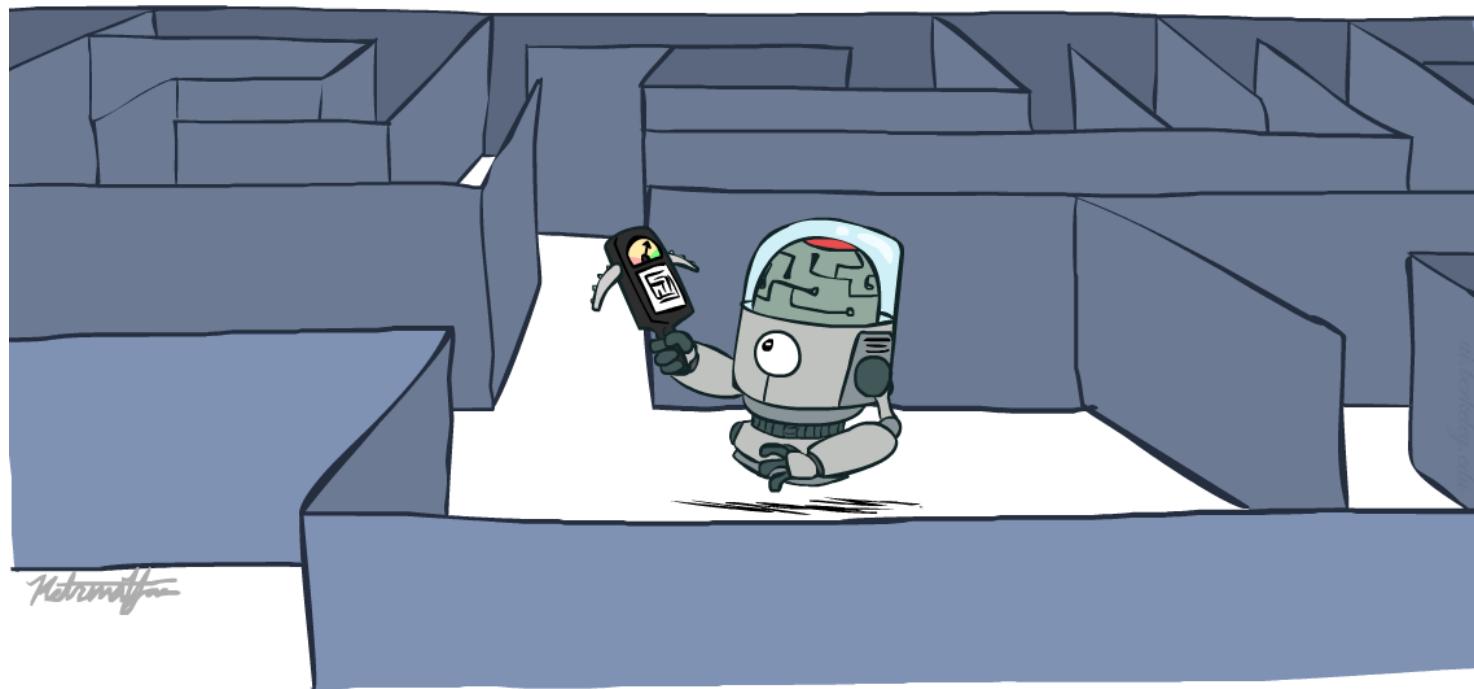


Artificial Intelligence

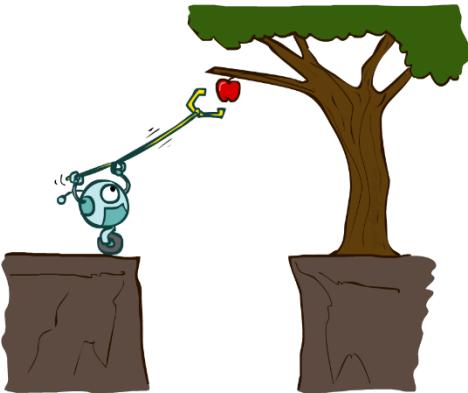
Informed Search



Homework Assignment

- Assignment 1 has been released
 - You are asked to write a very simple report on what you have observed when running the program

Course Topics



Search problems

Markov decision processes

Adversarial games

Constraint satisfaction problems

Bayesian networks

Reflex

States

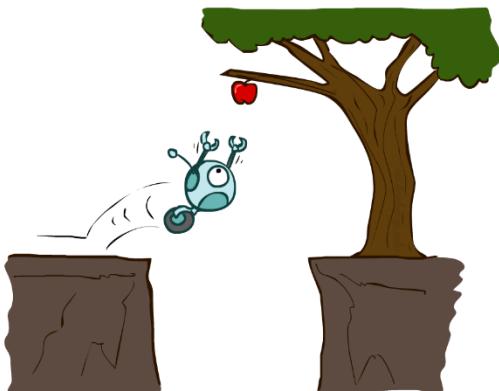
Variables

Logic

"Low-level intelligence"

"High-level intelligence"

Machine learning



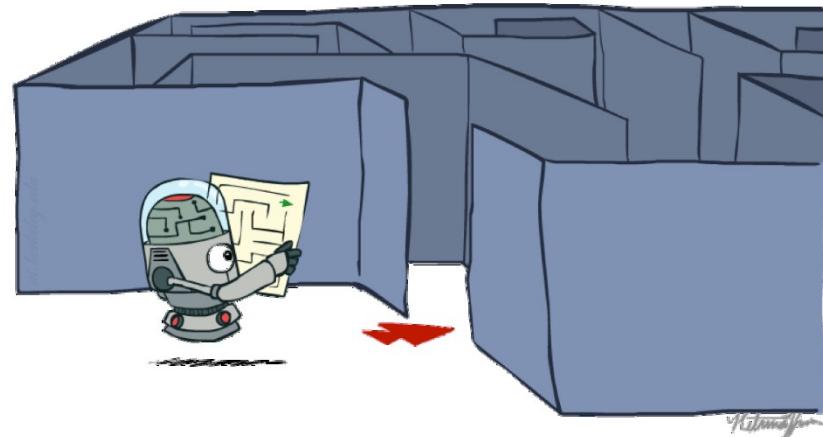
Today

- Uninformed Search
 - Uniform Cost Search
- Informed Search
 - Heuristics
 - Greedy Search
 - A* Search
- Graph Search



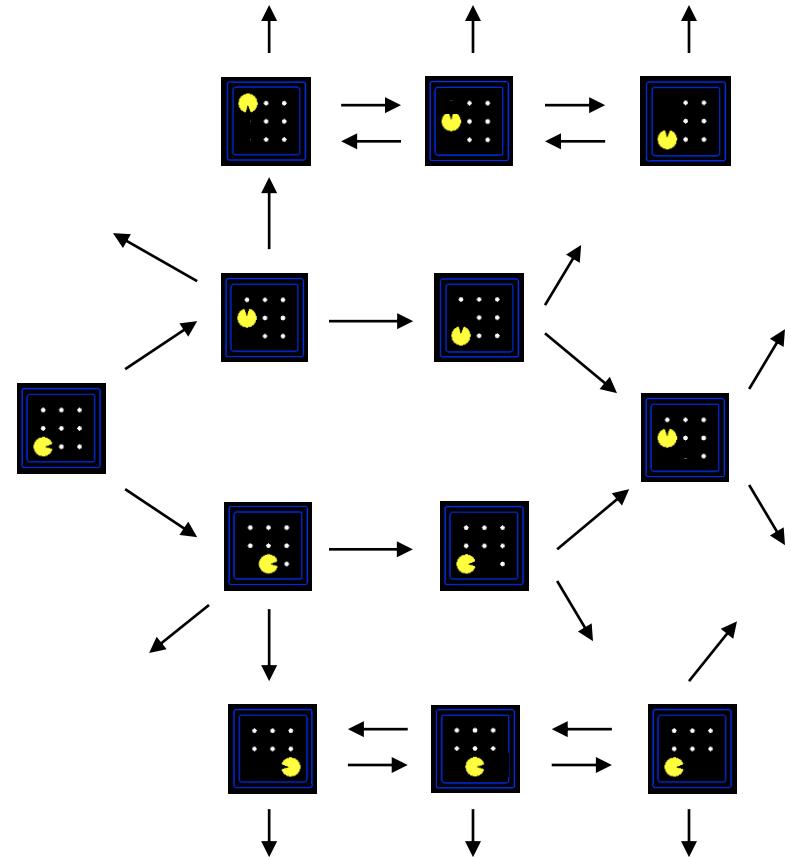
Recap: Search

- 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 goals
 - Plans have costs (sum of action costs, etc.)
- Search algorithm:
 - Systematically builds a search tree
 - Chooses an ordering of the fringe (unexplored nodes)
 - Optimal: finds least-cost plans



State Space Graphs

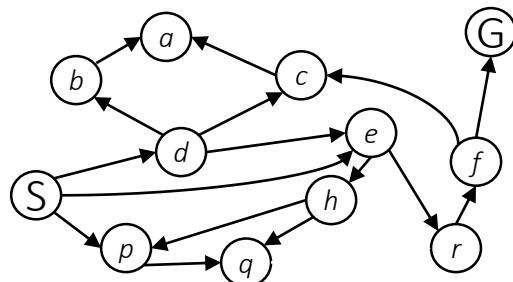
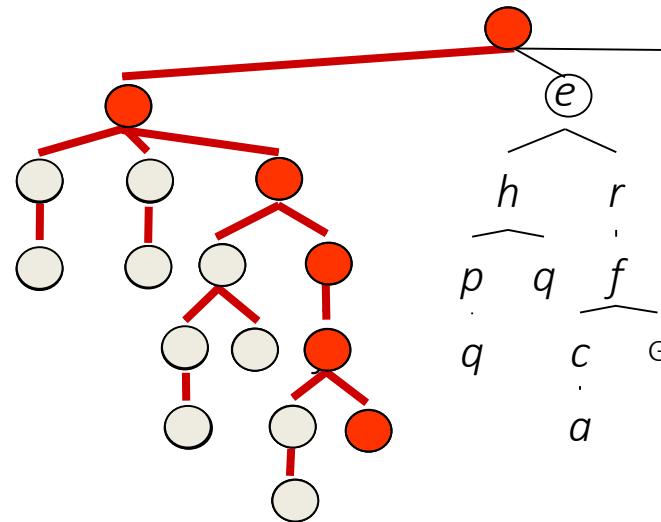
- State space graph: A mathematical representation of a search problem
 - Nodes are (abstracted) world configurations
 - Arcs represent successors (action results)
 - The goal test is a set of goal nodes (maybe only one)
- In a state space graph, each state occurs only once!
- We can rarely build this full graph in memory (it's too big), but it's a useful idea



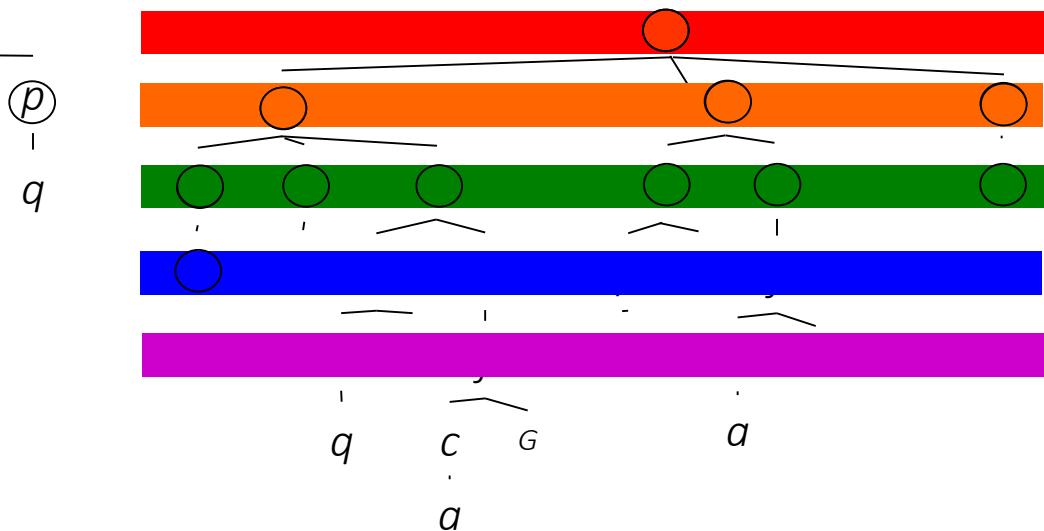
World states? $120 \times (2^{30}) \times (12^2) \times 4$

Search

Depth-First Search



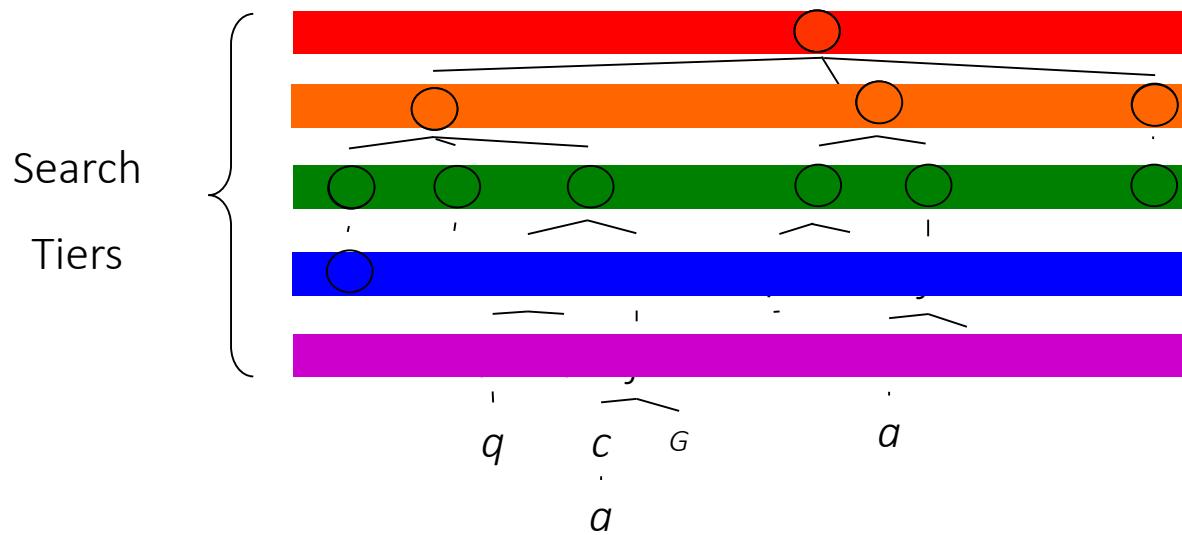
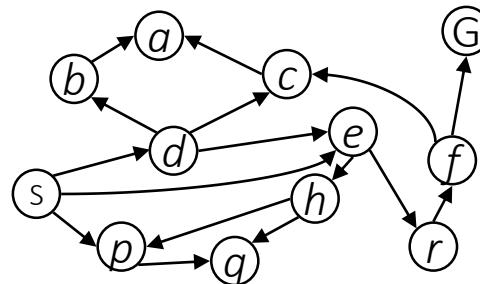
Breadth-First Search



