Lecture 3: Informed Search

Shuai Li

John Hopcroft Center, Shanghai Jiao Tong University

https://shuaili8.github.io

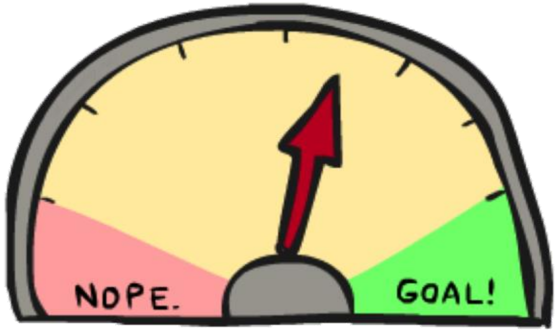
https://shuaili8.github.io/Teaching/CS410/index.html

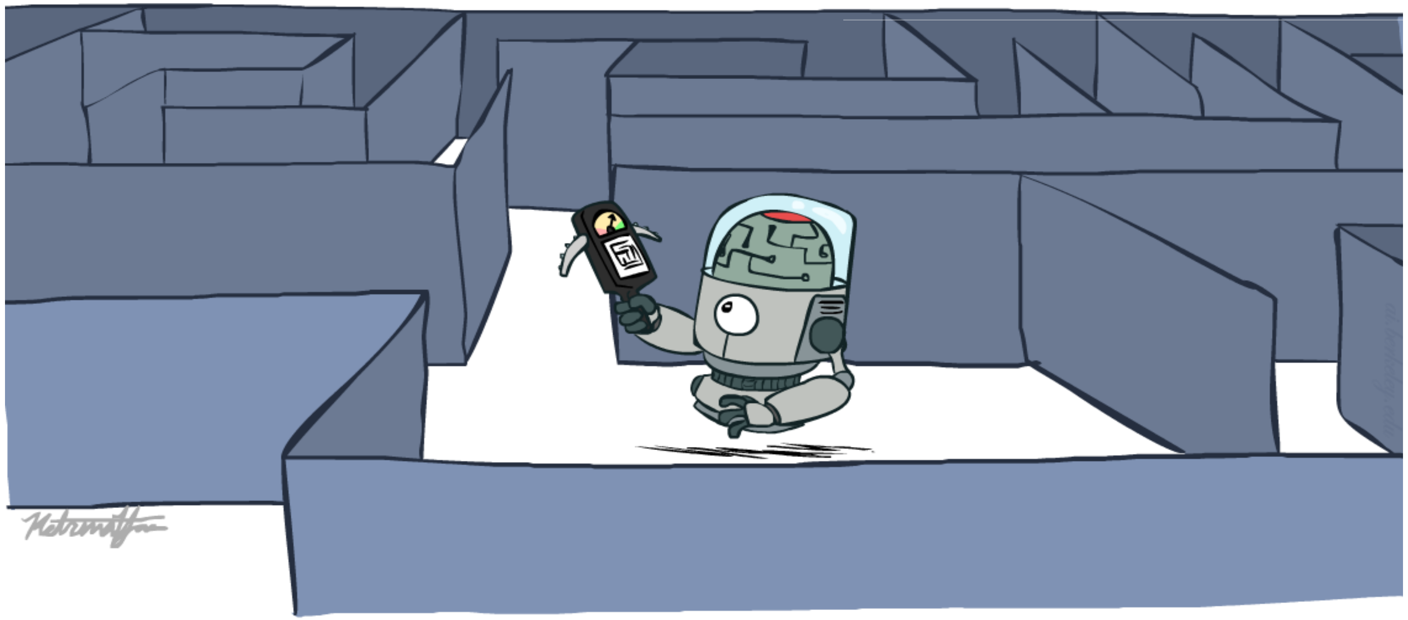
Part of slide credits: CMU AI & http://ai.berkeley.edu

Informed Search

- Uninformed Search
  - DFS
  - BFS
  - UCS

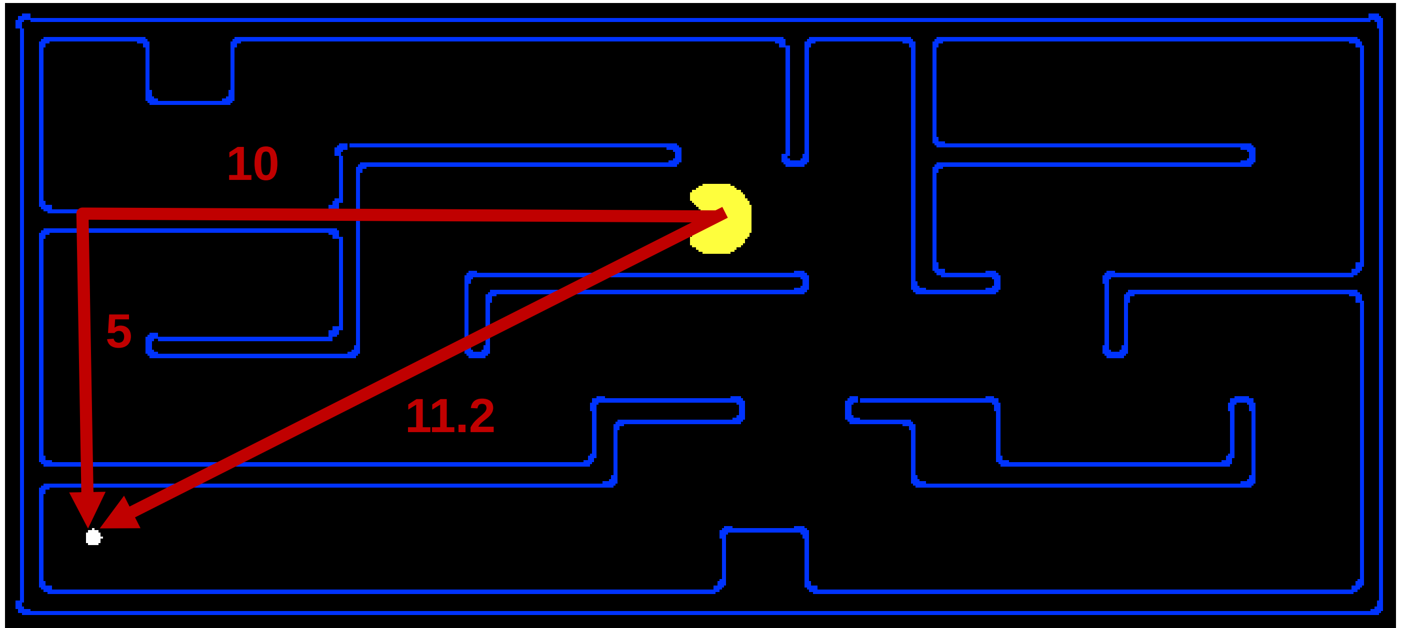
- Informed Search
  - Heuristics
  - Greedy Search
  - A* Search
  - Graph Search

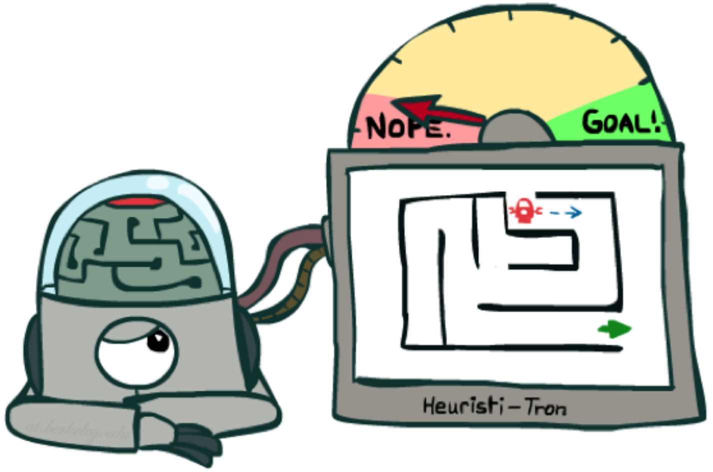


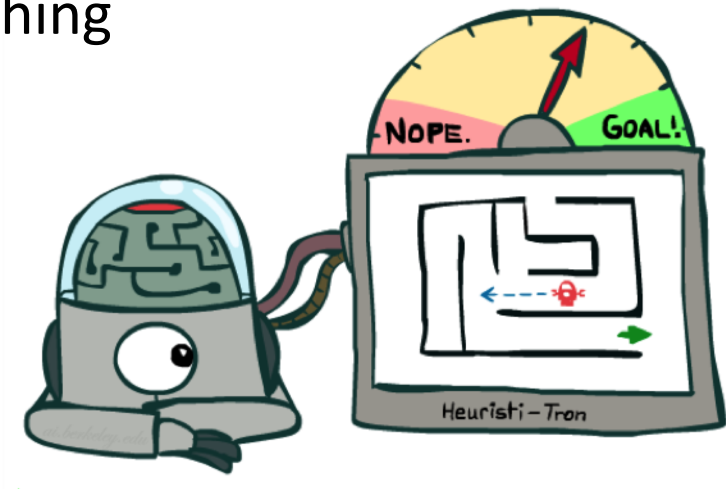


Search Heuristics

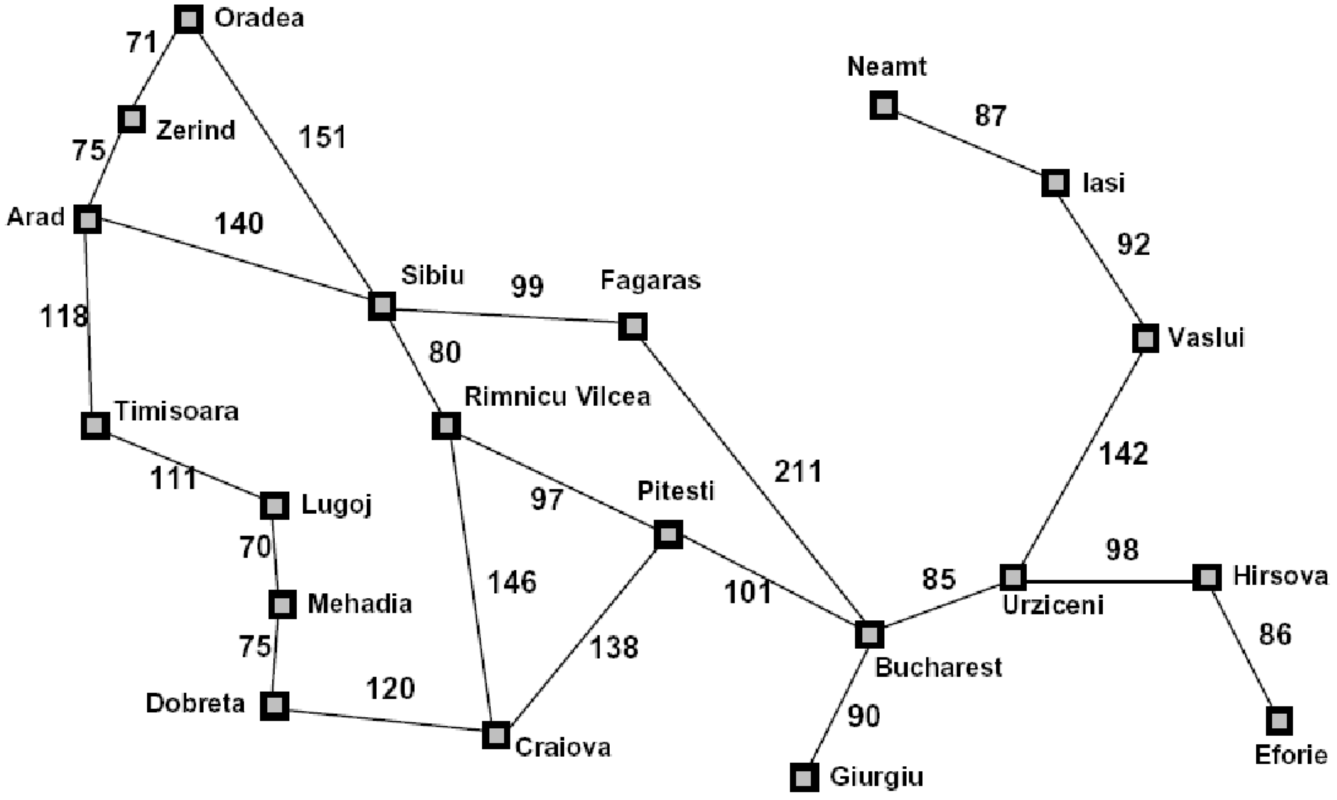
- A heuristic is:
  - A function that estimates how close a state is to a goal
  - Designed for a particular search problem
  - Pathing?
  - Examples: Manhattan distance, Euclidean distance for pathing







Example: Heuristic Function (Euclidean distance to Bucharest)



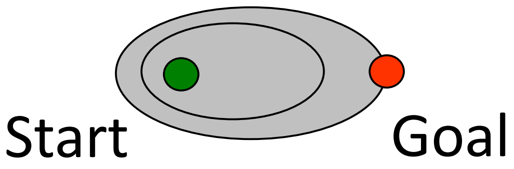
Straight−line distance to Bucharest

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

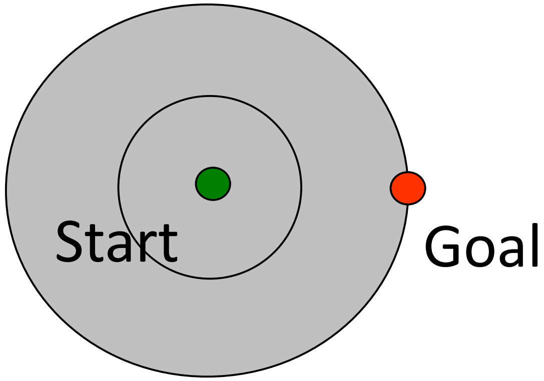
h(state) → value

Effect of heuristics

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



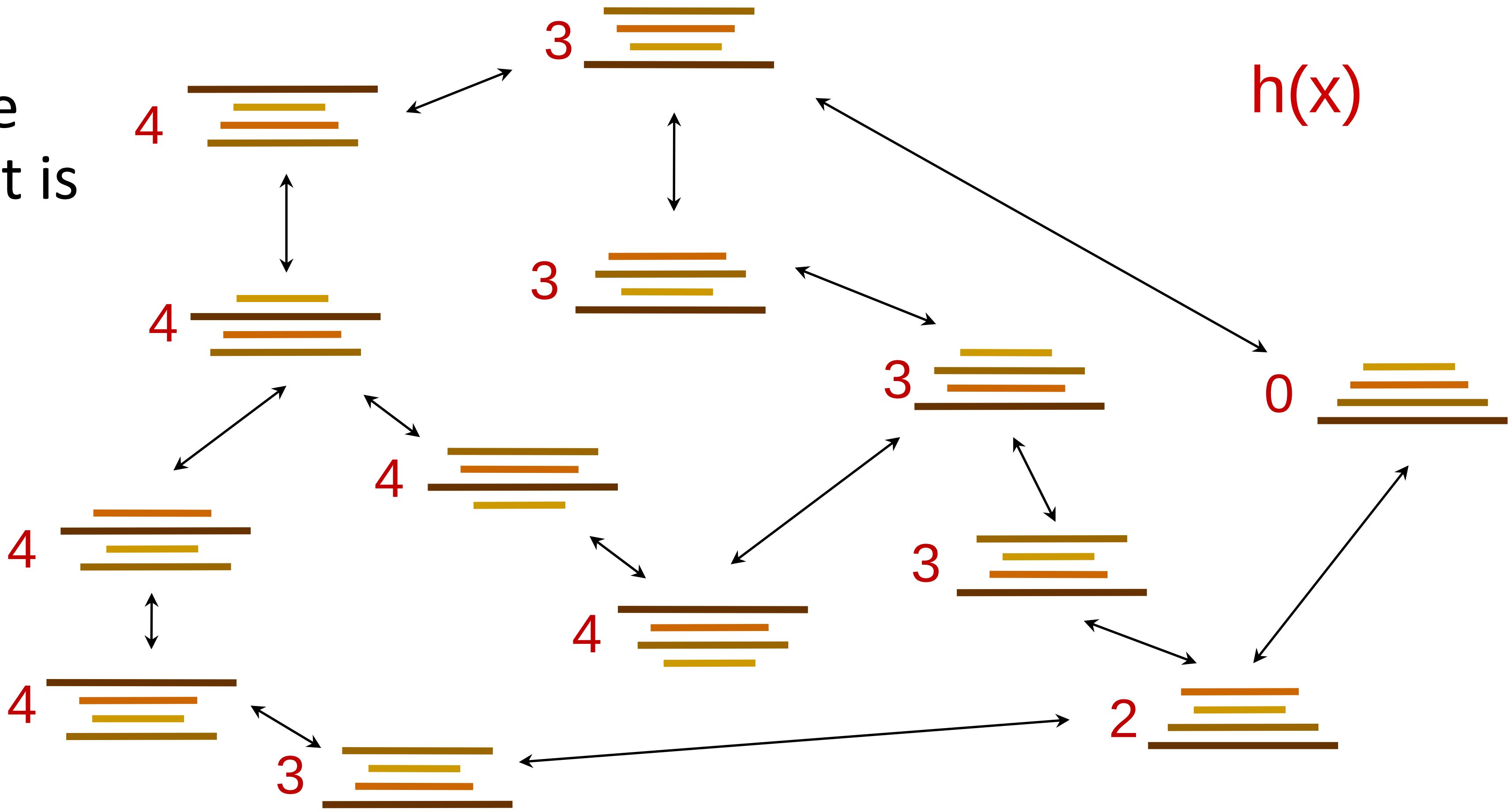
Informed



Uninformed

Example: Heuristic Function 2

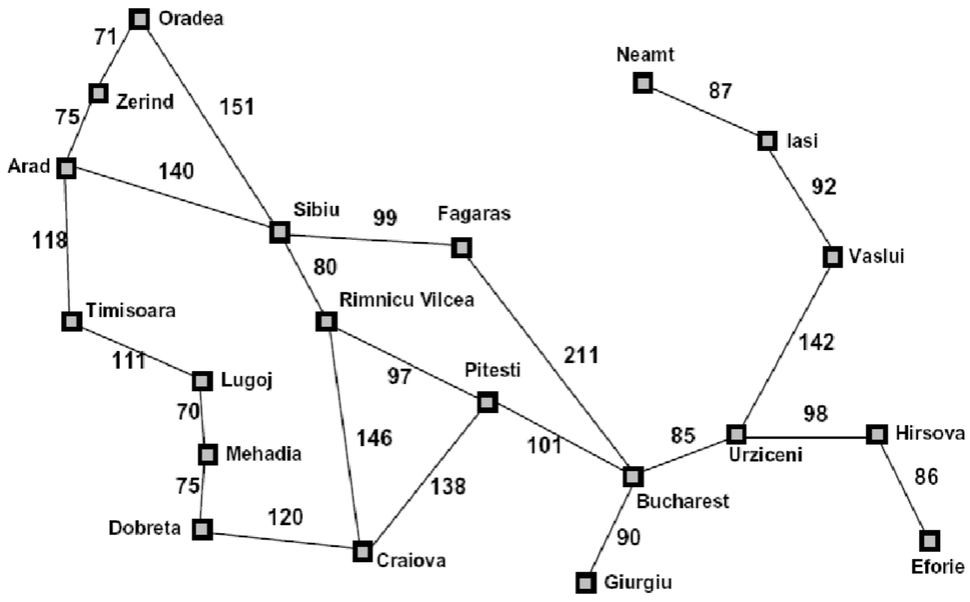
- Heuristic?
- E.g. the index of the largest pancake that is still out of place

