Informed search algorithms

Chapter 3, Section 3.5

Outline

- ♢ Best-first search algorithms
- ♢ Greedy search
- ♢ A* search

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-FRONT(fringe)
        if GOAL-TEST[problem] applied to STATE(node) succeeds return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

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

Problem

♢ In uninformed search, **we dont check which of the nodes on the frontier are most promising**. We never look-ahead to the goal.

♢ For the shortest path problem (Arad to Bucharest), all nodes that are less than the cost of the optimal solution (C*) will be expanded by the Uniform Cost search. Even if they are in the wrong direction!

Heuristic-based Informed Searches

Often we have some other knowledge about the **merit/goodness** of nodes.

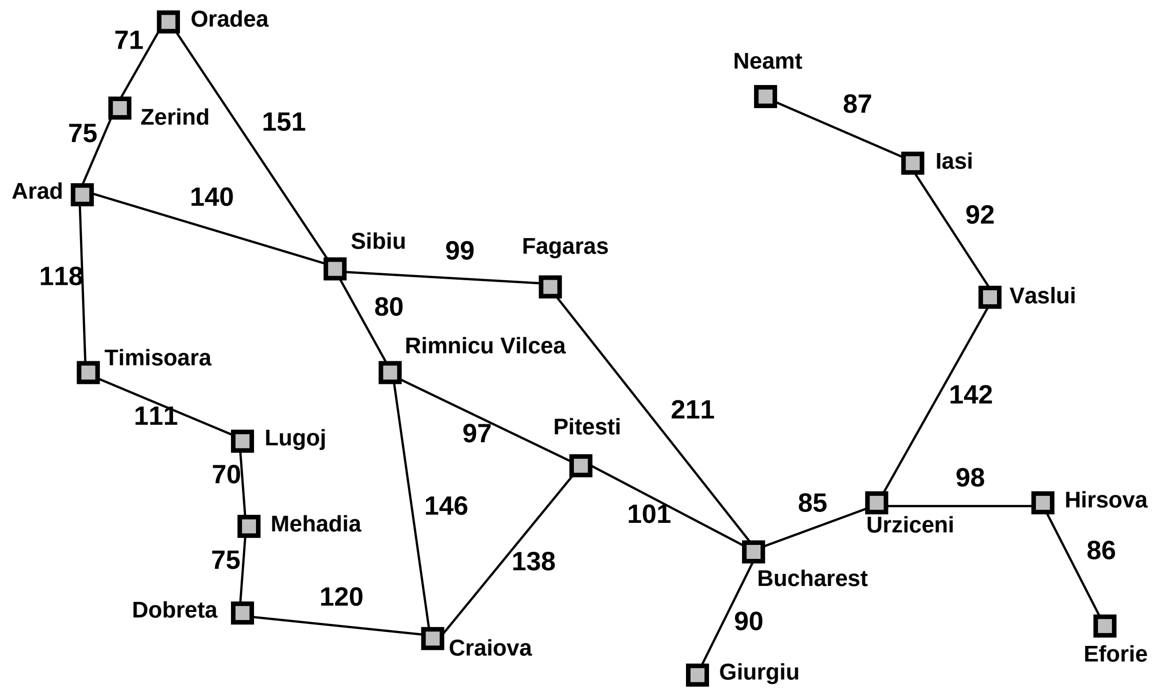
♢ If we are concerned about minimizing search effort, we might want a notion of **finding the goal from that node**.

Heuristics

♢ The idea is to develop a domain specific heuristic function h(n) that estimates the **cost of getting to the goal from node n.**

♢ Heuristics are domain specific.

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

Heuristic for the travel problem: Straight-line distance between a particular state A to the goal state G is a good heuristic estimate of reaching the goal G from A.

Best-first search algorithms

Best-first search is the generic name for the family of search algorithms that **expand the most desirable node in the fringe first**:

Implementation:

♢ insert in order of desirability and remove front

♢ insert to the end of the queue, and find most desirable while picking the front node from the fringe

These are equivalent. Remember that we also made the same comment talking about uninformed searches. They could be seen as using a Queue, a Priority Queue or Stack or just using different insertion/removal strategy to/from the queue that implemented the fringe.

Best-first search algorithms

Expand most desirable unexpanded node using its merit .

Different algorithms will differ in how they measure merit.