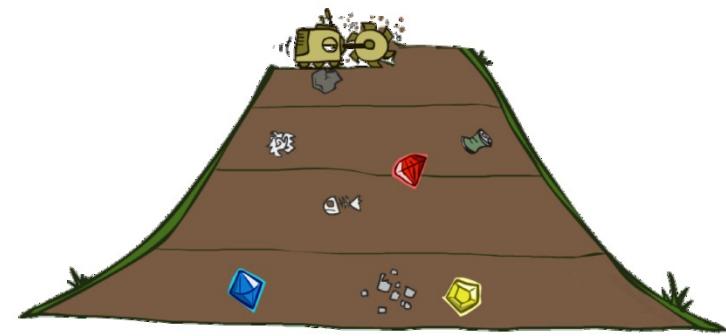
Breadth-First Search

Strategy: expand a shallowest node first

Implementation:
*Fringe is a **FIFO queue***



DFS vs BFS



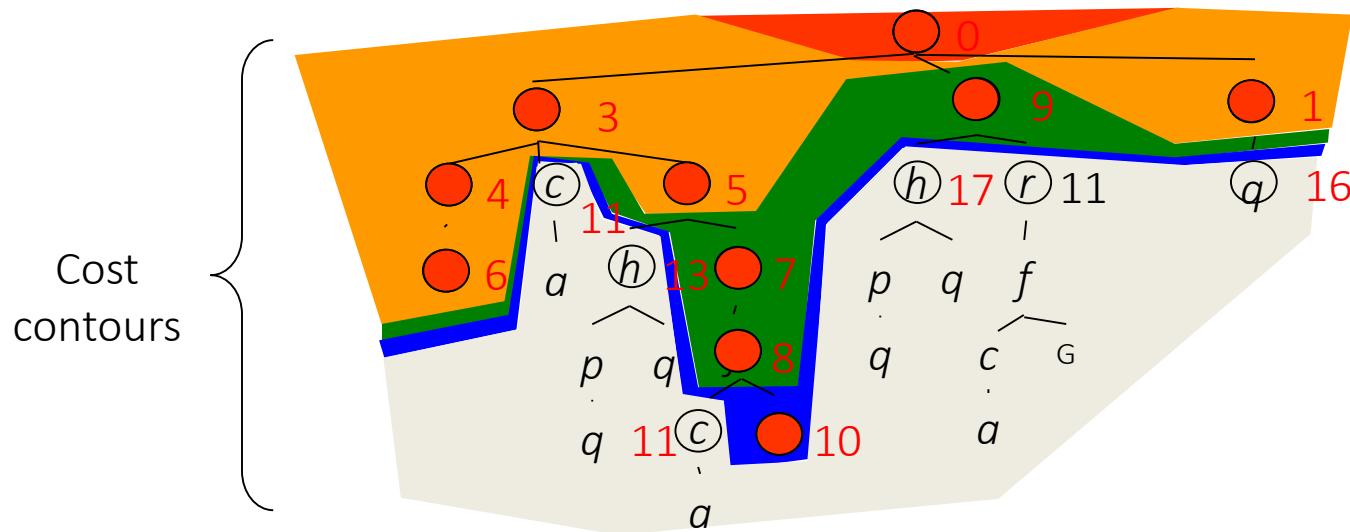
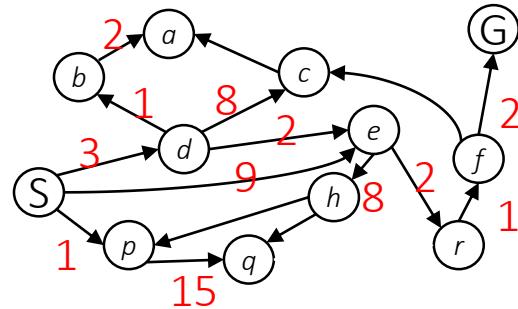
Uniform Cost Search



Uniform Cost Search

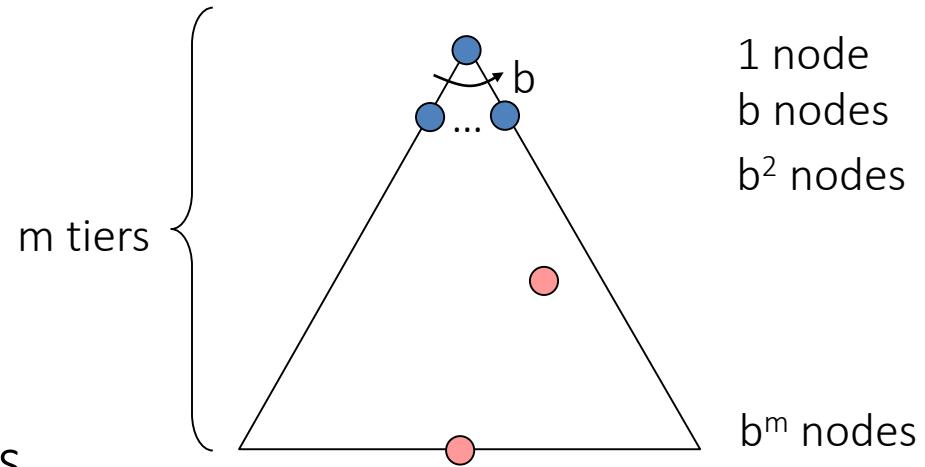
Strategy: expand a cheapest node first:

Fringe is a **priority queue**
(priority: cumulative cost)



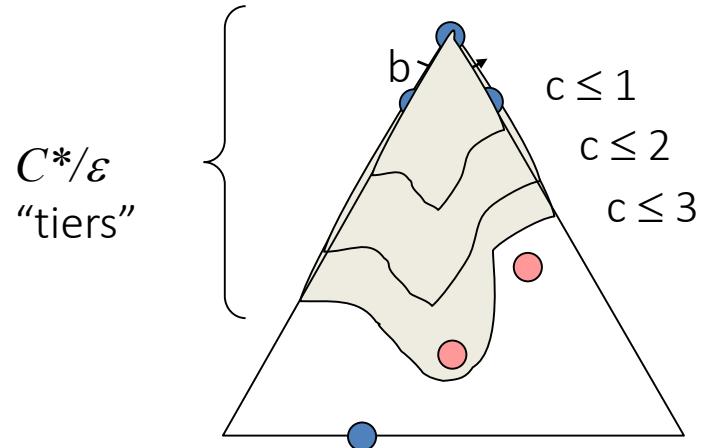
Search Algorithm Properties

- **Complete:** Guaranteed to find a solution if one exists?
- **Optimal:** Guaranteed to find the least cost path?
- **Time complexity?**
- **Space complexity?**
- Cartoon of search tree:
 - b is the branching factor
 - m is the maximum depth
 - solutions at various depths
- Number of nodes in entire tree?
 - $1 + b + b^2 + \dots + b^m = O(b^m)$



Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
 - Processes all nodes with cost less than cheapest solution!
 - If that solution costs C^* and arcs cost at least ε , then the “effective depth” is roughly C^*/ε
 - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)
- How much space does the fringe take?
 - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$
- Is it complete?
 - Assuming best solution has a finite cost and minimum arc cost is positive, yes!
- Is it optimal?
 - Yes!

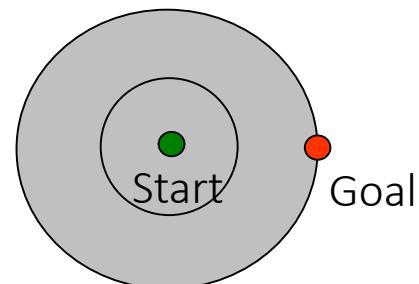
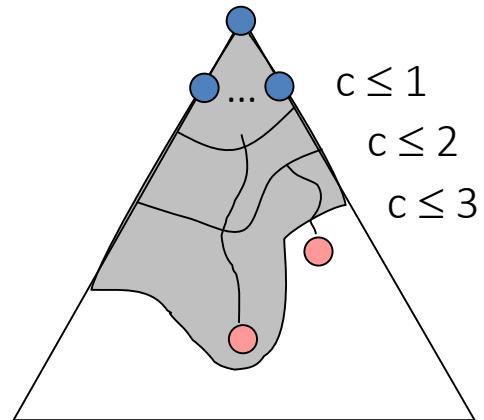


Uniform Cost Search

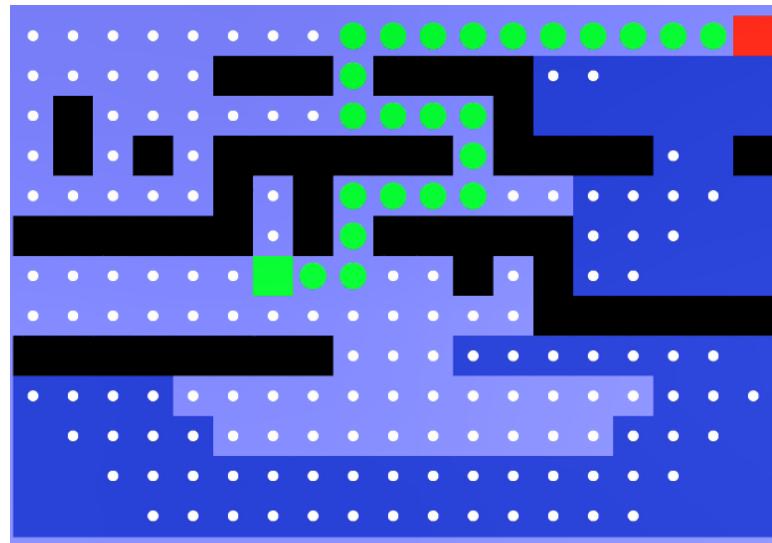
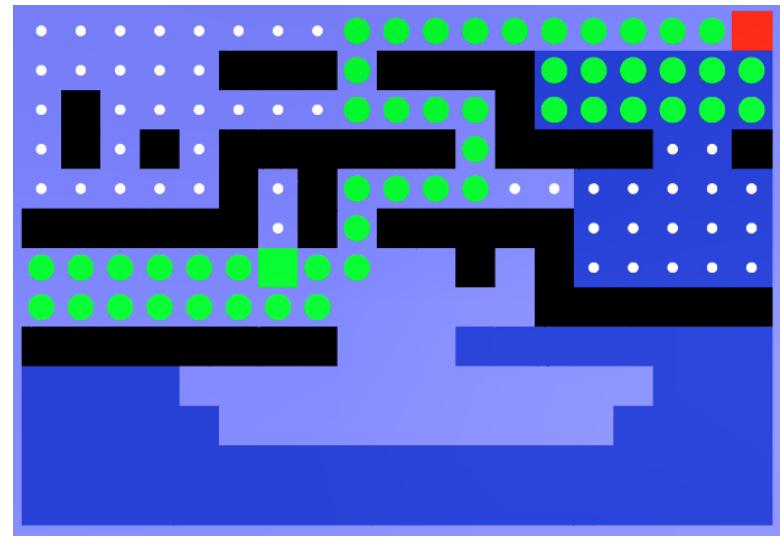
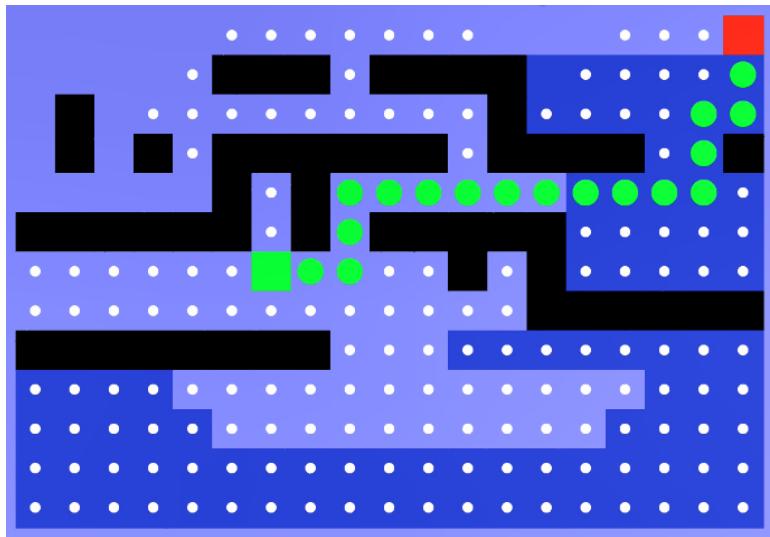
- Strategy: expand lowest path cost

- The good: UCS is complete and optimal!

