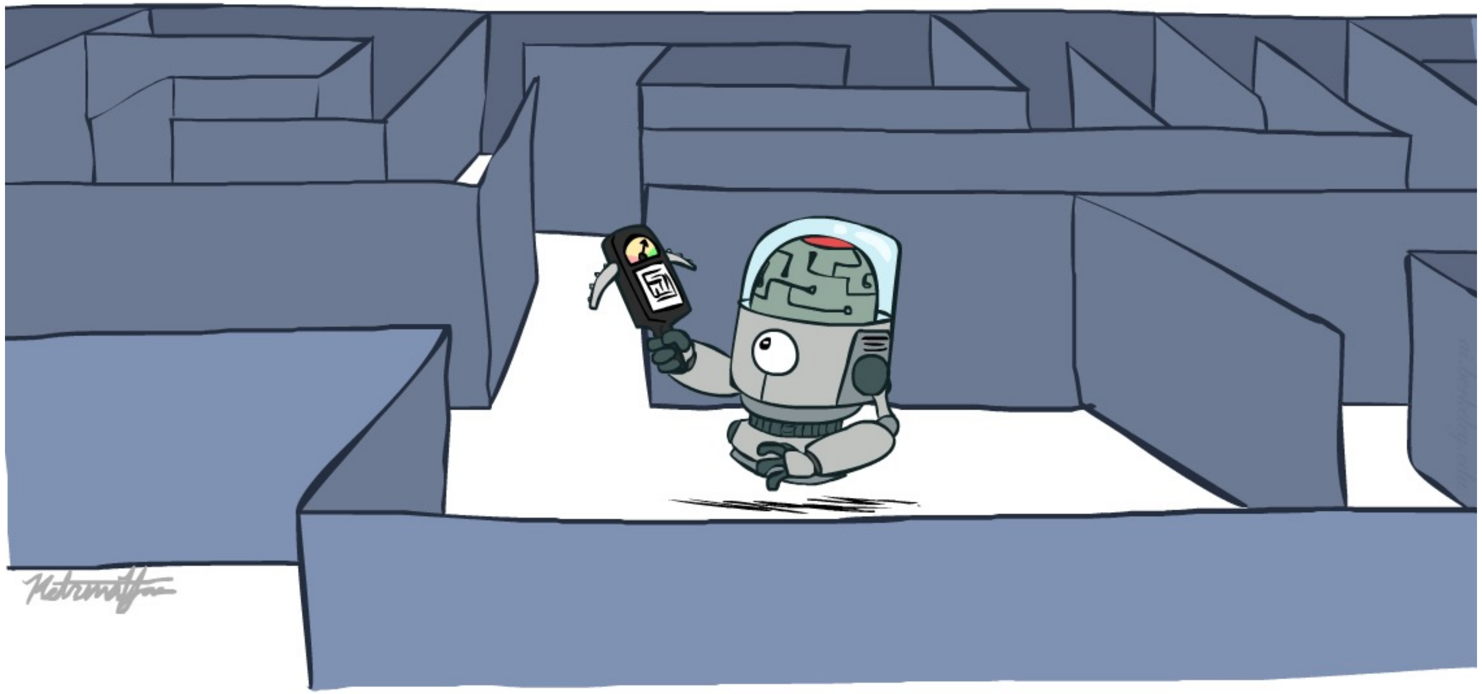
Introduction to Artificial Intelligence

Informed Search



Instructor: Laura Nenzi

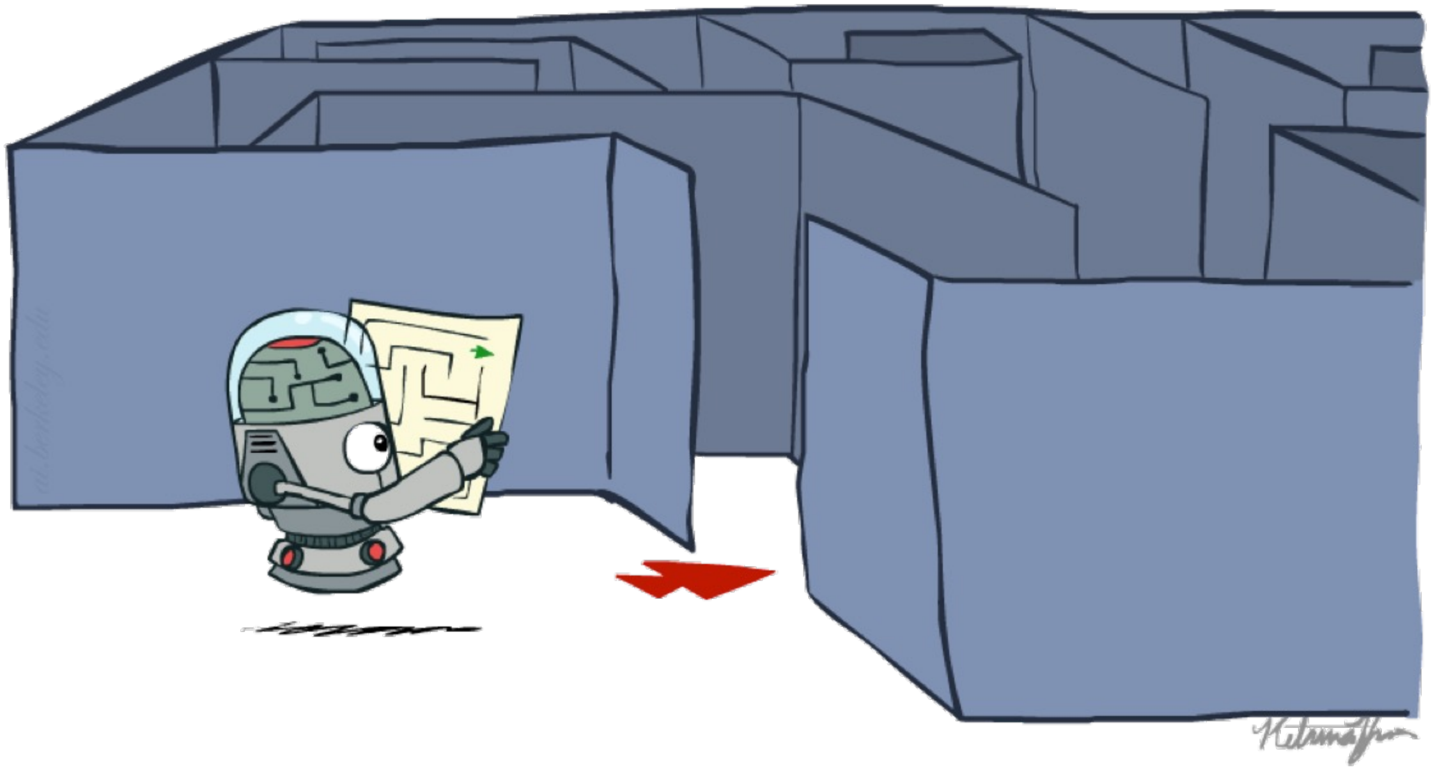
University of Trieste, Italy

[slides adapted from Dan Klein, Pieter Abbeel, Stuart Russell, et al for CS188 Intro to AI at UC Berkeley. All materials available at http://ai.berkeley.edu.]

