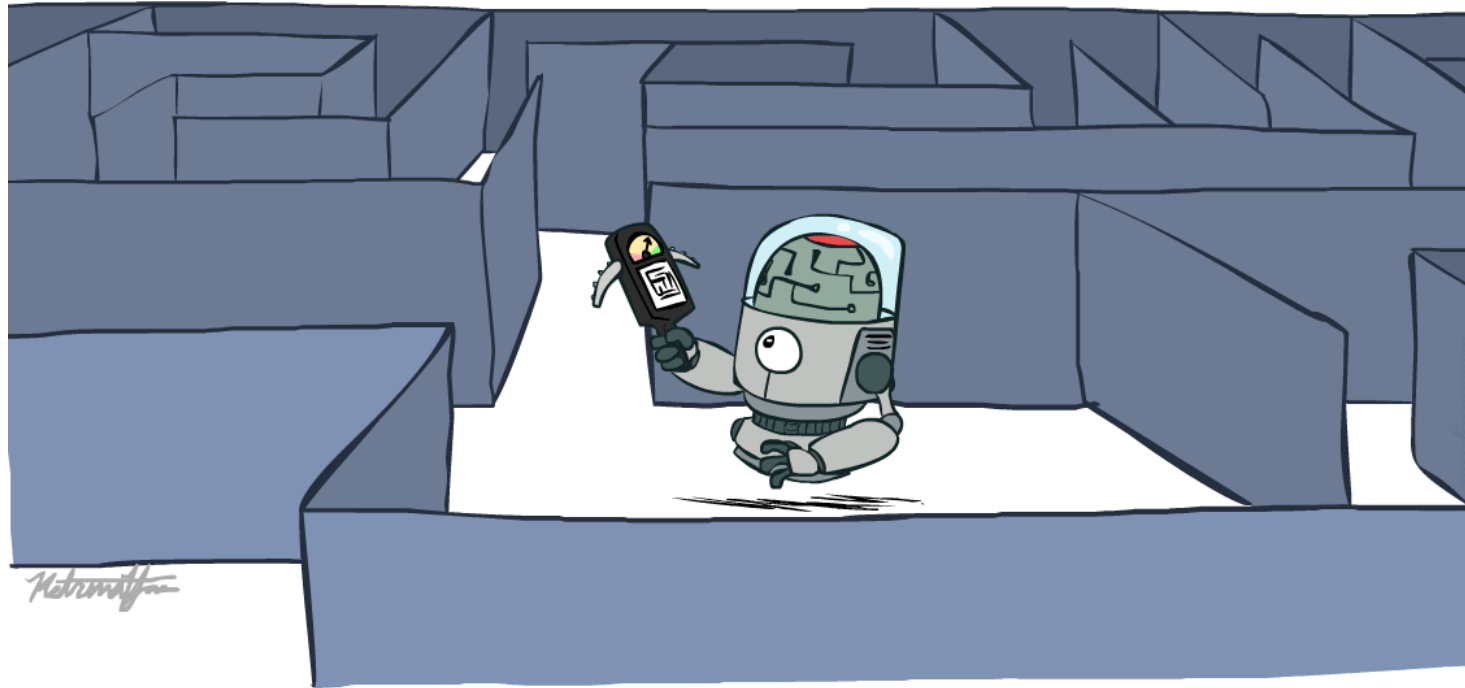


AI: Representation and Problem Solving

Informed Search



Instructors: Pat Virtue & Fei Fang

Slide credits: CMU AI, <http://ai.berkeley.edu>

function TREE_SEARCH(problem) returns a solution, or failure

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

for each resulting child from node

add child to the frontier

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**

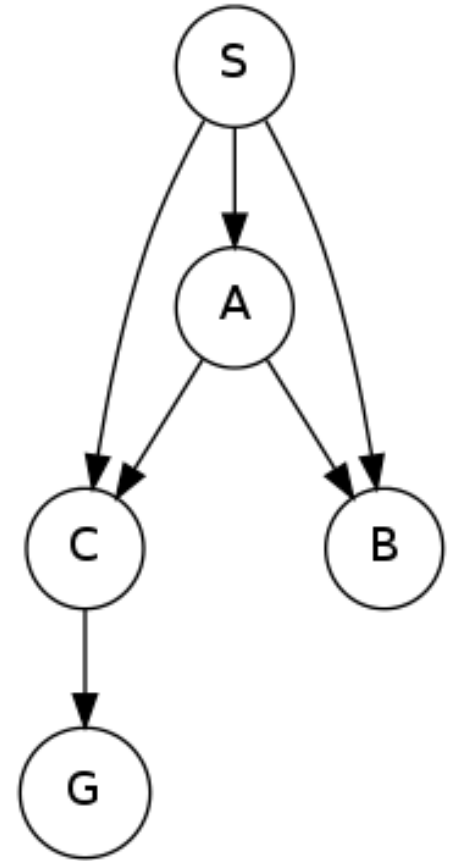
return the corresponding solution

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-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**

