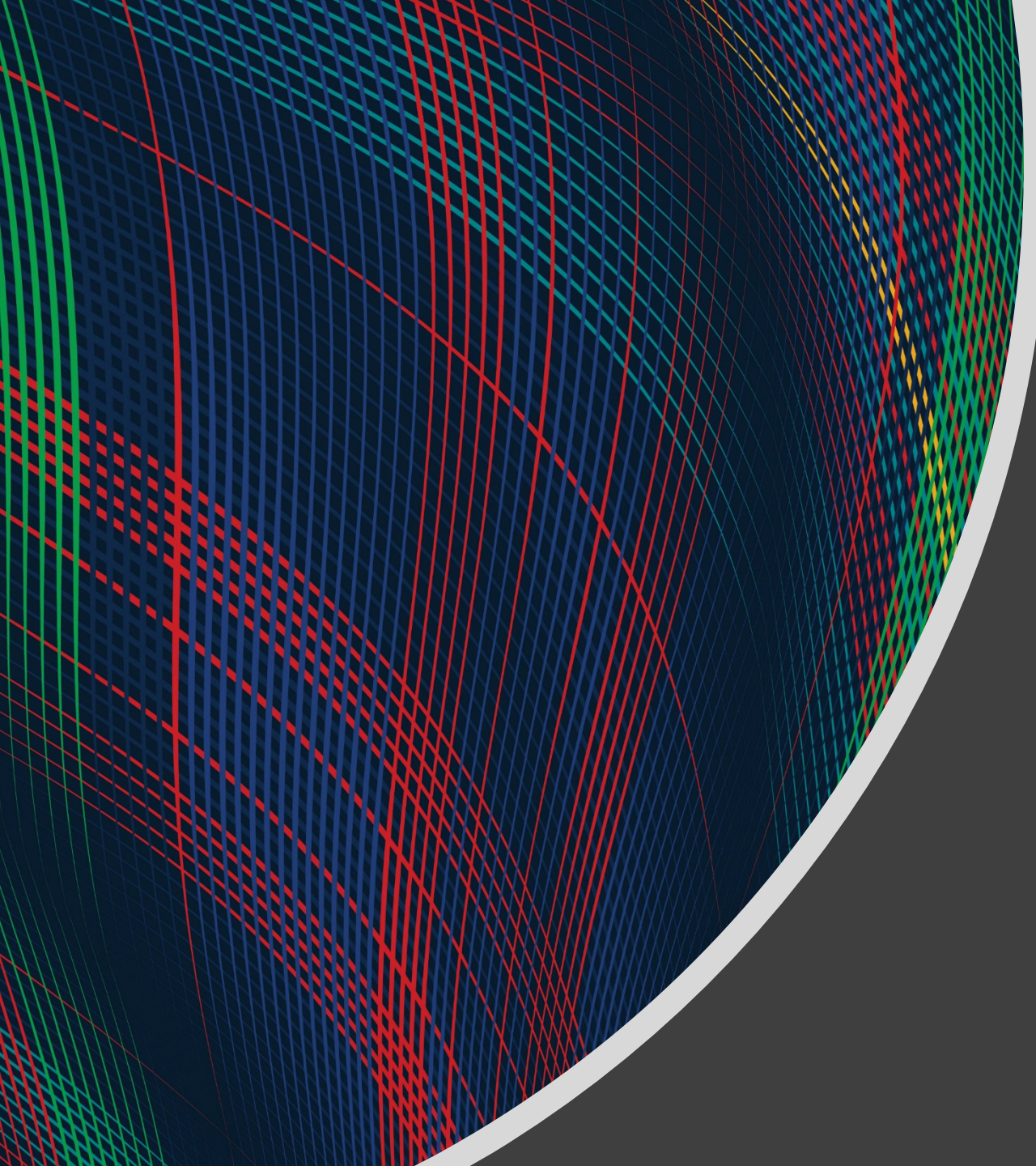


An abstract graphic on the left side of the slide, featuring a sphere-like shape composed of a dense grid of intersecting red, blue, and green lines. The lines are curved and follow the contours of the sphere, creating a complex, woven pattern. The sphere is set against a dark gray background.

07-280 AI & ML I Heuristics Search

Instructors:
Pat Virtue & Nihar Shah

Plan

Pre-reading

- Tree search vs graph search
- BFS, DFS, UCS

Today

- A few notes about sear
- UCS – one important modification
- Heuristics
- Greedy search
- A* search
 - Optimality
- [More on heuristics]

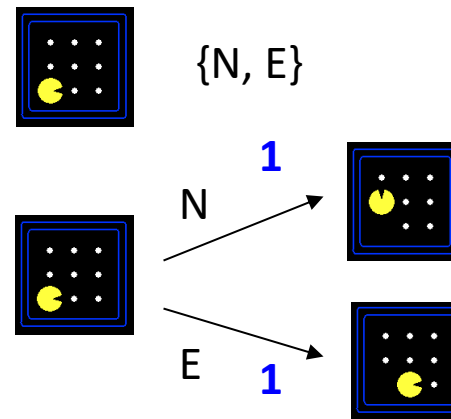
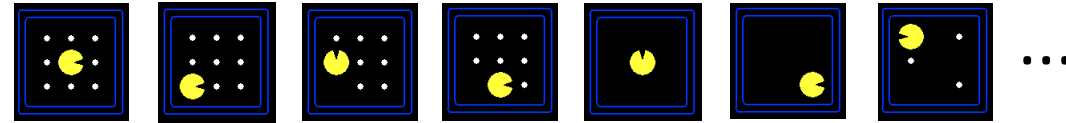


Search Problems

Search Problems

A **search problem** consists of:

- A state space
- For each state, a set
Actions(s) of allowable actions
- A transition model $\text{Result}(s,a)$
- A step cost function $c(s,a,s')$
- A start state and a goal test



A **solution** is a sequence of actions (a plan) which transforms the start state to a goal state

State Space Representation and Size?

World state:

- Agent positions: 120
- Food count: 30
- Ghost positions: 12
- Agent facing: NSEW

State representation

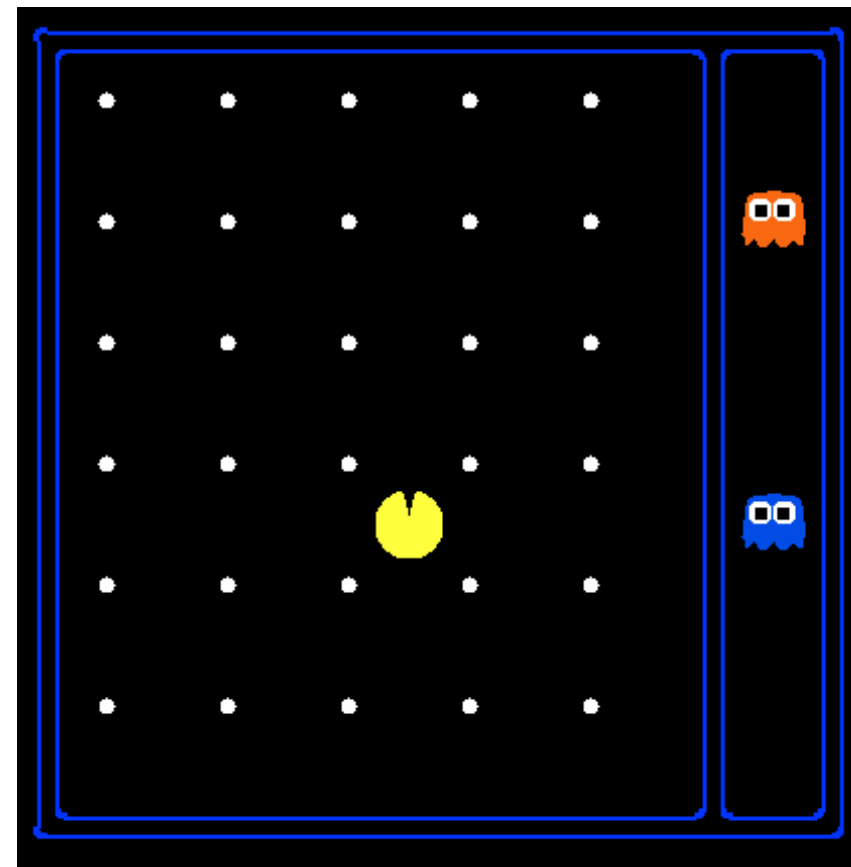
- World states?
 $(p, f_1, \dots, f_{30}, g_1, g_2, d)$
- Eat top left dot?
 (p)
- States for eat-all-dots?
 $(p, f_1, \dots, f_{30})$

State space size

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

$$120$$

$$120 \times (2^{30})$$



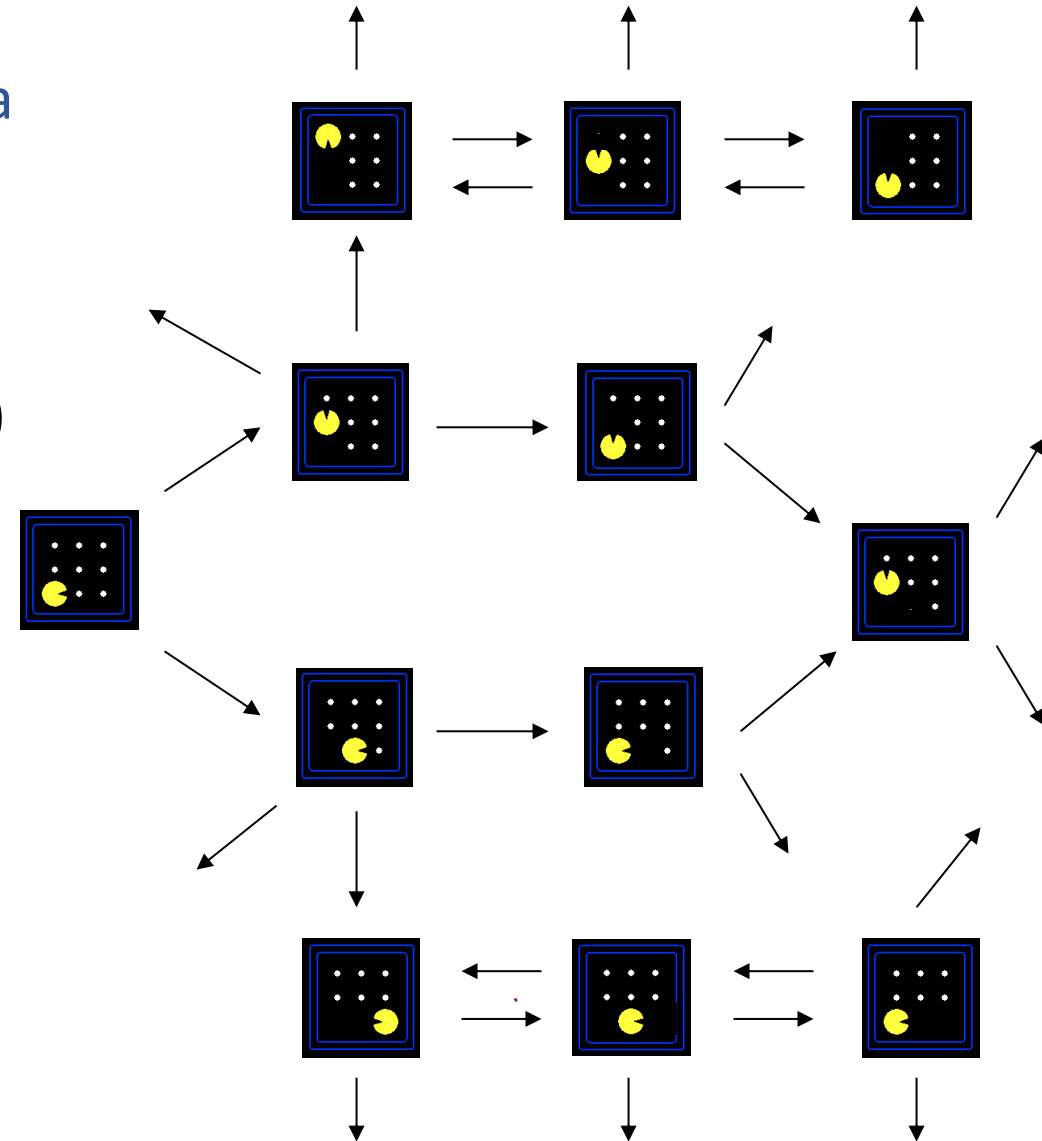
State Space Graphs

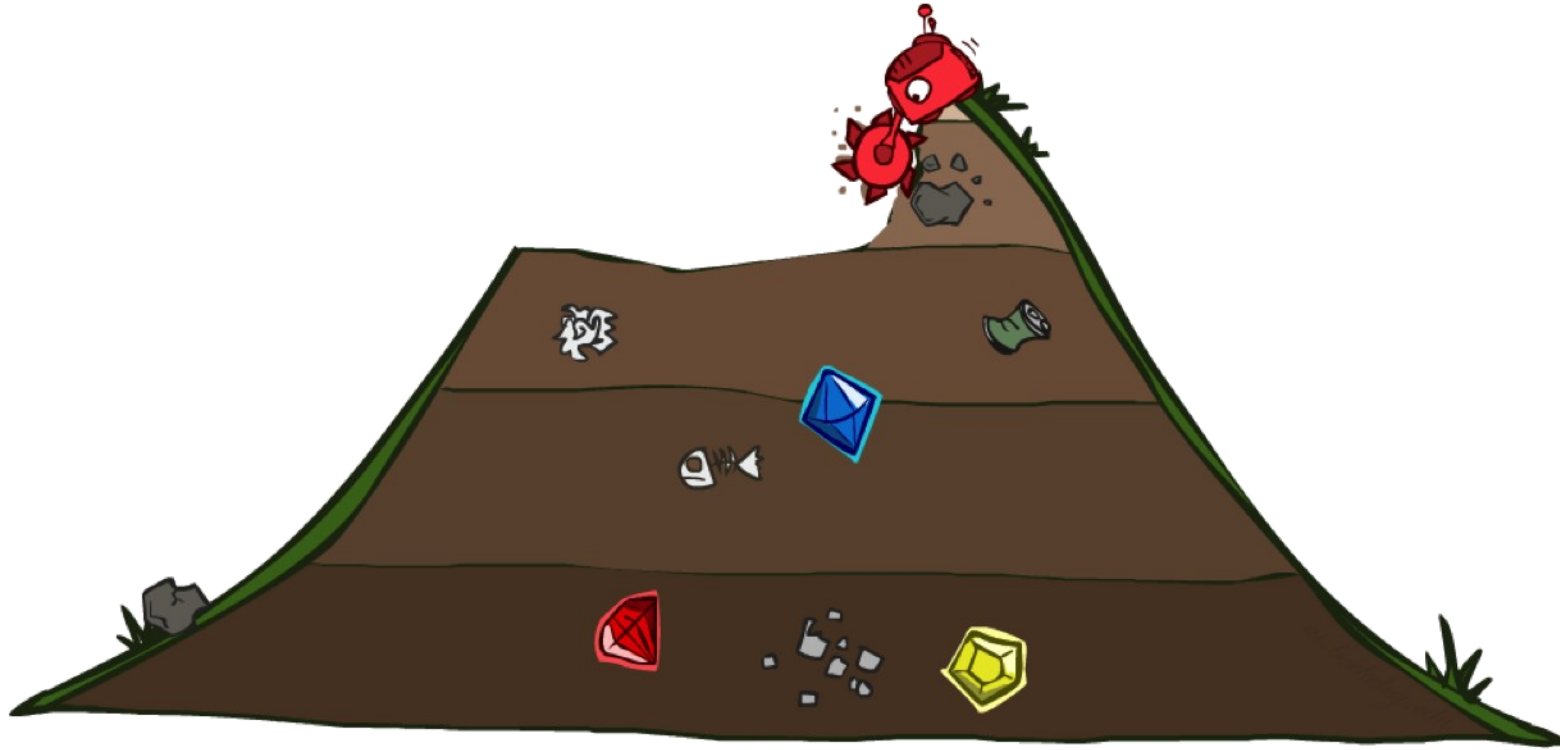
State space graph: A mathematical representation of a search problem

- Vertices are (abstracted) world configurations
- Edges represent transitions resulting from actions
- The goal test is a set of goal nodes (maybe only one)

In a state space graph, each state occurs only once!

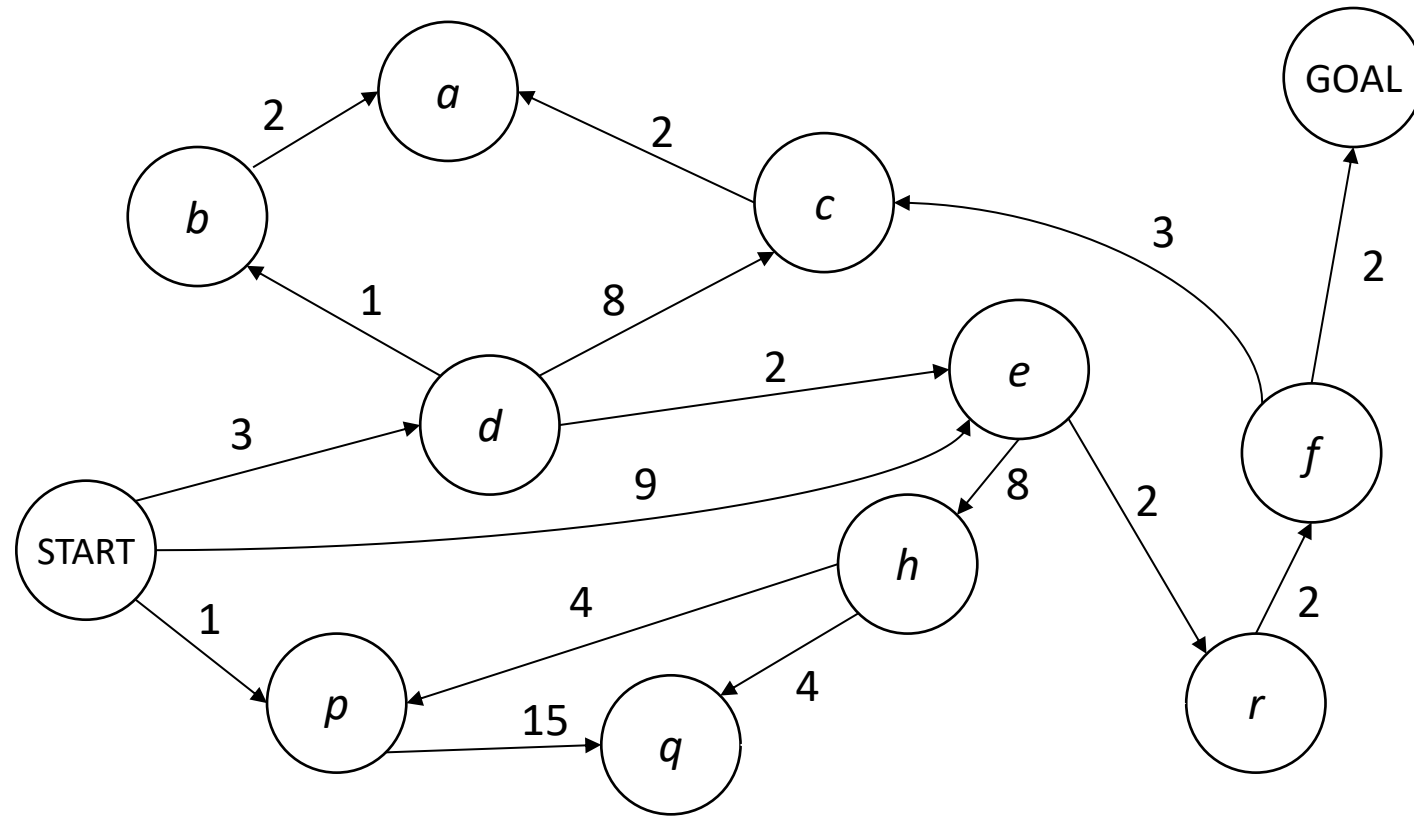
We can rarely build this full graph in memory (it's too big), but it's a useful idea