Today

- Creating Heurustic
- Graph Search

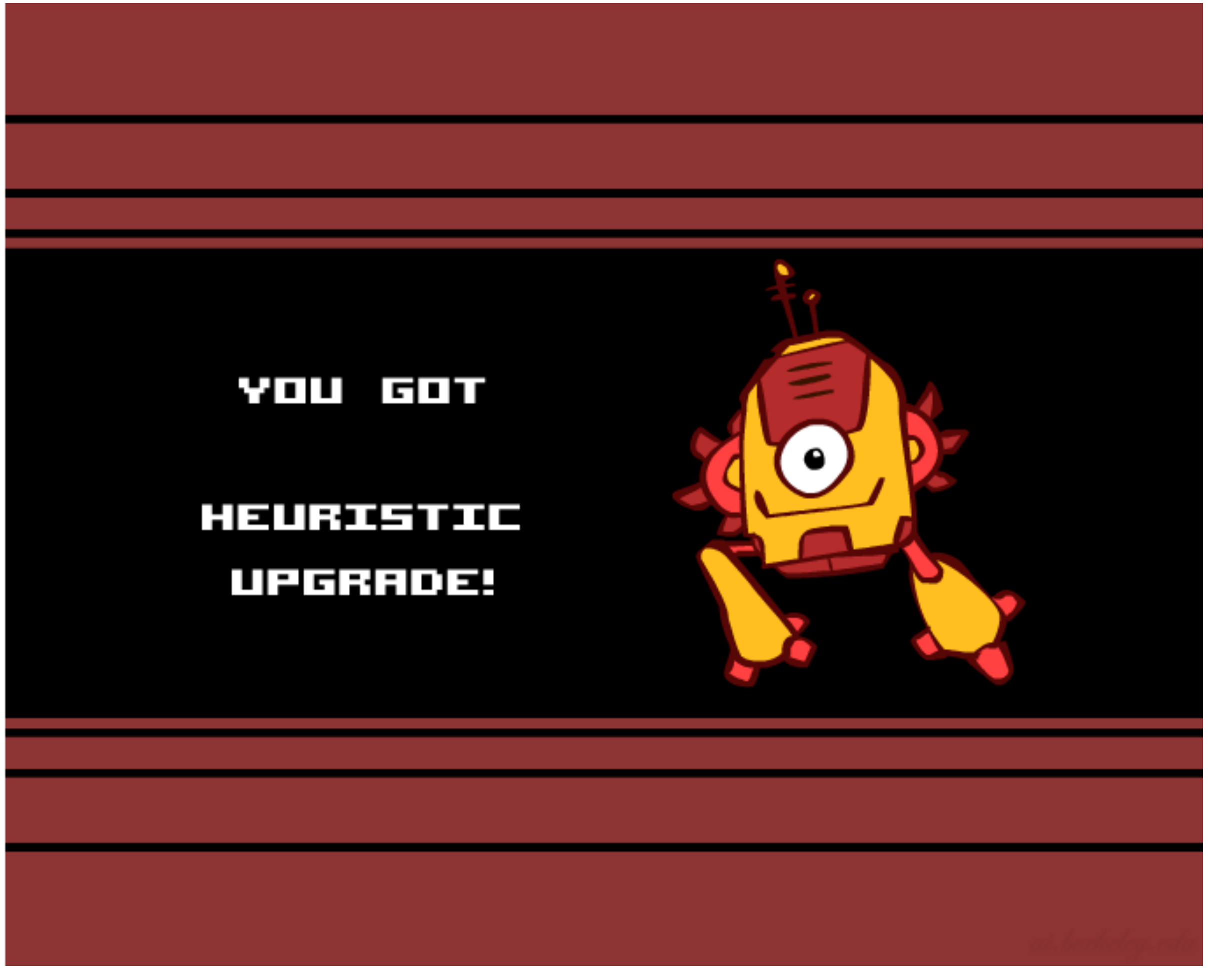
Recap: Search



Recap

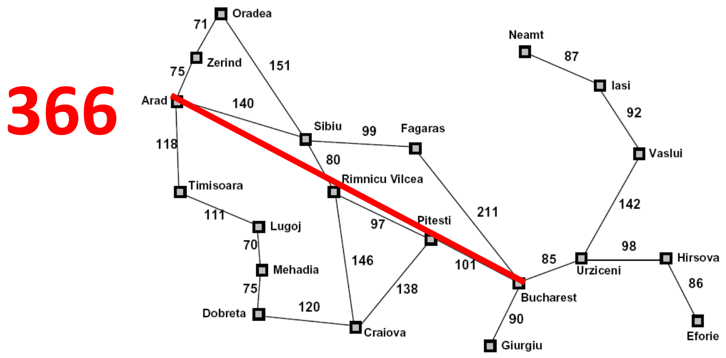
- A* expands the fringe node with lowest f value where
  - $f(n) = g(n) + h(n)$
  - $g(n)$ is the cost to reach n
  - $h(n)$ is an admissible estimate of the least cost from n to a goal node: $0 \leq h(n) \leq h^*(n)$
- A* tree search is optimal
- Its performance depends heavily on the heuristic h

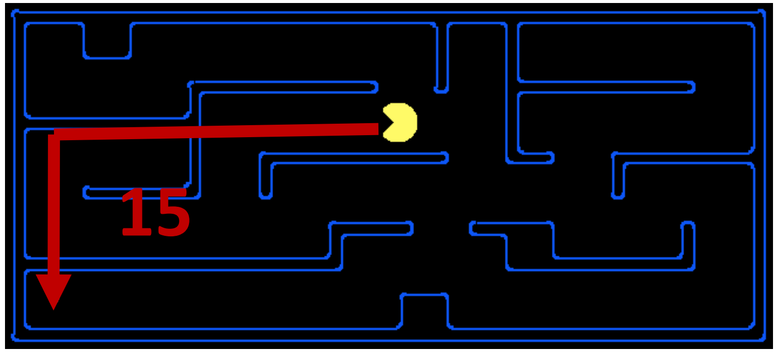
Creating Heuristics



Creating Admissible Heuristics

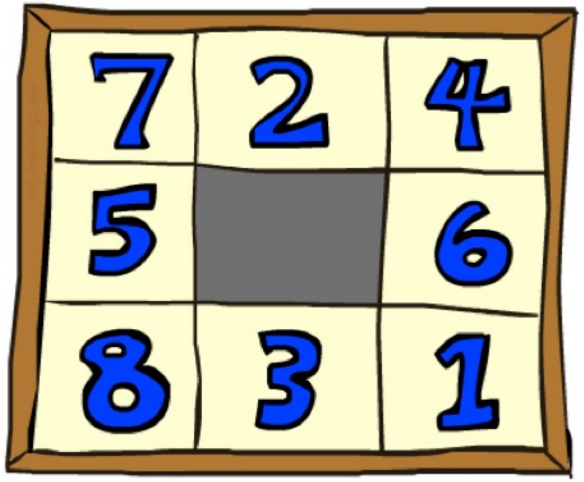
- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
- Often, admissible heuristics are solutions to ***relaxed problems***, where new actions are available



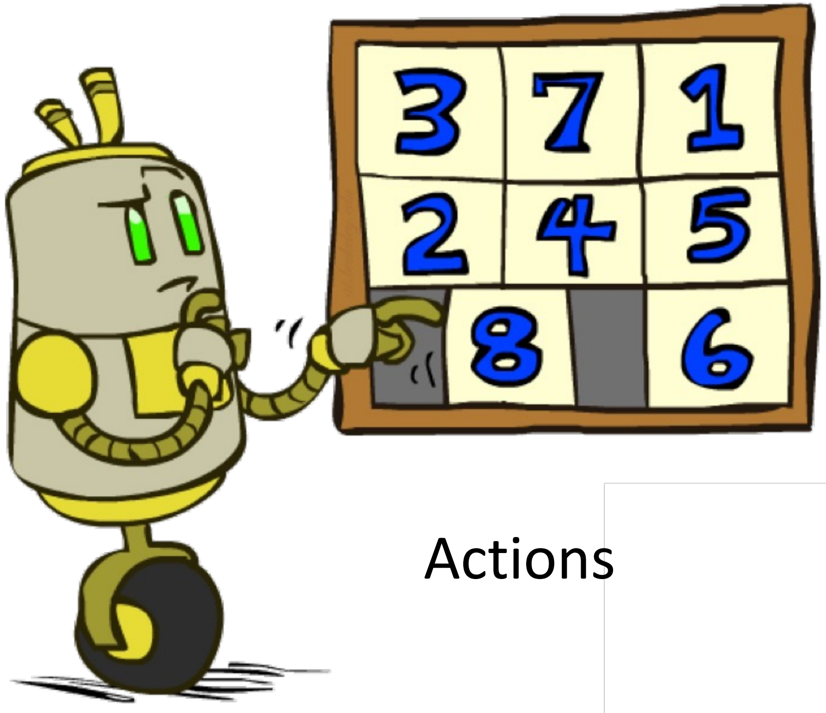


- Problem P_2 *is a relaxed version of* P_1 *if* $\mathcal{A}_2(s) \supseteq \mathcal{A}_1(s)$ *for every s*
- *Theorem:* $h_2^*(s) \leq h_1^*(s), \forall\, s$, *so* $h_2^*(s)$ *is admissible for* P_1

