Artificial Intelligence
CE-417, Group 1
Computer Eng. Department
Sharif University of Technology

Fall 2023

By Mohammad Hossein Rohban, Ph.D.

Courtesy: Most slides are adopted from CSE-573 (Washington U.), original slides for the textbook, and CS-188 (UC. Berkeley).

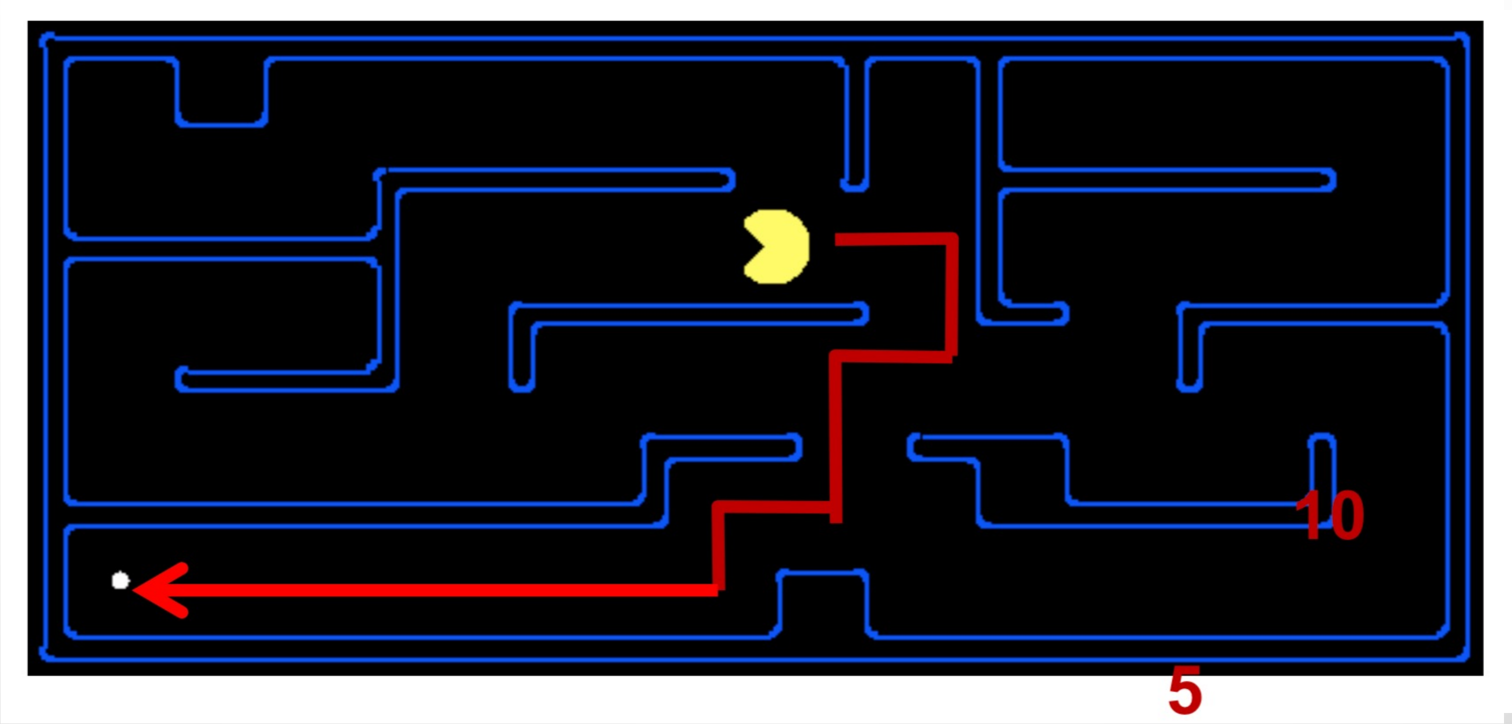
Informed search

Blind vs. Heuristic Search

- Blind:
  - Search in all directions systematically
- Heuristic Guidance:
  - How far is the goal state from a given state approximately?

What is a "Heuristic"?

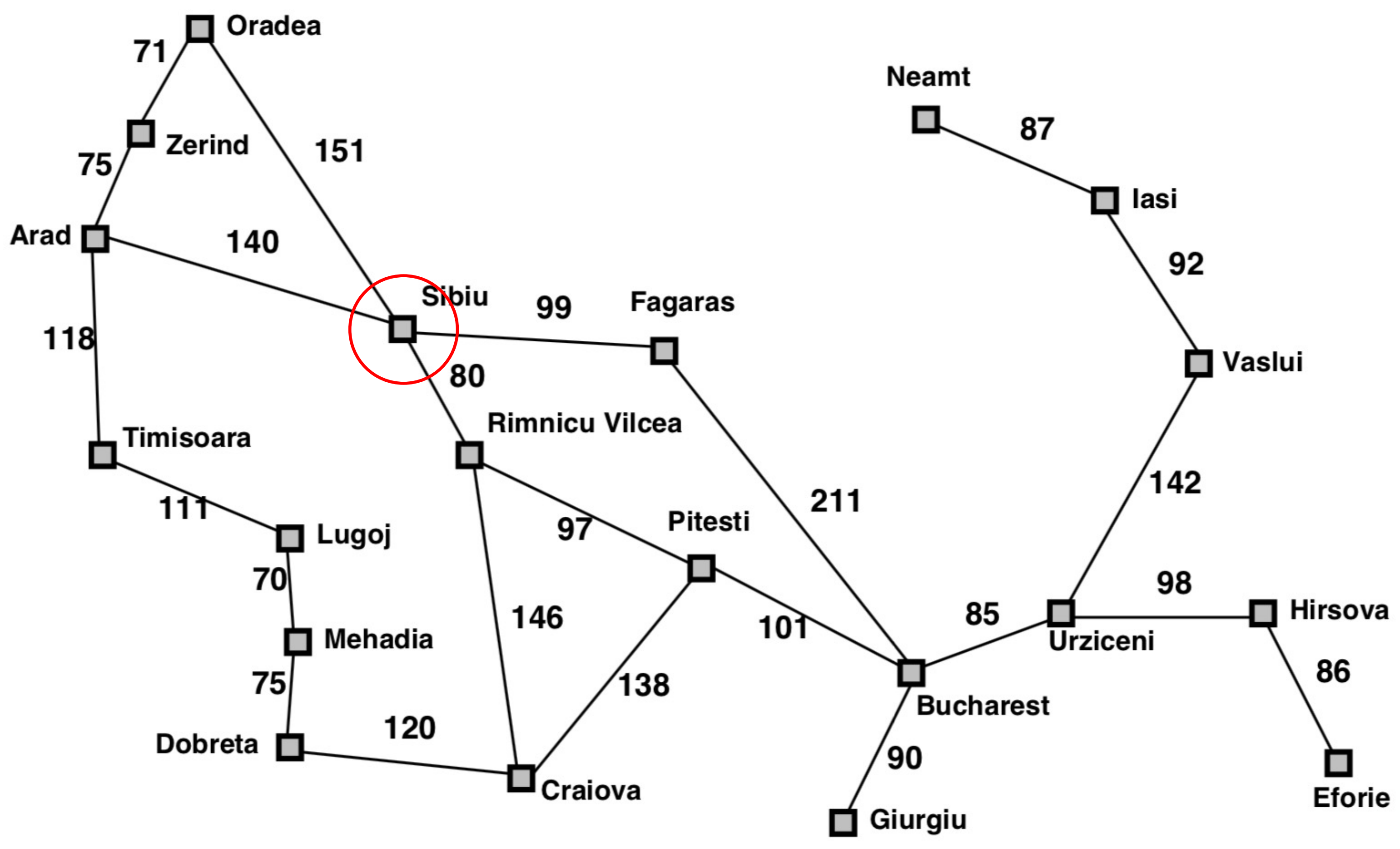
- An ***estimate*** of how close a state is to a goal
- Designed for a particular search problem



