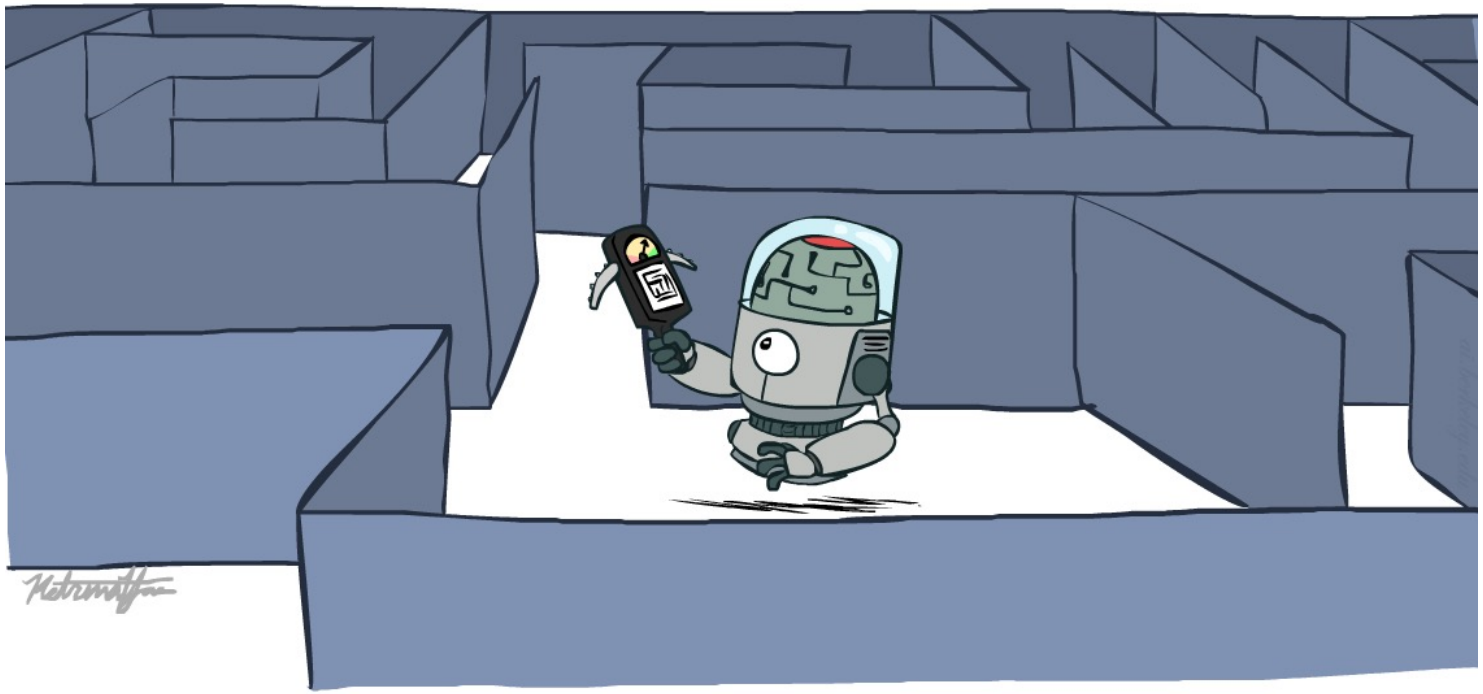
CSEP 573: Artificial Intelligence

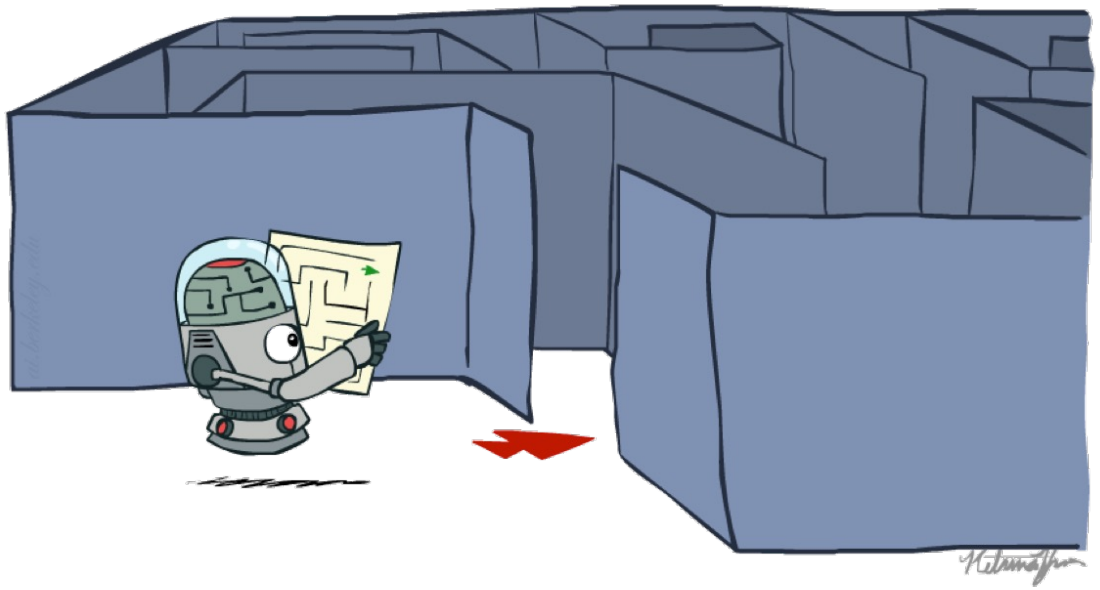
Informed Search



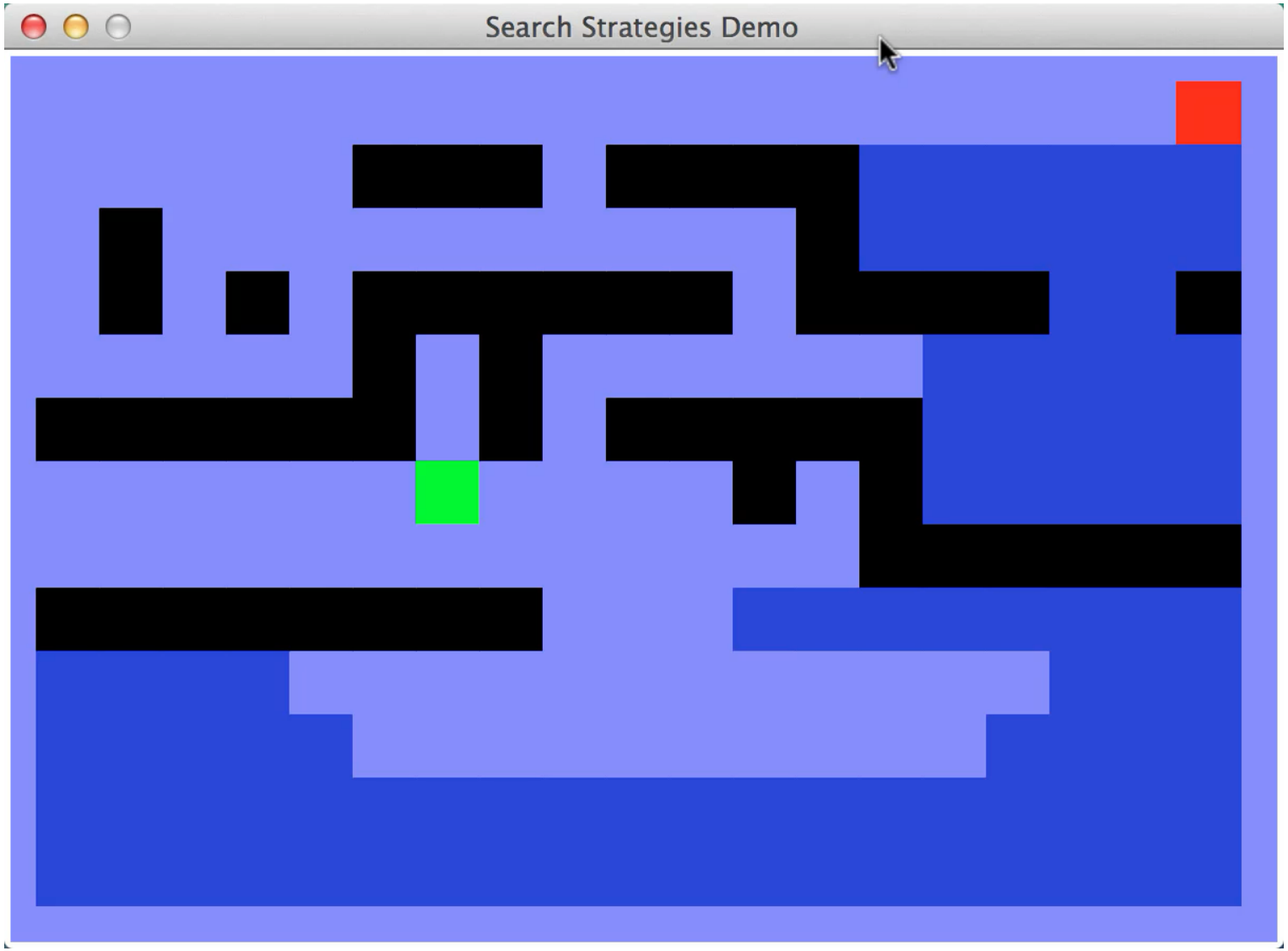
slides adapted from
Stuart Russel, Dan Klein, Pieter Abbeel from ai.berkeley.edu
And Hanna Hajishirzi, Jared Moore, Dan Weld

Recap: Search

- Search problem:
  - States (configurations of the world)
  - Actions and costs
  - Successor function (world dynamics)
  - Start state and goal test
- Search tree:
  - Nodes: represent plans for reaching states
- Search algorithm:
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: finds least-cost plans

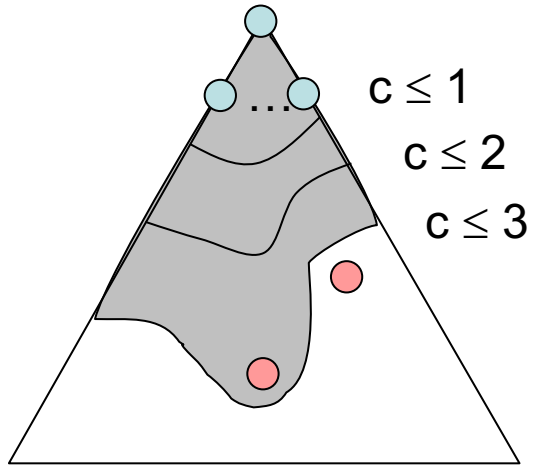


Video of Demo Maze with Deep/Shallow Water --- BFS or UCS? (part 2)



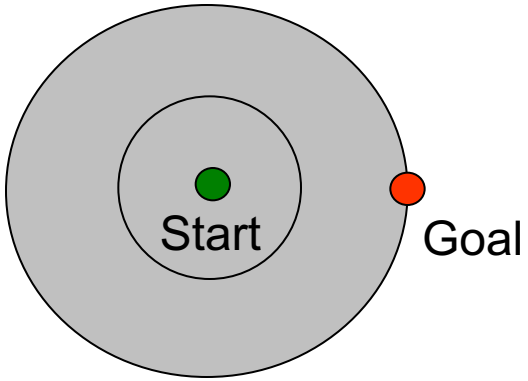
Uniform Cost Issues

- Remember: UCS explores increasing cost contours



- The good: UCS is complete and optimal!

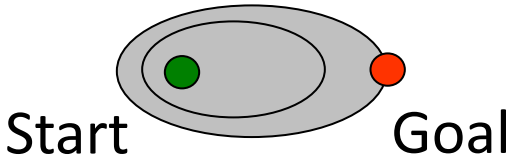
- The bad:
  - Explores options in every “direction”
  - No information about goal location



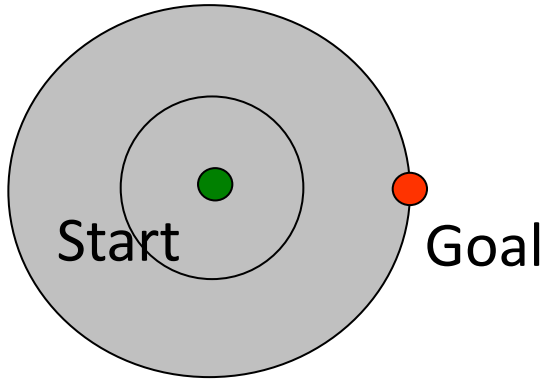
- We’ll fix that in this lecture!

What we would like to have happen

Guide search ***towards the goal*** instead of ***all over the place***



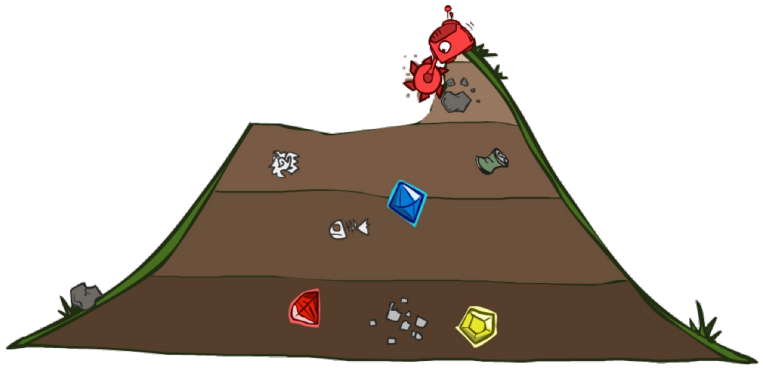
Informed



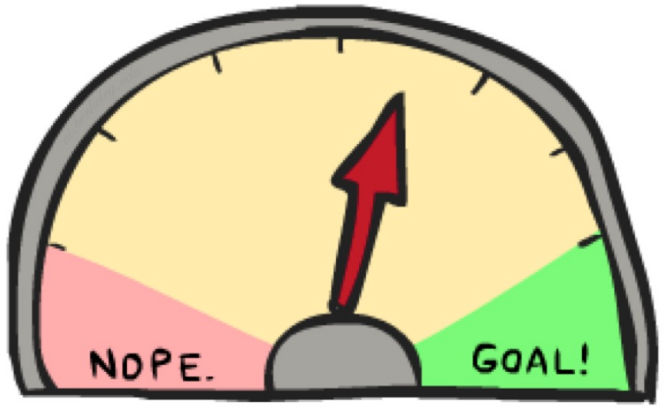
Uninformed

Up next: Informed Search

- Uninformed Search
  - DFS
  - BFS
  - UCS

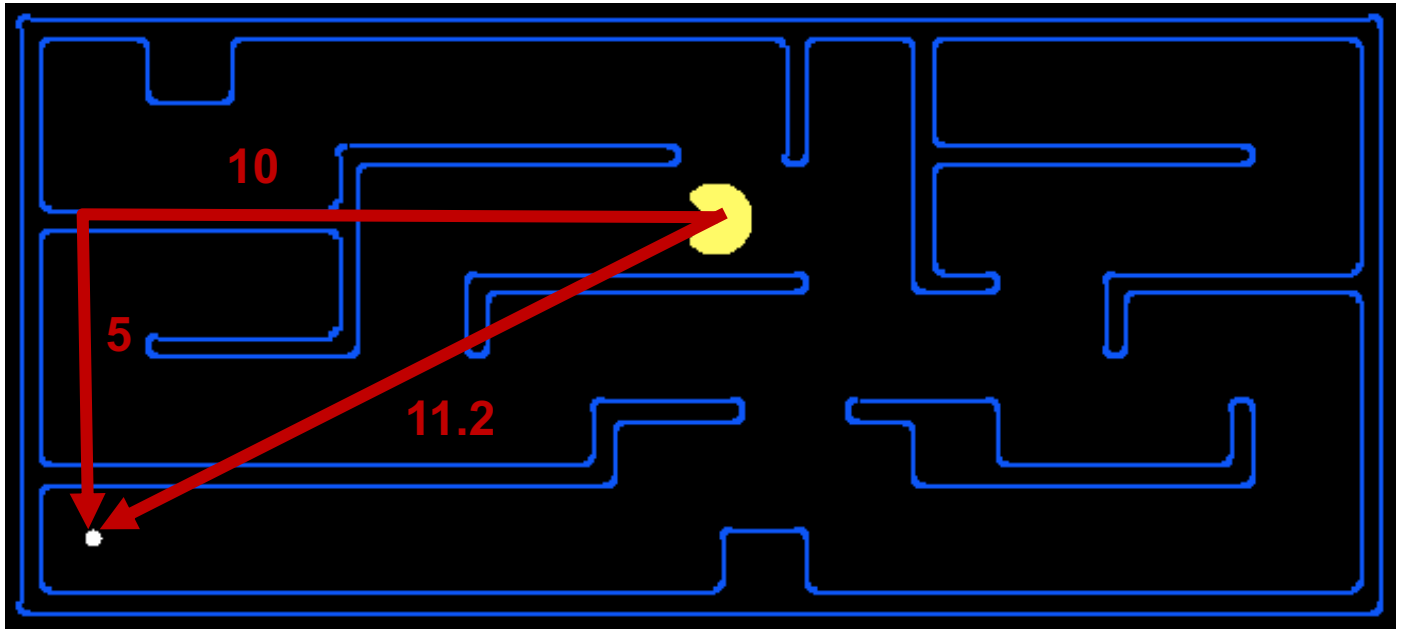


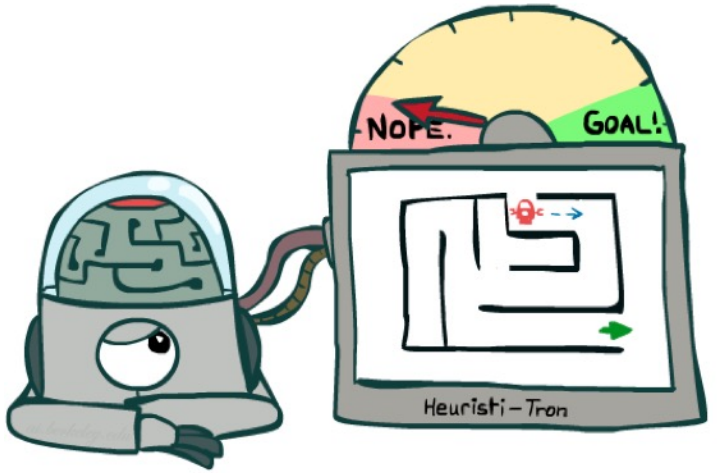
- Informed Search
  - Heuristics
  - Greedy Search
  - A* Search
  - Graph Search