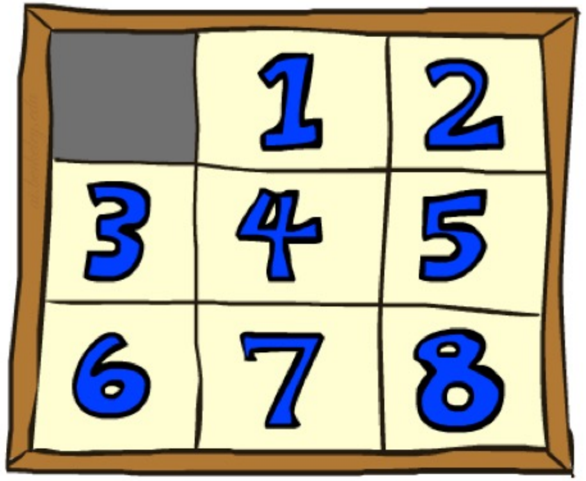
Example: 8 Puzzle



Start State



Actions

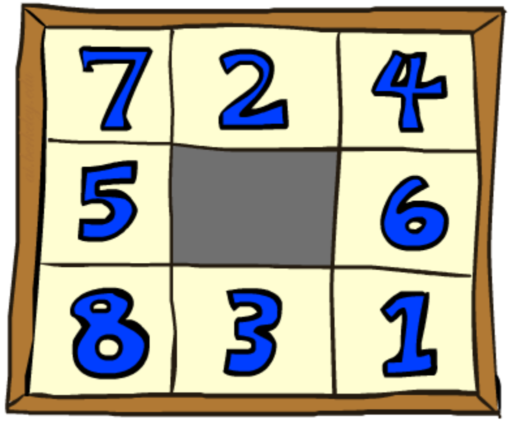


Goal State

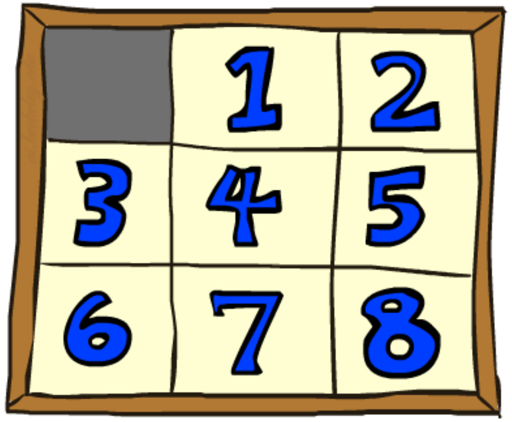
- What are the states?
- How many states?
- What are the actions?
- What should the costs be?

8 Puzzle I

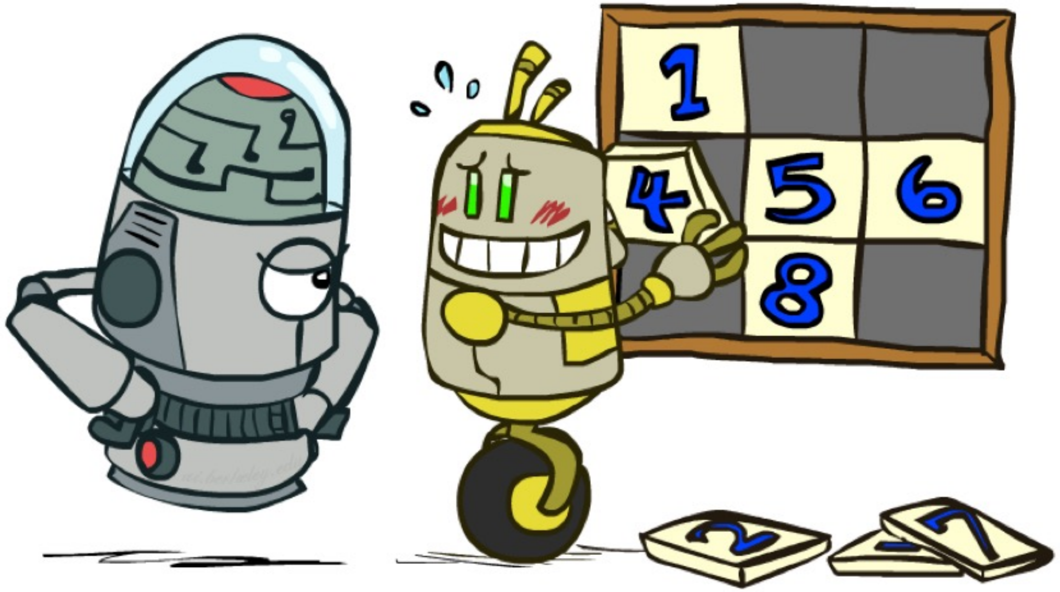
- Heuristic: Number of tiles misplaced
- Would it be admissible?
- h(start) = 8
- This is a *relaxed-problem* heuristic



Start State



Goal State

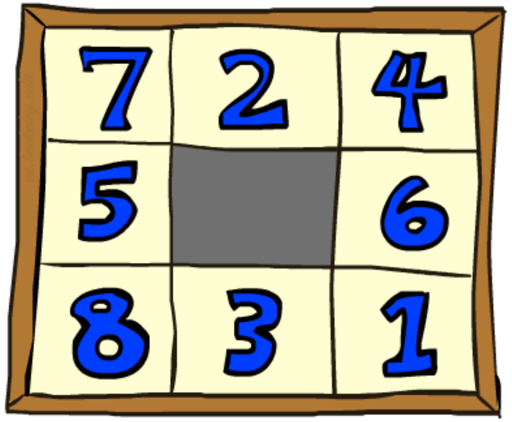


	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

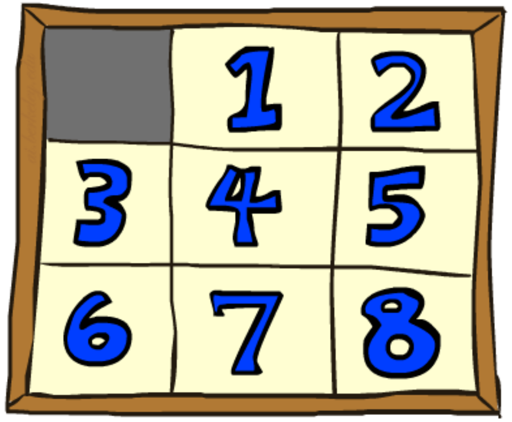
Statistics from Andrew Moore

8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
- Total *Manhattan* distance
- Would it be admissible?
- h(start) = 3 + 1 + 2 + … = 18



Start State

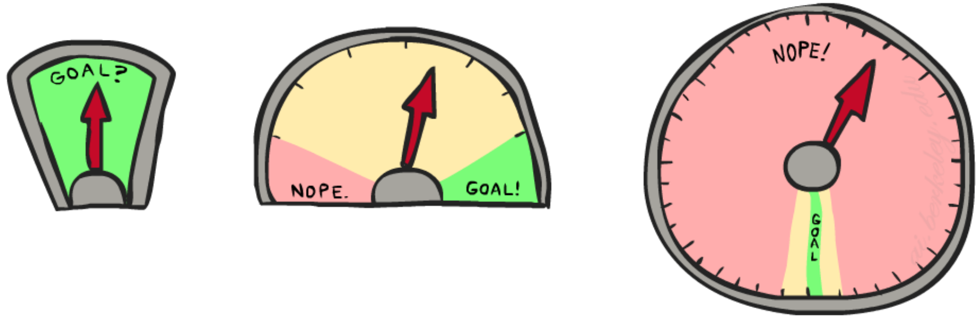


Goal State

	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
TILES	13	39	227
MANHATTAN	12	25	73

8 Puzzle III

- How about using the *actual cost* as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?