- Examples: Manhattan distance: 10+5 = 15; Euclidean distance: 11.2
- Actual distance to goal: 2+4+2+1+8= 17

Greedy Search

- *Best first with f(n) = heuristic estimate of distance to goal*

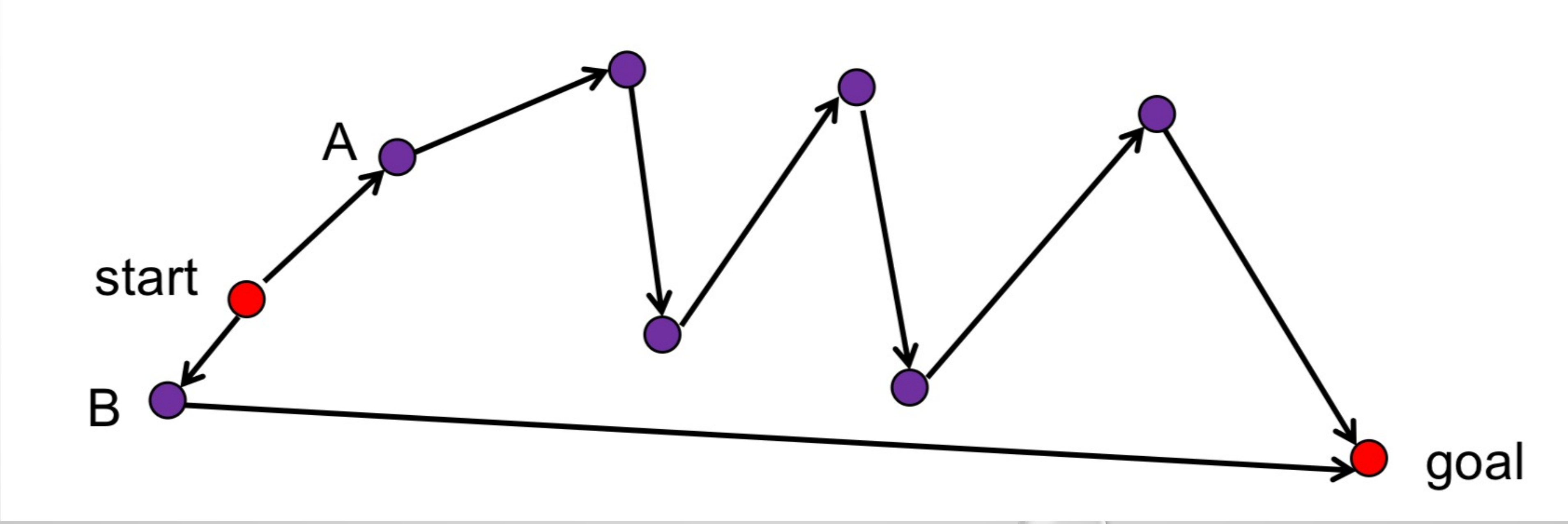


Straight–line distance to Bucharest

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

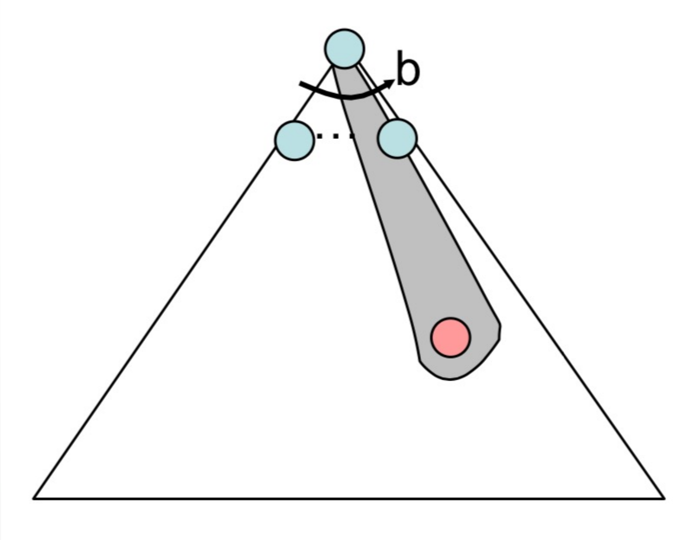
How can it go wrong?

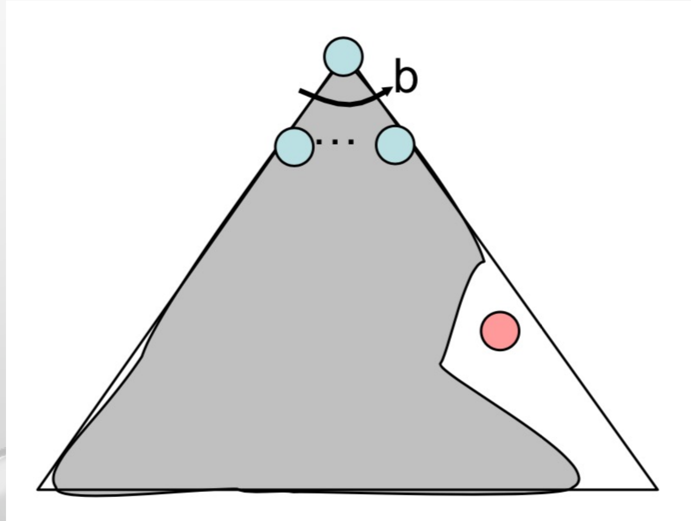
- Expand the node that seems closest…



Problems with the Greedy Search

- **Common case**:
  - Best-first takes you straight to a (suboptimal) goal
- **Worst-case**: like a badly-guided DFS
  - Can explore everything
  - Can get stuck in loops if no cycle checking
- Like DFS in completeness
  - Complete w/ cycle checking
  - *If* finite # states