- ♢ Greedy search
- ♢ A* search

Greedy search

♢ **Greedy algorithms** in general choose actions that seems to bring the biggest gain at each step of the algorithm.

♢ Greedy search expands nodes that **appears** to be closest to goal without thinking about the future.

Evaluation Function

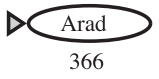
♢ Uniform cost used a function $f(n) = g(n)$ to measure the merit of a node n.

♢ Greedy search uses an evaluation function $f(n) = h(n)$.

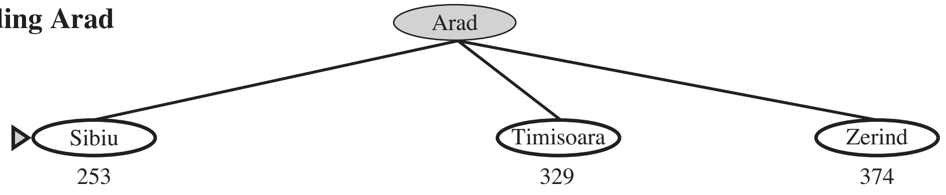
E.g. the $h_{\mathrm{SLD}}(n)$ = straight-line distance from n to Bucharest

Greedy search example

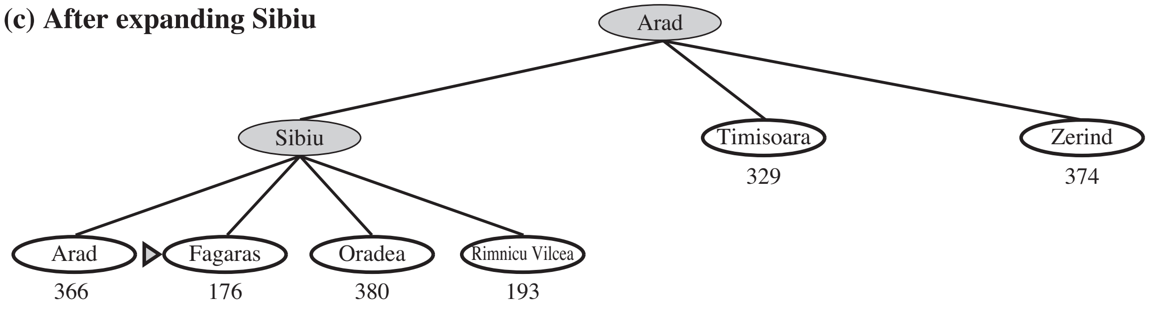
(a) The initial state



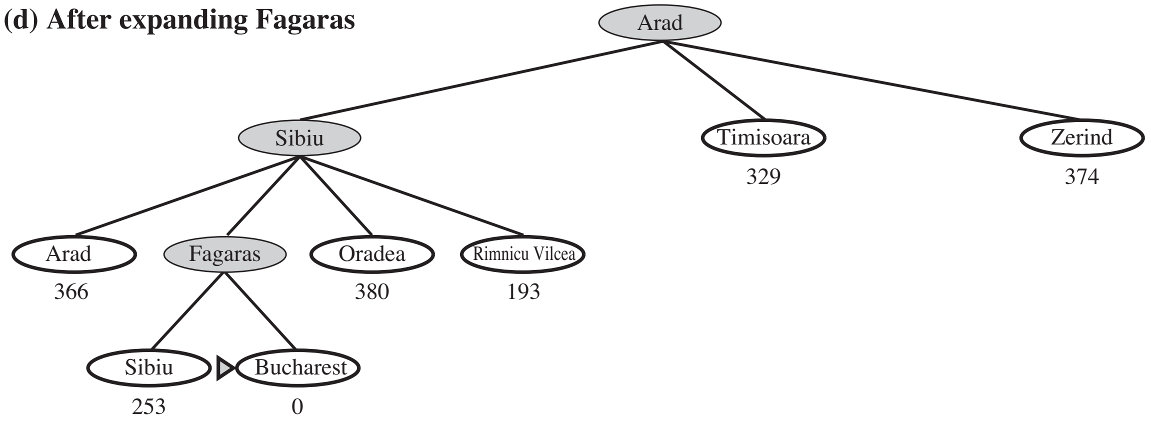
(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Fagaras



Properties of greedy search

Complete??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g., Problem is getting from Iasi to Oradea.