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

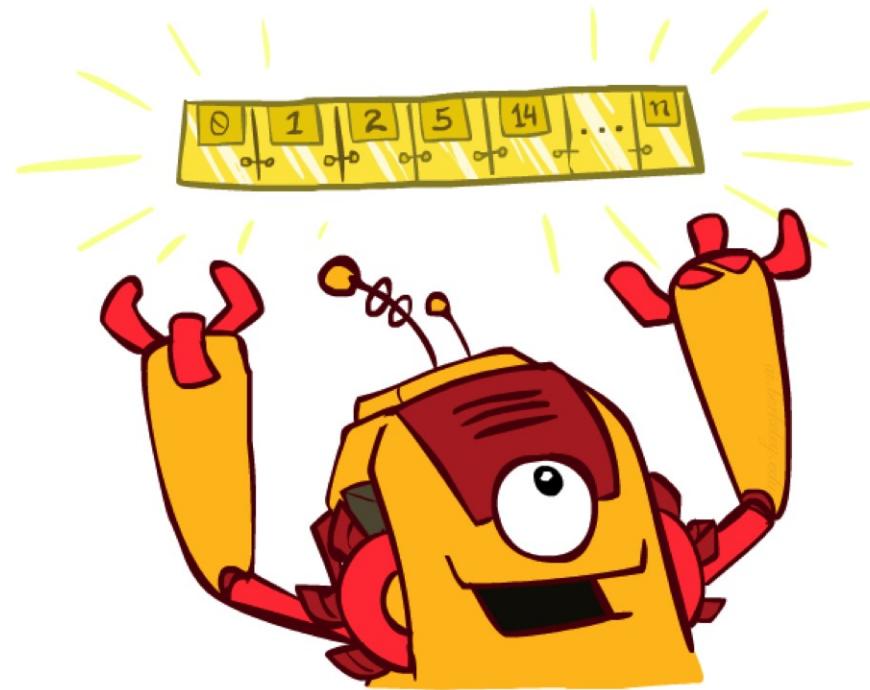


BFS, DFS, and UCF



The One Queue

- All these search algorithms are the same except for fringe strategies
 - Conceptually, all fringes are **priority queues** (i.e. collections of nodes with attached priorities)
 - Practically, for **DFS and BFS**, you can avoid the $\log(n)$ overhead from an actual priority queue, by using **stack and queues**
 - Can even code one implementation that takes a variable queuing object



Informed Search

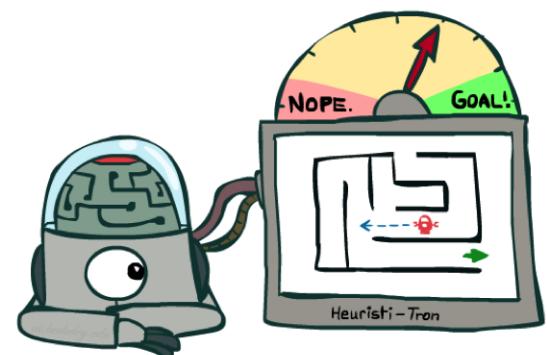
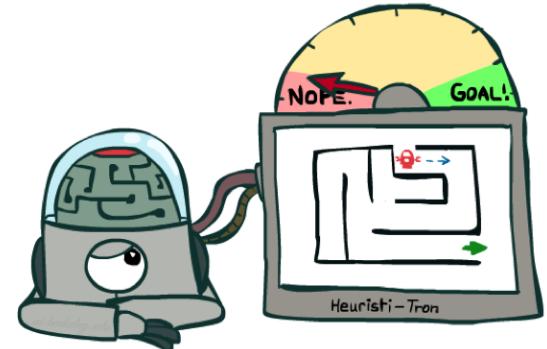
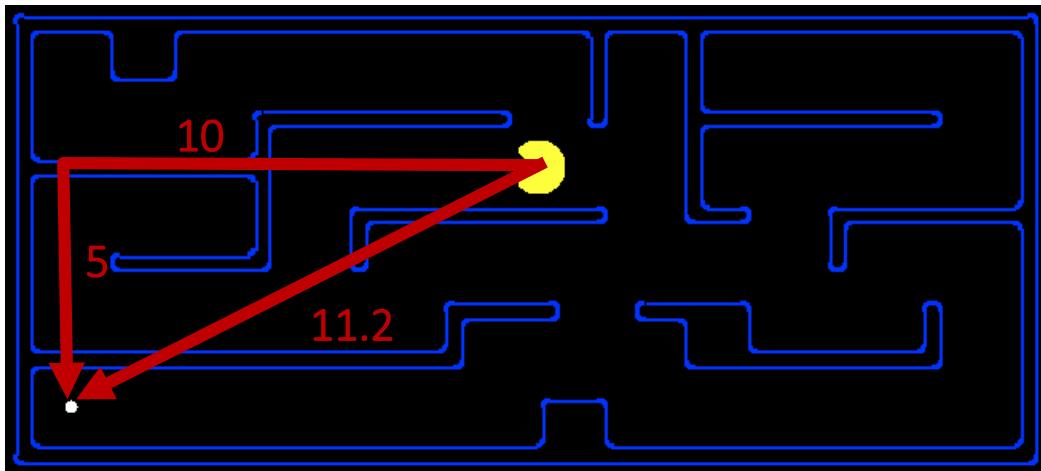




Search Heuristics

■ A heuristic is:

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



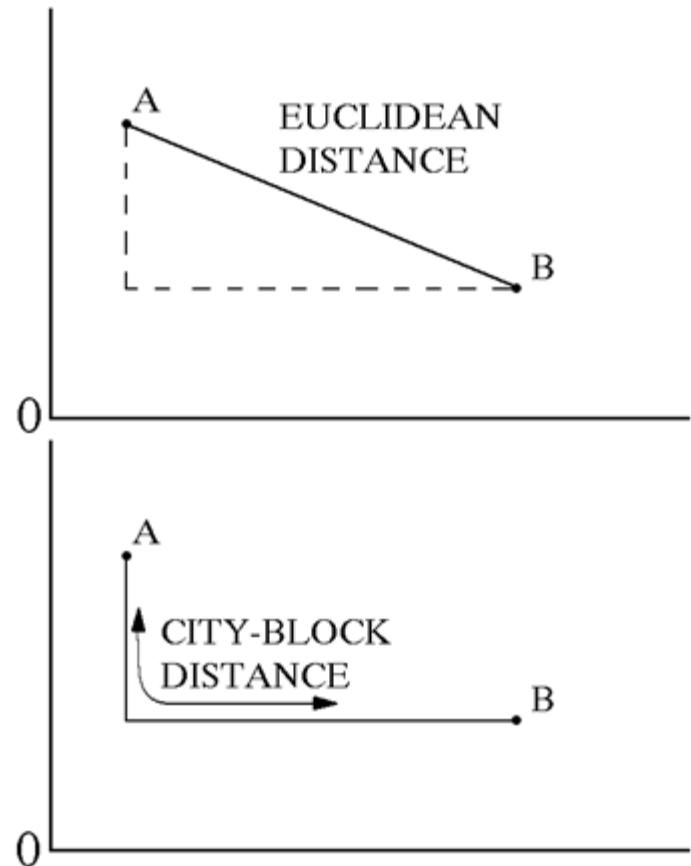
Manhattan and Euclidean Distance

- Euclidean distance

$$ED_{i,h} = \sqrt{\sum_{j=1}^p (a_{i,j} - a_{h,j})^2}$$

- Manhattan distance
 - =city-block distance

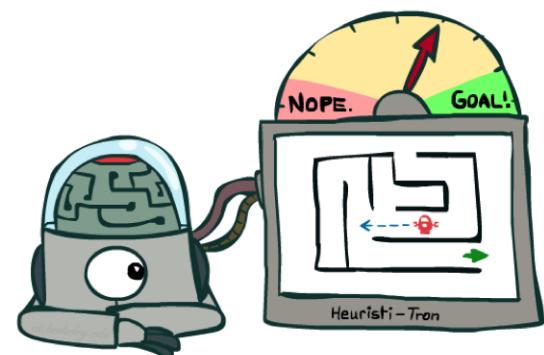
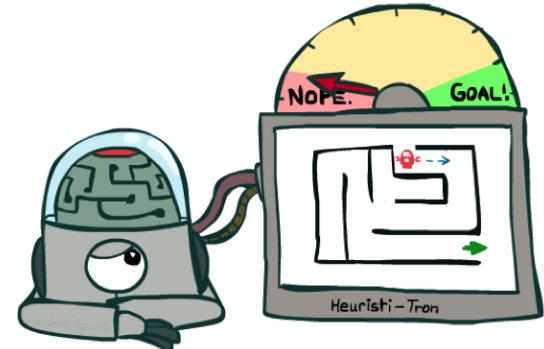
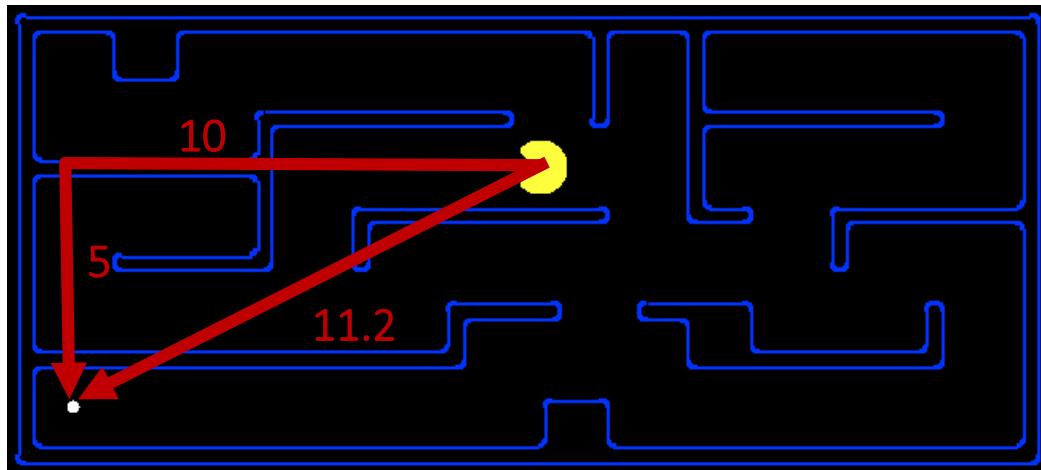
$$CB_{i,h} = \sum_{j=1}^p |a_{i,j} - a_{h,j}|$$



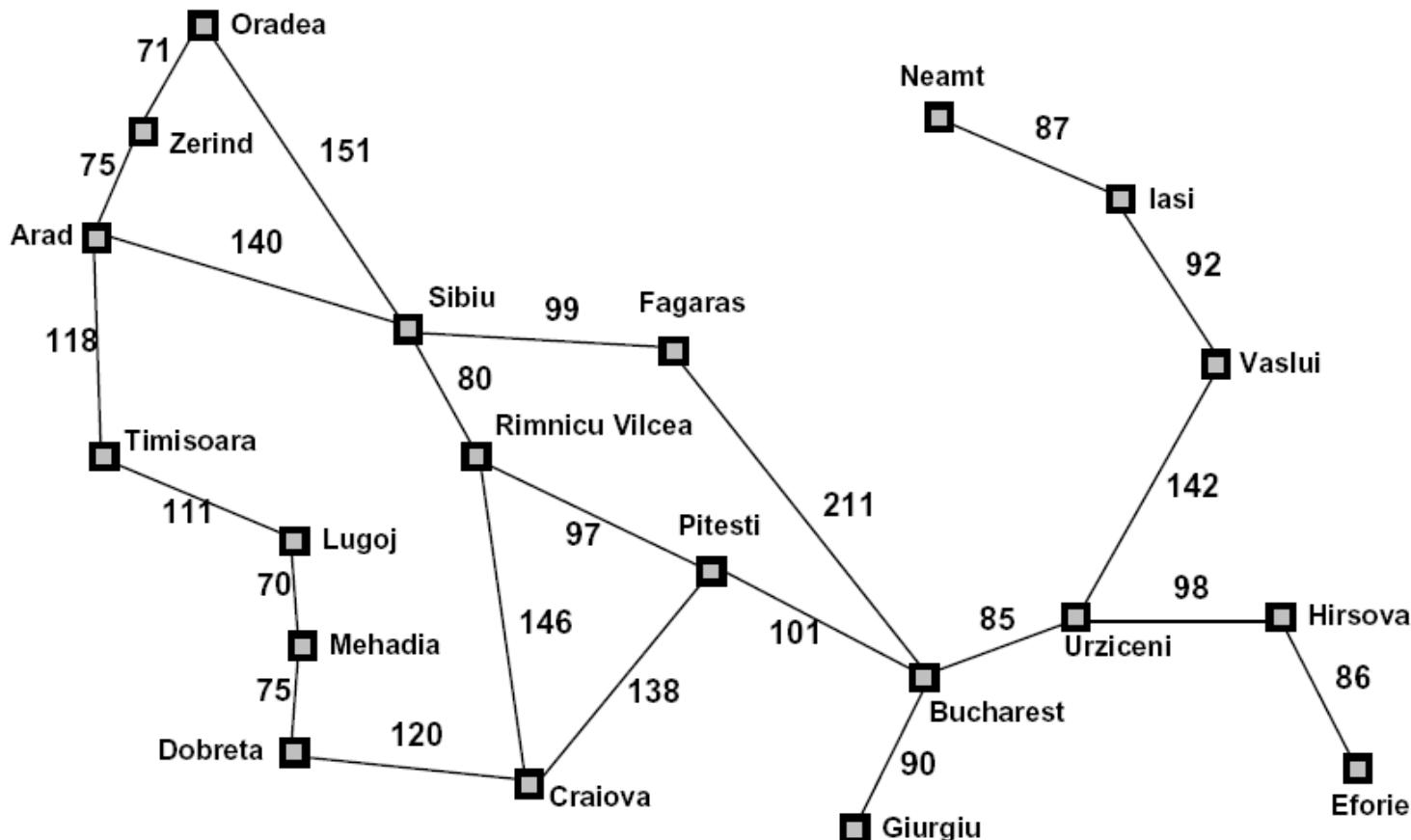
Search Heuristics

■ A heuristic is:

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- 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

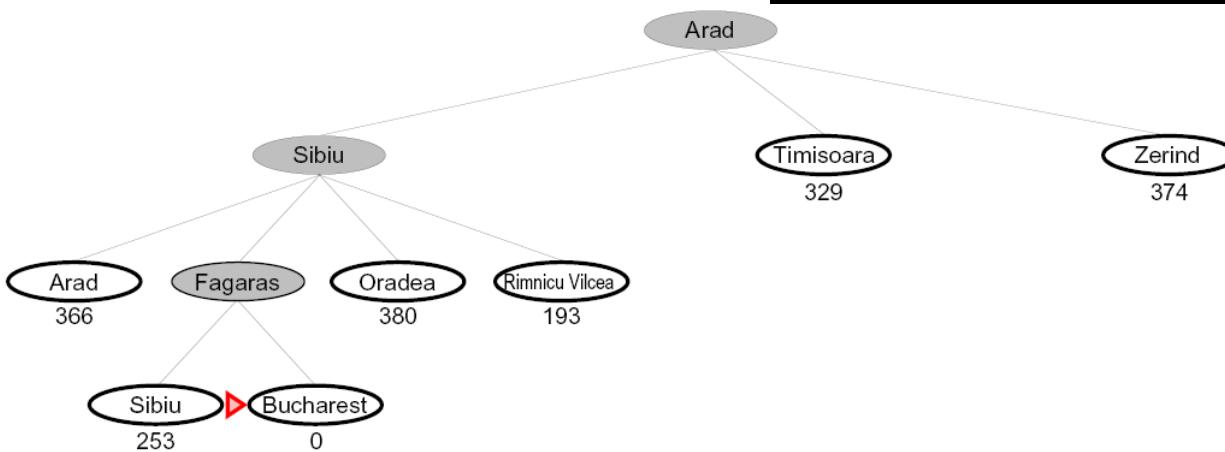
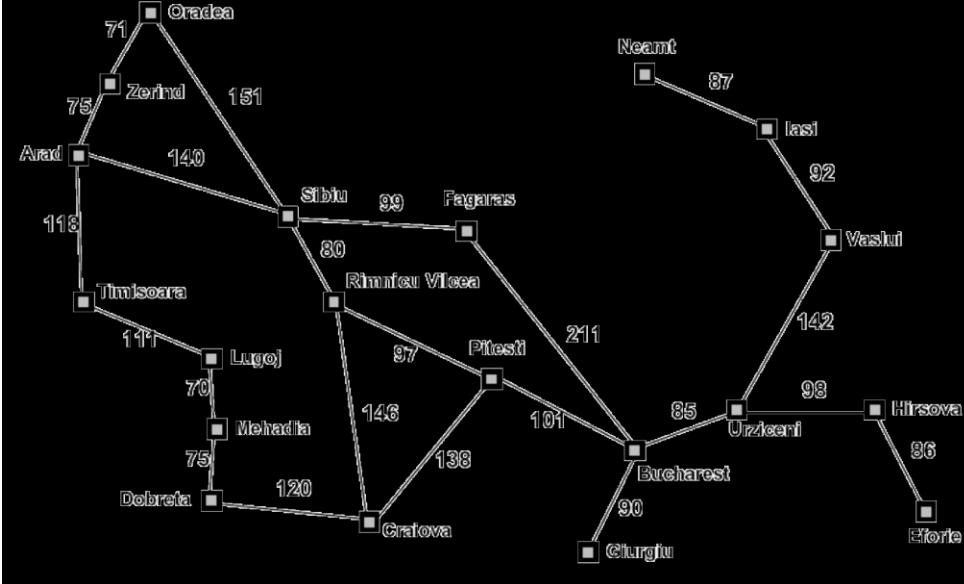
$h(x)$

Greedy Search

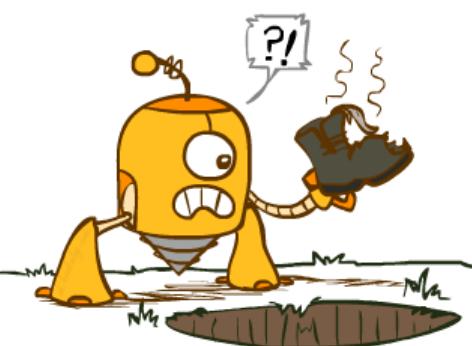


Greedy Search

- Expand the node that seems closest...