- With A*: a trade-off between quality of estimate and work per node
  - As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

Dominance, Trivial Heuristics

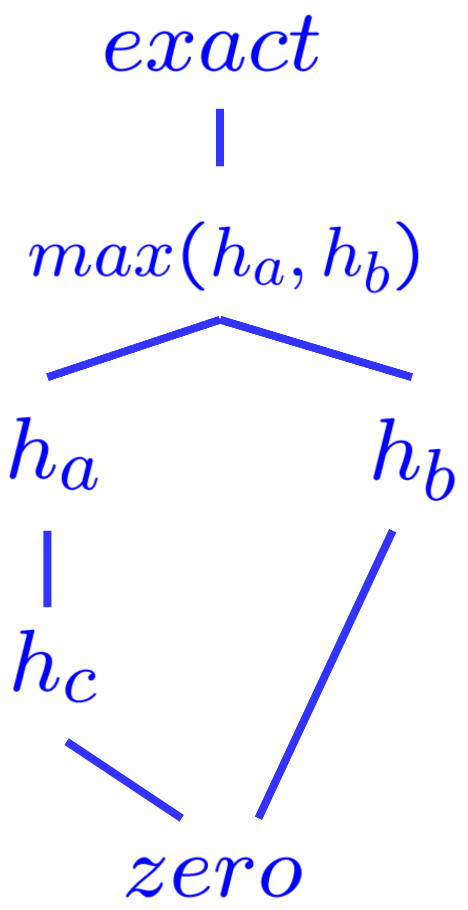
- Dominance: $h_a \geq h_c$ if

$$\forall n : h_a(n) \geq h_c(n)$$

- Max of admissible heuristics is admissible

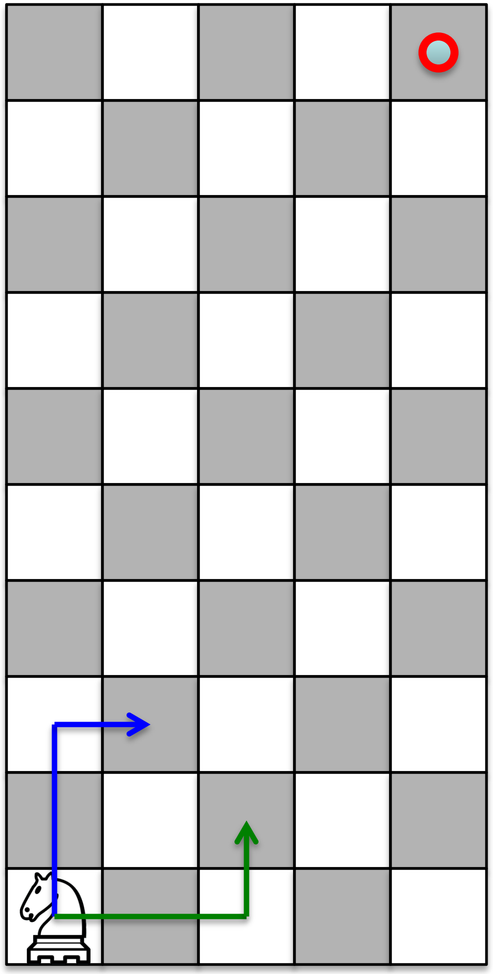
$$h(n) = max(h_a(n), h_b(n))$$

- Trivial heuristics
  - The zero heuristic: the smallest admissible heuiristic
  - The exact heuristic: the larger admissible heuristic

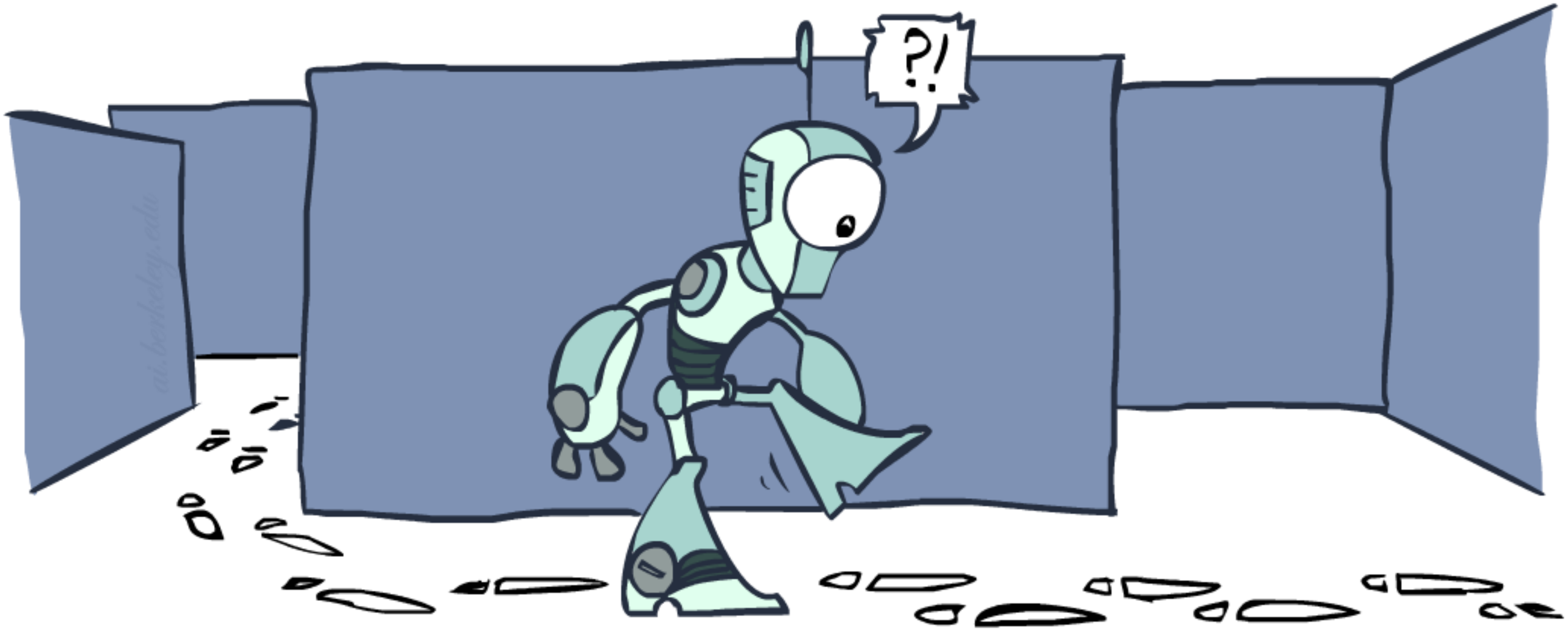


Example: Knight's moves

- Minimum number of knight's moves to get from A to B?
  - h_1 = (Manhattan distance)/3
    - h_1' = h_1 rounded up to correct parity (even if A, B same color, odd otherwise)
  - h_2 = (Euclidean distance)/√5 (rounded up to correct parity)
  - h_3 = (max x or y shift)/2 (rounded up to correct parity)
- $h(n) = \max(h_1'(n), h_2(n), h_3(n))$ is admissible!