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

Complete in finite space with repeated-state checking

Time??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,

Iasi $\to$ Neamt $\to$ Iasi $\to$ Neamt $\to$

Complete in finite space with repeated-state checking

Time?? $O(b^m)$ —that's the number of nodes in your search tree in the worst-case. But a good heuristic can give dramatic improvement

Space?? $O(b^m)$ —keeps all nodes in memory

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?? $O(b^m)$ —that‘s the number of nodes in your search tree in the worst-case. But a good heuristic can give dramatic improvement

Space?? $O(b^m)$ —keeps all nodes in memory

Optimal?? No. Because it only looks at what appears promising, but maybe the road was longer after all!

A* search

♢ **Idea:** avoid expanding paths that are found to be expensive (even if they appear good in terms of h(n)) by combining UniformCost (which was Optimal and Complete) and Greedy (which tends to be efficient).

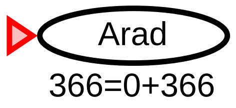
♢ Evaluation function $f(n) = g(n) + h(n)$:

$g(n)$ = cost to reach n

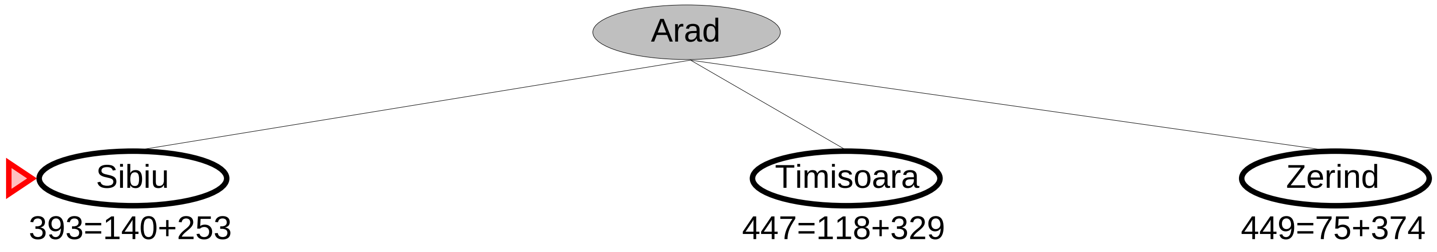
$h(n)$ = estimated cost to goal from n

$f(n)$ = estimated total cost of path **through** n to the goal

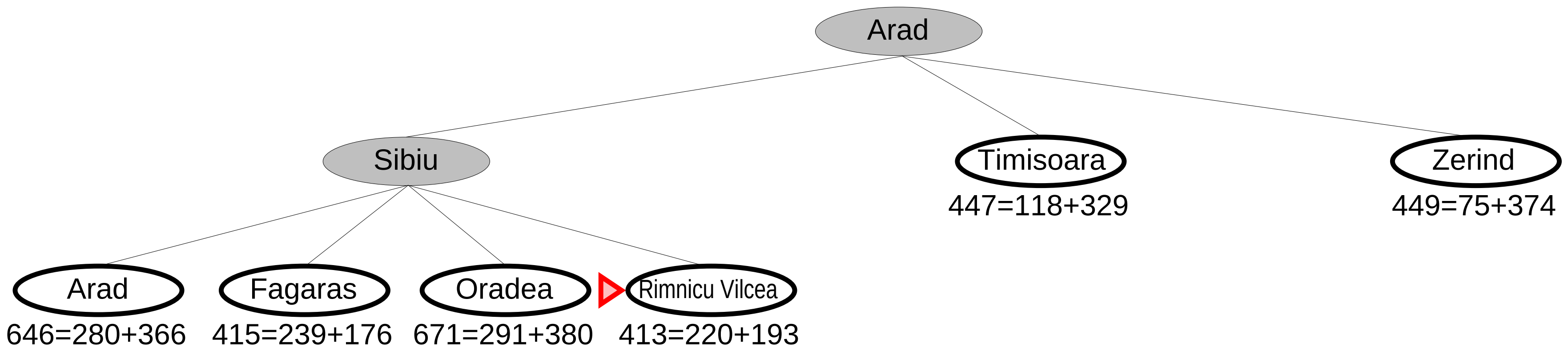
A* search example



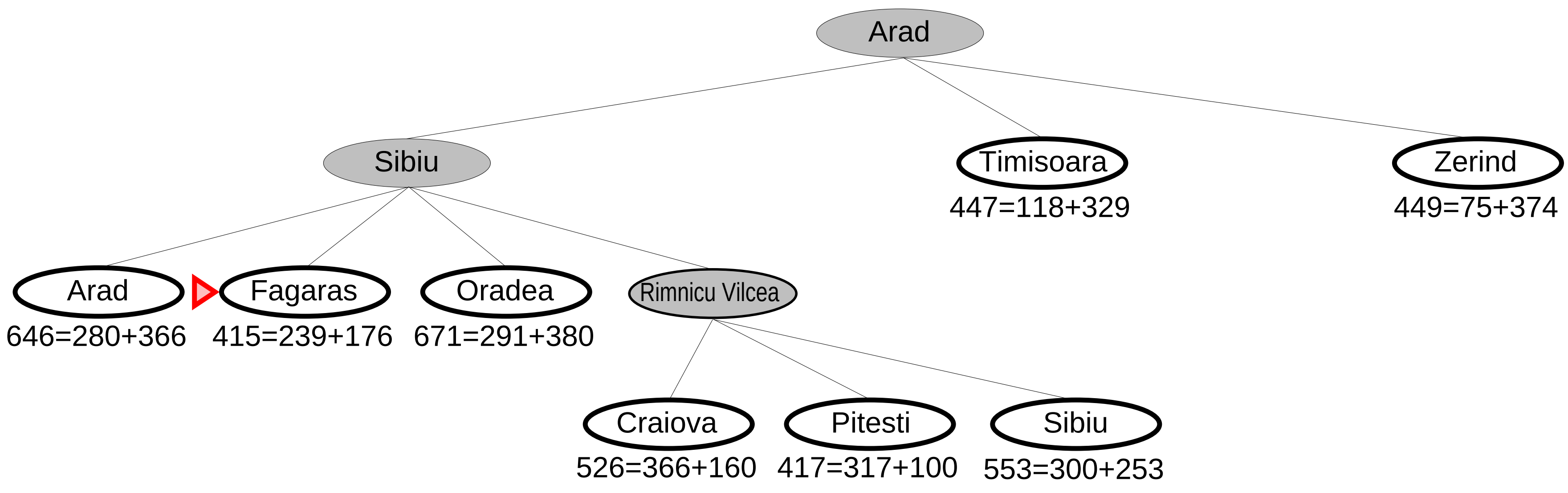
A* search example



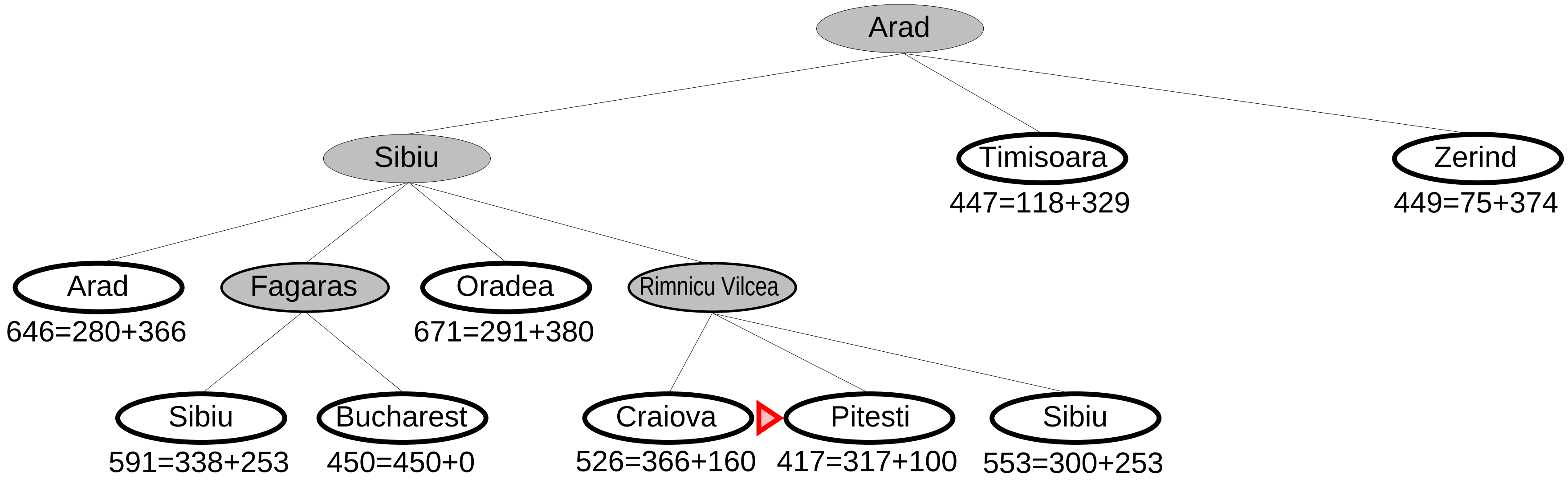
A* search example



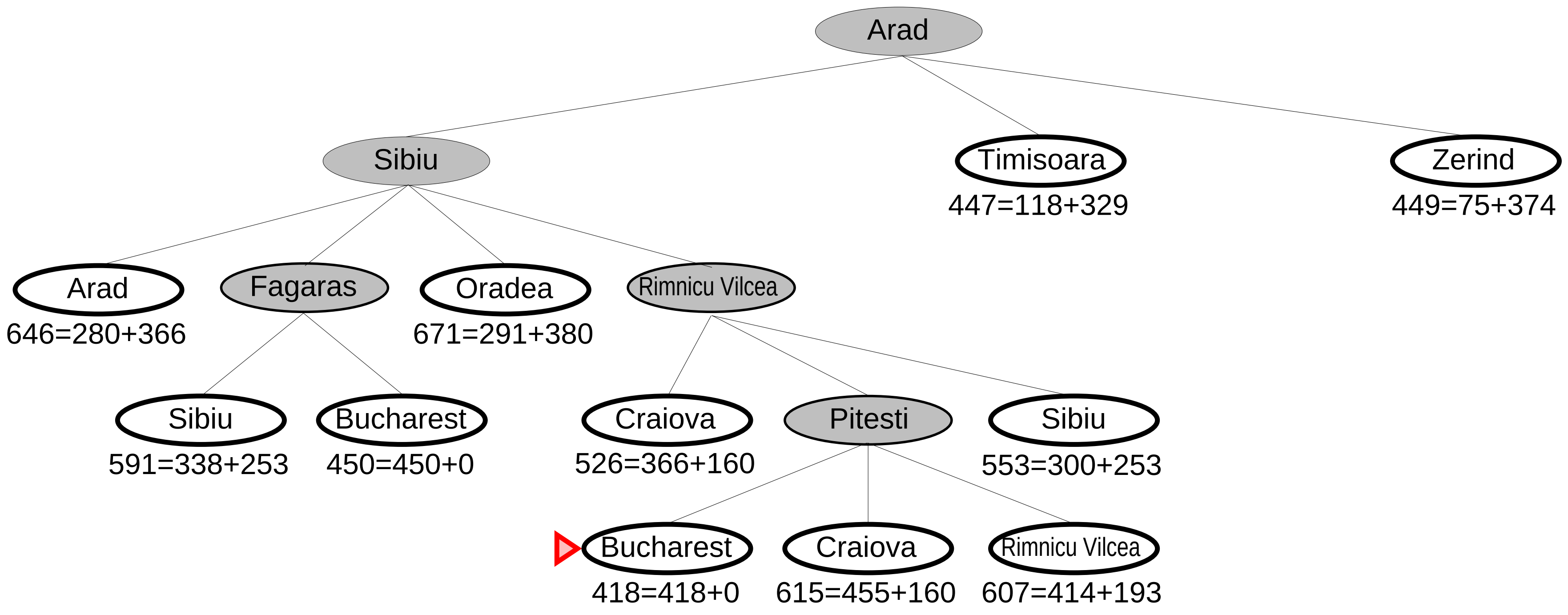
A* search example



A* search example



A* search example



A* search and Admissable heuristics

Assume $h^*(n)$ is the *true* cost of reaching the goal from n.