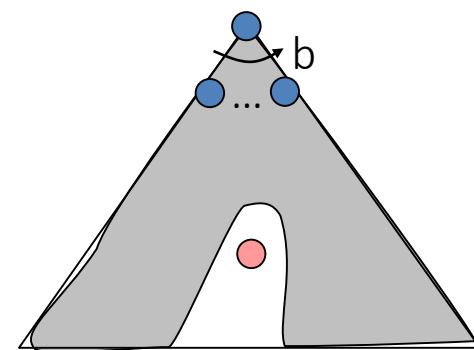
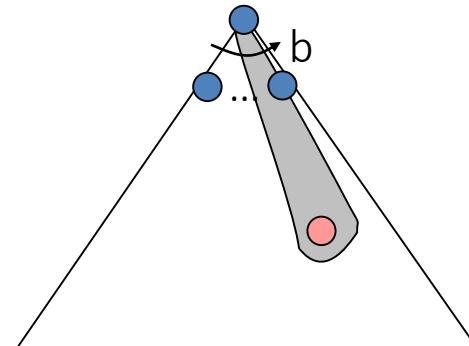
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



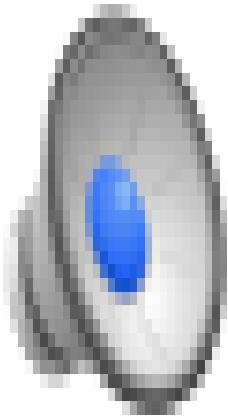
- What can go wrong?

Greedy Search

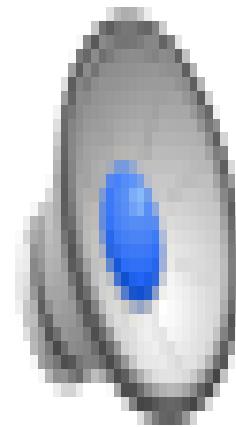
- Strategy: expand a node that you think is closest to a goal state
 - Heuristic: estimate of distance to nearest goal for each state
- A common case:
 - Best-first takes you straight to the (wrong) goal
- Worst-case: like a badly-guided DFS



Video of Demo Contours Greedy (Empty)



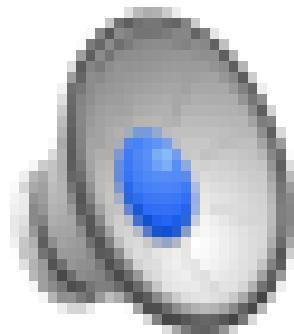
UCF



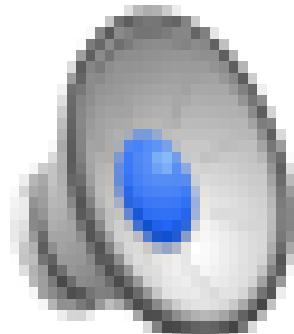
Greedy

Video of Demo Contours Greedy (Pacman Small Maze)

UCF



Greedy



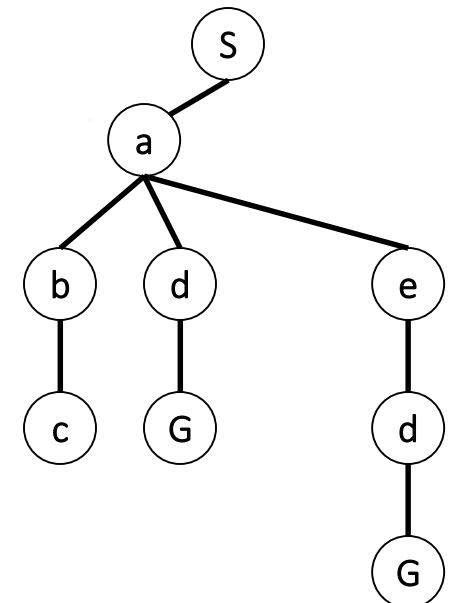
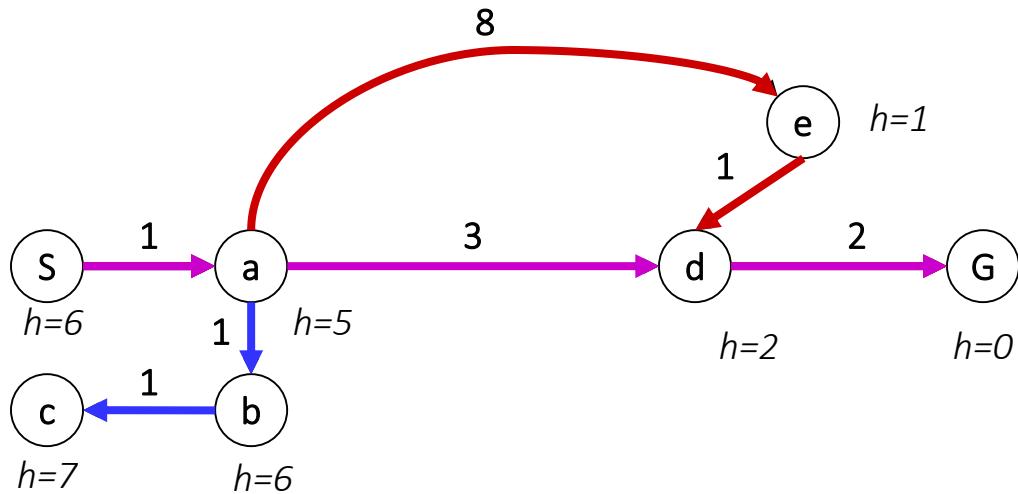
A* Search



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* $h(n)$

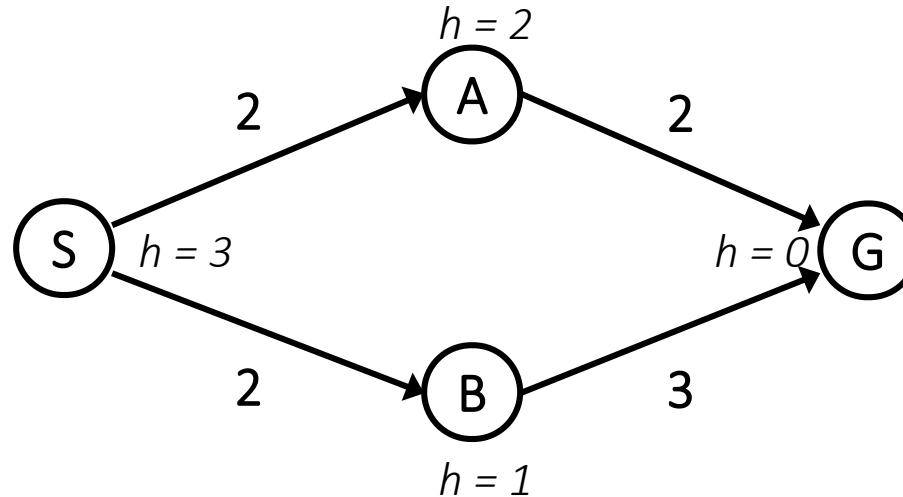


- A* Search orders by the sum: $f(n) = g(n) + h(n)$

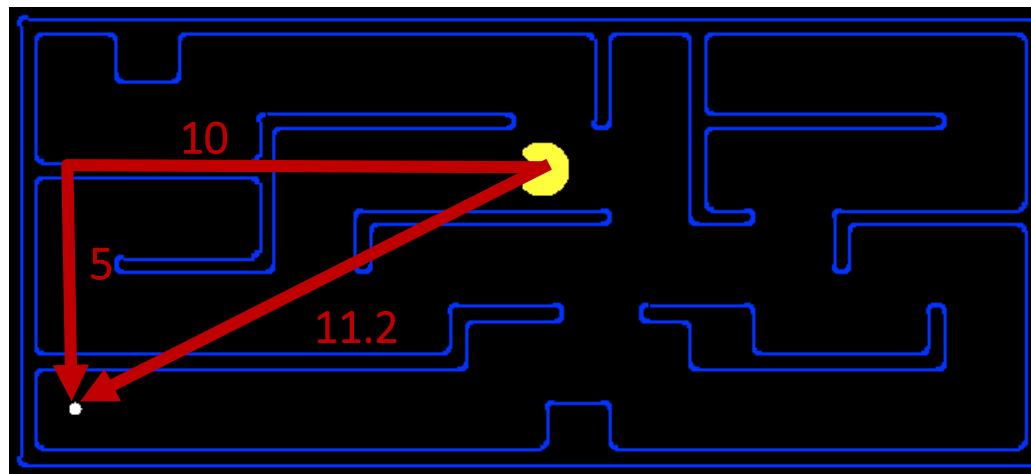
Example: Teg Grenager

When should A* terminate?

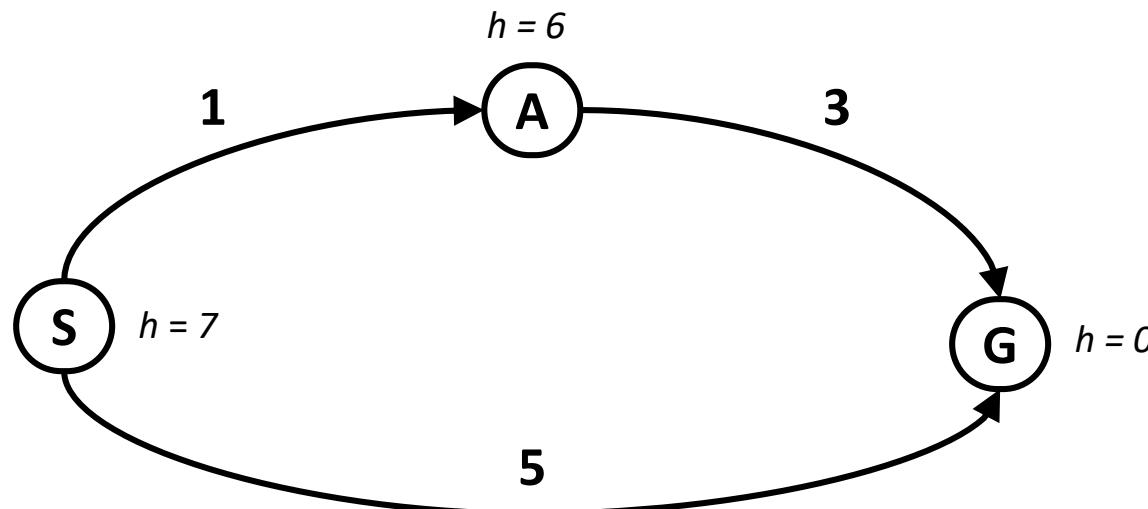
- Should we stop when we **enqueue** a goal?



- No: only stop when we **dequeue** a goal

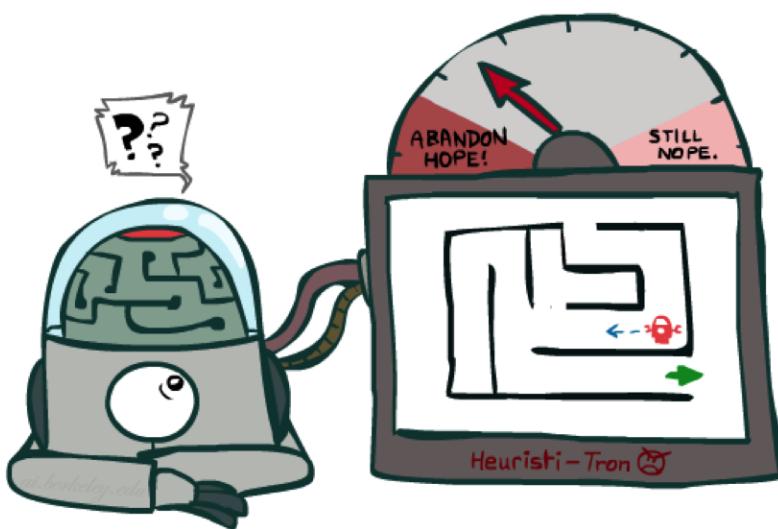


Is A* Optimal?

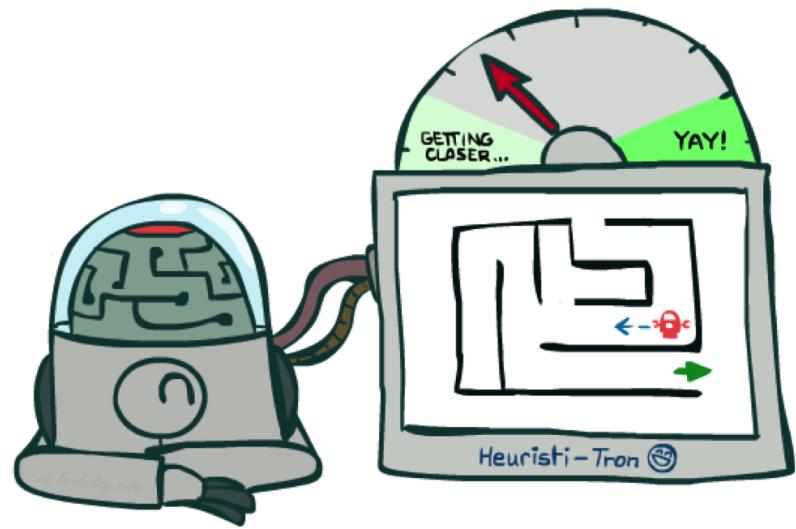


- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!