Recall: Breadth-First Search (BFS) Properties

What nodes does BFS expand?

- Processes all nodes above shallowest solution
- Let depth of shallowest solution be s
- Search takes time $O(b^s)$

How much space does the frontier take?

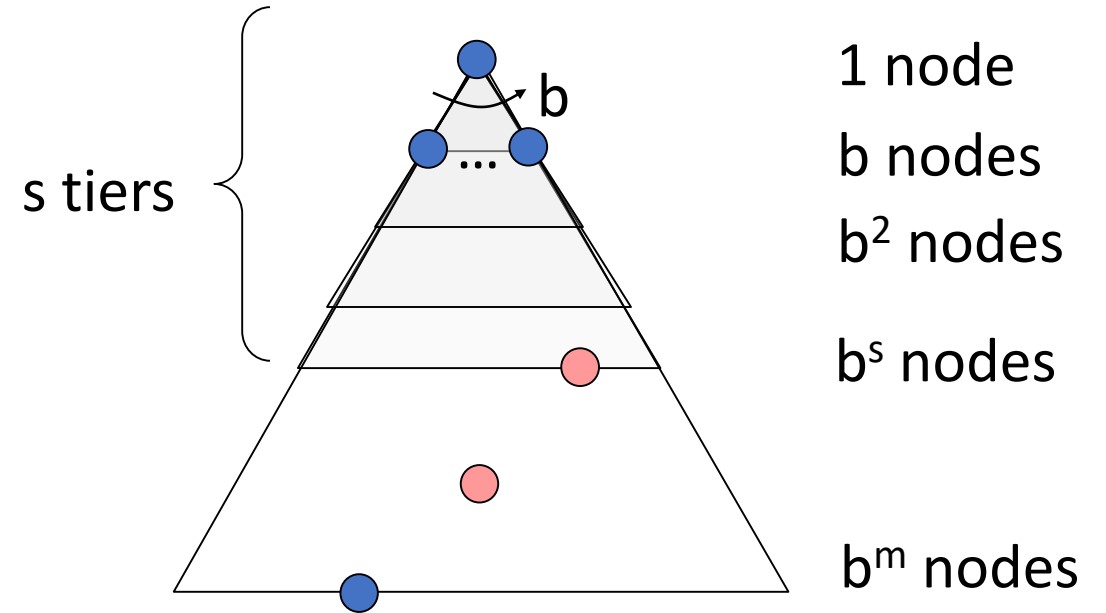
- Has roughly the last tier, so $O(b^s)$

Is it complete?

- s must be finite if a solution exists, so yes!

Is it optimal?

- Only if costs are all the same (more on costs later)



Size/cost of Search Trees

See Piazza post:

<https://piazza.com/class/jyuj6a88afh5u7?cid=30>

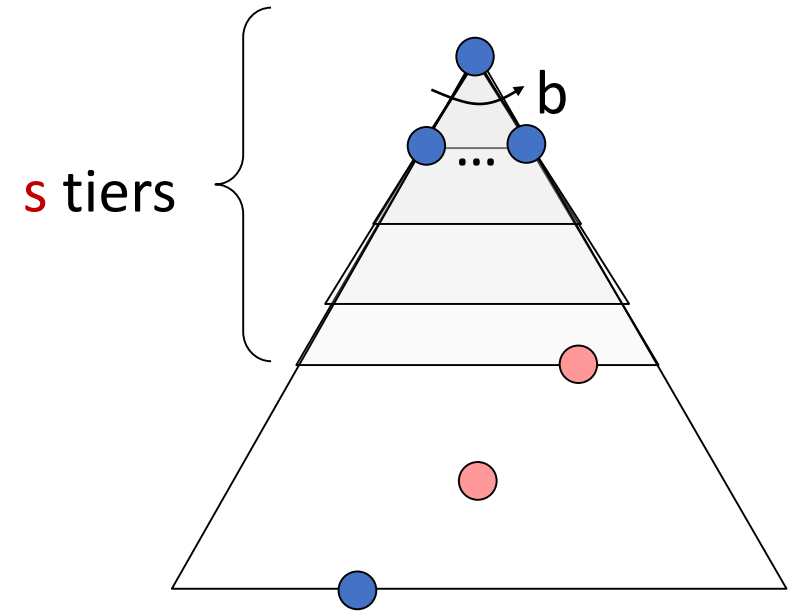
Recall: Breadth-First Search (BFS) Properties