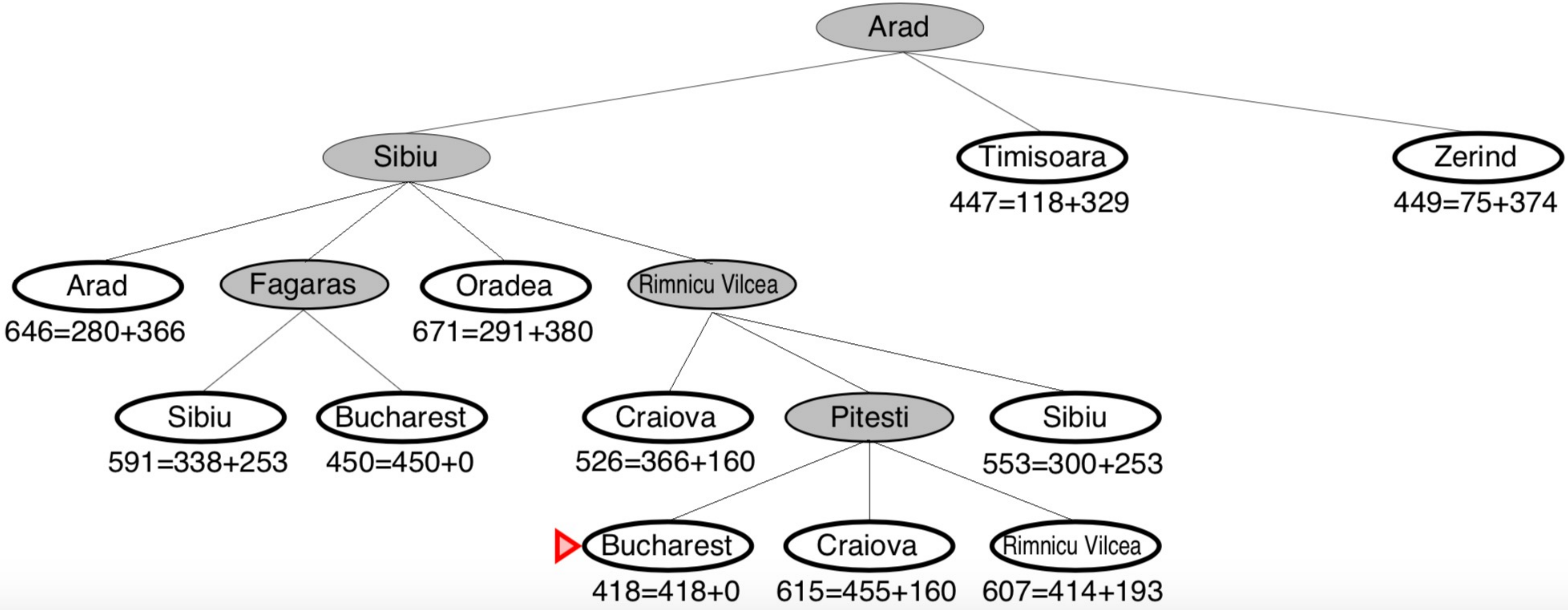
Properties of greedy search

- Complete:
  - No–can get stuck in loops, e.g., Lasi → Neamt → Lasi → Neamt →
  - 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

- Hart, Nilsson & Rafael 1968
- Best first search with $f(n) = g(n) + h(n)$
  - $g(n)$ = sum of costs from start to n
  - $h(n)$ = estimate of lowest cost path $n \rightarrow$ goal
- $h(goal) = 0$
- Can view as cross-breed:
  - $g(n) \sim$ uniform cost search
  - $h(n) \sim$ greedy search
- Best of both worlds...

A* example



Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

A* optimality (tree-search)?

Theorem: If h(n) is admissible then A* is optimal in tree search.

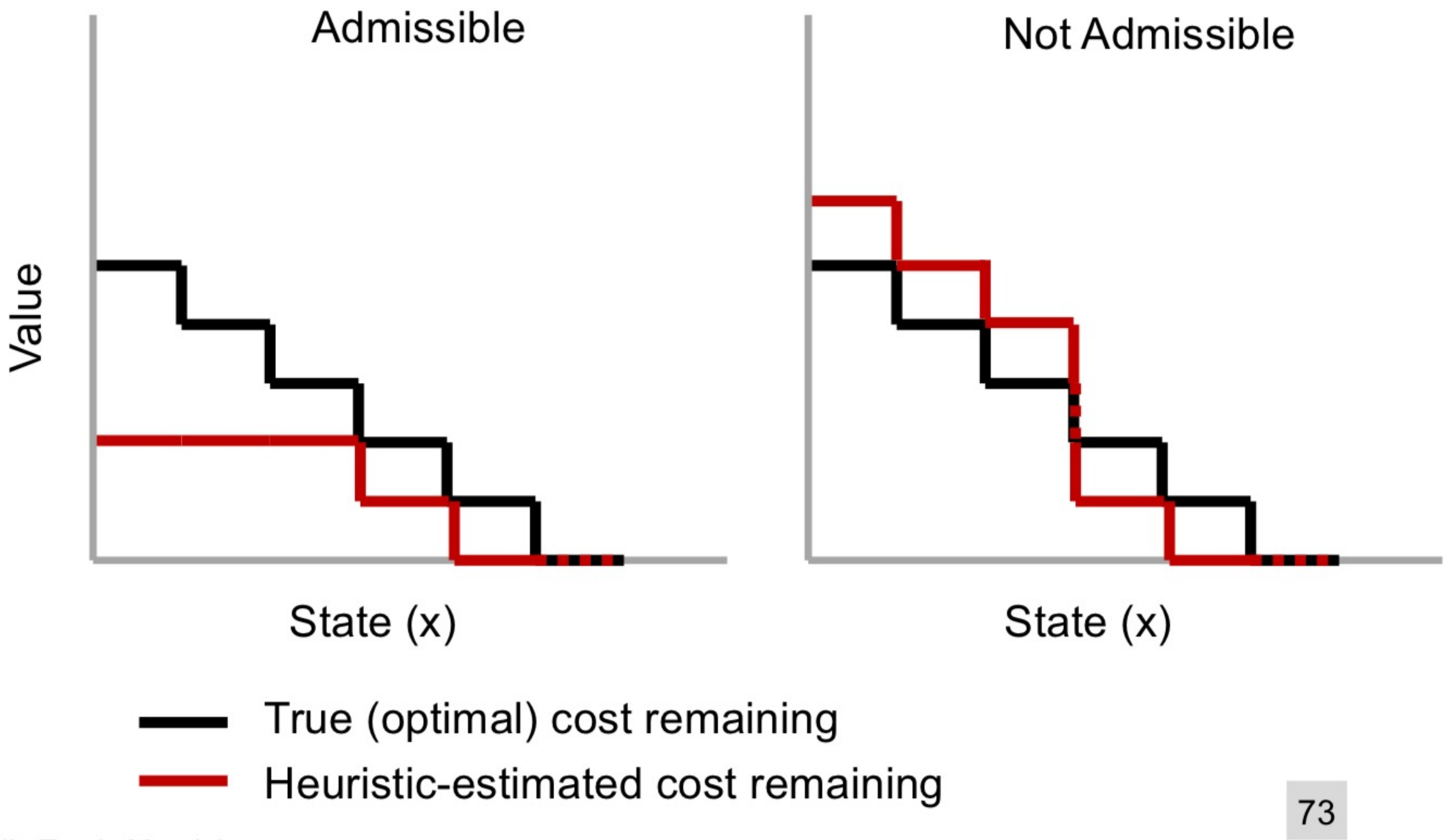
A* optimality (graph-search)?