Uniform Cost Search

Finding a Least-Cost Path

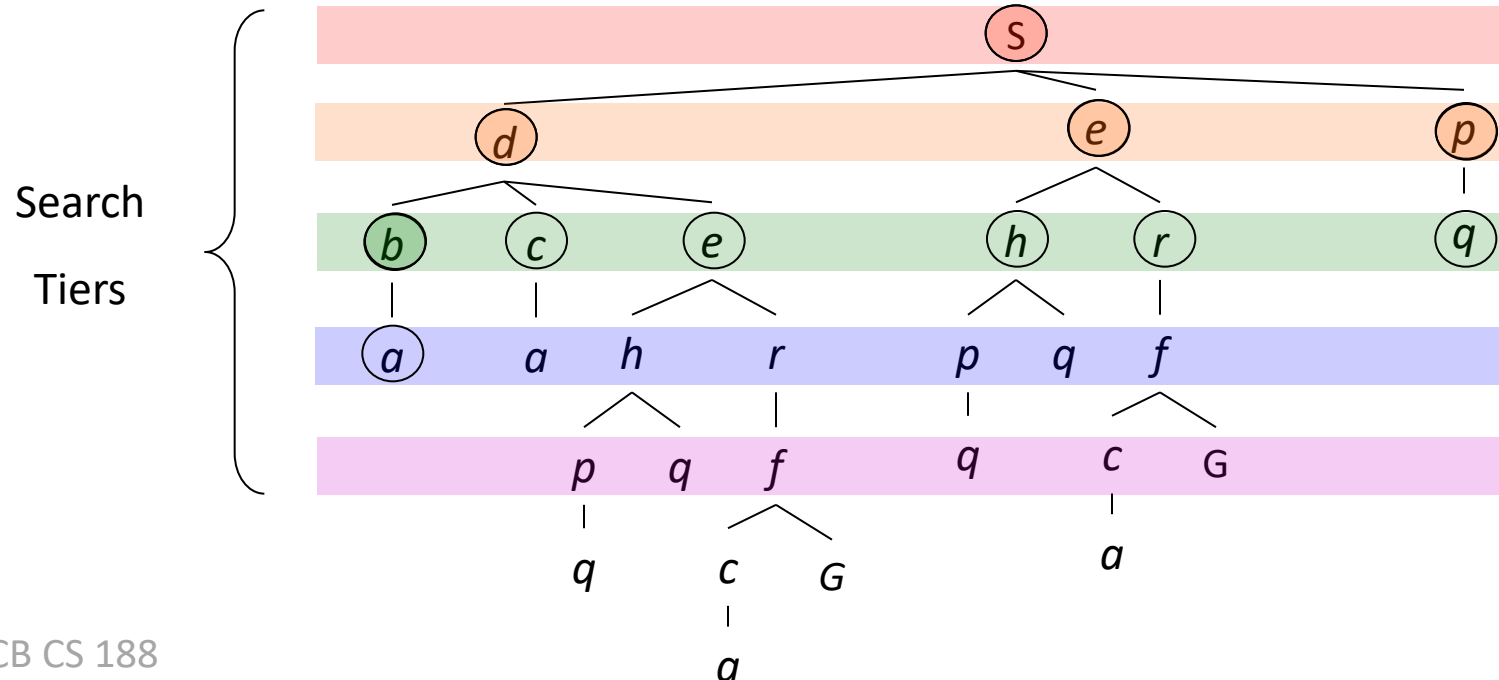
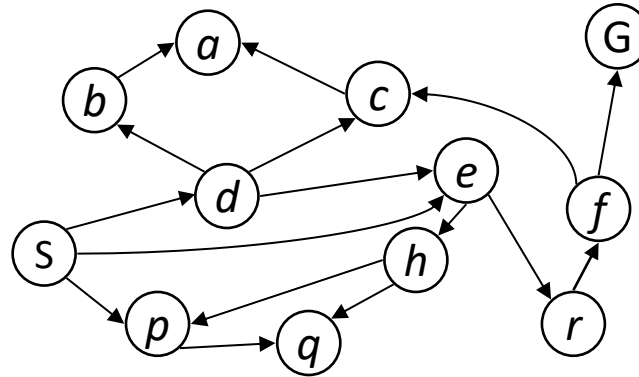


Breadth-First (Tree) Search

Strategy: expand a shallowest node first

Implementation:

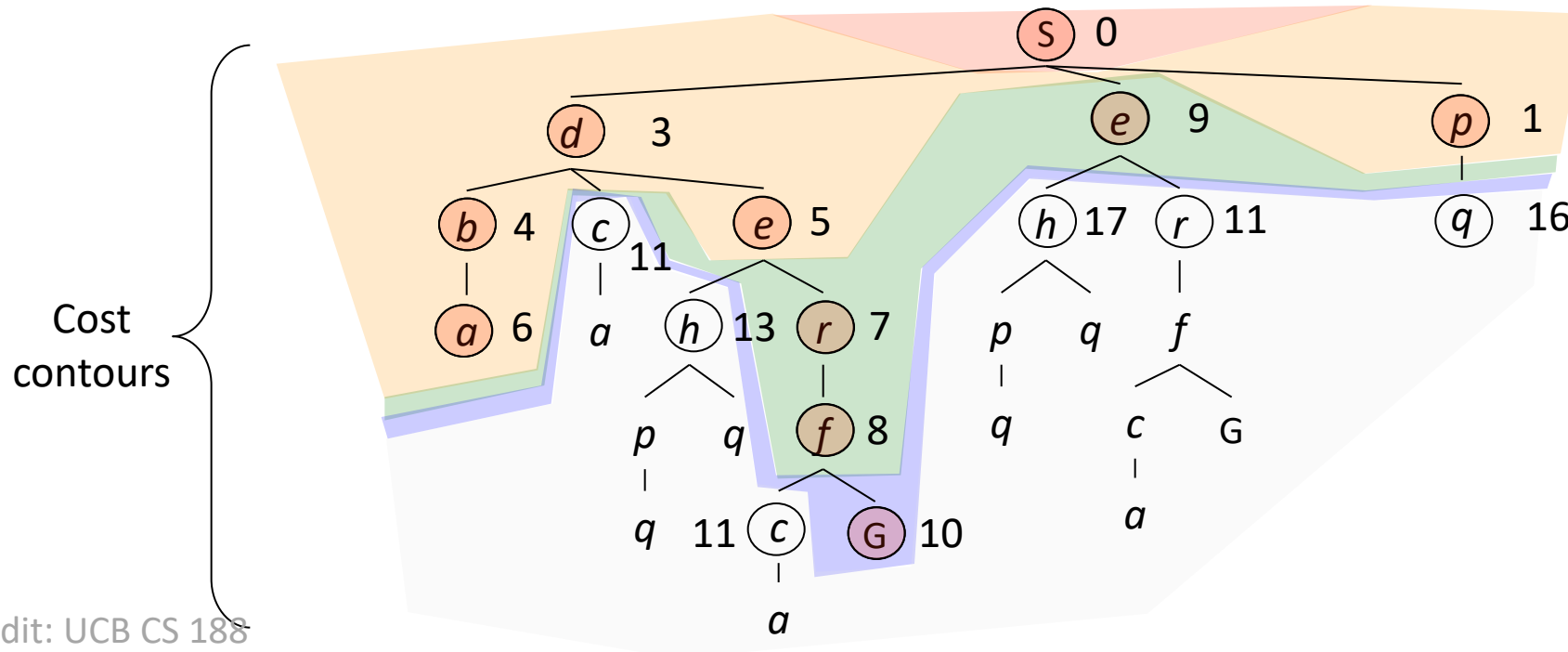
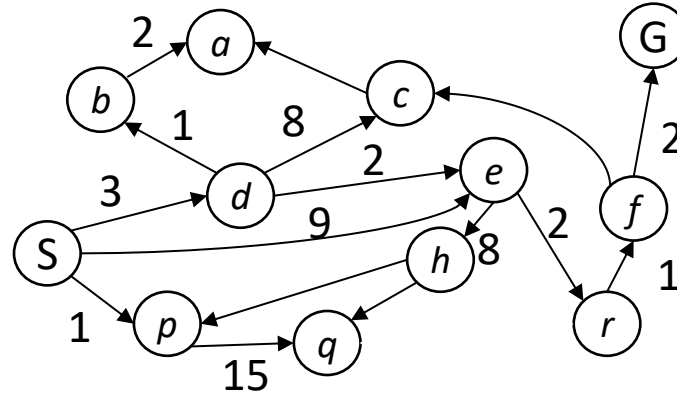
Frontier is a FIFO queue



Uniform Cost (Tree) Search

Strategy: expand a cheapest node first:

Frontier is a priority queue
(priority: cumulative cost)



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**

A Note on Implementation

Nodes have

`state`, `parent`, `action`, `path-cost`

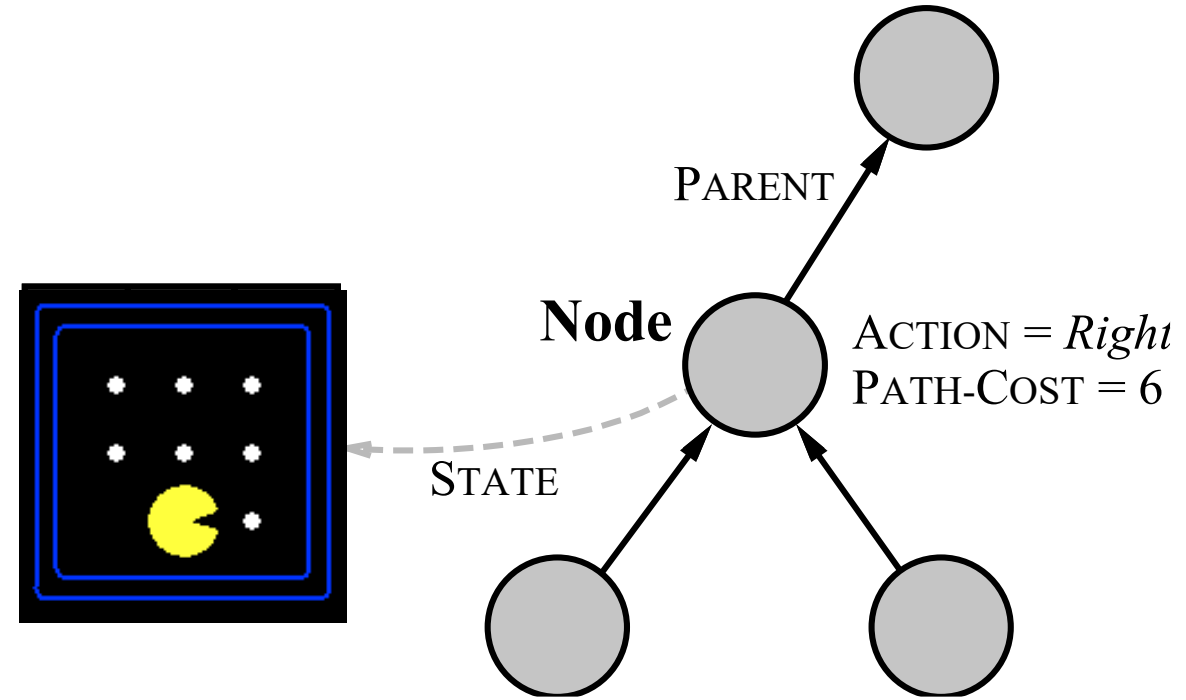
A child of `parent_node` by action `a` has:

`state` = `result(parent_node.state, a)`

`parent` = `parent_node`

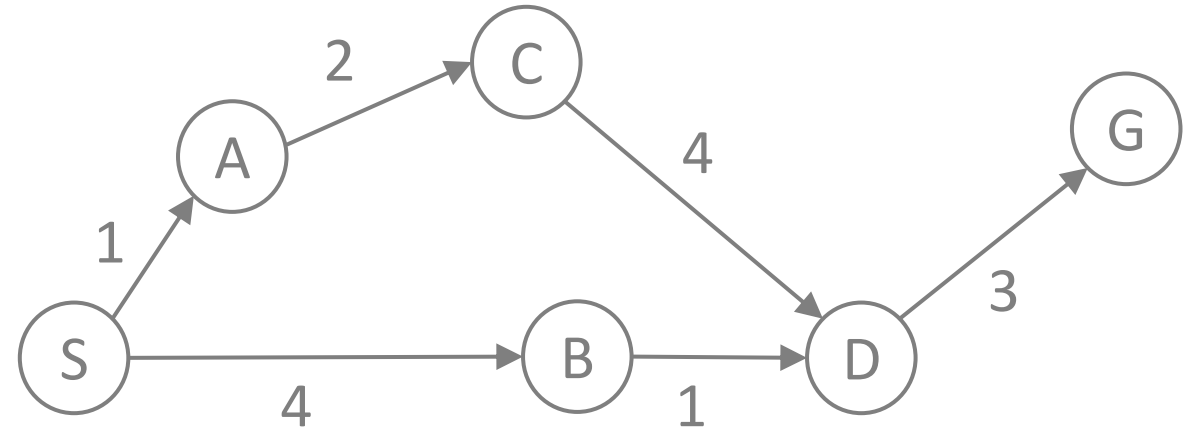
`action` = `a`

`path-cost` = `parent_node.path_cost` +
`step_cost(parent_node.state, a, self.state)`



Extract solution by tracing back parent pointers, collecting actions

Walk-through UCS



Walk-through UCS

Frontier

~~S: 0~~

~~S-A: 1~~

~~S-B: 4~~

~~S-A-C: 3~~

S-A-C-D: 7

S-B-D: 5 ??

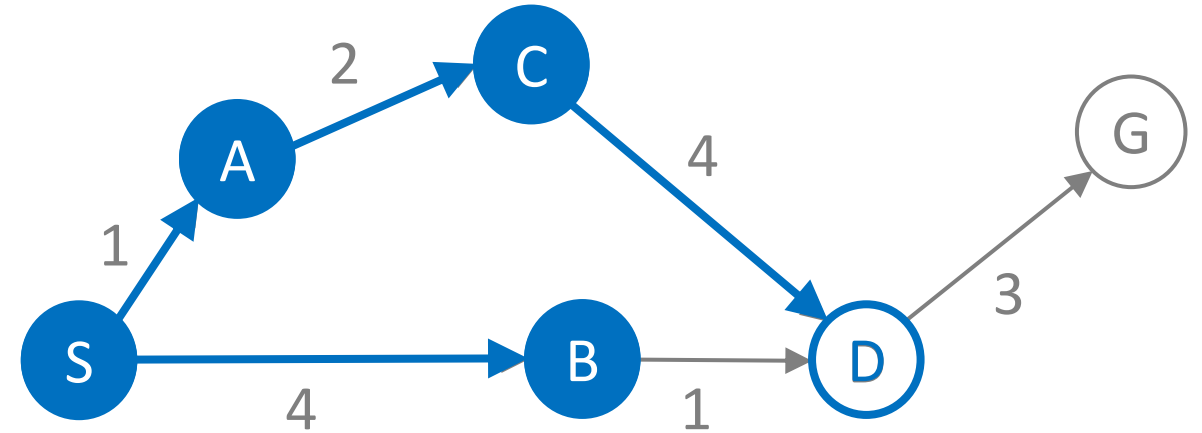
Explored

S

A

C

B



Walk-through UCS

Frontier

~~S: 0~~

~~S-A: 1~~

~~S-B: 4~~

~~S-A-C: 3~~

S-A-C-D: 7

~~S-B-D: 5~~

~~S-B-D-G: 8~~

Explored

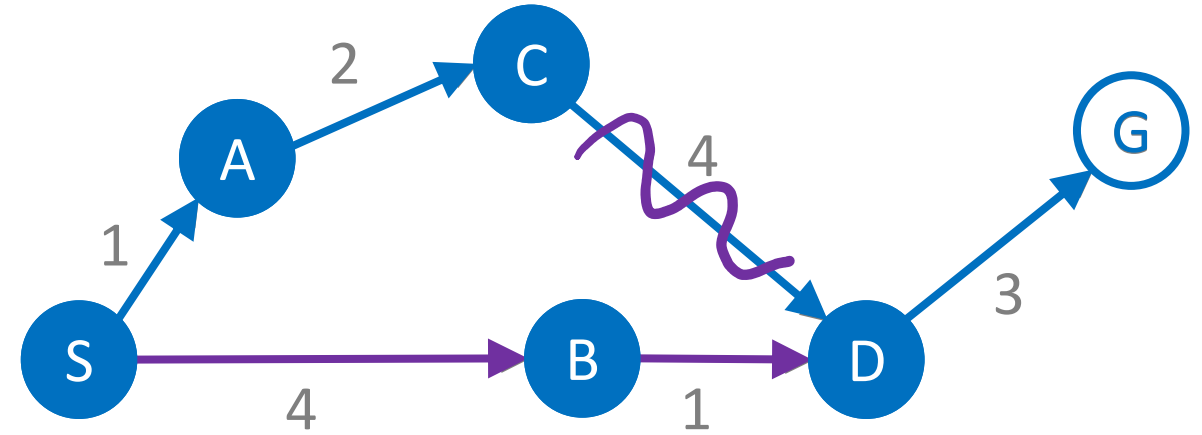
S

A

C

B

D



Replaced by
better path to D!

Result: S-B-D-G (path cost 8)

function UNIFORM-COST-SEARCH(**problem**) **returns** a solution, or failure

initialize the **explored set** to be empty

initialize the **frontier** as a **priority queue** using node **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**

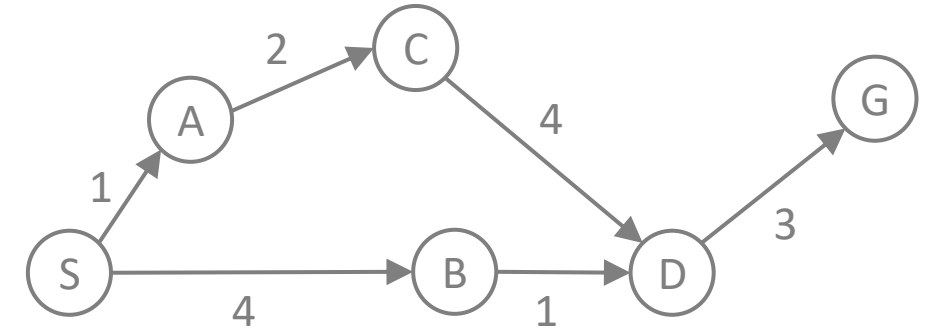
 for each resulting **child** from node

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



Uniform Cost Issues

Remember:

- UCS explores increasing cost contours

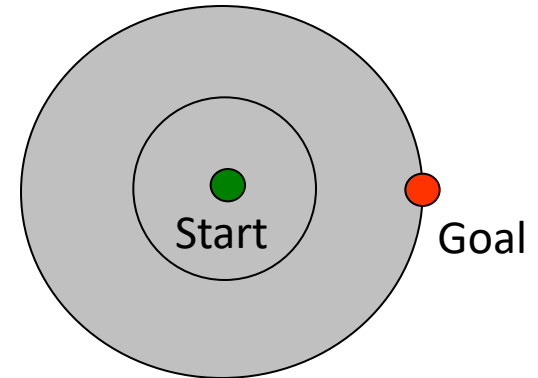
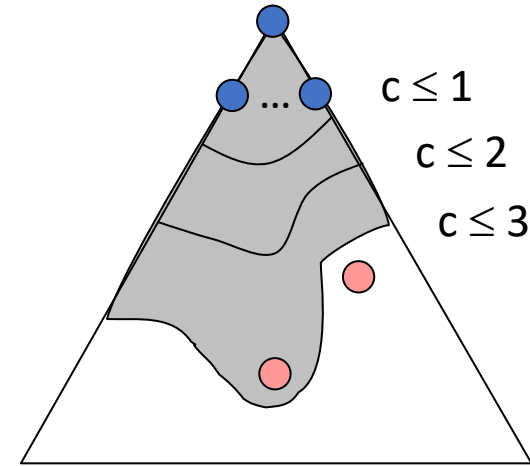
The good:

- UCS is complete and optimal!