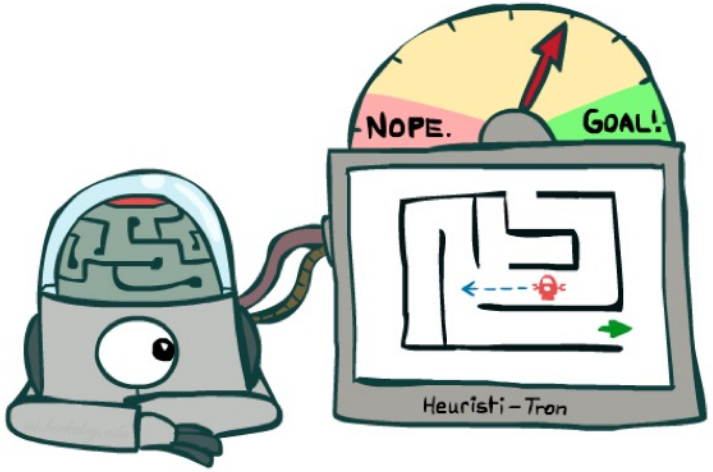


Search Heuristics

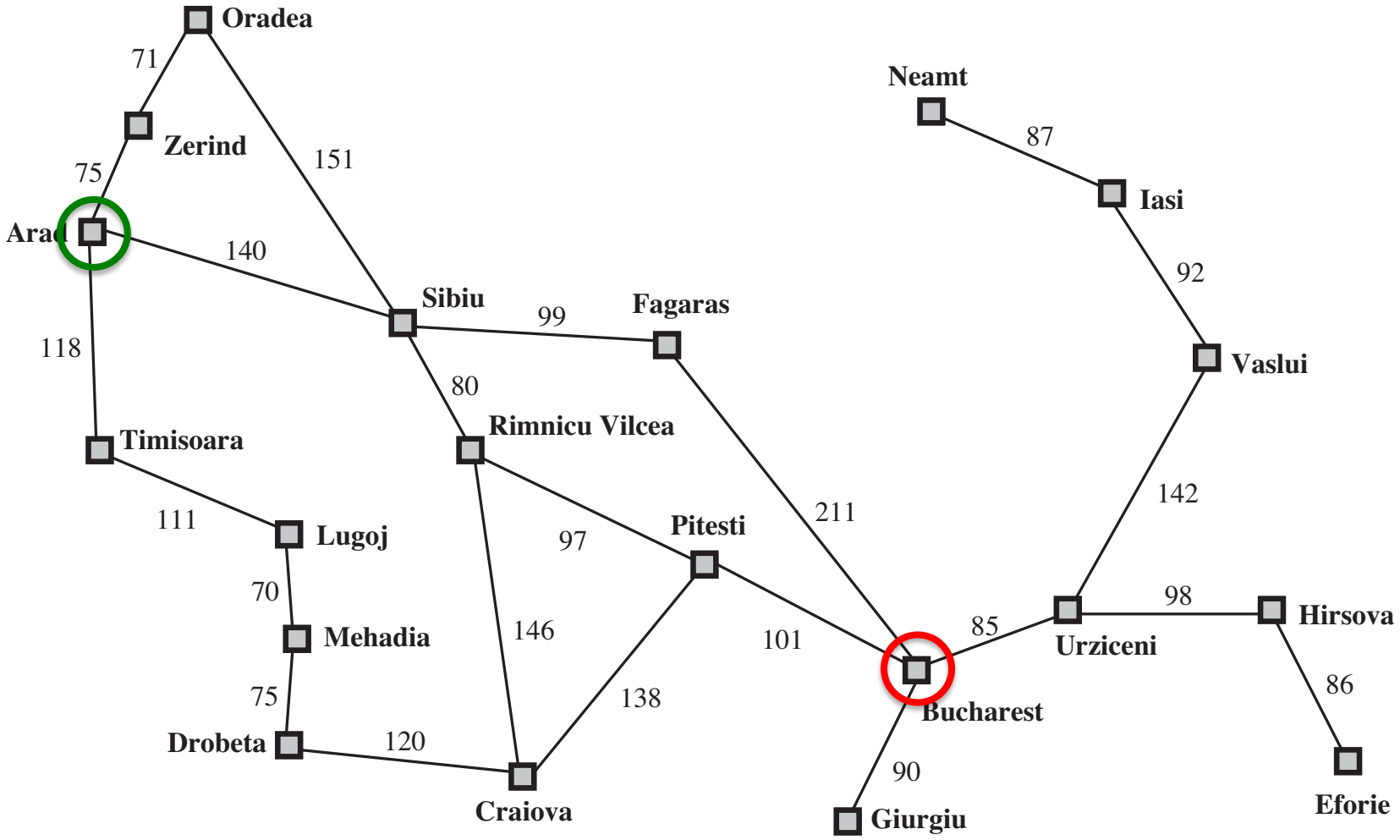
- A heuristic is:
  - A function that *estimates* how close a state is to a goal
  - Designed for a particular search problem
  - Pathing?
  - Examples: Manhattan distance, Euclidean distance for pathing



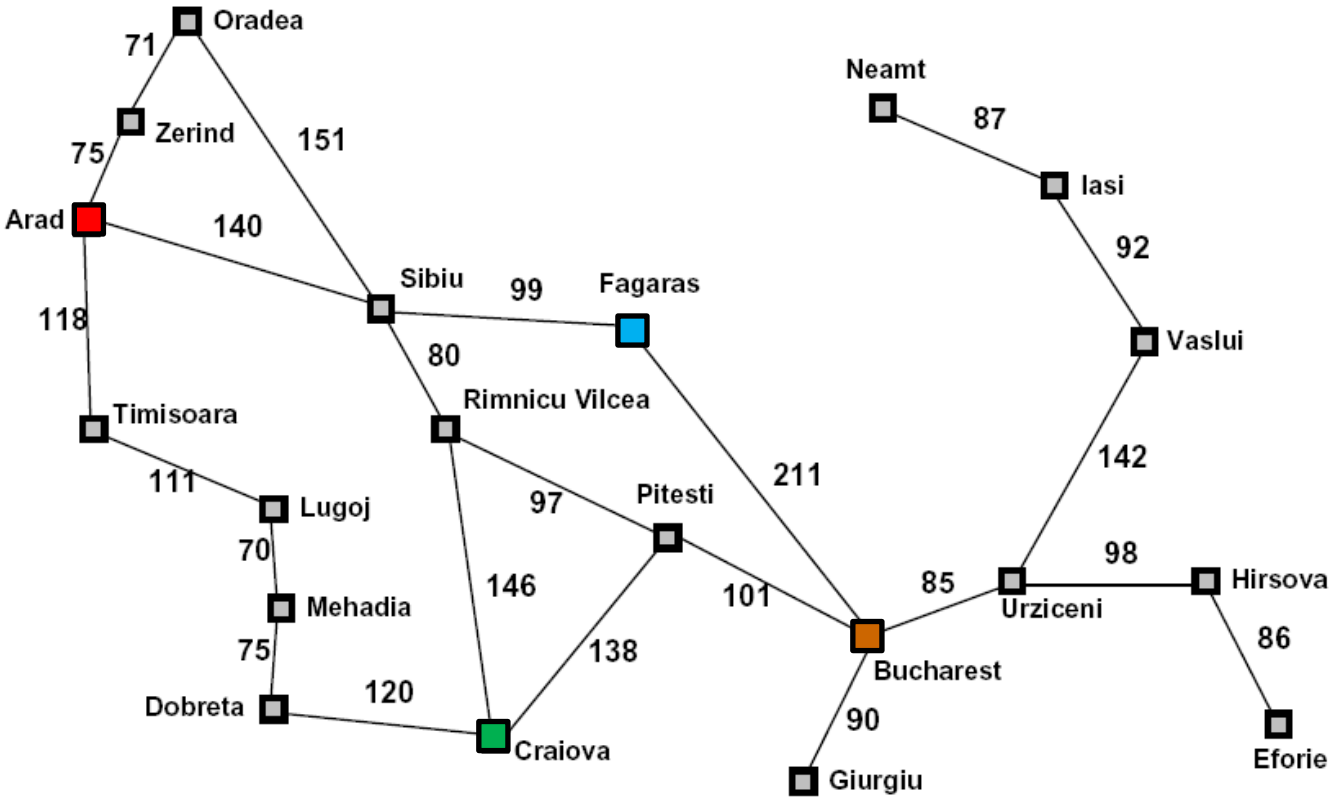




Example: route-finding in Romania



Example: Heuristic Function

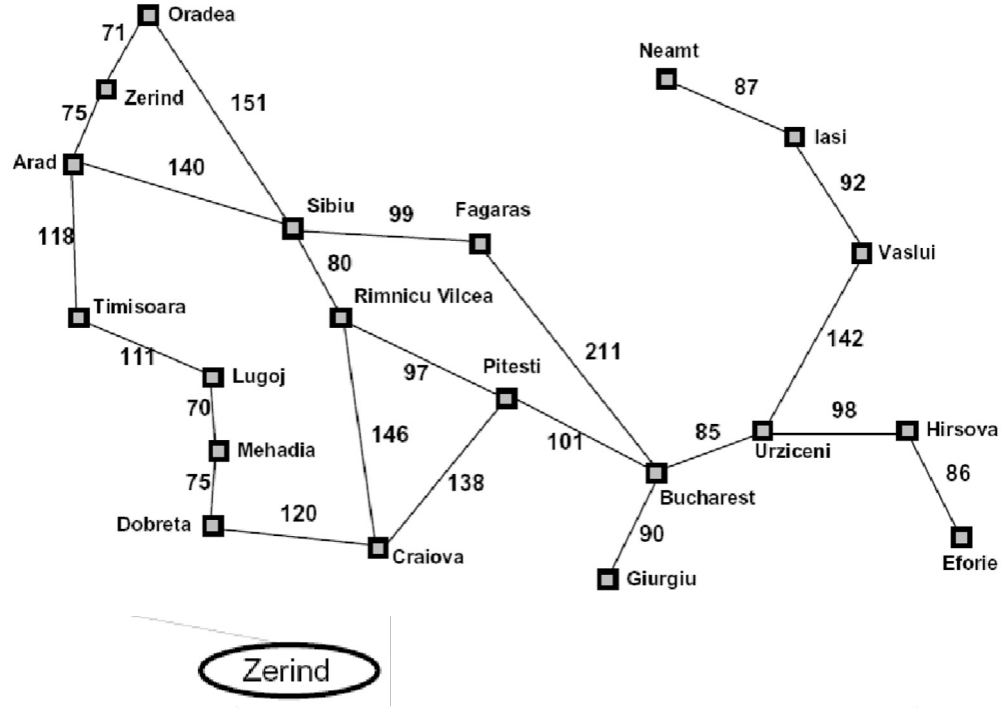


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

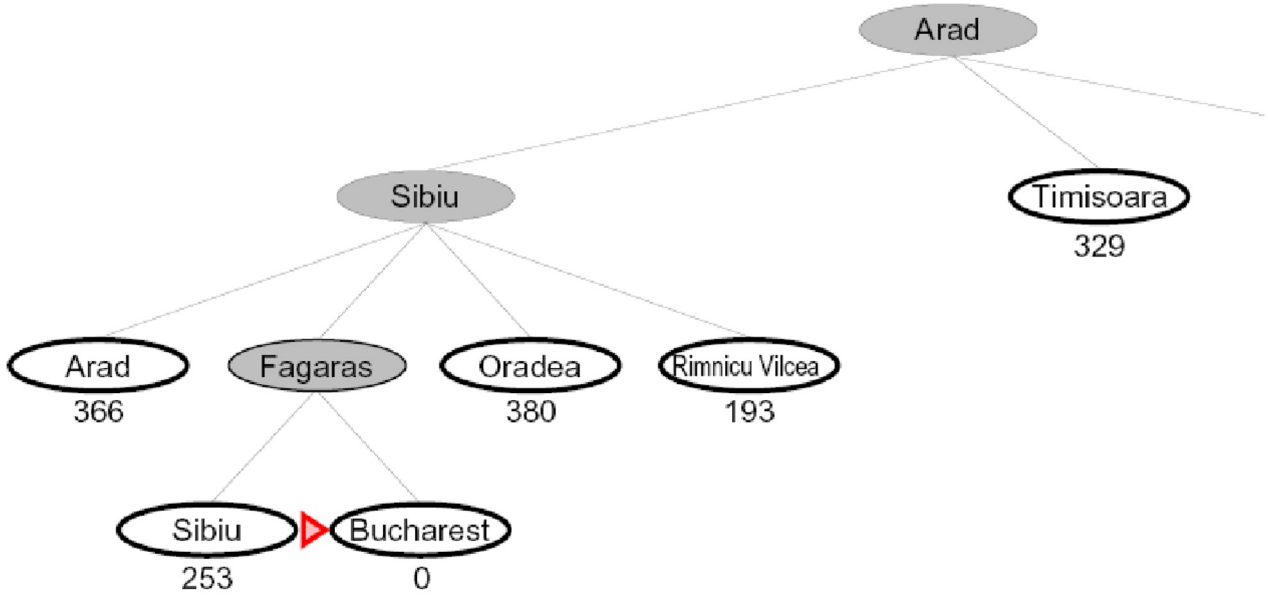
h(x)

Greedy Search

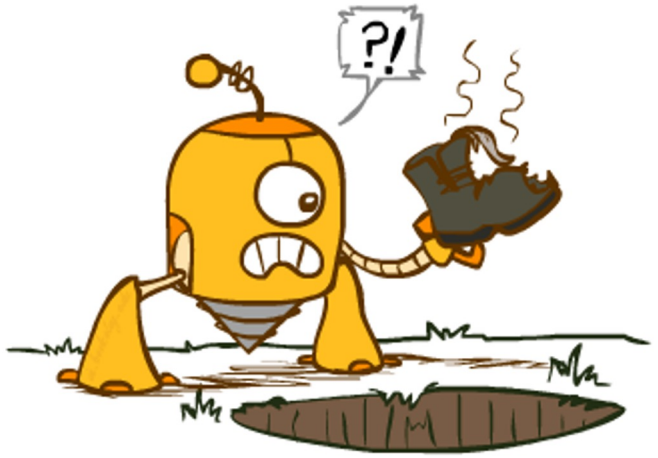
Greedy Search



Expand the node that seems closest…



Zerind



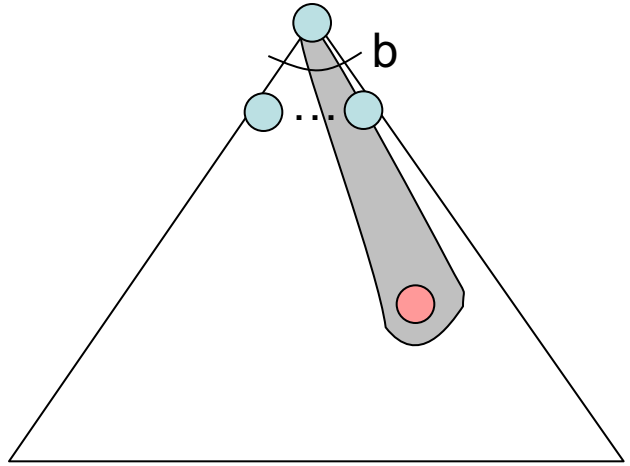
Is it optimal?

No. Resulting path to Bucharest is not the shortest!

Greedy Search

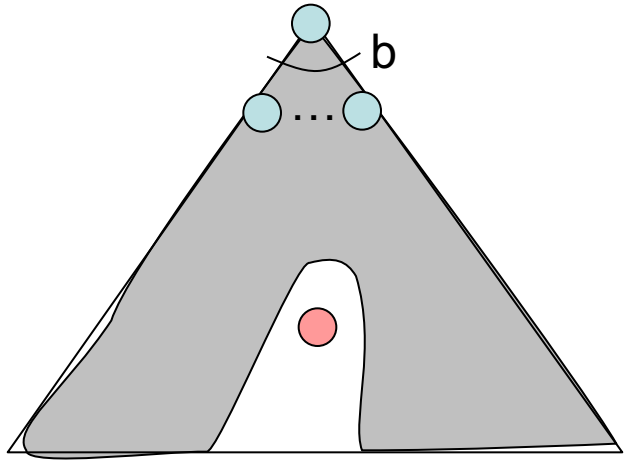
Strategy: expand a node that you think is closest to a goal state

- Heuristic: estimate of distance to nearest goal for each state



A common case:

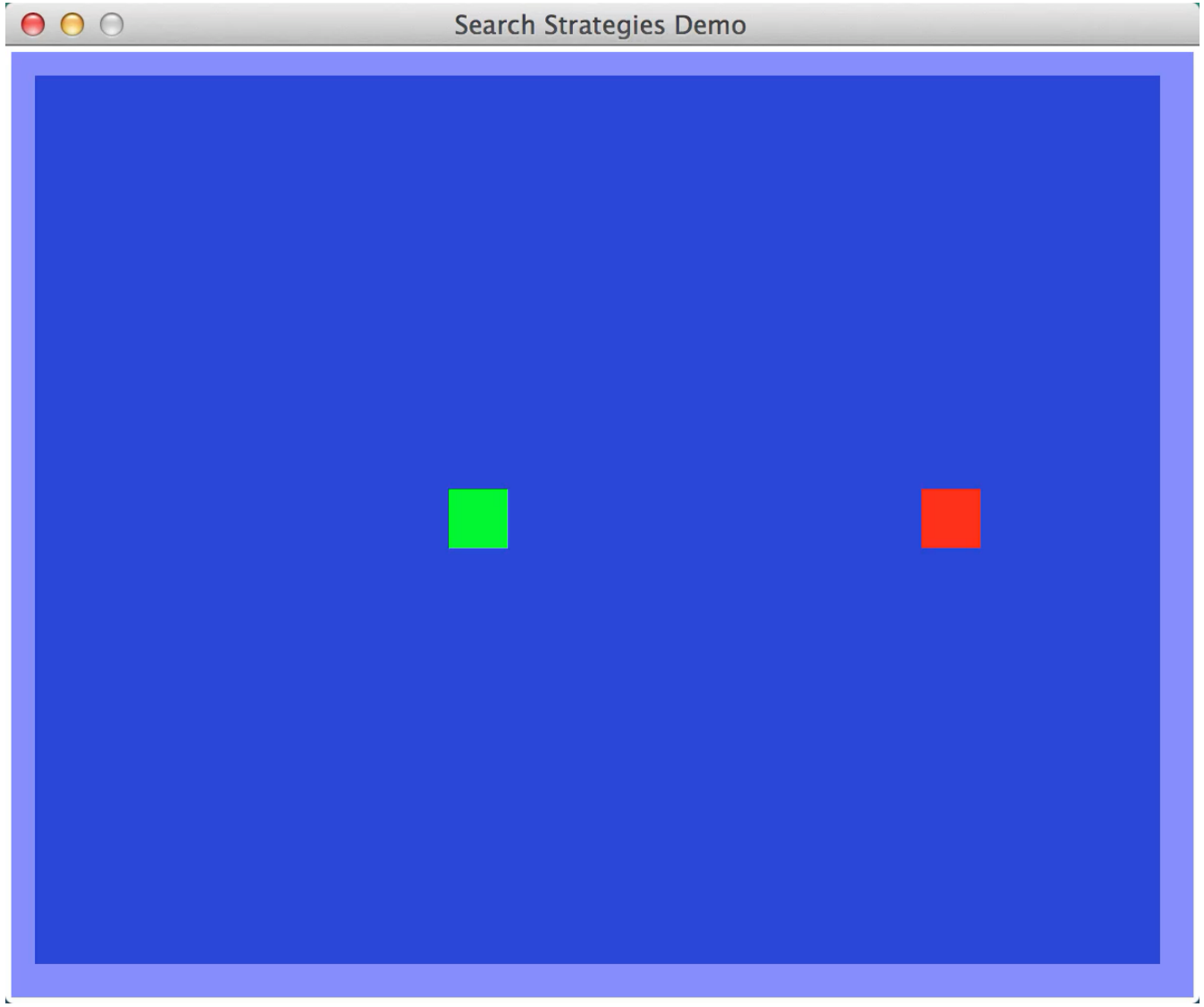
- Best-first takes you straight to the (wrong) goal



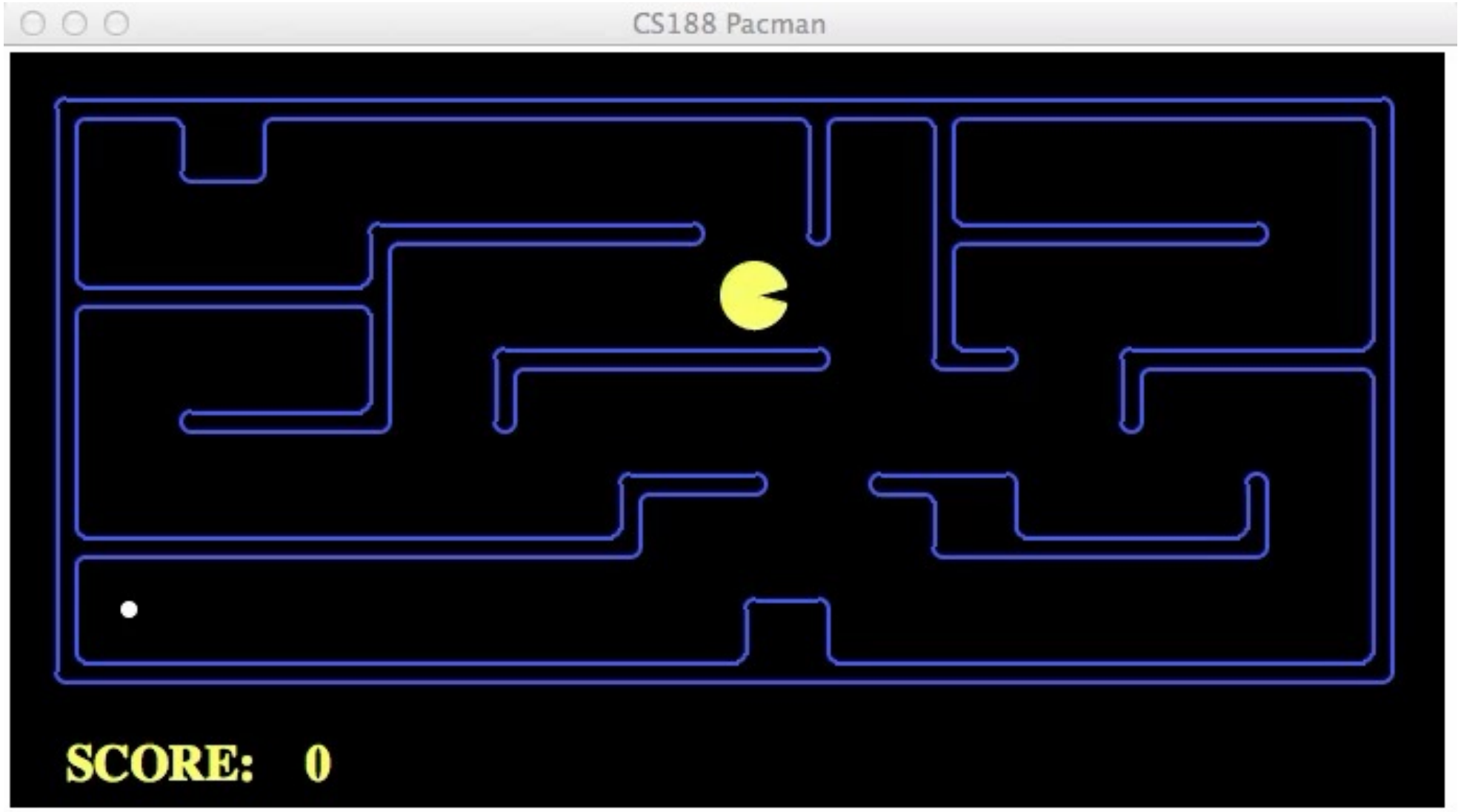
Worst-case: like a badly-guided DFS

```
python pacman.py --layout smallMaze --pacman=GreedyAgent
python pacman.py --layout smallMaze --pacman=SearchAgent -a fn=dfs
```