What nodes does BFS expand?

- Processes all nodes above shallowest solution
- Let depth of shallowest solution be s
- Search takes time $O(b^s)$

How much space does the frontier take?

- Has roughly the last tier, so $O(b^s)$



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 step costs are at least ϵ , then the “effective depth” is roughly C^*/ϵ
- Takes time $O(b^{C^*/\epsilon})$ (exponential in effective depth)

How much space does the frontier take?

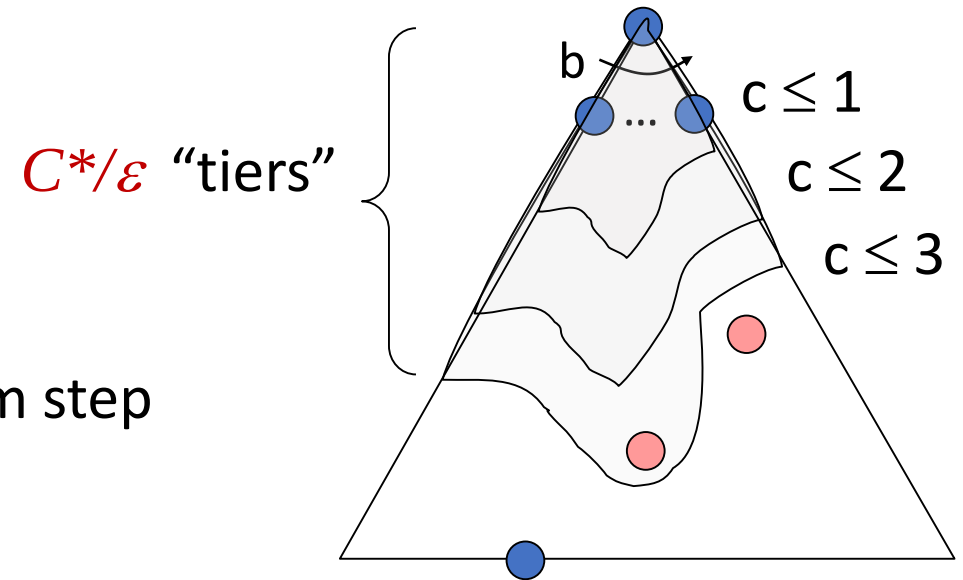
- Has roughly the last tier, so $O(b^{C^*/\epsilon})$

Is it complete?

- Assuming best solution has a finite cost and minimum step cost is positive, yes!

Is it optimal?

- Yes! (Proof via A^*)



Uniform Cost Issues

Strategy:

- Explore (expand) the lowest path cost on frontier

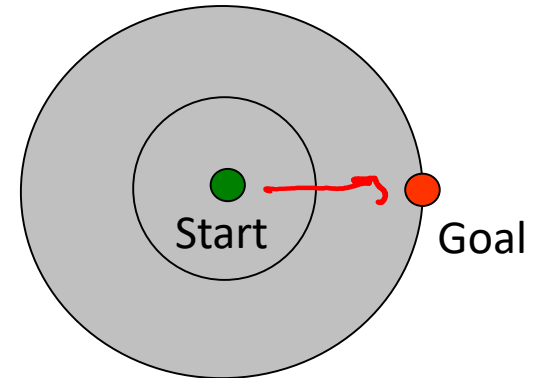
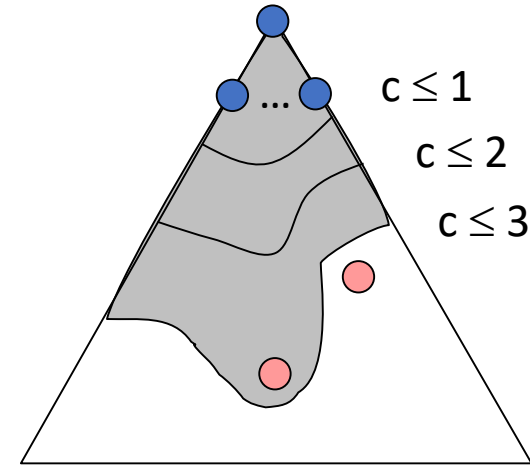
The good:

- UCS is complete and optimal!