Definition: $h(n)$ is **admissable** if for all n, $h(n) \leq h^*(n)$

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

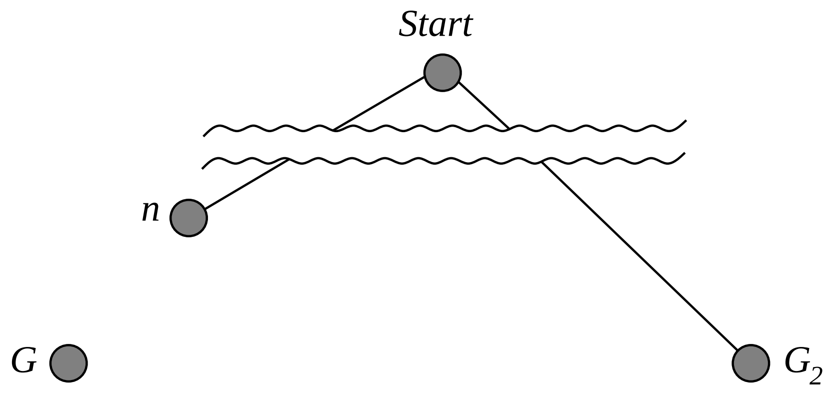
Theorem: A* using TREE search and an admissable heuristic is **optimal**.

We also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G.

A* search and Admissable heuristics

Optimality of A* using TREE Search

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.



We will show that even though G_2 is in the queue, it will not be picked (and lead to suboptimal solution) before n .

Optimality of A* using TREE Search

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

Also,

$$f(n) = g(n) + h(n) < g(G) \qquad \text{since } h \text{ is admissible}$$

Hence, $f(G_2) > f(n)$, and A* will never select G_2 for expansion.

Optimality of A* using Graph Search

In contrast to tree search, A* using graph search (if we are keeping track of previously explored/closed states) may discard the optimal path to a repeated state if it is not the first one generated, so the optimality no longer holds.

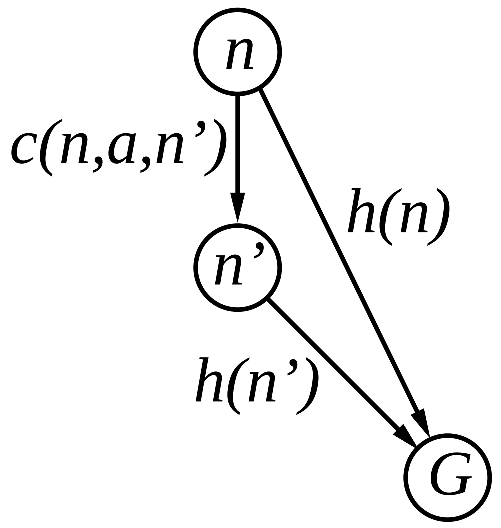
♢ Solution 1: We can change the code with some extra book keeping, so as to discard the more expensive of any two paths found to the same node.

♢ Solution 2: Use a *consistent* heuristic

Consistency

A heuristic is *consistent* if

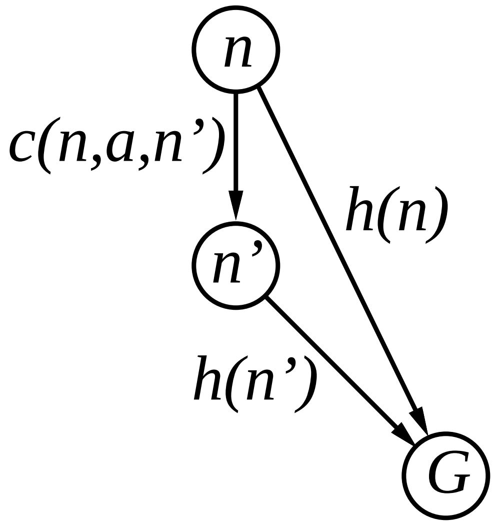
$$h(n) \leq c(n, a, n') + h(n')$$



Optimality of A*

A heuristic is *consistent* if

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



Roughly speaking, the heuristic value h(n) at any node n, should be an ‘underestimate‘ again: the actual cost to + remaining cost at a later node n‘ should be higher (i.e. more info should indicate at least as high a cost).