Video of Demo Contours Greedy (Empty)



Video of Demo Contours Greedy (Pacman Small Maze)

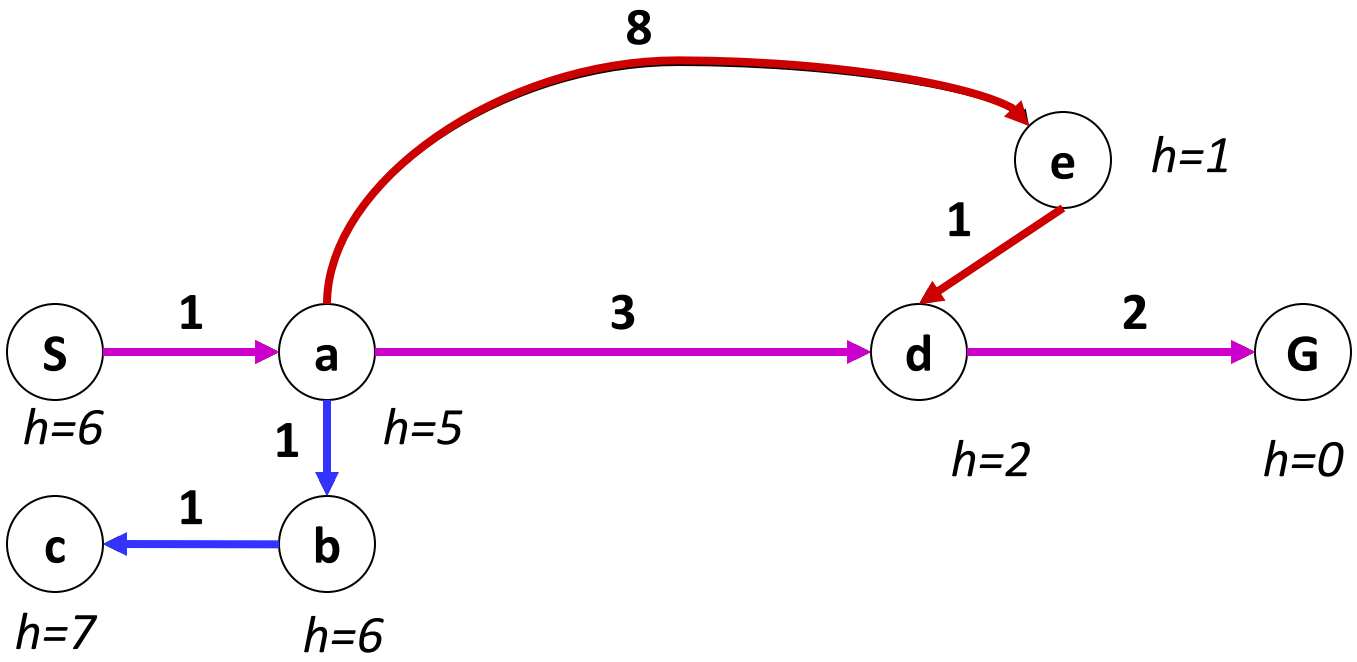


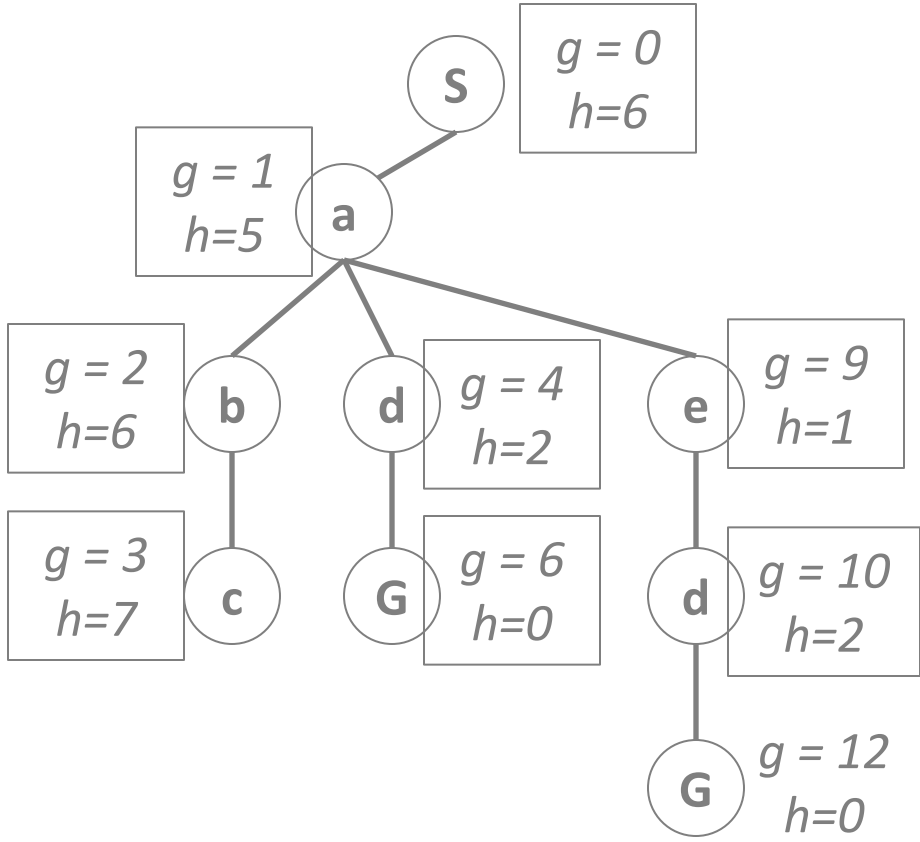
A* Search

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)

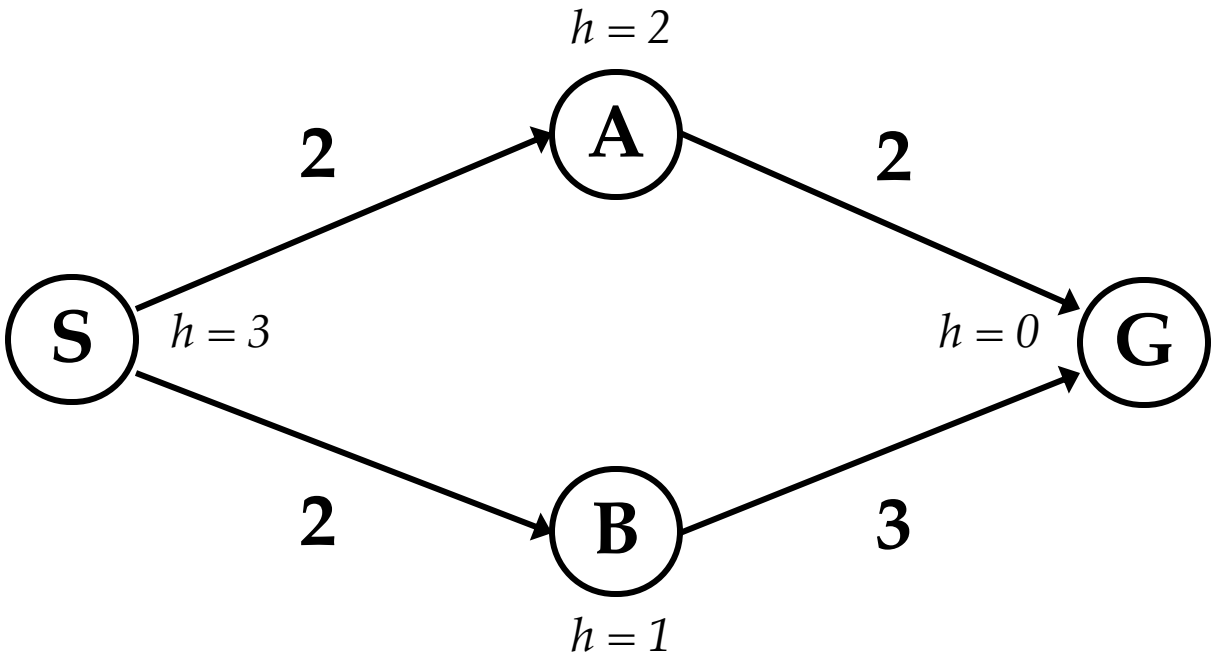
Example: Teg Grenager

A*: the core idea

- Expand a node *n* most likely to be on the optimal path
- Expand a node *n* s.t. the cost of the best solution through *n* is optimal
- Expand a node *n* with lowest value of $g(n) + h^*(n)$
  - $g(n)$ is the cost from root to *n*
  - $h^*(n)$ is the optimal cost from *n* to the closest goal
- We seldom know $h^*(n)$ but might have a heuristic approximation $h(n)$
- A* = tree search with priority queue ordered by $g(n) + h(n)$

When should A* terminate?

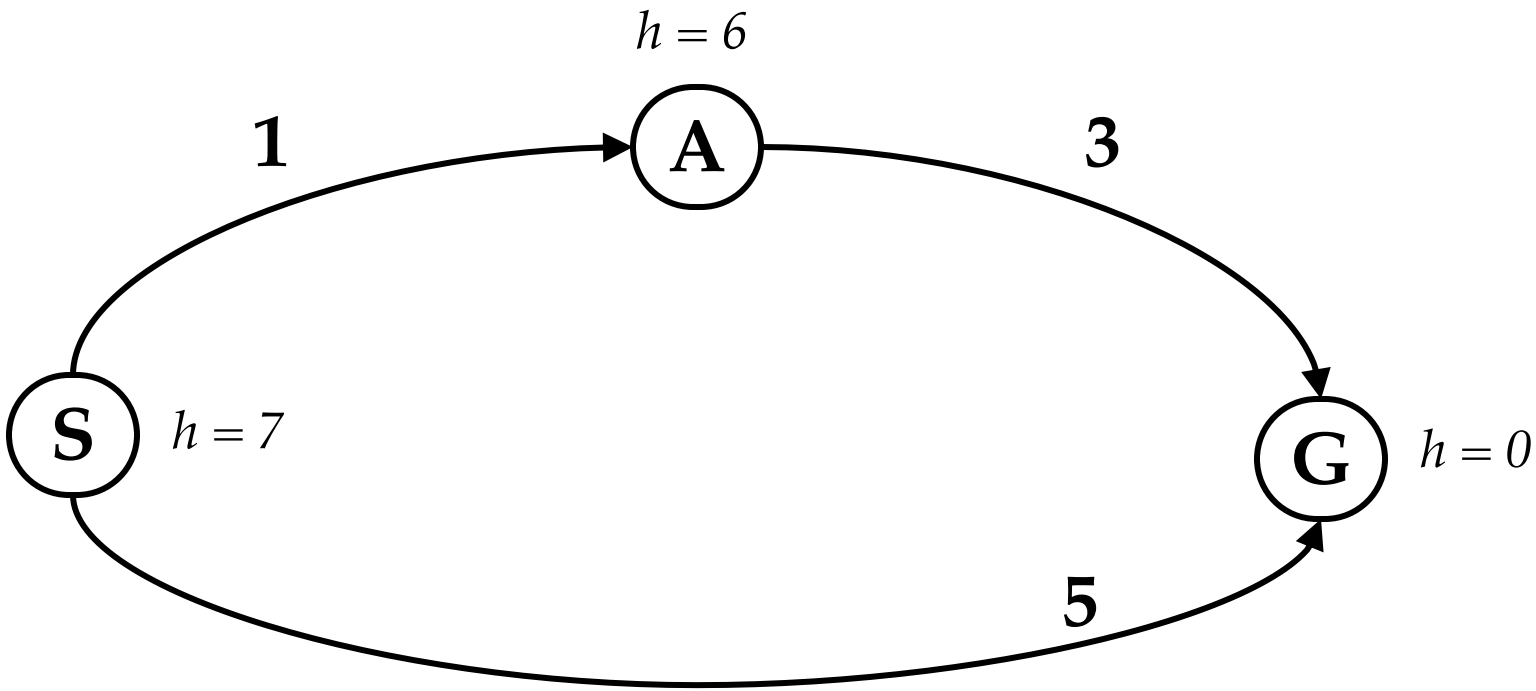
- Should we stop when we enqueue a goal?



	g	h	+
~~S~~	~~0~~	~~3~~	~~3~~
~~S->A~~	~~2~~	~~2~~	~~4~~
~~S->B~~	~~2~~	~~1~~	~~3~~
S->B->G	5	0	5
S->A->G	4	0	4

- No: only stop when we dequeue a goal

Is A* Optimal?

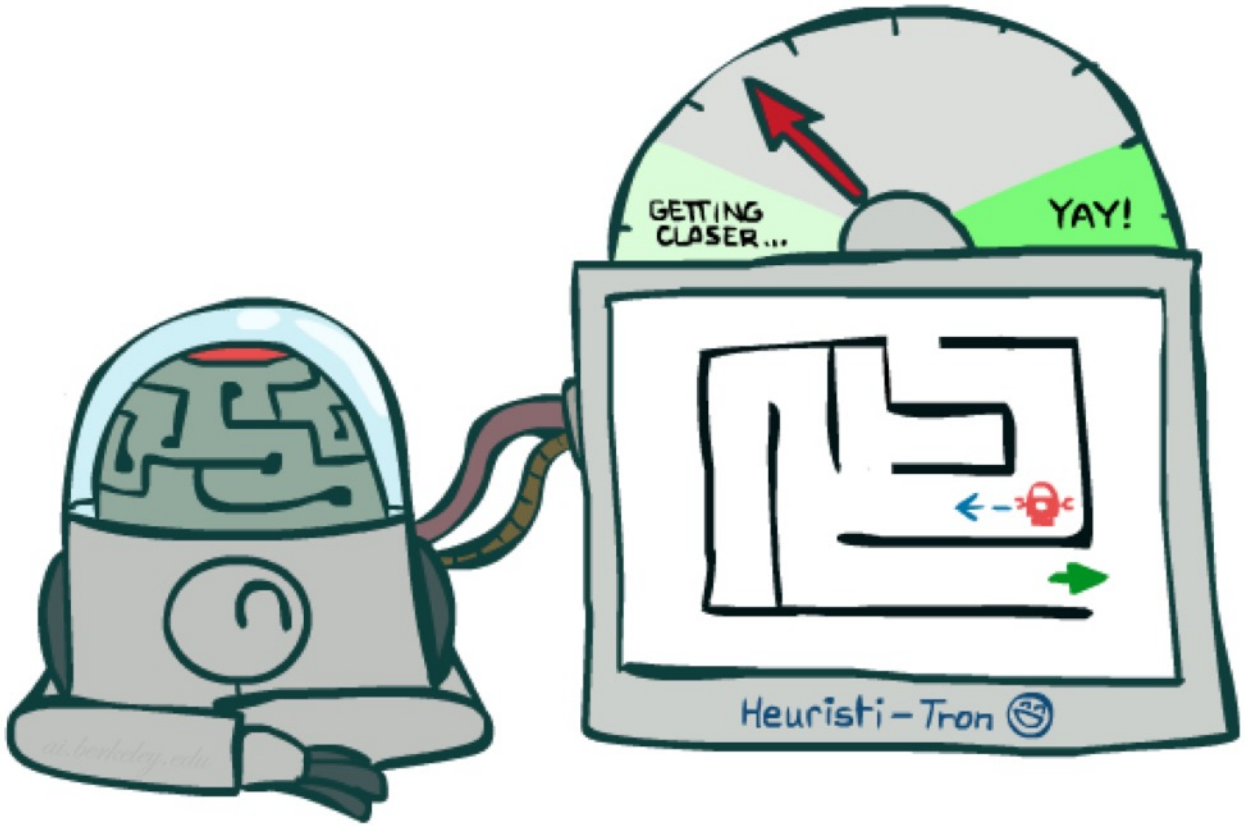


	g, h, f
~~S~~	~~0 7 7~~
S->A	1 6 7
S->G	5 0 5

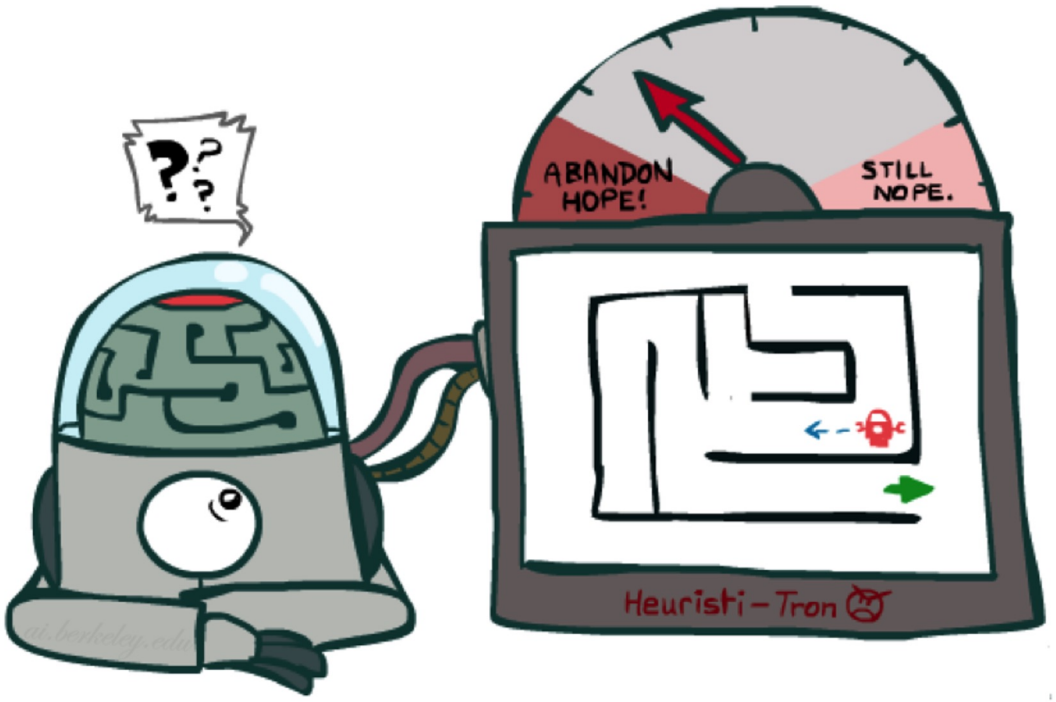
What went wrong?

- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!

