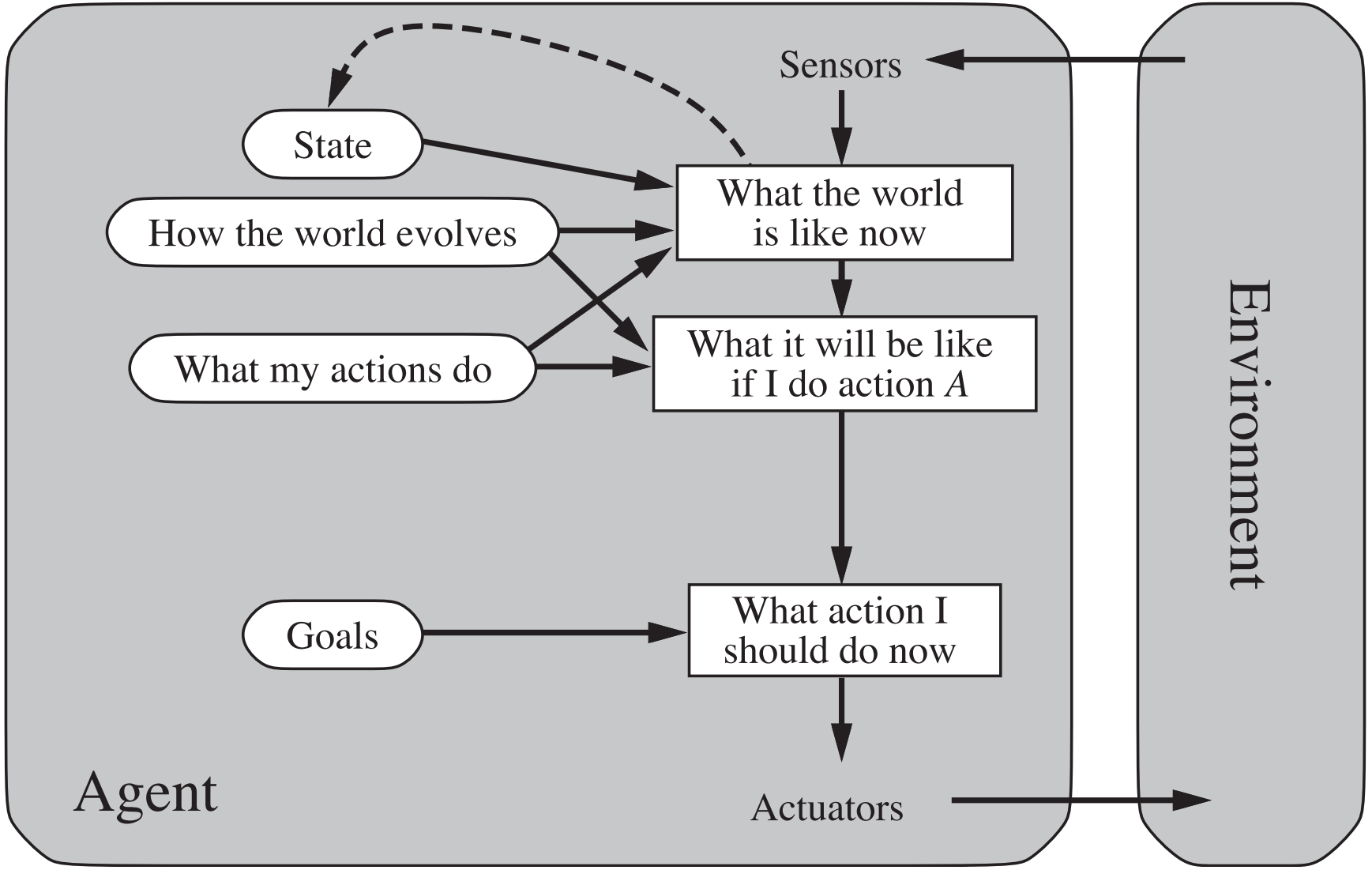
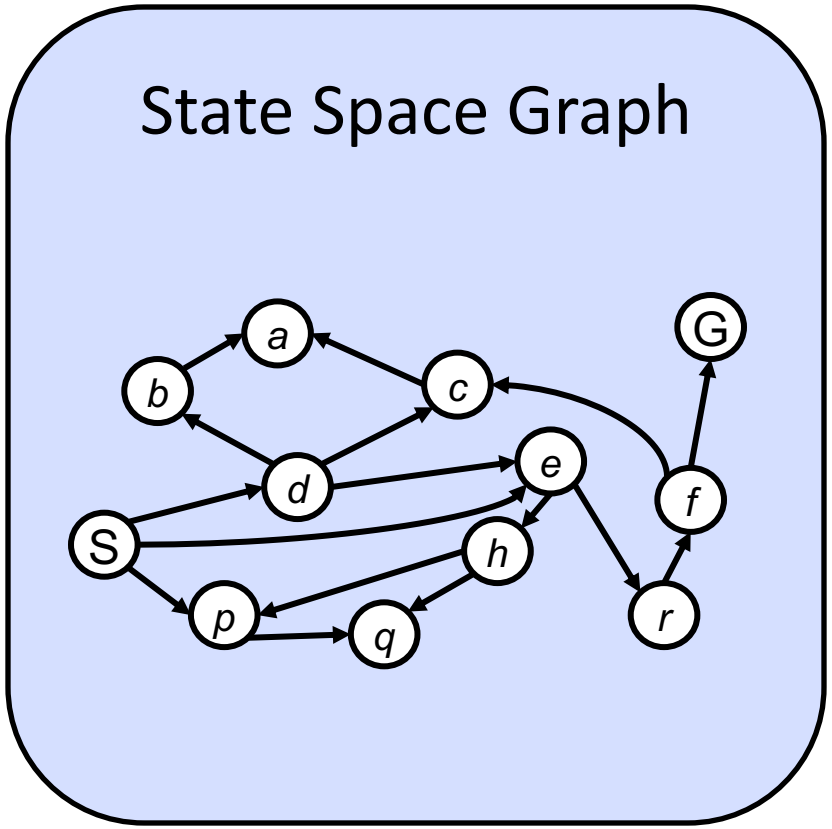
Last week: planning agents



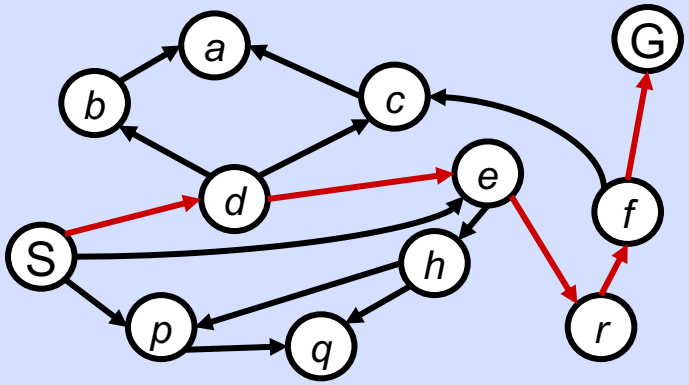
Last week: planning -> search

- Search problem:
  - States (configurations of the world)
  - Actions and costs
  - Successor function (world dynamics)
  - Start state and goal test



Last week: search tree

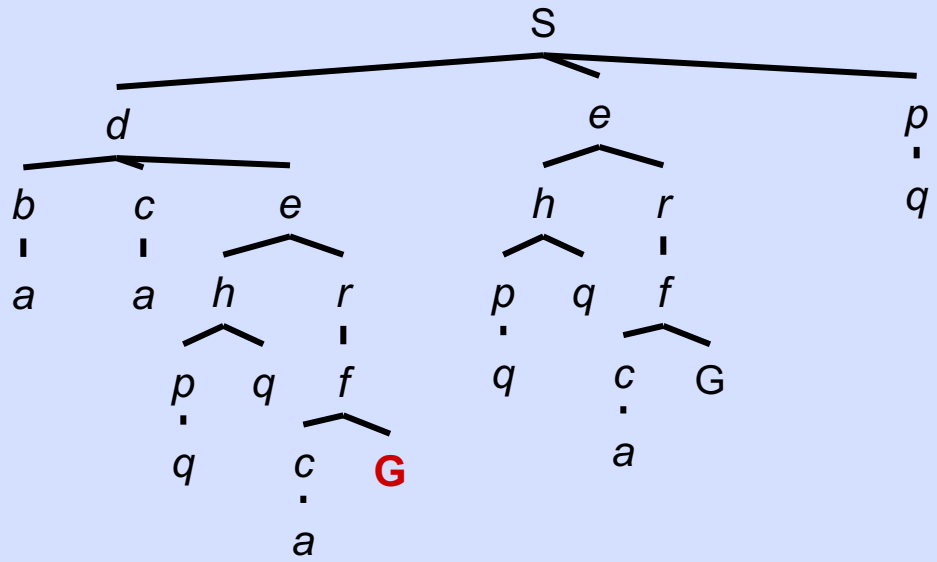
State Space Graph



Each NODE in in the search tree is an entire PATH in the state space graph.

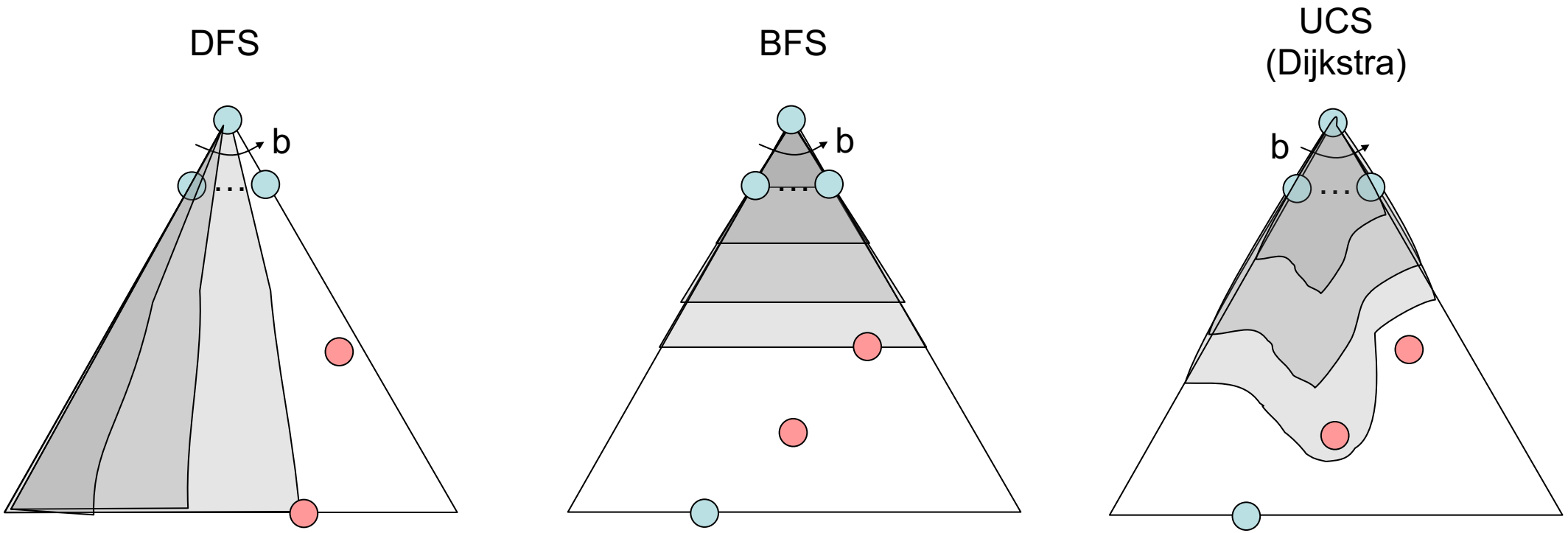
We construct both on demand – and we construct as little as possible.

Search Tree



Last week: search algorithms

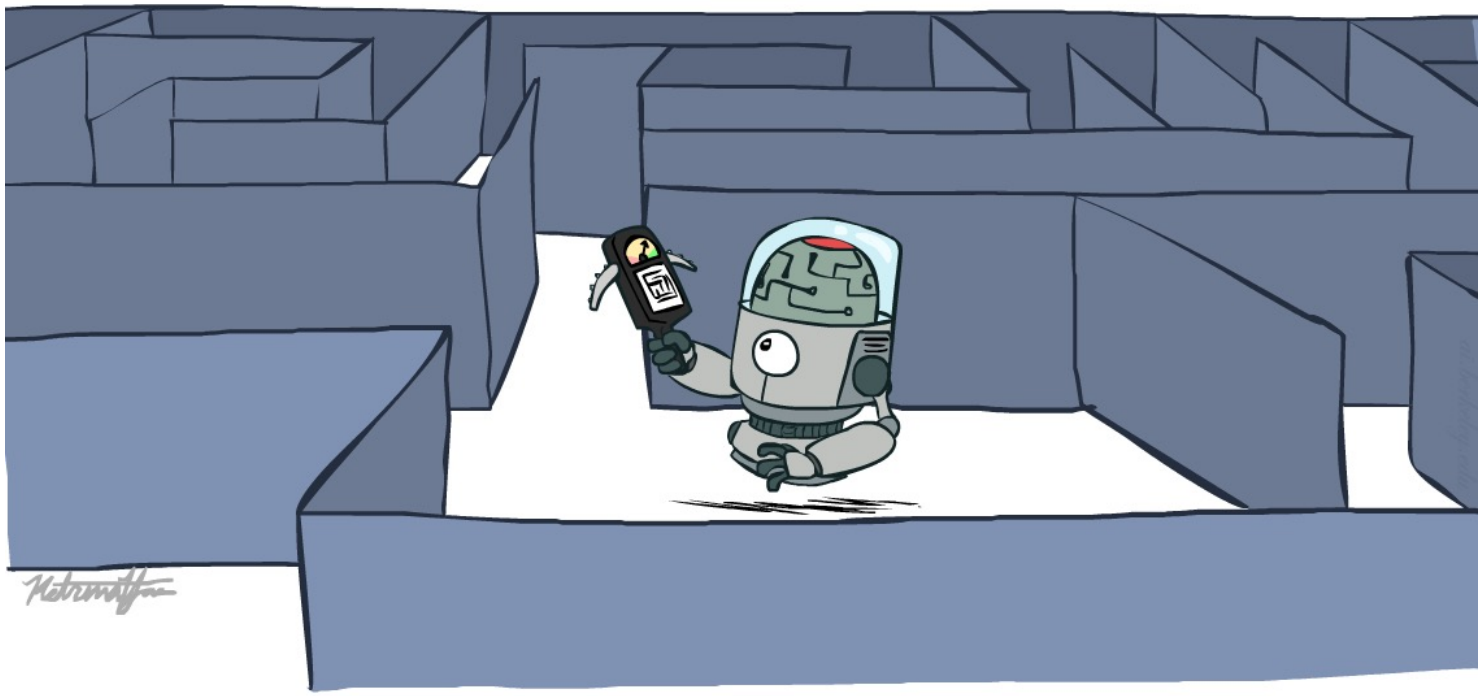
- Iteratively builds a (partial) search tree
- Maintains a fringe / priority queue of unexpanded nodes
- Optimal: finds least-cost plans