If h(n) is admissible and monotonic
then A* is ***optimal***

Underestimates (≤) cost of reaching goal from node

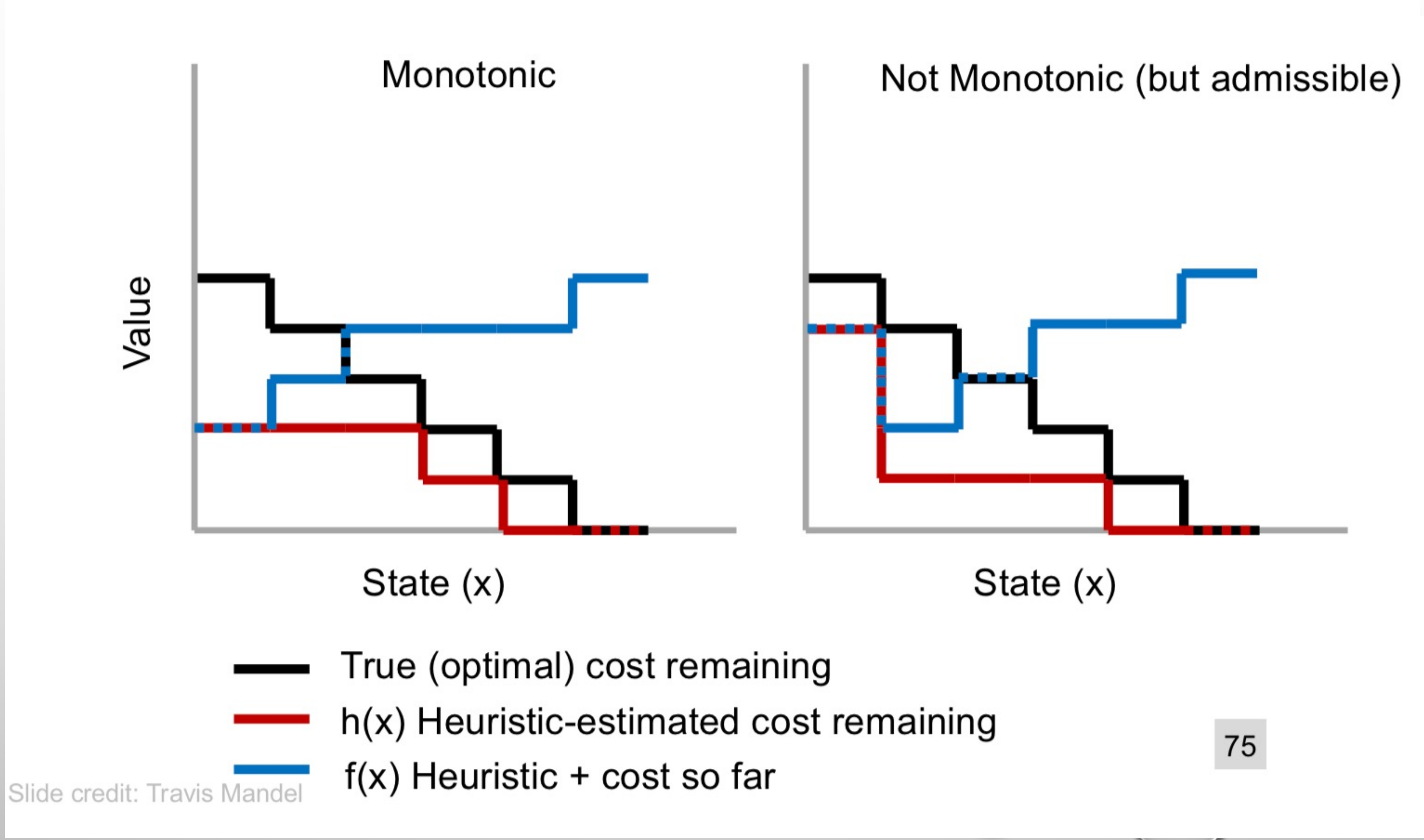
f values never decrease From node to descendants (triangle inequality)

Admissible Heuristics

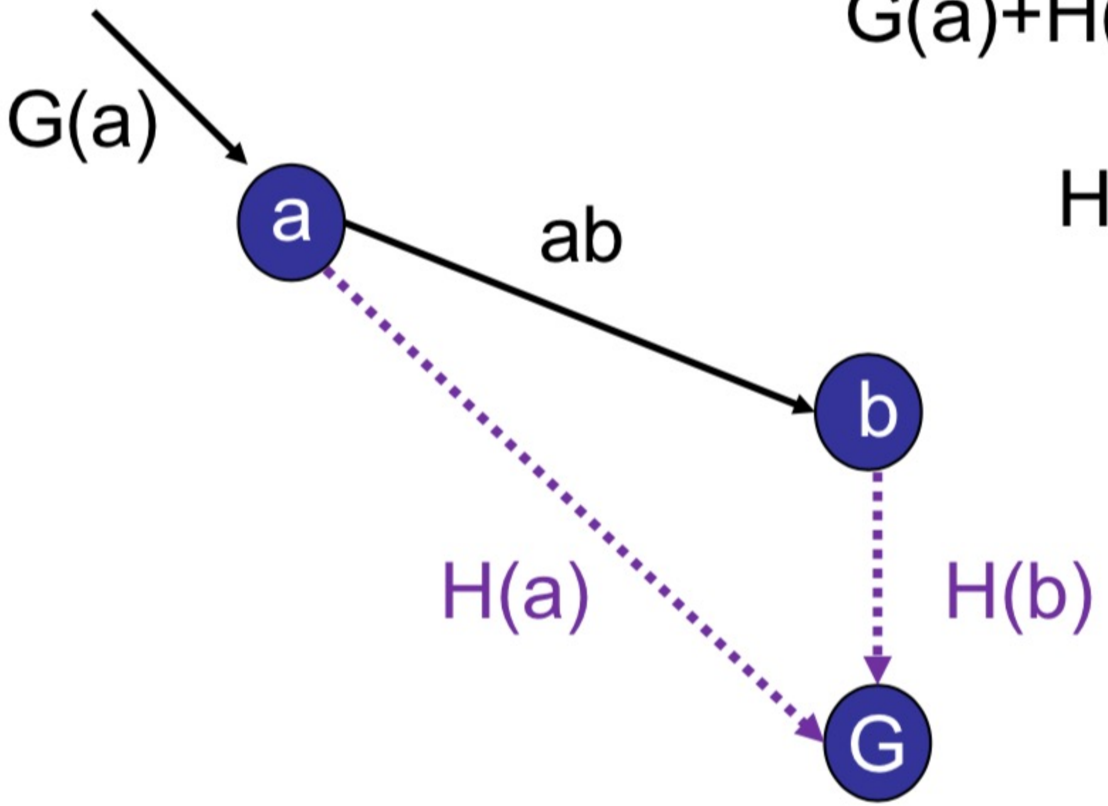


Slide credit: Travis Mandel

Monotonic (or Consistent) Heuristics



Monotonicity (or consistency)