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

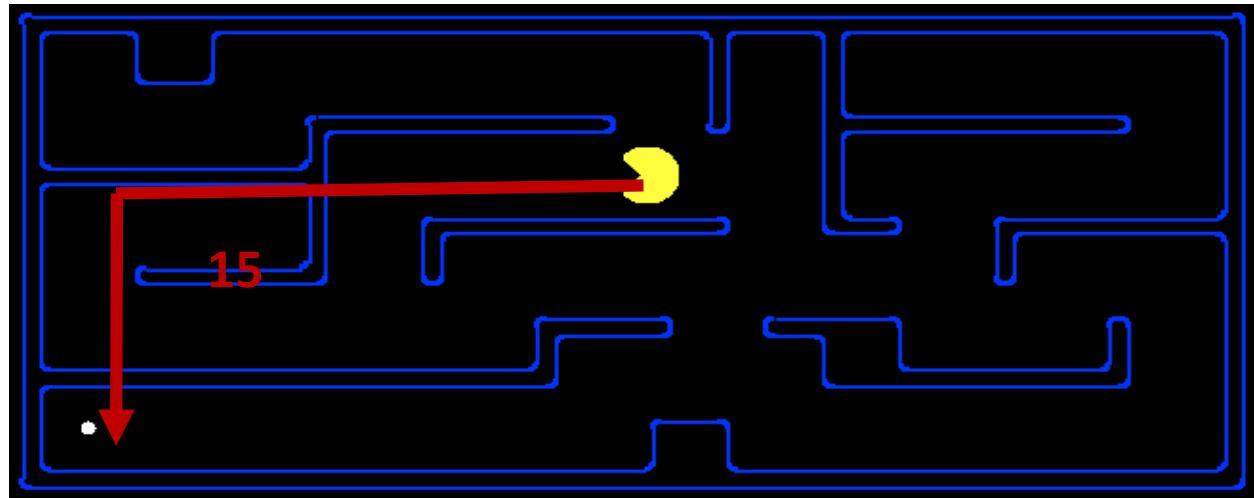
Admissible Heuristics

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

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

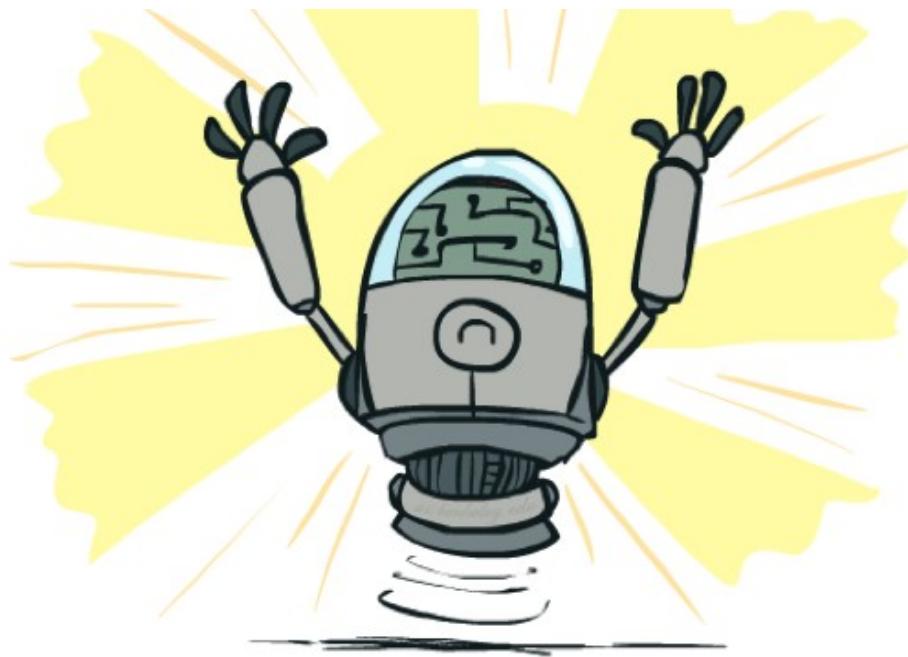
where $h^*(n)$ is the true cost to a nearest goal

- Example:



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

Optimality of A* Tree Search



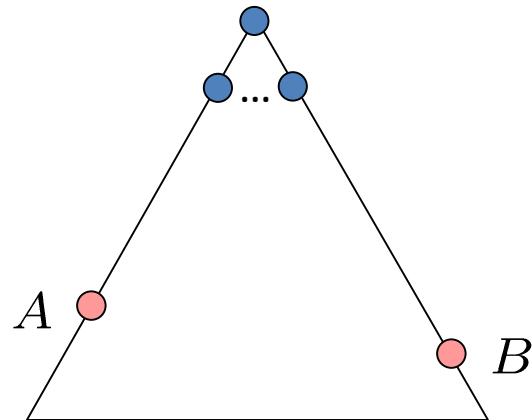
Optimality of A* Tree Search

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:

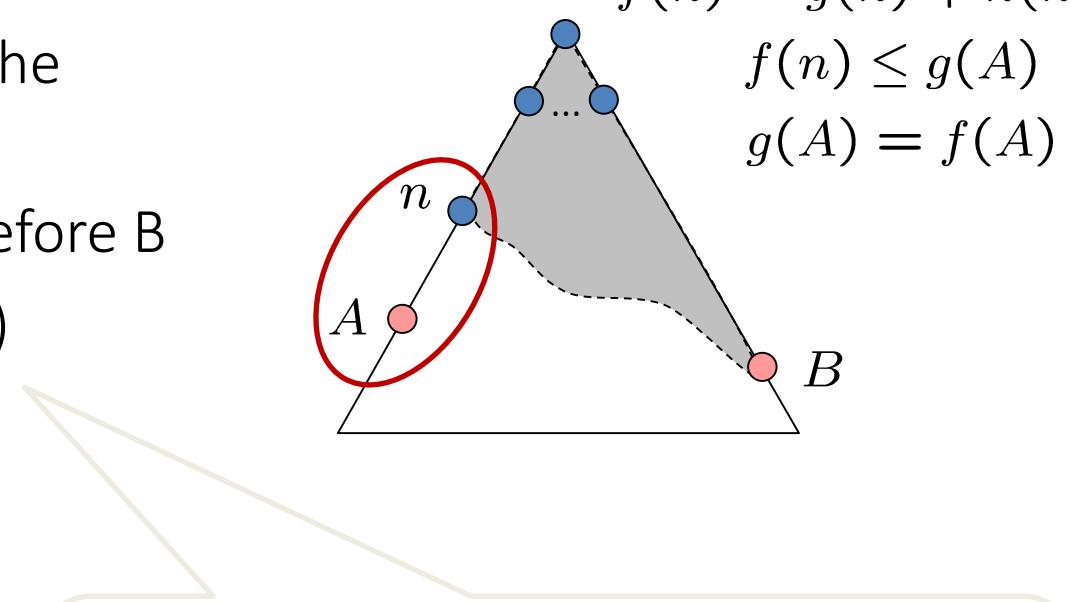
- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will be expanded before B
 1. $f(n)$ is less or equal to $f(A)$

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

$$f(n) = g(n) + h(n)$$

$$f(n) \leq g(A)$$

$$g(A) = f(A)$$



$$f(n) = g(n) + h(n)$$

$$f(n) \leq g(A)$$

$$g(A) = f(A)$$

Definition of f-cost

Admissibility of h

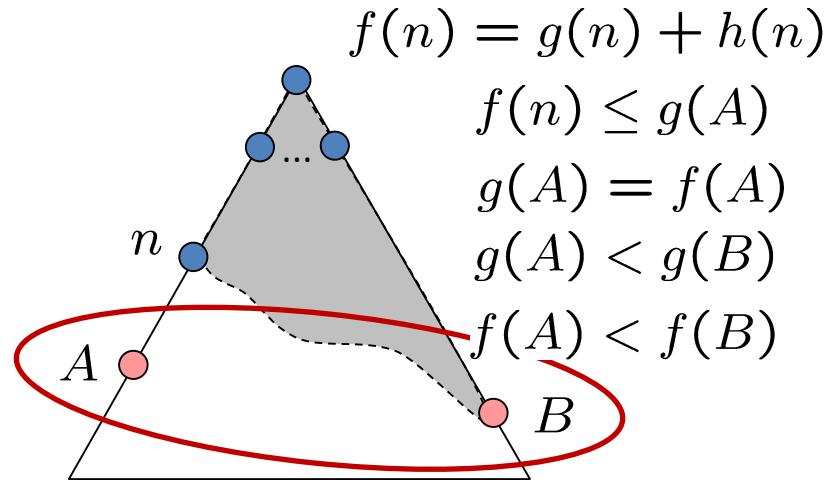
$h = 0$ at a goal

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 $f(B)$

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



$$\begin{aligned} g(A) &< g(B) \\ f(A) &< f(B) \end{aligned}$$

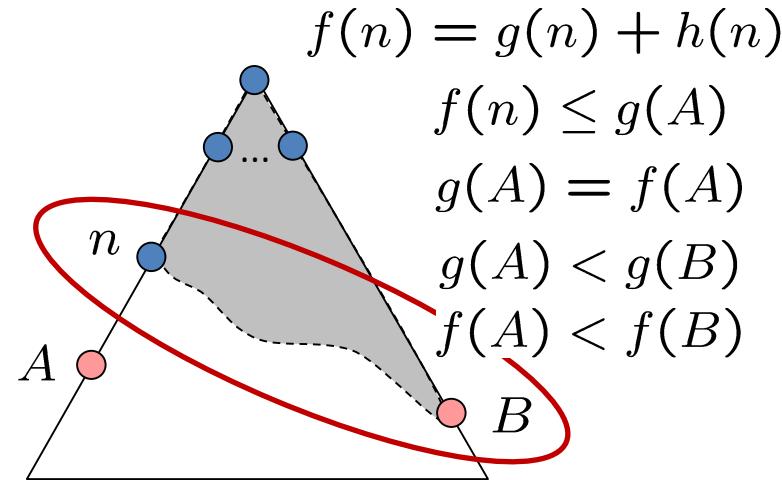
B is suboptimal
 $h = 0$ at a goal

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

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

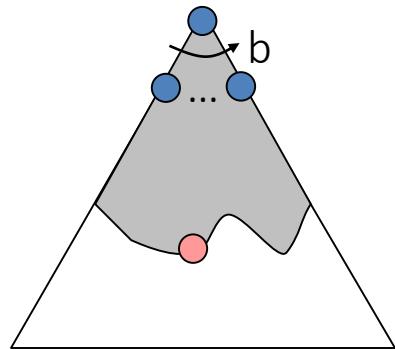


$$f(n) \leq f(A) < f(B)$$

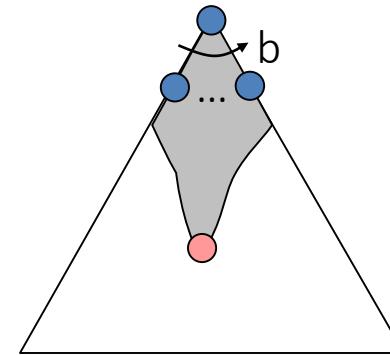
Properties of A*

Properties of A*

Uniform-Cost

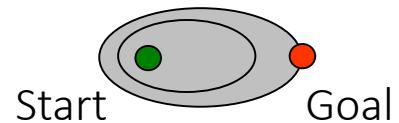
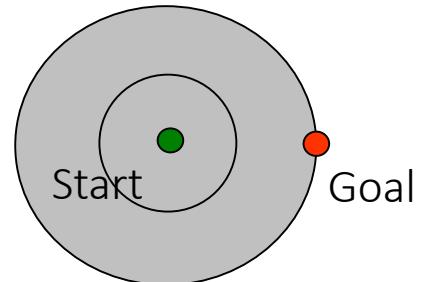


A*



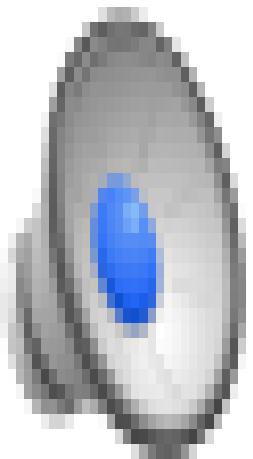
UCS vs A* Contours

- Uniform-cost expands equally in all “directions”
- A* expands mainly toward the goal, but does hedge its bets to ensure optimality

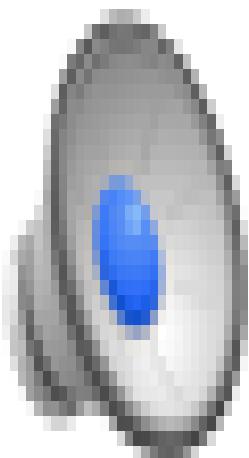


[Demo: contours UCS / greedy / A* empty (L3D1)]
[Demo: contours A* pacman small maze (L3D5)]

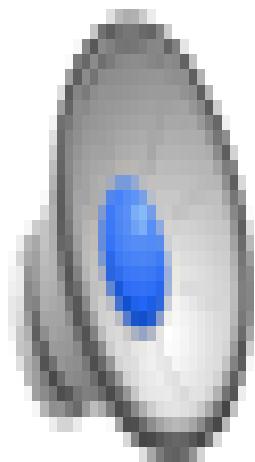
Video of Demo Contours (Empty) -- UCS



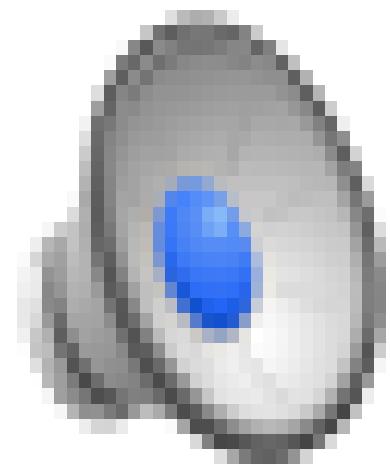
Video of Demo Contours (Empty) -- Greedy