Proof: Optimality of A*

♢ If h is consistent, then we have

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

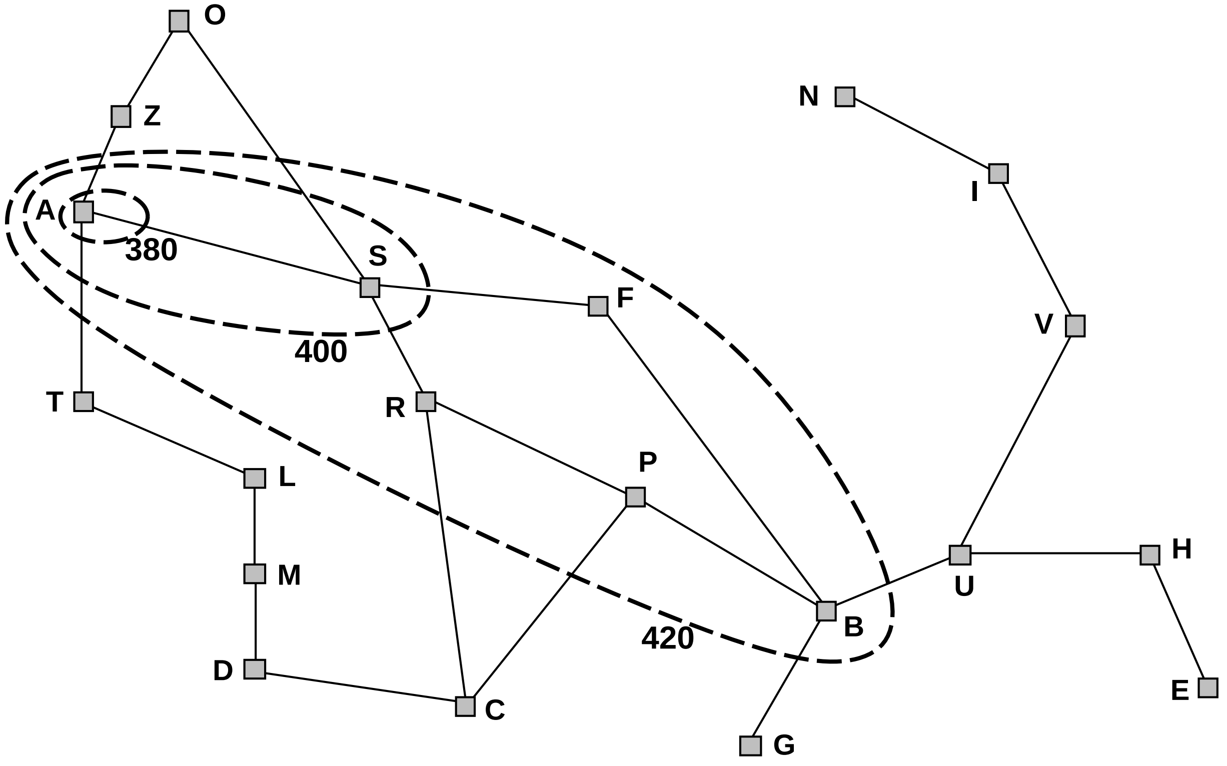
Hence, for consistent (also called **monotonic**) heuristics, the values of f(n) along any path are non-decreasing.

♢ Note that consistency is a stricter requirement than admissability: all consistent heuristics are admissable, but not necessarily the other way around (although most admissable heuristics are also consistent).

Behavior of A* Using GRAPH Search

Lemma: A* expands nodes in order of increasing f value*

Gradually adds "f-contours" of nodes
Contour i has all nodes with $f < f_i$ and $f_i < f_{i+1}$



Optimality of A* Using GRAPH Search

A* is optimal, since it 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 $f(n) > C^*$

Note: Breadth-first adds layers, uniform cost search adds concentric bands...

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 soln.]

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 $f(n) > C^*$

Properties of A*

A* is *optimally efficient*: among the search algorithms that start from the root and search using the same heuristic, it is the most efficient (may consider extra only those nodes with f cost exactly equal to C*).

Space is the main problem with A* like other Graph Search algorithms that keeps track of a Closed/Explored list.

Detail: The graph search algorithms typically need to check whether the reached **state** is a repeated state, but Version 3 of AIMA allows for checking repeated nodes in the given pseudocode, allowing different algorithms to implement what they need (to check for states or nodes). In anycase, space requirements are often large.

Advanced: A* Alternatives

There are variants of A* that are still complete and optimal, but try to reduce the space complexity.

♢ Iterative Deepening A*: adapt the idea of iterative deepening to the heuristic search context, where the cutoff used is the f -cost (g + h) rather than the depth. At each iteration, the cutoff value is the smallest f-cost of any node that exceeded the cutoff on the previous iteration.

♢ Memory-bounded A$^*(MA)$: expands the best leaf until memory is full; then drops the worst leaf node, backs up the value of f-cost of the forgotten node to its parent (to take it up if nothing else is better). The SMA* variant (simplified MA) is complete and optimal if any optimal solution is ‘reachable‘ and if not, it returns the best solution found.