Defn monotonic:

$$F(a) \le F(b)$$

$$G(a)+H(a) \le G(b)+H(b)$$

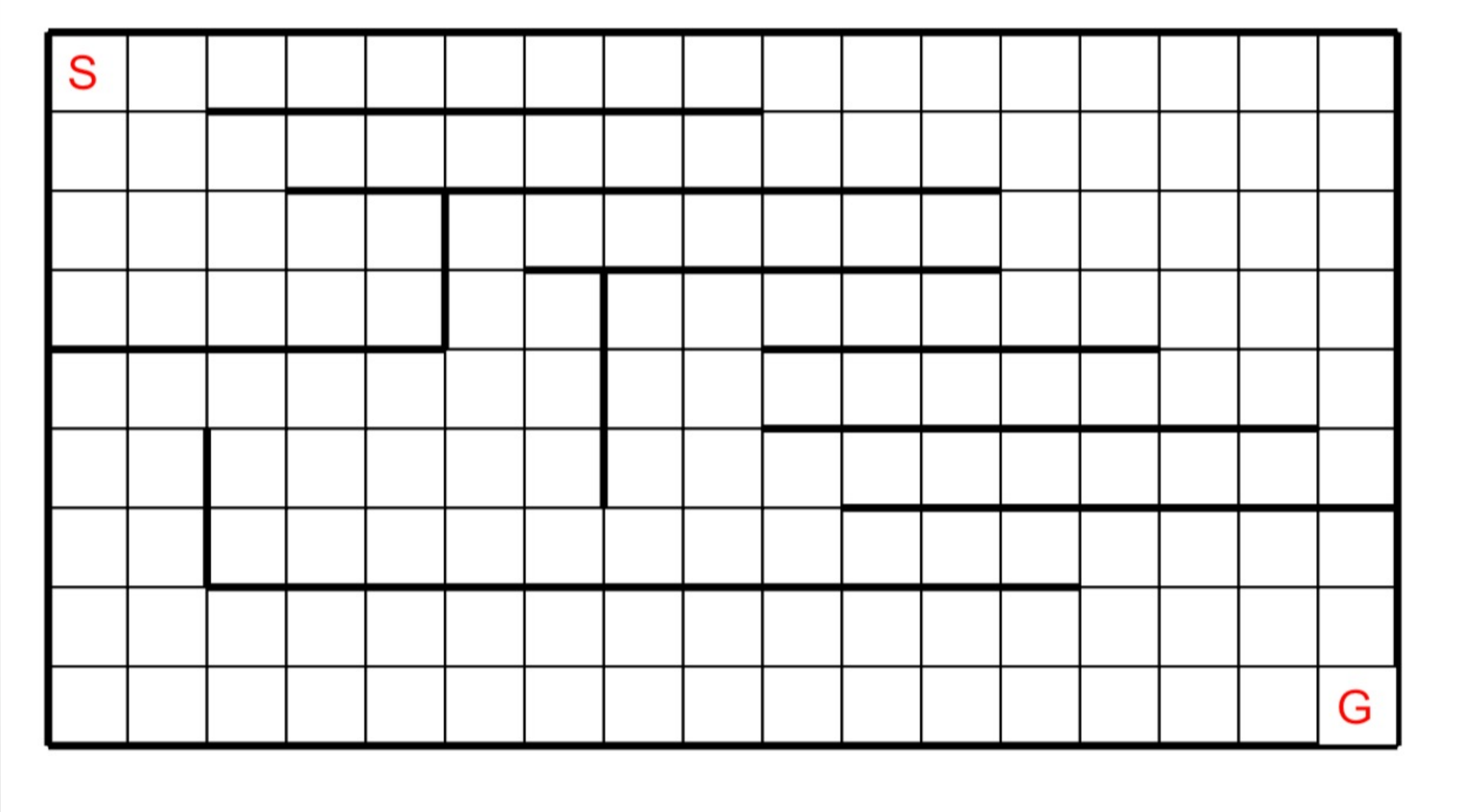
$$\le G(a)+ab + H(b)$$

$$H(a) \le ab + H(b)$$

Example: Maze

- Is Manhattan distance
  - Admissible
  - Monotonic

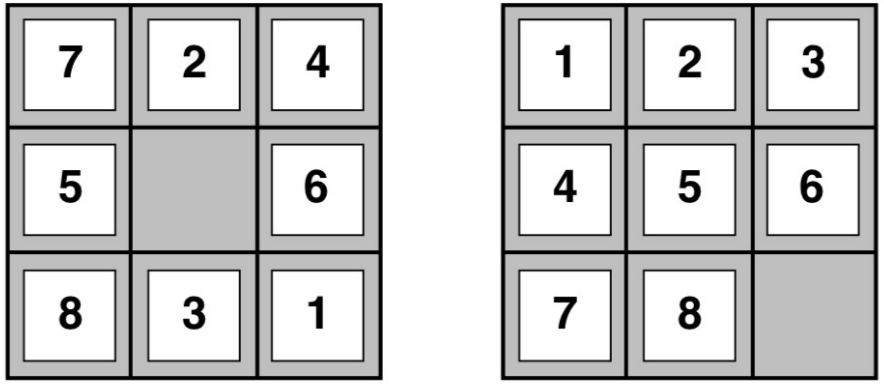
for Maze?



Another example: the 8-puzzle

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**

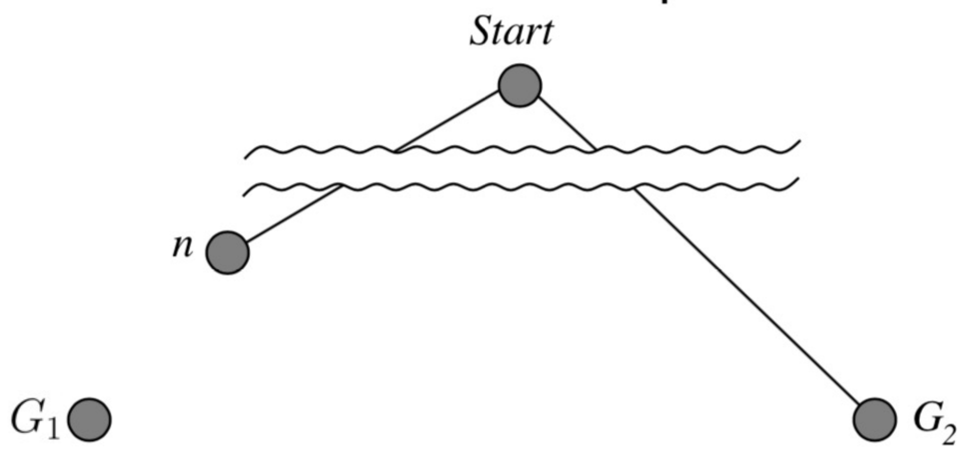
$h_1(S) =??$ 6
$h_2(S) =??$ $4+0+3+3+1+0+2+1 = 14$

Heuristics 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 for n-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
- Given any admissible heuristics h_a, h_b, $h(n) = \max(h_a(n), h_b(n))$ is also admissible and dominates h_a, h_b

Optimality of A* (tree search)