Video of Demo Empty UCS

CS 3317: Artificial Intelligence

Informed Search



Instructors: **Cai Panpan**

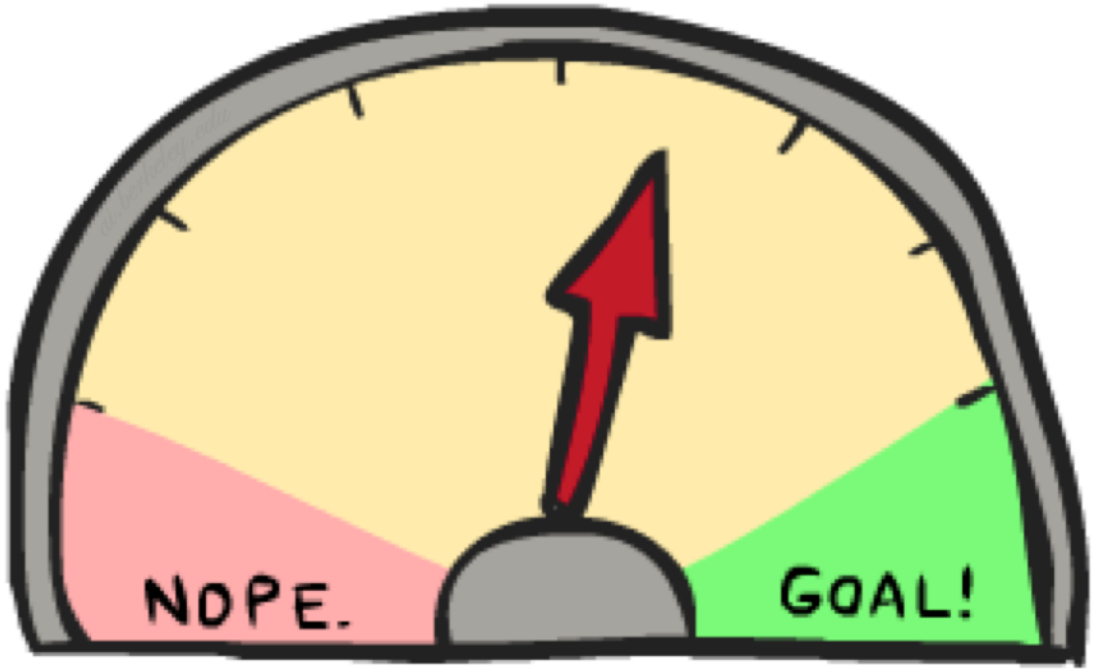
Shanghai Jiao Tong University

(slides adapted from UC Berkeley CS188)

Today

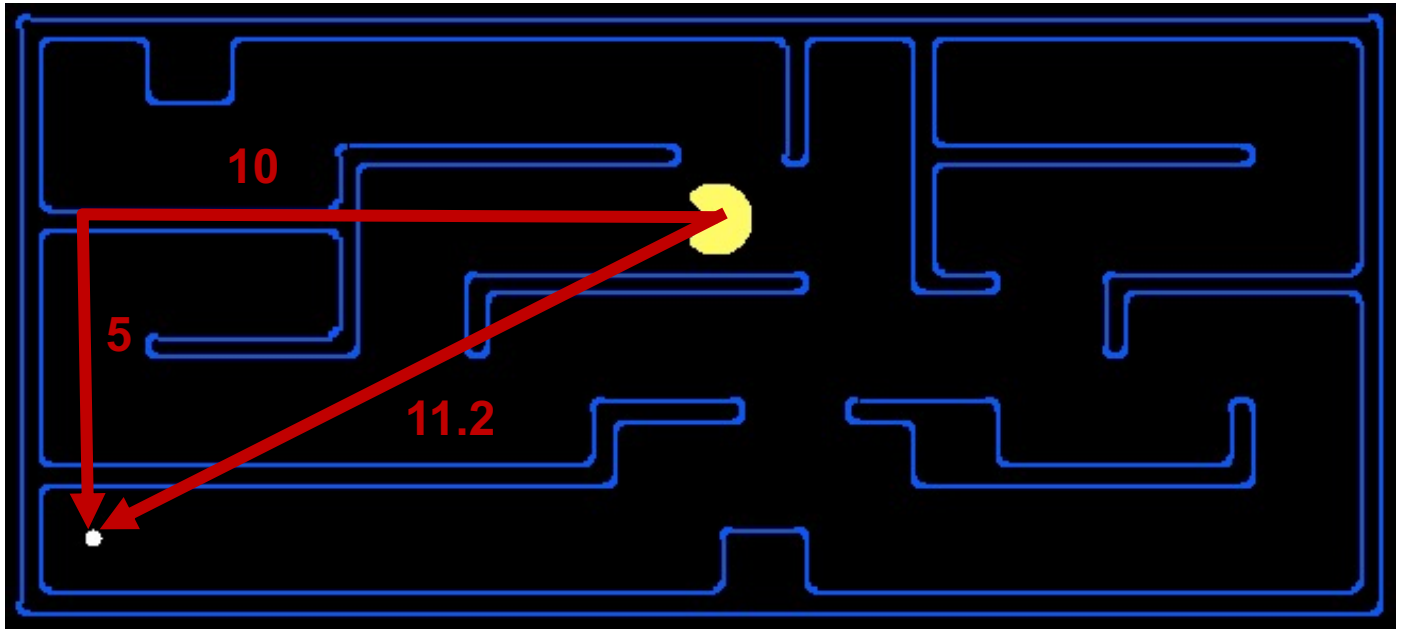
- Informed Search
  - Heuristics
  - Greedy Search
  - A* Search
    - Challenging contents!
- Graph Search

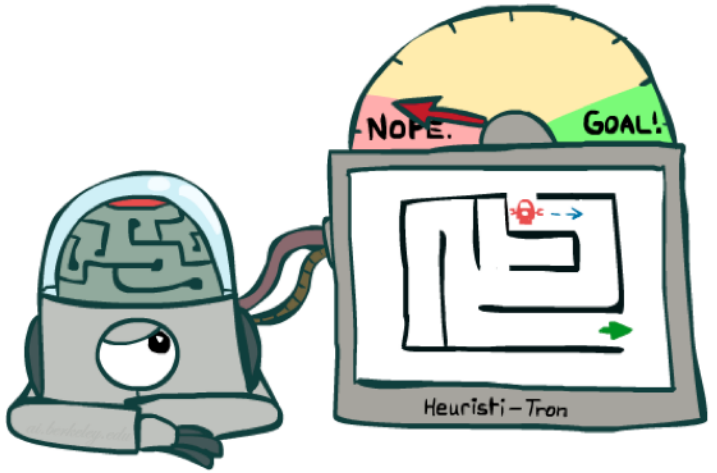
Informed Search

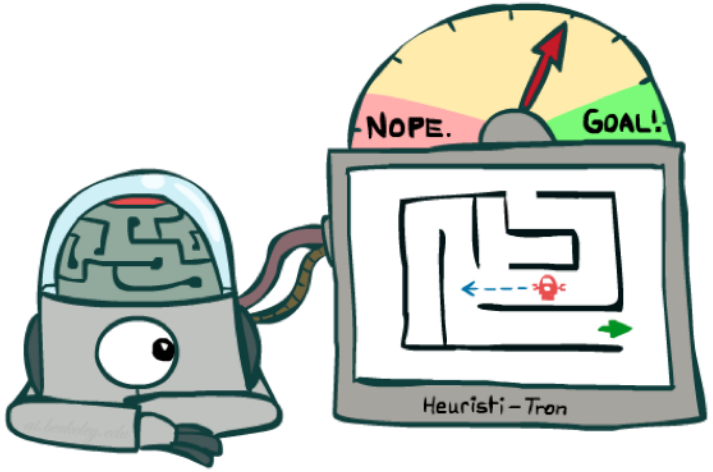


Search Heuristics

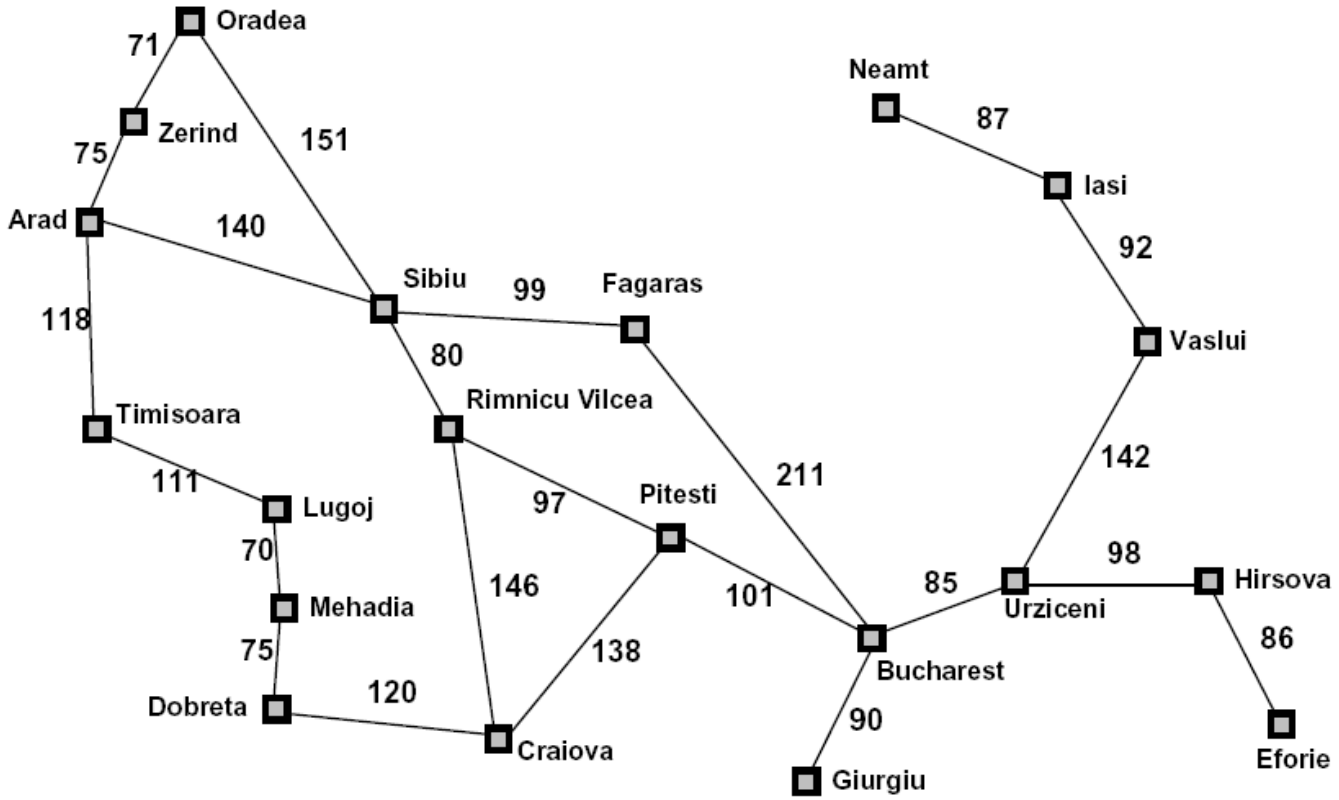
- A heuristic is:
  - A function that *estimates* how close a state is to a goal
  - Designed for a particular search problem
  - Preferably easy-to-compute
  - Examples: Manhattan distance, Euclidean distance for pathing







Example: Heuristic Function

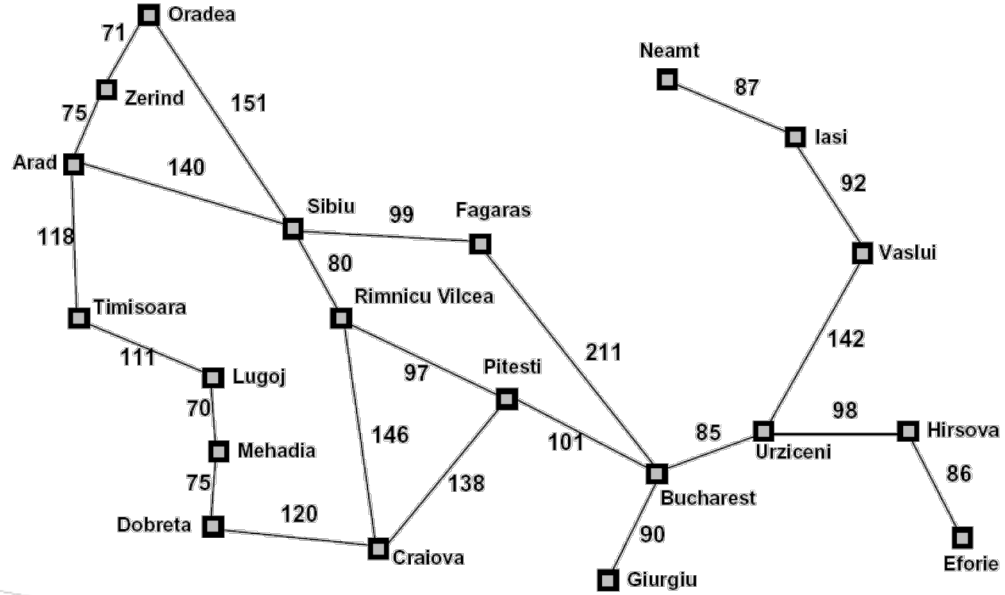


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

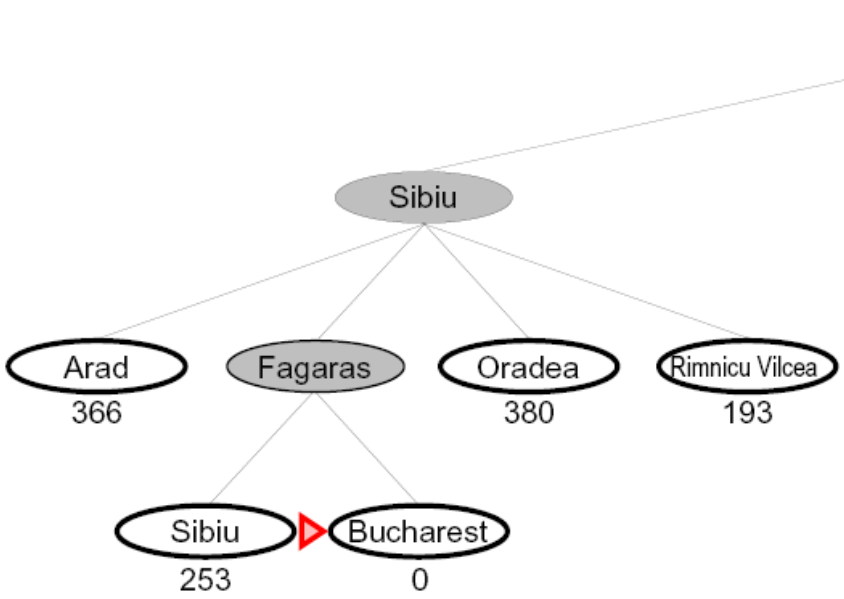
h(x)

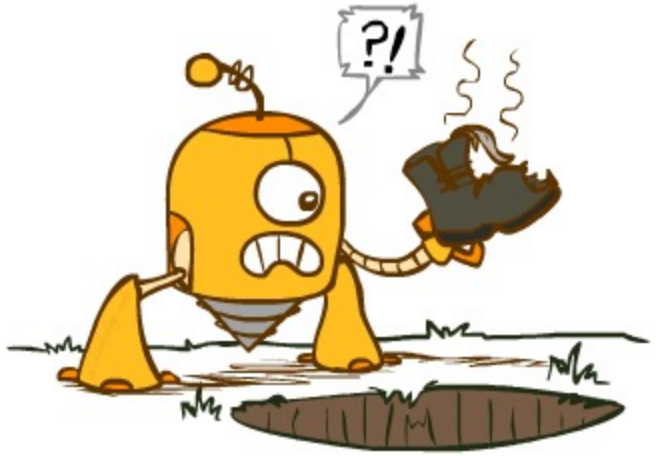
Greedy Search

Greedy Search



- Expand the node that seems closest…

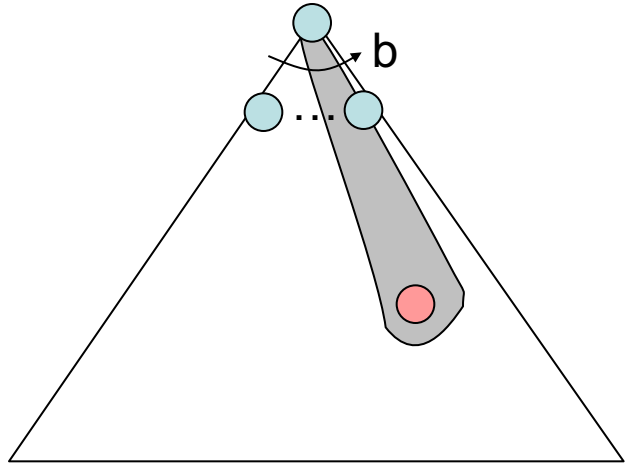




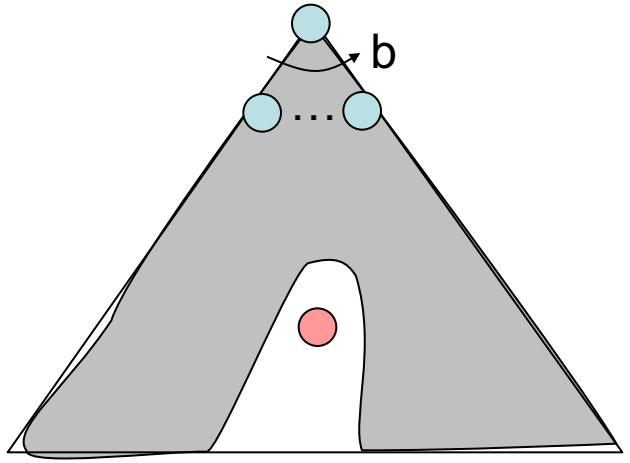
- What can go wrong?