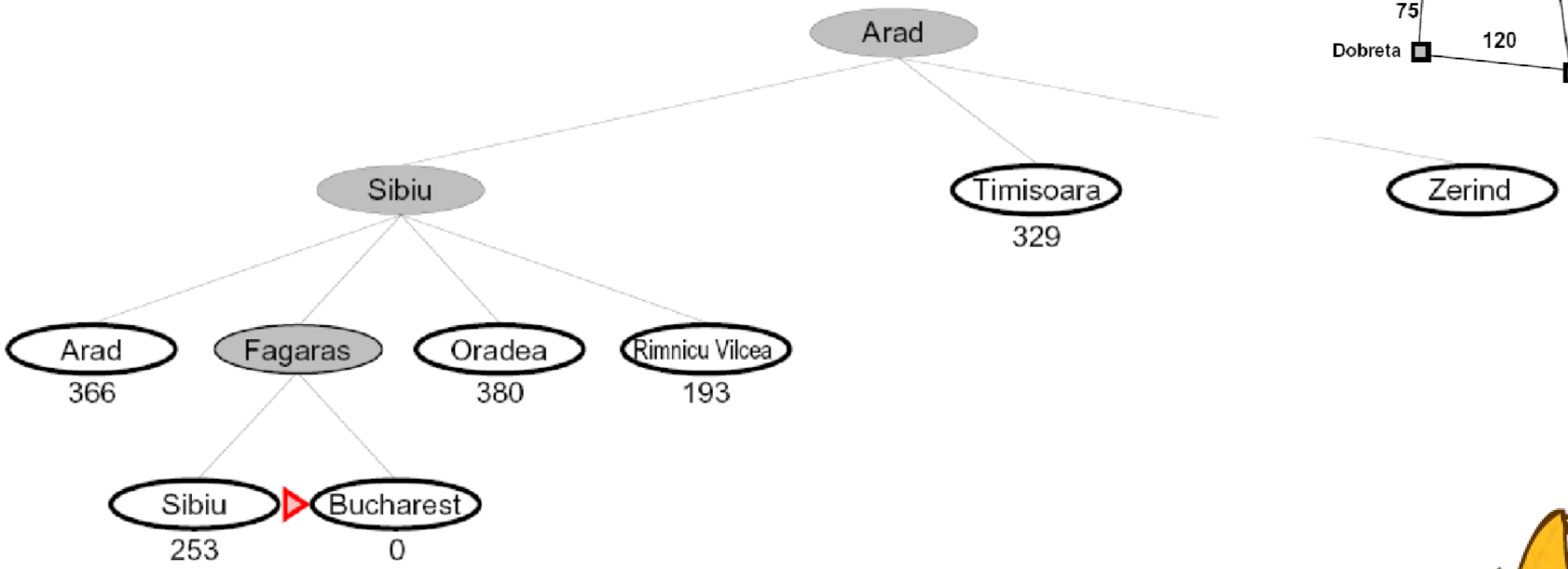


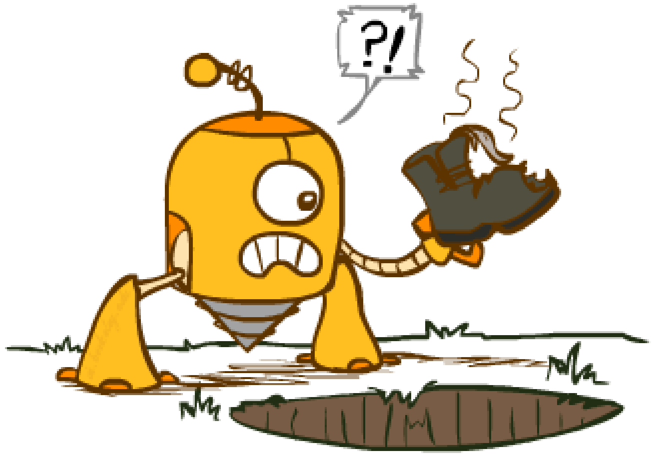
Greedy Search

Greedy Search

- Expand the node that seems closest to the goal



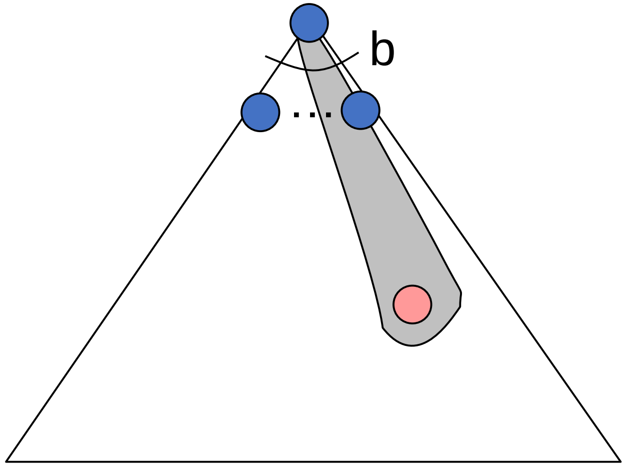




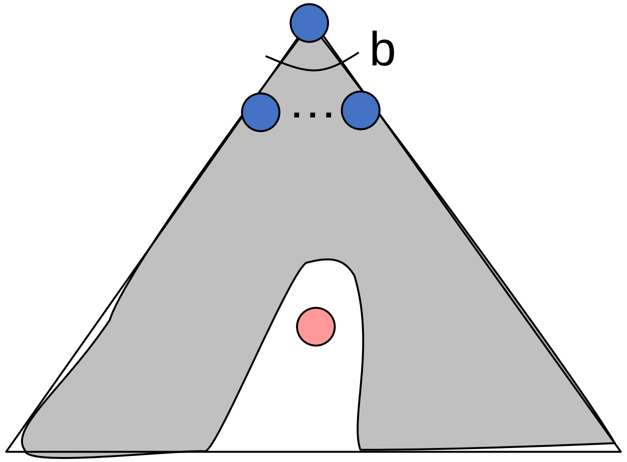
- Is it optimal?
  - No. Resulting path to Bucharest is not the shortest!
  - Why?
  - Heuristics might be wrong

Greedy Search 2

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



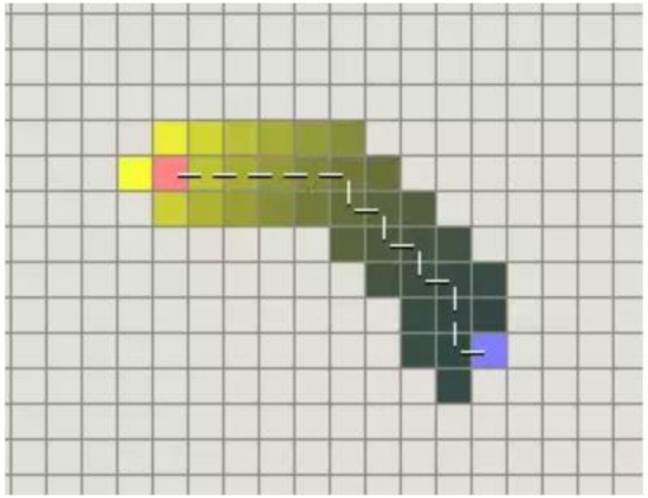
- A common case:
  - Best-first takes you straight to the (wrong) goal
  - (It chooses a node even if it’s at the end of a very long and winding road)
- Worst-case: like a badly-guided DFS
  - (It takes h literally even if it’s completely wrong)

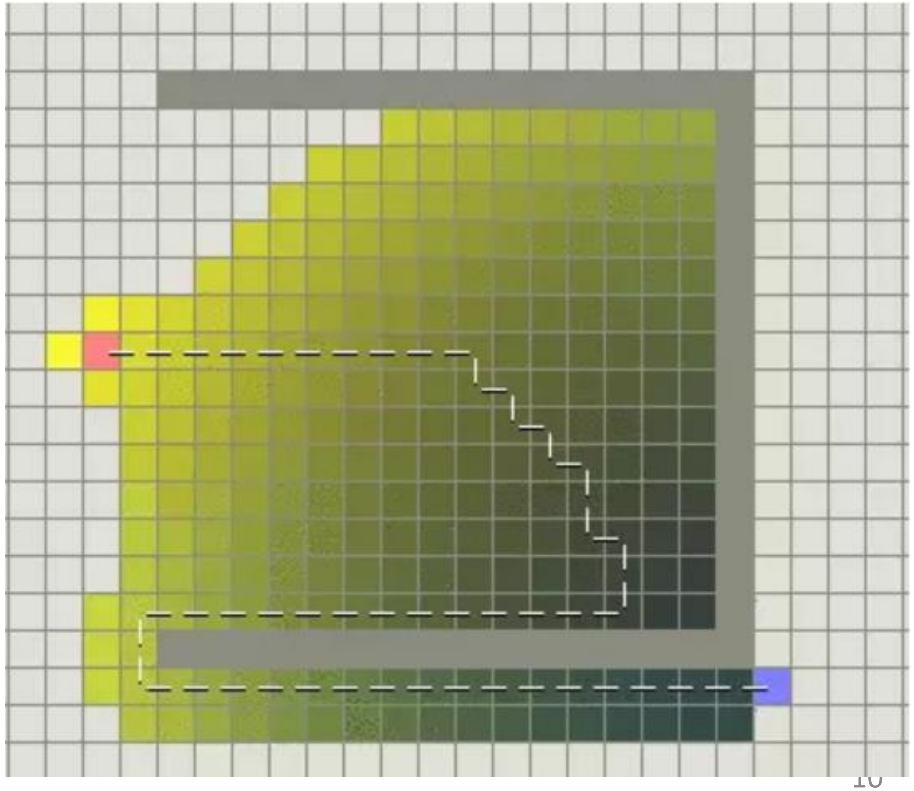


[Demo: contours greedy empty (L3D1)]
[Demo: contours greedy pacman small maze (L3D4)]

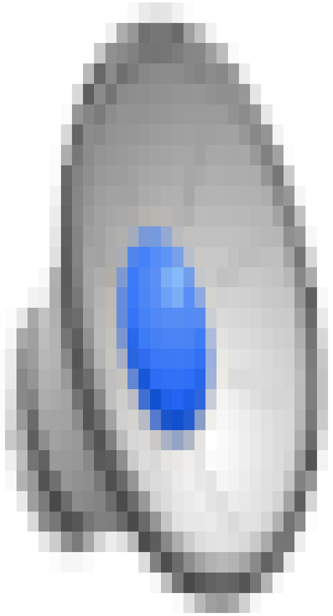
Greedy Search 3

- Each time it visit or expand the point with least $h(n)$ value
  - $h(n)$ is the distance from point n to end point. It works fine when there is no obstacles.
- The cost doubles when there is obstacles

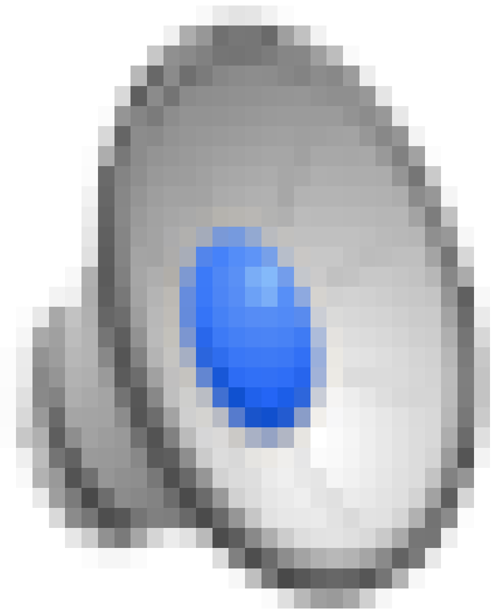




Video of Demo Contours Greedy (Empty)



Video of Demo Contours Greedy (Pacman Small Maze)