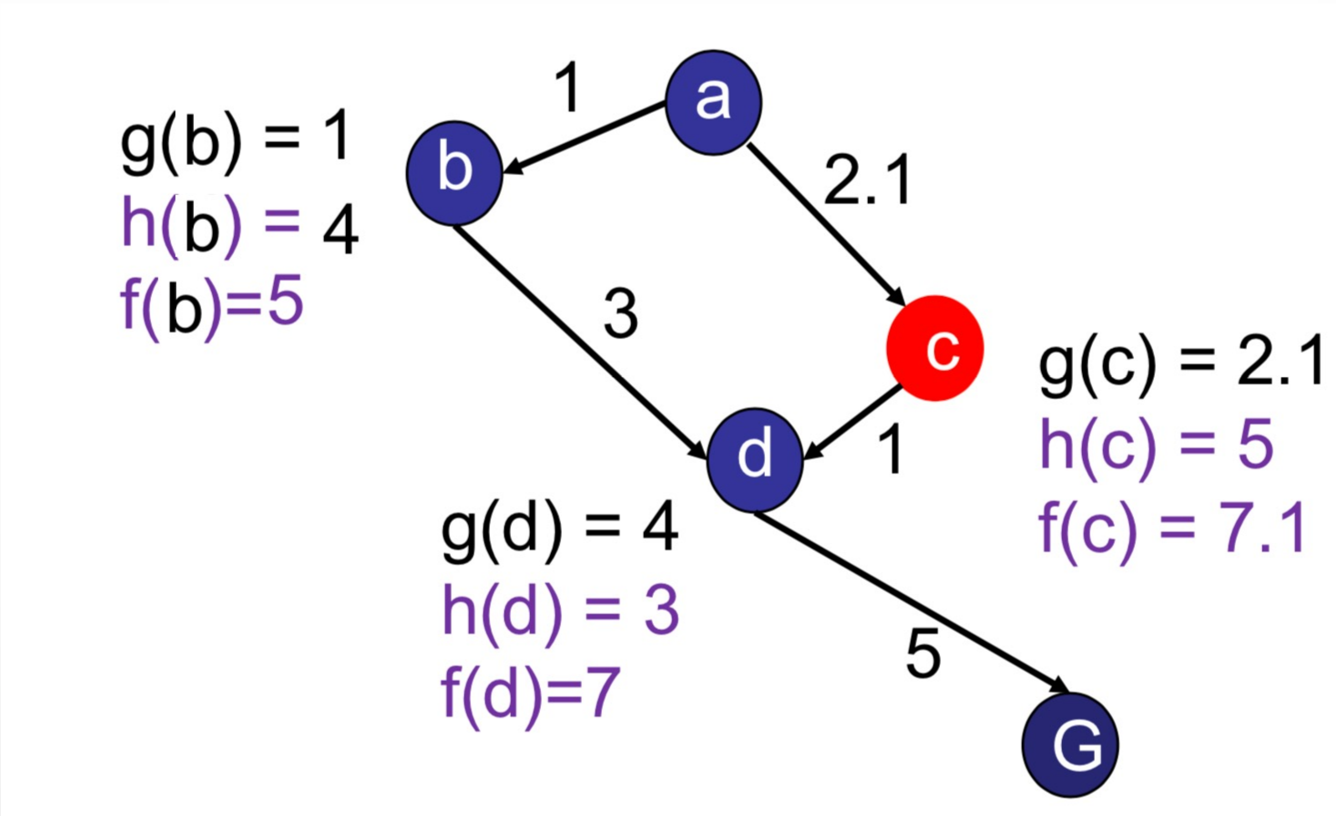
Suppose some suboptimal goal G_2 has been generated and is in the queue. Let n be an unexpanded node on a shortest path to an optimal goal G_1.



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

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

Why monotonicity is required for optimality in the graph search?

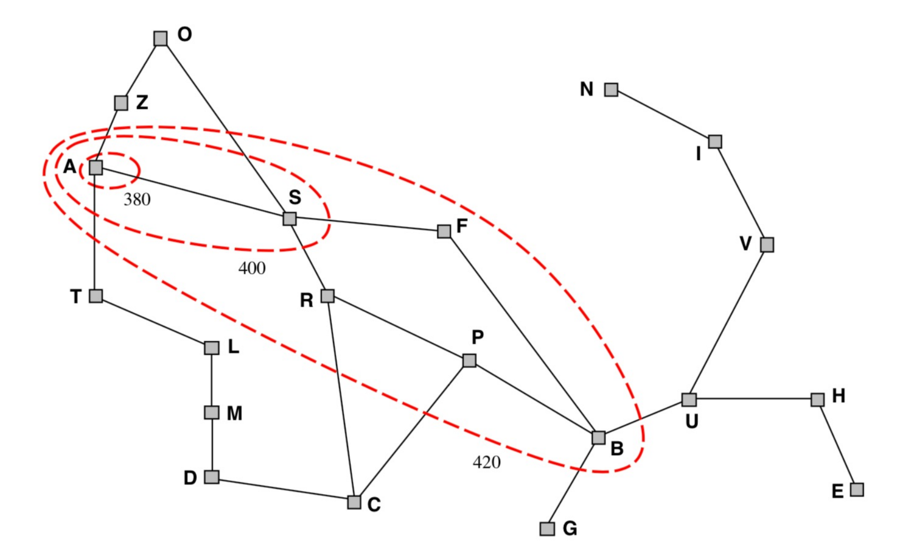


- C will not be expanded. Why?
- How does monotonicity help in avoiding such cases?

Optimality of A* in graph search

- **Lemma 1**: If h(n) is monotonic, then the values of f along any path are non-decreasing.
- **Lemma 2**: Whenever A* selects node n for expansion, the optimal path to that node has been found.
- **Lemma 3**: Optimal goal, G, has the lowest f(G) among all the goals, when selected for expansion.
- **Lemma 4**: A* expands all nodes in order of non-decreasing f value.

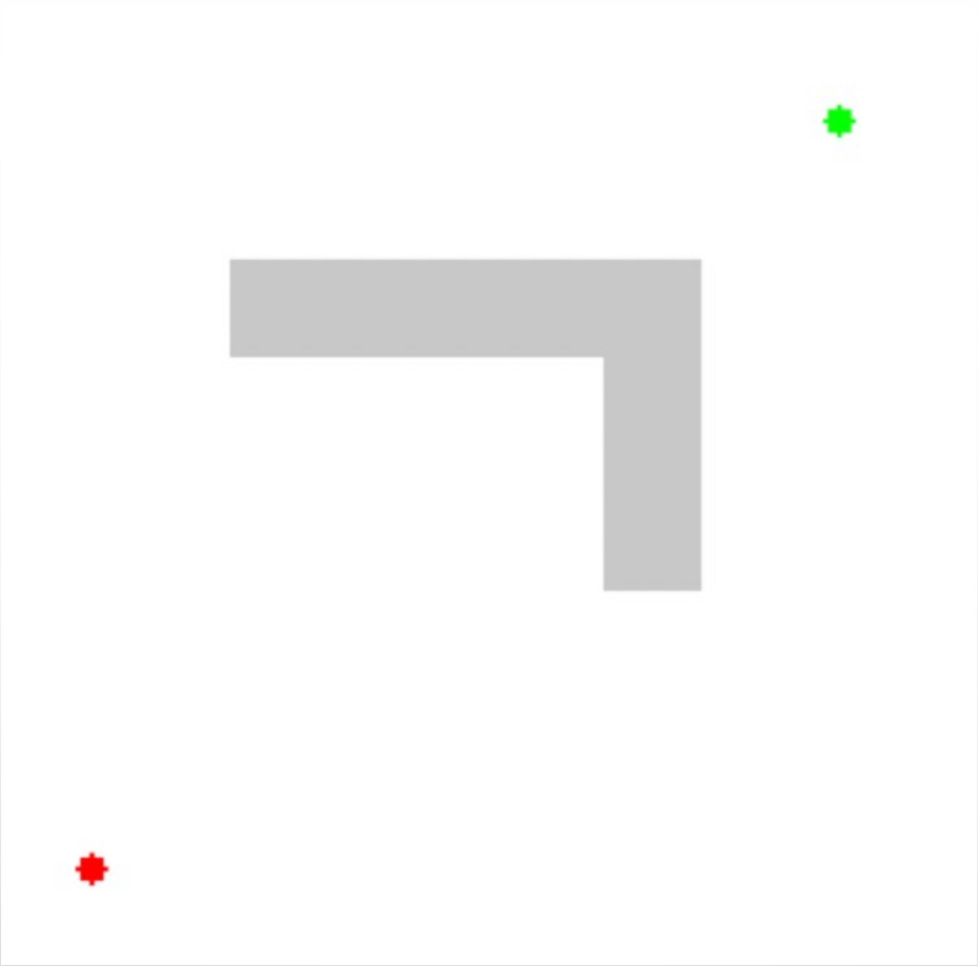
⟹ Optimal goal G will be expanded first among all the goals.