The bad:

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

We'll fix that today!



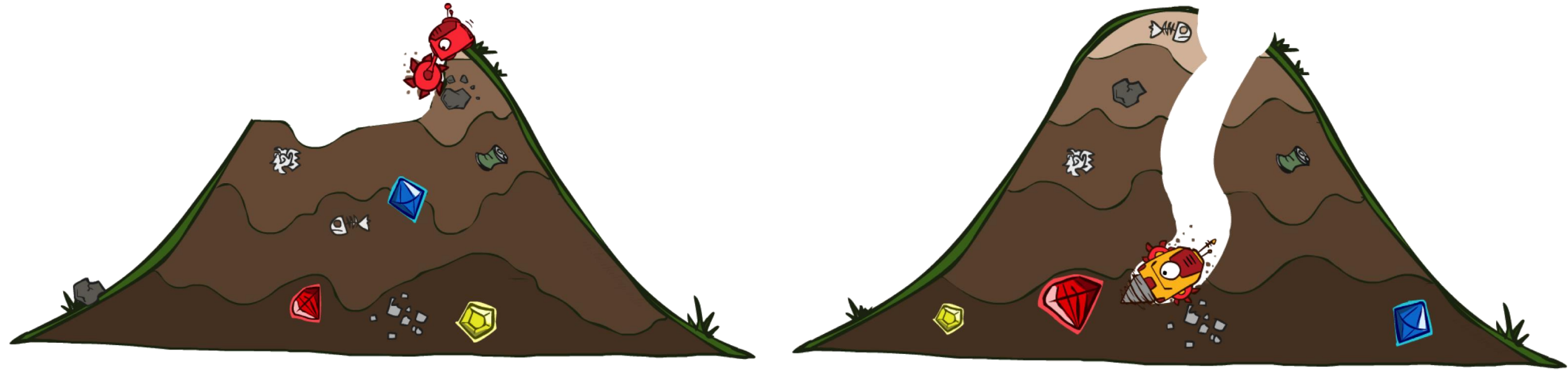
[Demo: contours UCS empty (L3D1)]

[Demo: contours UCS pacman small maze (L3D3)]