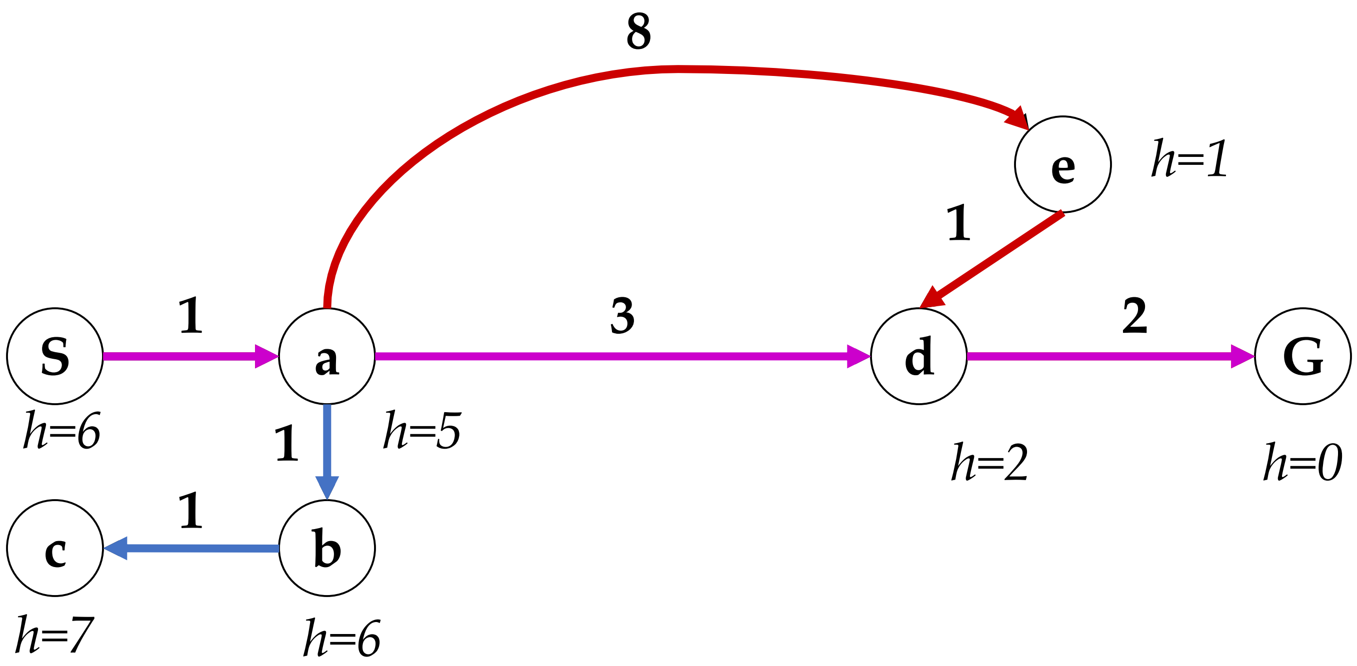


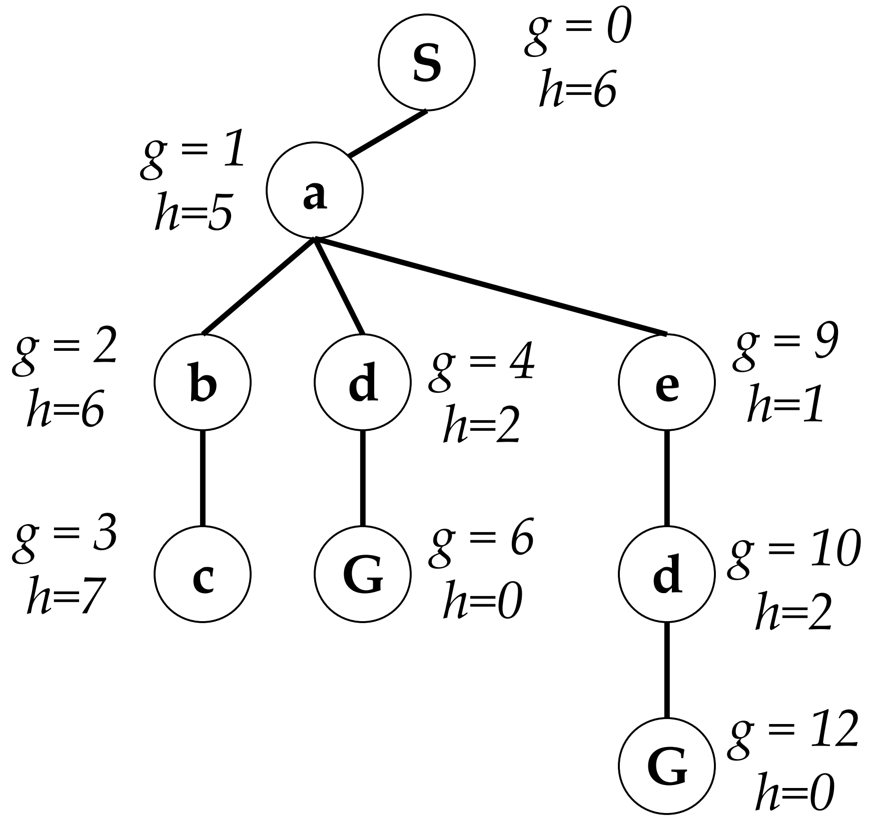
A* Search

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* $h(n)$



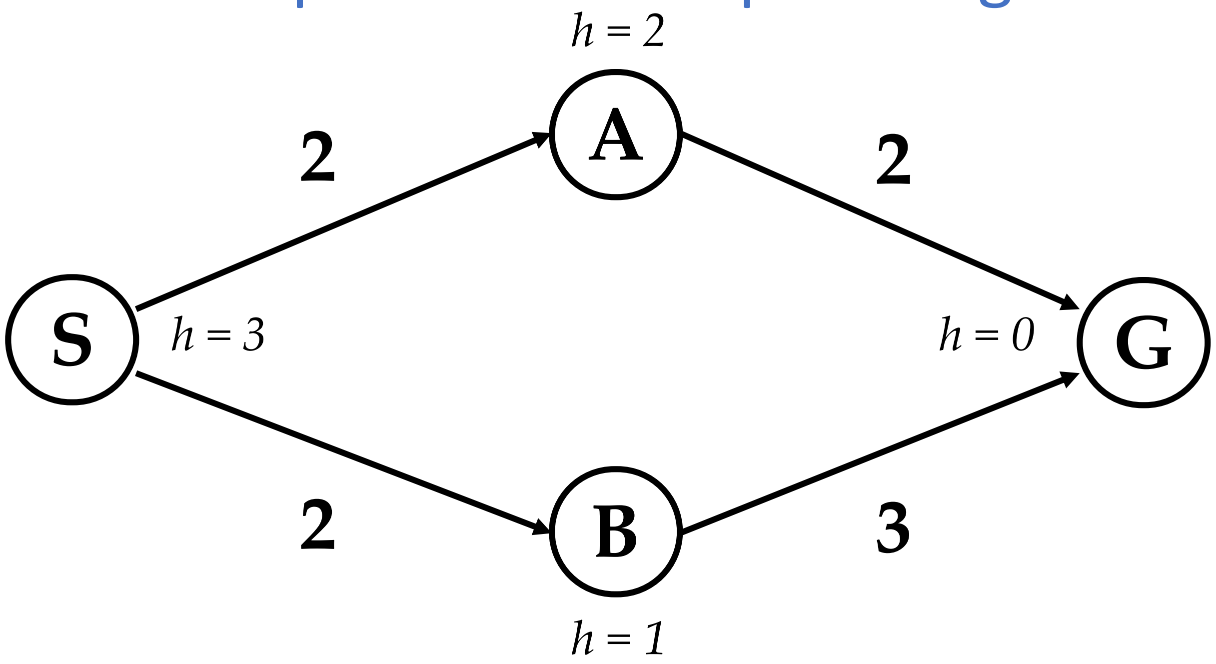


- A* Search orders by the sum: $f(n) = g(n) + h(n)$

Example: Teg Grenager

When should A* terminate?

- Should we stop when we enqueue a goal?

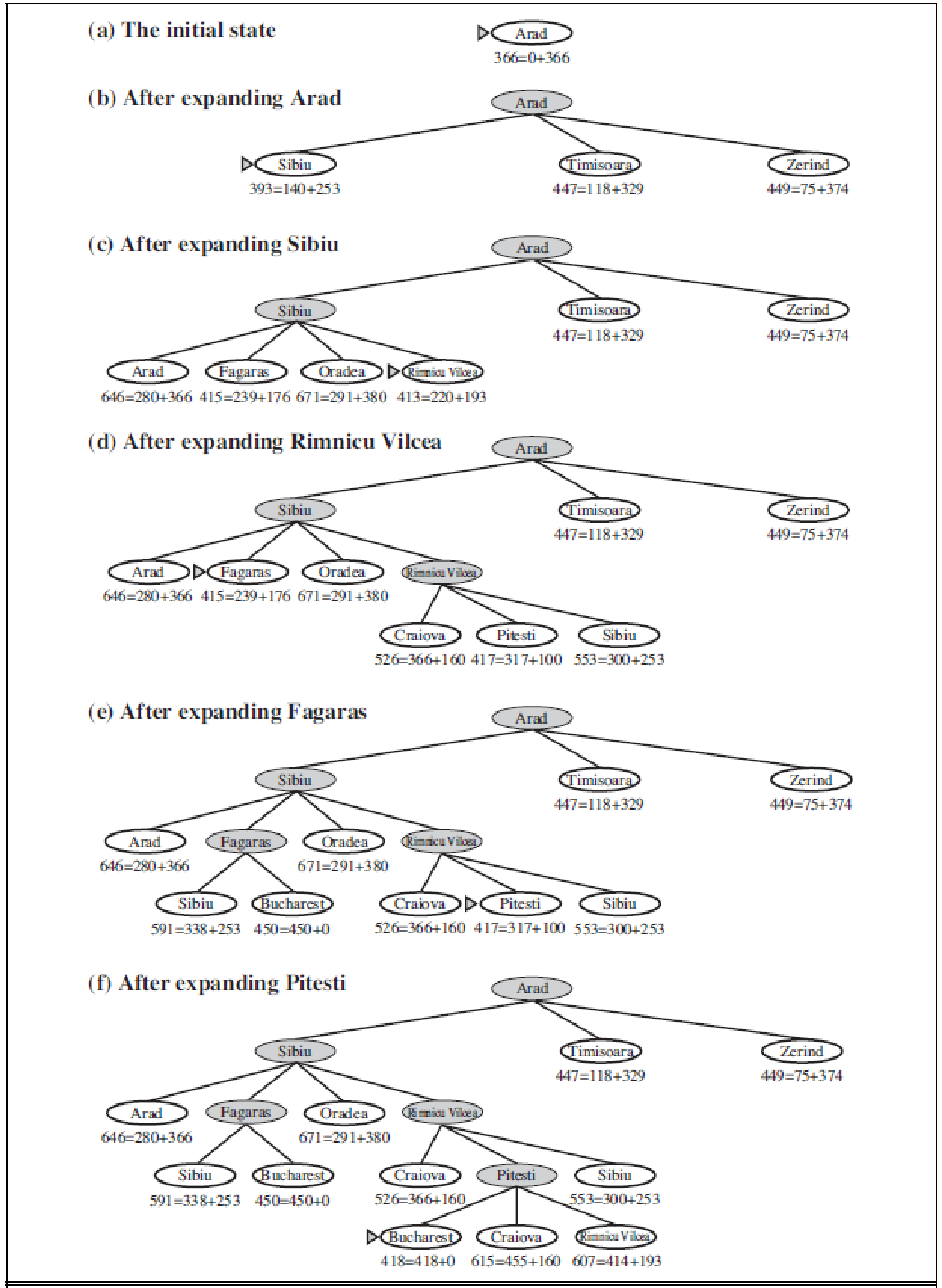


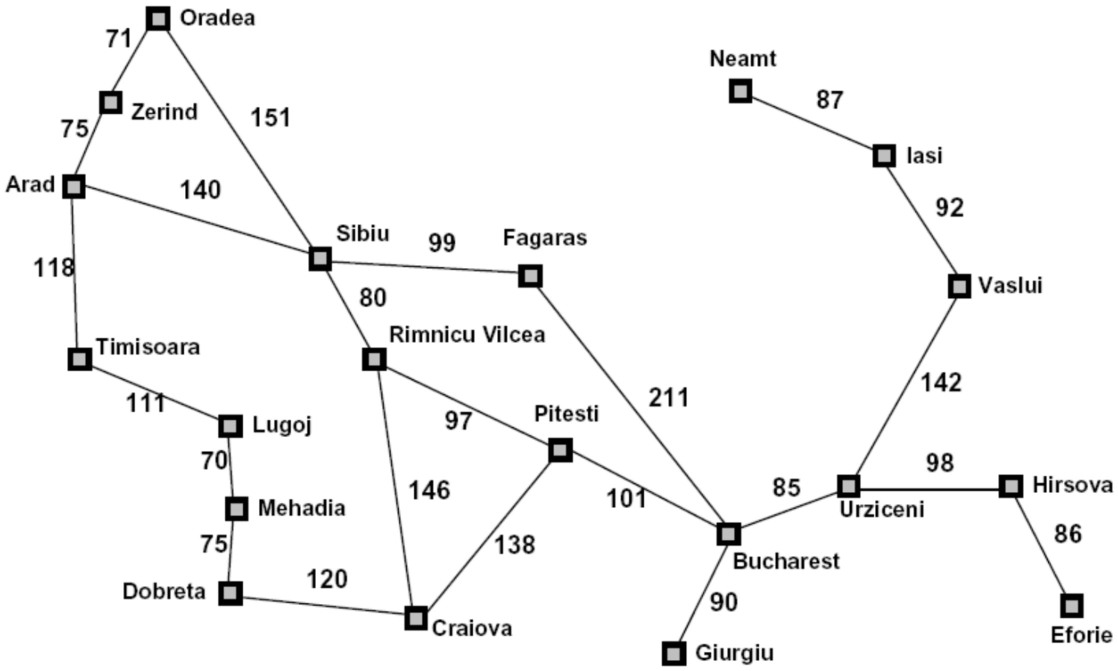
	g	h	+
~~S~~	~~0~~	~~3~~	~~3~~
~~S->A~~	~~2~~	~~2~~	~~4~~
~~S->B~~	~~2~~	~~1~~	~~3~~
S->B->G	5	0	5
S->A->G	4	0	4

- No: only stop when we dequeue a goal

A* Search

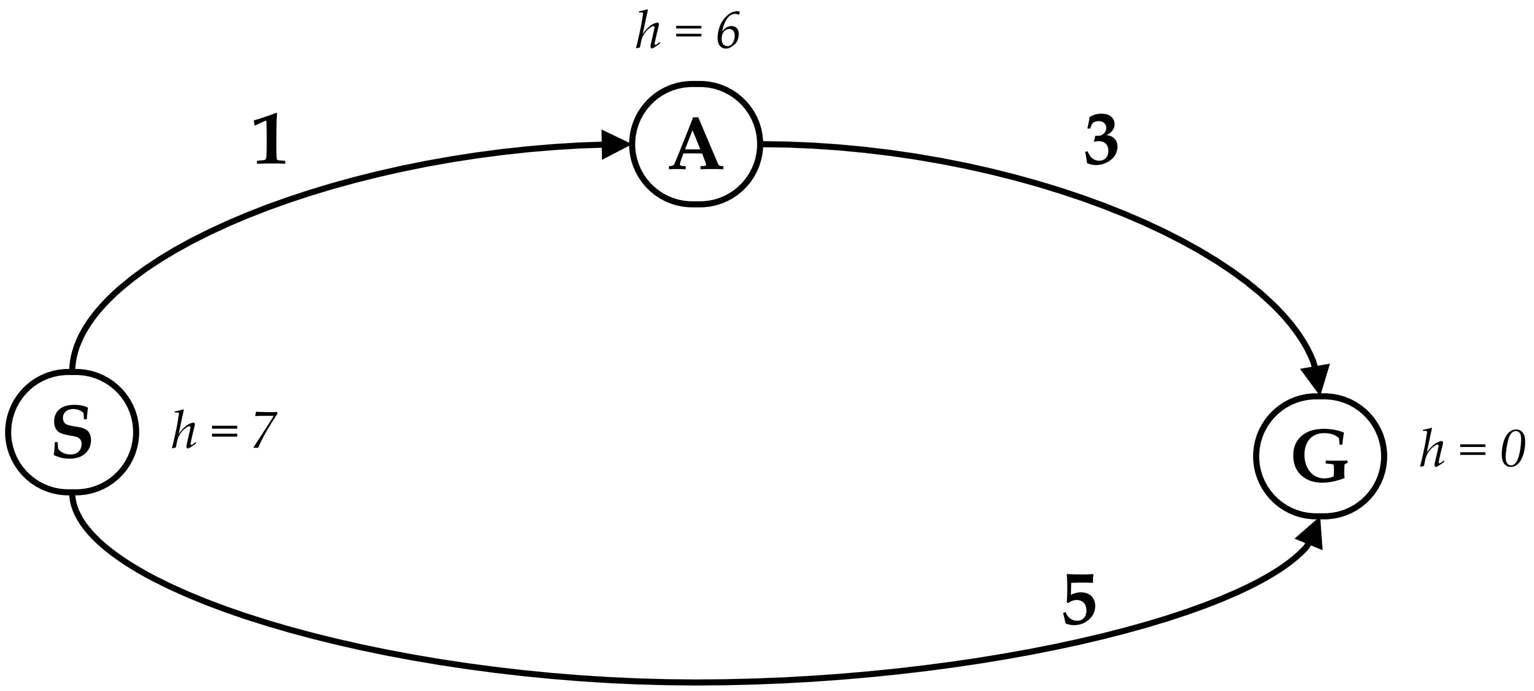
```
function A-STAR-SEARCH(problem) returns a solution, or failure
    initialize the frontier as a priority queue using f(n)=g(n)+h(n) as the priority
    add initial state of problem to frontier with priority f(S)=0+h(S)
    loop do
        if the frontier is empty then
            return failure
        choose a node and remove it from the frontier
        if the node contains a goal state then
            return the corresponding solution
        for each resulting child from node
            add child to the frontier with f(n)=g(n)+h(n)
```





Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Is A* Optimal?



	g	h	+
~~S~~	~~0~~	~~7~~	~~7~~
S->A	1	6	7
$\boxed{\text{S->G}}$	$\boxed{5}$	$\boxed{0}$	$\boxed{5}$

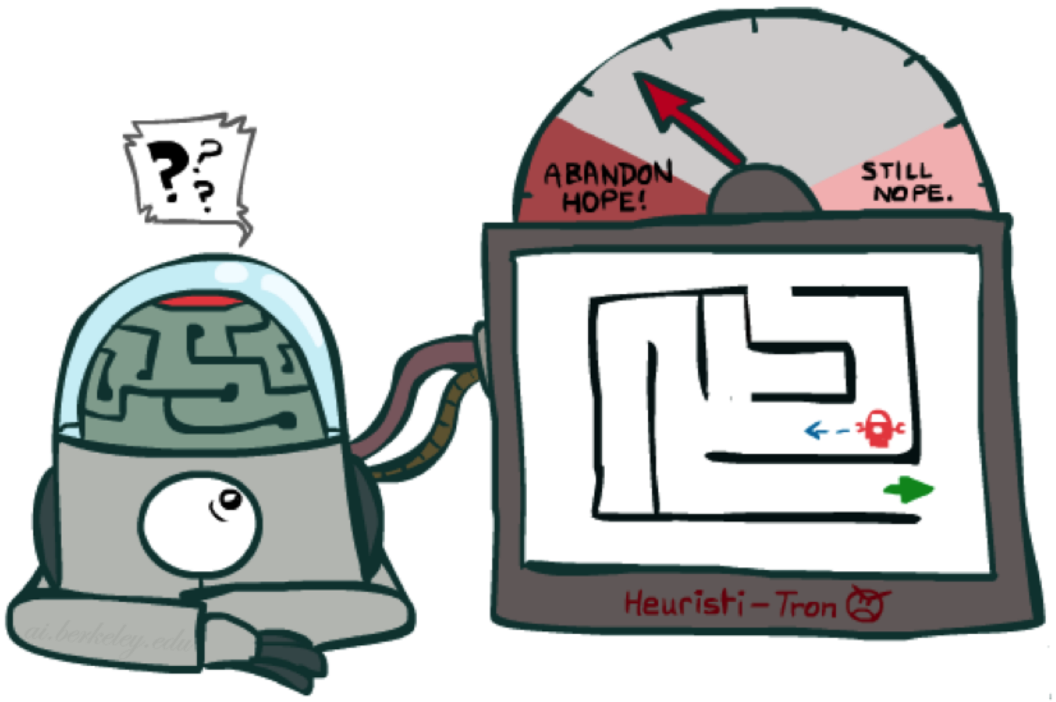
- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!