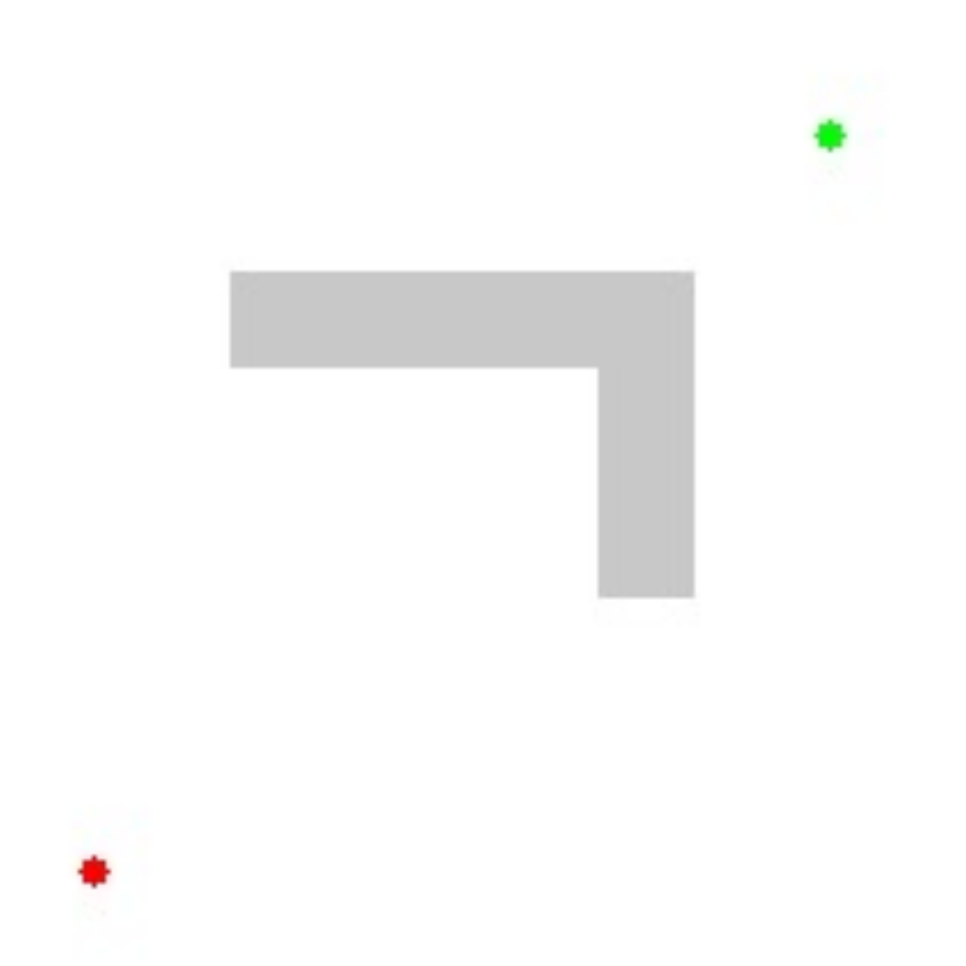
Properties of A∗

- **Complete**:
  - Yes, if there is a lower bound on costs.
- **Time**:
  - For uniform cost, reversible action : exponential in [relative error in h × depth of soln.]
- **Space**:
  - Keeps all nodes in memory
- **Optimal**:
  - Yes (when the mentioned precondition(s) are satisfied).
- A^* expands all nodes with $f(n) < C^*$, some nodes with $f(n) = C^*$, and no nodes with $f(n) > C^*$.

A* demo



A* demo



A* Summary

- **Pros**
  - Produces optimal cost solution!
  - Does so quite quickly (focused)
    - A* is **optimally efficient** for any given heuristics function.
- **Cons**
  - Maintains priority queue…
  - Which can get exponentially big
  - Theorem: Exponential growth will occur unless $|h(n) - h^*(n)| \leq O(\log h^*(n))$.