The bad:

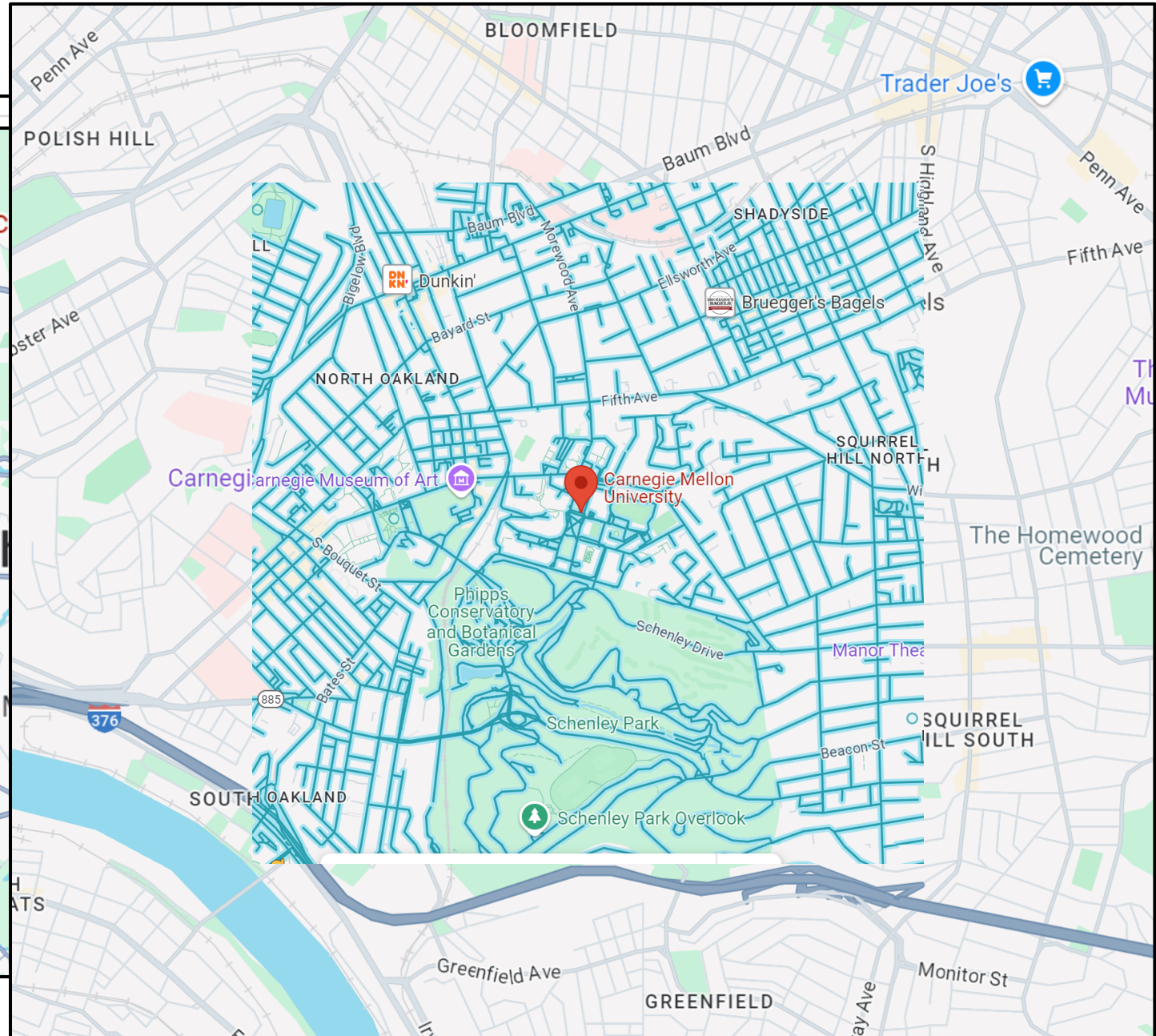
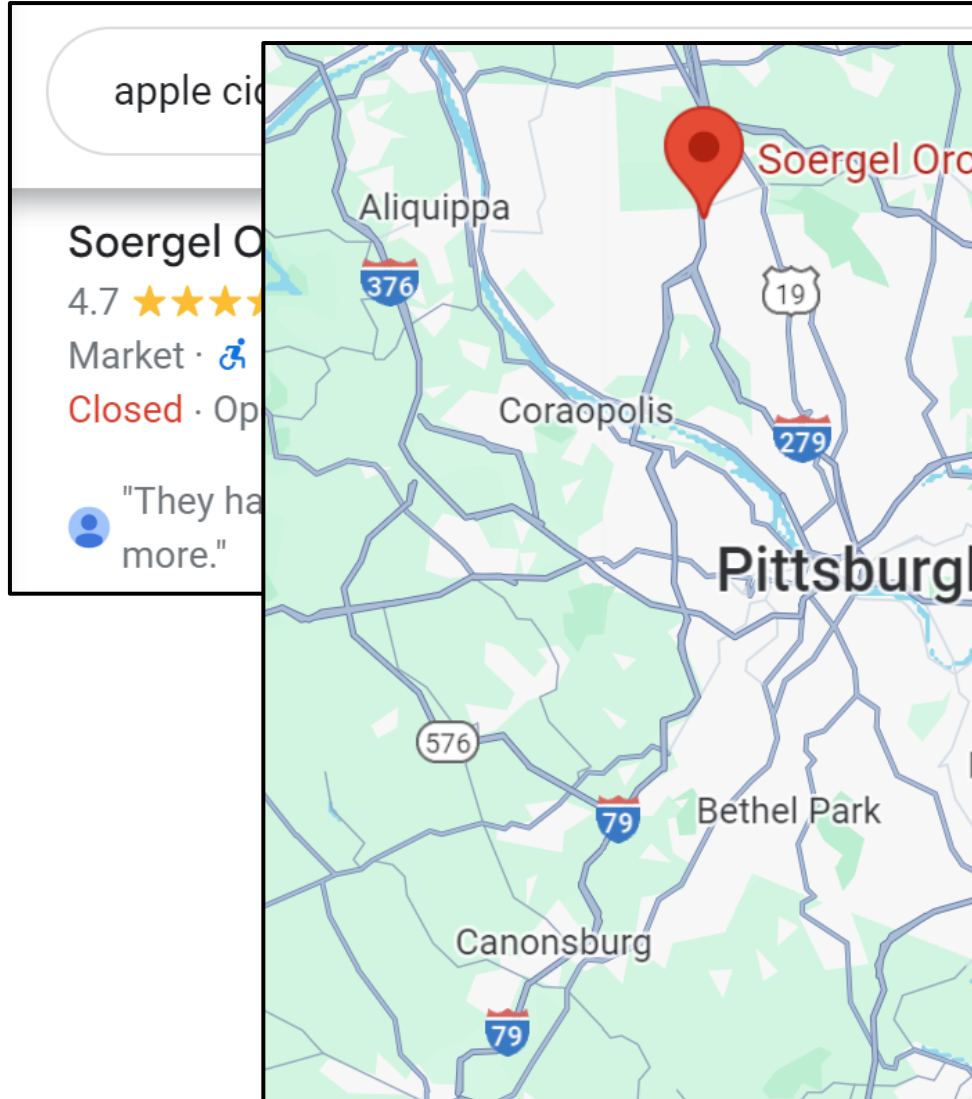
- Explores options in every “direction”
- No information about goal location

We'll fix that!

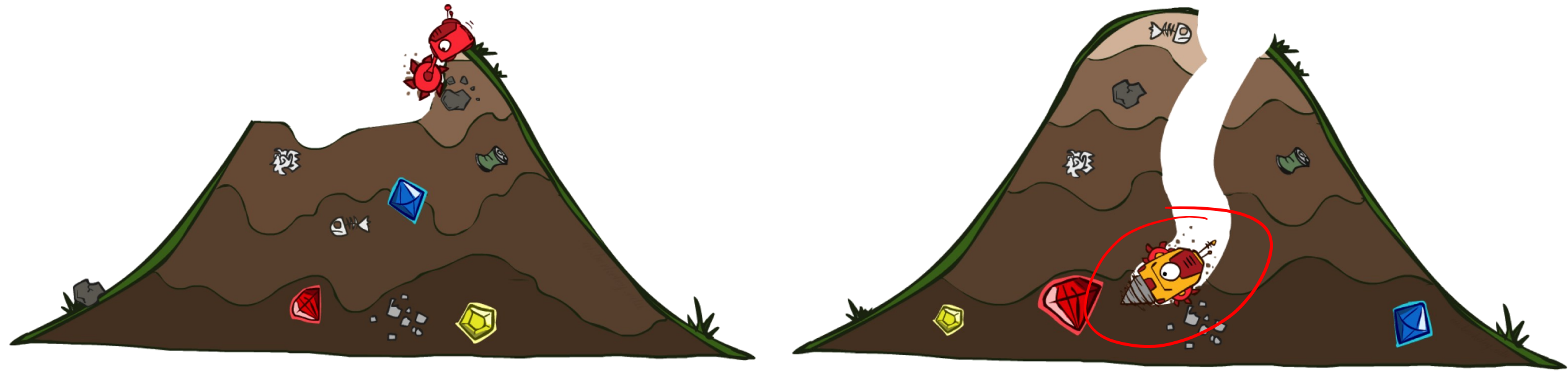


Informed Search

Donuts ASAP!



Uninformed vs Informed Search



Today

Informed Search

- Heuristics
- Greedy Search
- A* Search



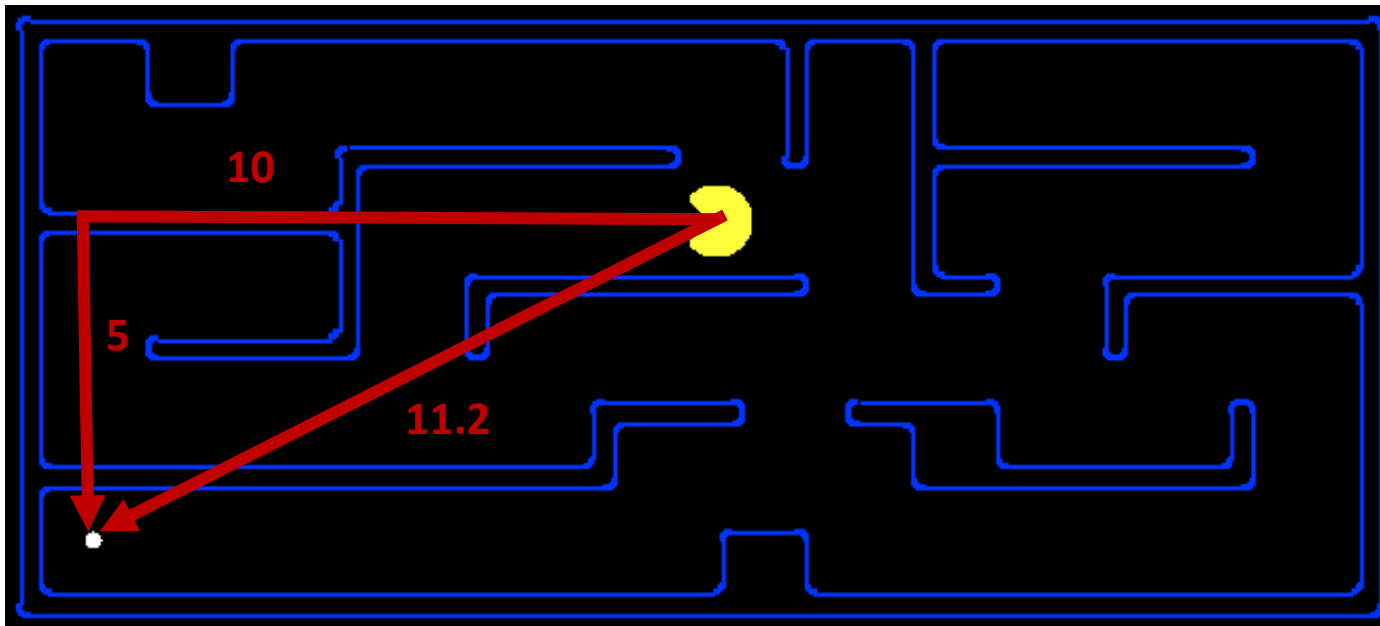
Informed Search



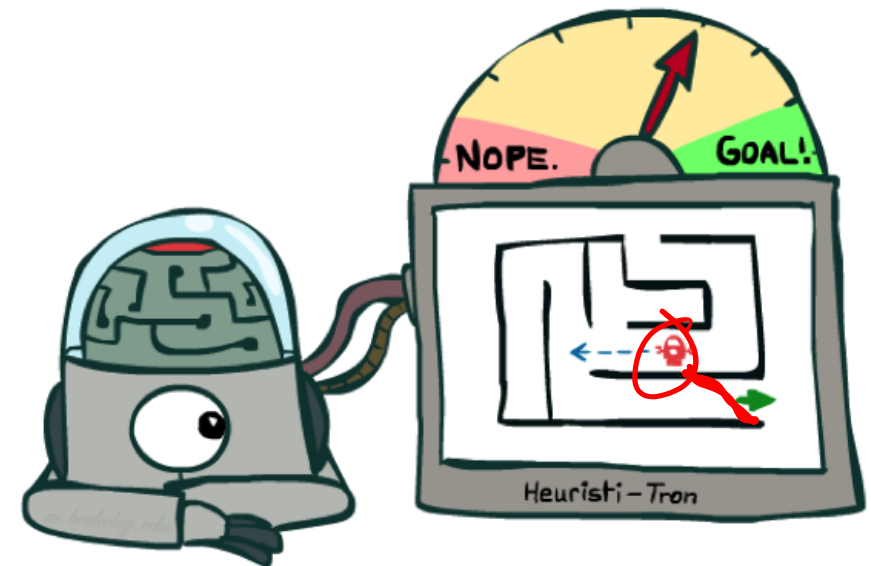
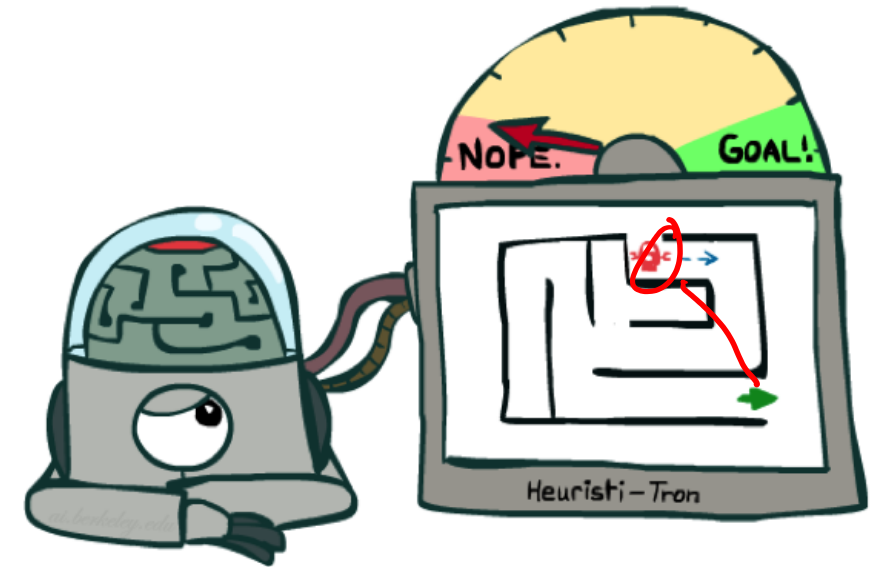
Search Heuristics

A heuristic is:

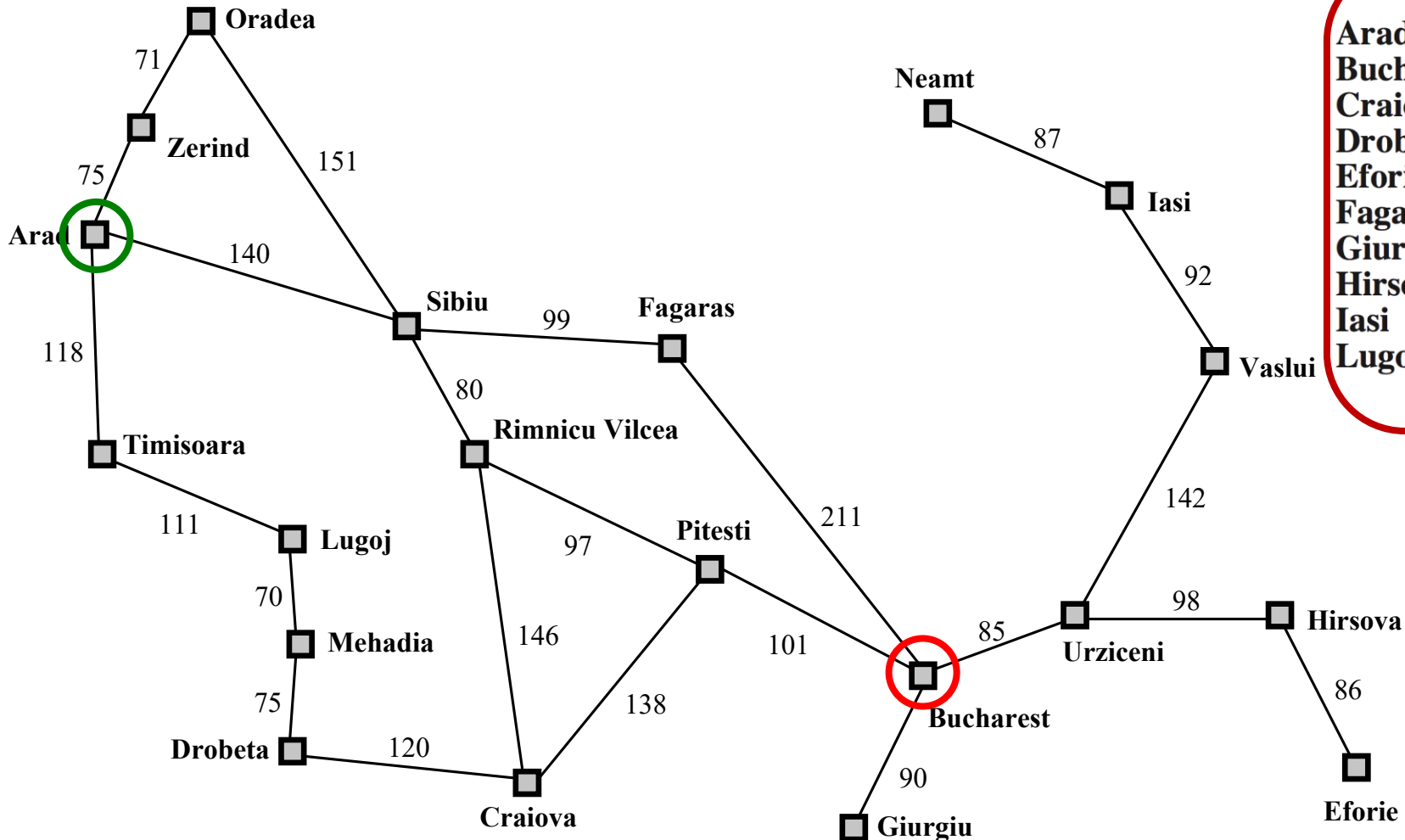
- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- Examples: Manhattan distance, Euclidean distance for pathing



Image/slide credit: Ketrina Yim, UCB CS 188



Example: Euclidean distance to Bucharest

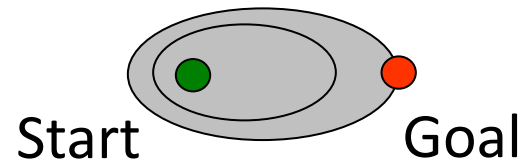


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

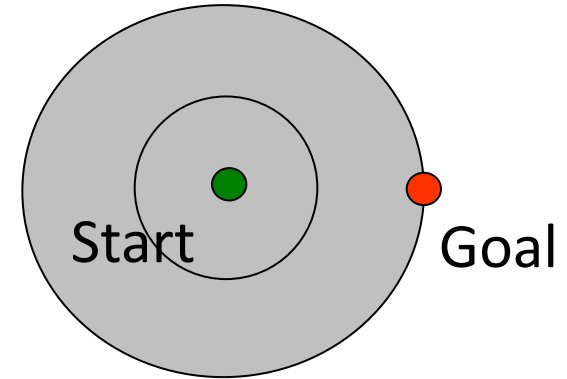
$h(\text{state}) \rightarrow \text{value}$

