f_{i+1} until f_i is finished
IDA* expands all nodes with $f(n) < C^*$
IDA* expands no nodes with $f(n) \geq C^*$

Summary

Heuristic functions estimate costs of shortest paths

Good heuristics can dramatically reduce search cost

Greedy search expands lowest h
– incomplete and not always optimal

A* search expands lowest $g + h$
– complete, returns optimal solutions

IDA* is like a combination of A* and IDS
– complete, returns optimal solutions
– much lower space requirement than A*
– same big-O time *if* number of nodes grows exponentially with cost

Admissible heuristics can be derived from exact solution of relaxed problems