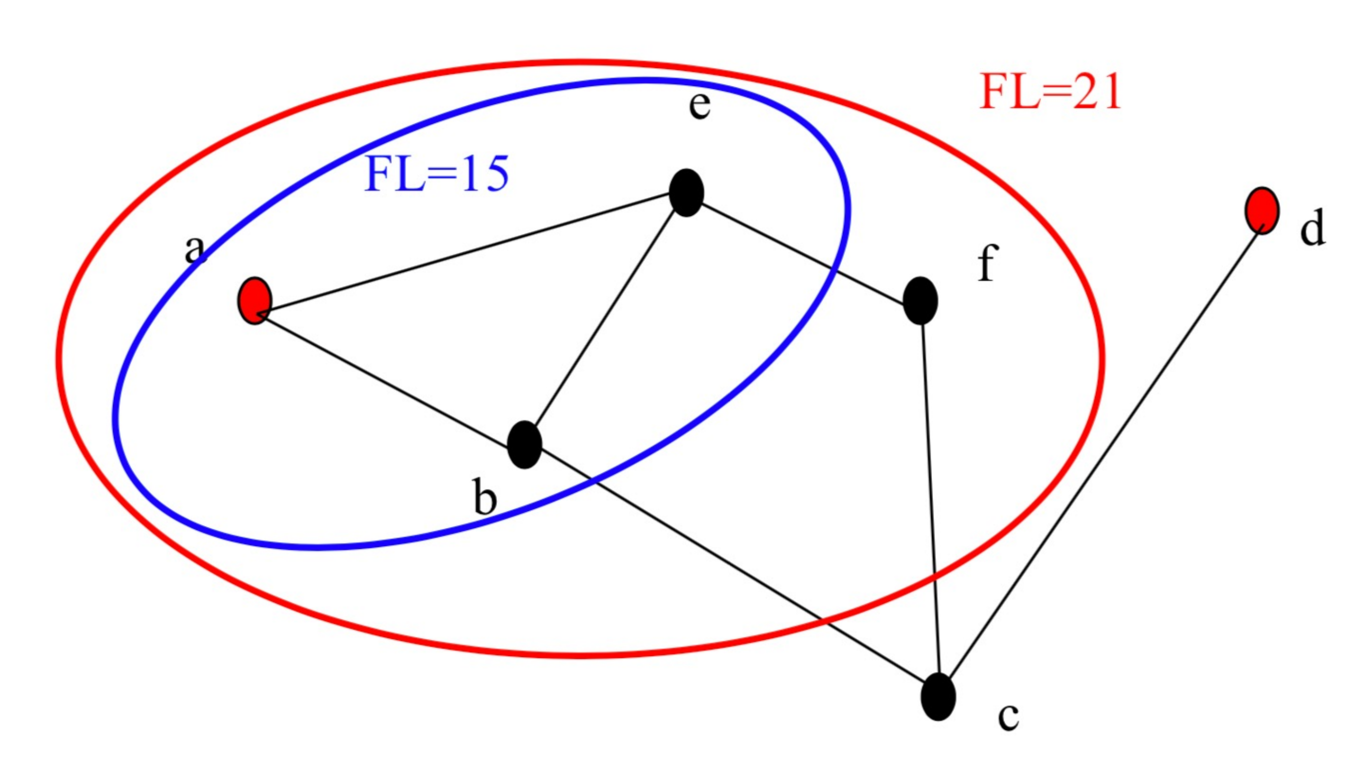
Theorem

A^* is optimally efficient.

- Let f^* be the cost of the shortest path to a goal. Consider any algorithm A' which has the same start node as A^*, uses the same heuristic and fails to expand some path p' expanded by A^* for which $cost(p') + h(p') < f^*$. Assume that A' is optimal.
- Consider a different search problem which is identical to the original and on which h returns the same estimate for each path, except that p' has a child path p'' which is a goal node, and the true cost of the path to p'' is $f(p')$.
  - that is, the edge from p' to p'' has a cost of $h(p')$: the heuristic is exactly right about the cost of getting from p' to a goal.
- A' would behave identically on this new problem.
  - The only difference between the new problem and the original problem is beyond path p', which A' does not expand.
- Cost of the path to p'' is lower than cost of the path found by A'.
- This violates our assumption that A' is optimal.

Iterative-Deepening A*

- Like iterative-deepening depth-first, but...
- Depth bound modified to be an f-limit
  - Start with f-limit = h(start)
  - Perform depth-first search (using stack, no queue)
  - Prune any node if f(node) > f-limit
  - Next f-limit = min-cost of any node pruned

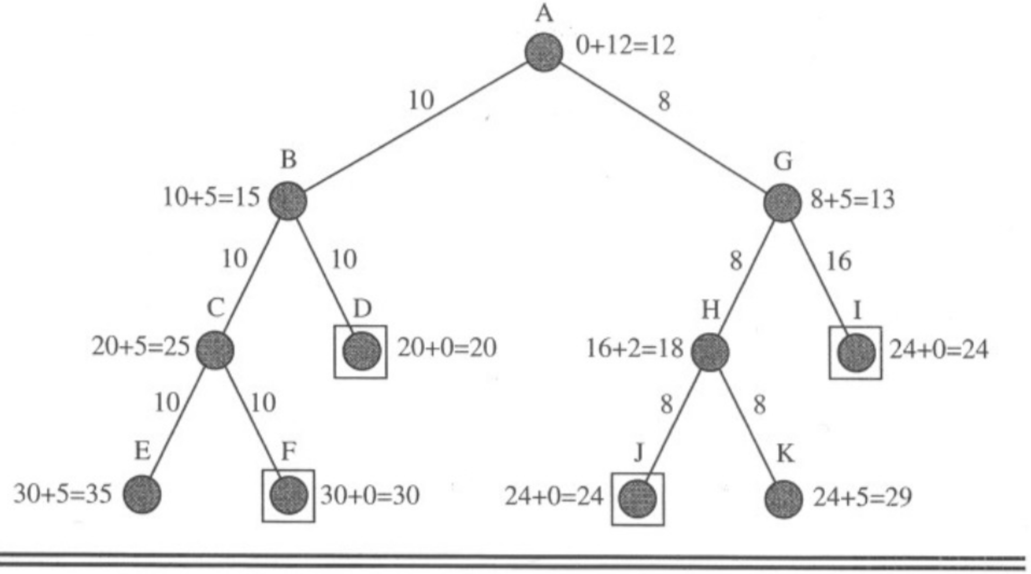


IDA* Analysis

- Complete & Optimal (like A*)
  - Space usage ∝ depth of solution
  - Each iteration is DFS - no priority queue!
- nodes expanded relative to A* ?
  - Depends on # unique values of heuristic function
  - In traveling salesman: each f value is unique $\Longrightarrow$ $1+2+\ldots+n = O(n^2)$ where n = nodes A* expands
    - if n is too big for main memory, n^2 is too long to wait!
  - In 8 puzzle: few values $\Longrightarrow$ close to # A* expands

Forgetfulness

- A* used exponential memory
- Simplified memory-bounded A* : SMA*
  - Store all expanded (unlike A*) and open nodes in the memory.
  - If memory is full,
    - deletes the leaf with highest f value and backs up the value in its parent.



1) f of the nodes get updated, once all the children of the node are opened.
2) If a goal state is opened and no node or backed up node has a lower f value, the algorithm would terminate.

1. At each stage, one successor is added to the deepest lowest-f-cost node that has some successors not currently in the tree. The left child B is added to the root A.
2. Now $f(A)$ is still 12, so we add the right child G ($f = 13$). Now that we have seen all the children of A, we can update its f-cost to the minimum of its children, that is, 13. The memory is now full.
3. G is now designated for expansion, but we must first drop a node to make room. We drop the shallowest highest-f-cost leaf, that is, B. When we have done this, we note that A's best forgotten descendant has $f = 15$, as shown in parentheses. We then add H, with $f(H) = 18$. Unfortunately, H is not a goal node, but the path to H uses up all the available memory. Hence, there is no way to find a solution through H, so we set $f(H) = \infty$.
4. G is expanded again. We drop H, and add I, with $f(I) = 24$. Now we have seen both successors of G, with values of ∞ and 24, so $f(G)$ becomes 24. $f(A)$ becomes 15, the minimum of 15 (forgotten successor value) and 24. Notice that I is a goal node, but it might not be the best solution because A's f-cost is only 15.
5. A is once again the most promising node, so B is generated for the second time. We have found that the path through G was not so great after all.
6. C, the first successor of B, is a nongoal node at the maximum depth, so $f(C) = \infty$.
7. To look at the second successor, D, we first drop C. Then $f(D) = 20$, and this value is inherited by B and A.
8. Now the deepest, lowest-f-cost node is D. D is therefore selected, and because it is a goal node, the search terminates.

Demo: Different search methods

http://qiao.github.io/PathFinding.js/visual/