Effect of heuristics

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



Informed

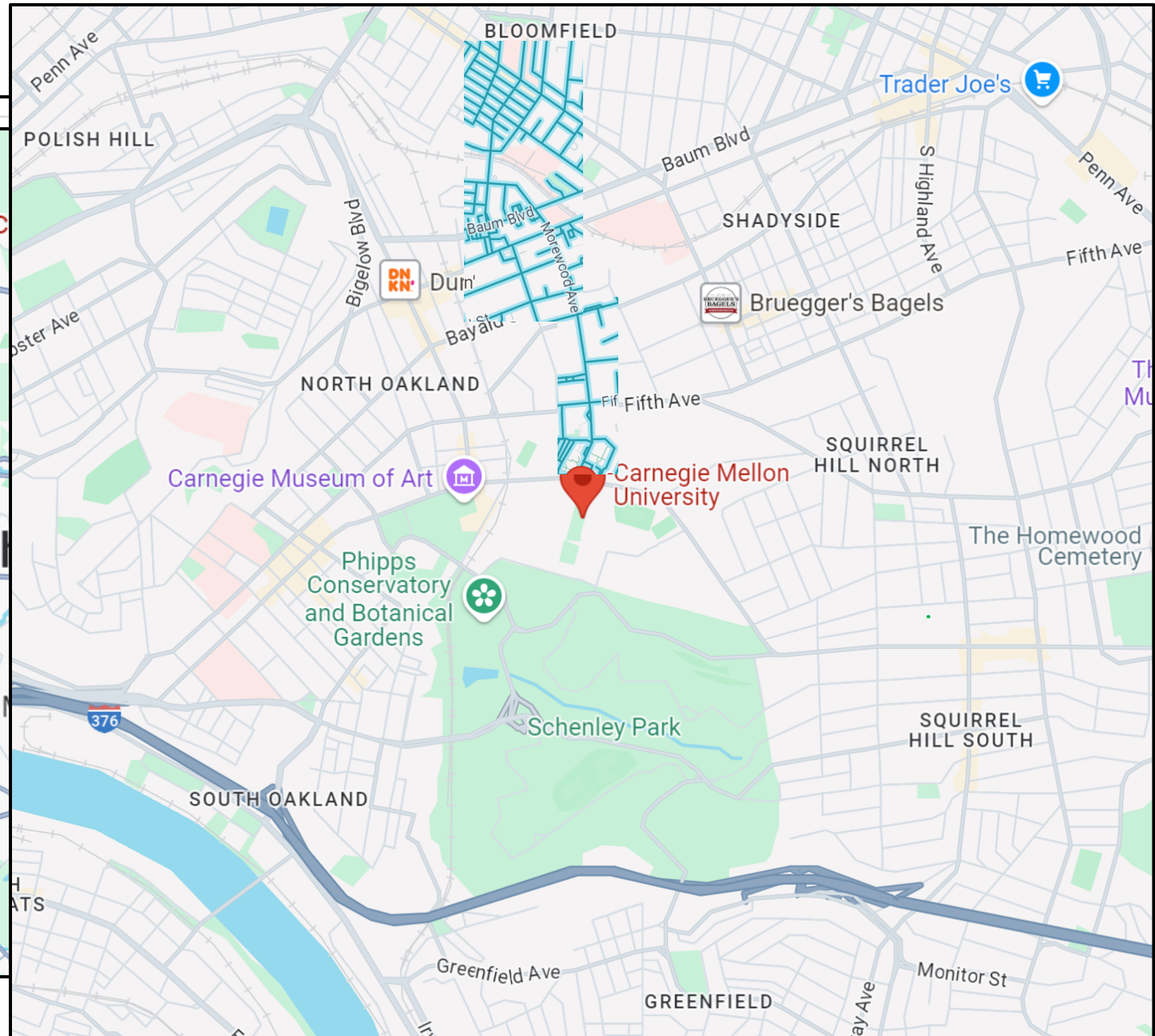
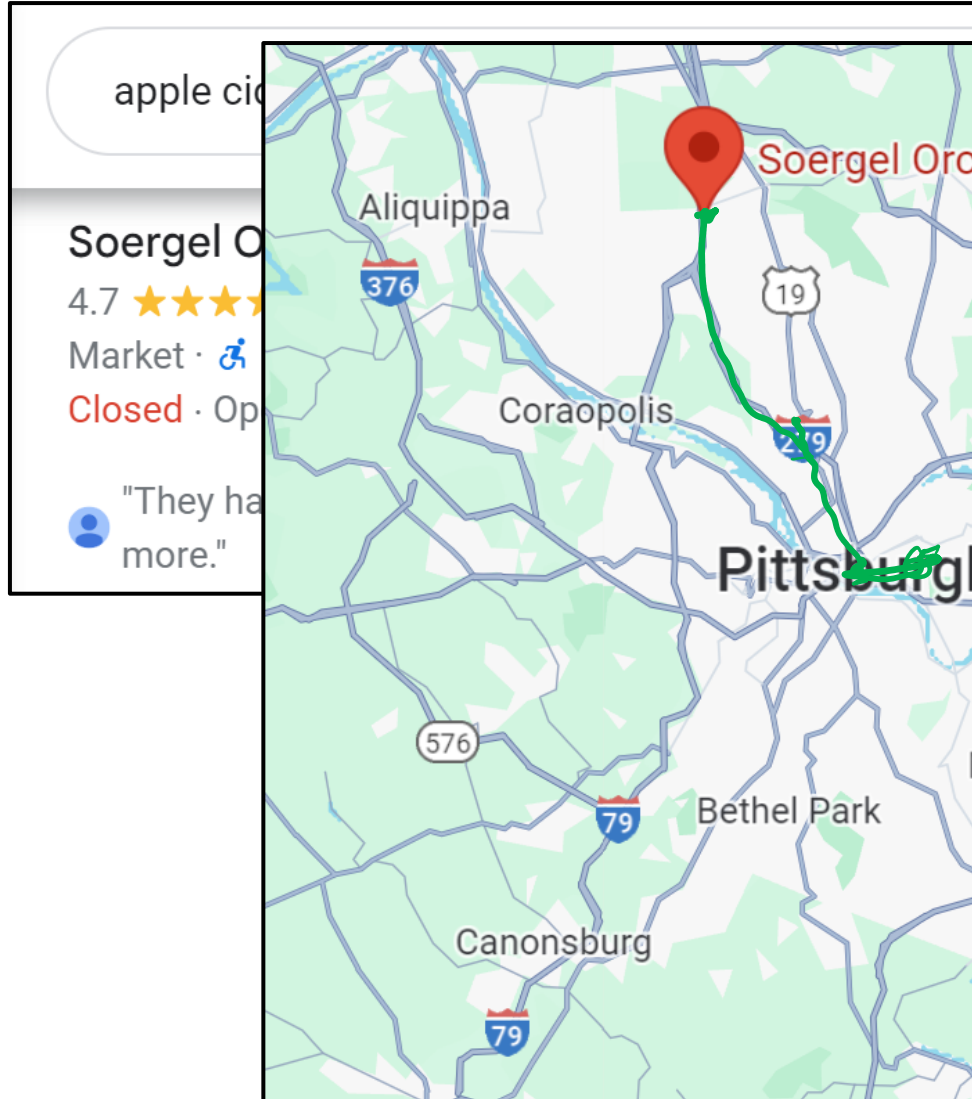


Uninformed

Greedy Search



Donuts ASAP!



A* Search



A* Search



UCS



Greedy



A*

A* Search

$$\underline{f(S-A-B)} = g(S-A-B) + h(B)$$



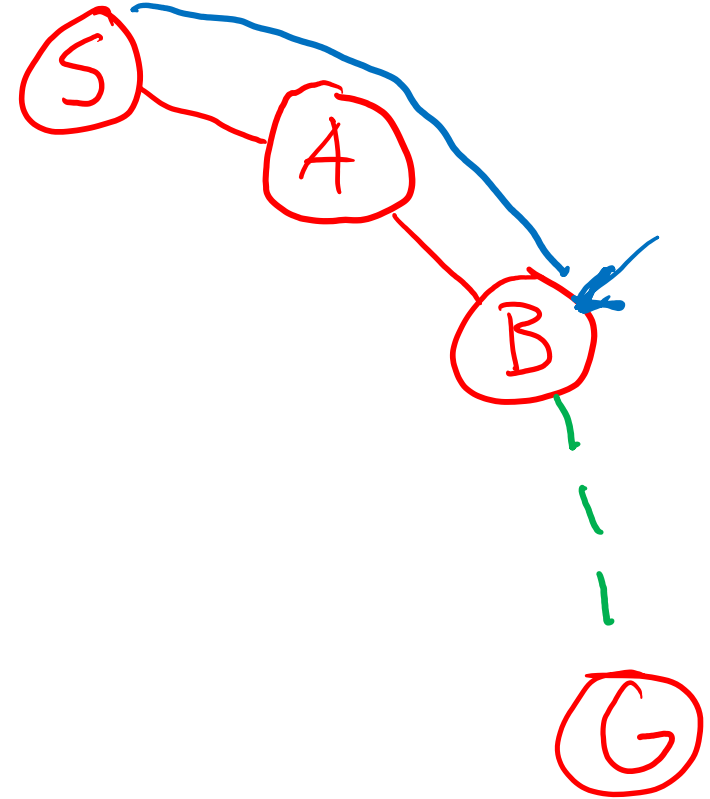
UCS



Greedy



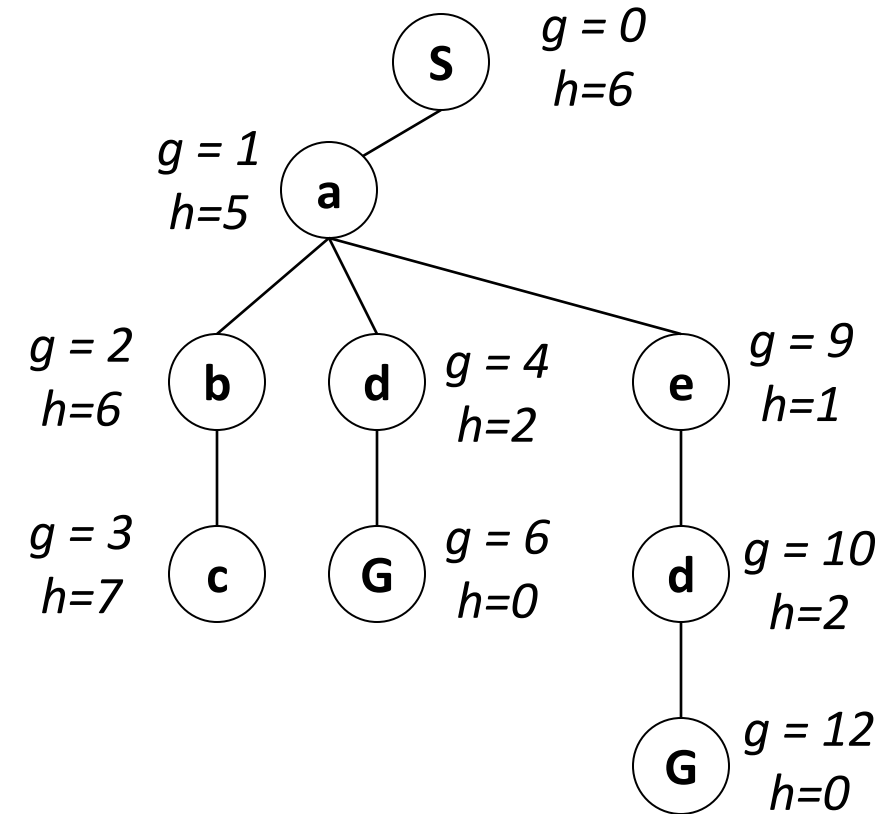
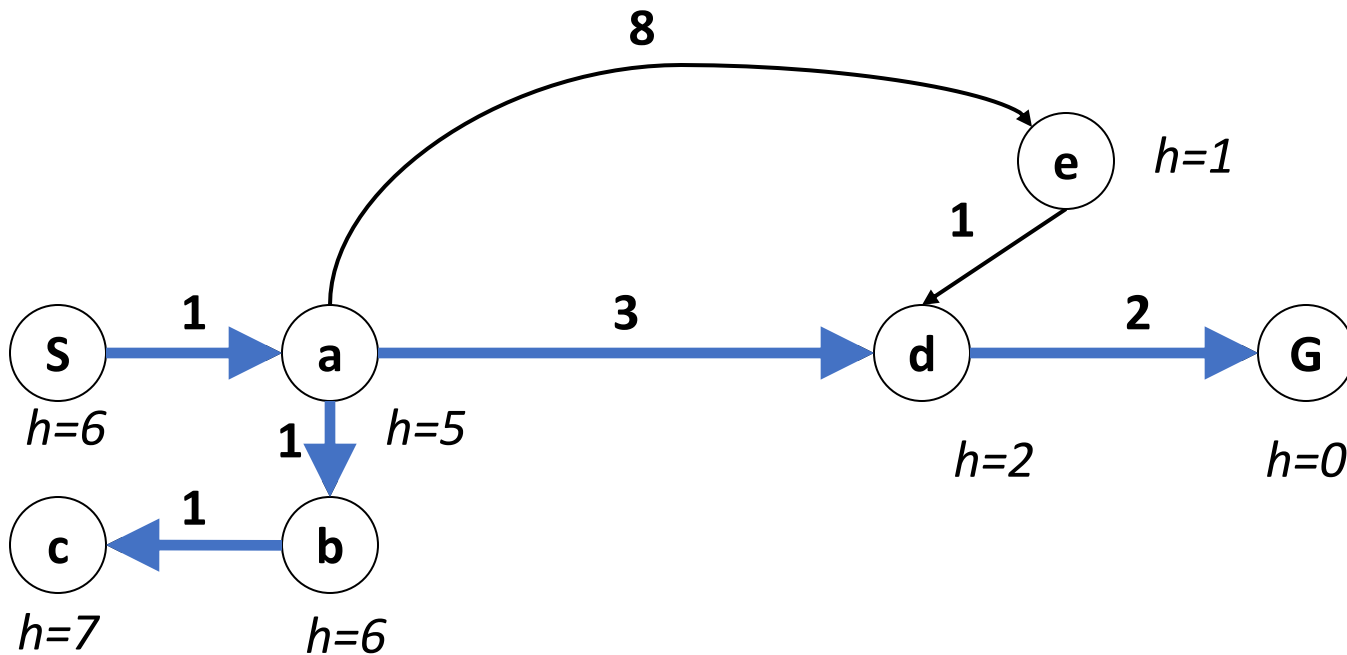
A*



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

Combining UCS and Greedy

Uniform-cost orders by path cost, or *backward cost* $g(n)$

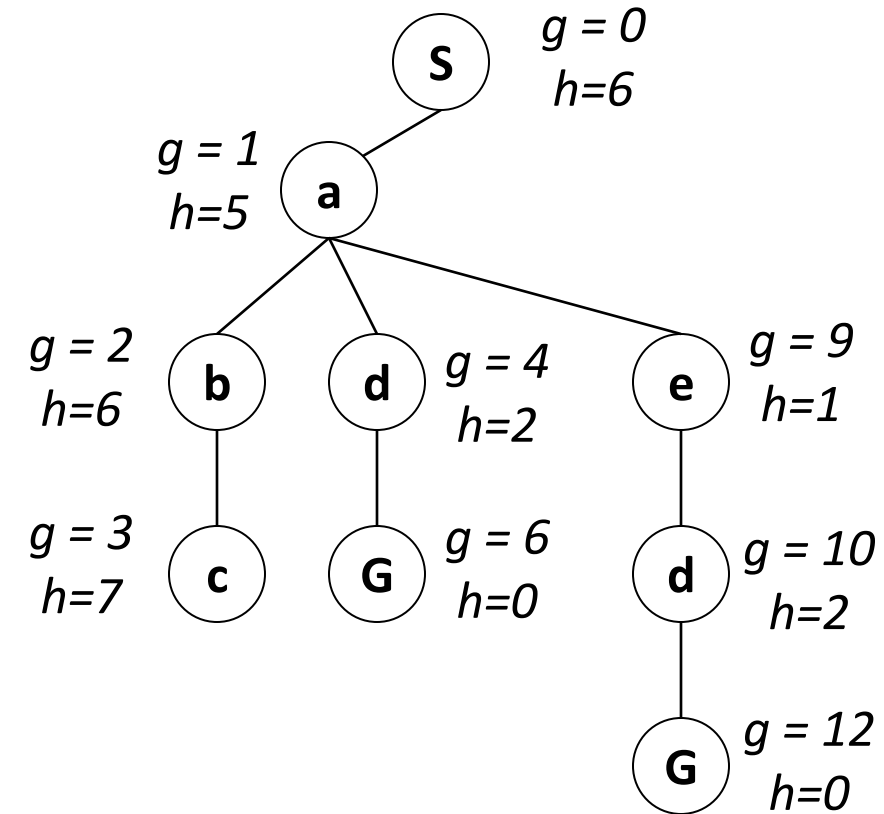
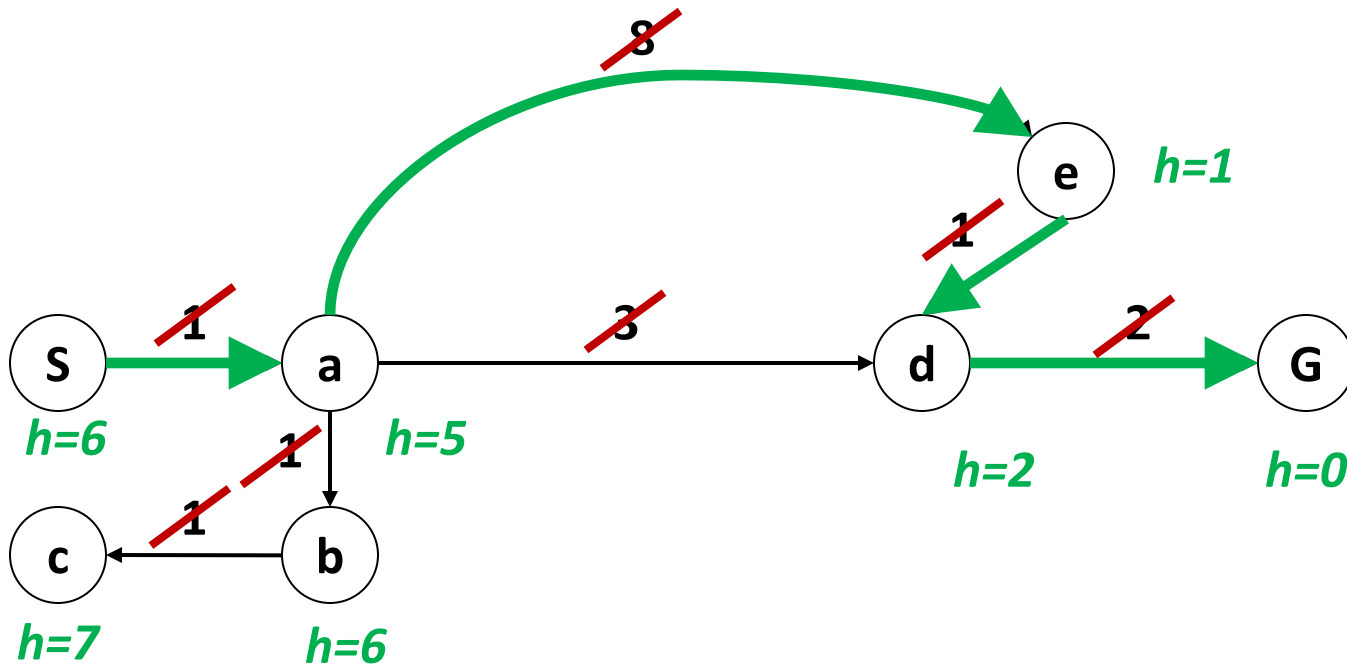


Example: Teg Grenager

Combining UCS and Greedy

Uniform-cost orders by path cost, or *backward cost* $g(n)$

Greedy orders by goal proximity, or *forward cost* $h(n)$

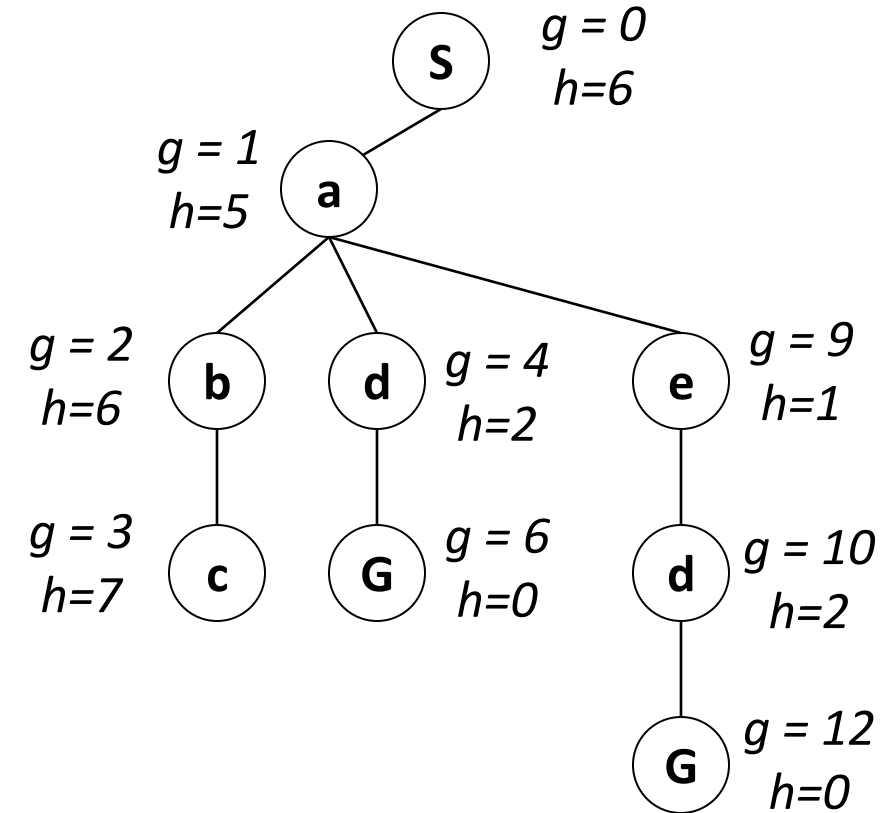
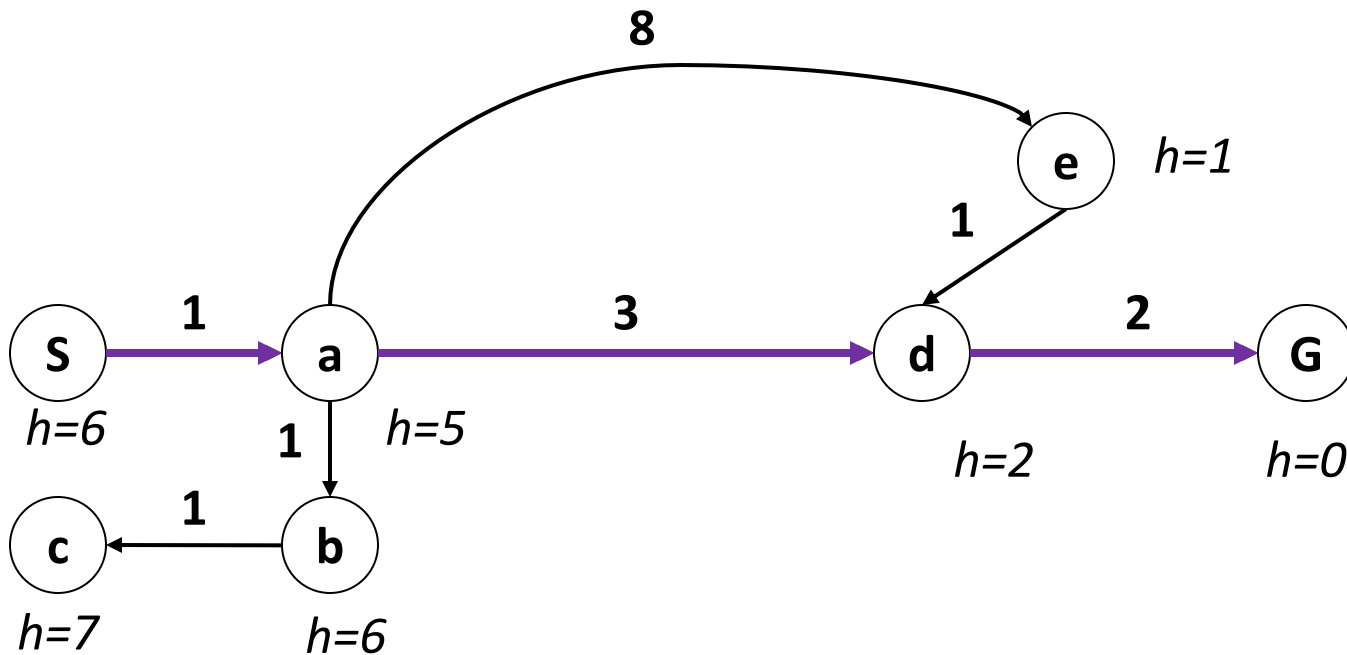