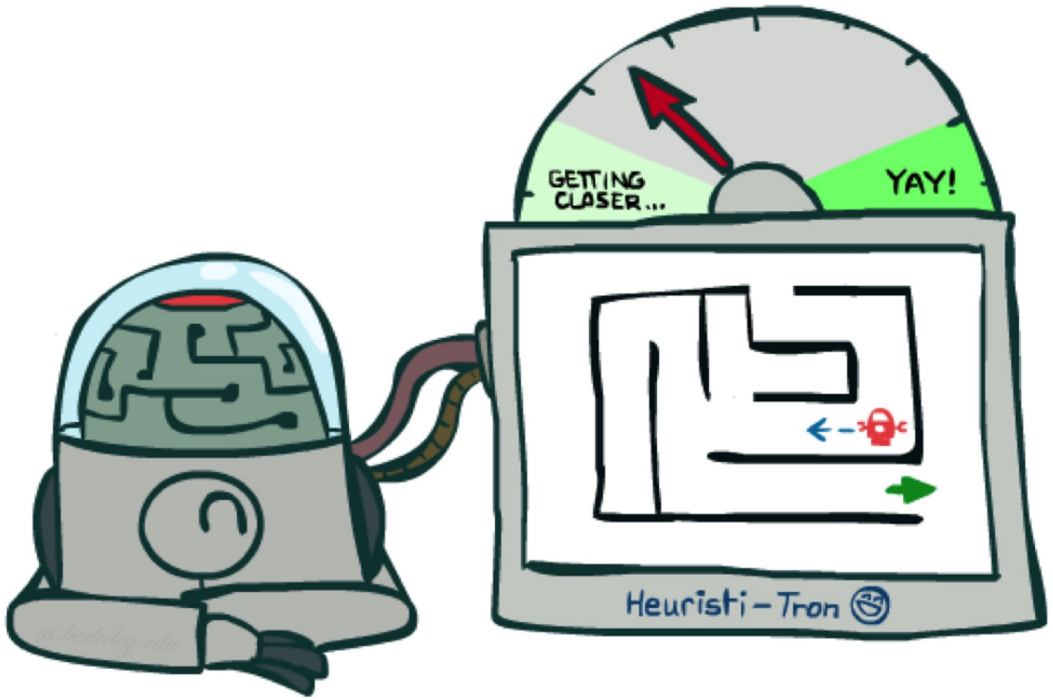
Admissible Heuristics



Idea: Admissibility



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



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

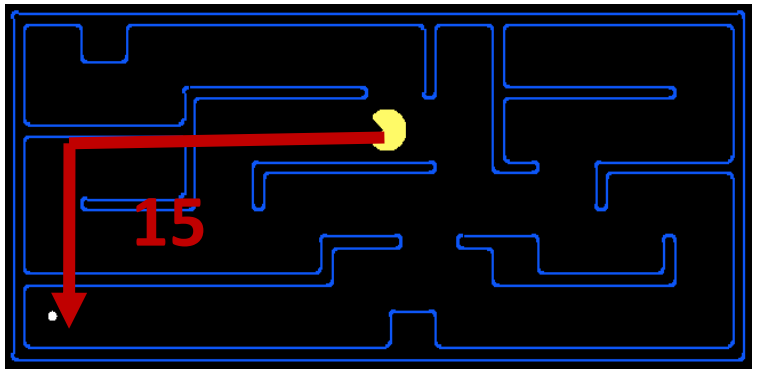
Admissible Heuristics

- 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

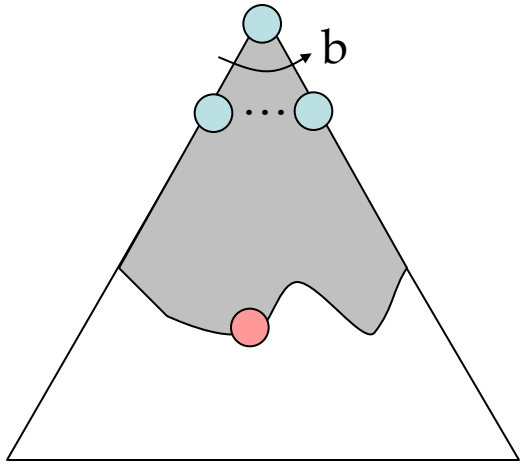
- Example:



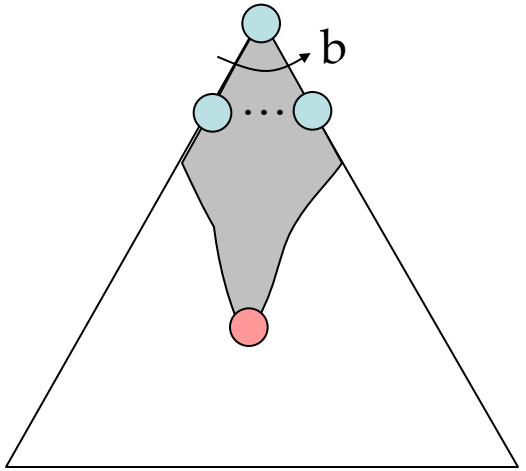
- Finding good, cheap admissible heuristics is the key to success

Properties of A*

Uniform-Cost

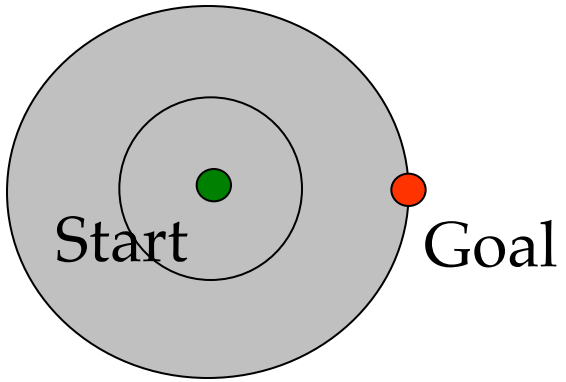


A*

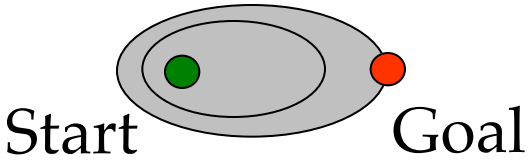


UCS vs A* Contours

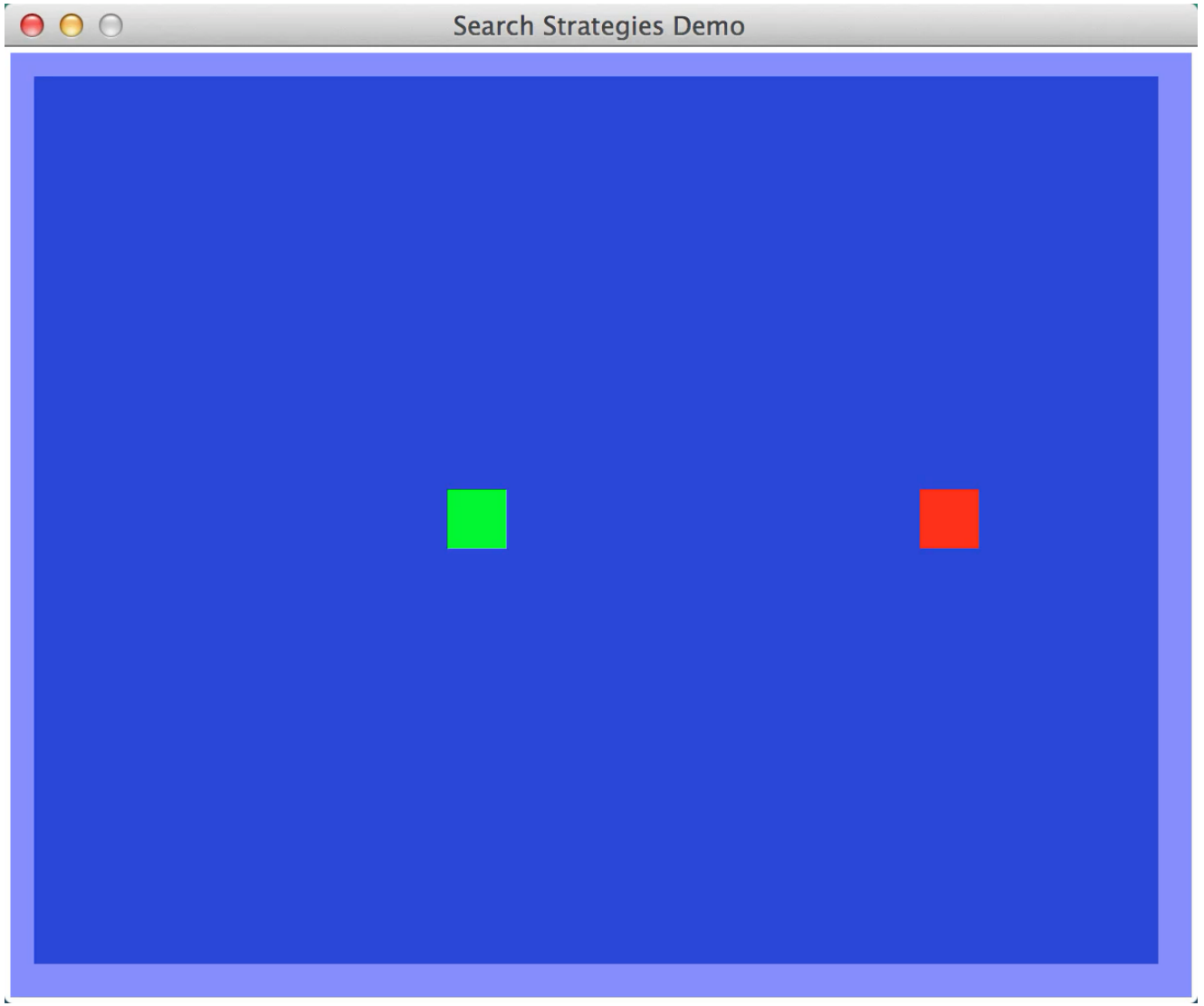
- Uniform-cost expands equally in all "directions"



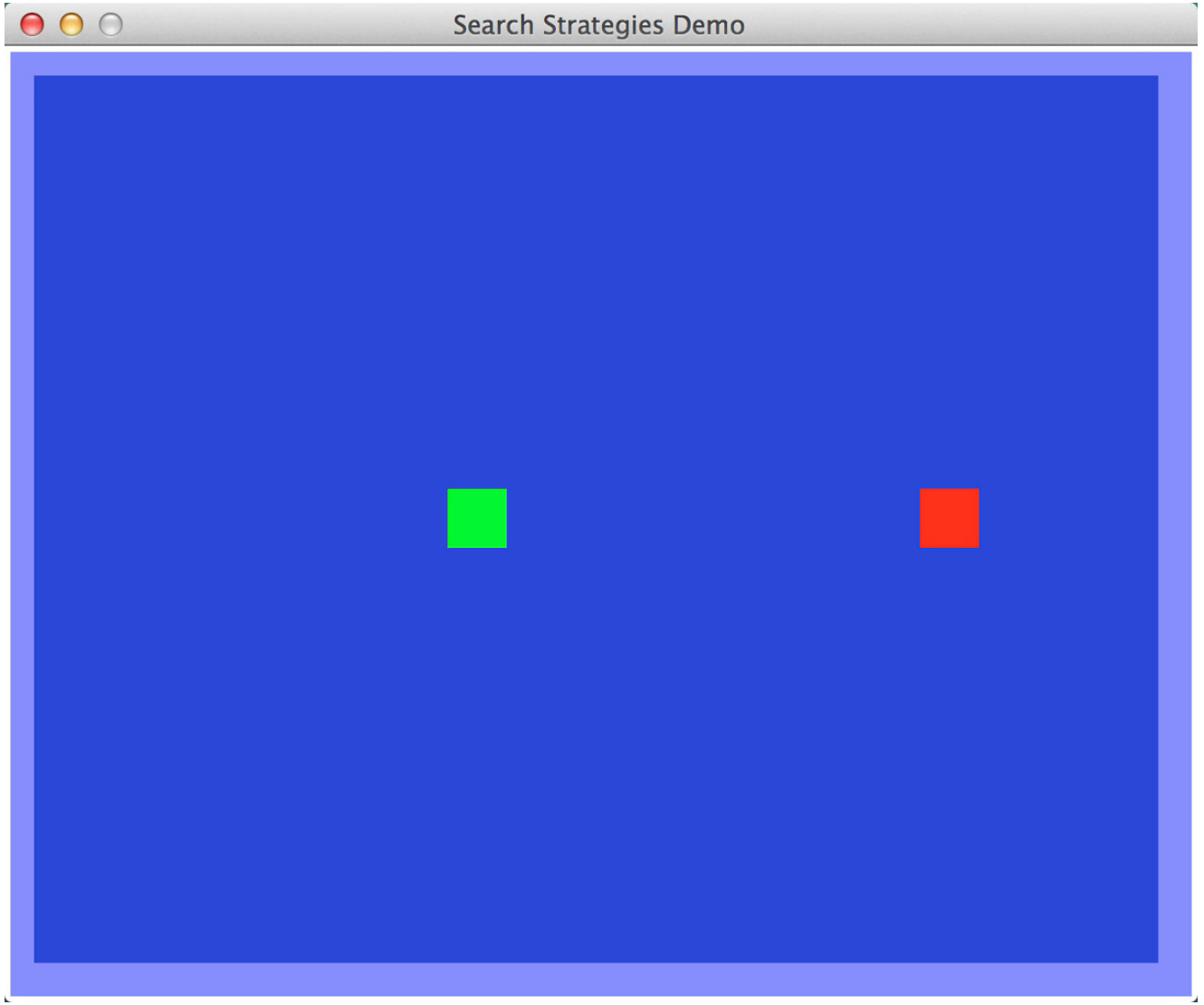
- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



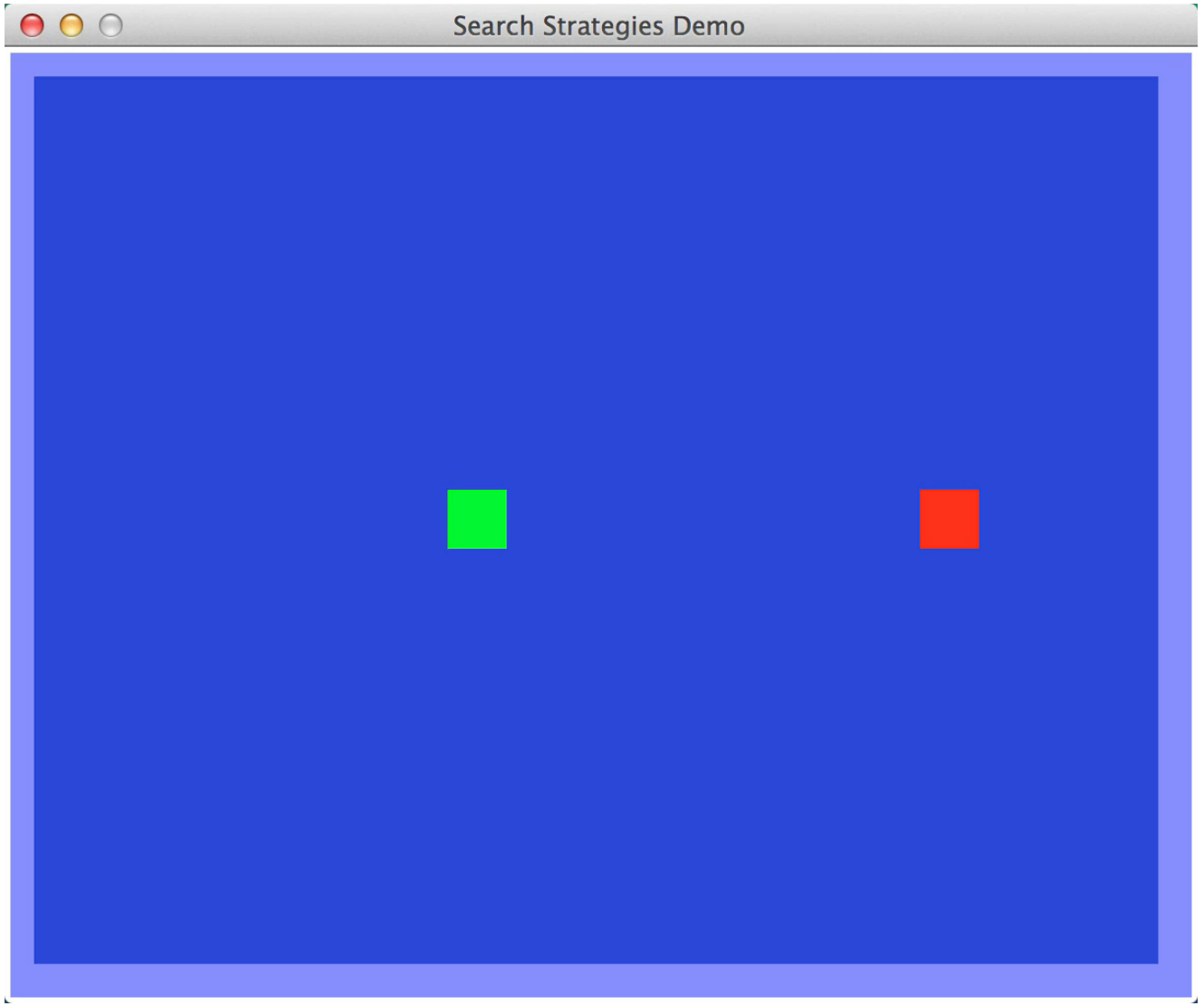
Video of Demo Contours (Empty) -- UCS



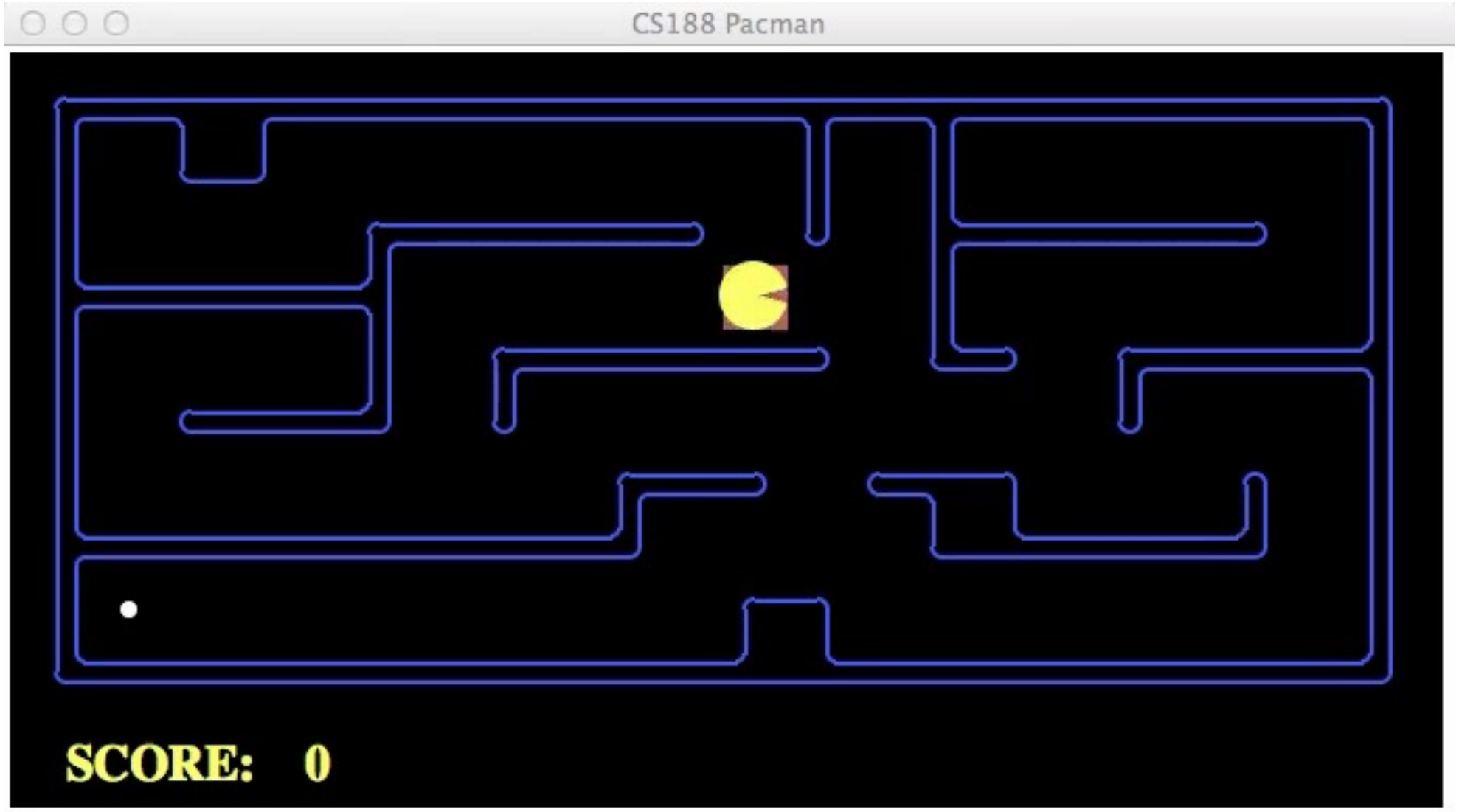
Video of Demo Contours (Empty) -- Greedy



Video of Demo Contours (Empty) – A*



Which Algorithm?