Video of Demo Contours (Empty) – A*



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



Comparison



Greedy



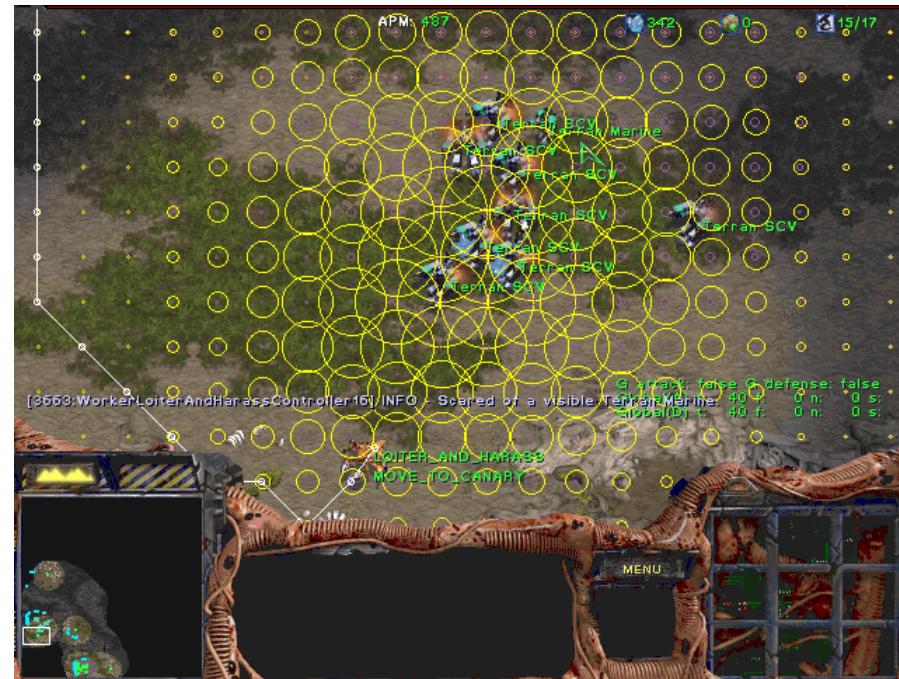
Uniform Cost



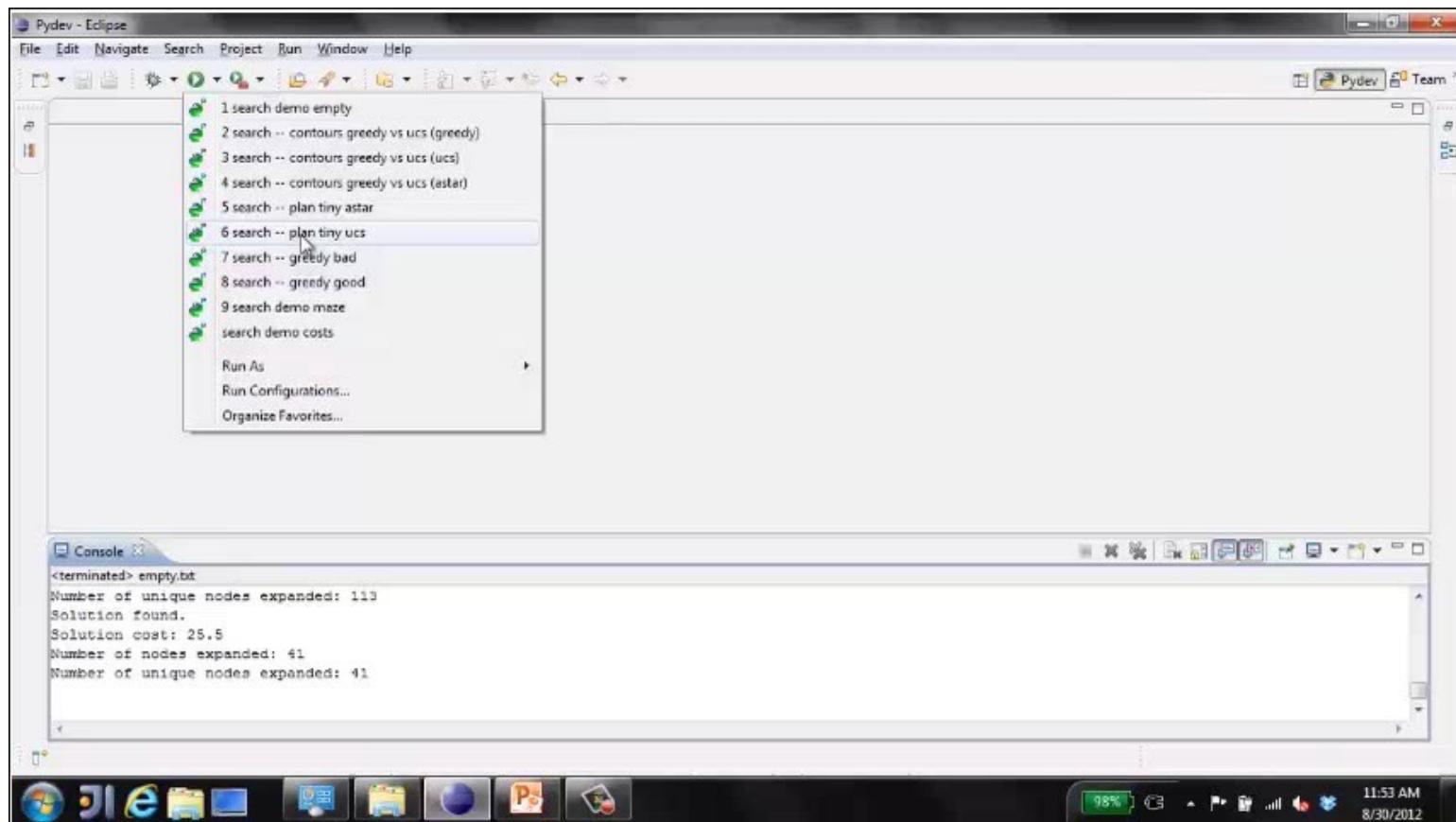
A*

A* Applications

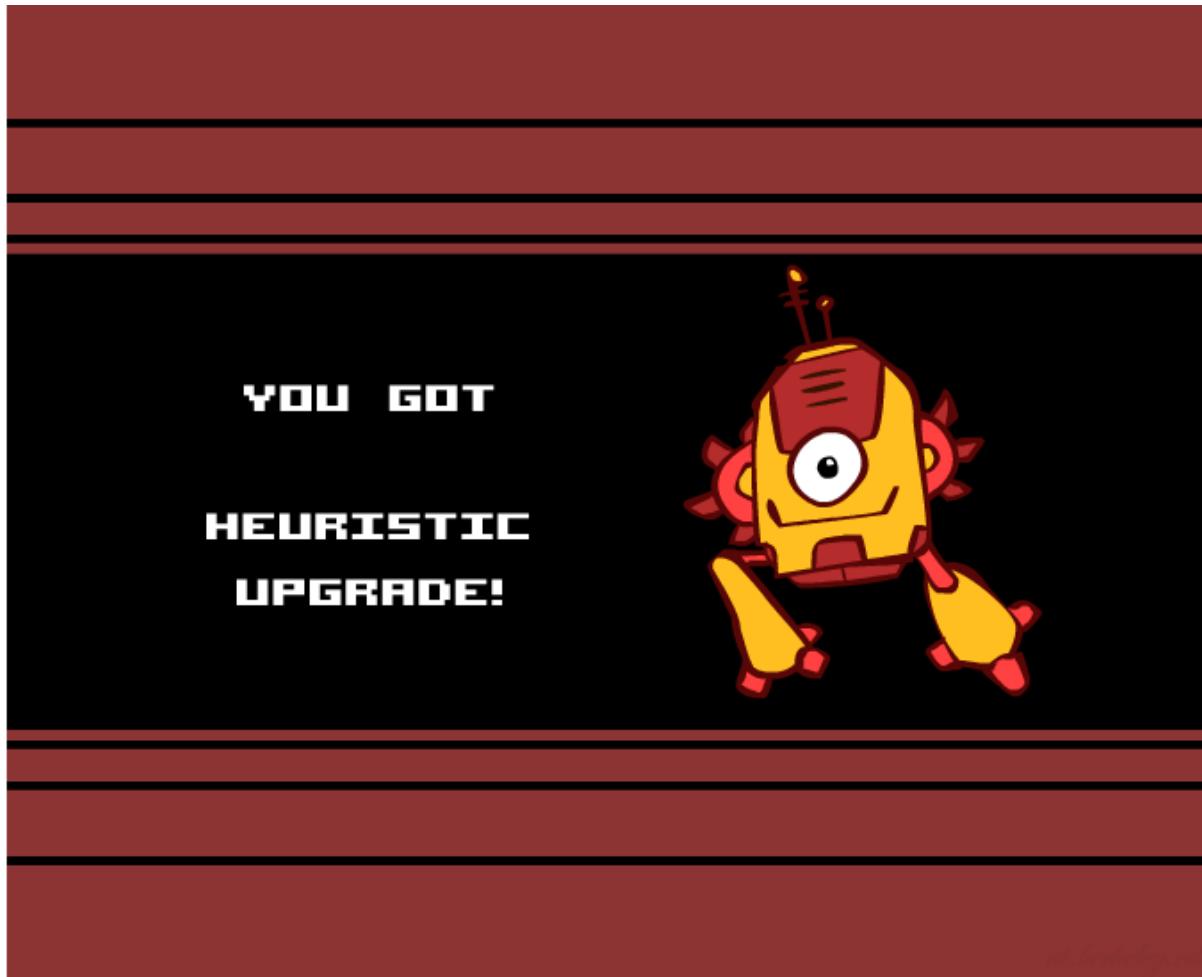
- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- ...



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



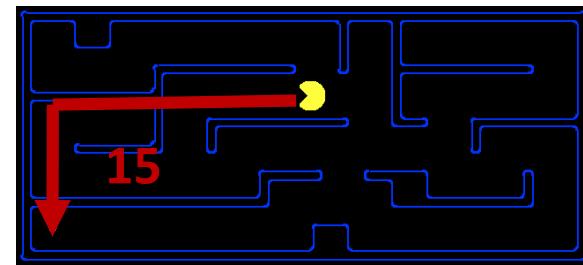
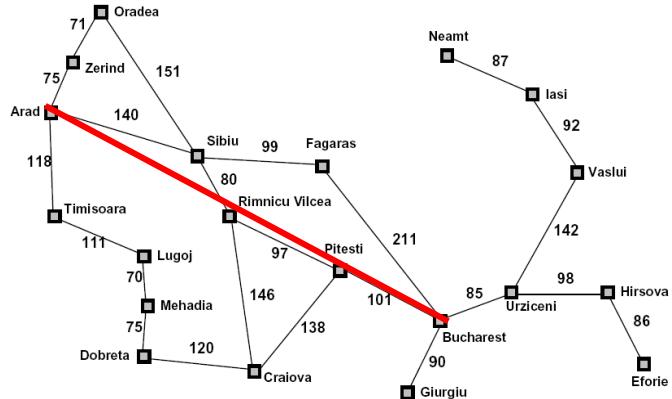
Creating Heuristics



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

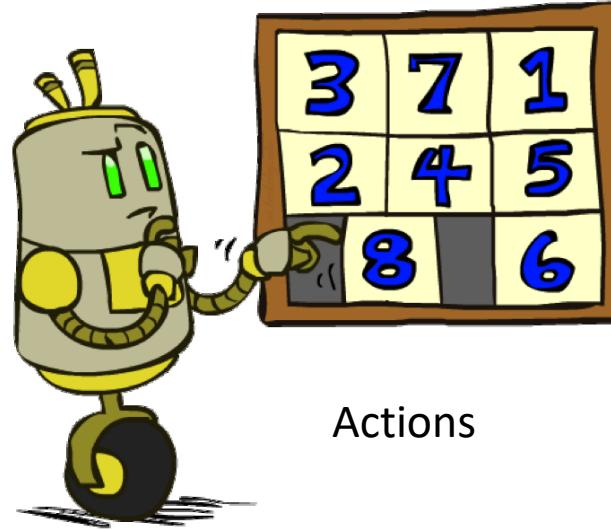


- Inadmissible heuristics are often useful too

Example: 8 Puzzle

7	2	4
5		6
8	3	1

Start State



Actions

	1	2
3	4	5
6	7	8

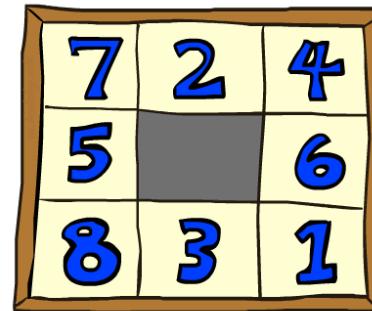
Goal State

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

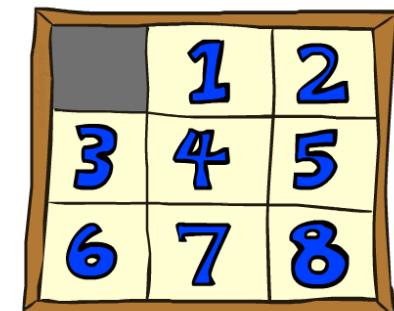
https://en.wikipedia.org/wiki/15_puzzle#Solvability

8 Puzzle I

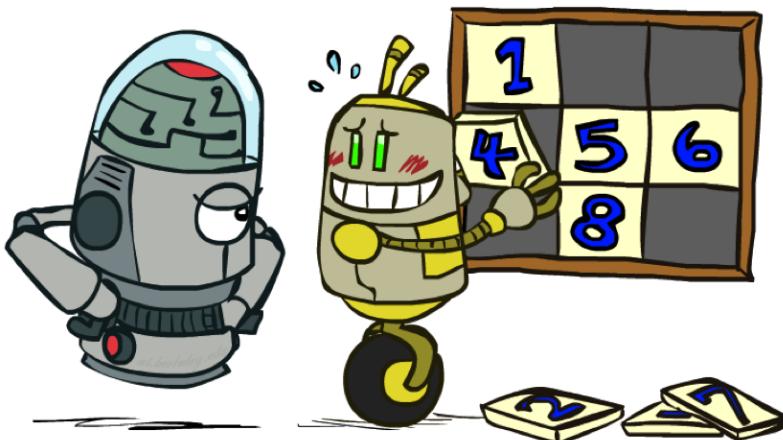
- Heuristic: Number of tiles misplaced
- Why is it admissible?
- $h(\text{start}) = 8$
- This is a *relaxed-problem* heuristic



Start State



Goal State

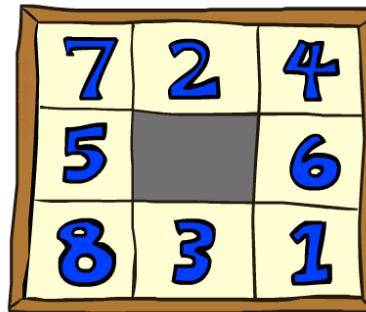


Average nodes expanded when the optimal path has...			
	...4 steps	...8 steps	...12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

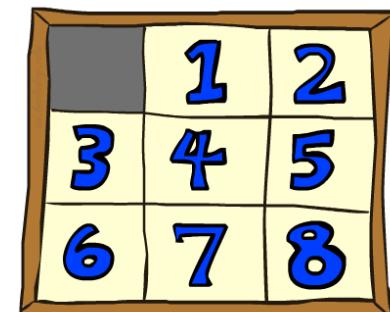
Statistics from Andrew Moore

8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?