Example: Teg Grenager

Combining UCS and Greedy

Uniform-cost orders by path cost, or *backward cost* $g(n)$

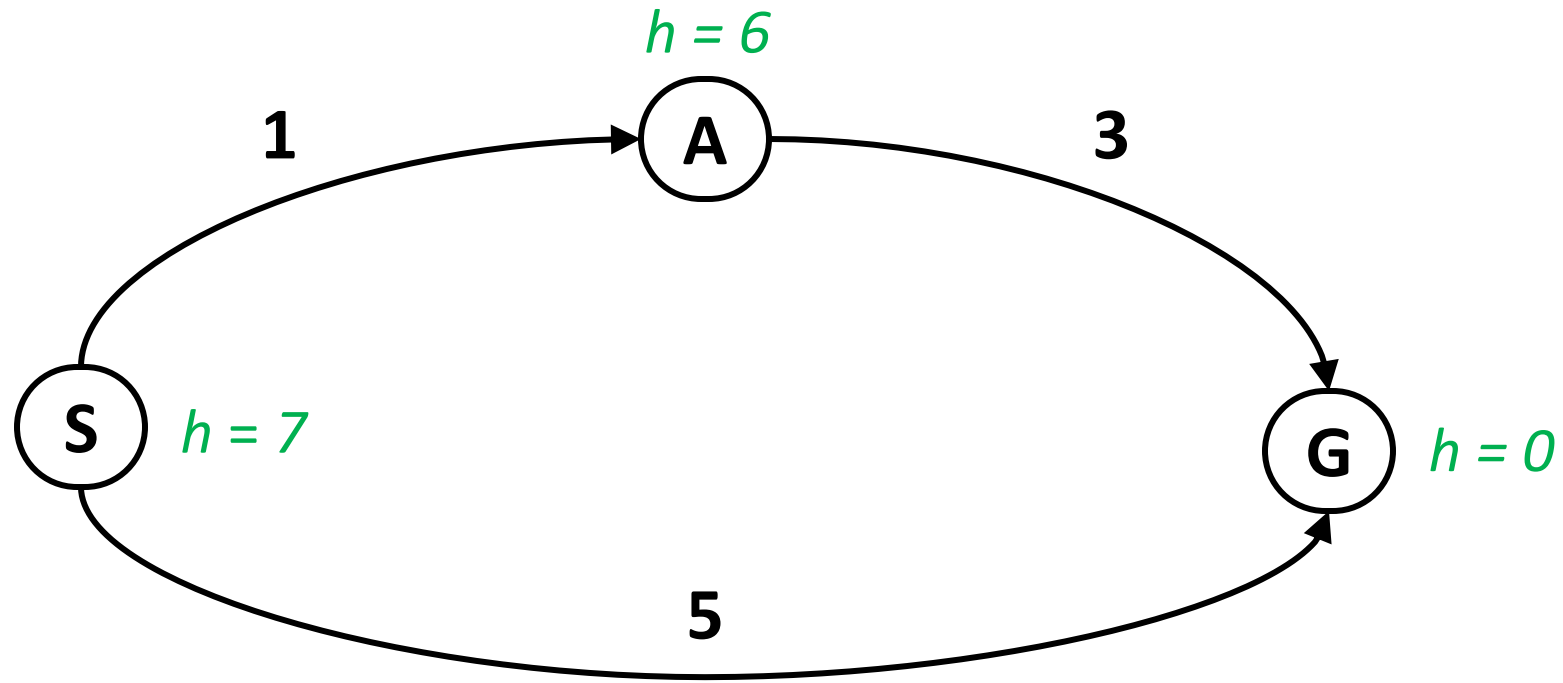
Greedy orders by goal proximity, or *forward cost* $h(n)$



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

Example: Teg Grenager

Is A* Optimal?



What went wrong?

Estimated good goal cost > **Actual** future cost!

We need estimates to be less than actual costs!

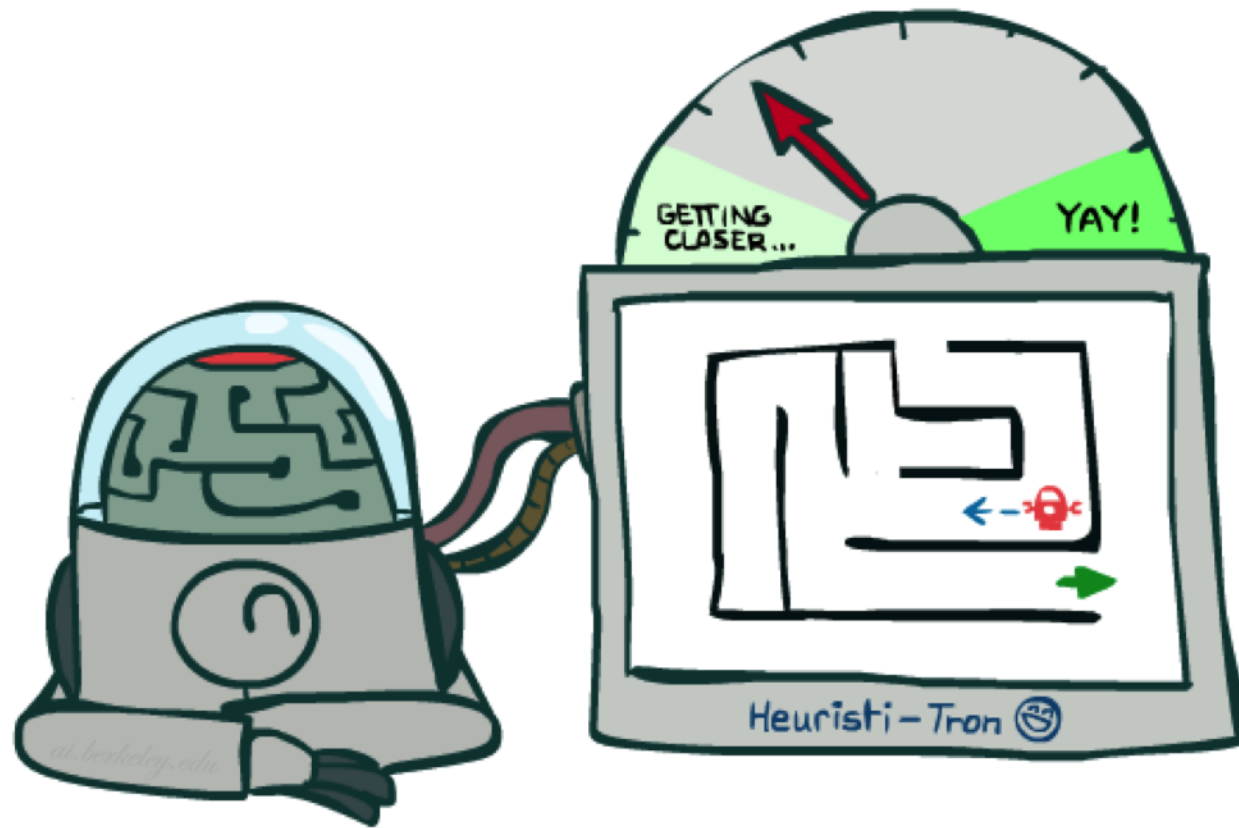
The Price is Wrong...

Closest bid without going over...



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

Admissible Heuristics



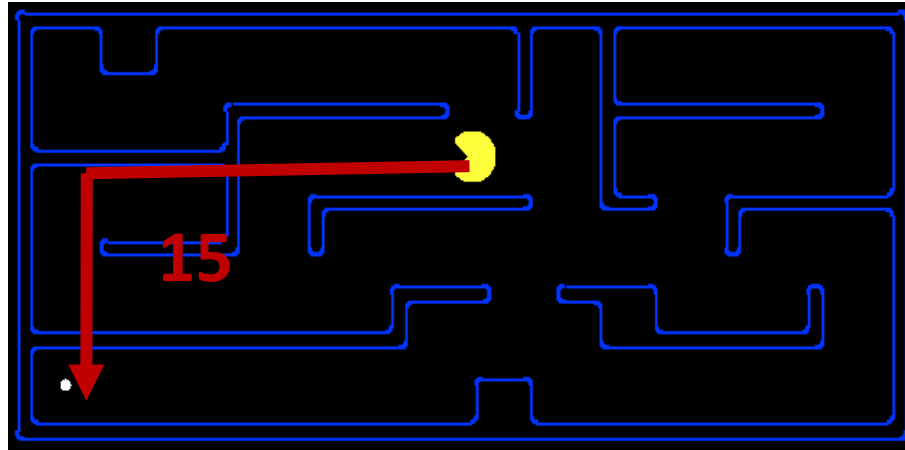
Admissible Heuristics

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

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

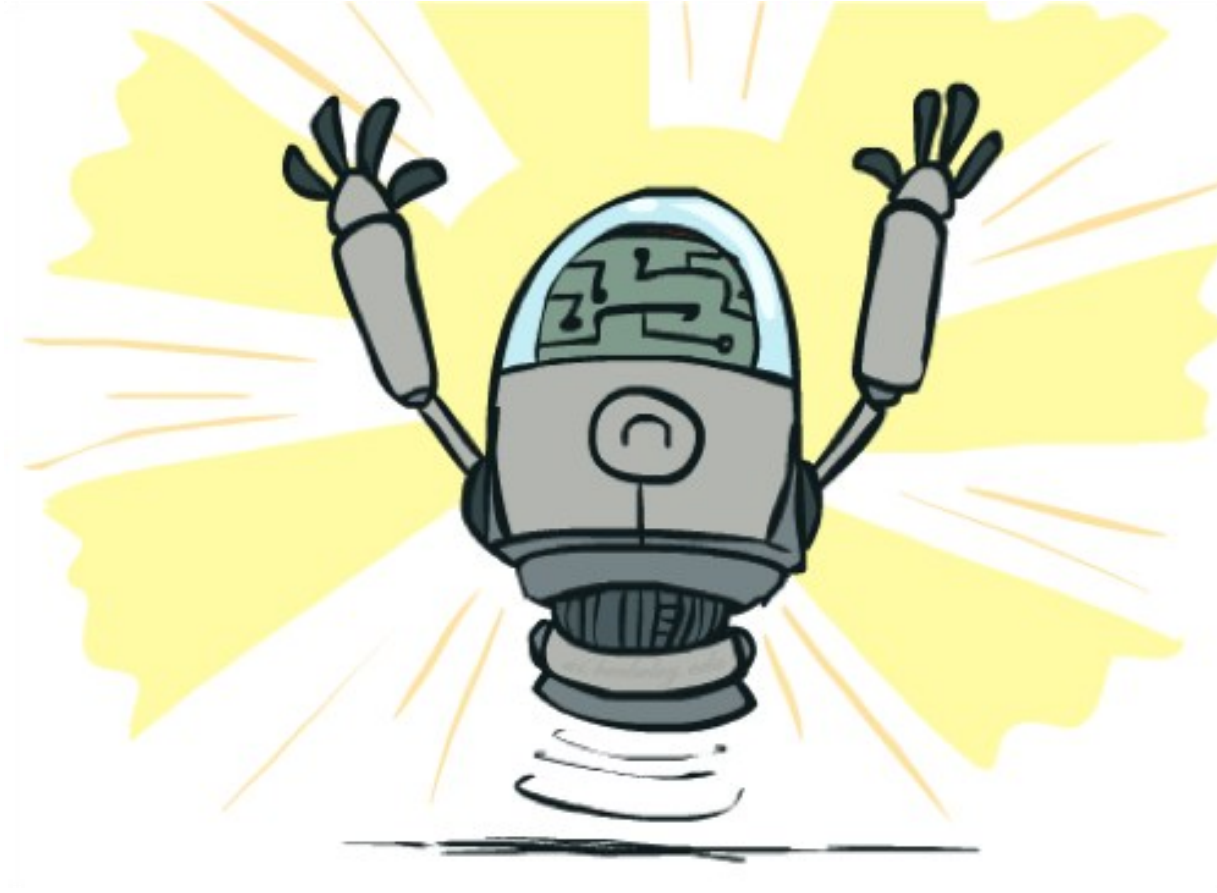
where $h^*(n)$ is the true cost to a nearest goal

Example:



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

Optimality of A* Tree Search



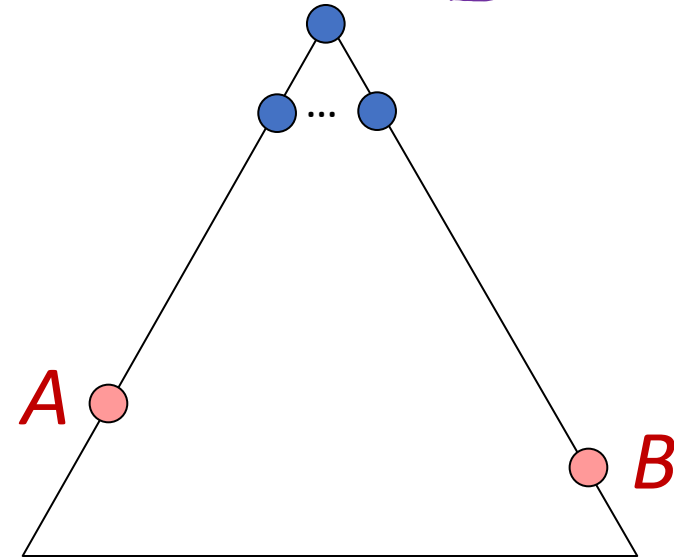
Optimality of A* Tree Search

Assume:

A is an optimal goal node

B is a suboptimal goal node

h is admissible



Claim:

A will be chosen for exploration (popped off the frontier) before B

Optimality of A* Tree Search: Blocking

Proof:

Imagine B is on the frontier

Some ancestor n of A is on the frontier, too
(Maybe the start state; maybe A itself!)

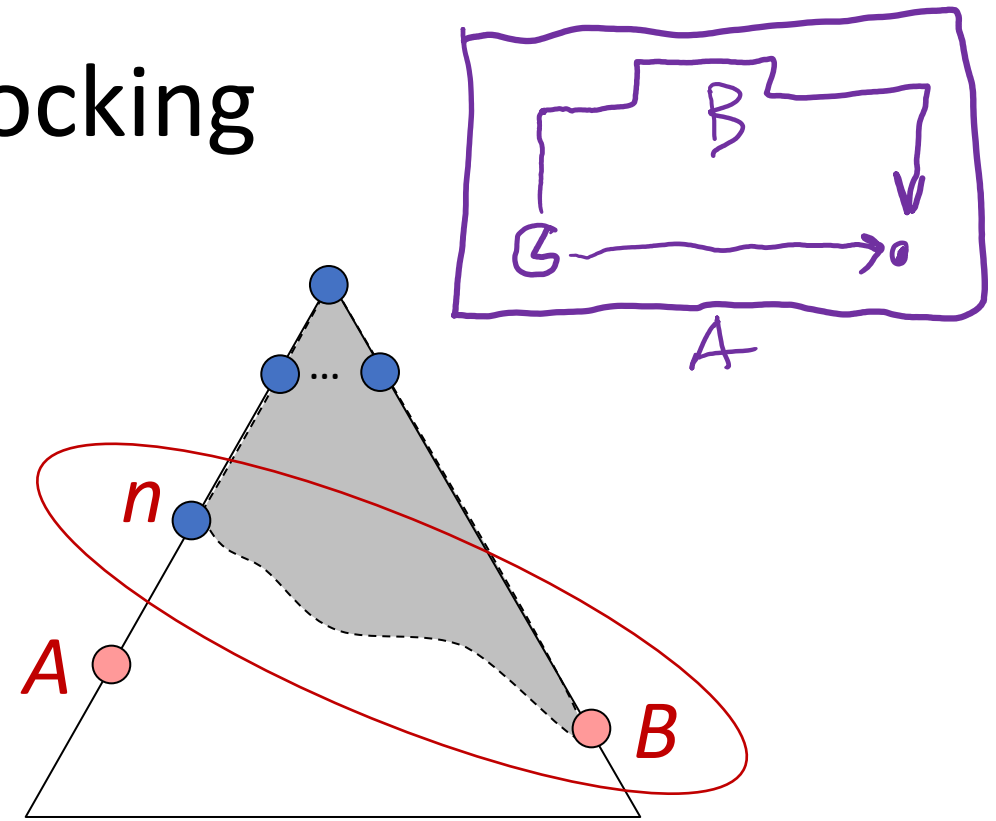
Claim: n will be explored before B

- 1.
- 2.
- 3.

All ancestors of A are explored before B

A is explored before B

A* 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 the start state; maybe A itself!)

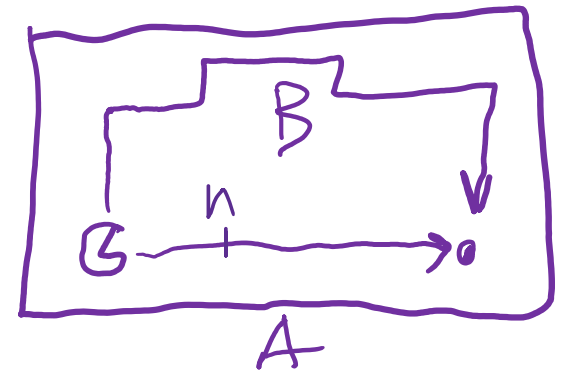
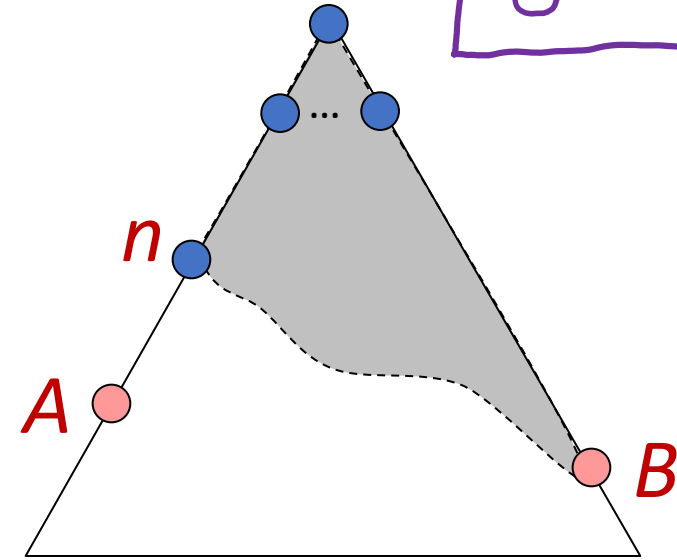
Claim: n will be explored before B

1. $f(n) \leq f(A)$ $\leftarrow$ TODO
2. $f(A) < f(B)$ $\leftarrow$ TODO
3. $f(n) < f(B)$ then n is explored before B

All ancestors of A are explored before B

A is explored before B

A* search is optimal



Optimality of A* Tree Search: Blocking

$$f(x) = g(x) + h(x)$$
$$h(x) \leq h^*(x)$$

Proof:

Imagine B is on the frontier

Some ancestor n of A is on the frontier, too
(Maybe the start state; maybe A itself!)