The Price is Wrong…

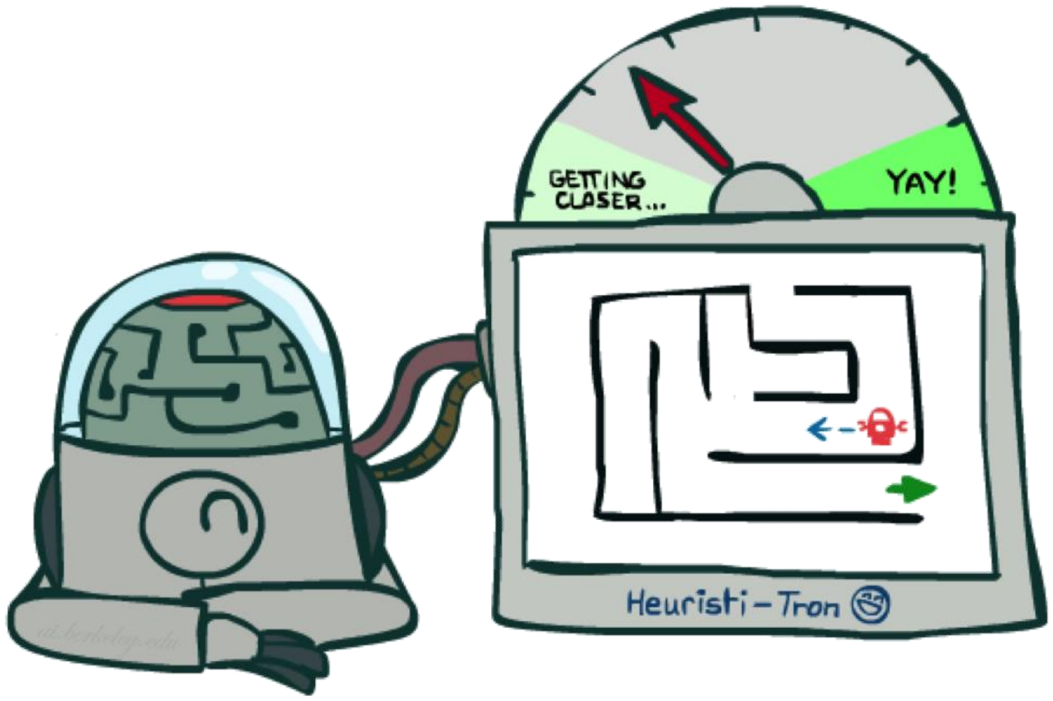
- Closest bid without going over…

https://www.youtube.com/watch?v=9B0ZKRurC5Y

Admissible Heuristics: Ideas



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



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

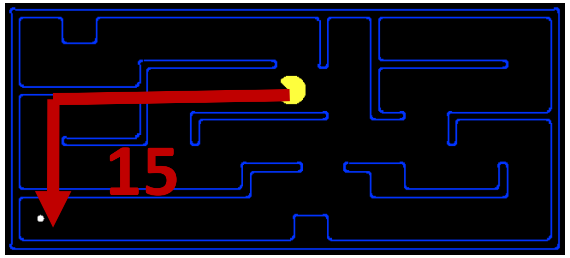
Admissible Heuristics

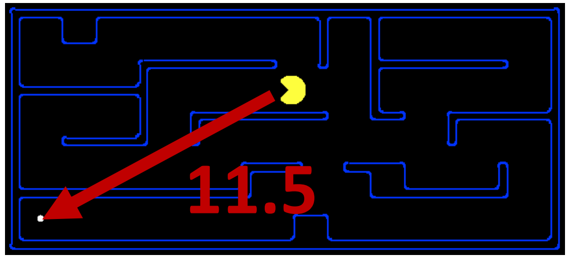
- A heuristic h is *admissible* (optimistic) if

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

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

- Examples:





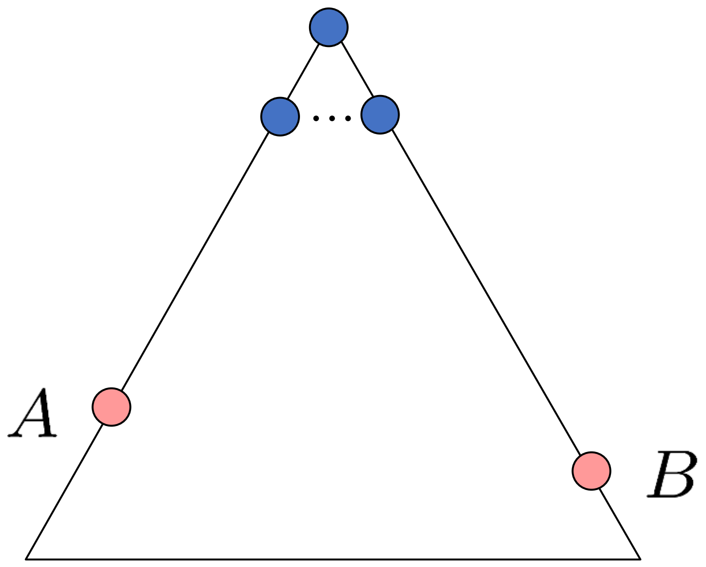
0.0

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

Optimality of A* Tree Search

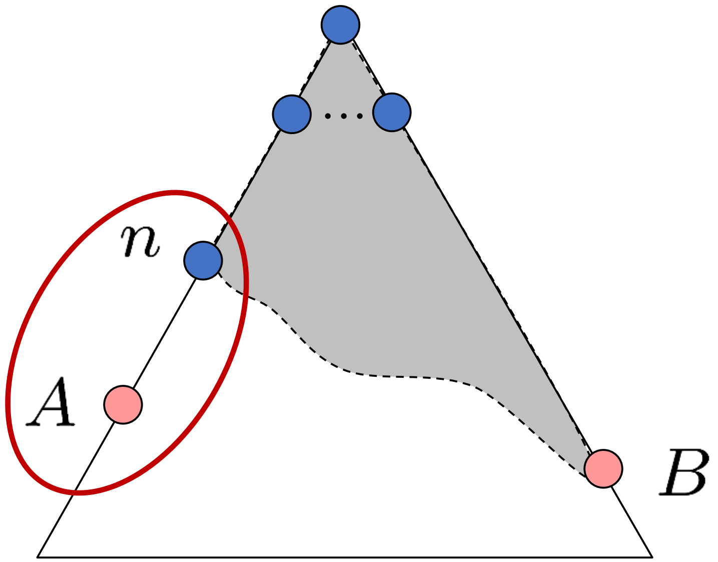
- Assume:
  - A is an optimal goal node
  - B is a suboptimal goal node
  - h is admissible

- Claim:
  - A will exit the fringe before B



Optimality of A* Tree Search: Blocking

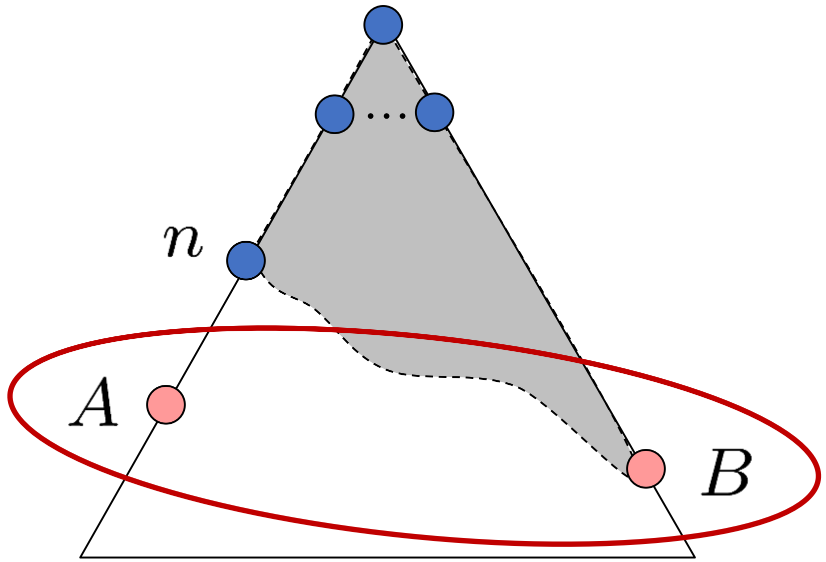
- Proof:
  - Imagine B is on the fringe
  - Some ancestor n of A is on the fringe, too (maybe A!)
  - Claim: n will be expanded before B
    1. f(n) is less or equal to f(A)



$f(n) = g(n) + h(n)$	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 2

- 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)

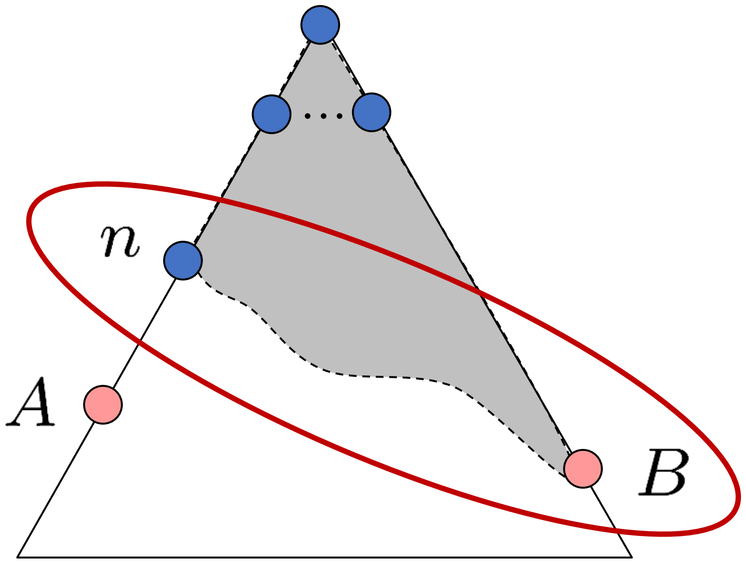


$$g(A) < g(B)$$ B is suboptimal

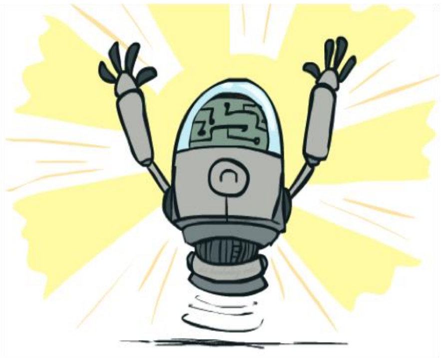
$$f(A) < f(B)$$ h = 0 at a goal

Optimality of A* Tree Search: Blocking 3

- 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

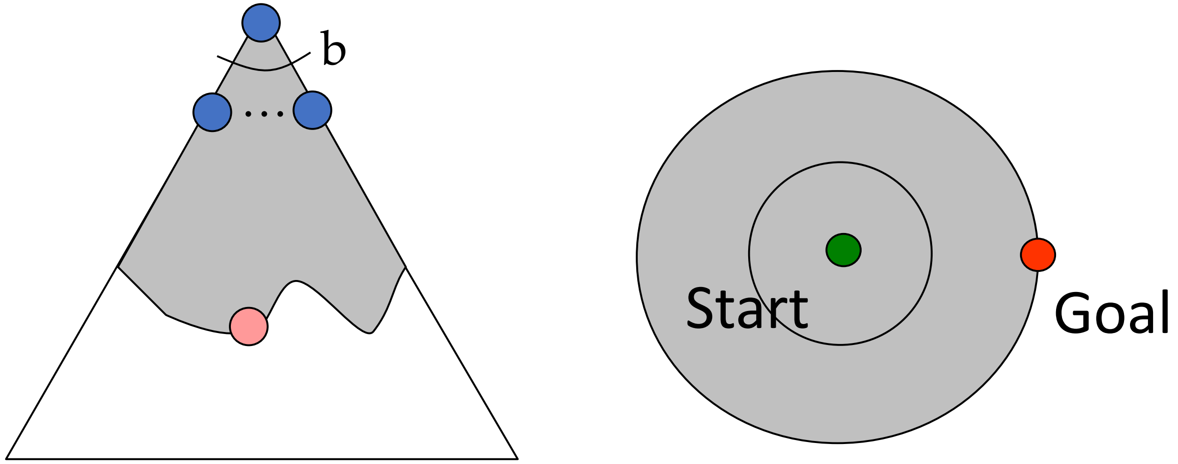


$$f(n) \leq f(A) < f(B)$$

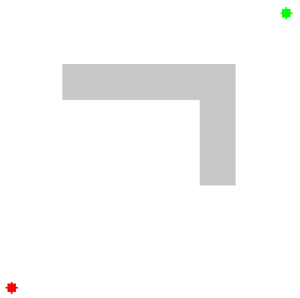


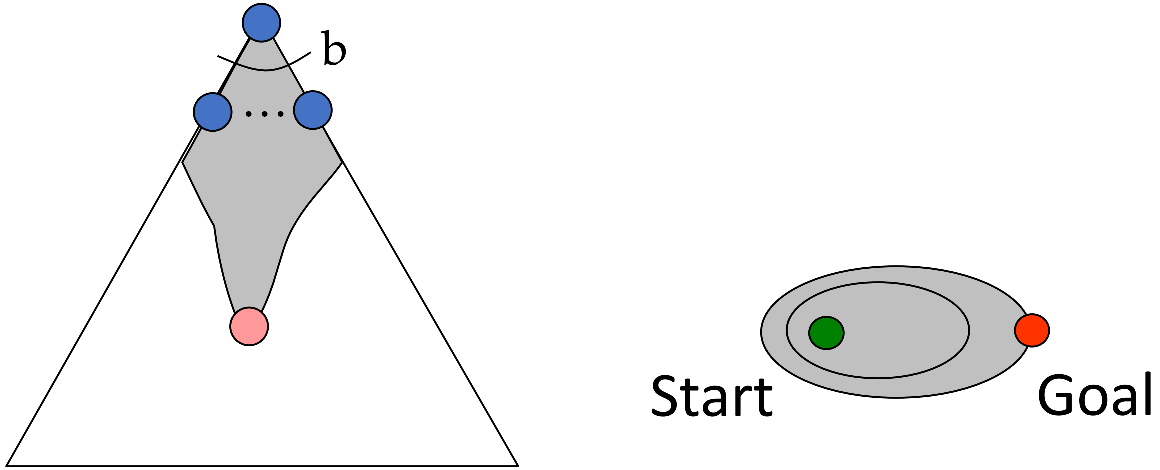
UCS vs A*

- Uniform-cost expands equally in all "directions"



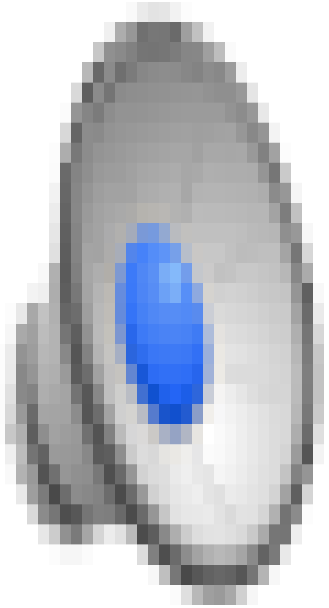
- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



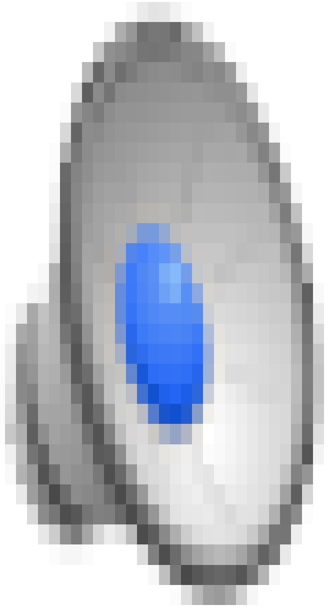


[Demo: contours UCS / greedy / A* empty (L3D1)]
[Demo: contours A* pacman small maze (L3D5)]

Video of Demo Contours (Empty) -- UCS

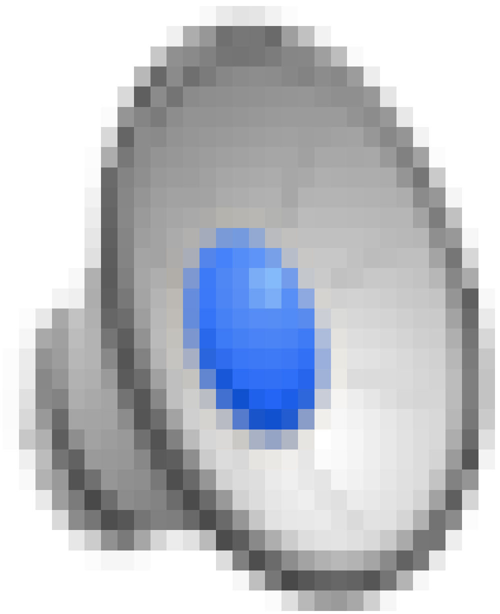


Video of Demo Contours (Empty) -- Greedy

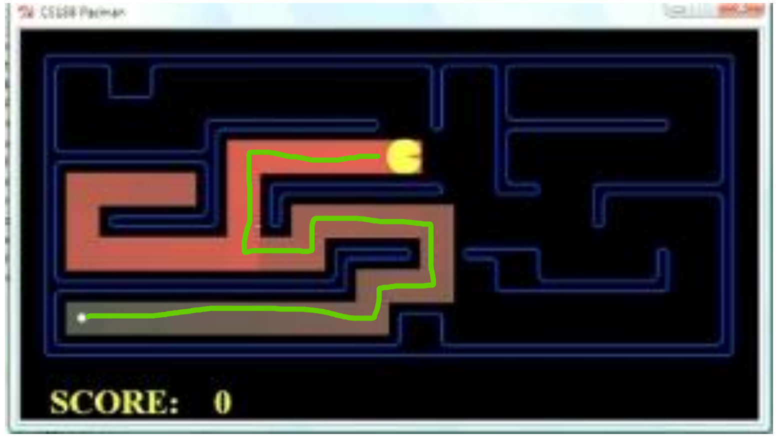


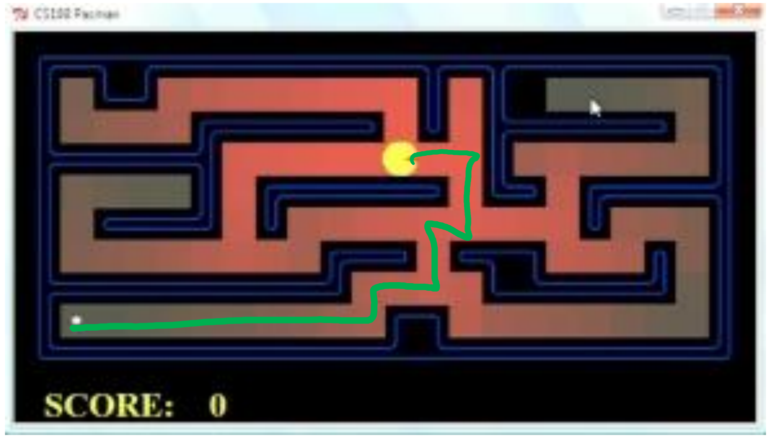
Video of Demo Contours (Empty) – A*

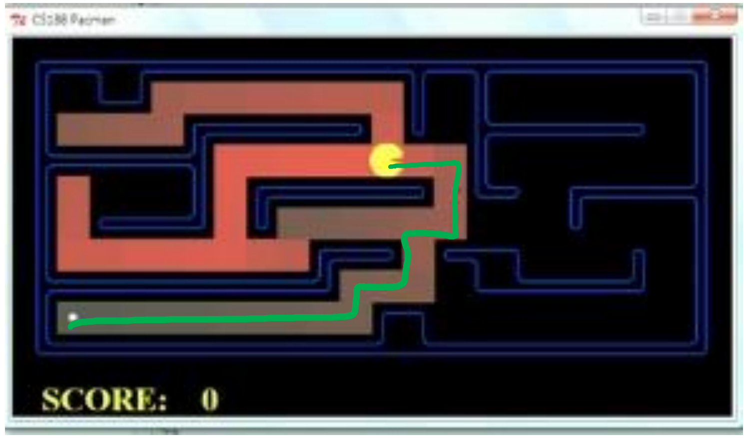
Video of Demo Contours (Pacman Small Maze) – A*