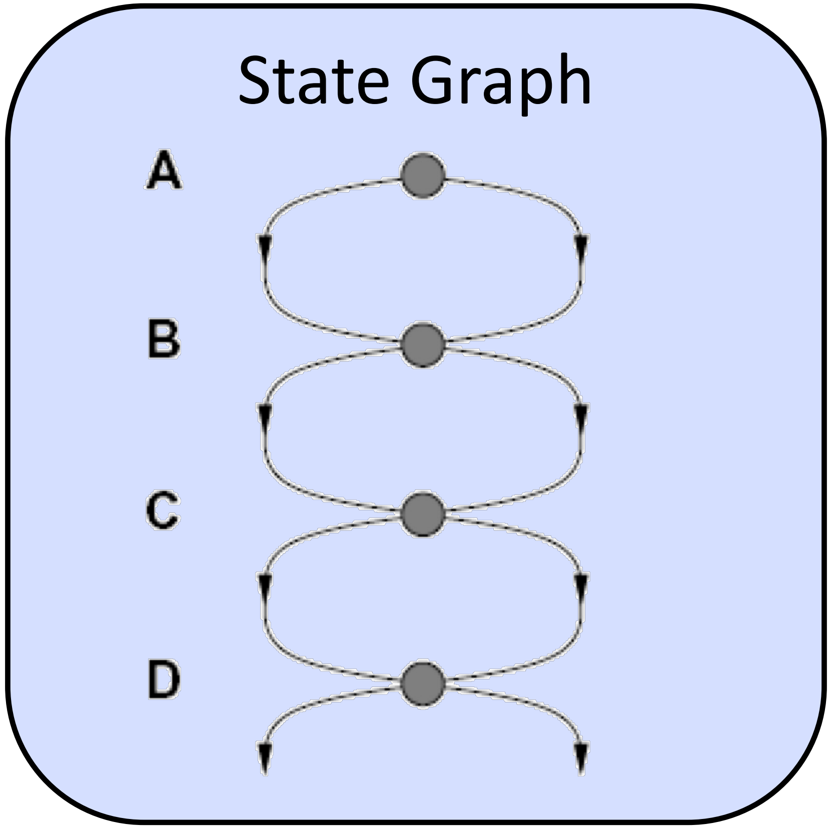


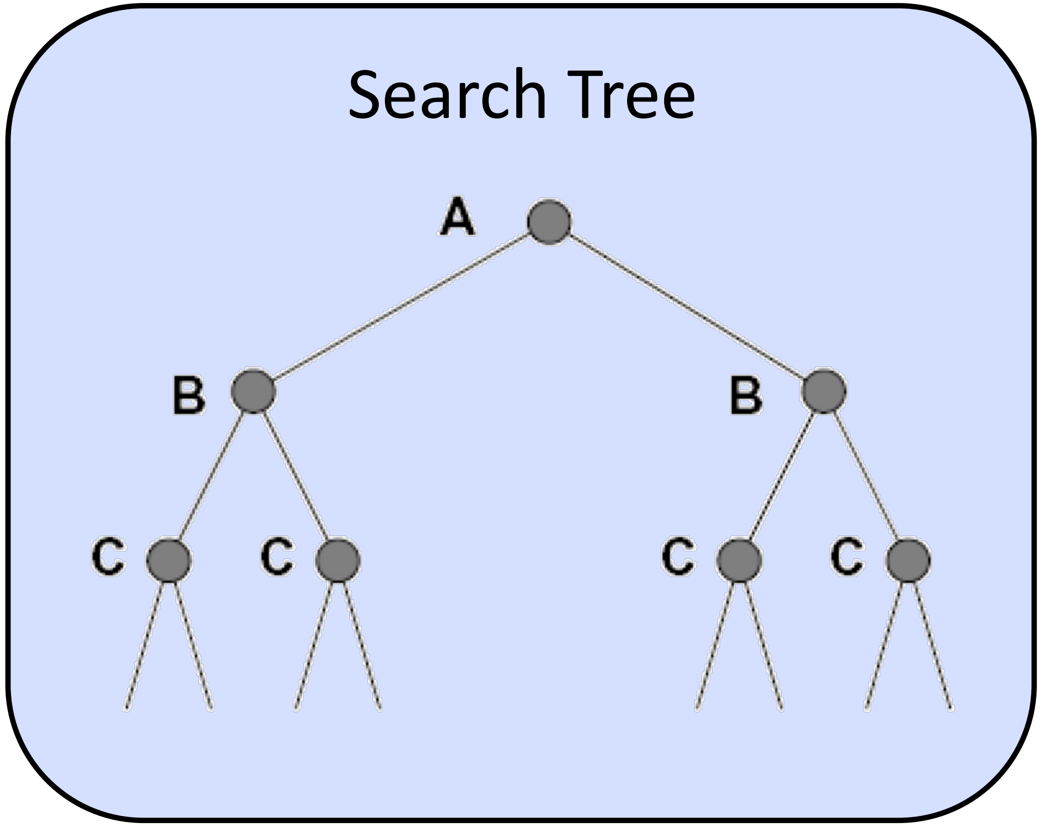
Graph Search



Tree Search: Extra Work!

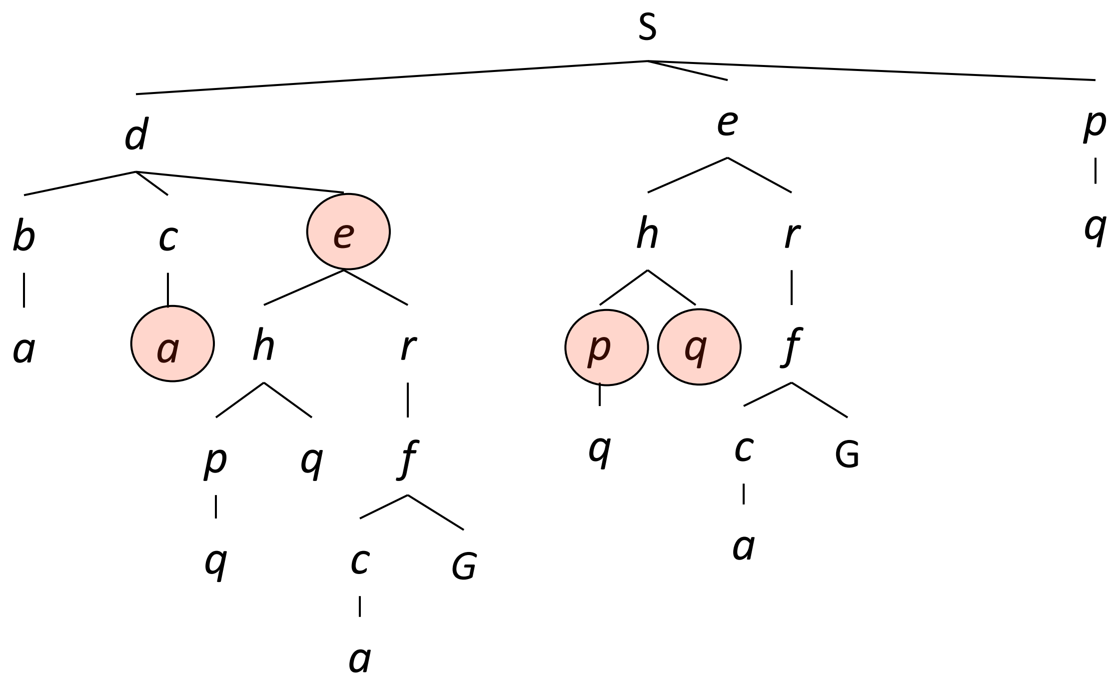
- Failure to detect repeated states can cause exponentially more work.





Graph Search

- In BFS, for example, we shouldn't bother expanding the circled nodes (why?)