Start State



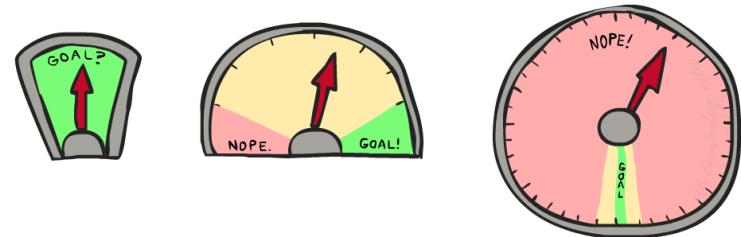
Goal State

- Total *Manhattan* distance
- $h(\text{start}) = 3 + 1 + 2 + \dots = 18$
- Why is it admissible?

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

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

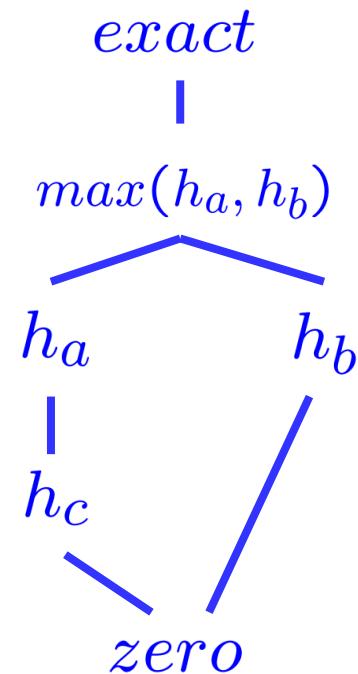
- Heuristics form a semi-lattice:

- Max of admissible heuristics is admissible

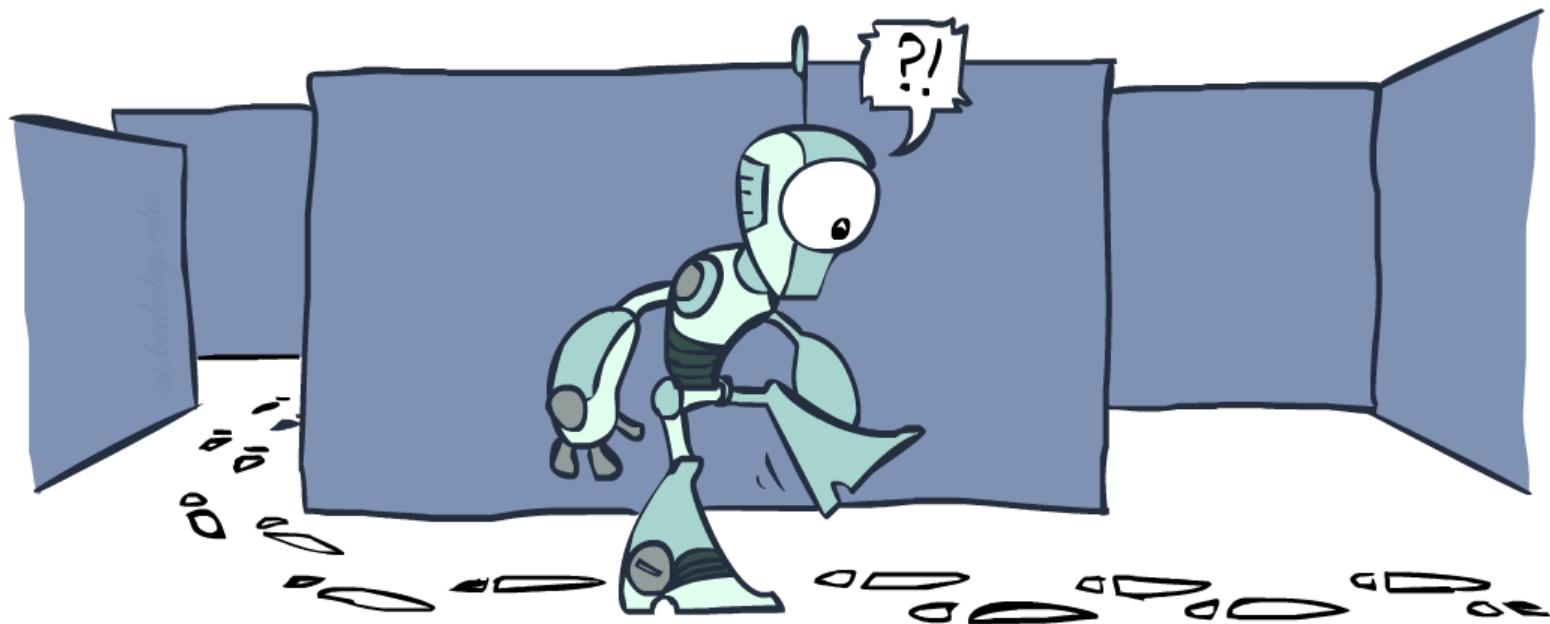
$$h(n) = \max(h_a(n), h_b(n))$$

- Trivial heuristics

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

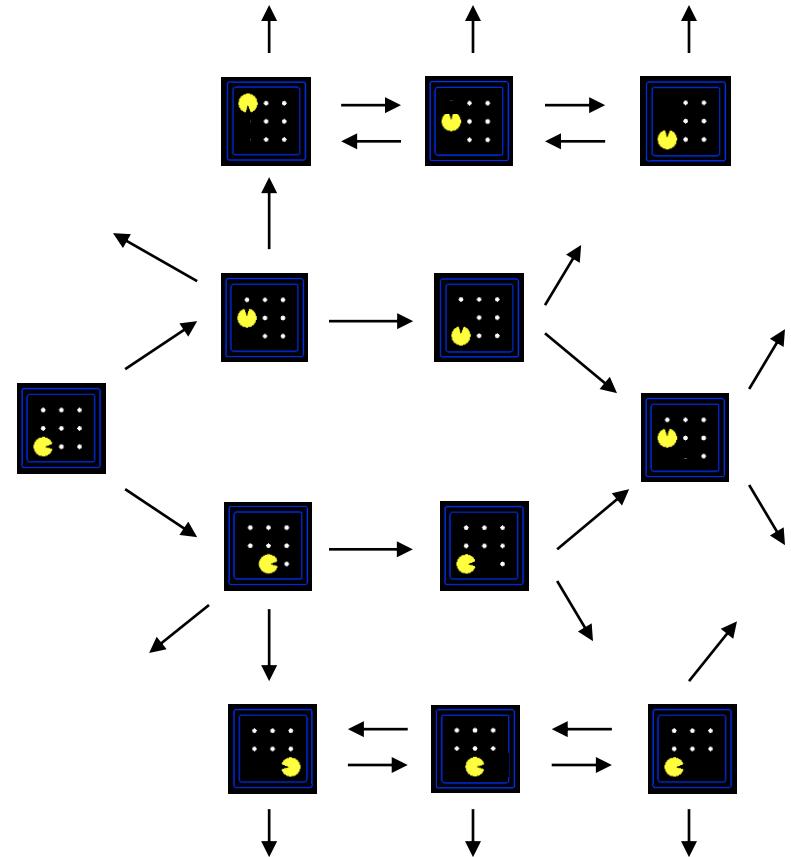


Graph Search



State Space Graphs

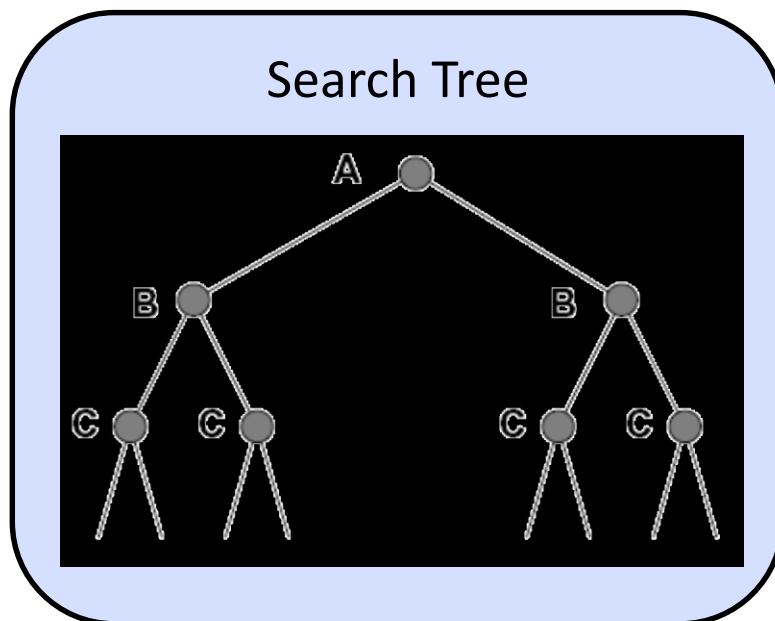
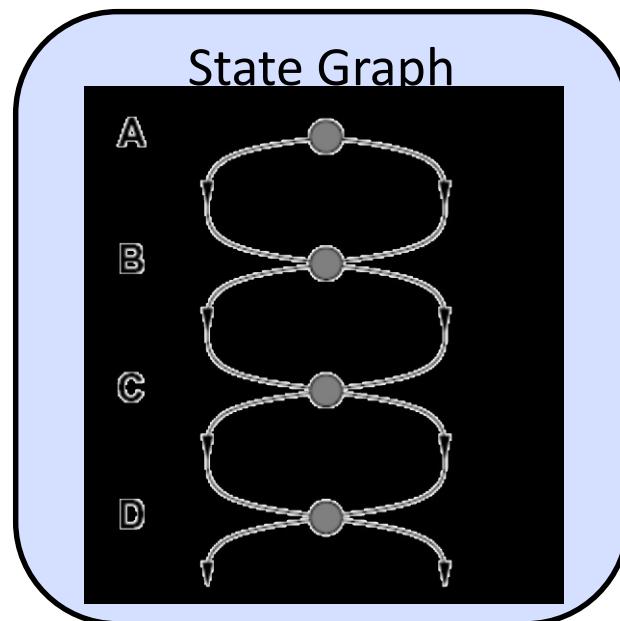
- State space graph: A mathematical representation of a search problem
 - Nodes are (abstracted) world configurations
 - Arcs represent successors (action results)
 - The goal test is a set of goal nodes (maybe only one)
- In a state space graph, each state occurs only once!
- We can rarely build this full graph in memory (it's too big), but it's a useful idea



World states? $120 \times (2^{30}) \times (12^2) \times 4$

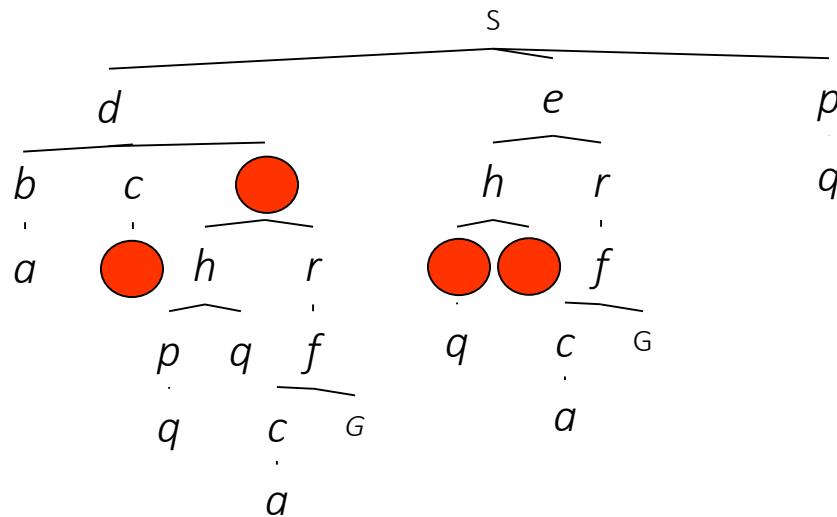
Tree Search: Extra Work!

- 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?)

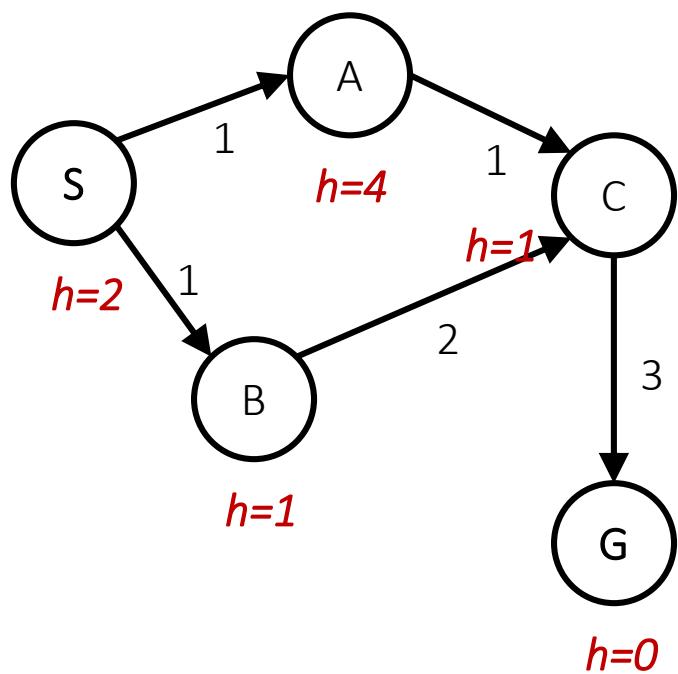


Graph Search

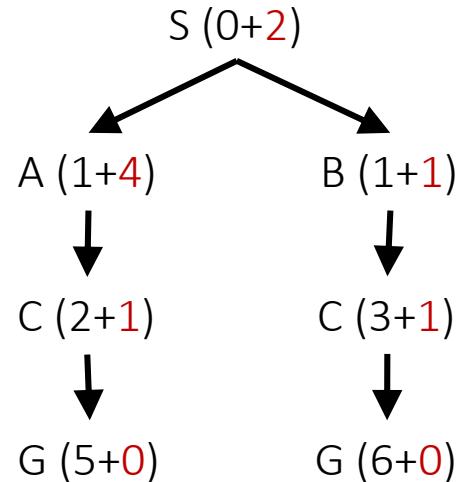
- Idea: never **expand** a state twice
- How to implement:
 - Tree search + set of expanded states (“closed set”)
 - Expand the search tree node-by-node, but...
 - Before **expanding a node**, check to make sure its state has **never been expanded before**
 - If not new, skip it, if new add to closed set
- Important: **store the closed set as a set**, not a list

A* Graph Search Gone Wrong?