Demo Contours UCS Empty

Demo Contours UCS Pacman Small Maze

Uninformed vs Informed Search



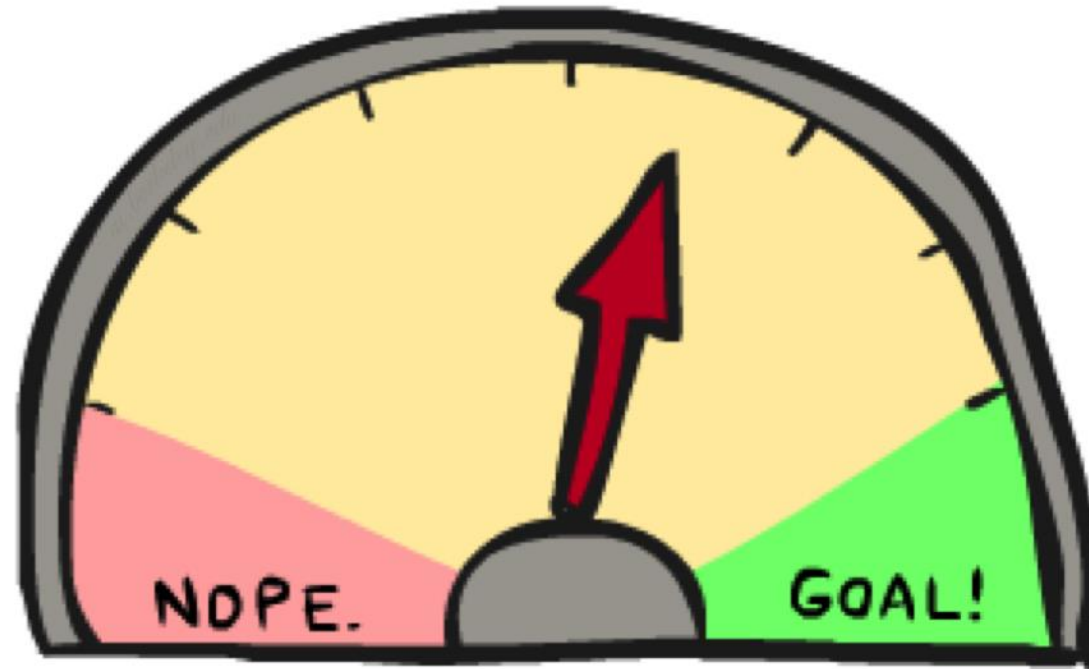
Today

Informed Search

- Heuristics