Comparison







Greedy

Uniform Cost

A*

A* Applications

A* Applications 2

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

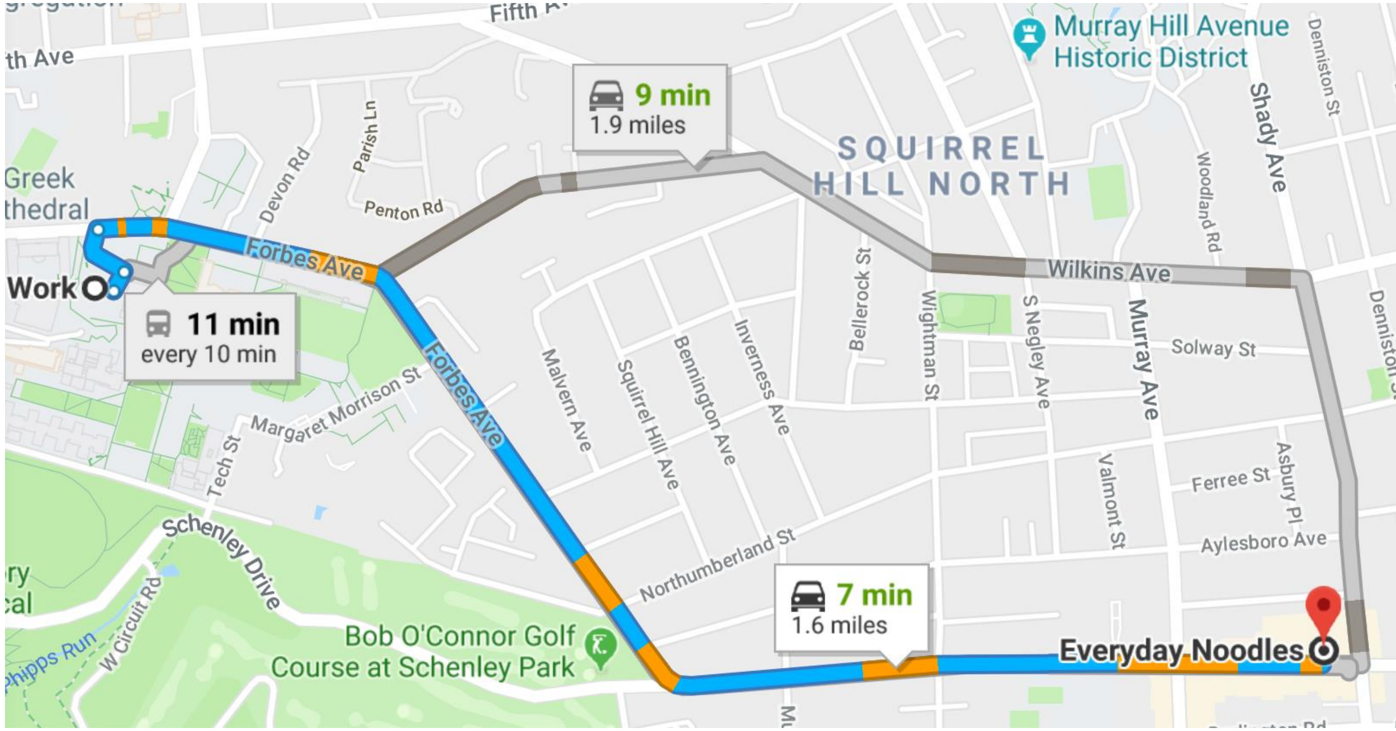
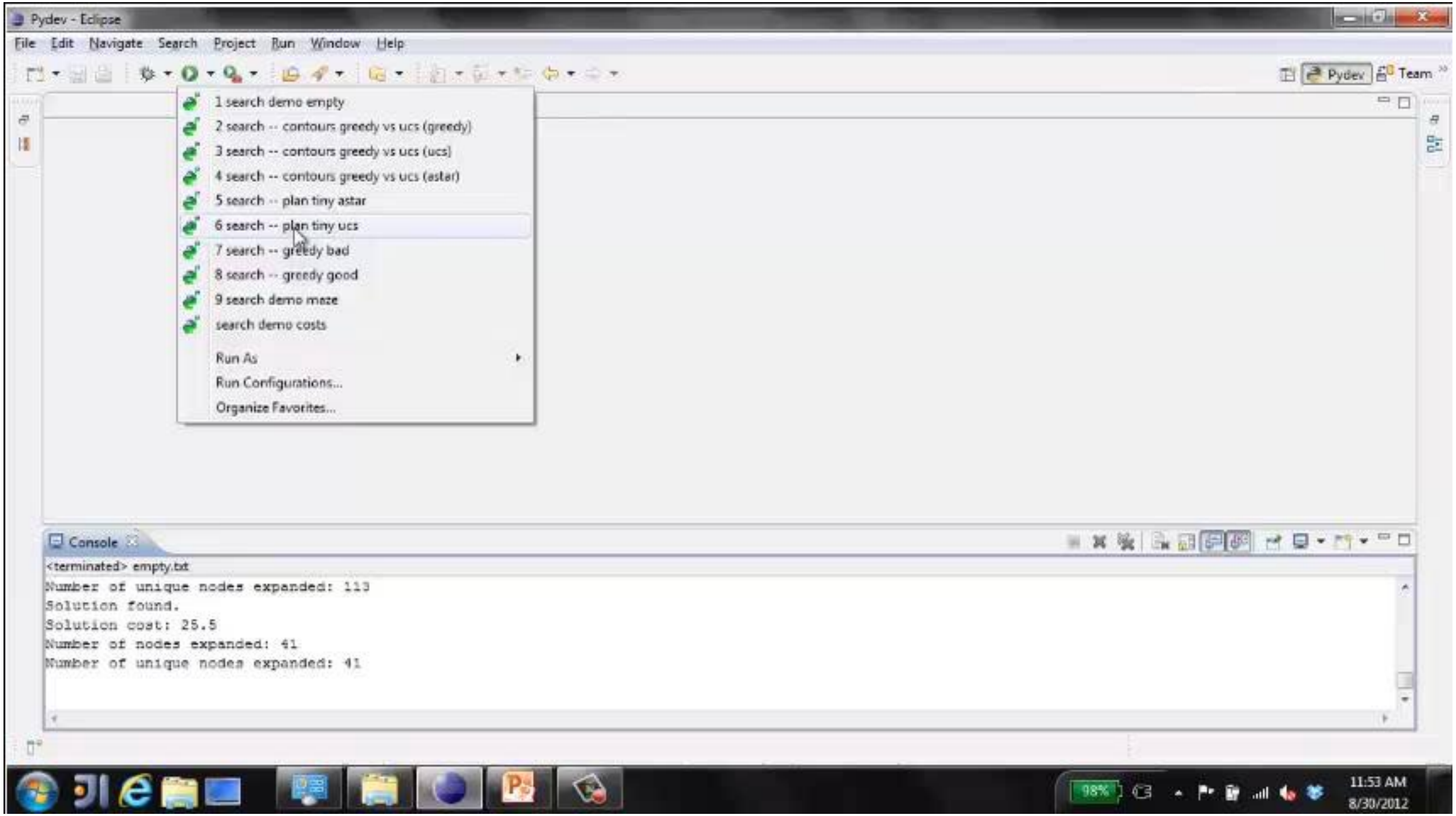


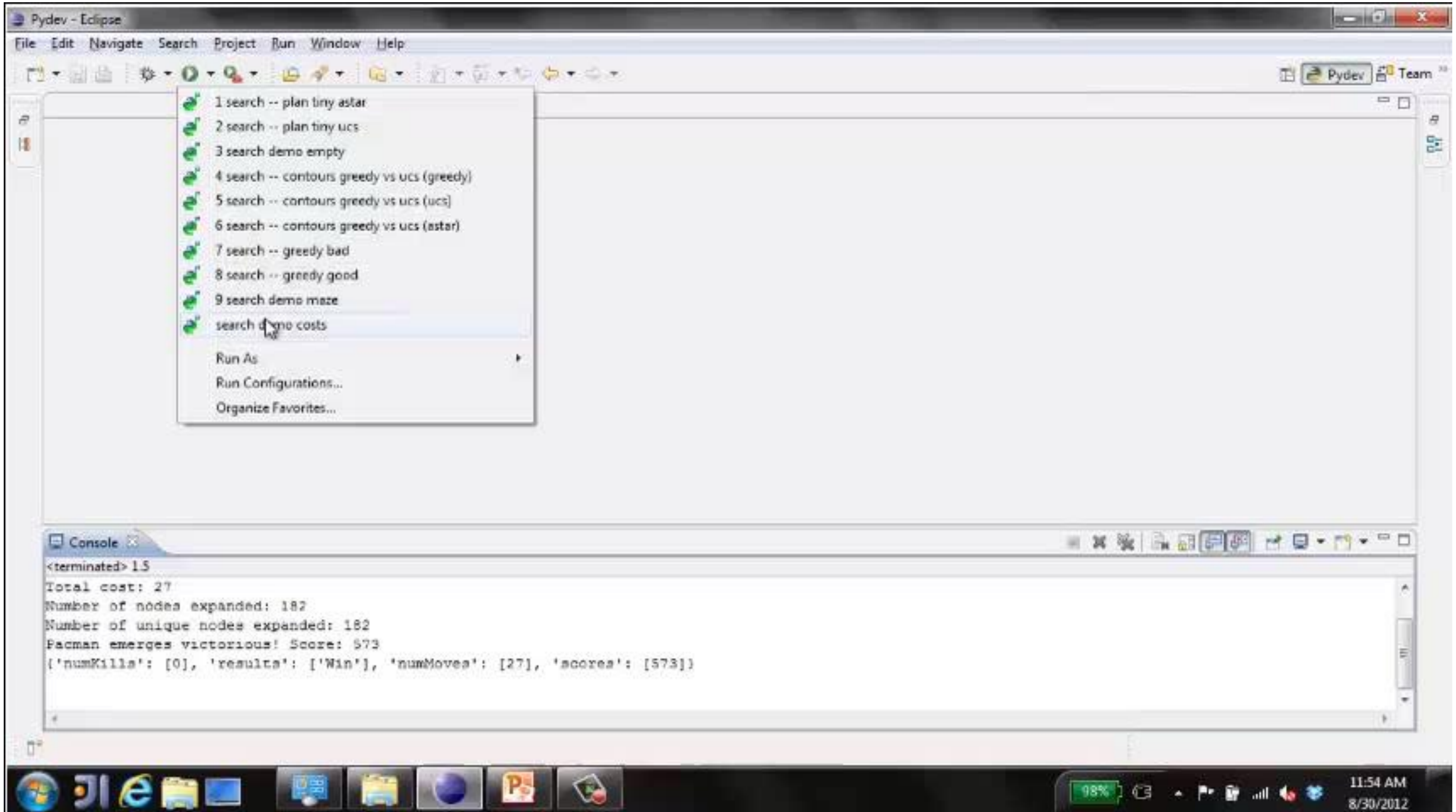
Image: maps.google.com

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

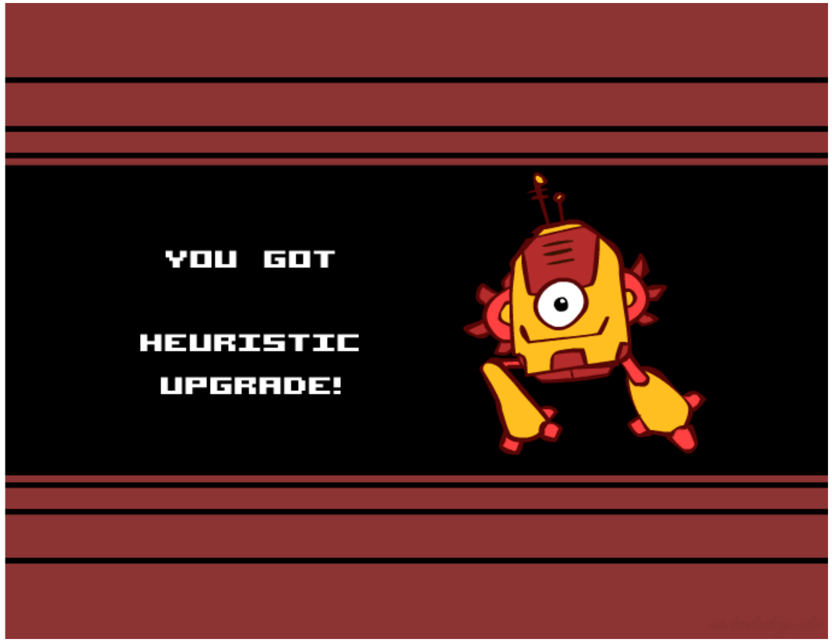
Video of Demo Pacman (Tiny Maze) – UCS / A*



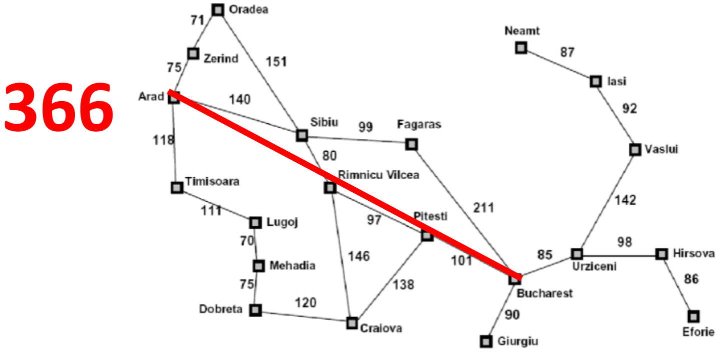
Video of Demo Empty Water Shallow/Deep – Guess Algorithm

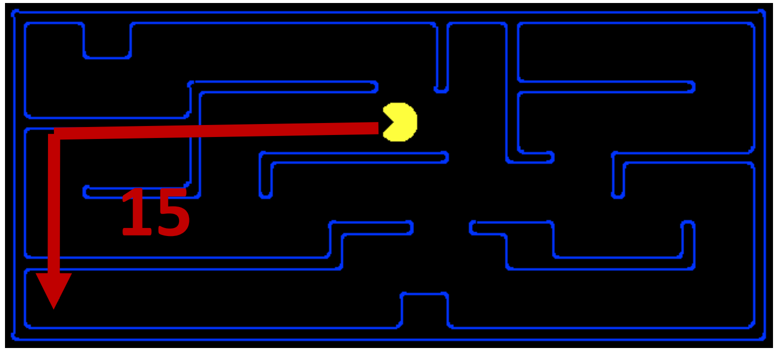


Creating Heuristics



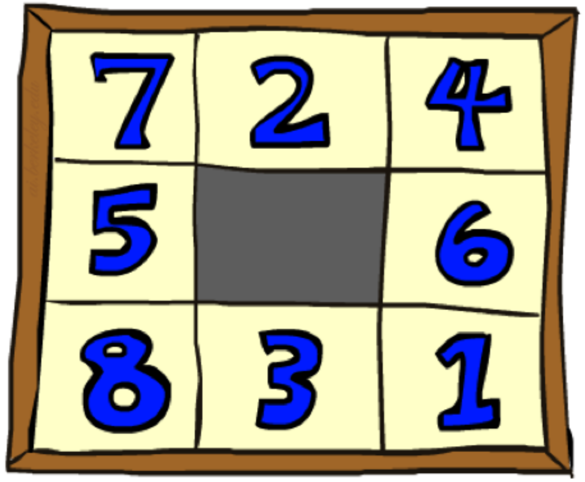
- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
- Often, admissible heuristics are solutions to relaxed problems, where new actions are available