Greedy Search

- Strategy: expand a node that you think is closest to a goal state
  - Heuristic: estimate of distance to nearest goal for each state



- A common case:
  - Best-first takes you straight to the (wrong) goal



- Worst-case: like a badly-guided DFS

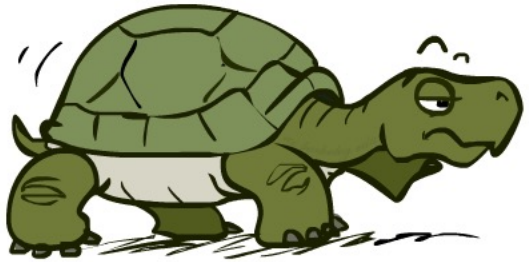
[Demo: contours greedy empty (L3D1)]
[Demo: contours greedy pacman small maze (L3D4)]

Video of Demo Contours Greedy (Empty)

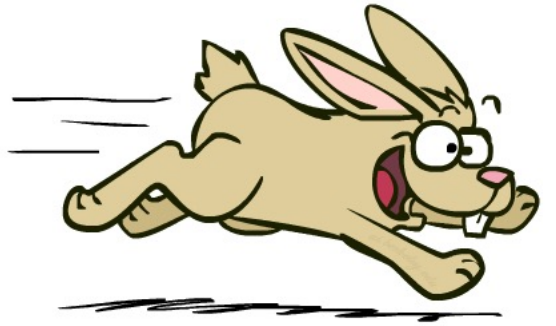
Video of Demo Contours Greedy (Pacman Small Maze)

A* Search

A* Search



UCS

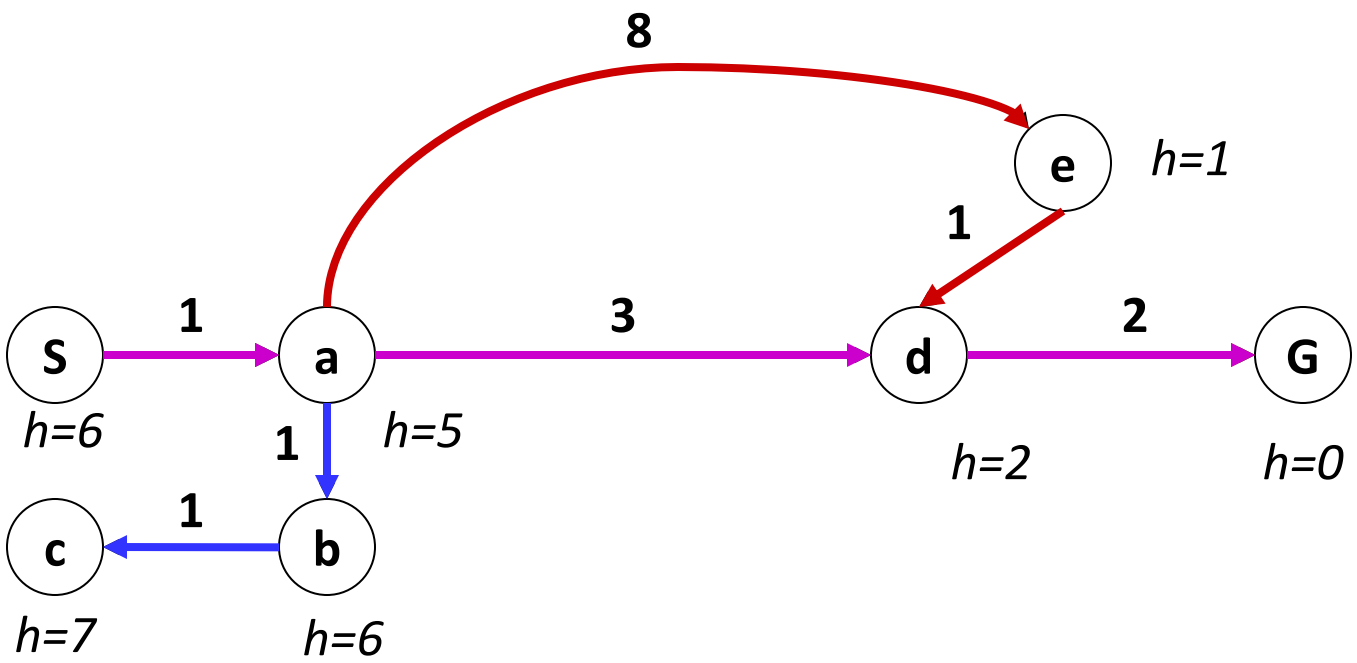


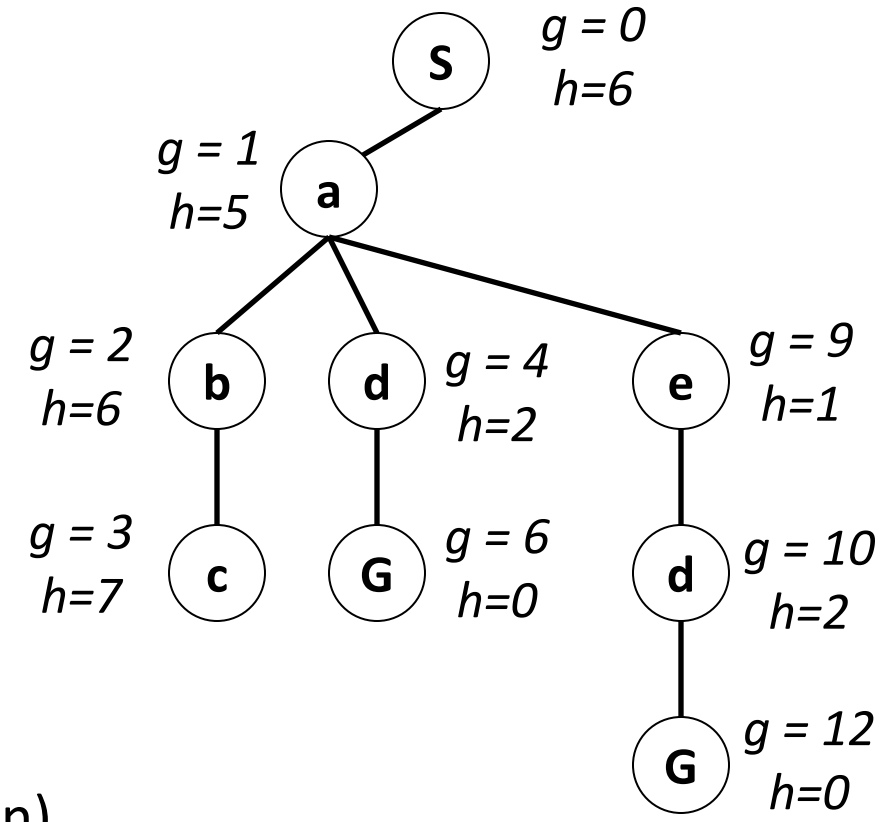
Greedy

A*

Combining UCS and Greedy

- Uniform-cost orders by path cost, or *backward cost* g(n)
- Greedy orders by goal proximity, or *forward cost* h(n)



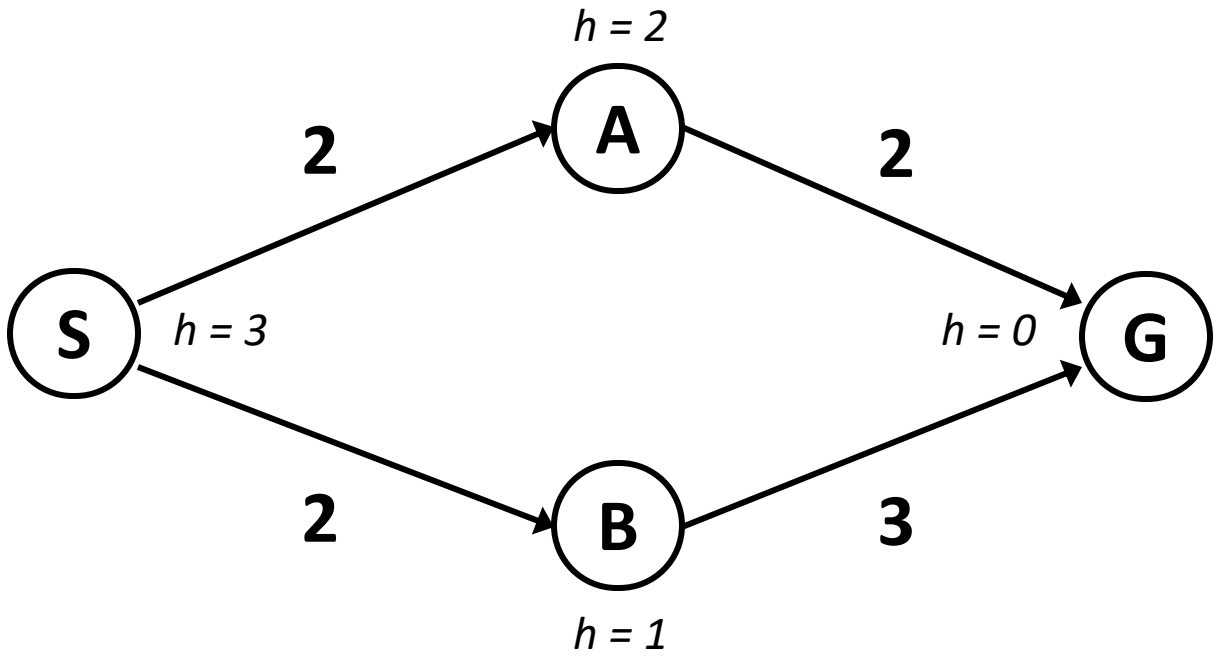


- A* Search orders by the sum: f(n) = g(n) + h(n)

Example: Teg Grenager

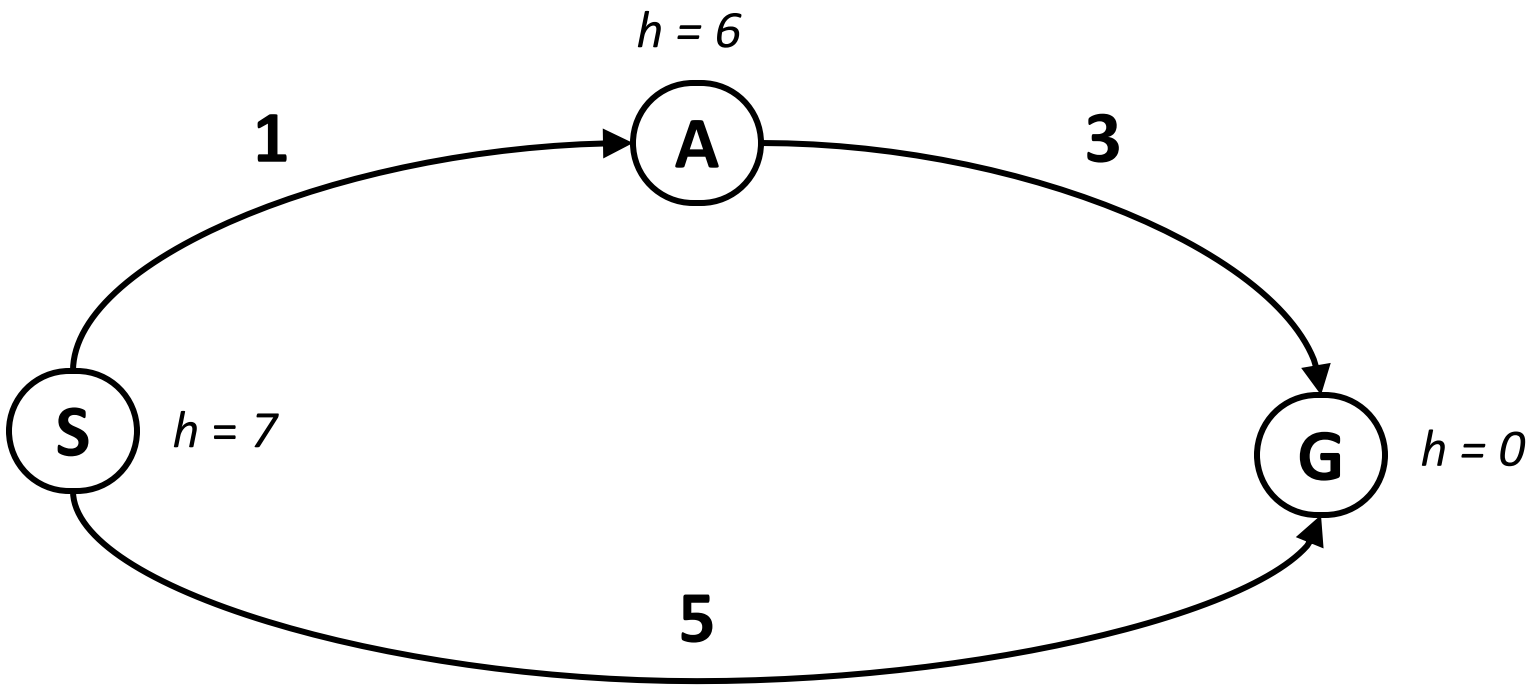
When should A* terminate?

- Should we stop when we enqueue a goal?



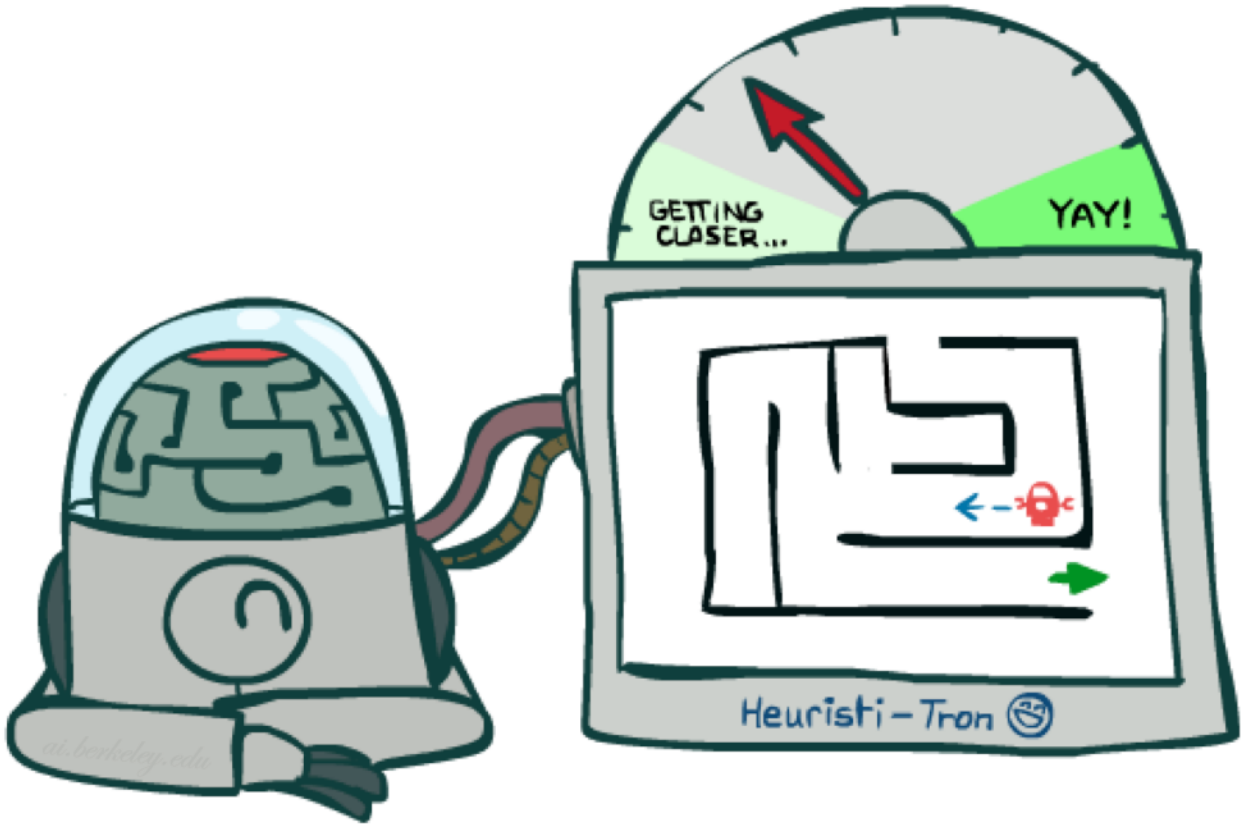
- No: only stop when we dequeue a goal

Is A* Optimal?