- Greedy Search
- A* Search



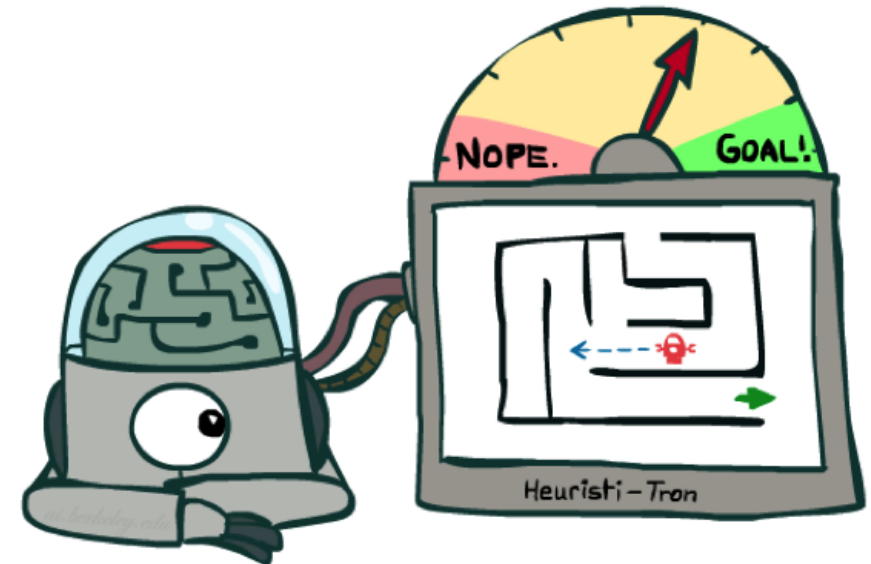
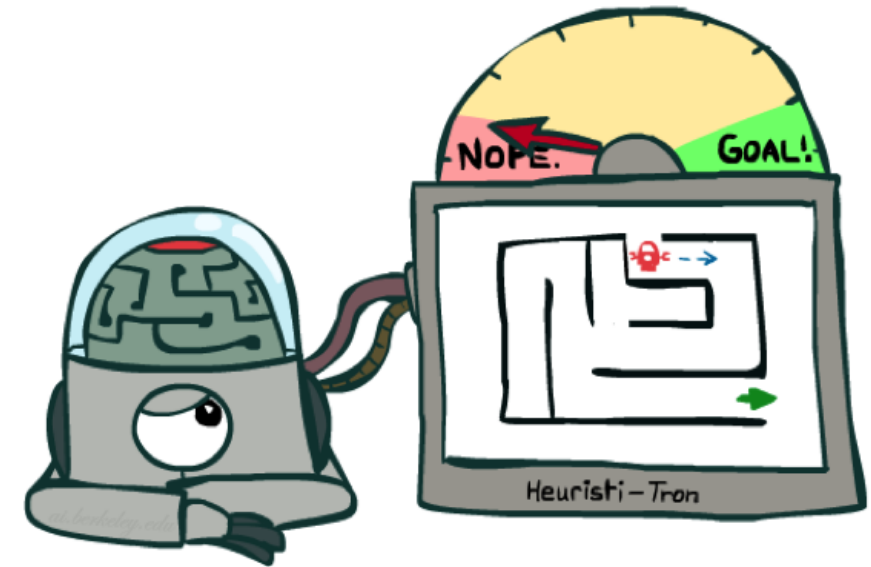
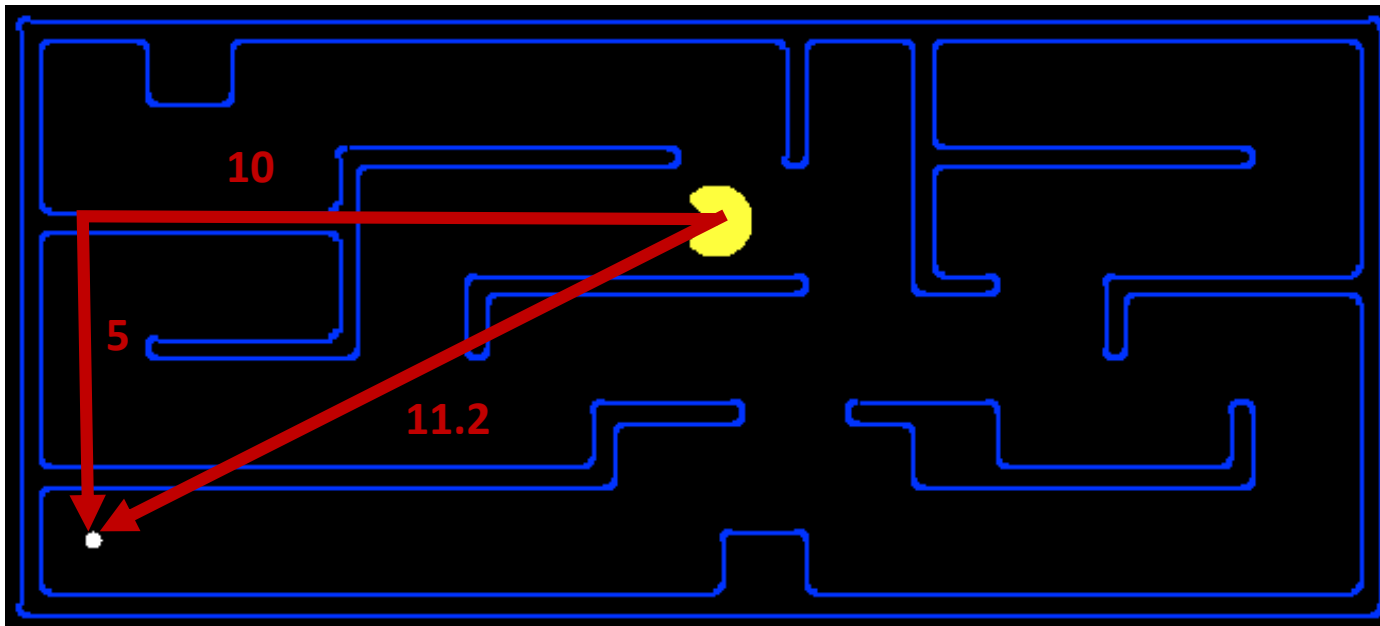
Informed Search