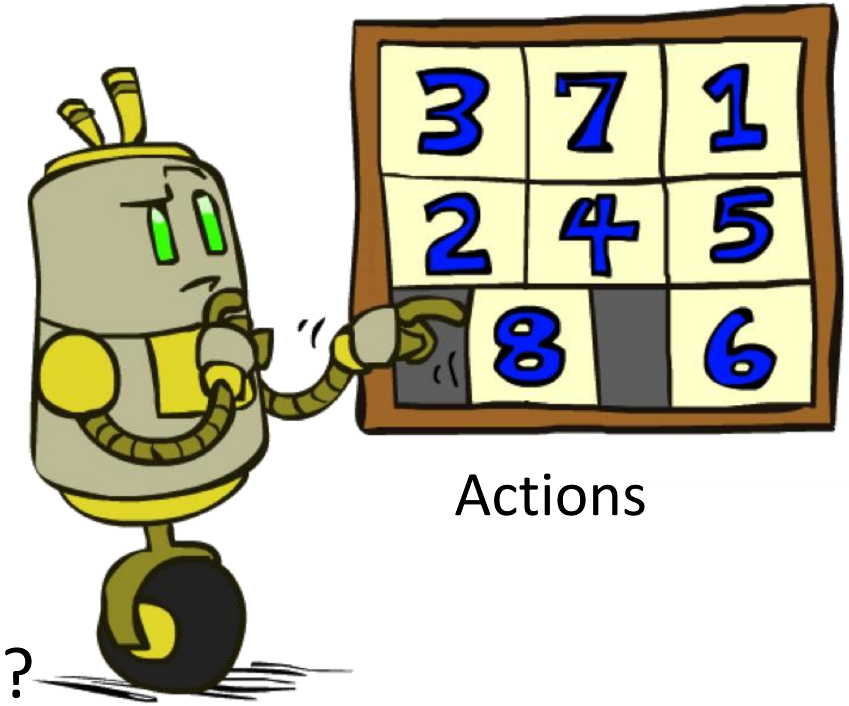


- Inadmissible heuristics are often useful too

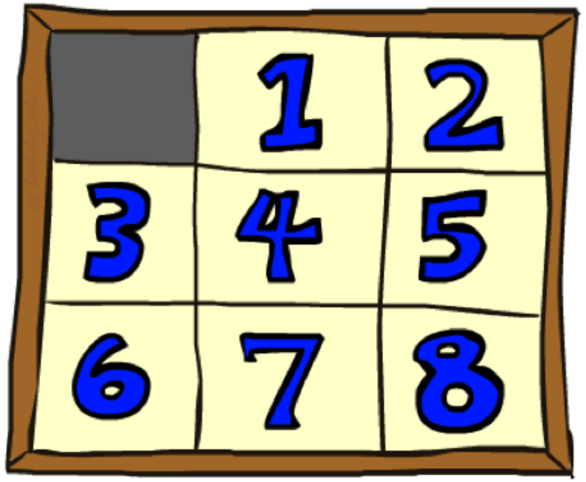
Example: 8 Puzzle



Start State



Actions



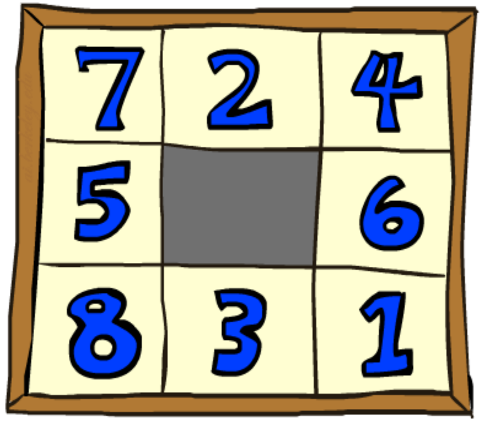
Goal State

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

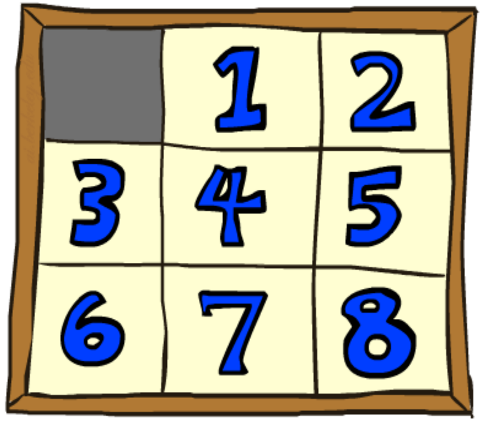
Admissible heuristics?

Example: 8 Puzzle 2

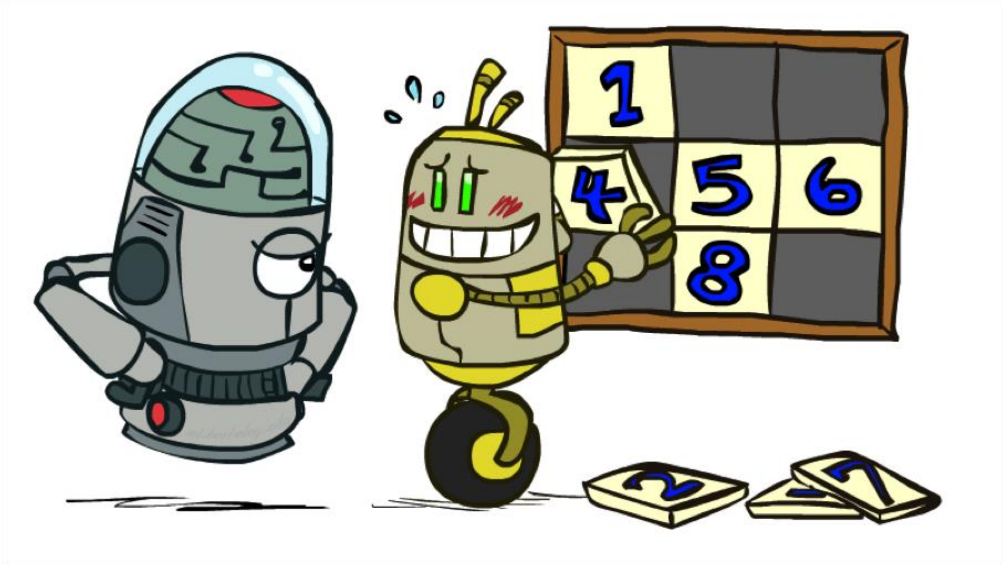
- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8
- This is a relaxed-problem heuristic



Start State



Goal State

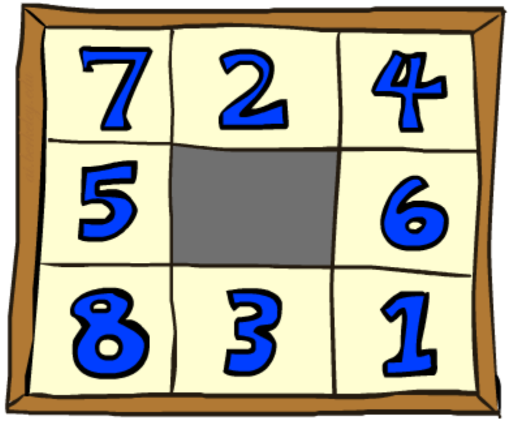


	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

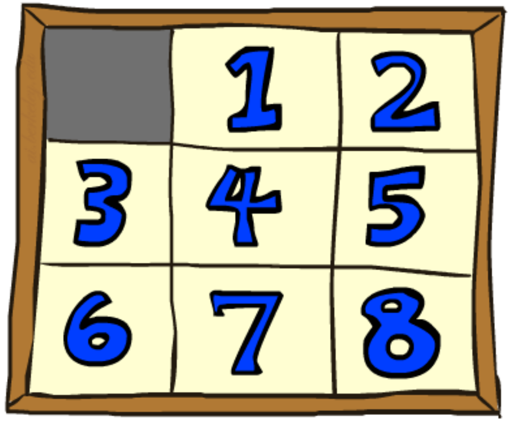
Statistics from Andrew Moore

Example: 8 Puzzle 3

- 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

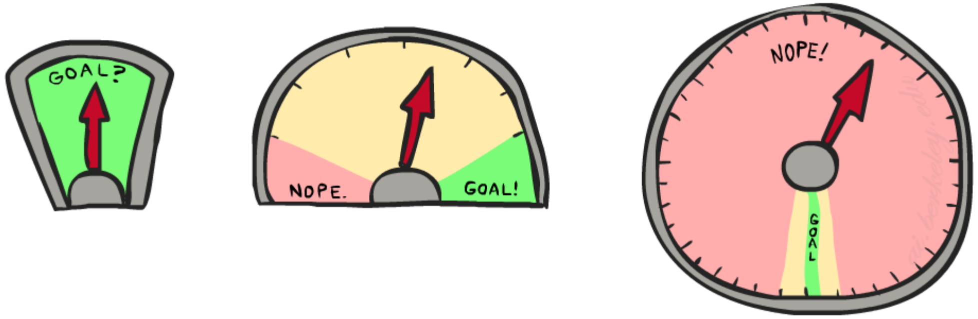
- Why is it admissible?

- h(start) = 3 + 1 + 2 + … = 18

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

Example: 8 Puzzle 4

- 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

Combining Heuristics, Dominance

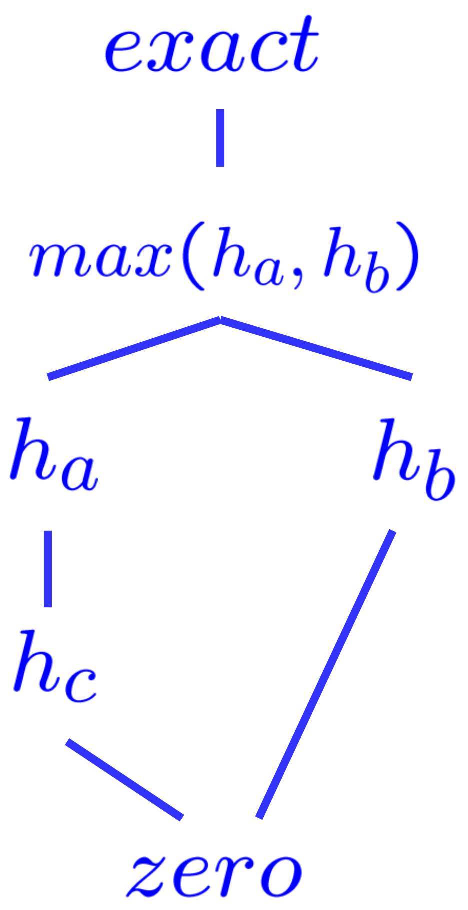
- Dominance: $h_a \geq h_c$ if

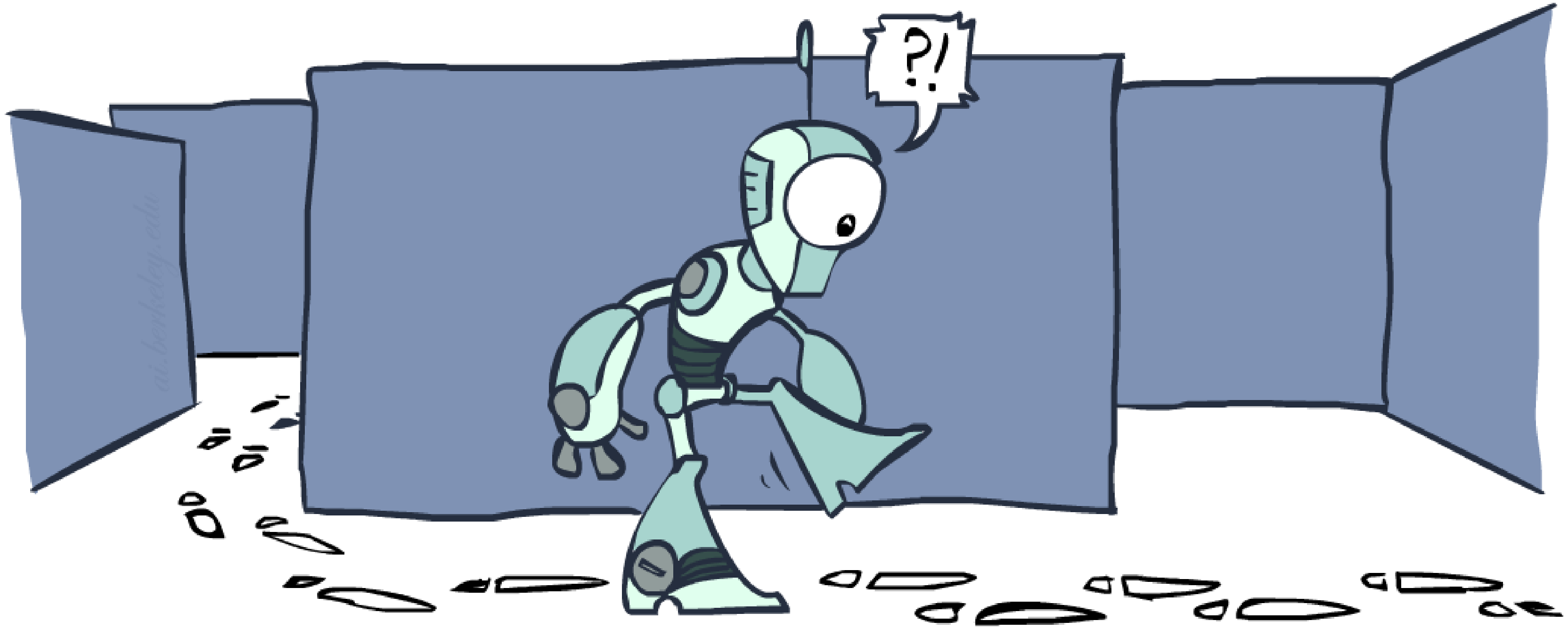
$$\forall n : h_a(n) \geq h_c(n)$$

  - Roughly speaking, larger is better as long as both are admissible
- 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, but usually too expensive



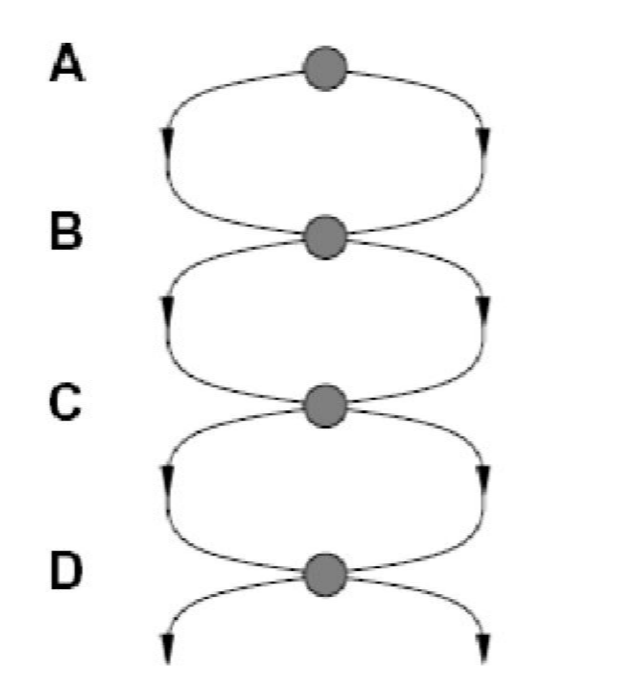


Graph Search

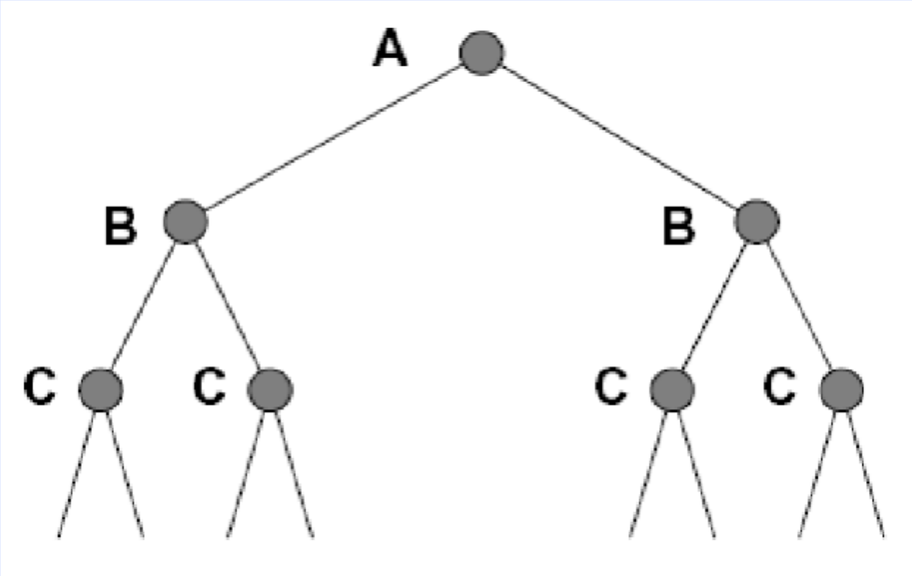
Tree Search: Extra Work!