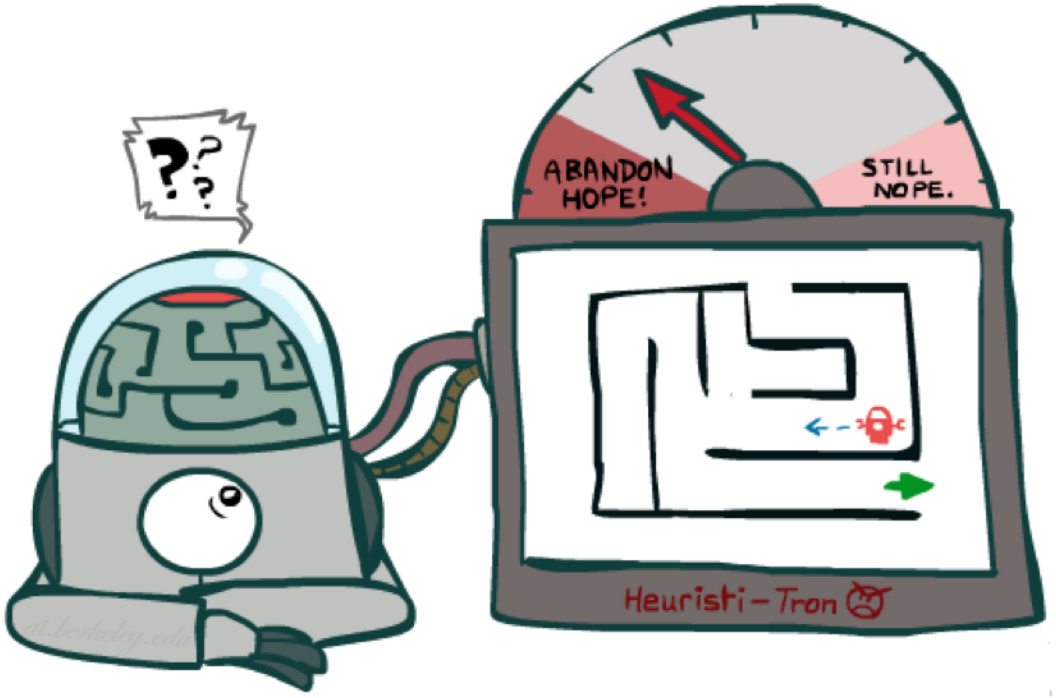


- What went wrong?
- Cost estimate of the optimal path > Actual cost of the suboptimal path
- We need estimates to be less than actual costs!

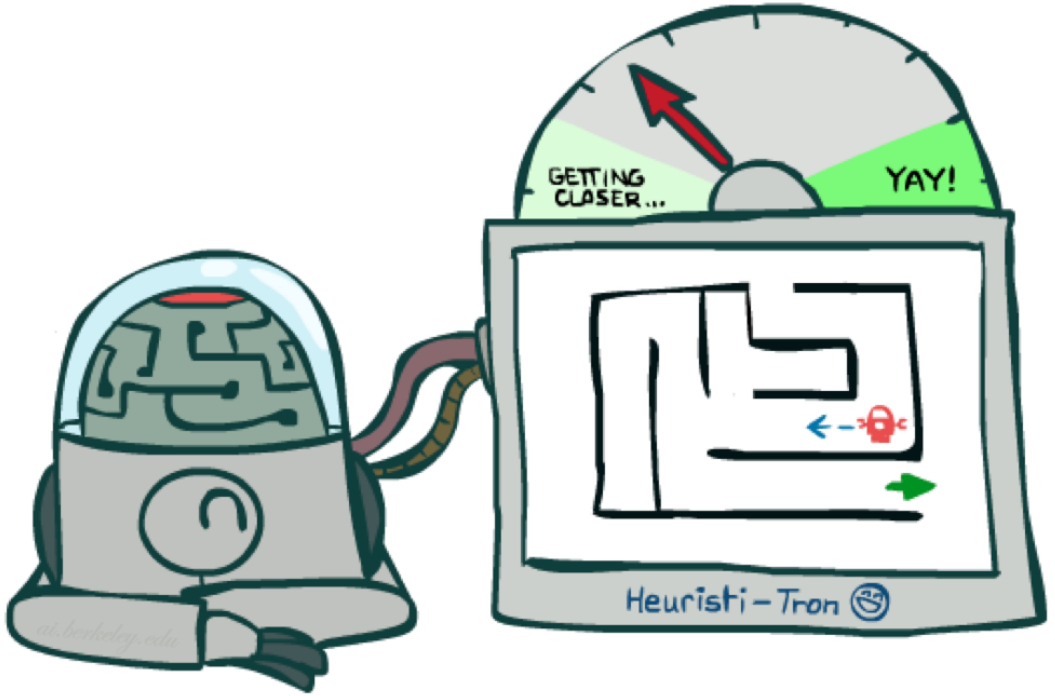
Admissible Heuristics



Idea: Admissibility



Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe



Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs

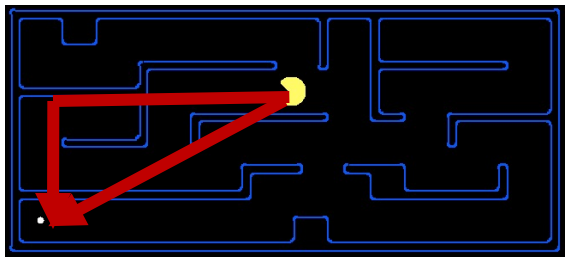
Admissible Heuristics

- A heuristic h is *admissible* (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

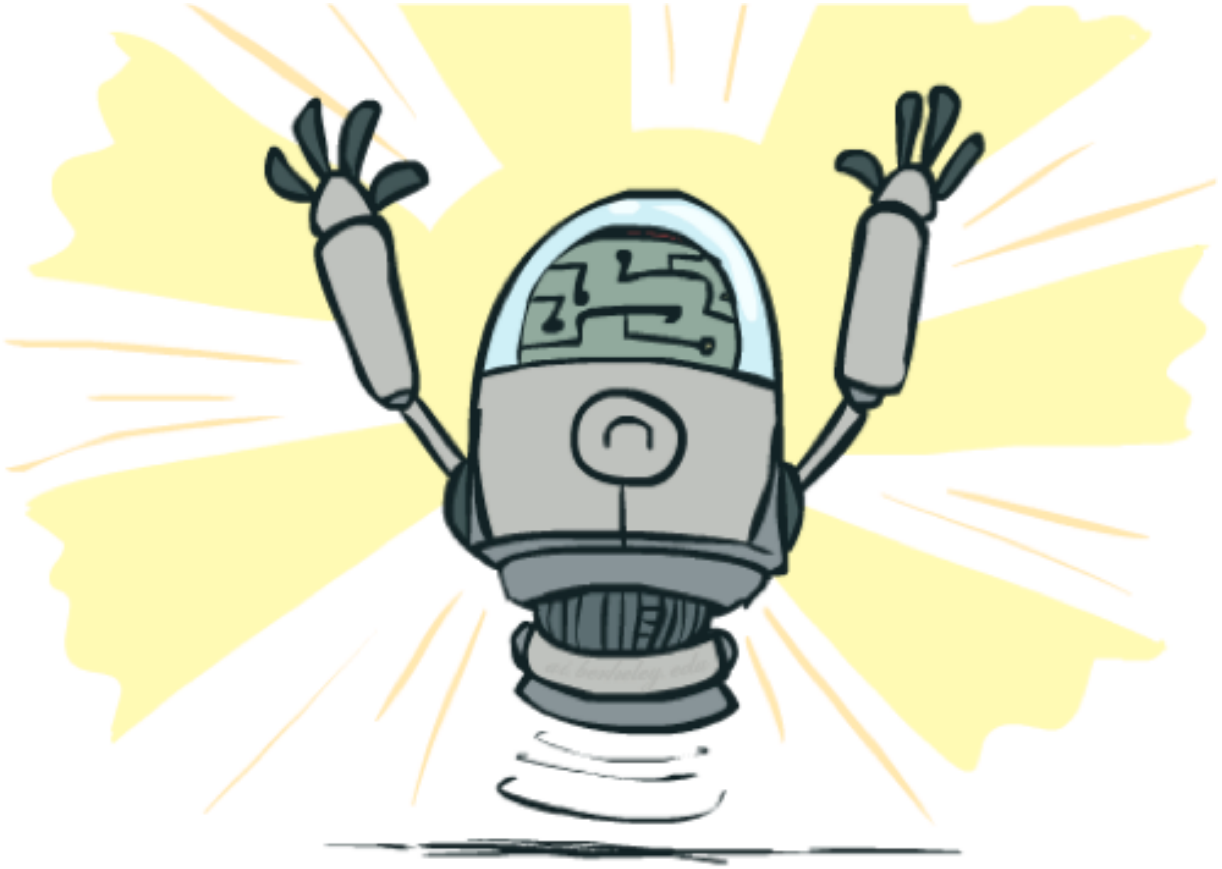
where $h^*(n)$ is the true cost to a nearest goal

- Examples:



- Coming up with admissible heuristics is most of what's involved in using A* in practice.

Optimality of A* Tree Search



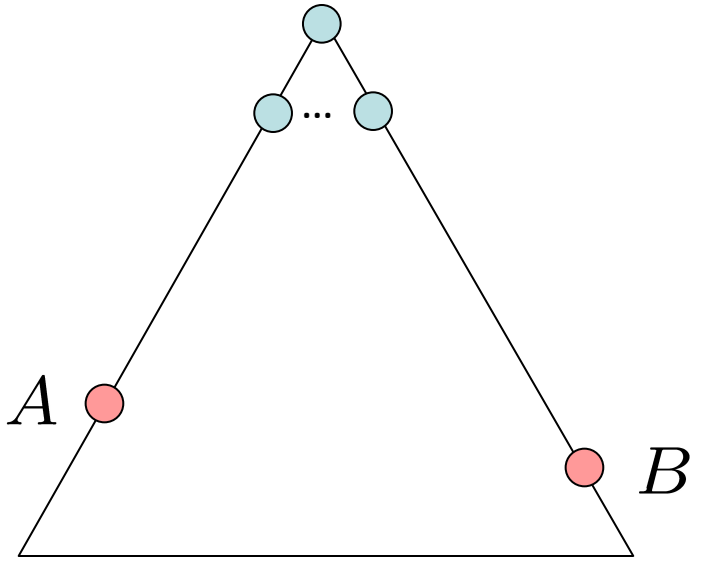
Optimality of A* Tree Search

Assume:

- A is an optimal goal node
- B is a suboptimal goal node
- h is admissible

Claim:

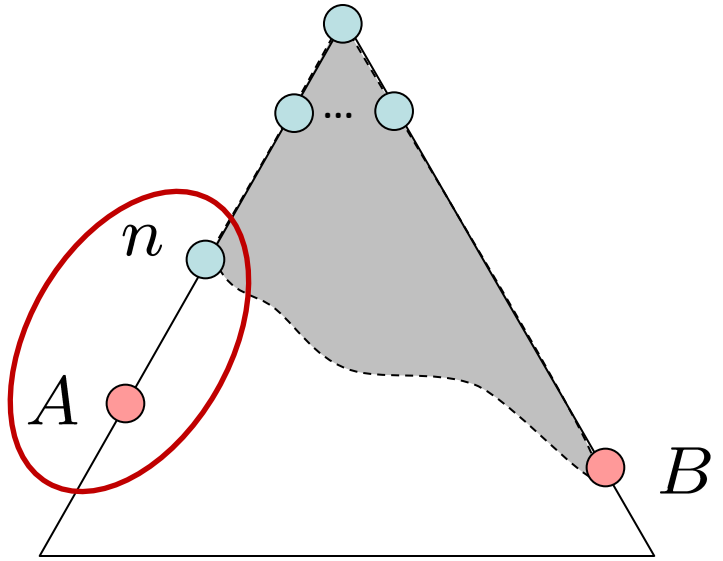
- A will exit the fringe before B



Optimality of A* Tree Search: Blocking

Proof:

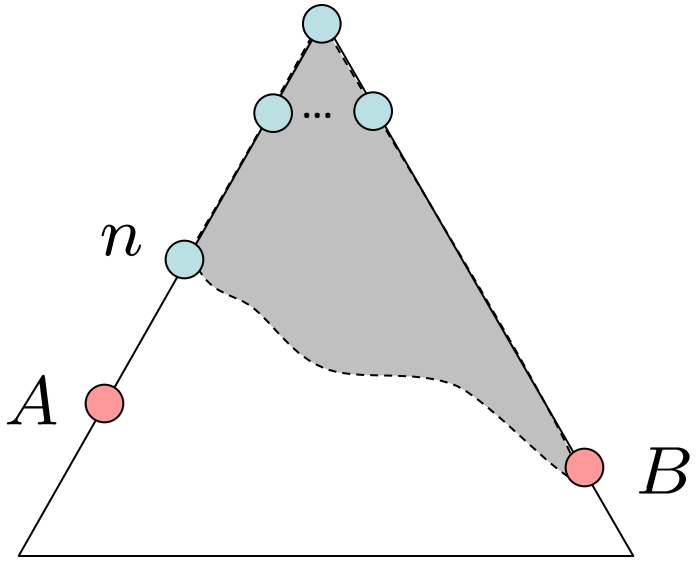
- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too (maybe A!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)



$$f(n) = g(n) + h(n) \quad \text{Definition of f-cost}$$
$$f(n) \leq g(A) \quad \text{Admissibility of h}$$
$$g(A) = f(A) \quad \text{h(A) = 0 (goal)}$$

Optimality of A* Tree Search: Blocking

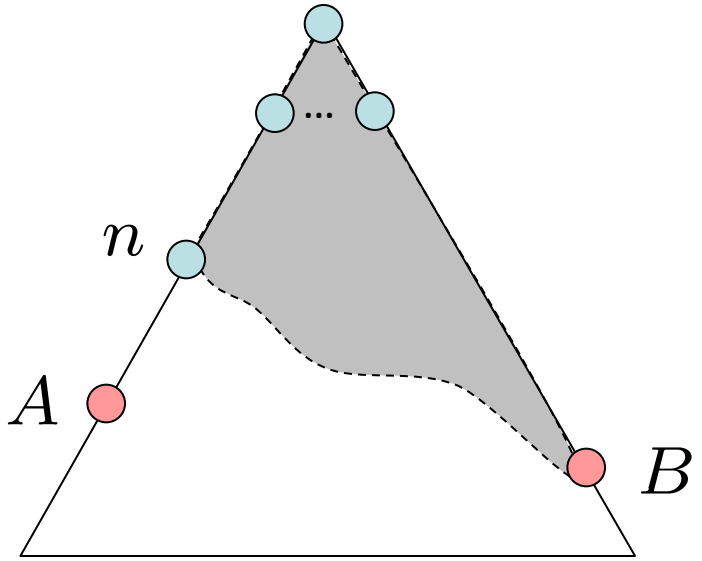
1. f(n) is less than or equal to f(A)
   - Definition of f-cost says:
     f(n) = g(n) + h(n) = (path cost to n) + (est. cost of n to A)
     f(A) = g(A) + h(A) = (path cost to A) + (est. cost of A to A)
   - The admissible heuristic must underestimate the true cost
     h(A) = (est. cost of A to A) = 0
   - So now, we have to compare:
     f(n) = g(n) + h(n) = (path cost to n) + (est. cost of n to A)
     f(A) = g(A) = (path cost to A)
   - h(n) must be an underestimate of the true cost from n to A
     (path cost to n) + (est. cost of n to A) ≤ (path cost to n) + (path cost of n to A)
     g(n) + h(n) ≤ g(A)
     f(n) ≤ f(A)



Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too (maybe A!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)