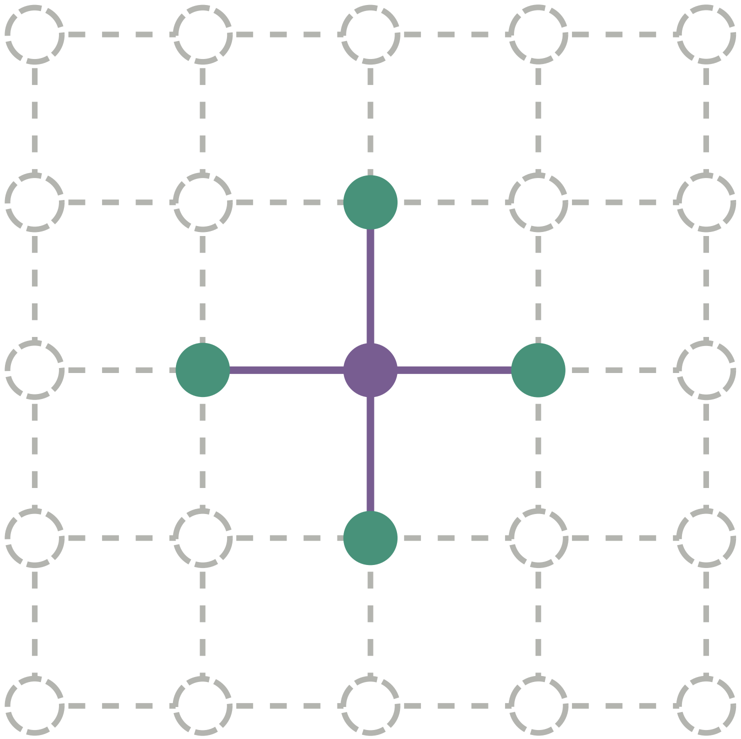


Graph Search

- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states (“closed set”)
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to closed set
- Important: store the closed set as a set, not a list
- Can graph search wreck completeness? Why/why not?
- How about optimality?

Quiz: State Space Graphs vs. Search Trees

Consider a rectangular grid:



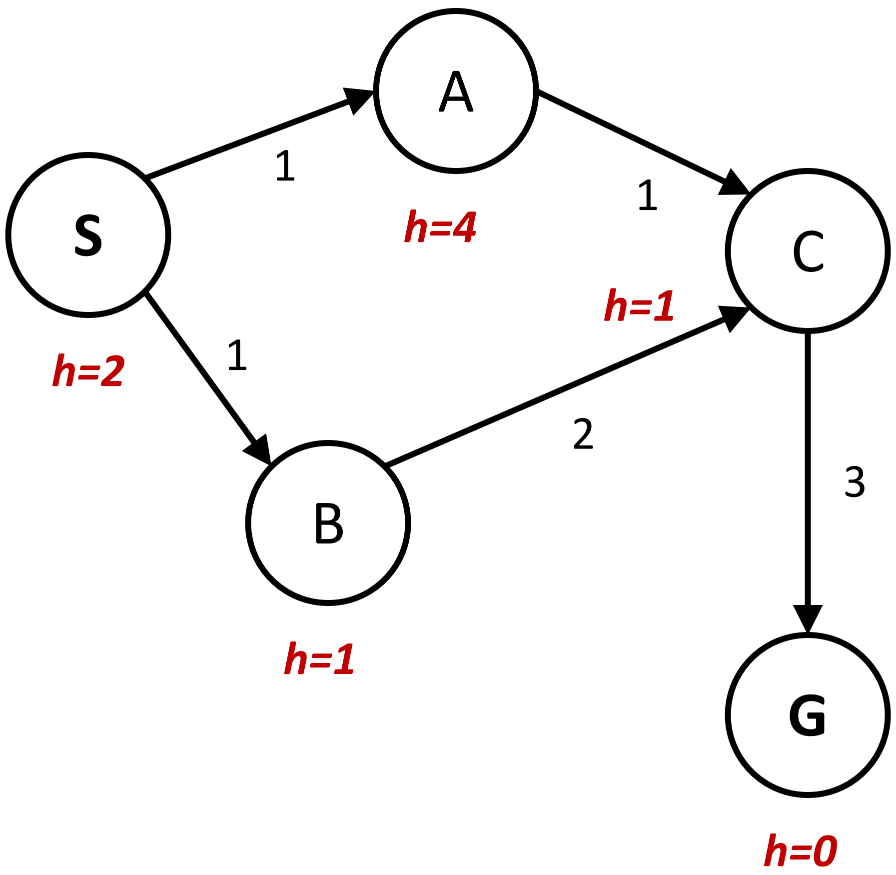
How many states within *d* steps of start?

How many states in search tree of depth *d*?

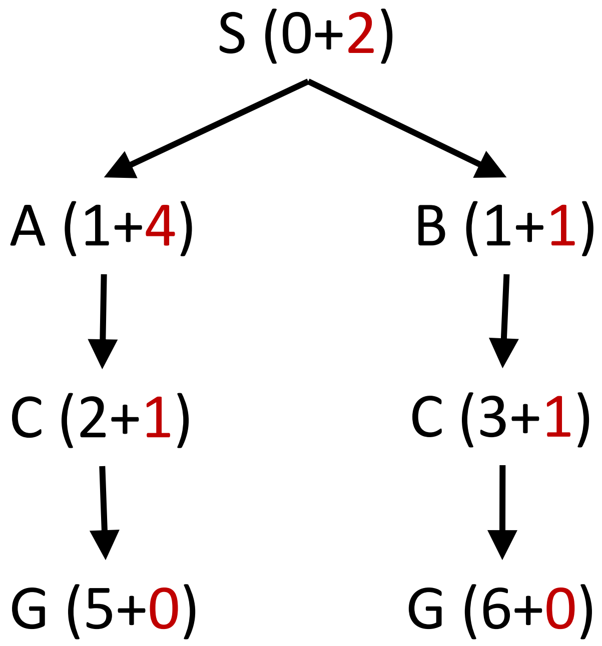
Basic idea of graph search: don't re-expand a state that has been expanded previously

A* Graph Search Gone Wrong?