- Failure to detect repeated states can cause exponentially more work

State Graph

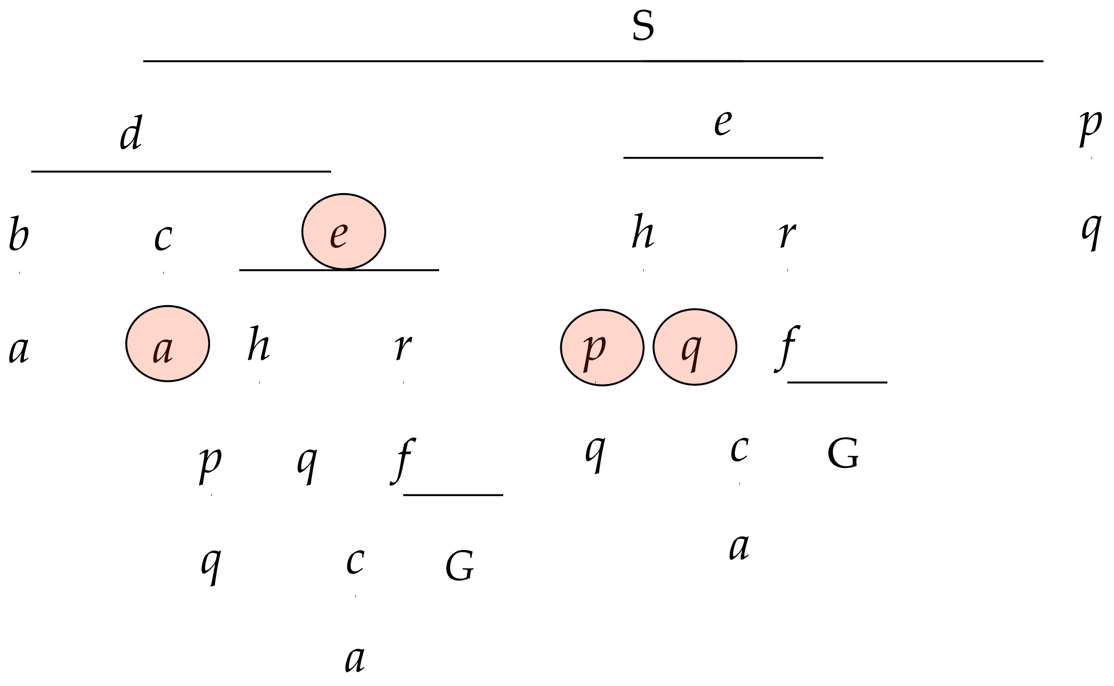


Search Tree



Graph Search

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

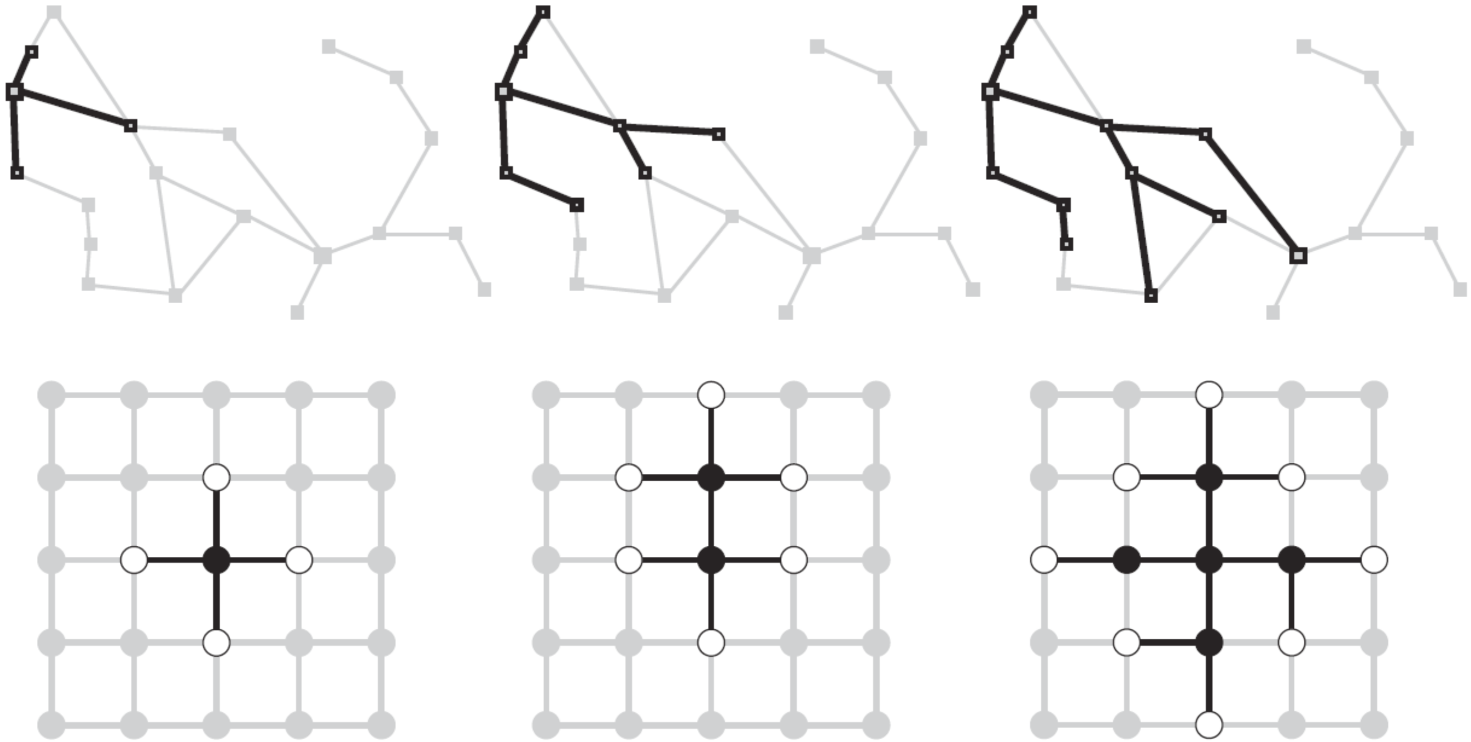


Graph Search 2

- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states (“closed set”, “explored 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?

Graph Search 3

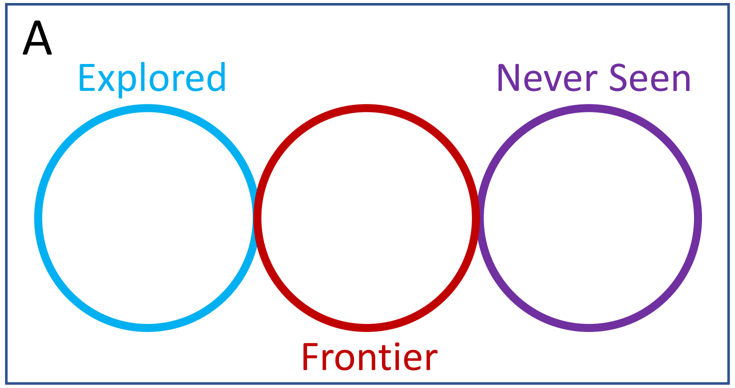
- This graph search algorithm overlays a tree on a graph
- The frontier states separate the explored states from never seen states

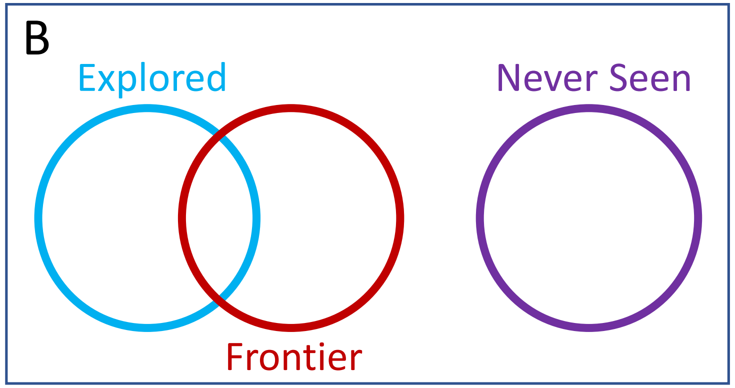


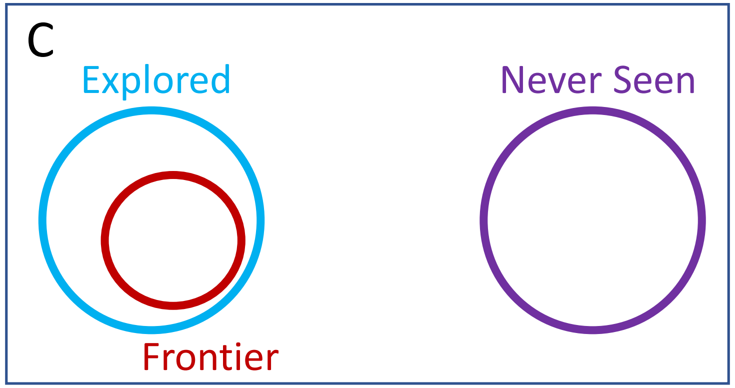
Images: AIMA, Figure 3.8, 3.9

Quiz

- What is the relationship between these sets of states after each loop iteration in GRAPH_SEARCH?
- (Loop invariants!!!)







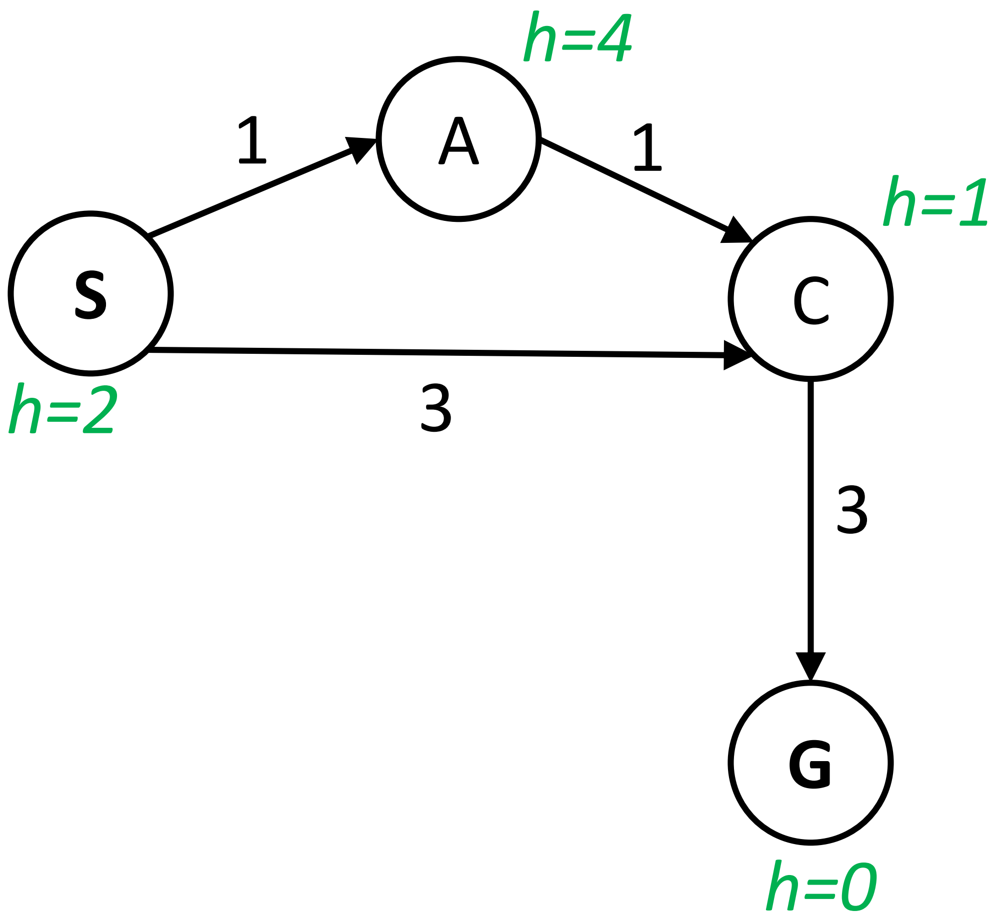
```
function GRAPH_SEARCH(problem) returns a solution, or failure
    initialize the explored set to be empty
    initialize the frontier as a specific work list (stack, queue, priority queue)
    add initial state of problem to frontier
    loop do
        if the frontier is empty then
            return failure
        choose a node and remove it from the frontier
        if the node contains a goal state then
            return the corresponding solution
        add the node state to the explored set
        for each resulting child from node
            if the child state is not already in the frontier or explored set then
                add child to the frontier
```

```
function UNIFORM-COST-GRAPH-SEARCH(problem) returns a solution, or failure
    initialize the explored set to be empty
    initialize the frontier as a priority queue using node's path_cost as the priority
    add initial state of problem to frontier with path_cost = 0
    loop do
        if the frontier is empty then
            return failure
        choose a node and remove it from the frontier
        if the node contains a goal state then
            return the corresponding solution
        add the node state to the explored set
        for each resulting child from node
            if the child state is not already in the frontier or explored set then
                add child to the frontier
            else if the child is already in the frontier with higher path_cost then
                replace that frontier node with child
```

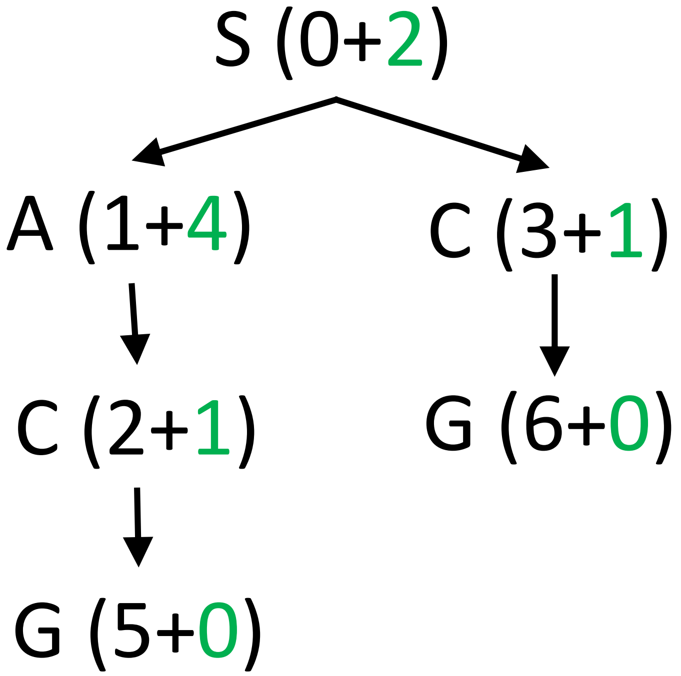
```
function A-STAR-GRAPH-SEARCH(problem) returns a solution, or failure
    initialize the explored set to be empty
    initialize the frontier as a priority queue using f(n) = g(n) + h(n) as the priority
    add initial state of problem to frontier with priority f(S) = 0 + h(S)
    loop do
        if the frontier is empty then
            return failure
        choose a node and remove it from the frontier
        if the node contains a goal state then
            return the corresponding solution
        add the node state to the explored set
        for each resulting child from node
            if the child state is not already in the frontier or explored set then
                add child to the frontier
            else if the child is already in the frontier with higher f(n) then
                replace that frontier node with child
```

A* Tree Search

State space graph

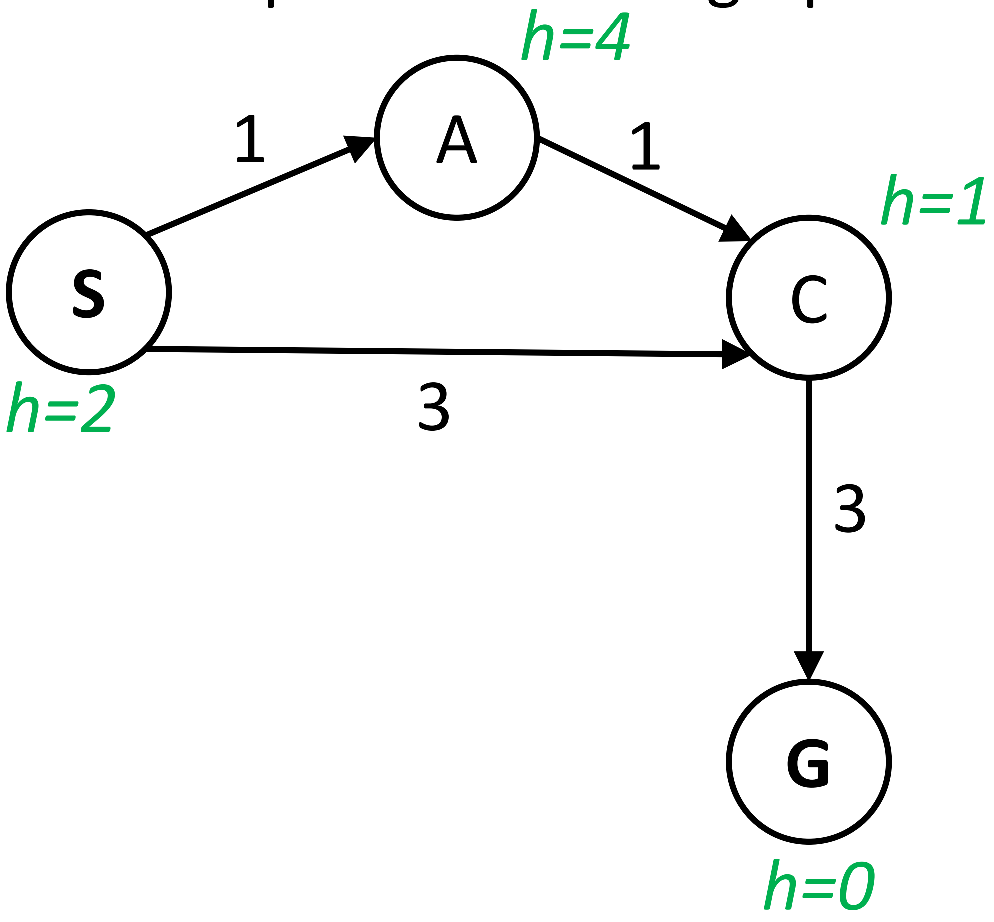


Search tree



Quiz: A* Graph Search

- What paths does A* graph search consider during its search?