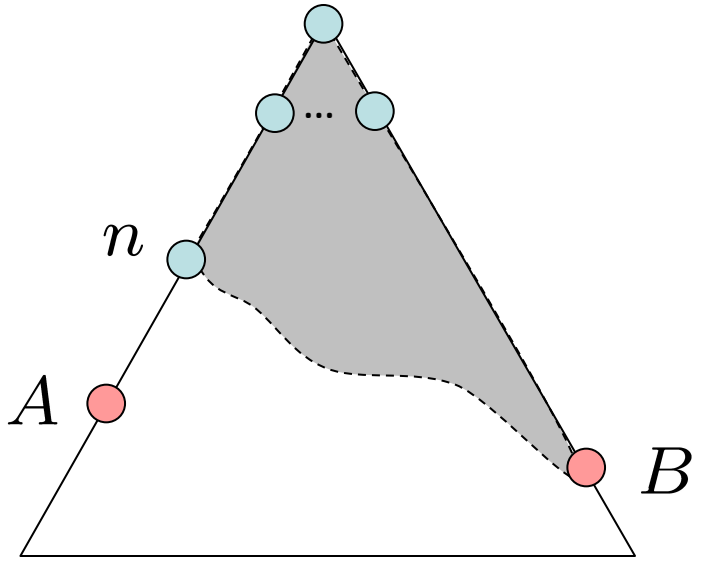


$g(A) < g(B)$ B is suboptimal

$f(A) < f(B)$ h = 0 at a goal

Optimality of A* Tree Search: Blocking

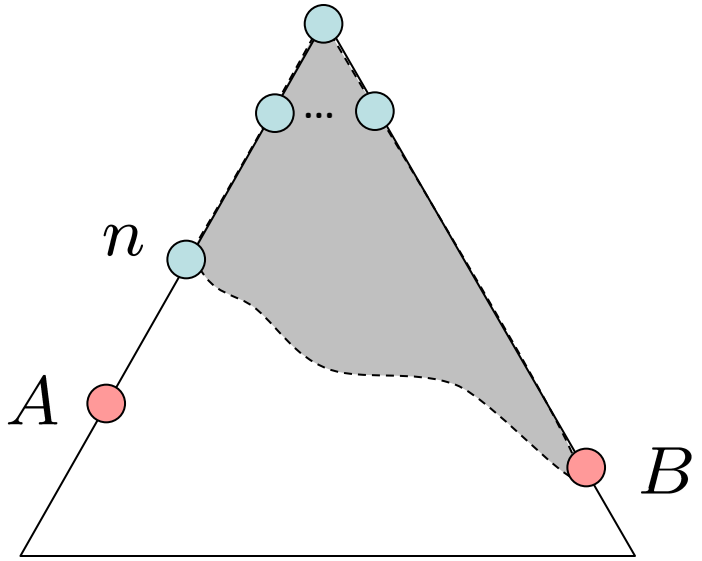
2. f(A) is less than f(B)
   - We know that:
     f(A) = g(A) + h(A) = (path cost to A) + (est. cost of A to A)
     f(B) = g(B) + h(B) = (path cost to B) + (est. cost of B to B)
   - The heuristic must underestimate the true cost:
     h(A) = h(B) = 0
   - So now, we have to compare:
     f(A) = g(A) = (path cost to A)
     f(B) = g(B) = (path cost to B)
   - We assumed that B is suboptimal! So
     (path cost to A) < (path cost to B)
     f(A) < f(B)



Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too (maybe A!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)
  3. *n* expands before B
- All ancestors of A expand before B
- A expands before B
- A* search is optimal

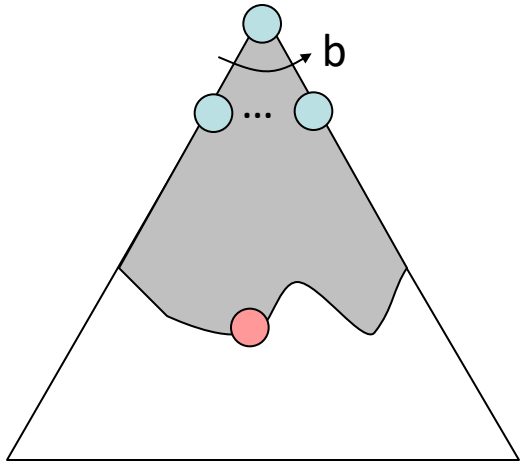


$$f(n) \leq f(A) < f(B)$$

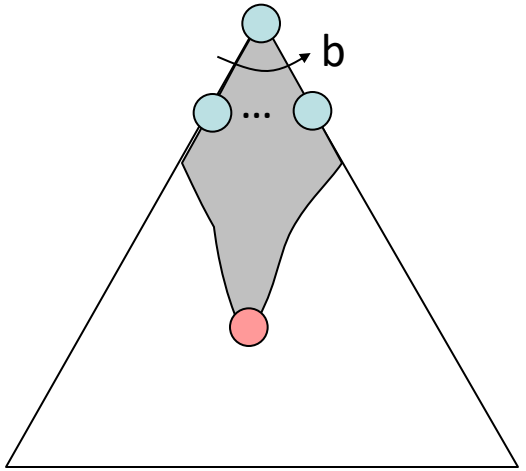
Properties of A*

Properties of A*

Uniform-Cost

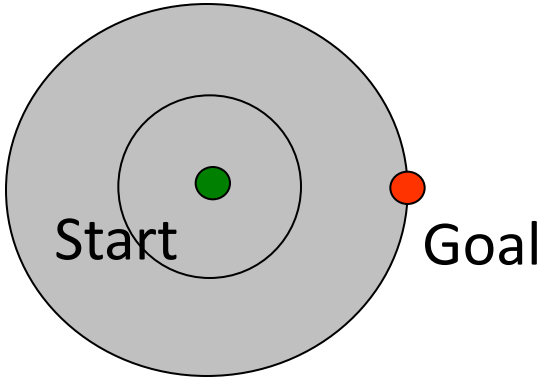


A*

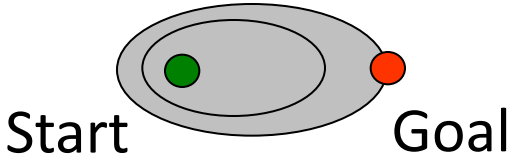


UCS vs A* Contours

- Uniform-cost expands equally in all "directions"



- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



[Demo: contours UCS / greedy / A* empty (L3D1)]
[Demo: contours A* pacman small maze (L3D5)]

Video of Demo Contours (Empty) -- UCS

Video of Demo Contours (Empty) -- Greedy

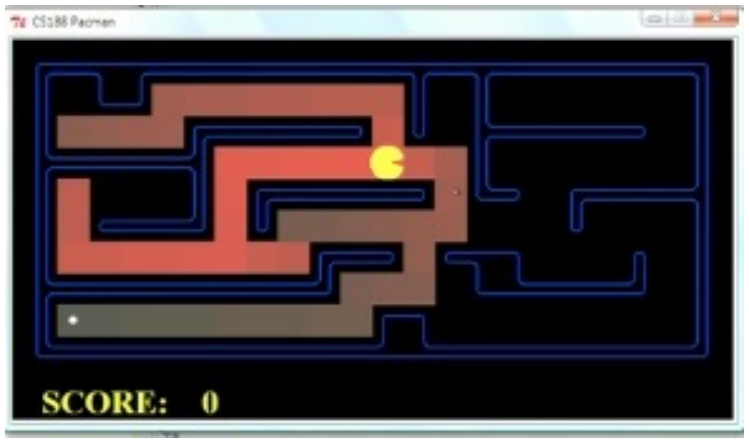
Video of Demo Contours (Empty) – A*

Video of Demo Contours (Pacman Small Maze) – A*

Comparison

Greedy

Uniform Cost



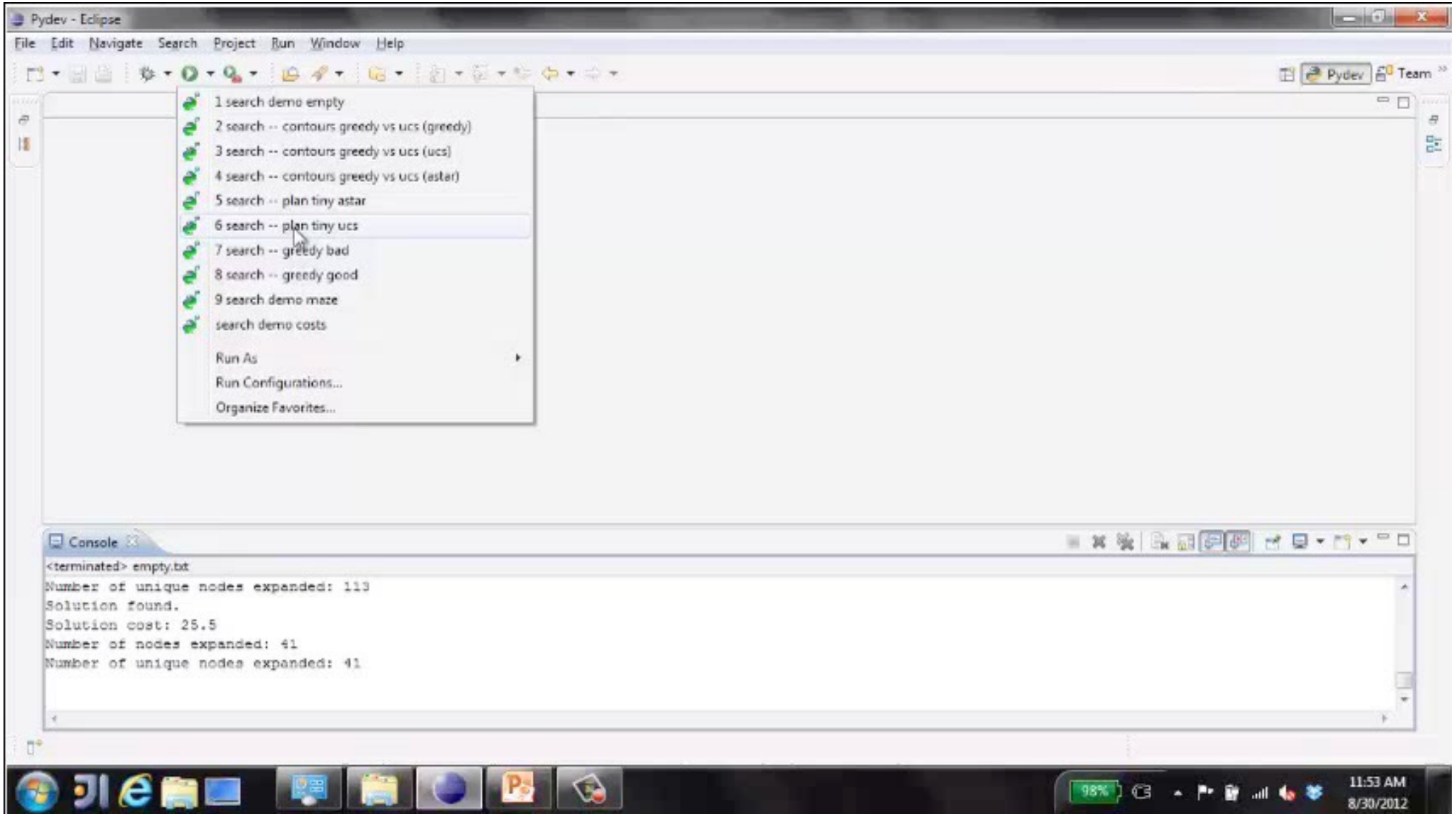
A*

A* Applications

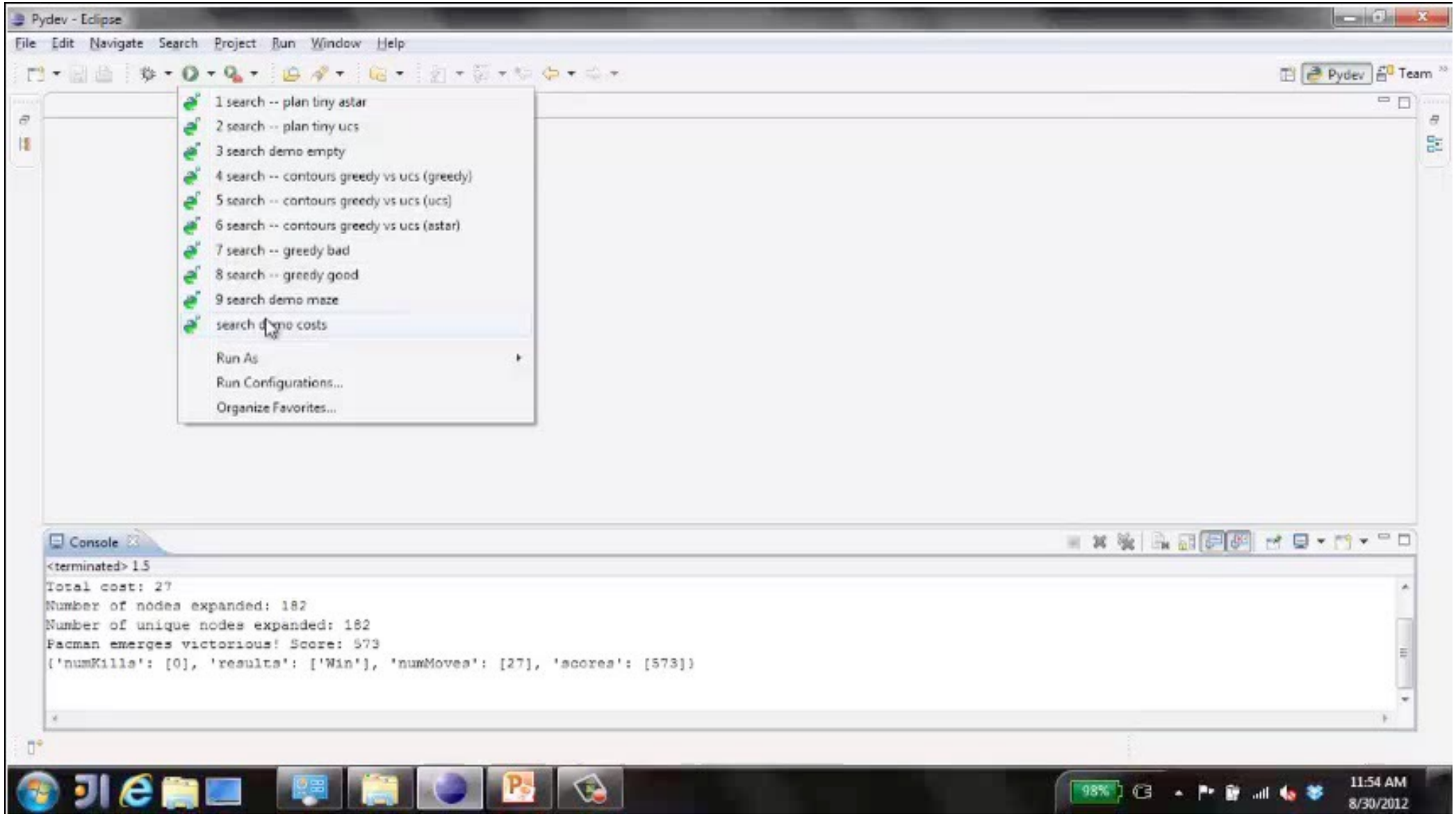
- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …

[Demo: UCS / A* pacman tiny maze (L3D6,L3D7)]
[Demo: guess algorithm Empty Shallow/Deep (L3D8)]