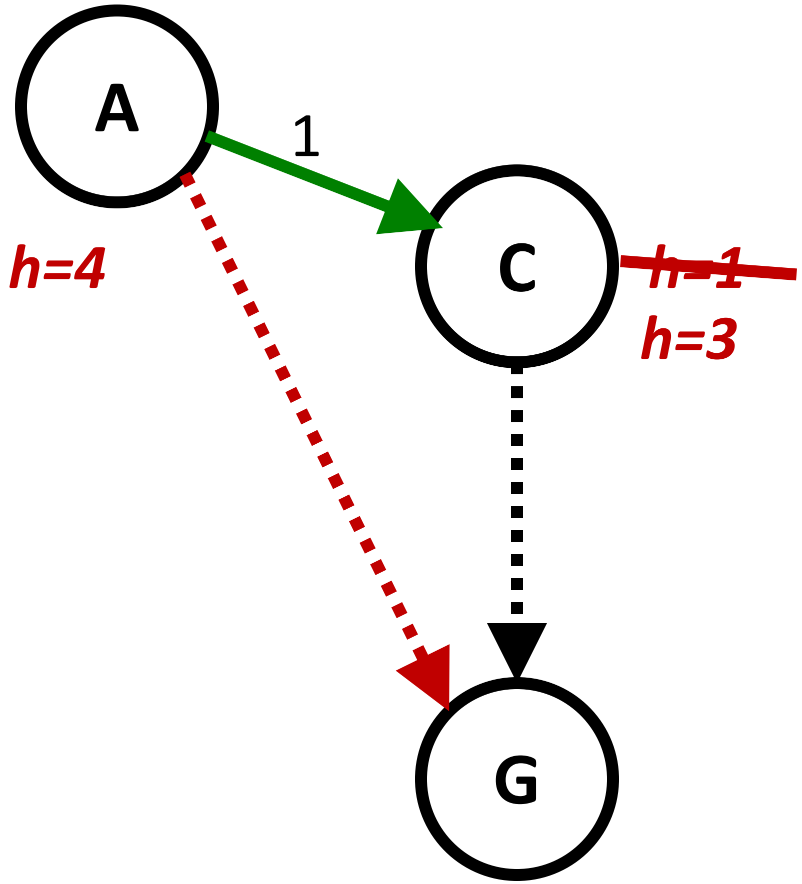
State space graph



Search tree



Consistency of Heuristics



- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal

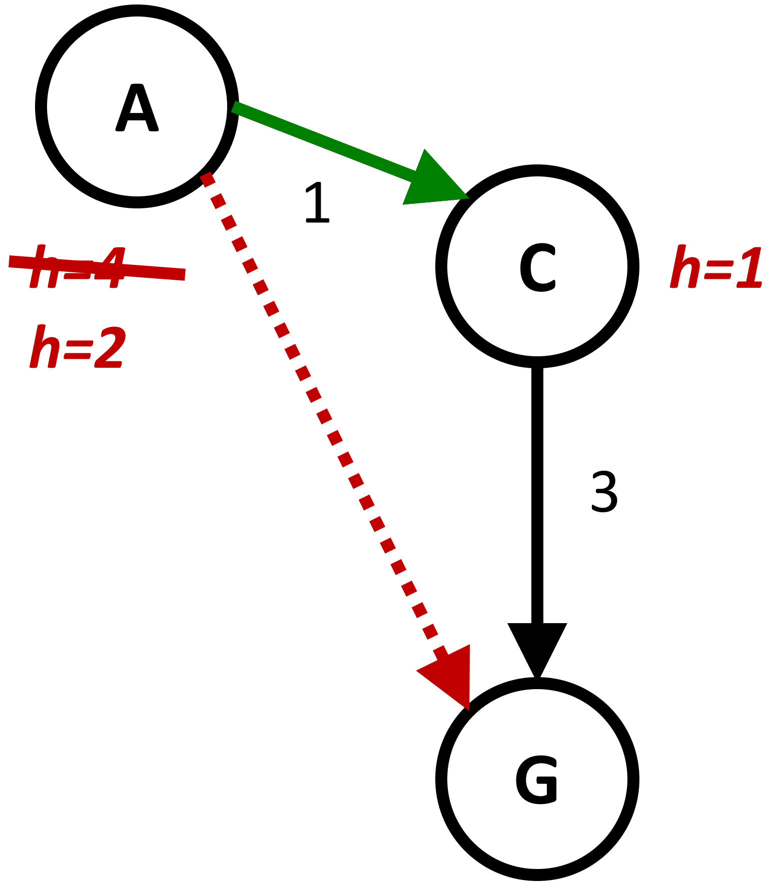
    h(A) ≤ h^*(A)
  - Consistency: heuristic "arc" cost ≤ actual cost for each arc

    h(A) – h(C) ≤ c(A,C)

    or h(A) ≤ c(A,C) + h(C) (triangle inequality)
    - Note: h* *necessarily* satisfies triangle inequality
- Consequences of consistency:
  - The *f* value along a path never decreases:

    h(A) ≤ c(A,C) + h(C) => g(A) + h(A) ≤ g(A) + c(A,C) + h(C)
  - A* graph search is optimal

Consistency of Heuristics



- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal

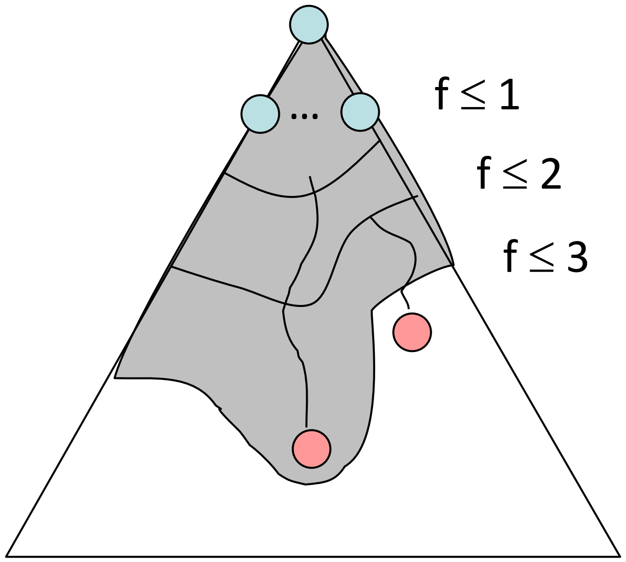
    h(A) ≤ actual cost from A to G
  - Consistency: heuristic “arc” cost ≤ actual cost for each arc

    h(A) – h(C) ≤ cost(A to C)
- Consequences of consistency:
  - The f value along a path never decreases

    h(A) ≤ cost(A to C) + h(C)
  - A* graph search is optimal

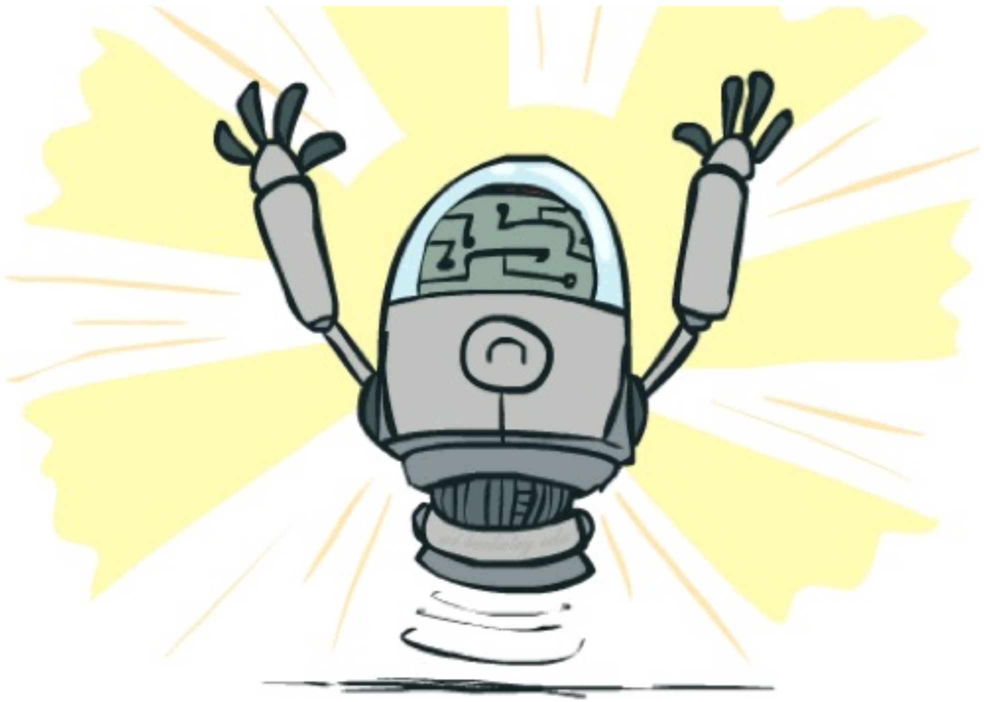
Optimality of A* Graph Search

- Sketch: consider what A* does with a consistent heuristic:
  - Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)
  - Fact 2: For every state s, nodes that reach s optimally are expanded before nodes that reach s suboptimally
  - Result: A* graph search is optimal



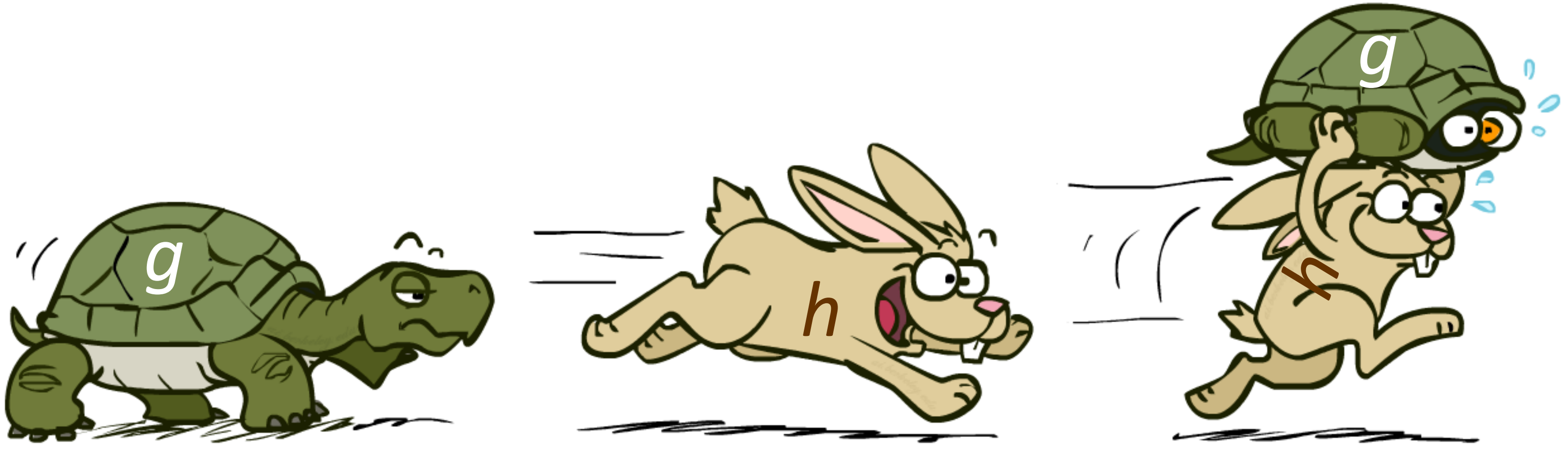
Optimality

- Tree search:
  - A* is optimal if heuristic is admissible
  - UCS is a special case (h = 0)
- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal (h = 0 is consistent)
- Consistency implies admissibility
- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems