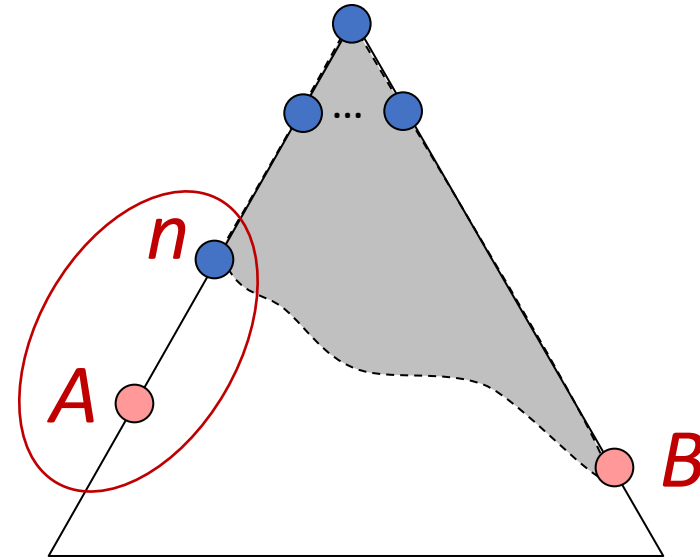
Claim: n will be explored before B

1. $f(n) \leq f(A)$
2. $f(A) < f(B)$
3. $f(n) < f(B)$ then n is explored before B

All ancestors of A are explored before A

A is explored before B

A* search is optimal



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

$$f(n) \leq g(n) + h^*(n)$$

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

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

Definition of f -cost

Admissibility of h

n on optimal path to A

$h = 0$ at a goal

Optimality of A* Tree Search: Blocking

$$f(x) = g(x) + h(x)$$
$$h(x) \leq h^*(x)$$

Proof:

Imagine B is on the frontier

Some ancestor n of A is on the frontier, too
(Maybe the start state; maybe A itself!)

Claim: n will be explored before B

1. $f(n) \leq f(A)$

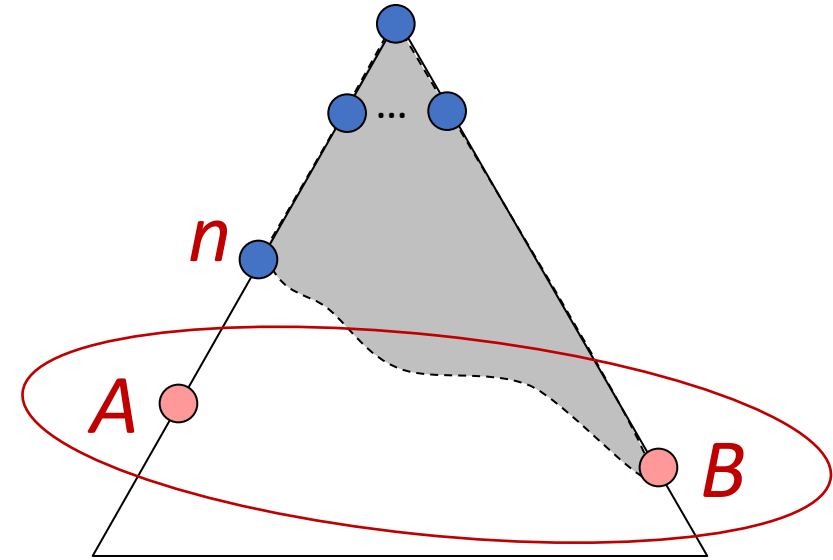
2. $f(A) < f(B)$

3. $f(n) < f(B)$ then n is

All ancestors of A are explored before

A is explored before B

A* search is optimal



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

$$f(B) = g(B)$$

$$g(A) < g(B)$$

$$f(A) < f(B)$$

Def. of $f(x)$

$h = 0$ at a goal

Similarly for B

Suboptimality of B

Optimality of A* Tree Search: Blocking

$$f(x) = g(x) + h(x)$$
$$h(x) \leq h^*(x)$$

Proof:

Imagine B is on the frontier

Some ancestor n of A is on the frontier, too
(Maybe the start state; maybe A itself!)

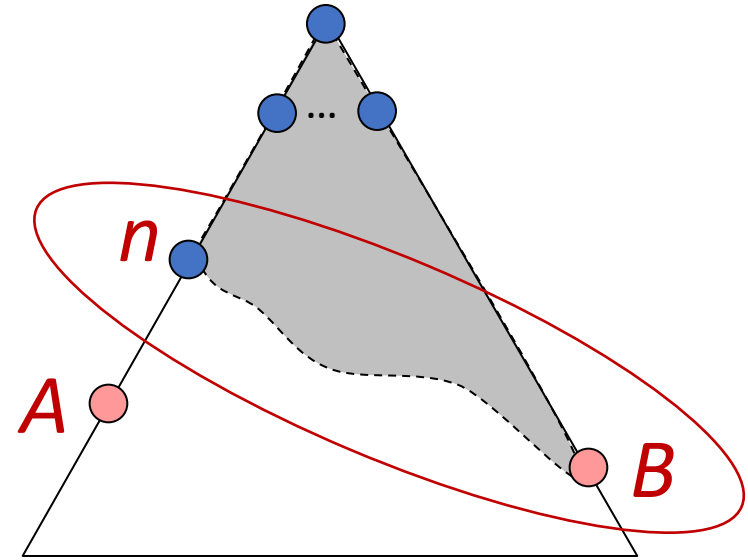
Claim: n will be explored before B

1. $f(n)$ is less than or equal to $f(A)$
2. $f(A)$ is less than $f(B)$
3. $f(n) < f(B)$ then n is explored before B

All ancestors of A are explored before B

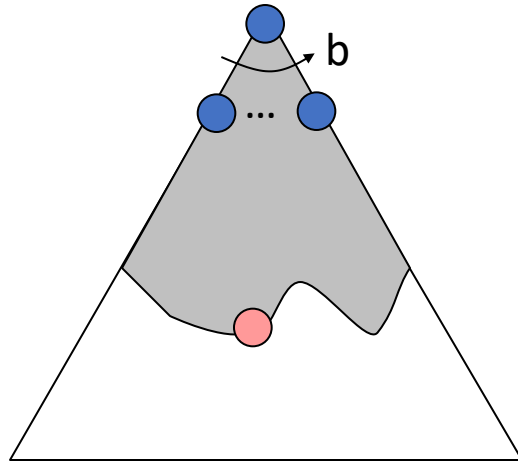
A is explored before B

A* search is optimal

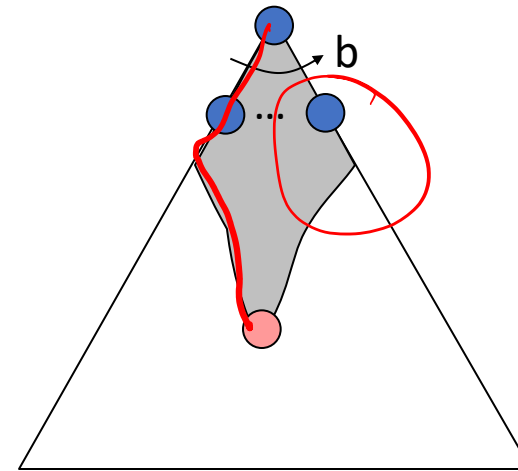


UCS vs A* Contours

Uniform-Cost

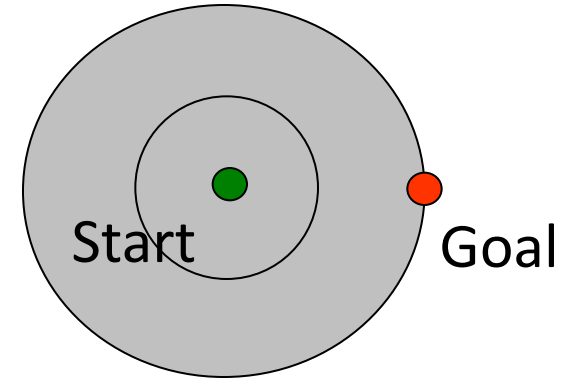


A*

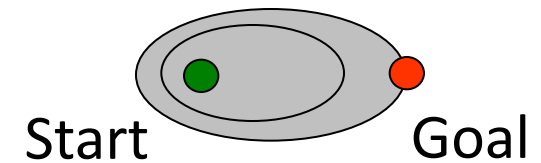


UCS vs A* Contours

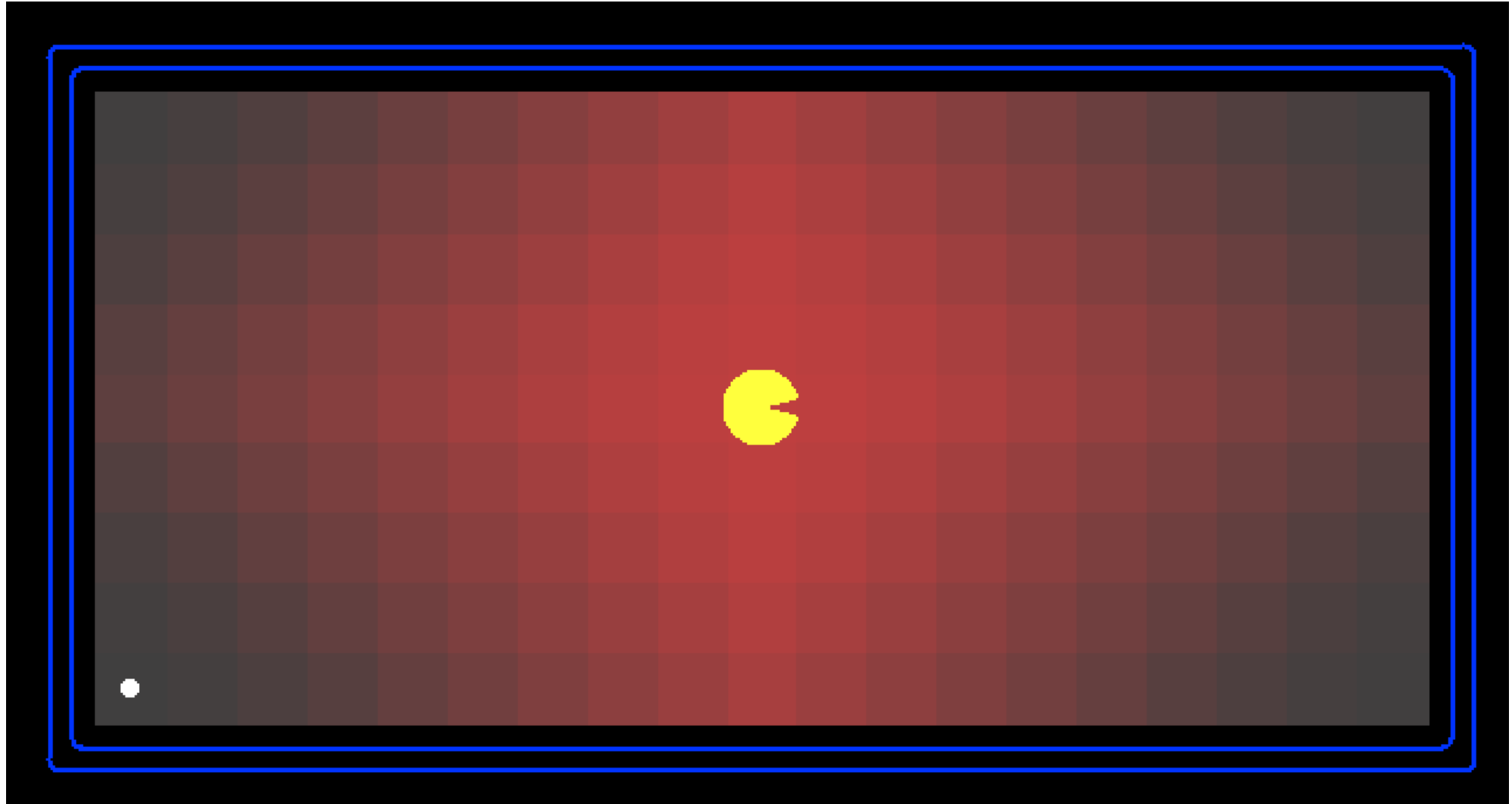
Uniform-cost expands equally in all “directions”



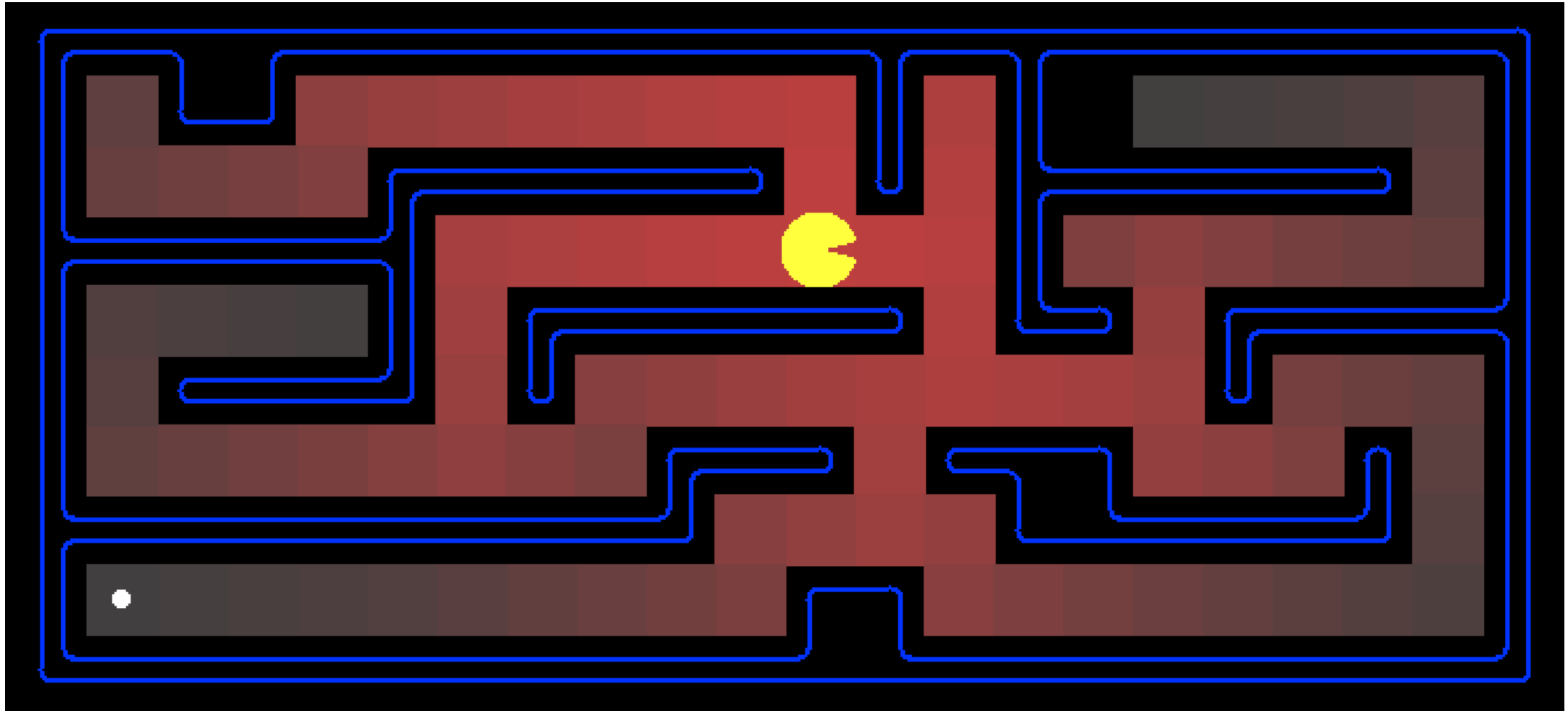
A* expands mainly toward the goal, but does hedge its bets to ensure optimality



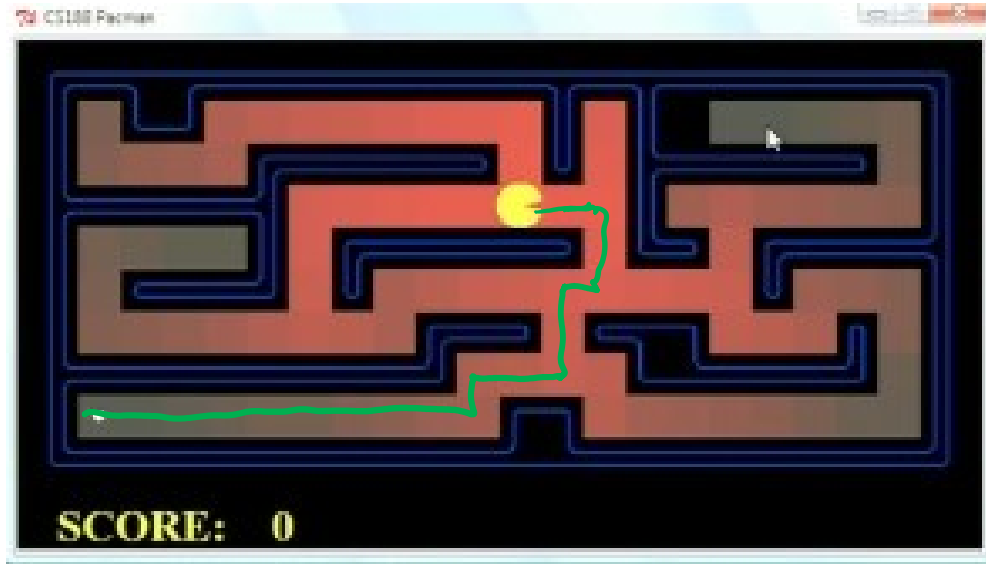
Demo Contours UCS Empty



Demo Contours UCS Pacman Small Maze

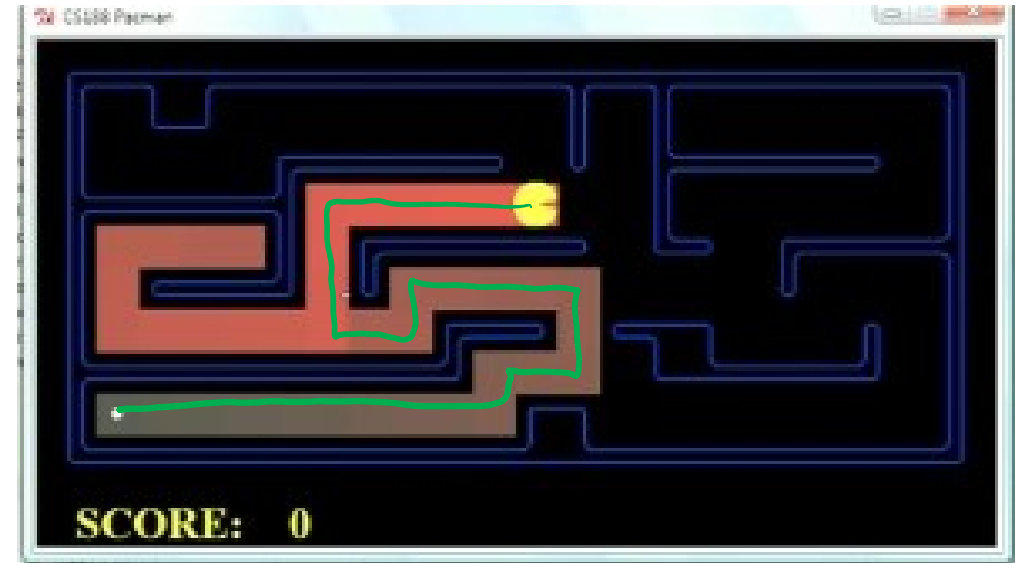


Comparison

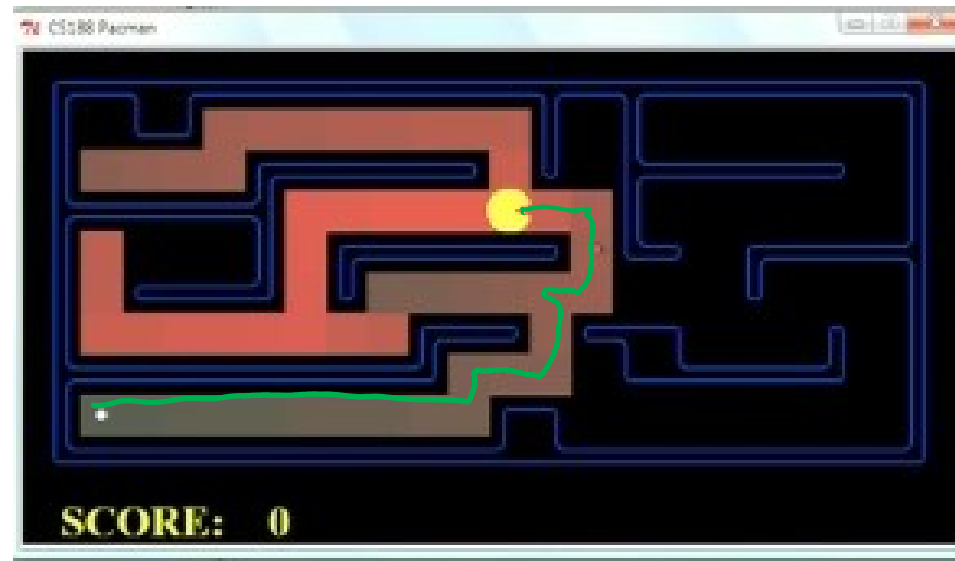


UCS

A*



Greedy



A* Search Algorithms

A* Tree Search

- Same **tree search** algorithm as before but with a **frontier** that is a **priority queue** using priority $f(n) = g(n) + h(n)$