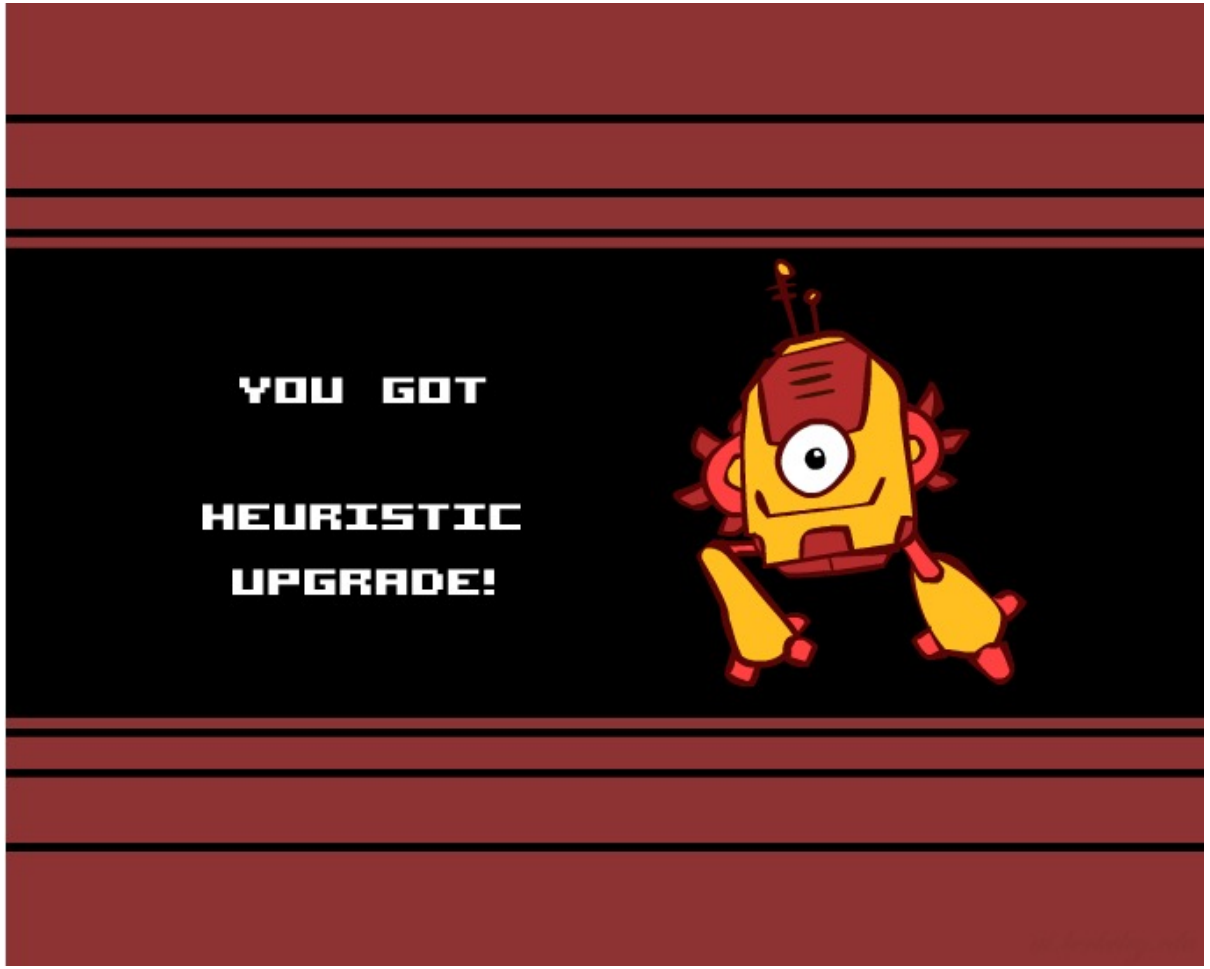
Video of Demo Pacman (Tiny Maze) – UCS / A*



Video of Demo Empty Water Shallow/Deep – Guess Algorithm



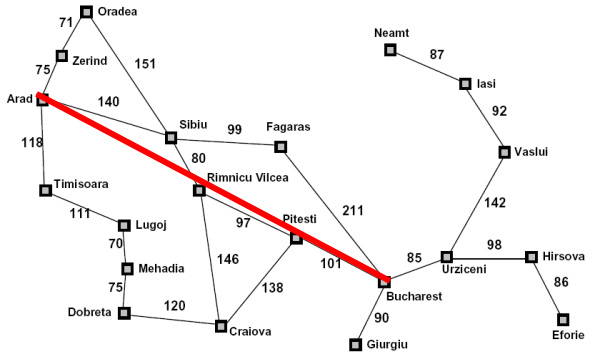
Creating Heuristics

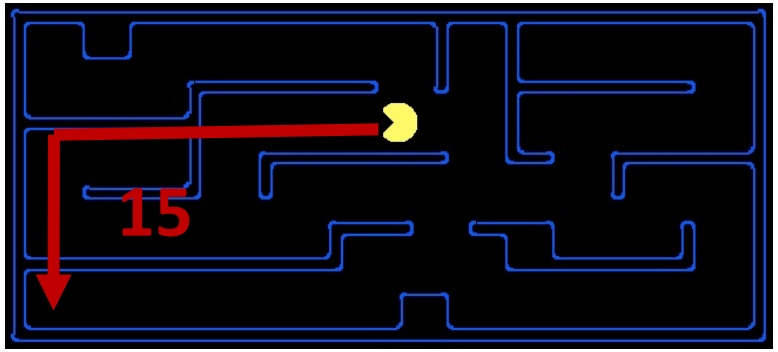


Creating Admissible Heuristics

- 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

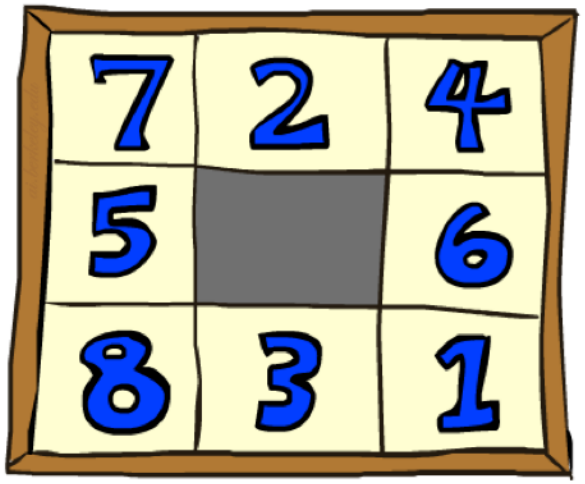
366



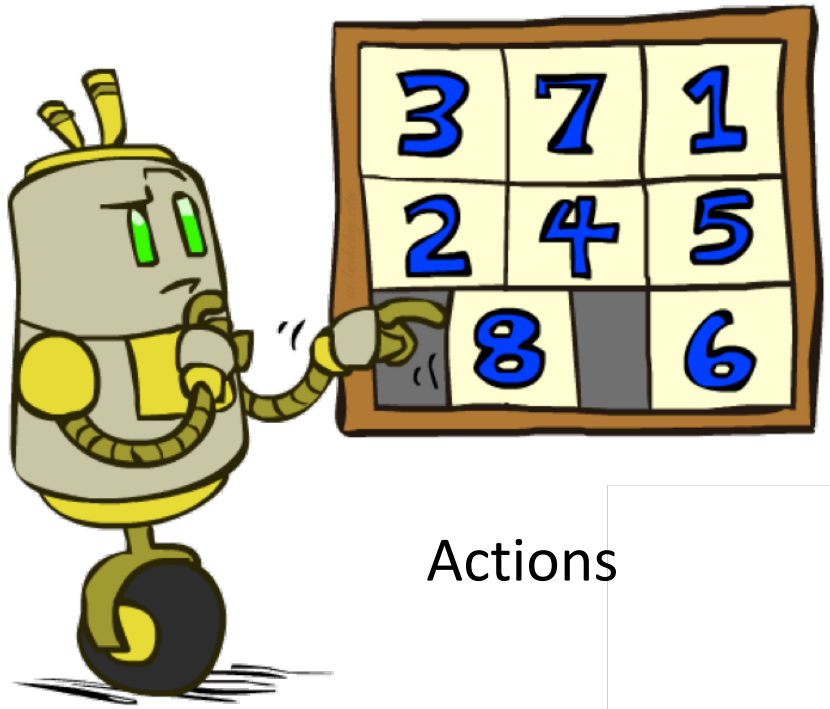


- Inadmissible heuristics are often useful too

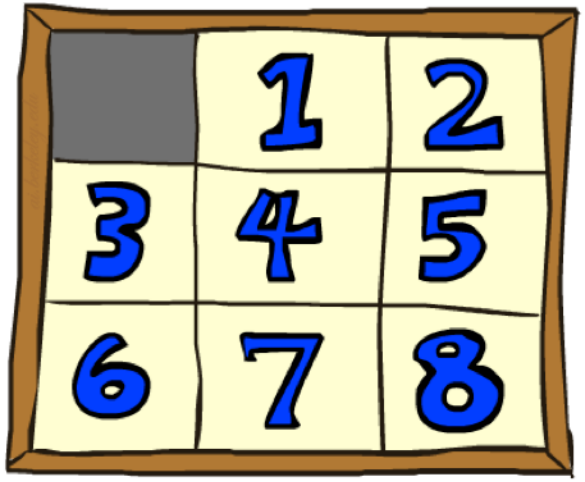
Example: 8 Puzzle



Start State



Actions

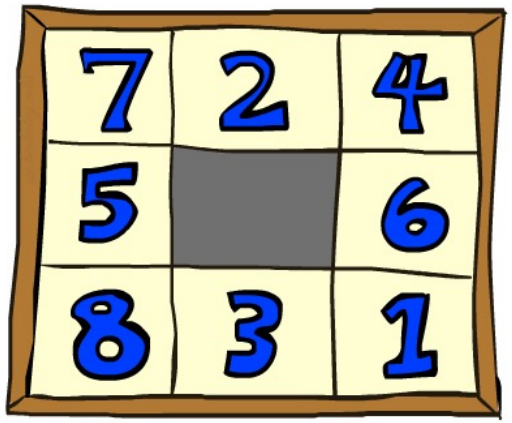


Goal State

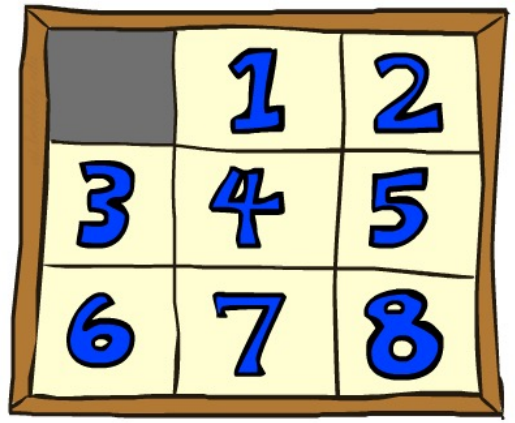
- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

8 Puzzle I

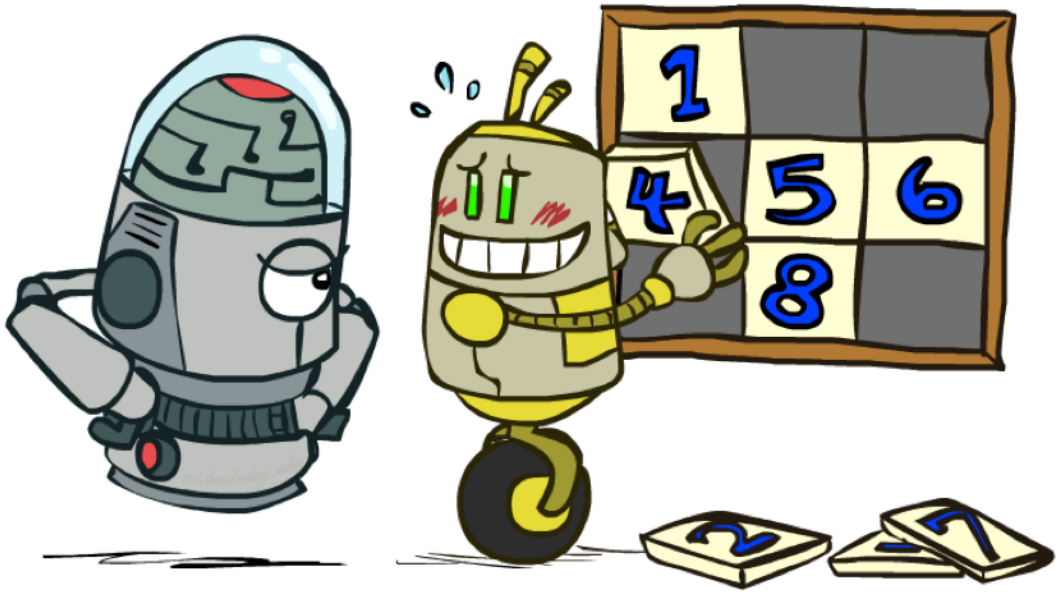
- Heuristic: Number of tiles misplaced
- Why is it admissible?
- This is a *relaxed-problem* heuristic
- h(start) = 8



Start State



Goal State

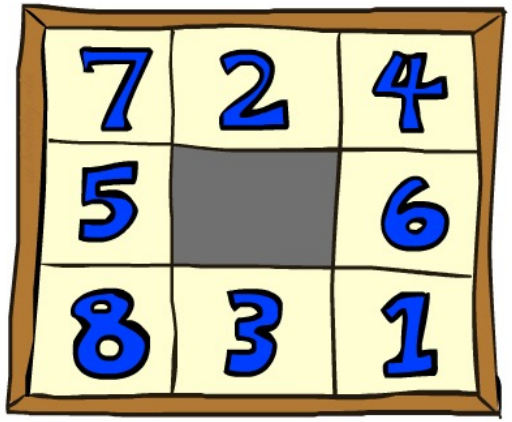


	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

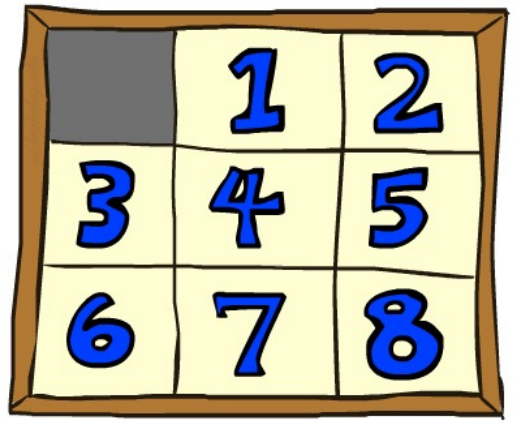
Statistics from Andrew Moore

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
- Why is it admissible?
- h(start) = 3 + 1 + 2 + … = 18