A*: Summary

- A* orders nodes in the queue by $f(n) = g(n) + h(n)$
- A* is optimal for trees/graphs with admissible/consistent heuristics
- Heuristic design is key: often use relaxed problems



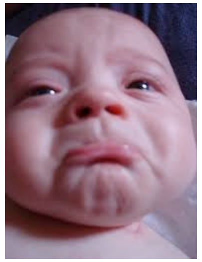
Tree Search Pseudo-Code

```
function TREE-SEARCH(problem, fringe) return 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, STATE[node]) then return node
        for child-node in EXPAND(STATE[node], problem) do
            fringe ← INSERT(child-node, fringe)
        end
    end
```

Graph Search Pseudo-Code

```
function GRAPH-SEARCH(problem, fringe) return 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
         for child-node in EXPAND(STATE[node], problem) do
            fringe ← INSERT(child-node, fringe)
         end
   end
```

But…



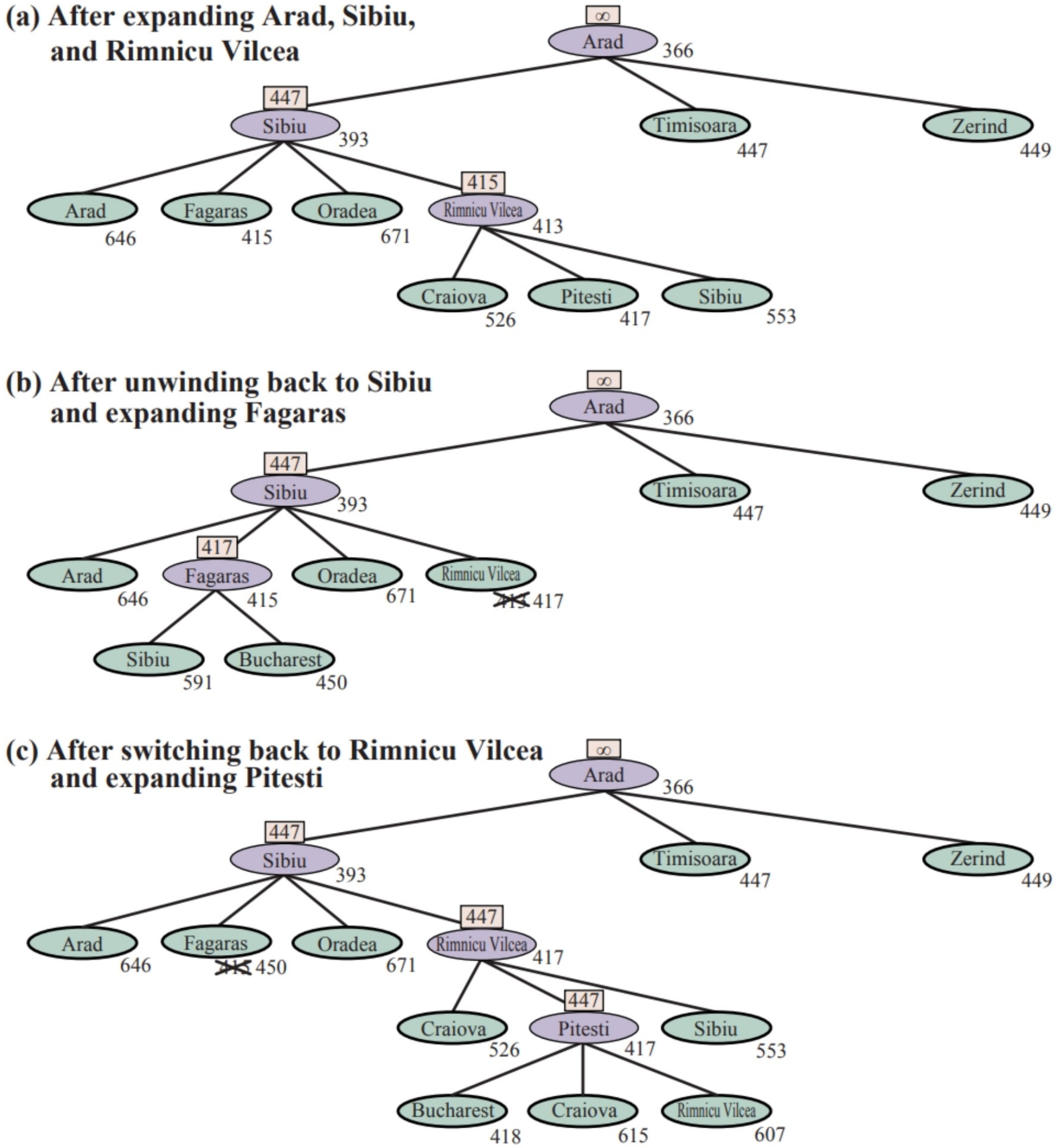
- A* keeps the entire explored region in memory
- => will run out of space before you get bored waiting for the answer
- There are variants that use less memory (Section 3.5.5):

Iterative-deepening A* search (IDA*)

- IDA* works like iterative deepening, except it uses an f-limit instead of a depth limit
  - The the cutoff is the f-cost (g+h);
  - On each iteration, remember the smallest f-value that exceeds the current limit, use as new limit
  - When each path’s f-cost is an integer, this works very well, resulting in steady progress towards the goal each iteration
  - Very inefficient when f is real-valued and each node has a unique value

Recursive best-first search (RBFS)

- RBFS is a recursive depth-first search that uses an f-limit = the f-value of the best alternative path available from any ancestor of the current node
  - When the limit is exceeded, the recursion unwinds back to the alternative path
  - But it also remember the best reachable f-value on that branch, **backed-up value**
  - It can therefore decide whether it’s worth reexpanding the subtree at some later time
  - More efficient than IDA∗, but still suffers from excessive node re-generation.


(a) After expanding Arad, Sibiu, and Rimnicu Vilcea
∞
Arad
366
447
Sibiu
393
Timisoara
447
Zerind
449
Arad
646
Fagaras
415
Oradea
671
415
Rimnicu Vilcea
413
Craiova
526
Pitesti
417
Sibiu
553
(b) After unwinding back to Sibiu and expanding Fagaras
∞
Arad
366
447
Sibiu
393
Timisoara
447
Zerind
449
Arad
646
417
Fagaras
415
Oradea
671
Rimnicu Vilcea
413 417
Sibiu
591
Bucharest
450
(c) After switching back to Rimnicu Vilcea and expanding Pitesti
∞
Arad
366
447
Sibiu
393
Timisoara
447
Zerind
449
Arad
646
Fagaras
415 450
Oradea
671
447
Rimnicu Vilcea
417
Craiova
526
447
Pitesti
417
Sibiu
553
Bucharest
418
Craiova
615
Rimnicu Vilcea
607

Simplified memory-bounded (SMA*).

- SMA* uses all available memory for the queue, minimizing thrashing
  - When full, drop worst node on the queue but remember its value in the parent
  - It regenerates the subtree only when all other paths have been shown to look worse than the path it has forgotten.