A* Graph Search

- Same **UCS graph search** algorithm but with a **frontier** that is a **priority queue** using priority $f(n) = g(n) + h(n)$

function UNIFORM-COST-SEARCH(**problem**) **returns** a solution, or failure

initialize the **explored set** to be empty

initialize the **frontier** as a priority queue using $g(n)$ as the priority

add initial state of **problem** to **frontier** with priority $g(S) = 0$

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 $g(n)$ **then**

 replace that **frontier** node with **child**

function A-STAR-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

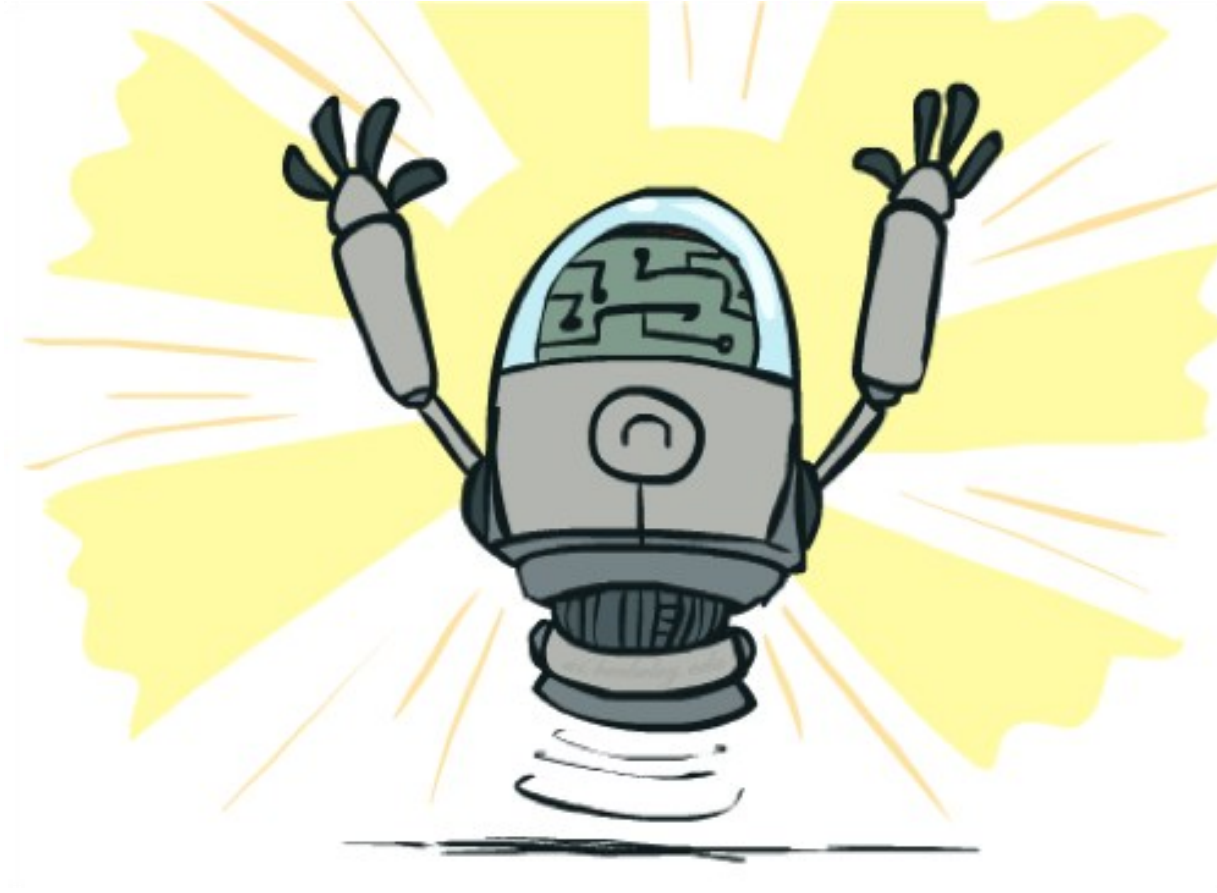
if the **child** state is not already in the **frontier** or **explored set** **then**

 add **child** to the **frontier**

else if the **child** is already in the **frontier** with higher $f(n)$ **then**

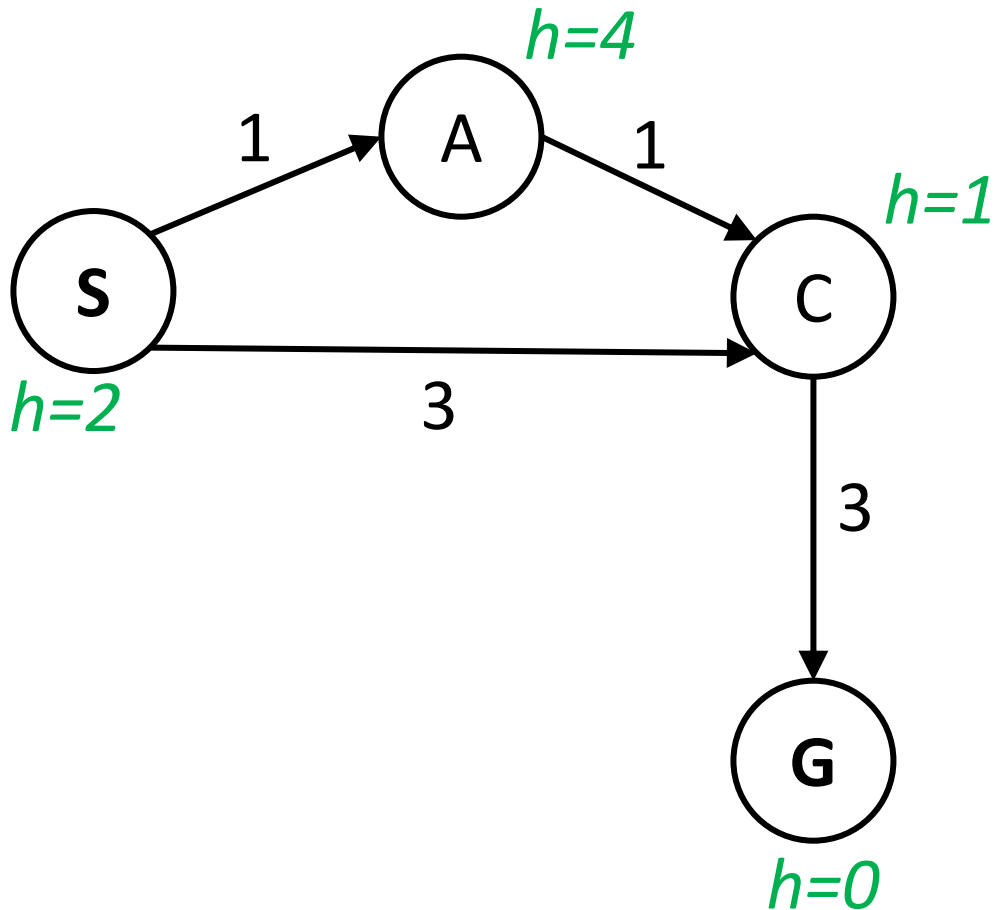
 replace that **frontier** node with **child**

Optimality of A* Graph Search

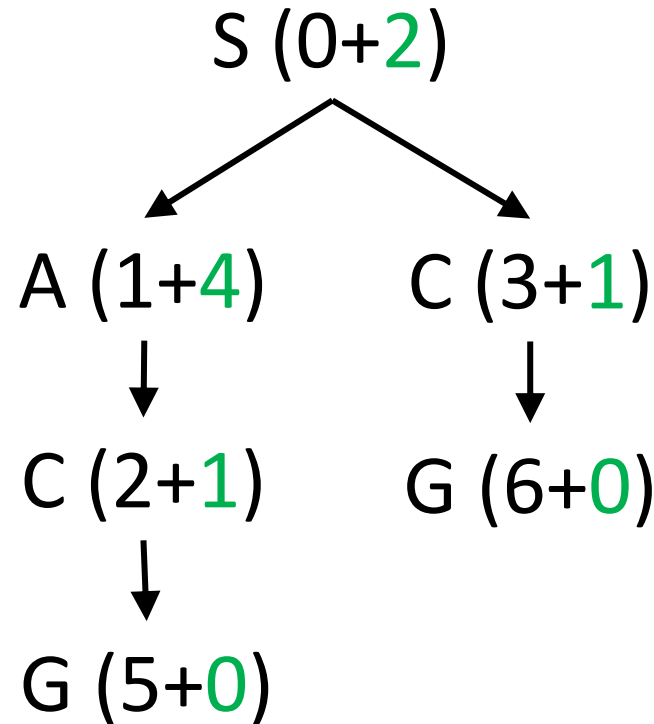


A* Tree Search

State space graph



Search tree



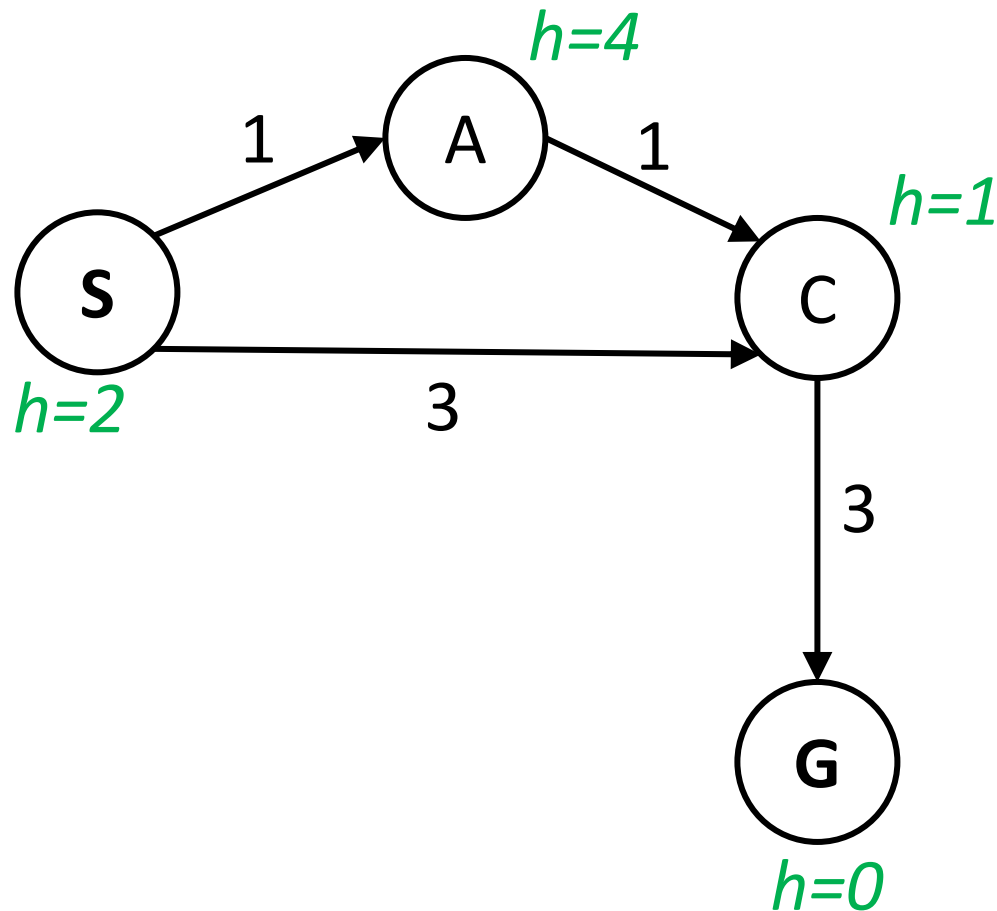
Frontier

~~S (0+2)~~
~~S-A (1+4)~~
~~S-C (3+1)~~
S-C-G (6+0)
~~S-A-C (2+1)~~
~~S-A-C-G (5+0)~~

Result: S-A-C-G cost 5
Correct!

Poll 1: A* Graph Search

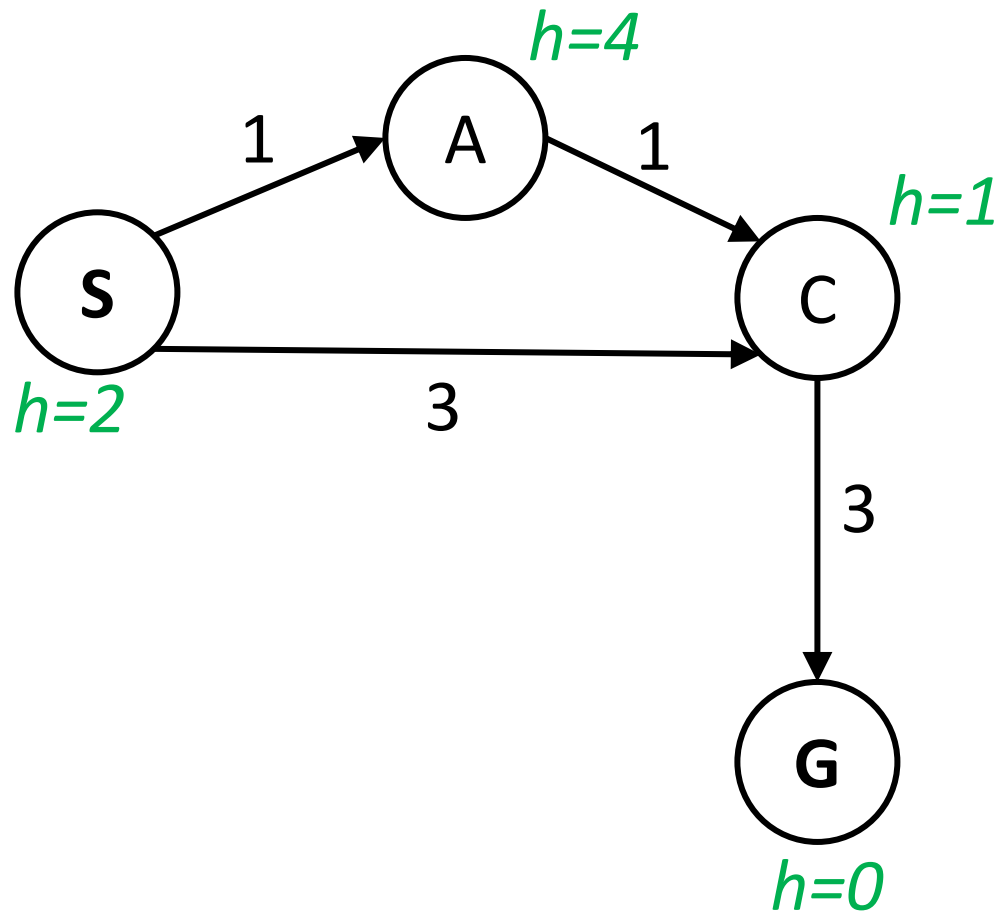
What paths does A* graph search consider during its search?
(What does your work for the frontier look like?)



- 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

Poll 1: A* Graph Search

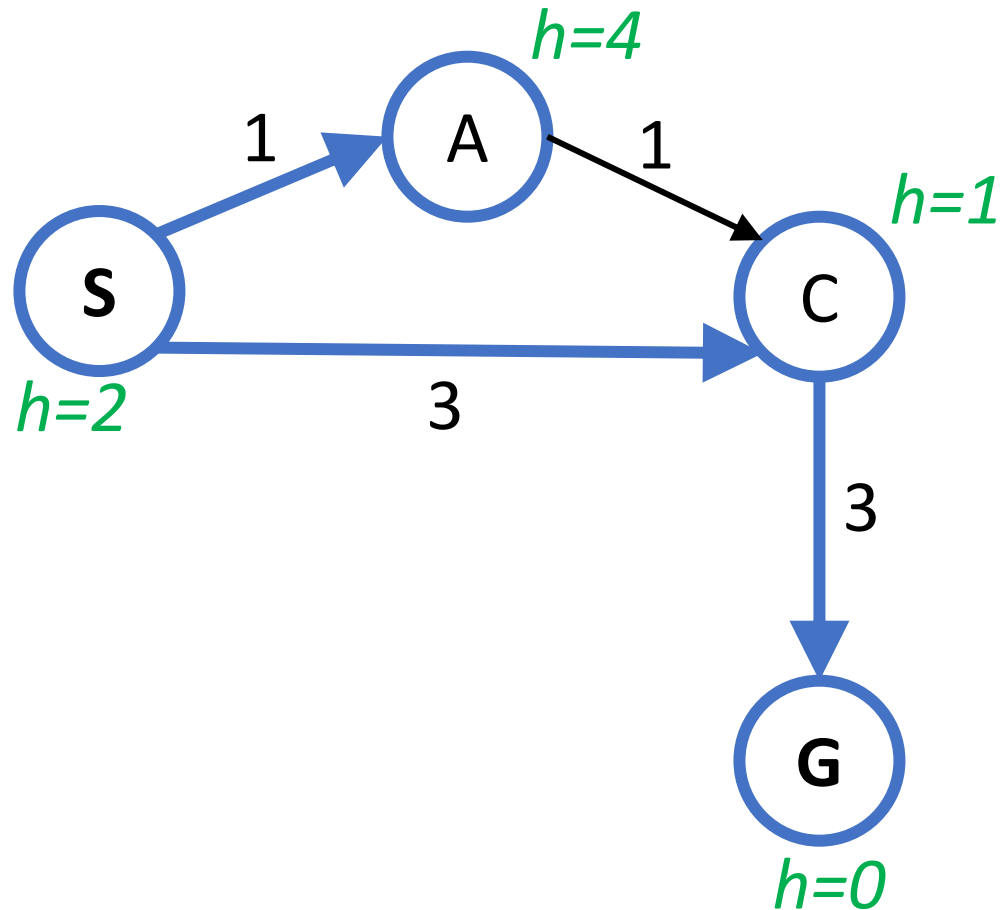
What paths does A* graph search consider during its search?
(What does your work for the frontier look like?)



- 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

A* Graph Search Gone Wrong?