Start State

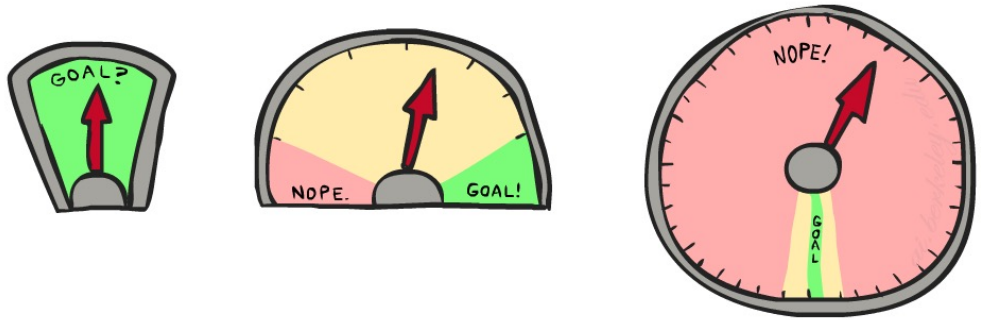


Goal State

	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

Semi-Lattice of Heuristics

Trivial Heuristics, Dominance

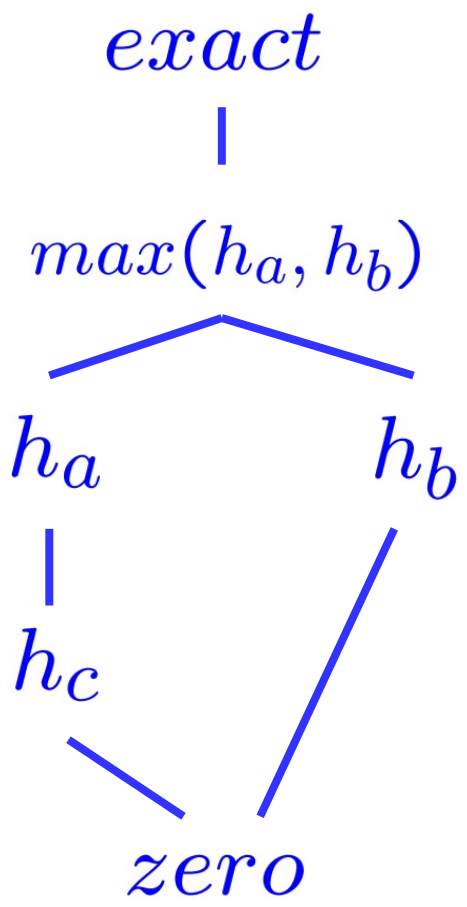
- Dominance: $h_a \geq h_c$ if

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

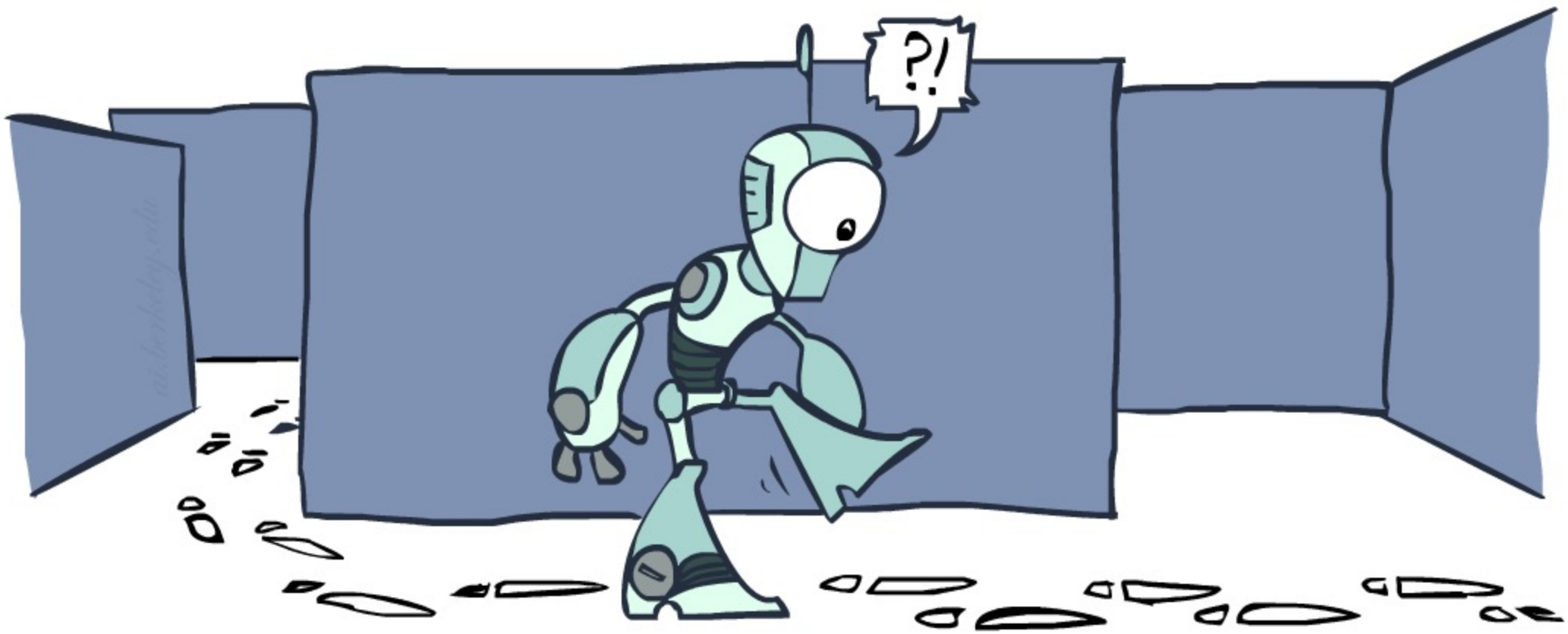
- Heuristics form a semi-lattice:
  - Max of admissible heuristics is admissible

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

- Trivial heuristics
  - Bottom of lattice is the zero heuristic (what does this give us?)
  - Top of lattice is the exact heuristic

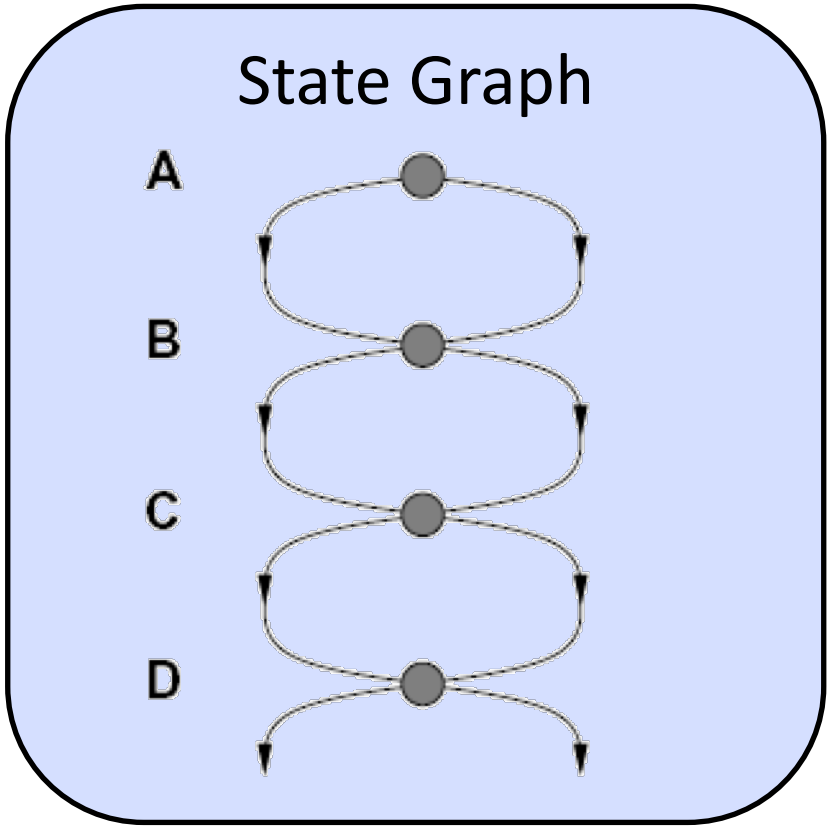


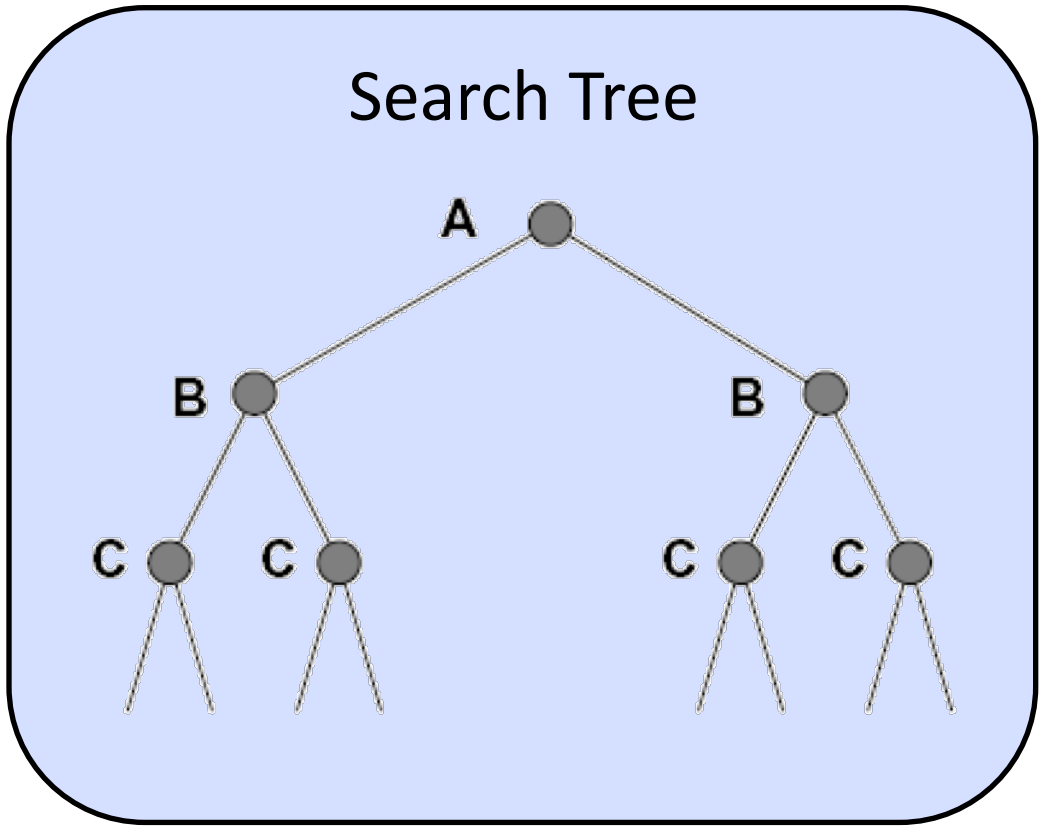
Graph Search



Tree Search: Extra Work!

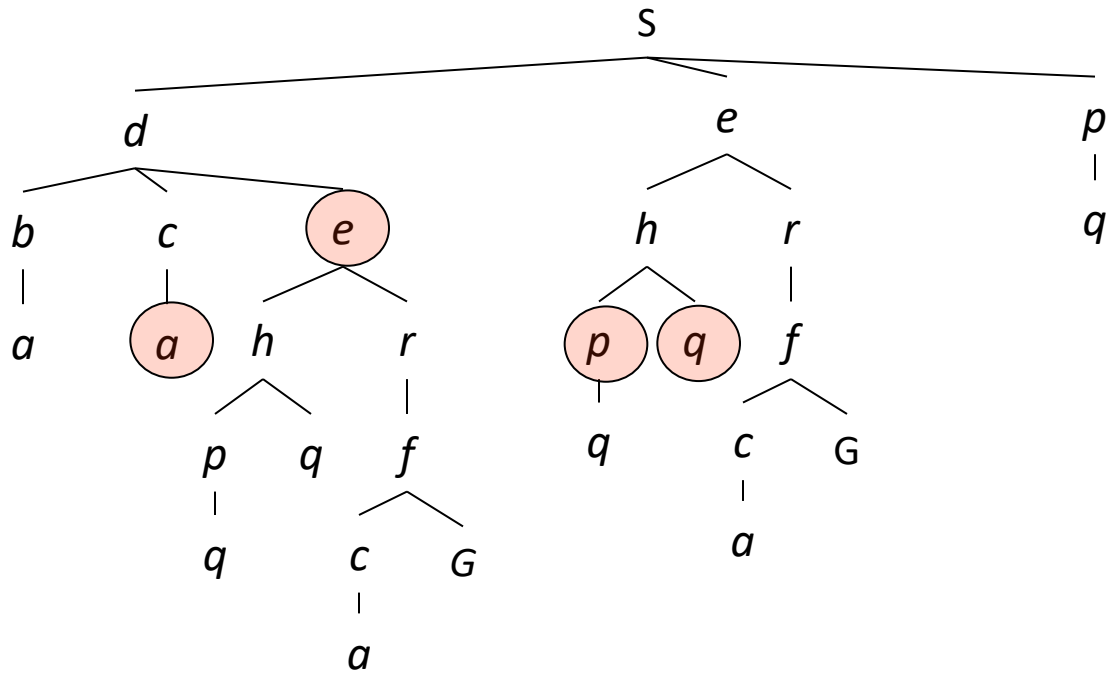
- Failure to detect repeated states can cause exponentially more work.





Graph Search

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

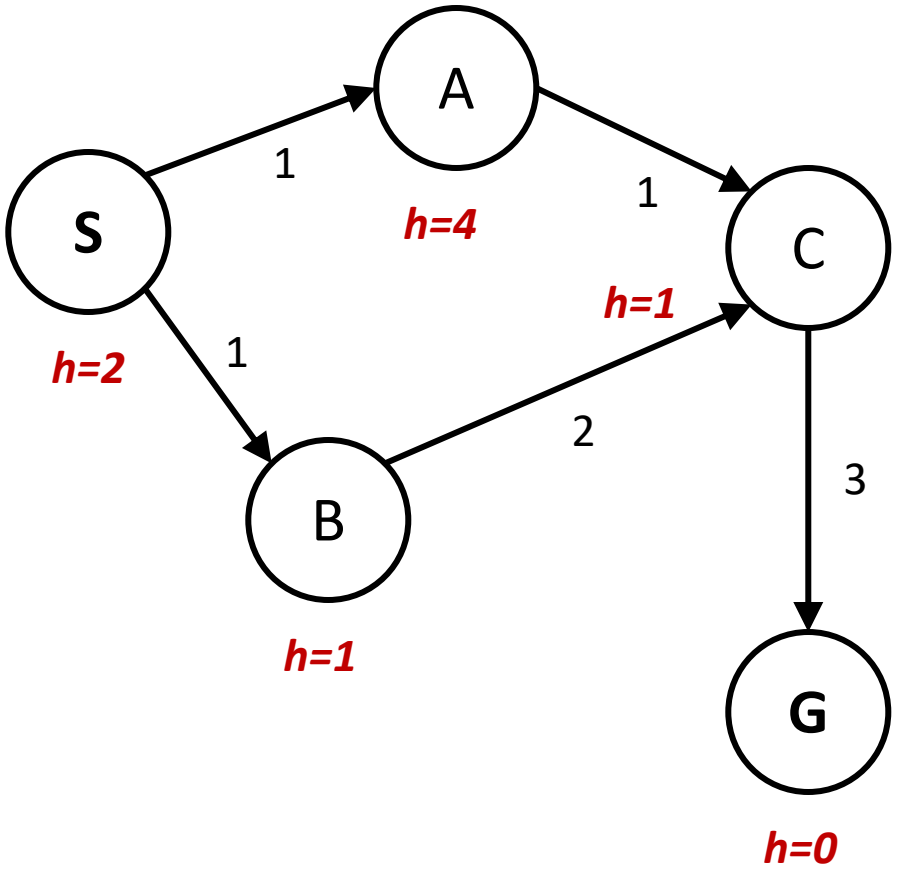


Graph Search

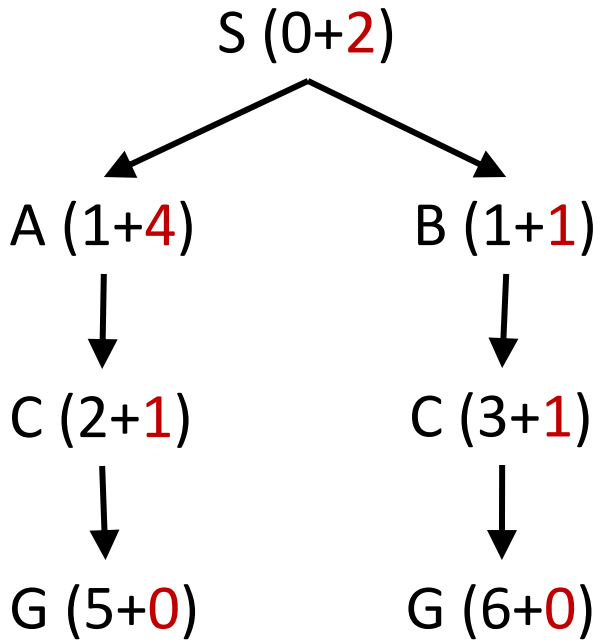
- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states (“closed set”)
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to closed set
- Important: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

A* Graph Search Gone Wrong?