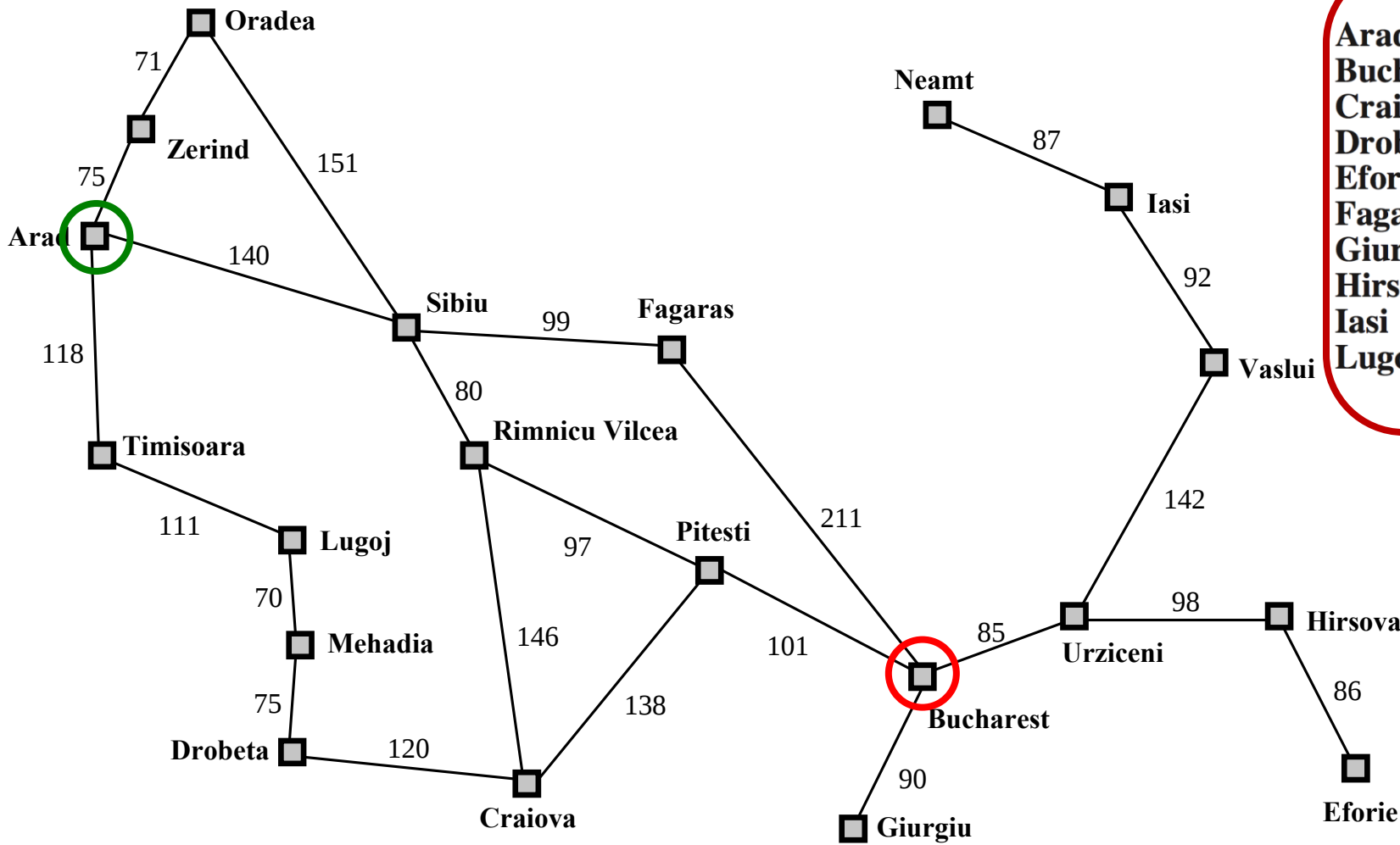
Search Heuristics

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



Example: Euclidean distance to Bucharest

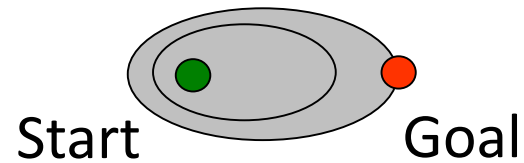


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

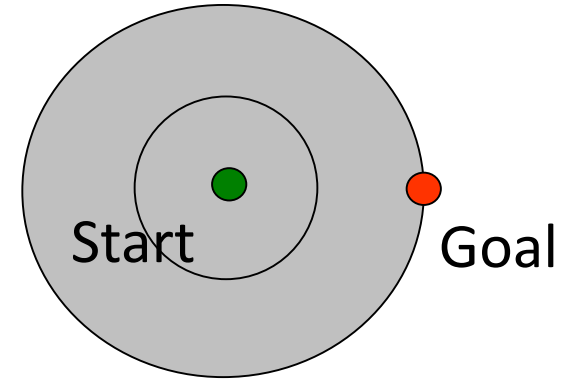
$h(\text{state}) \rightarrow \text{value}$