State space graph

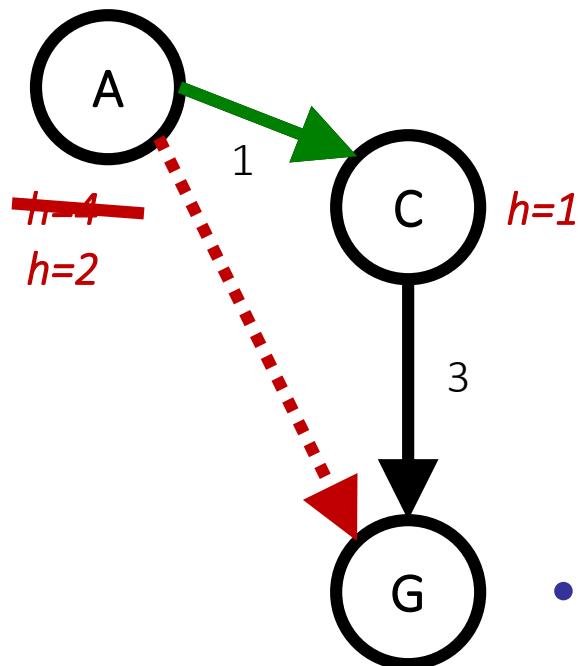


Search tree



Consistency of Heuristics

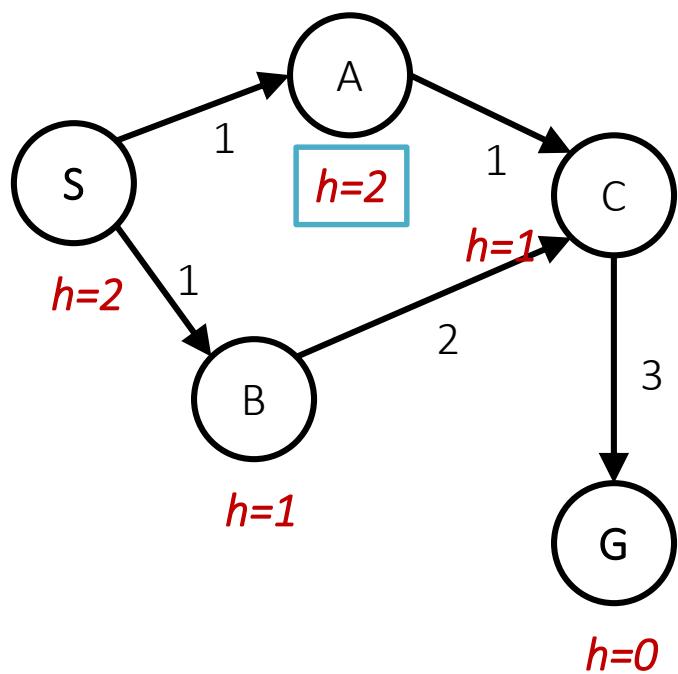
- Main idea: estimated heuristic costs \leq actual costs



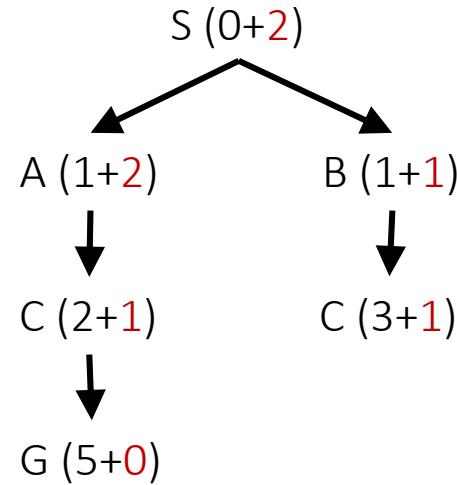
- Admissibility: heuristic cost \leq actual cost to goal
$$h(A) \leq \text{actual cost from } A \text{ to } G$$
- Consistency: heuristic “arc” cost \leq actual cost for each arc
$$h(A) - h(C) \leq \text{cost}(A \text{ to } C)$$
- Consequences of consistency:
 - The f value along a path never decreases
$$h(A) \leq \text{cost}(A \text{ to } C) + h(C)$$
 - A* graph search is optimal

A* Graph Search Gone Wrong?

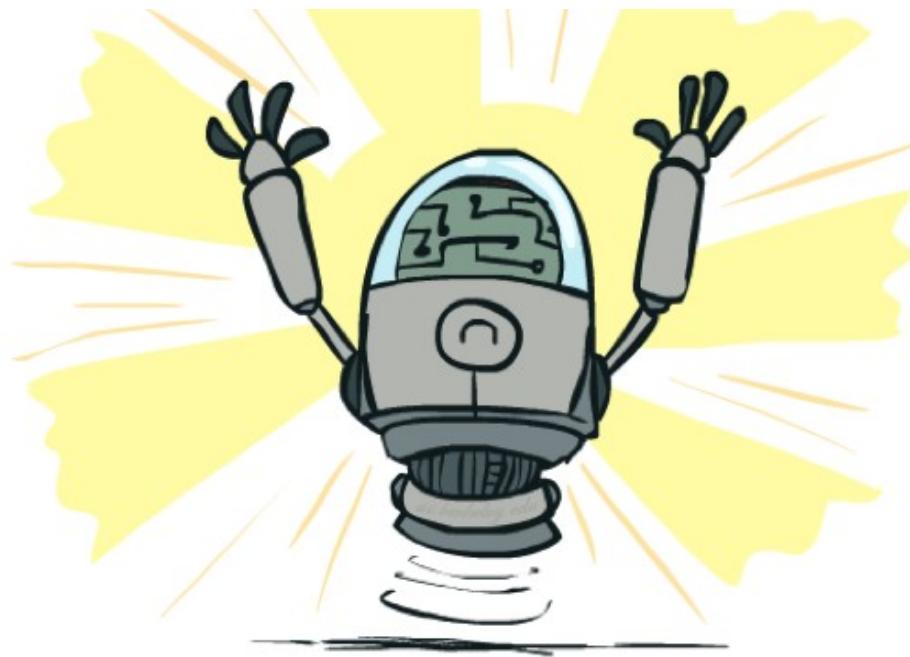
State space graph



Search tree

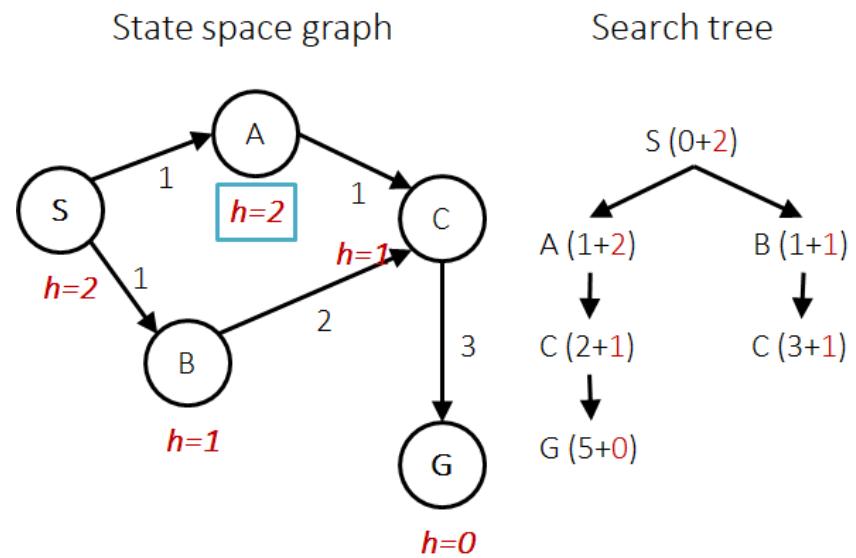
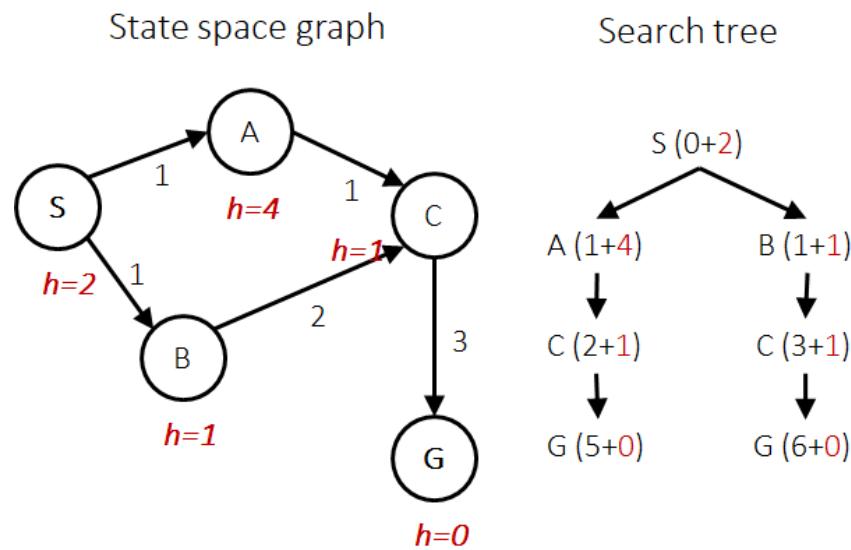


Optimality of A* Graph Search



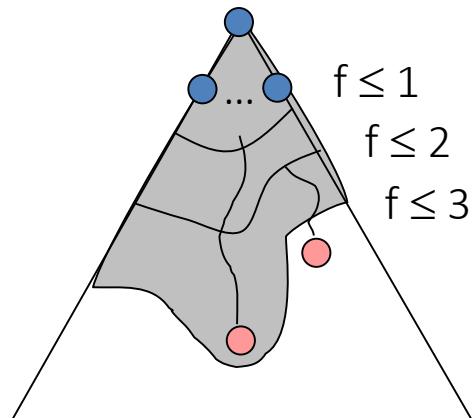
Optimality of A* Graph Search

- Consider what A* does:
 - Expands nodes in increasing total f value (f-contours)
Reminder: $f(n) = g(n) + h(n) = \text{cost to } n + \text{heuristic}$
 - Proof idea: the optimal goal(s) have the lowest f value, so it must get expanded first



Optimality of A* Graph Search

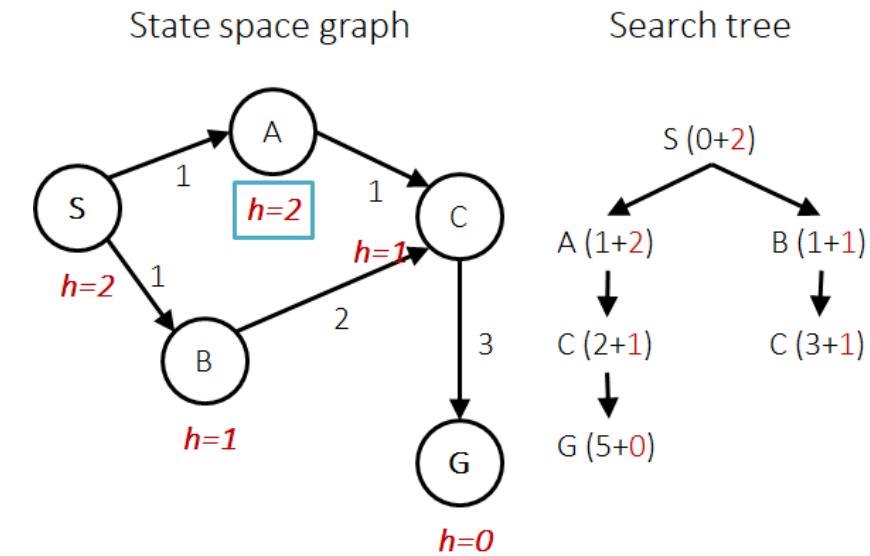
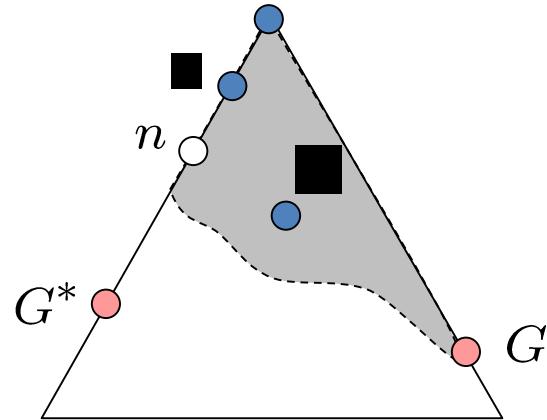
- Sketch: consider what A* does with a **consistent** heuristic:
 - Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)
 - Fact 2: For every state s, nodes that reach s optimally are expanded before nodes that reach s suboptimally
 - Result: A* graph search is optimal



Optimality of A* Graph Search

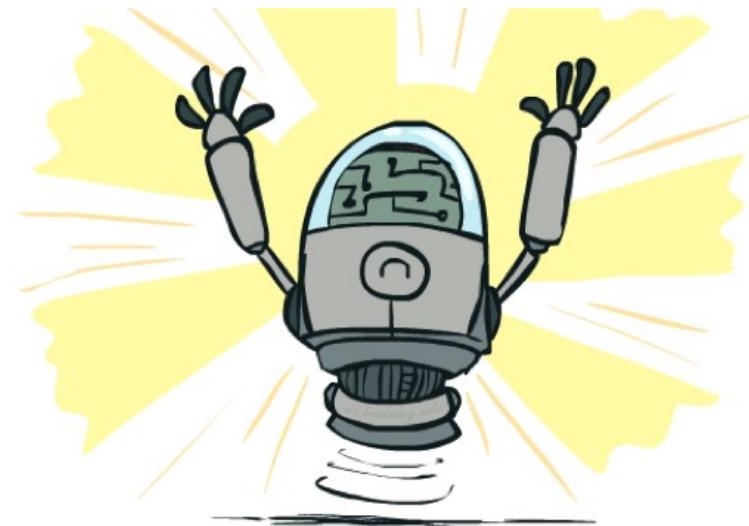
Proof:

- New possible problem: some n on path to G^* isn't in queue when we need it, because some worse n' for the same state dequeued and expanded first (disaster!)
- Take the highest such n in tree
- Let p be the ancestor of n that was on the queue when n' was popped
- $f(p) < f(n)$ because of consistency
- $f(n) < f(n')$ because n' is suboptimal
- p would have been expanded before n'
- Contradiction!



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



Tree/ Graph Search Pseudo-Code

```
function TREE-SEARCH(problem, fringe) return a solution, or failure
  fringe  $\leftarrow$  INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node  $\leftarrow$  REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    for child-node in EXPAND(STATE[node], problem) do
      fringe  $\leftarrow$  INSERT(child-node, fringe)
    end
  end
```

```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
  closed  $\leftarrow$  an empty set
  fringe  $\leftarrow$  INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node  $\leftarrow$  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  $\leftarrow$  INSERT(child-node, fringe)
    end
  end
```

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