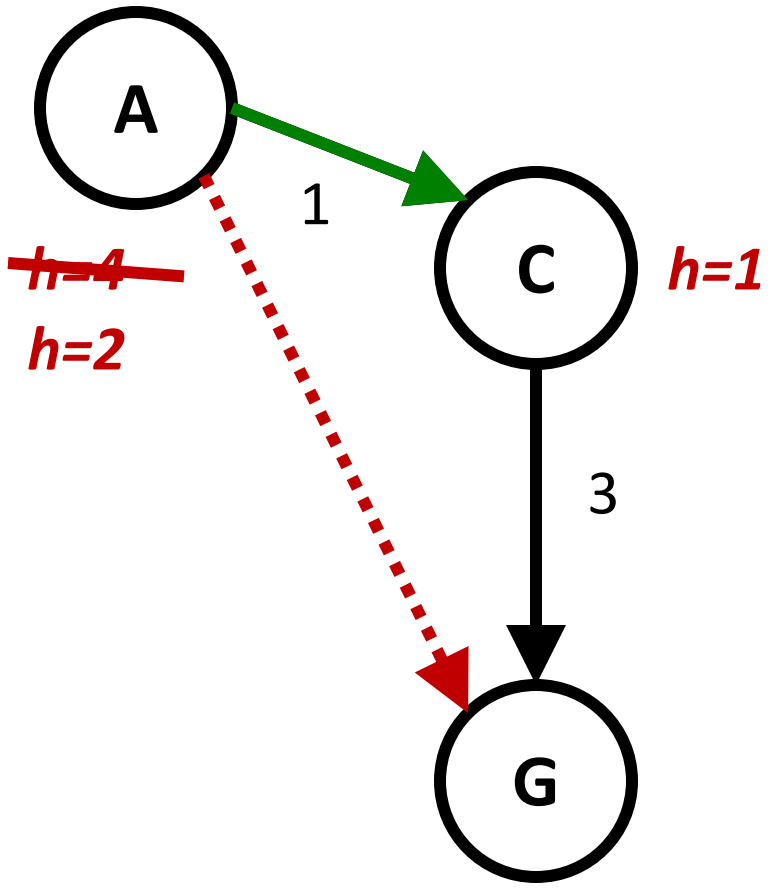
State space graph



Search tree



Consistency of Heuristics



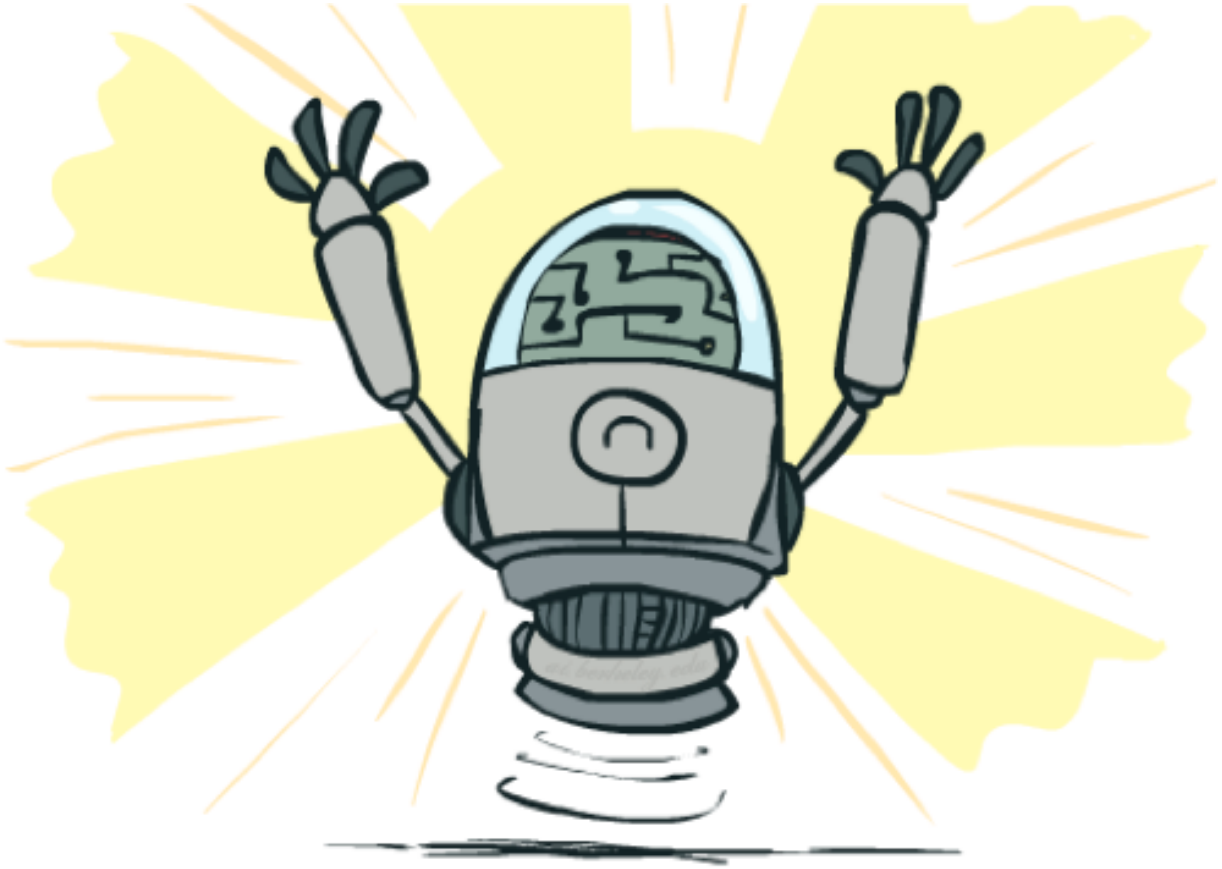
- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal

    h(A) ≤ 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

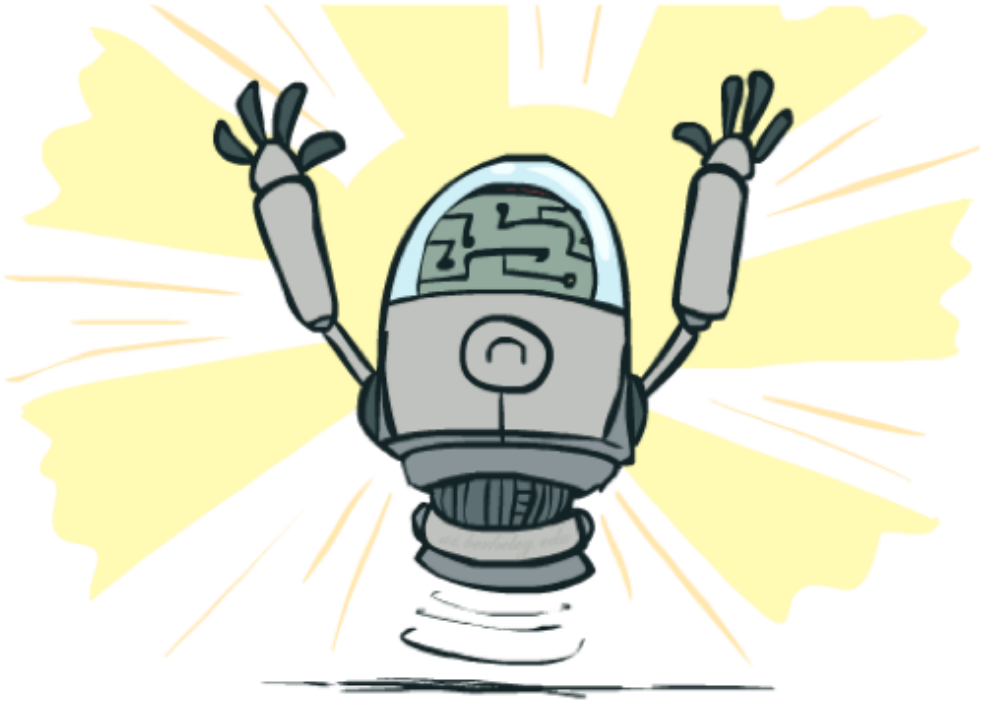
    h(A) ≤ cost(A to C) + h(C)
  - A* graph search is optimal

Optimality of A* Graph Search



Optimality

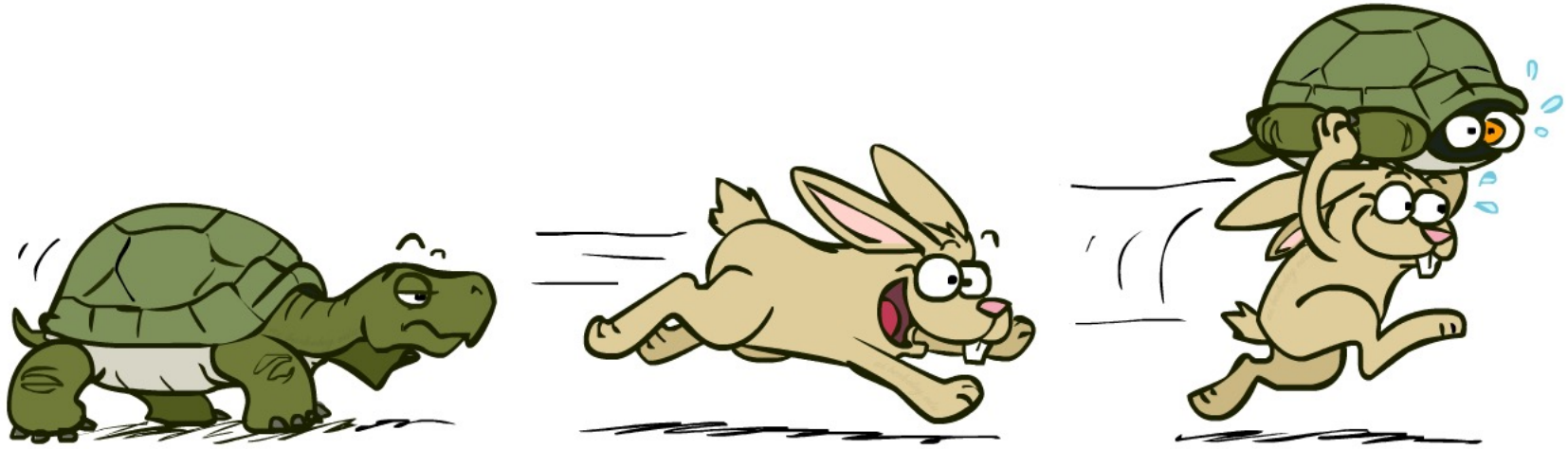
- Tree search:
  - A* is optimal if heuristic is admissible
  - UCS is a special case (h = 0)
- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal (h = 0 is consistent)
- Consistency implies admissibility
- In general, most natural admissible heuristics tend to be consistent, especially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



A*: Summary

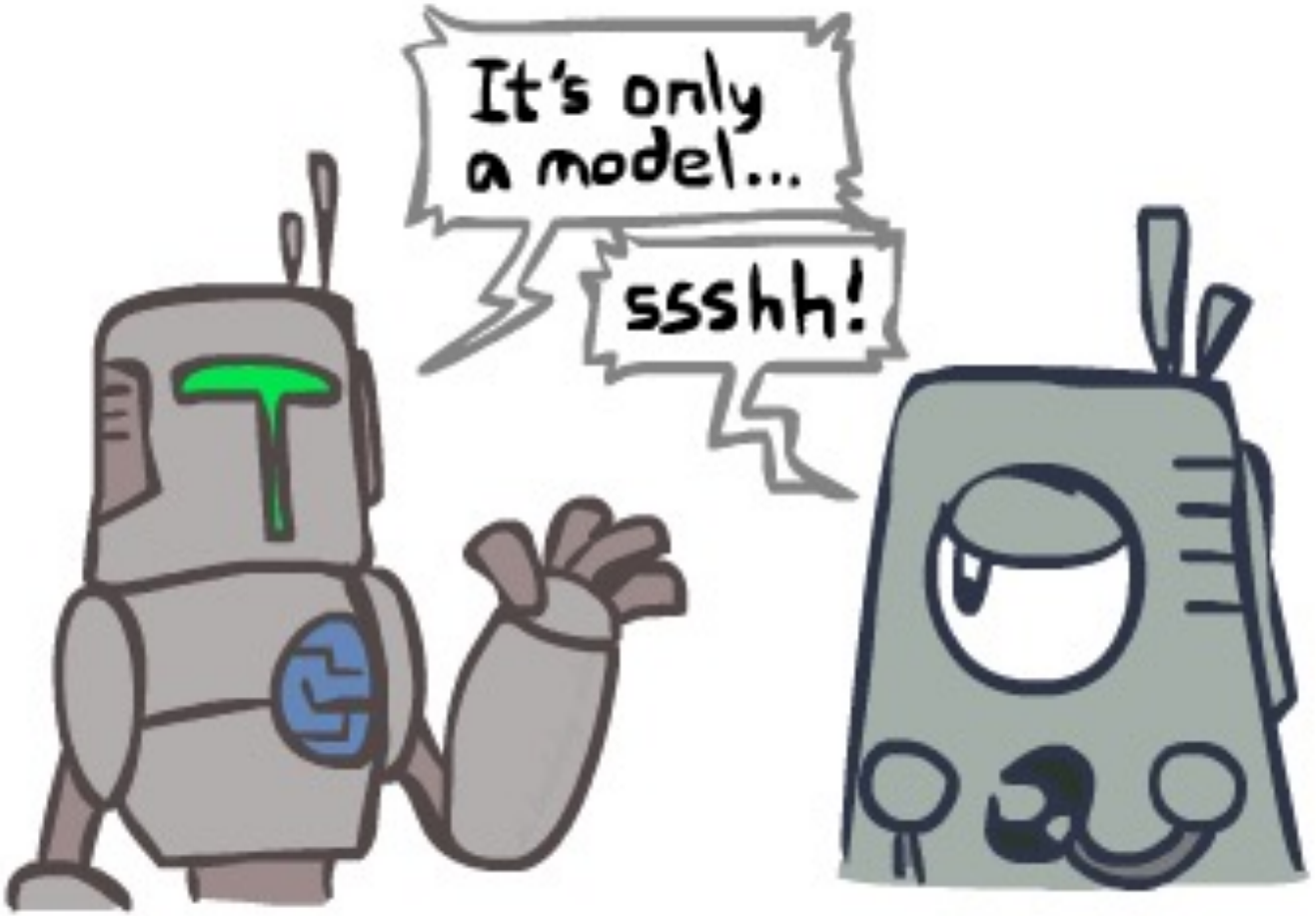
A*: Summary

- A* uses both backward costs and (estimates of) forward costs
- A* is optimal with admissible / consistent heuristics
- Heuristic design is key: often use relaxed problems



Search and Models

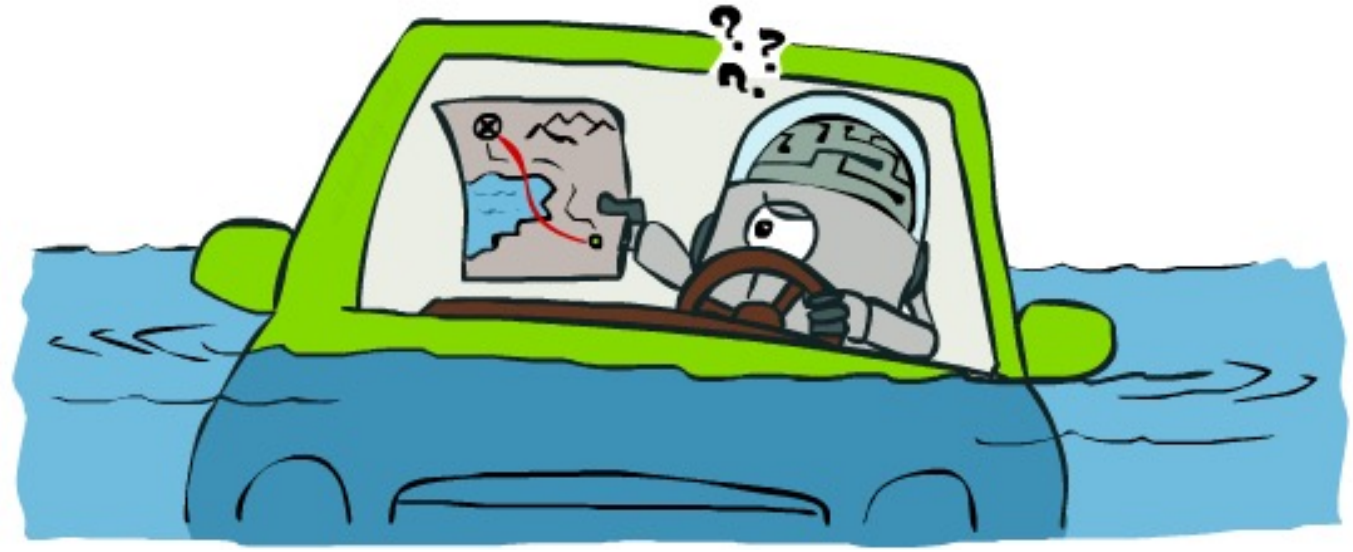
- Search operates over models of the world
  - The agent doesn't actually try all the plans out in the real world!
  - Planning is all "in simulation"
  - Your search is only as good as your models…



Search Gone Wrong?

Start: Haugesund, Rogaland, Norway
End: Trondheim, Sør-Trøndelag, Norway
Total Distance: 2713.2 Kilometers
Estimated Total Time: 47 hours, 31 minutes

Search Gone Wrong?



Appendix: Search Pseudo-Code

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
```