Summary

Greedy

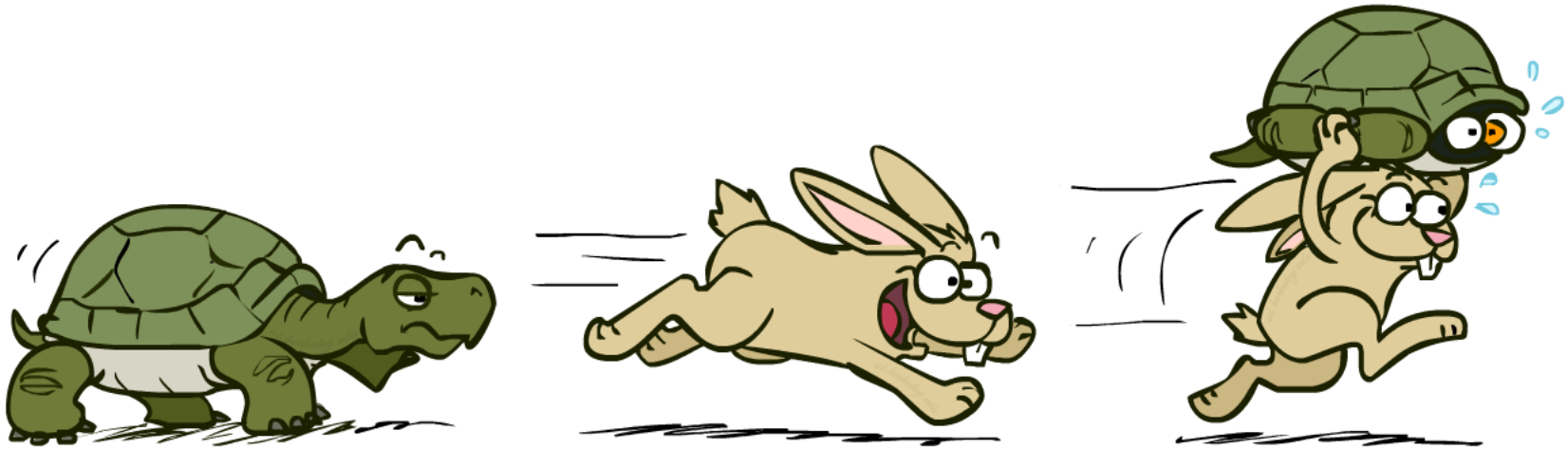
Uniform Cost

A*

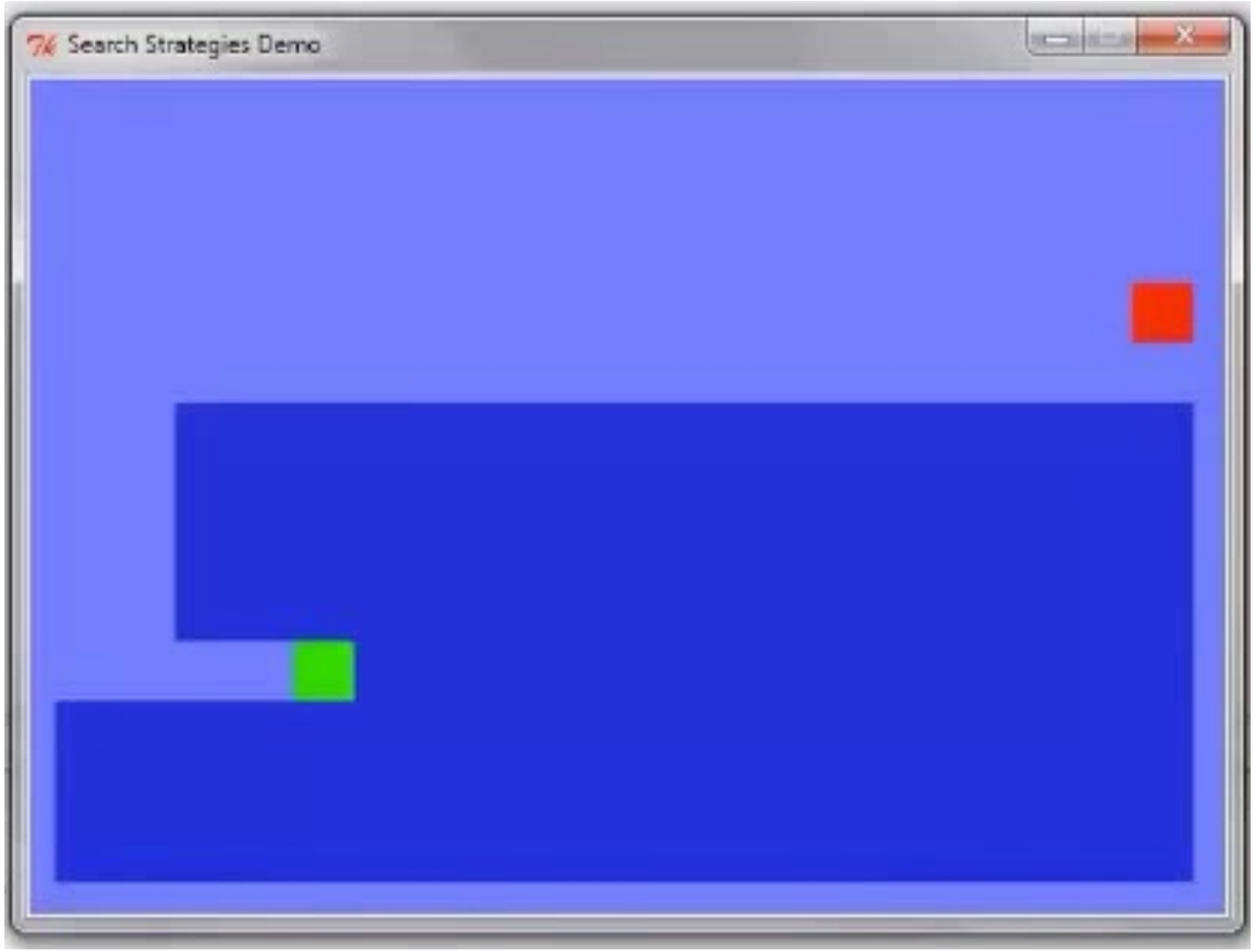
A*: Summary

A*: Summary

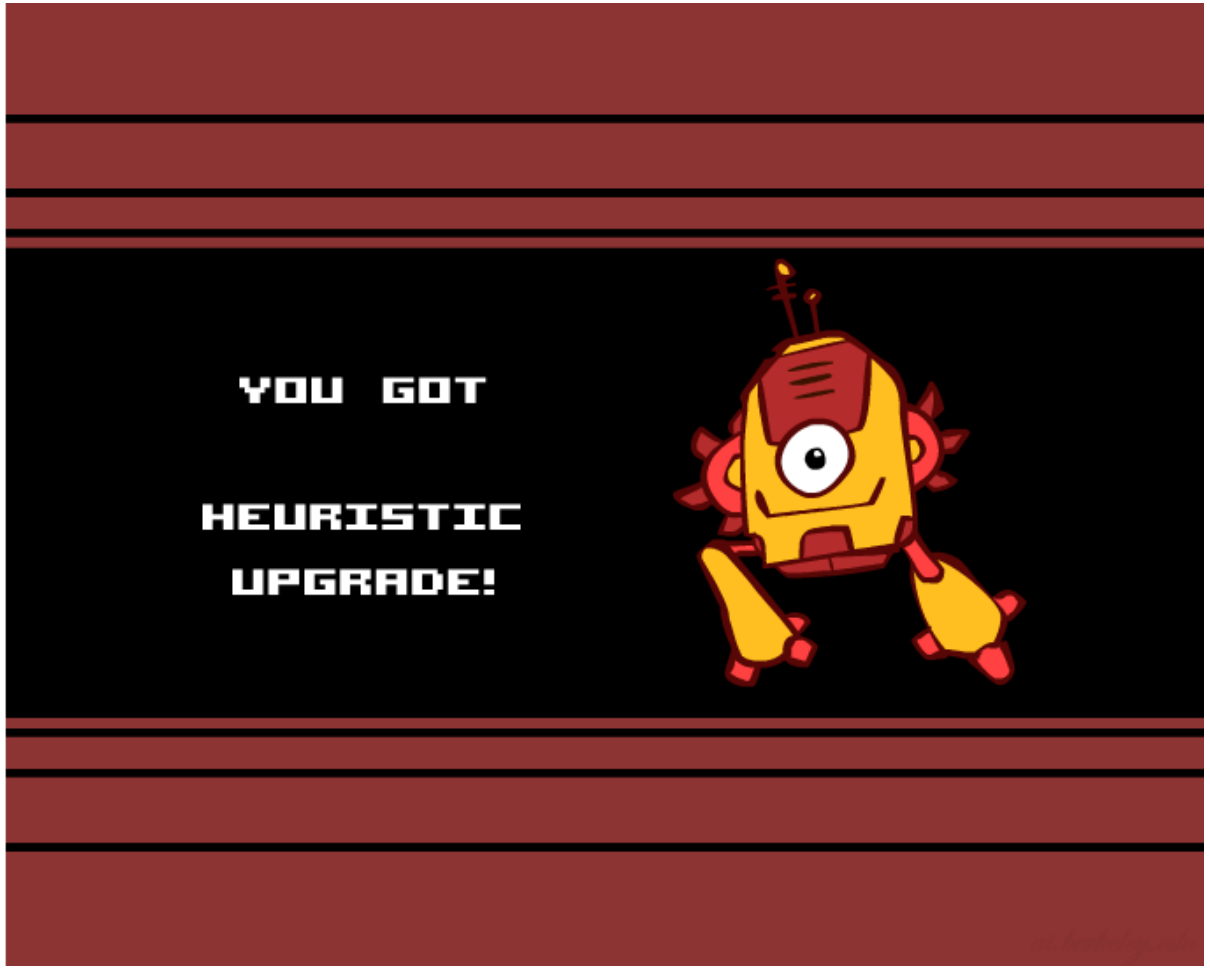
- A* uses both backward costs and (estimates of) forward costs
- A* is optimal with admissible (optimistic) heuristics
- Heuristic design is key: often use relaxed problems



Video of Demo Empty Water Shallow/Deep – Guess Algorithm

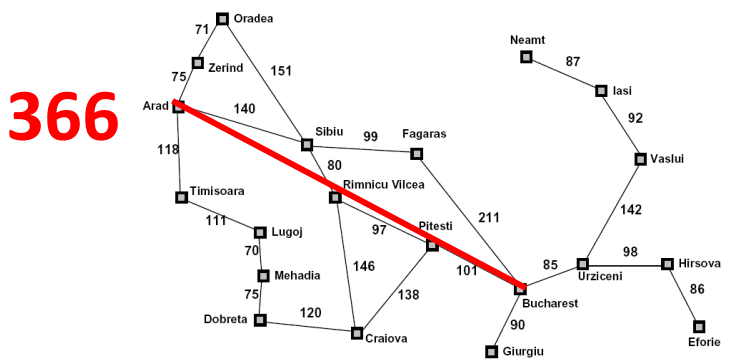


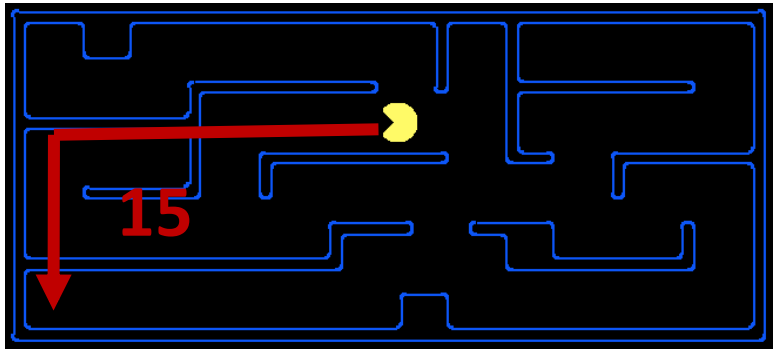
Creating Heuristics



Creating Admissible Heuristics

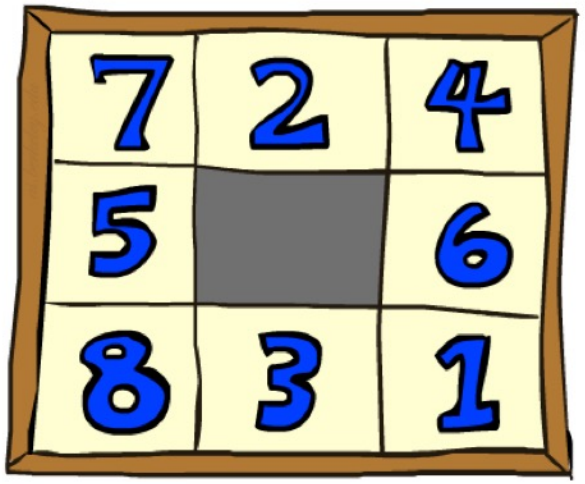
- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
- Often, admissible heuristics are solutions to *relaxed problems,* where new actions are available



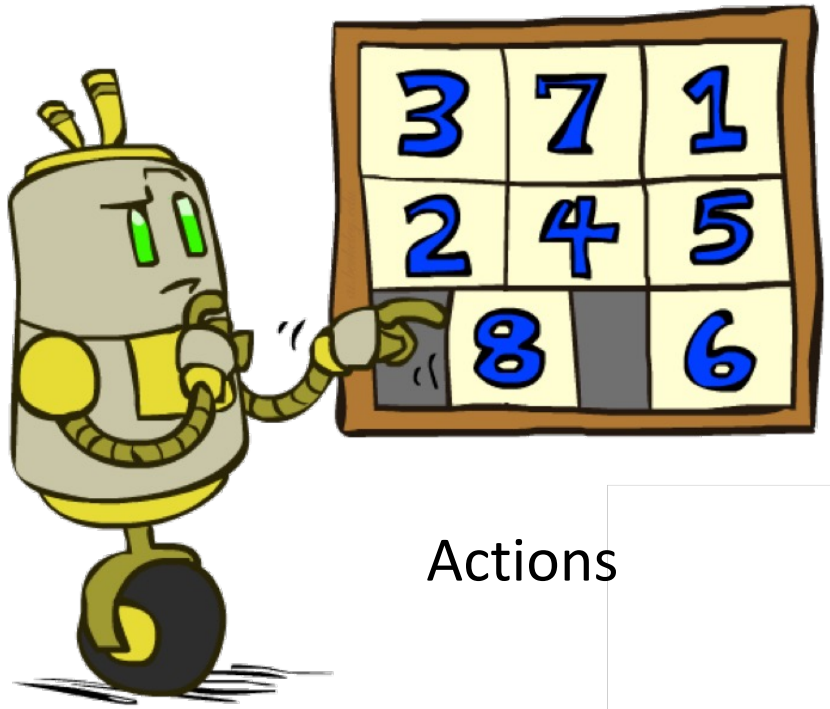


- Inadmissible heuristics are often useful too

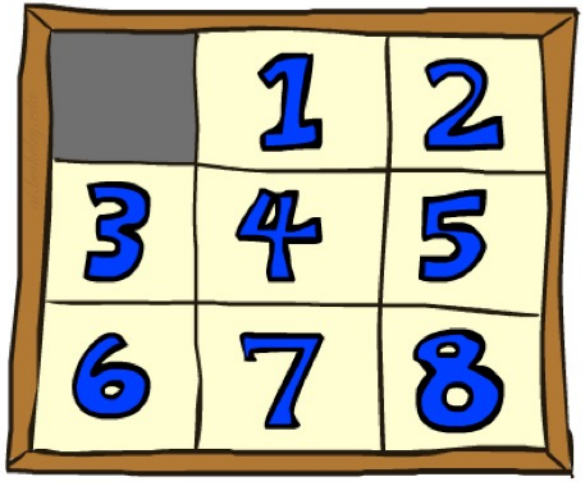
Example: 8 Puzzle



Start State



Actions



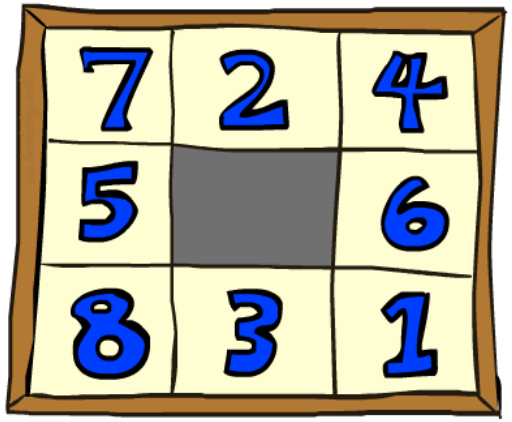
Goal State

- What are the states?
- How many states?
- What are the actions?
- What are the step costs?

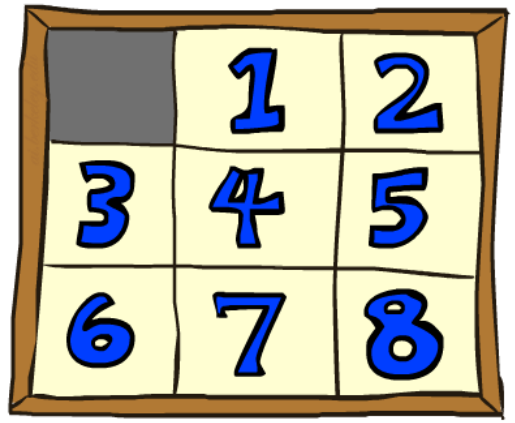
Admissible heuristics?

8 Puzzle I

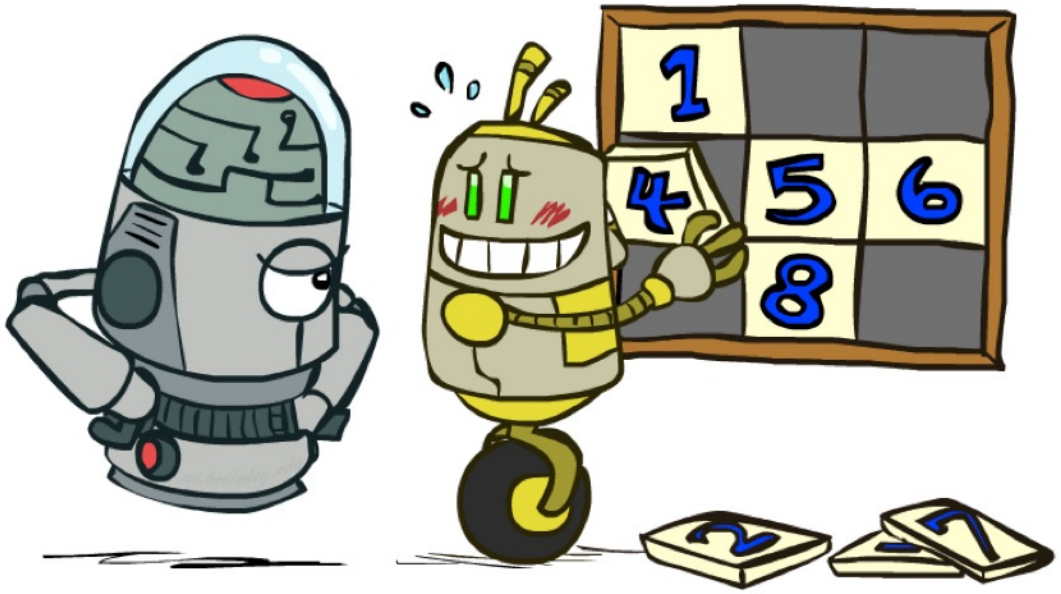
- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8