A) ~~S~~, ~~S-A~~, ~~S-C~~, S-C-G

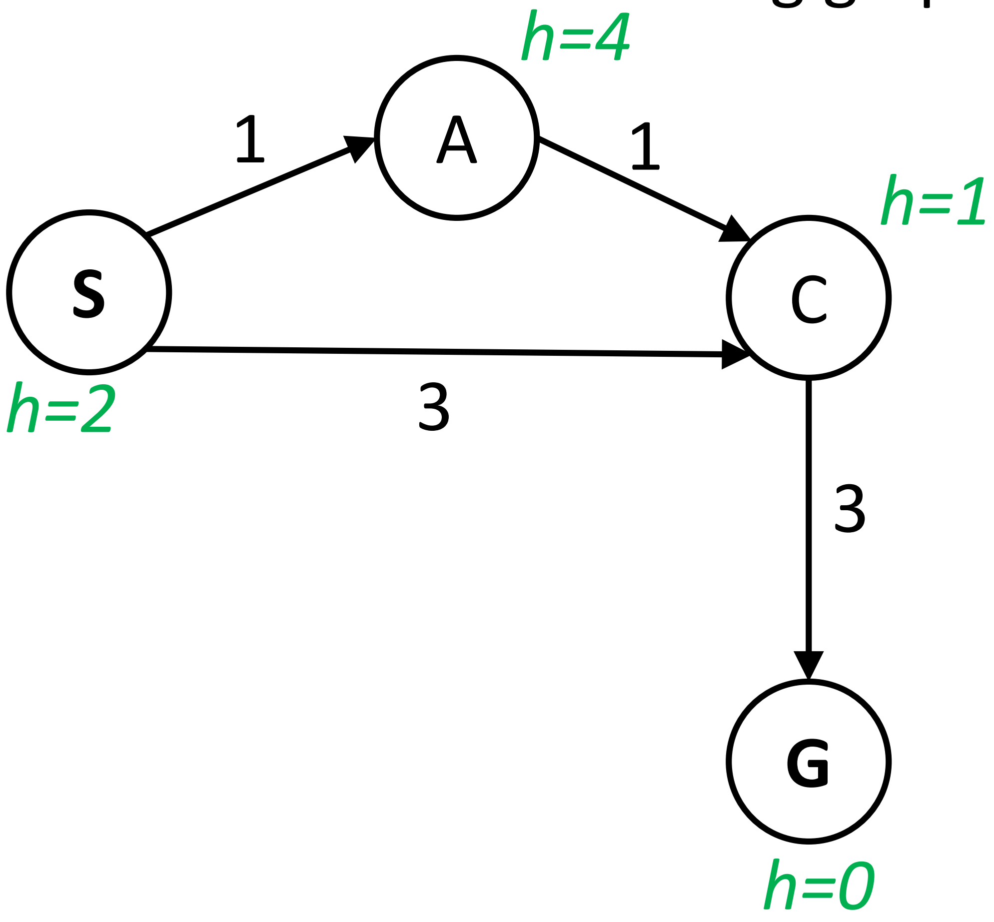
B) ~~S~~, ~~S-A~~, S-C, ~~S-A-C~~, S-C-G

C) ~~S~~, ~~S-A~~, ~~S-A-C~~, S-A-C-G

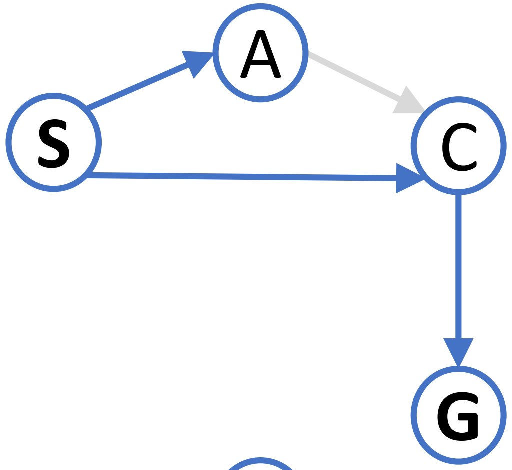
D) ~~S~~, ~~S-A~~, ~~S-C~~, ~~S-A-C~~, S-A-C-G

Quiz: A* Graph Search 2

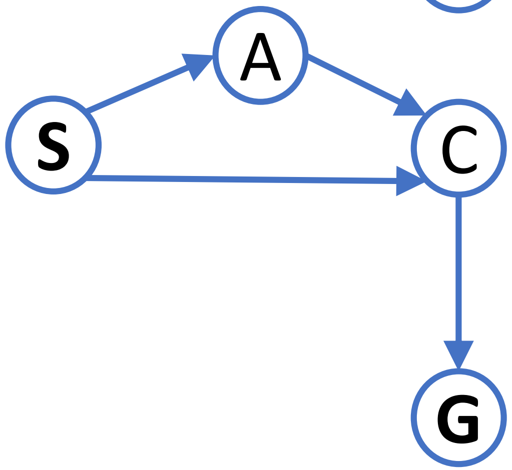
- What does the resulting graph tree look like?



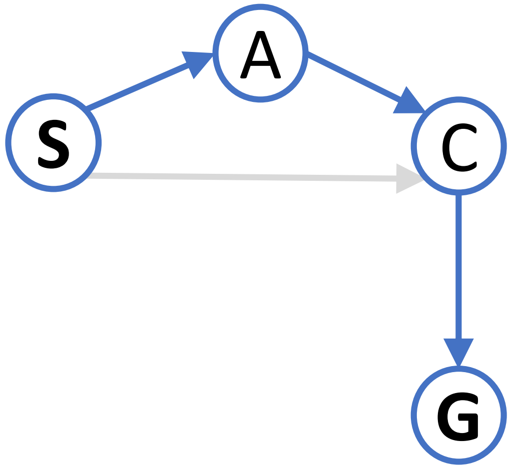
A)



B)

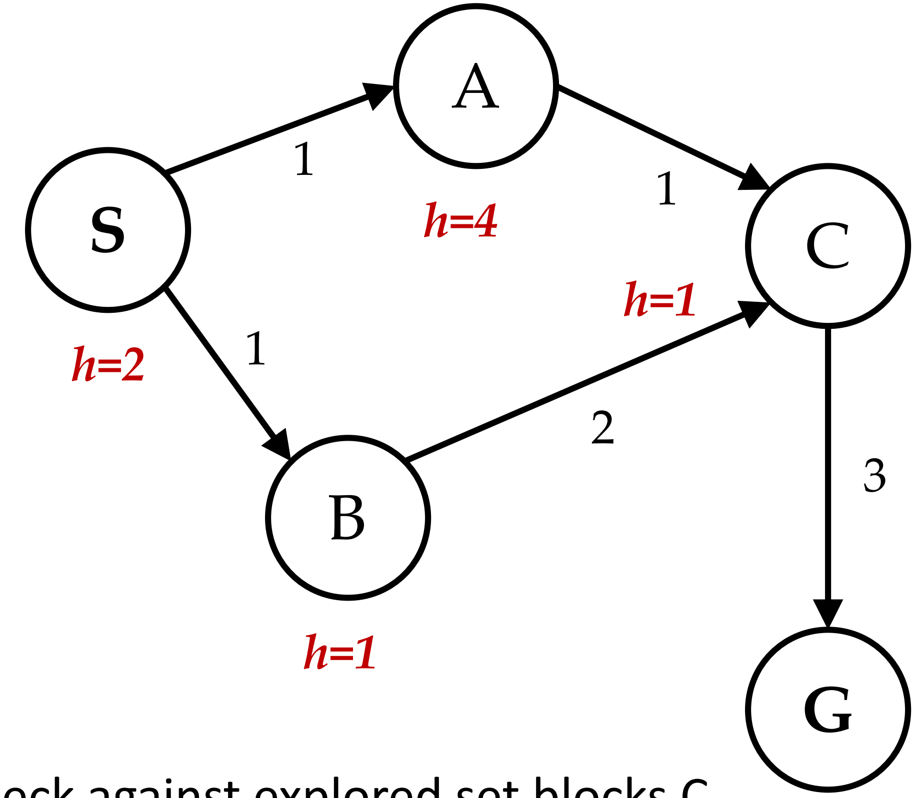


C & D)

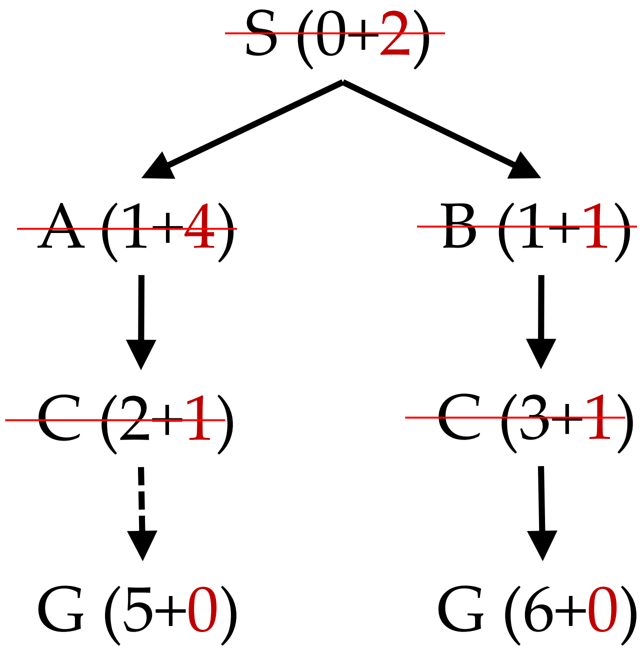


A* Graph Search Gone Wrong?

State space graph



Search tree

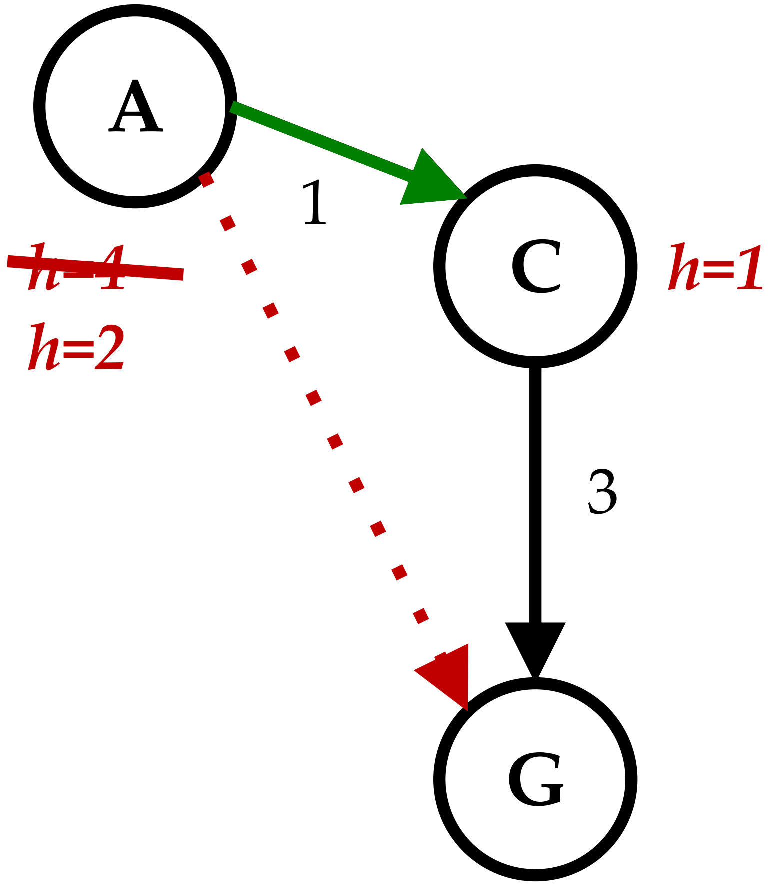


-Simple check against explored set blocks C
-Fancy check allows new C if cheaper than old but requires recalculating C's descendants

Explored Set: S B C A

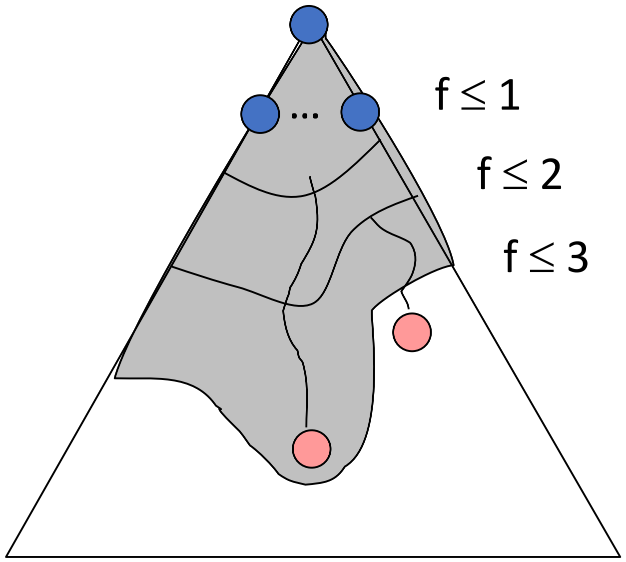
Consistency of Heuristics

- 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)
    - triangle inequality: h(A) ≤ c(A-C) + h(C)
- Consequences of consistency:
  - The f value along a path never decreases
    - h(A) ≤ cost(A to C) + h(C)
  - A* graph search is optimal



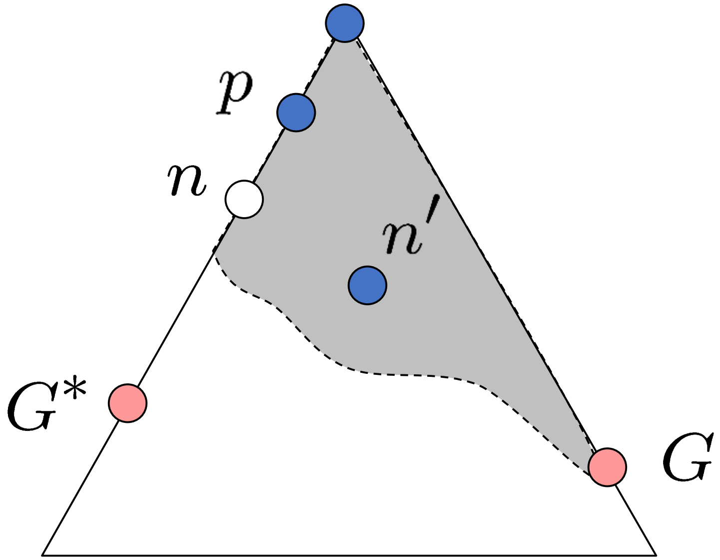
Optimality of A* Graph Search

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



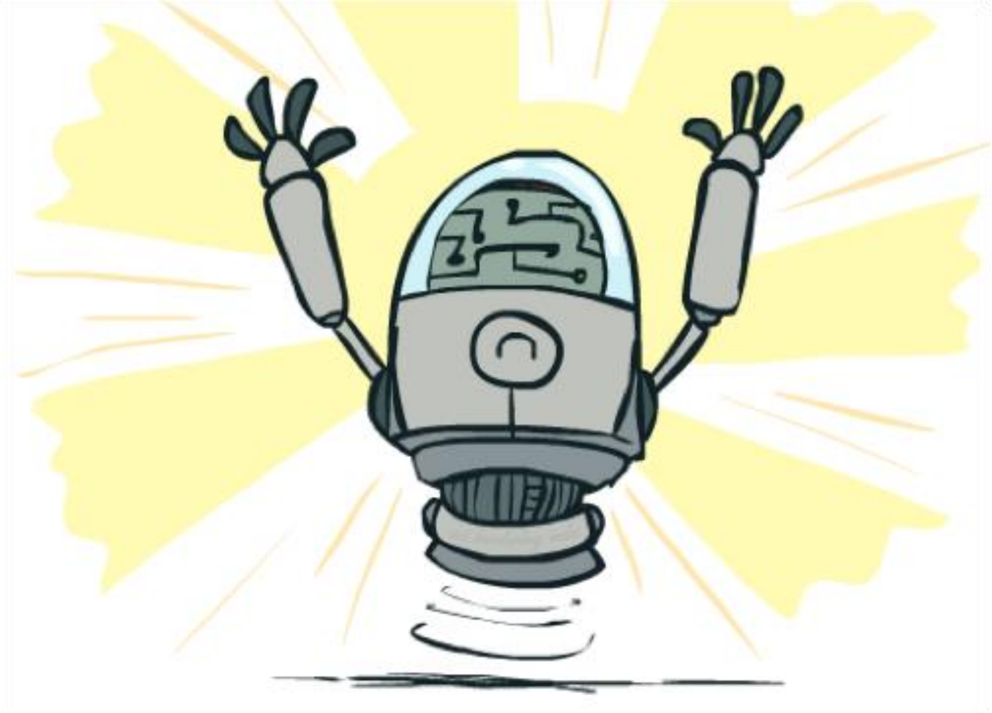
Optimality of A* Graph Search: Proof

- 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!



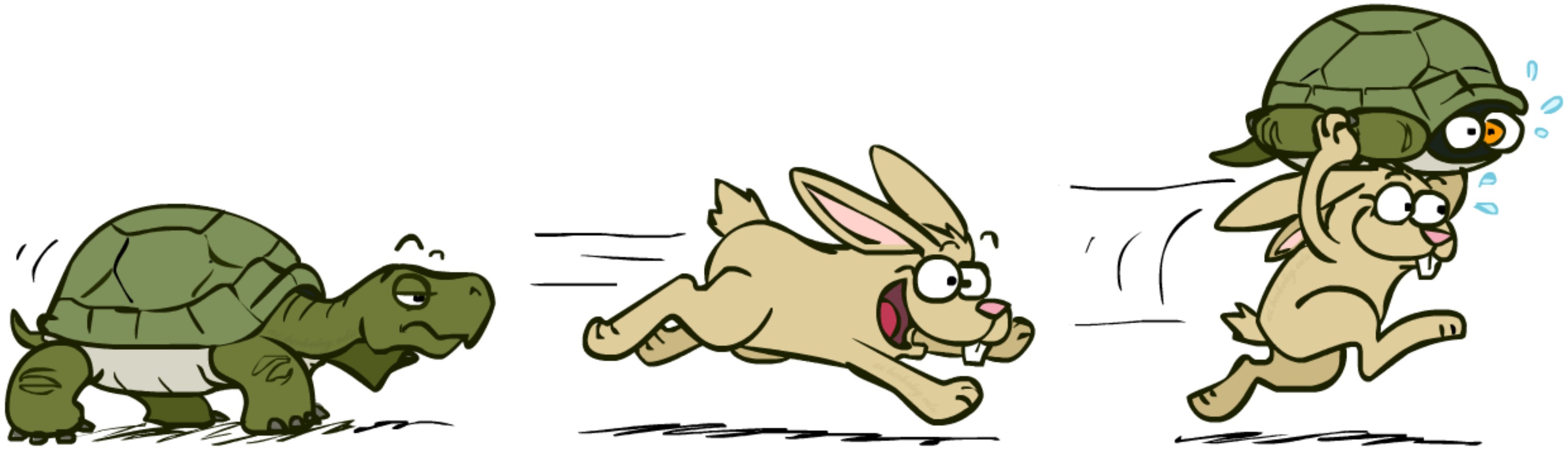
Optimality of A* Search

- 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



Summary of A*

- 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



Summary

Shuai Li
https://shuaili8.github.io

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

Questions?