State space graph



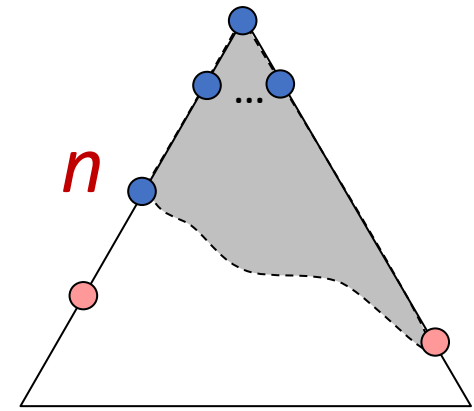
~~S: $0+2 = 2$~~

S-A: $1+4 = 5$

~~S-C: $2+1 = 3$~~

S-C-G: $6+0 = 6$

C (specifically S-C) is no longer on the frontier, so it can't be replaced with the better S-A-C option



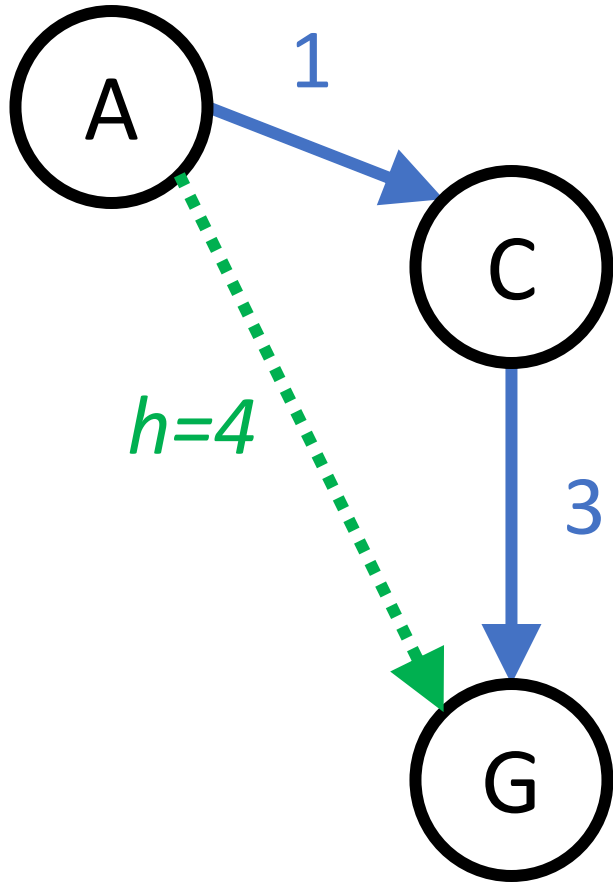
Admissibility of Heuristics

Main idea: Estimated heuristic values $\leq$ actual costs

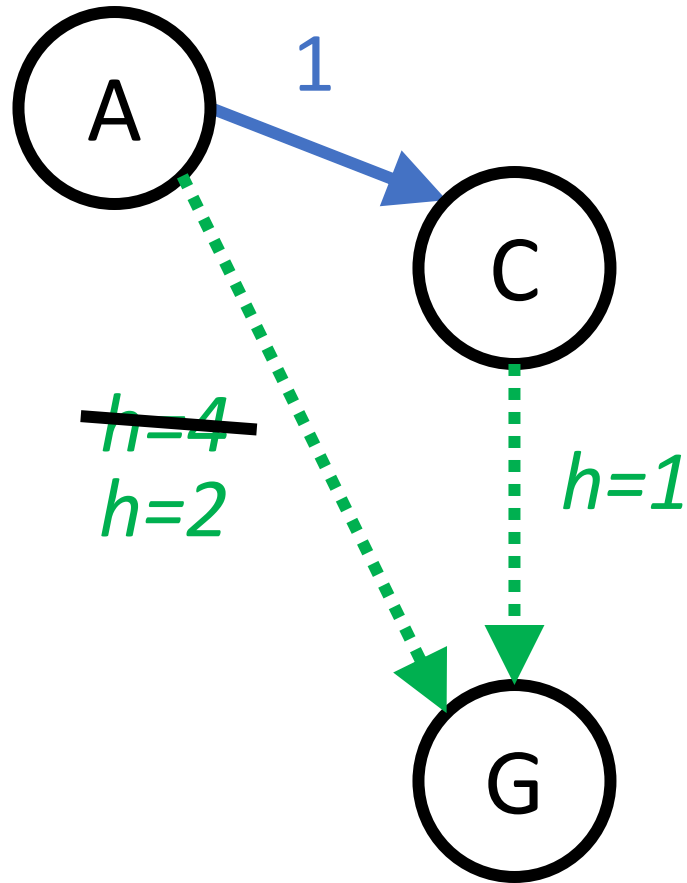
■ Admissibility:

$0 \leq \text{heuristic value} \leq \text{actual cost to goal}$

$0 \leq h(A) \leq \text{actual cost from A to G}$



Consistency of Heuristics



Main idea: Estimated heuristic costs $\leq$ actual costs

- Admissibility:

heuristic cost $\leq$ actual cost to goal

$$h(A) \leq \text{actual cost from A to G}$$

- Consistency:

“heuristic step cost” $\leq$ actual cost for each step

$$h(A) - h(C) \leq \text{cost}(A \text{ to } C)$$

triangle inequality (kind of)

$$h(A) \leq \text{cost}(A \text{ to } C) + h(C)$$

Consequences of consistency:

- The f value along a path never decreases
- A* graph search is optimal

Optimality

Tree search:

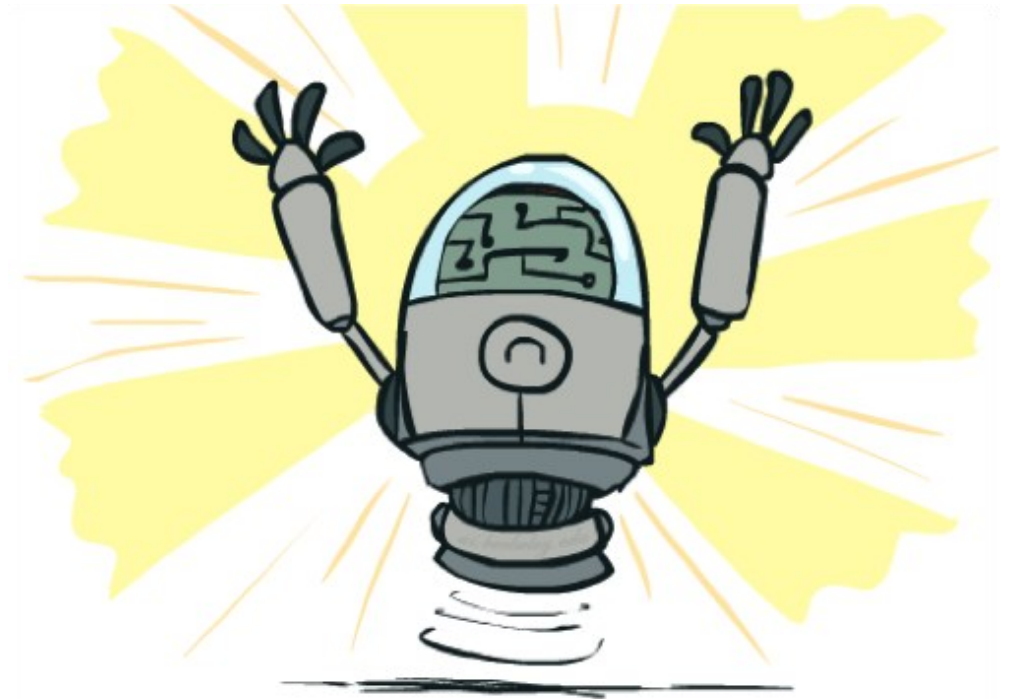
- A* is optimal if heuristic is admissible
- UCS is a special case ($h = 0$)

Graph search:

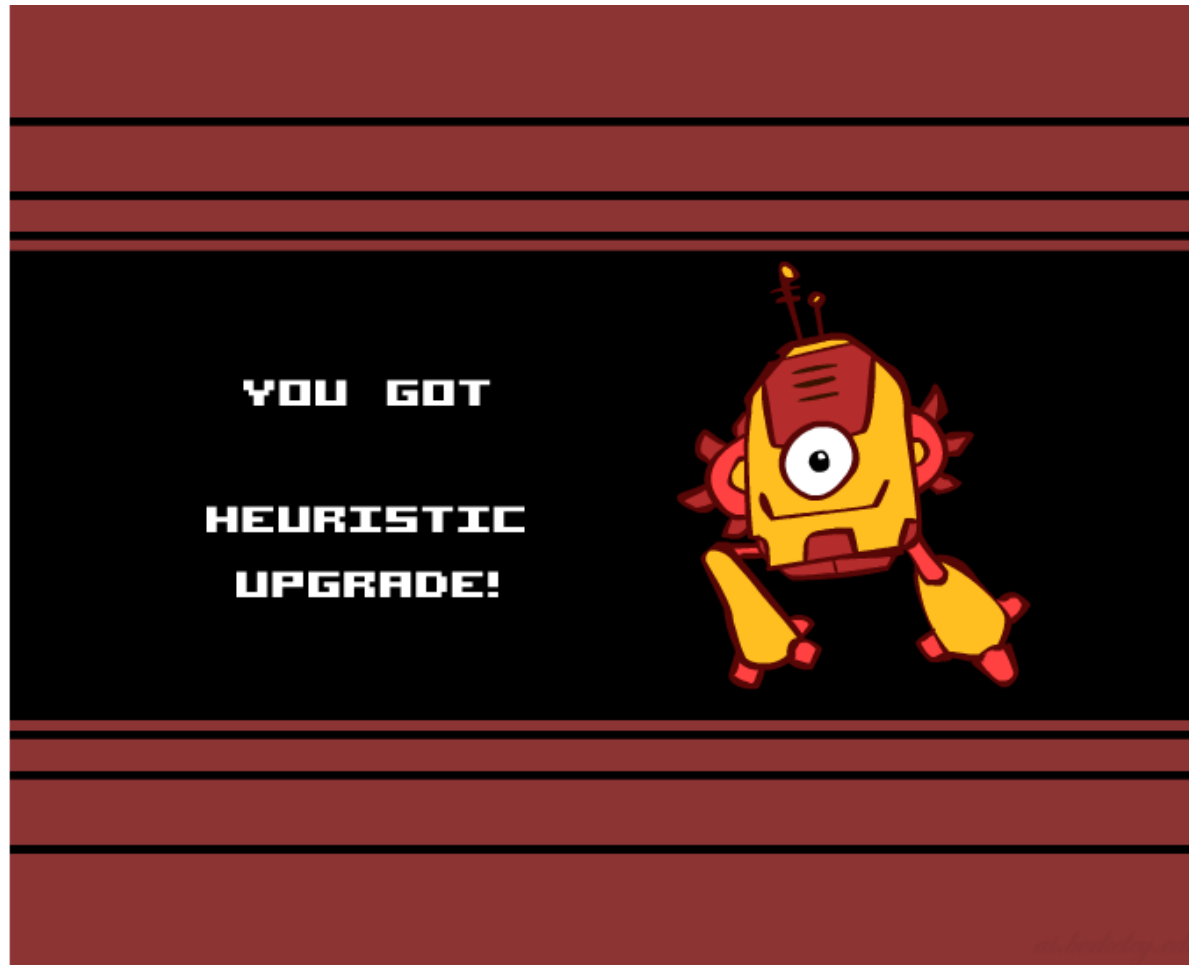
- A* optimal if heuristic is consistent
- UCS optimal ($h = 0$ is consistent)

Consistency implies admissibility

In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems



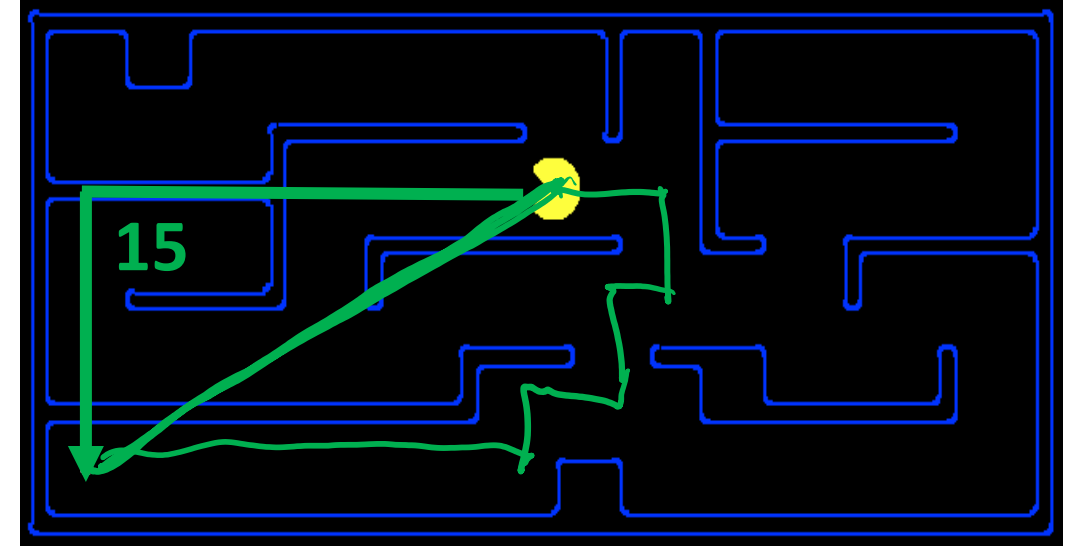
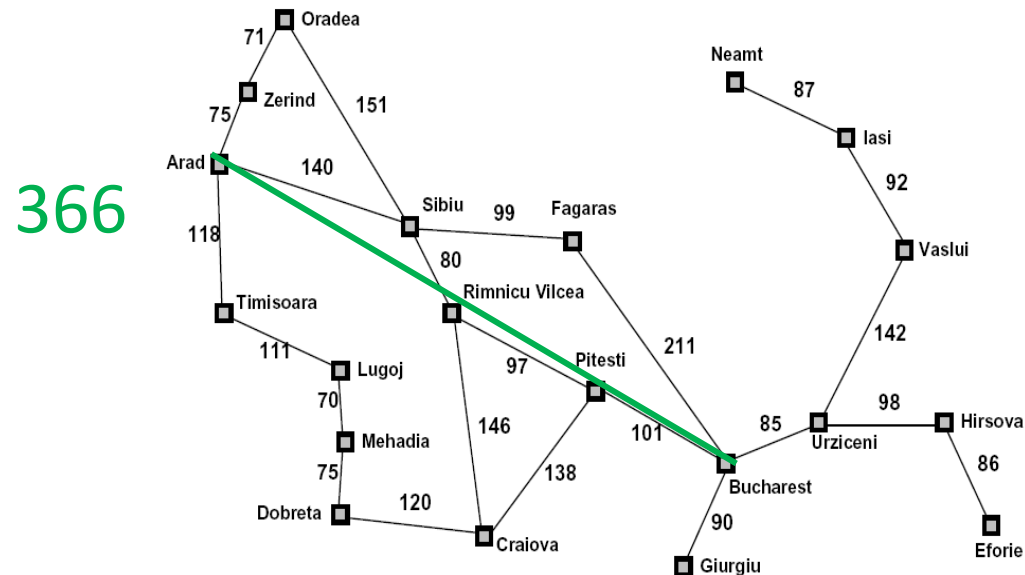
Creating Heuristics



Creating Admissible Heuristics

Most of the work in solving hard search problems optimally is in coming up with admissible heuristics

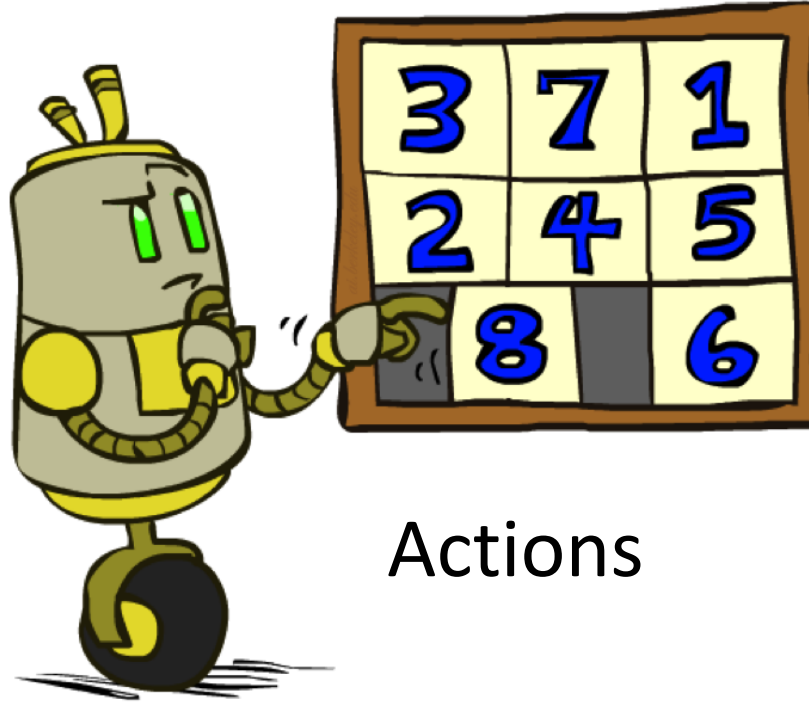
Often, admissible heuristics are solutions to *relaxed problems*, where new actions are available



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 actions from the start state?

What should the step costs be?

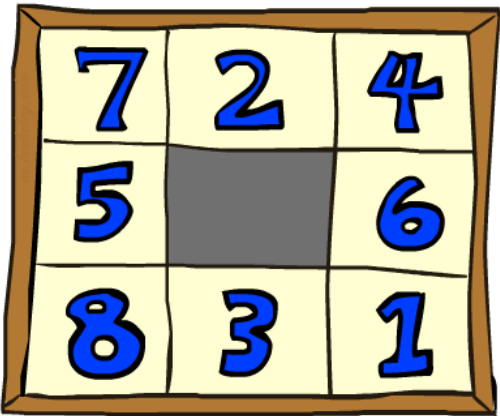
8 Puzzle I

Heuristic: Number of tiles misplaced

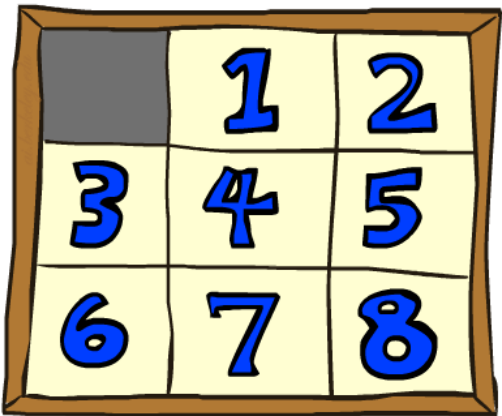
Why is it admissible?

$h(\text{start}) = 8$

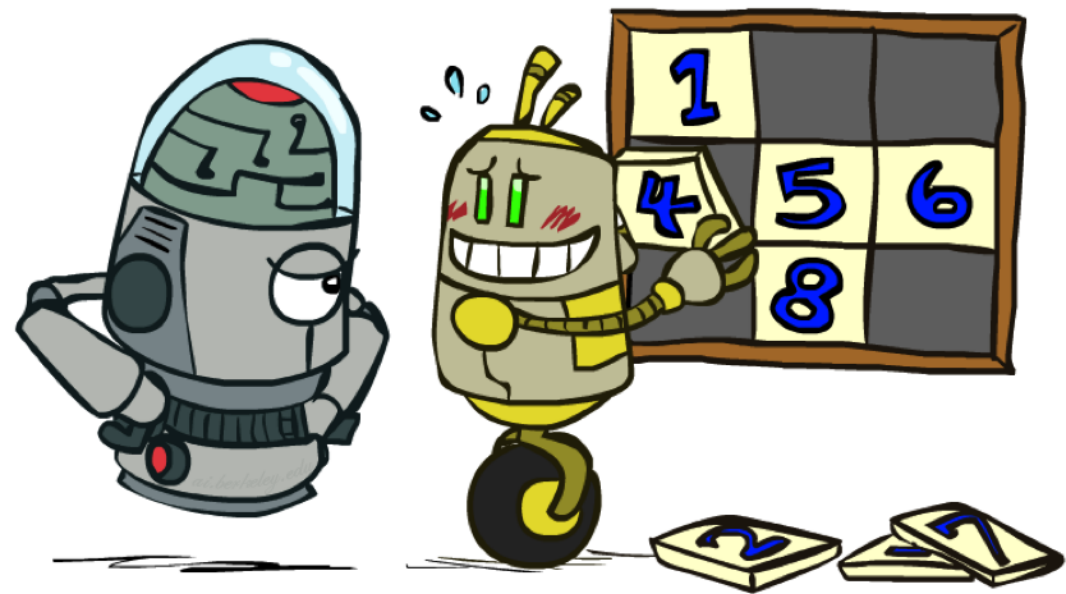
This is a *relaxed-problem* heuristic



Start State



Goal State

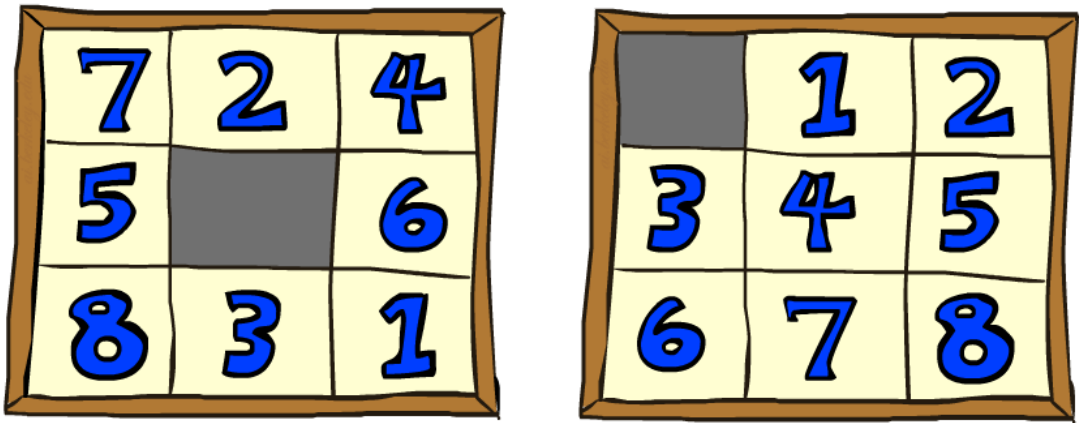


Average nodes expanded when the optimal path has...

	...4 steps	...8 steps	...12 steps
UCS	112	6,300	3.6×10^6
A*TILES	13	39	227

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

Why is it admissible?

$$h(\text{start}) = 3 + 1 + 2 + \dots = 18$$

	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
A* _{MANHATTAN}	12	25	73

Combining heuristics

Dominance: $h_a \geq h_c$ if

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

- Roughly speaking, larger 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:

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

- Max of admissible heuristics is admissible and dominates both!

A*: Summary



A*: Summary

A* uses both backward costs and (estimates of) forward costs

A* is optimal with admissible / consistent heuristics

Heuristic design is key: often use relaxed problems