Start State



Goal State

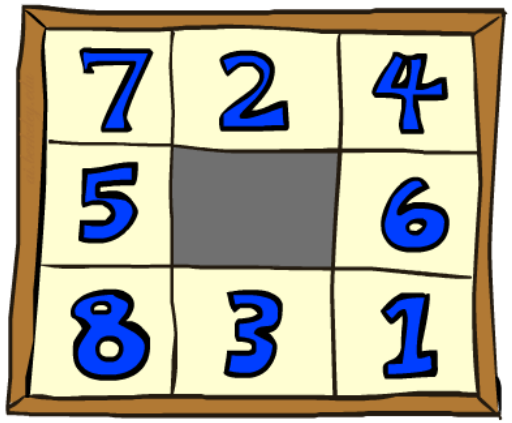


	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
A*TILES	13	39	227

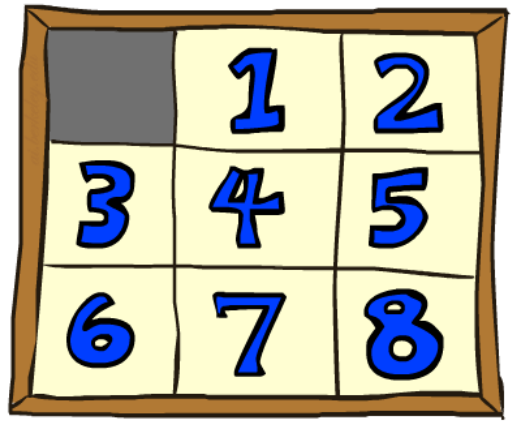
Statistics from Andrew Moore

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

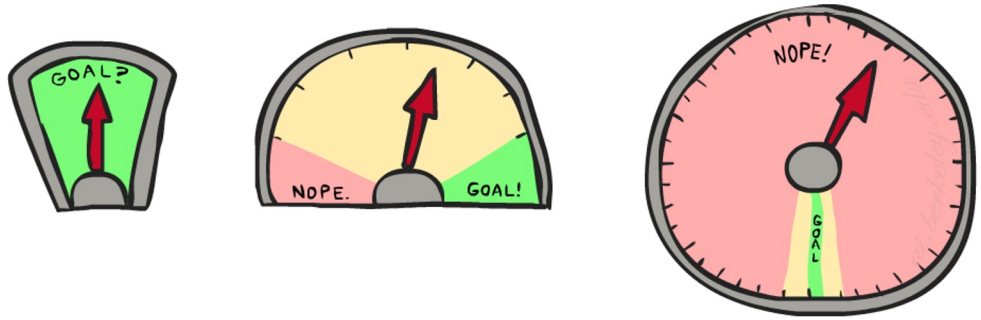


Goal State

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

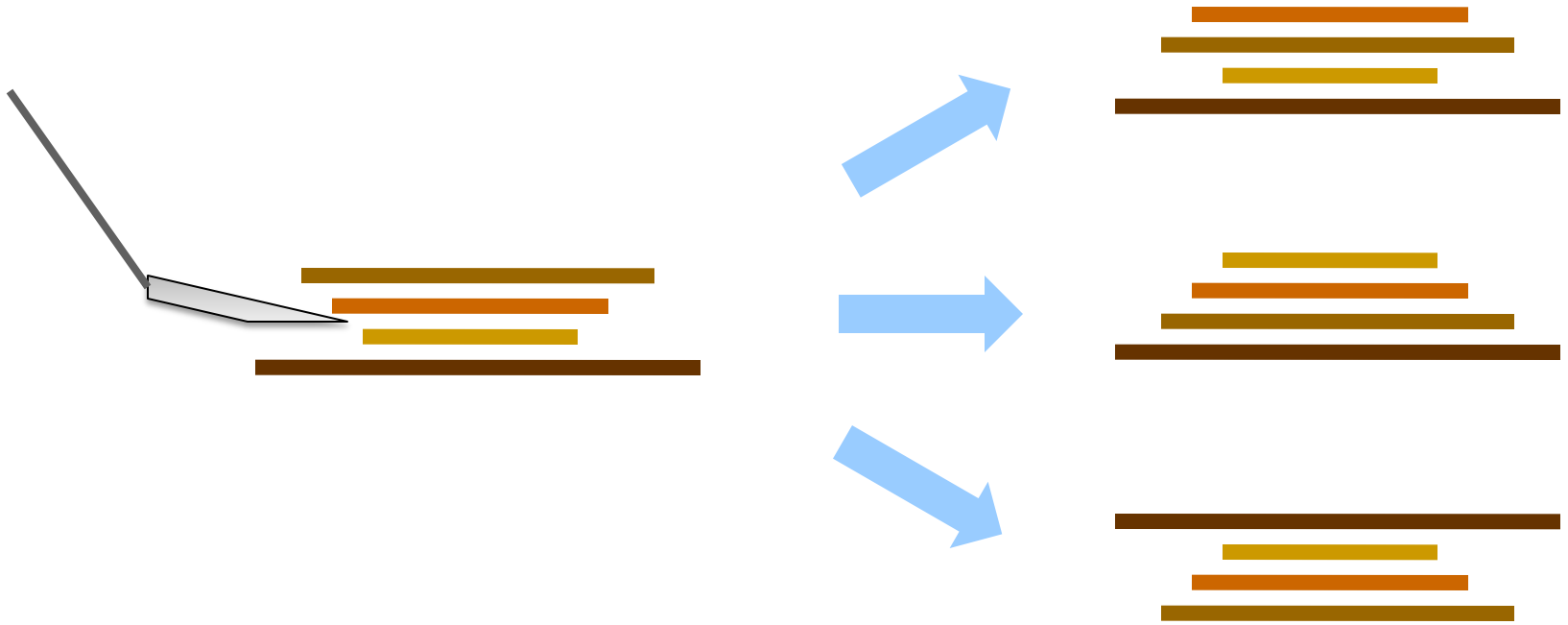
8 Puzzle III

- 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

Example: Pancake Problem



Cost: Number of pancakes flipped

Example: Pancake Problem

BOUNDS FOR SORTING BY PREFIX REVERSAL

William H. GATES

Microsoft, Albuquerque, New Mexico

Christos H. PAPADIMITRIOU*†

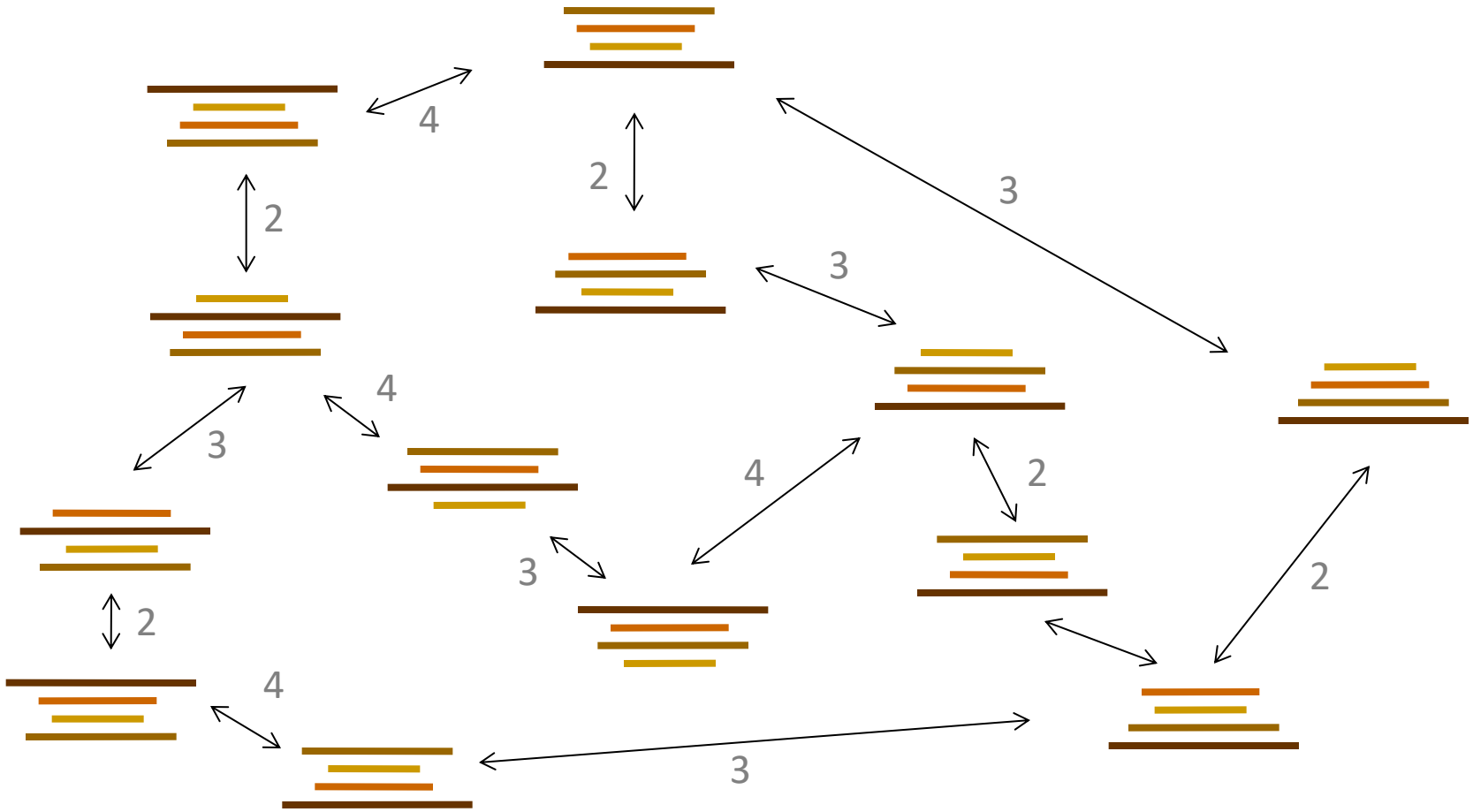
Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.

Received 18 January 1978
Revised 28 August 1978

For a permutation σ of the integers from 1 to n, let $f(\sigma)$ be the smallest number of prefix reversals that will transform σ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all σ in (the symmetric group) S_n. We show that $f(n) \leqslant (5n+5)/3$, and that $f(n) \geqslant 17n/16$ for n a multiple of 16. If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function $g(n)$ is shown to obey $3n/2 - 1 \leqslant g(n) \leqslant 2n+3$.

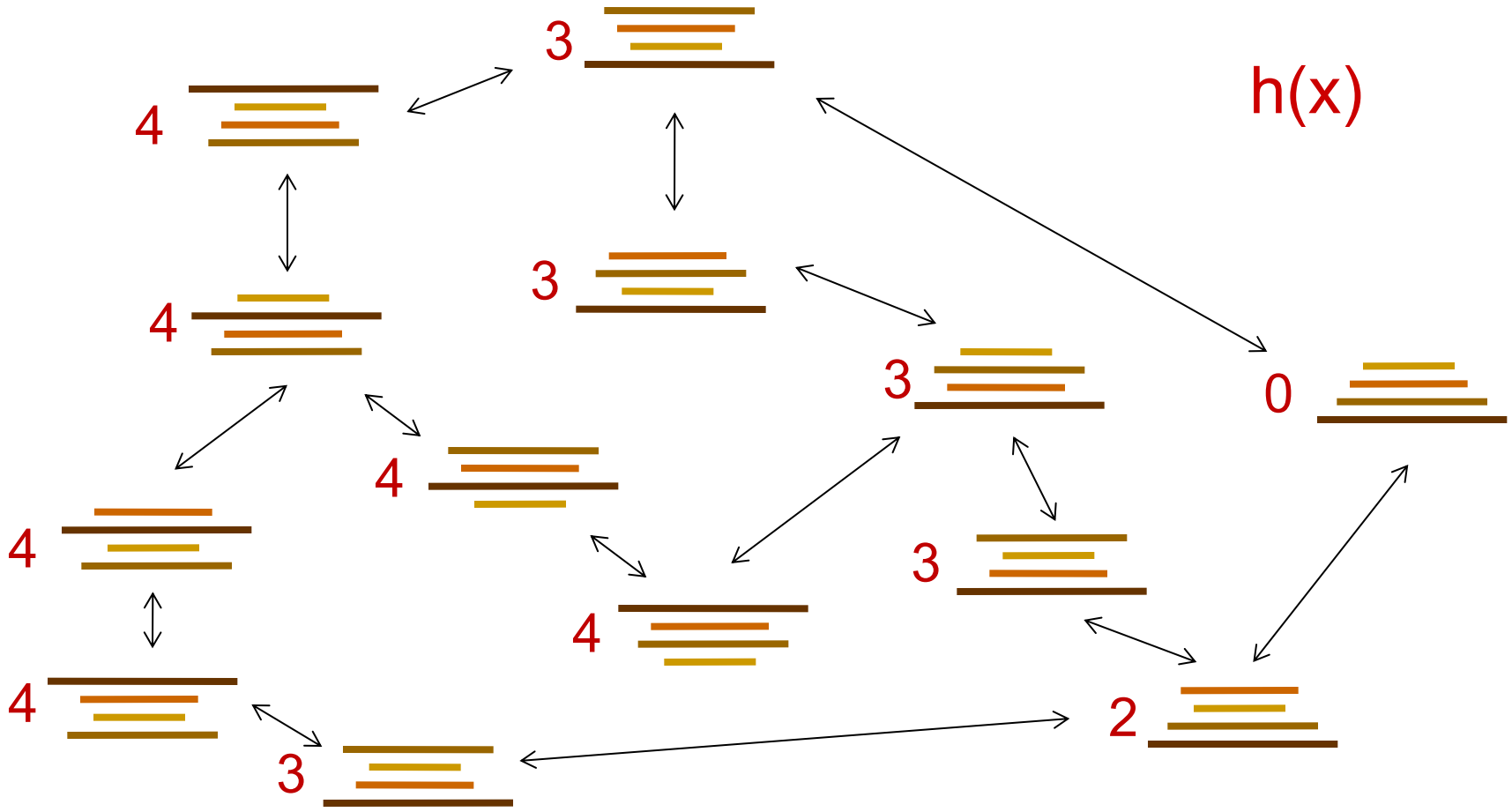
Example: Pancake Problem

State space graph with costs as weights



Example: Heuristic Function

Heuristic: the number of the largest pancake that is still out of place



Combining heuristics

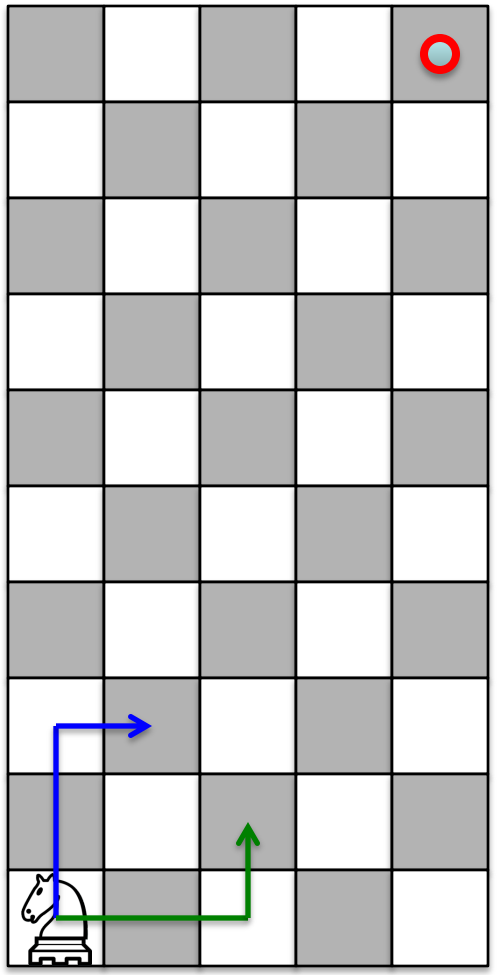
- Dominance: $h_1 \geq h_2$ if

  $$\forall n \;\; h_1(n) \geq h_2(n)$$

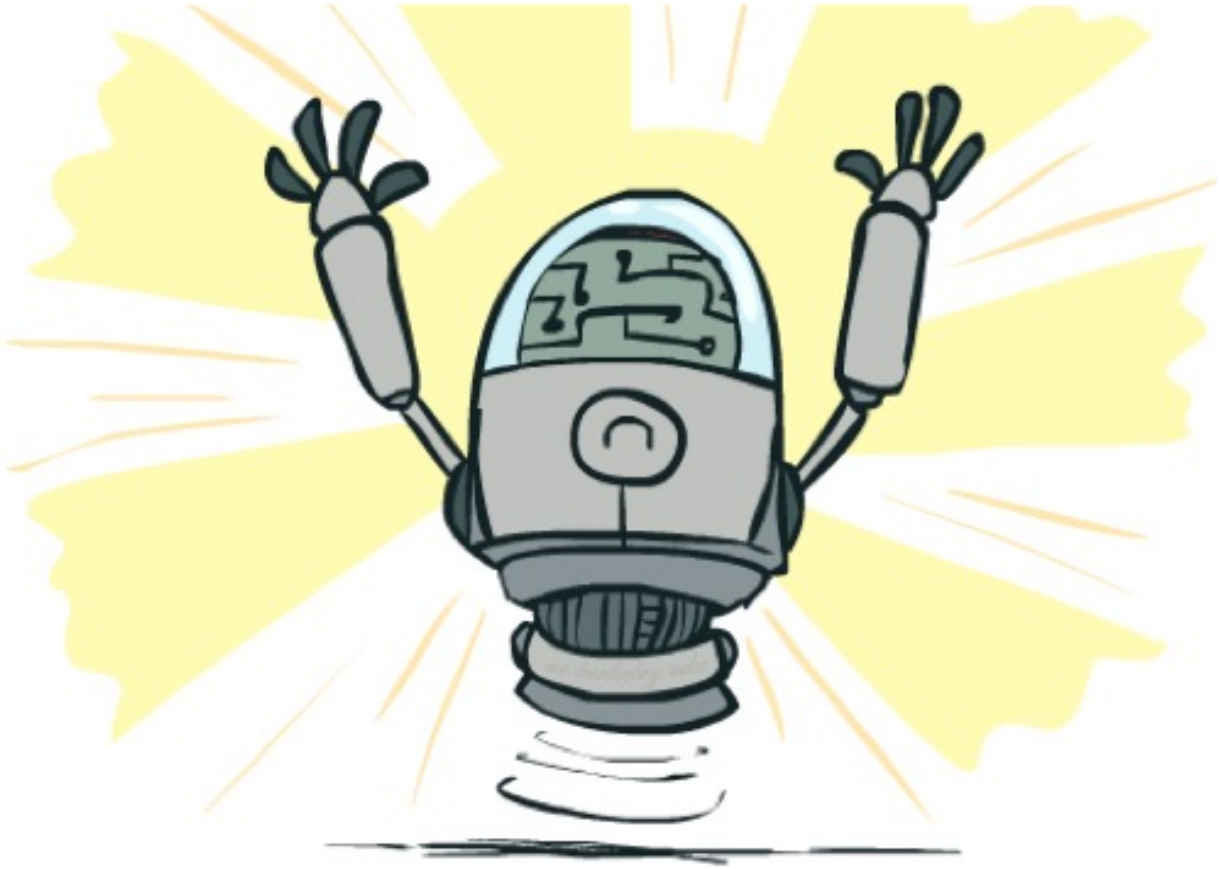
  - Roughly speaking, larger value is better as long as both are admissible
  - The zero heuristic is pretty bad (what does A* do with h=0?)
  - The exact heuristic is pretty good, but usually too expensive!
- What if we have two heuristics, neither dominates the other?
  - Form a new heuristic by taking the max of both:

    $$h(n) = \max(\, h_1(n), h_2(n))$$

  - Max of admissible heuristics is admissible and dominates both!
  - Example: number of knight's moves to get from A to B
    - h1 = (Manhattan distance)/3 (rounded up to correct parity)
    - h2 = (Euclidean distance)/√5 (rounded up to correct parity)
    - h3 = (max x or y shift)/2 (rounded up to correct parity)



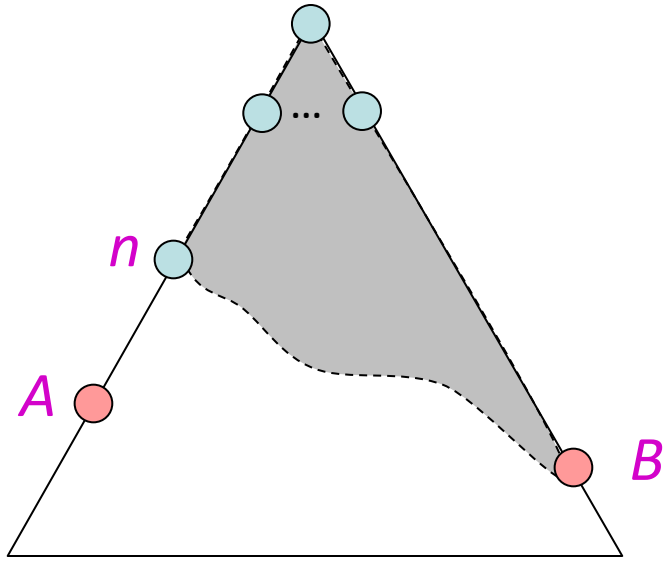
Optimality of A* Tree Search



Optimality of A* Tree Search: Blocking

Proof:

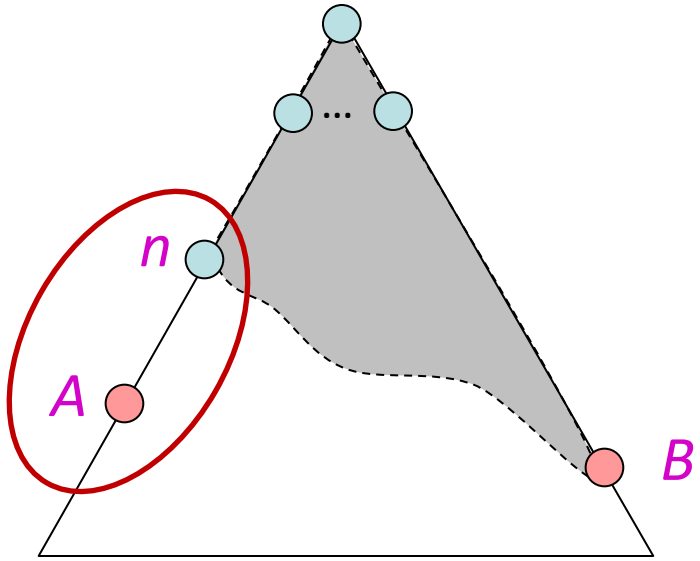
- Imagine *B* is on the frontier
- Some ancestor *n* of *A* is on the frontier, too (maybe *A* itself!)
- Claim: *n* will be expanded before *B*
  1. $f(n)$ is less than or equal to $f(A)$
  2. $f(A)$ is less than $f(B)$
  3. *n* is expanded before *B*
- All ancestors of *A* are expanded before *B*
-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



Optimality of A* Tree Search: Blocking

Proof:

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



$f(n) = g(n) + h(n)$ Definition of *f*-cost

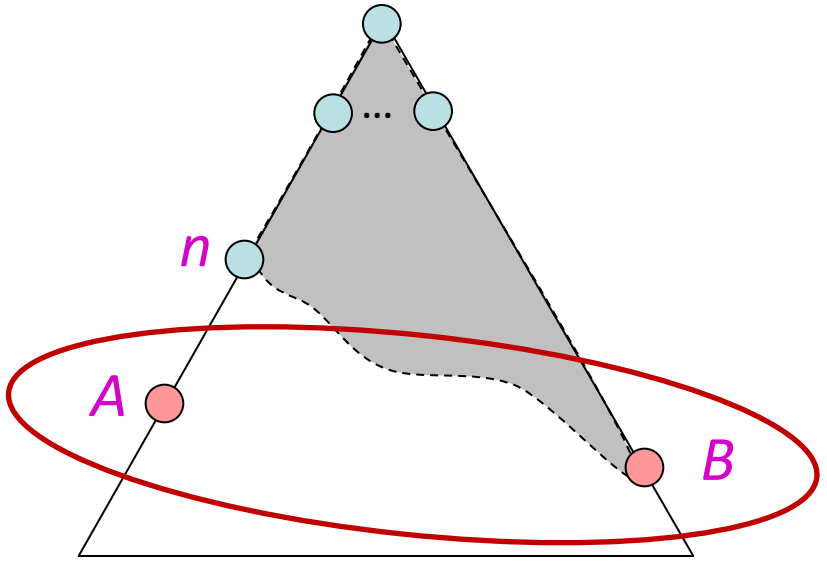
$f(n) \leq g(A)$ Admissibility of *h*

$g(A) = f(A)$ *h* = 0 at a goal

Optimality of A* Tree Search: Blocking

Proof:

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



$g(A) < g(B)$

$f(A) < f(B)$

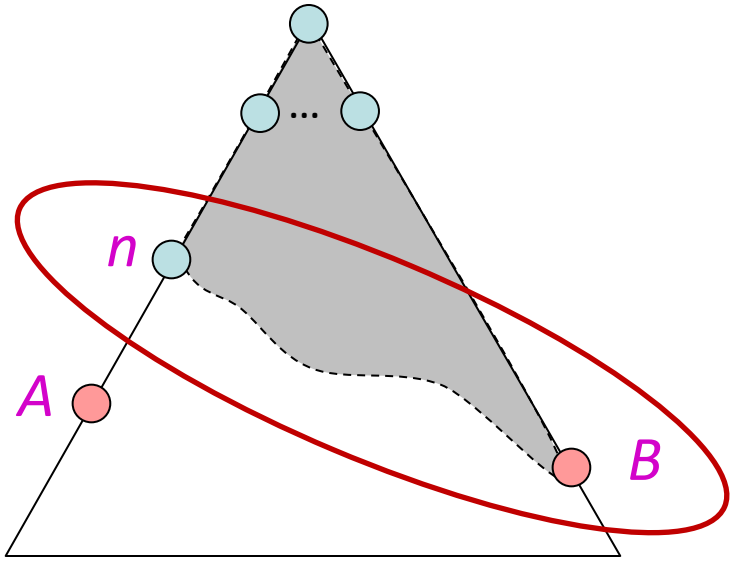
Suboptimality of *B*

$h = 0$ at a goal

Optimality of A* Tree Search: Blocking

Proof:

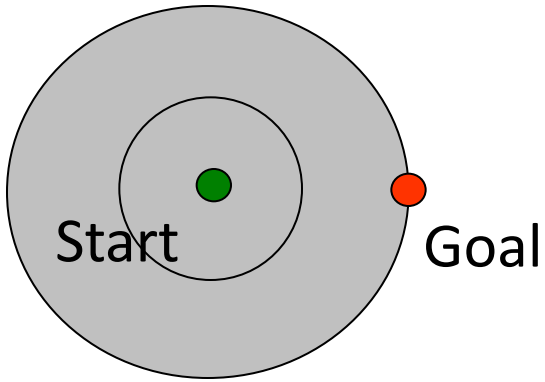
- Imagine *B* is on the frontier
- Some ancestor *n* of *A* is on the frontier, too (maybe *A* itself!)
- Claim: *n* will be expanded before *B*
  1. $f(n)$ is less than or equal to $f(A)$
  2. $f(A)$ is less than $f(B)$
  3. *n* is expanded before *B*
- All ancestors of *A* are expanded before *B*
-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



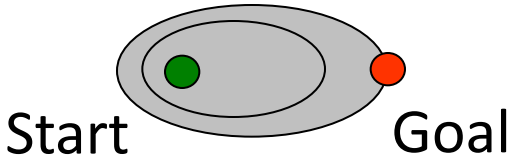
$$f(n) \leq f(A) < f(B)$$

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



What do each of these functions measure?

- h(n)
- h*(n)
- g(n)
- g*(n)
- f(n) (from A*)

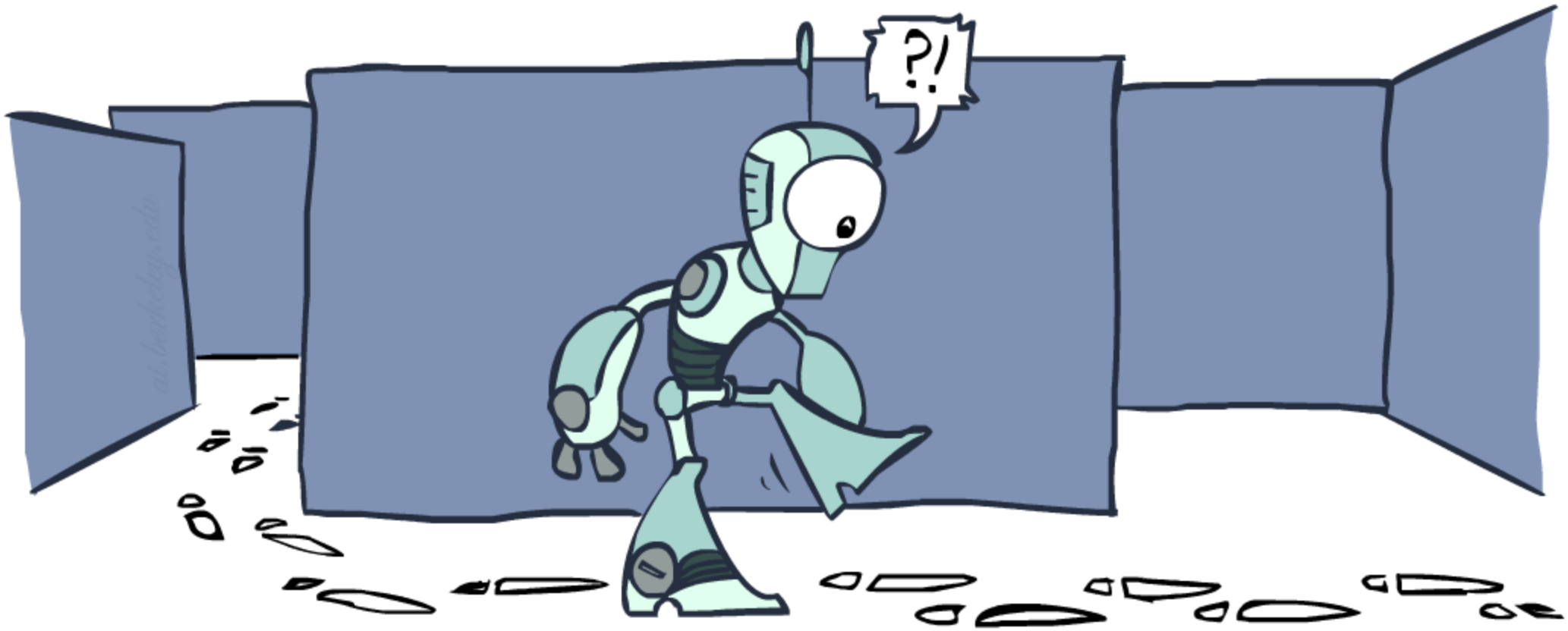
Comparison

Greedy (h)

Uniform Cost (g)

A* (g+h)

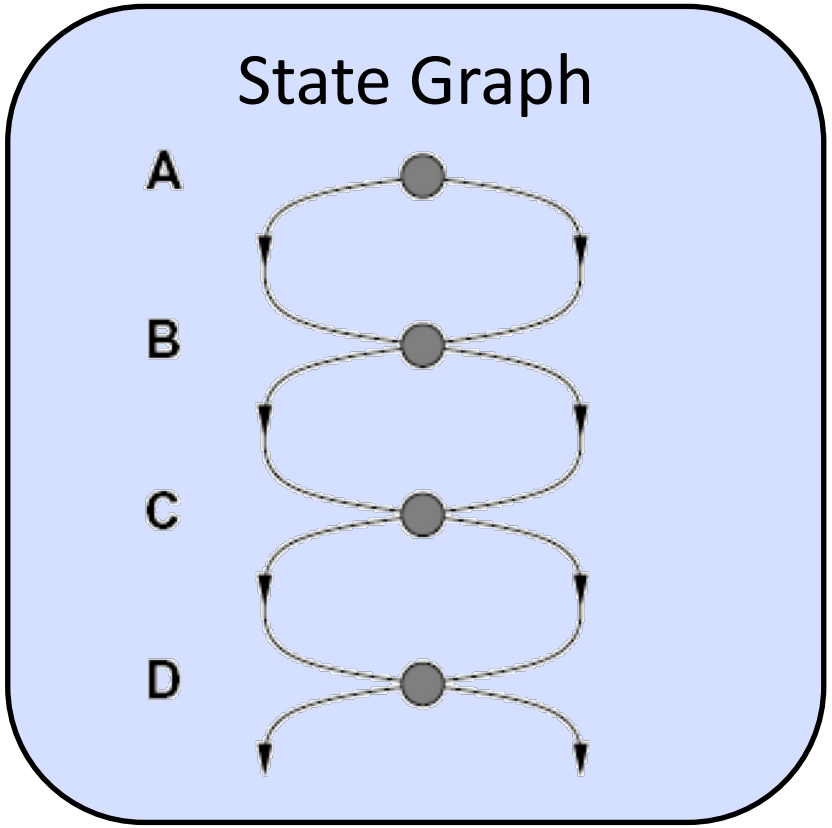
Optimality of A* Graph Search

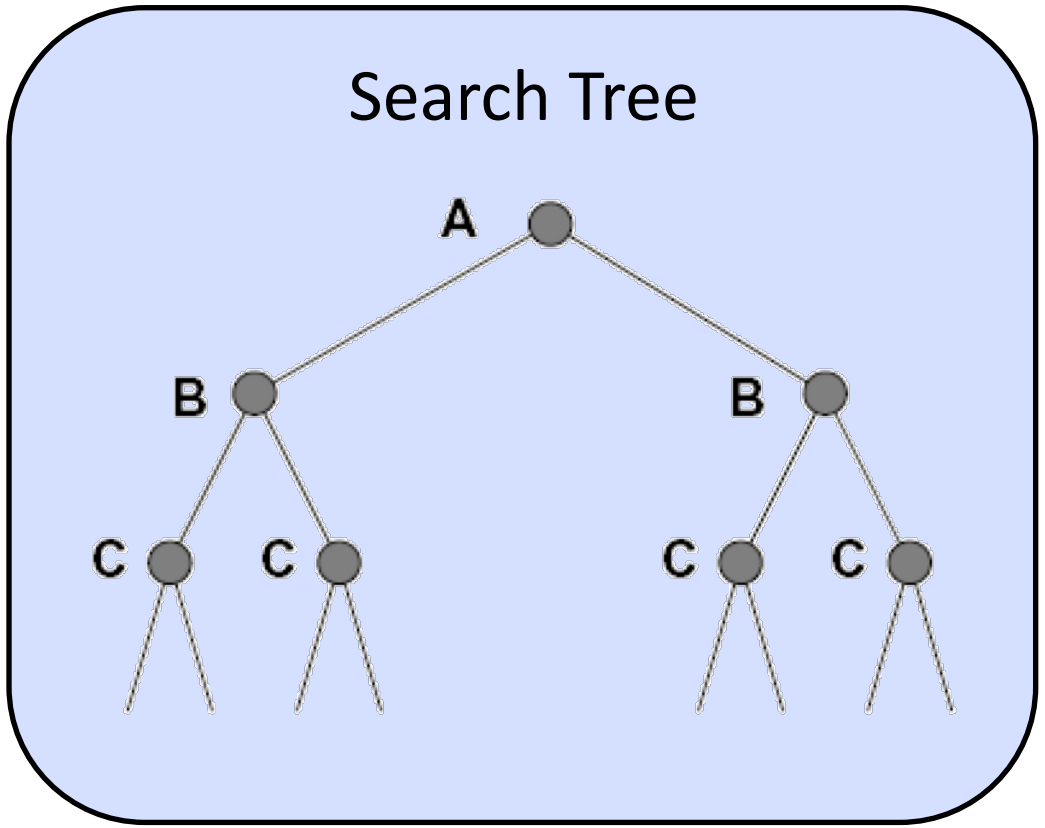


This part is a bit technical...

Tree Search: Extra Work!

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



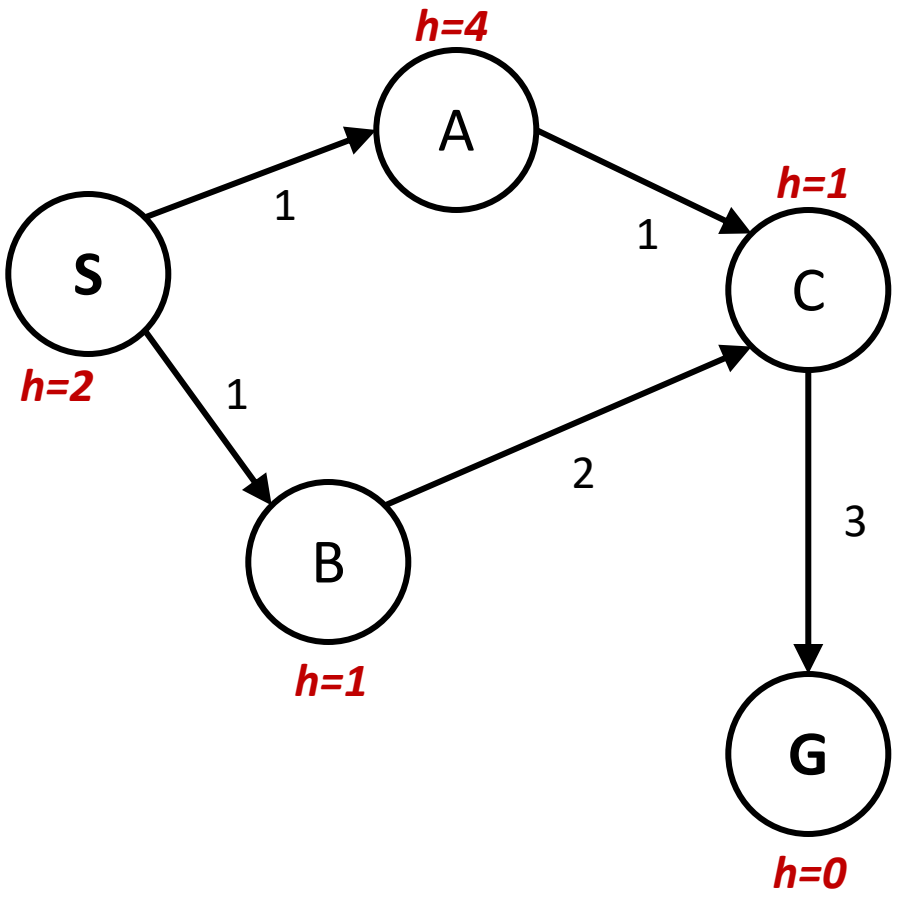


Graph Search

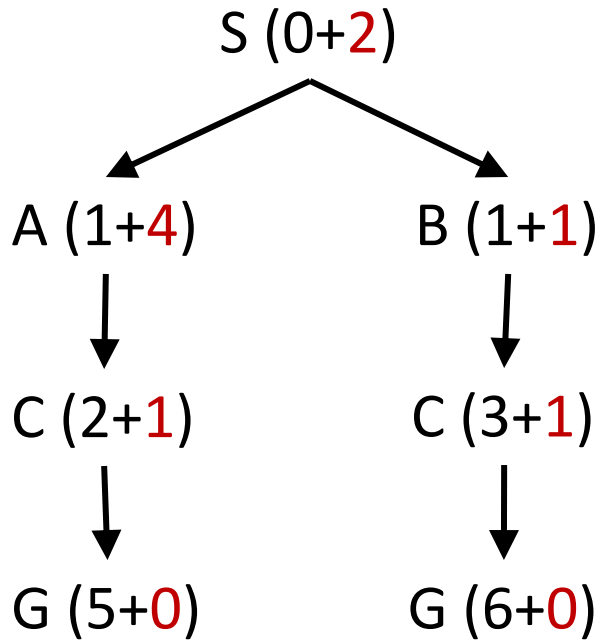
- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states (“closed set”)
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to closed set
- Important: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

A* Graph Search Gone Wrong?

State space graph



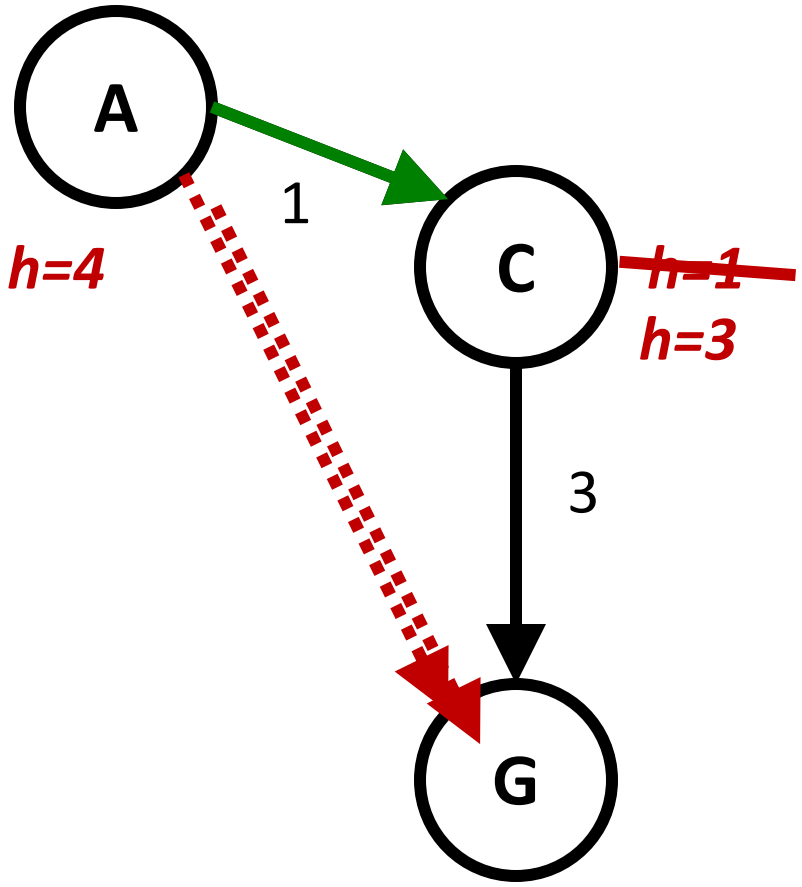
Search tree



Simple check against expanded set blocks C

Fancy check allows new C if cheaper than old but requires recalculating C's descendants

Consistency of Heuristics



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

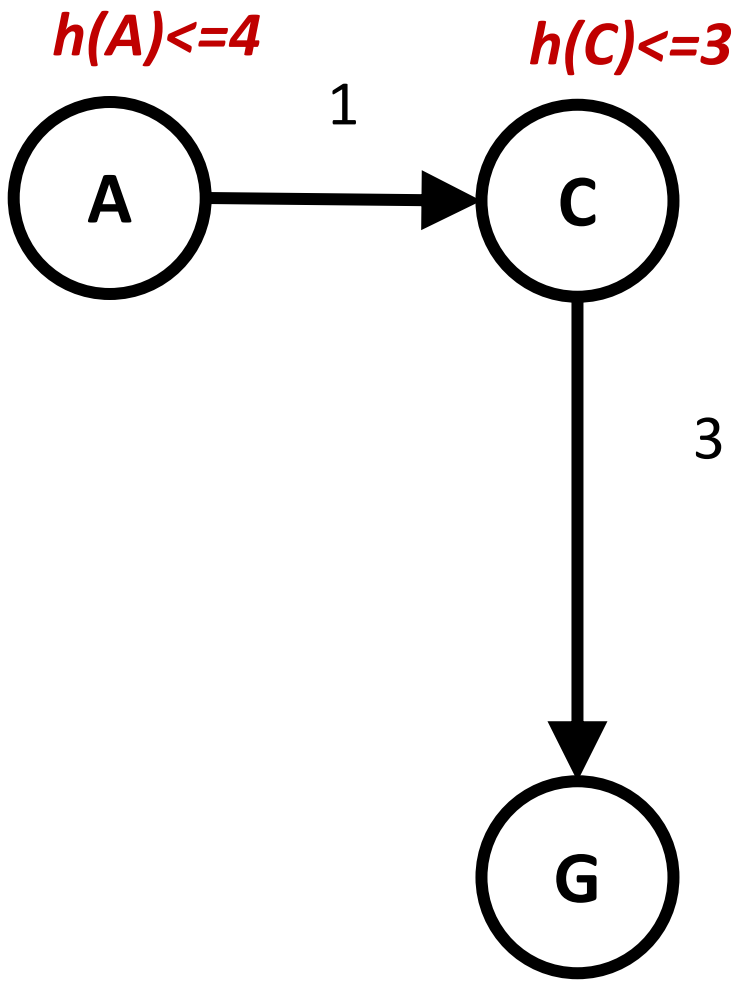
    $h(A) \leq h^*(A)$
  - Consistency: heuristic "arc" cost ≤ actual cost for each arc

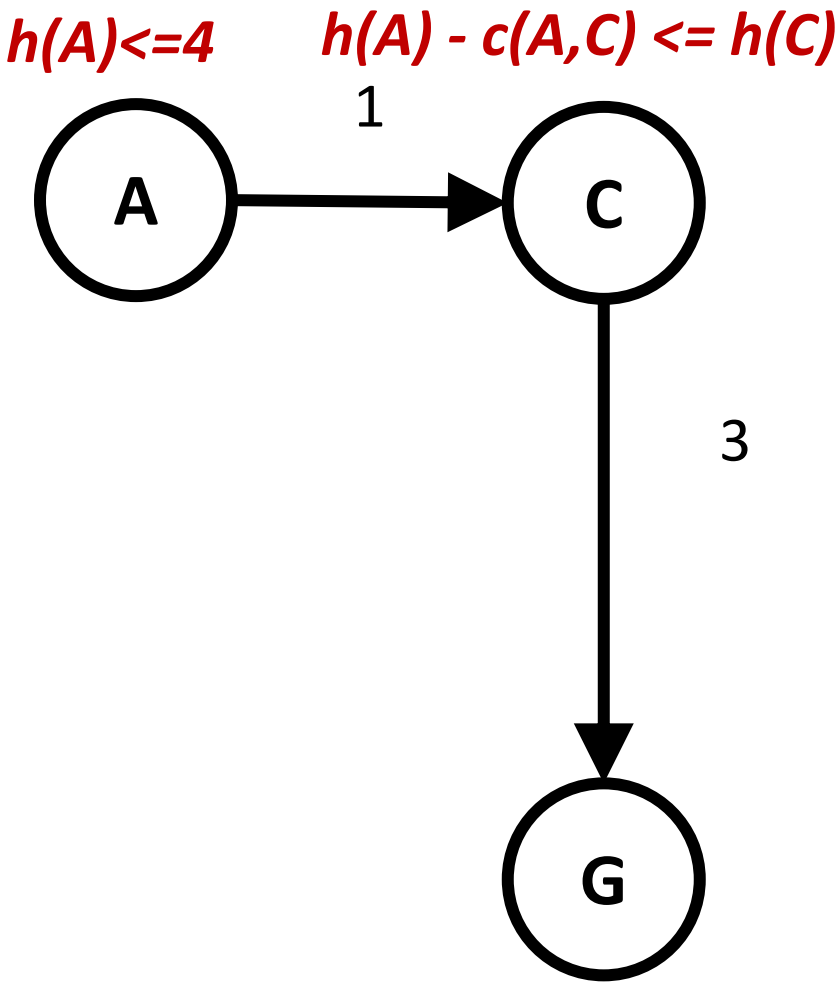
    $h(A) - h(C) \leq c(A,C)$

    or $h(A) \leq c(A,C) + h(C)$ (triangle inequality)
- Consequences of consistency:
  - The *f* value along a path never decreases:

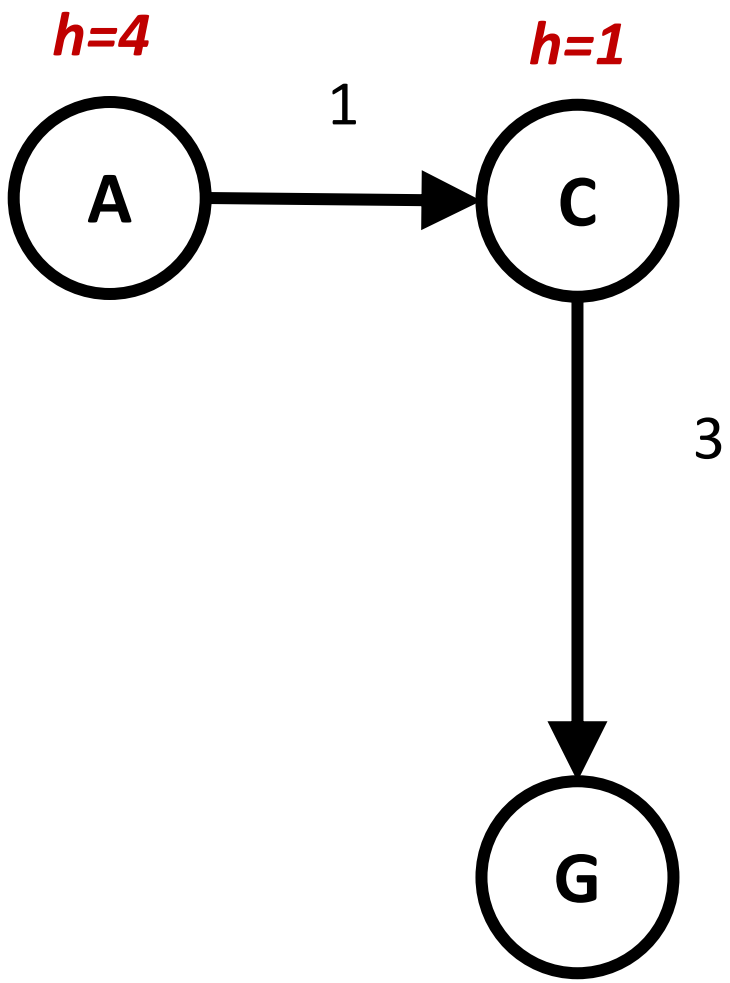
    $h(A) \leq c(A,C) + h(C) \Rightarrow g(A) + h(A) \leq g(A) + c(A,C) + h(C)$
  - A* graph search is optimal

Admissibility and Consistency Constraints



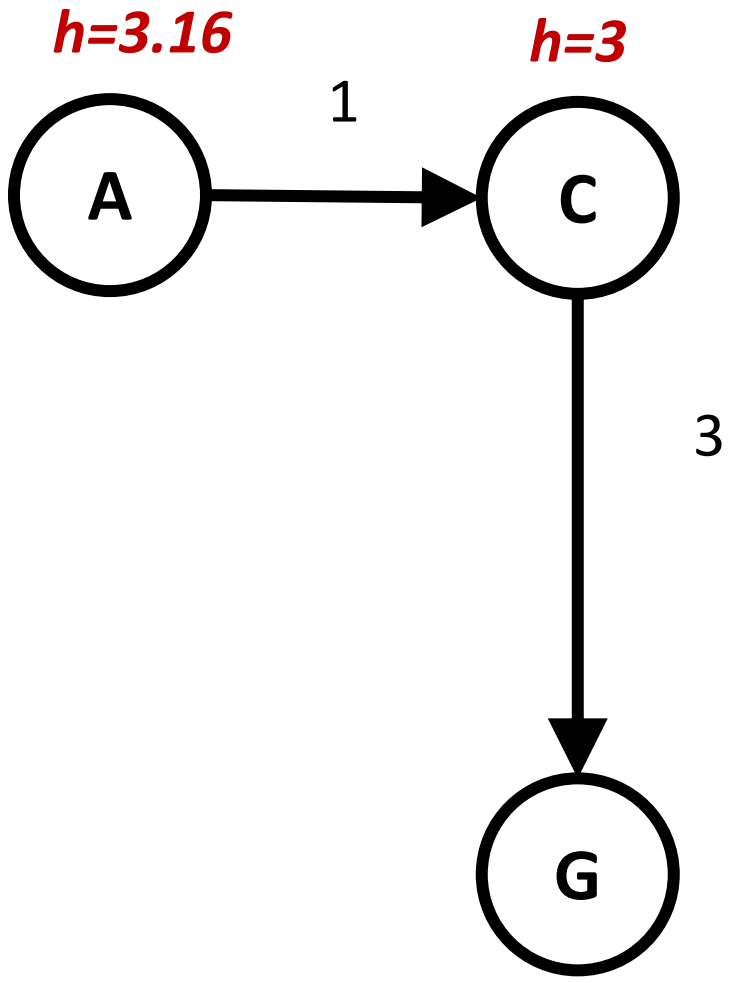


Admissible?

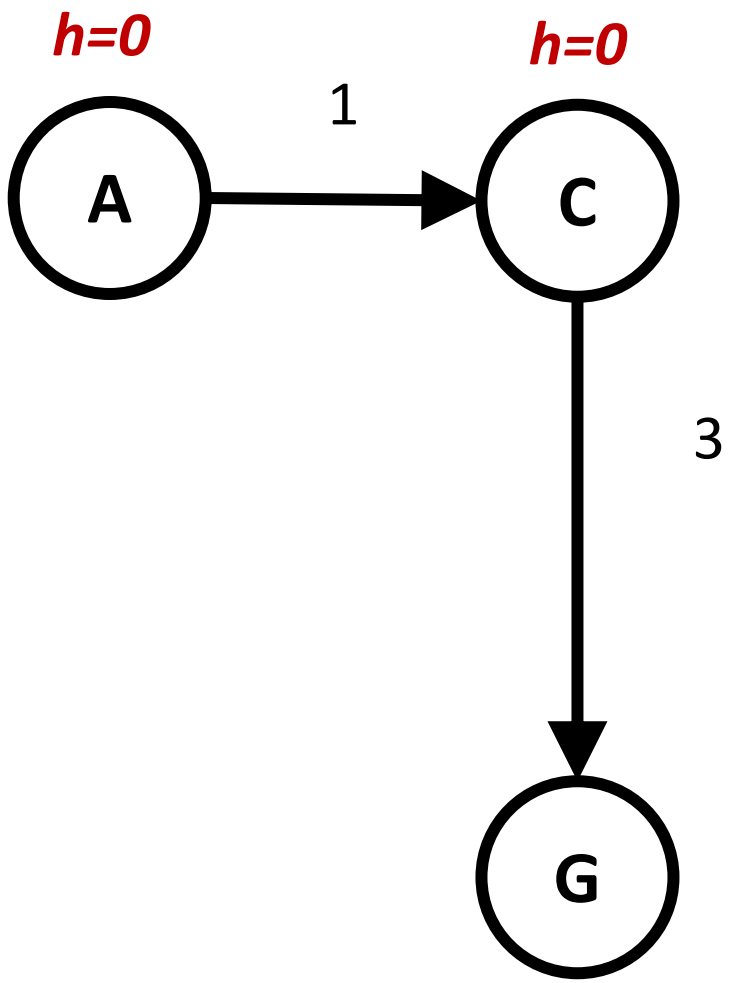


Consistent?

h(A) - c(A,C) <= h(C)
3.16 – 1 <= 3

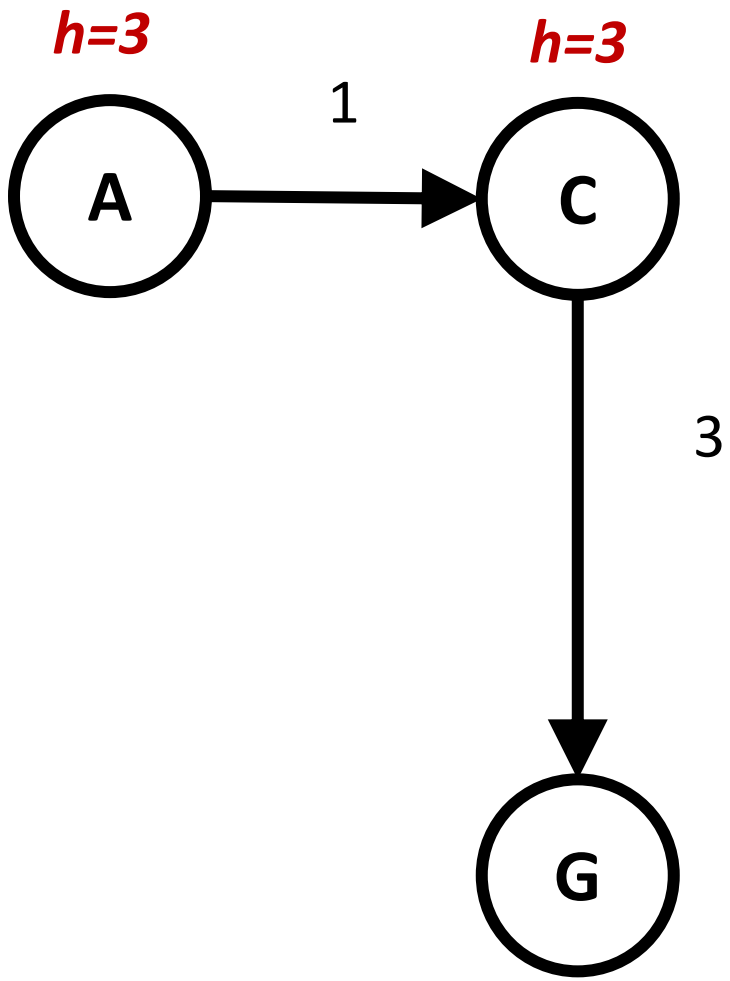


Admissible?

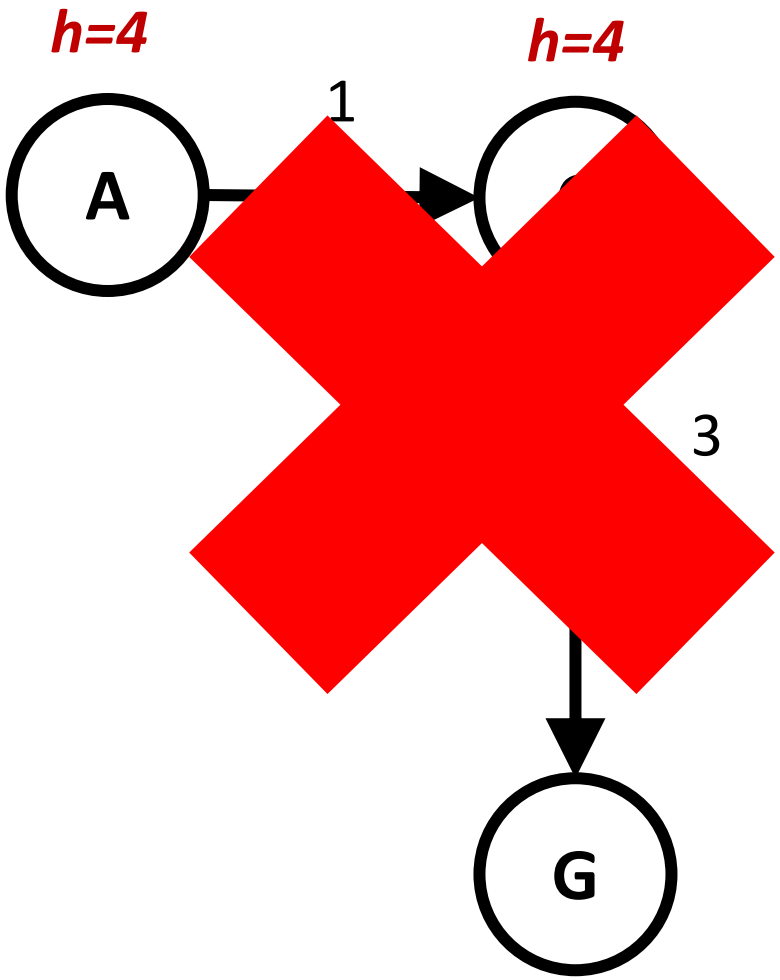


Consistent?

h(A) - c(A,C) <= h(C)
3 – 1 <= 3

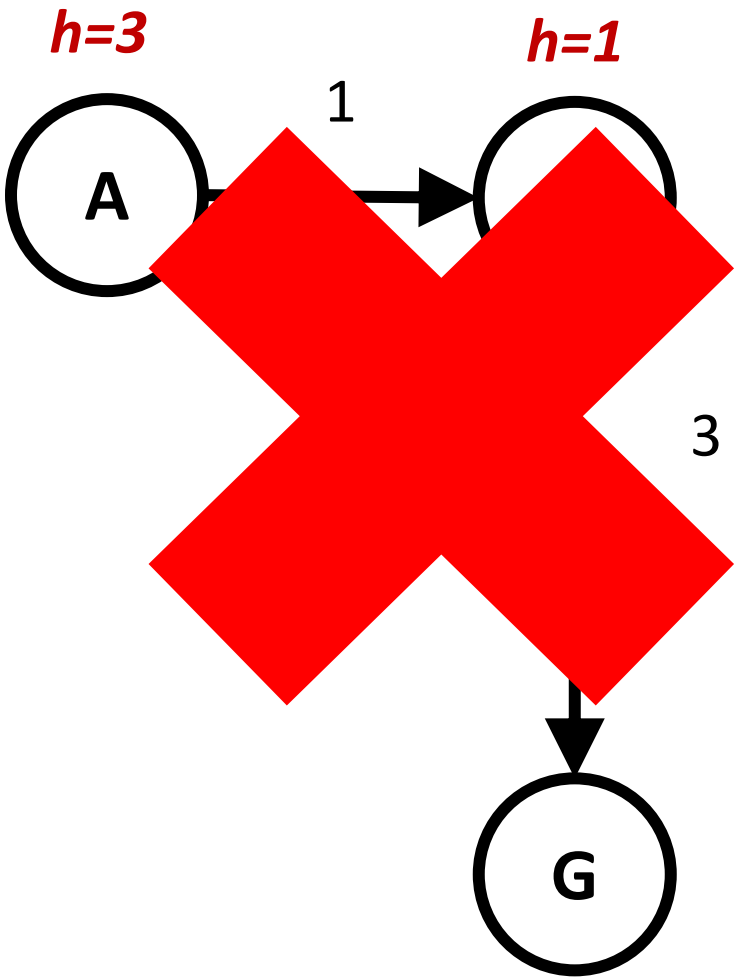


Admissible?



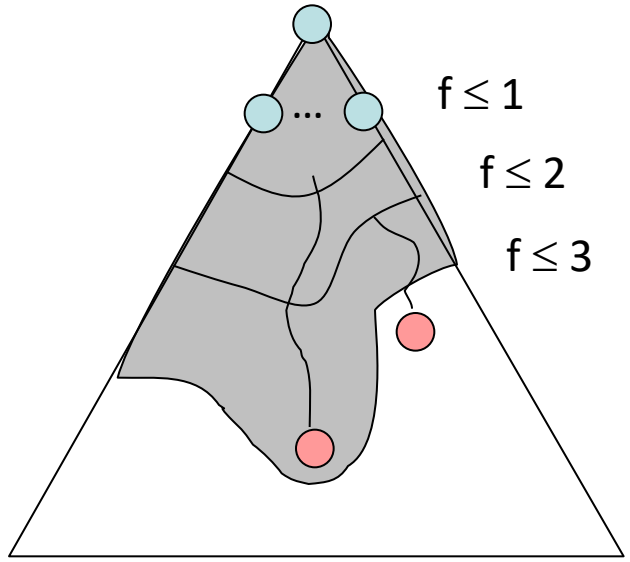
Consistent?

h(A) - c(A,C) <= h(C)
3 – 1 <= 1



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
- Graph search:
  - A* optimal if heuristic is consistent
- Consistency implies admissibility
- Most natural admissible heuristics tend to be consistent, especially if from relaxed problems
  - For admissible but inconsistent heuristics see https://www.aaai.org/Papers/AAAI/2007/AAAI07-192.pdf