Effect of heuristics

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



Informed



Uninformed

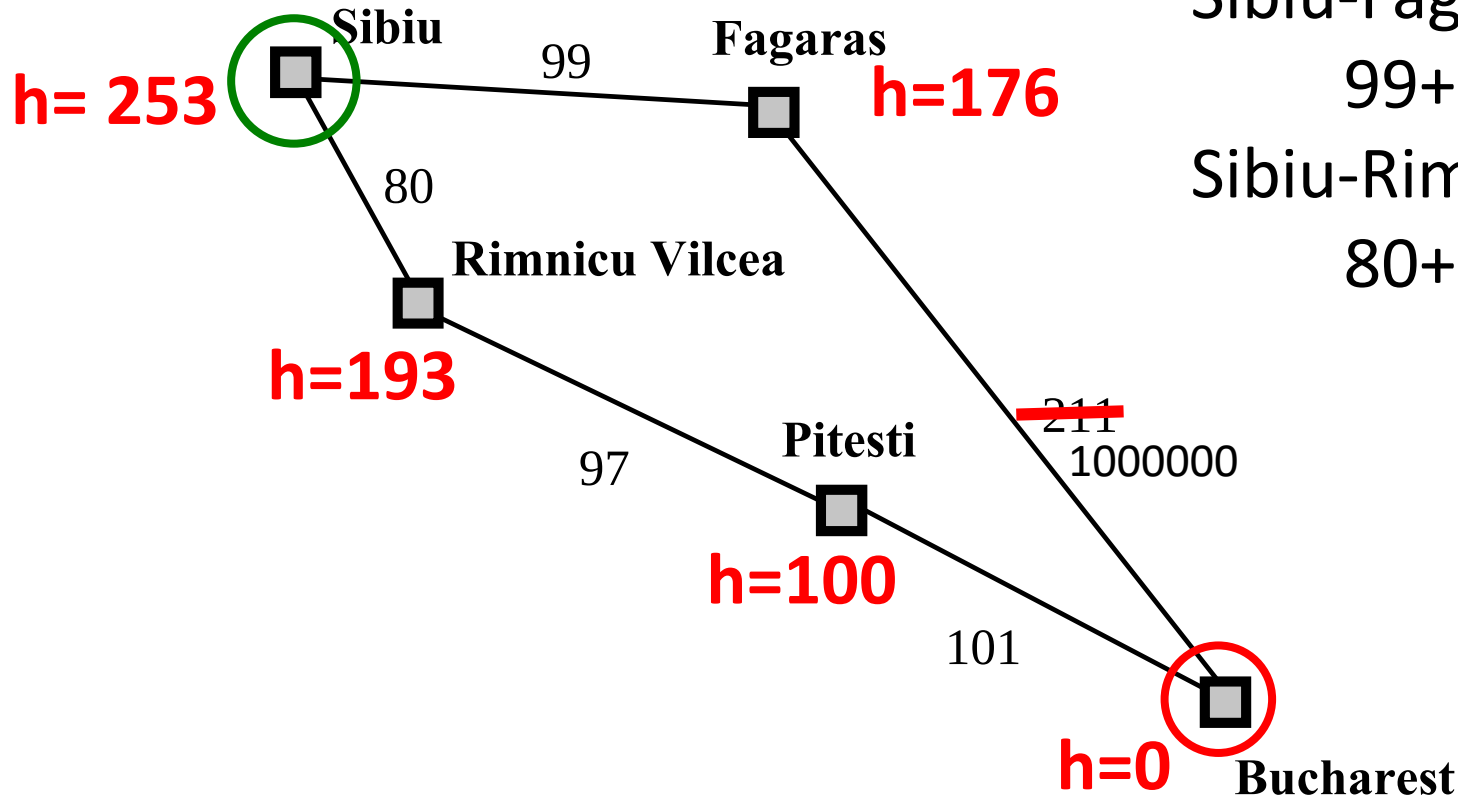
Greedy Search



Greedy Search

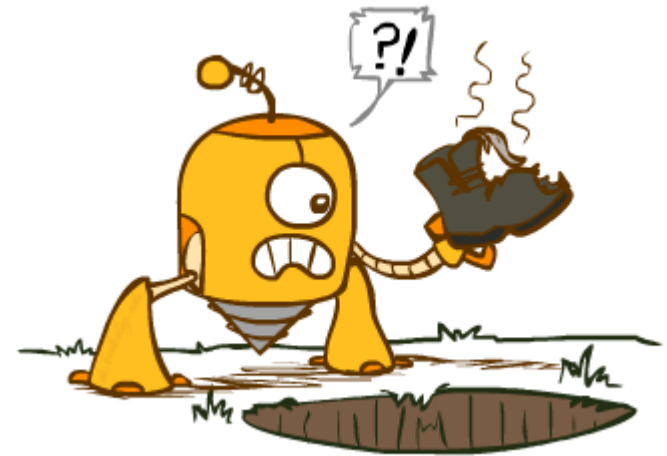
Expand the node that seems closest...(order frontier by h)

What can possibly go wrong?



$$\text{Sibiu-Fagaras-Bucharest} = 99 + 211 = \mathbf{310}$$

$$\text{Sibiu-Rimnicu Vilcea-Pitesti-Bucharest} = 80 + 97 + 101 = \mathbf{278}$$

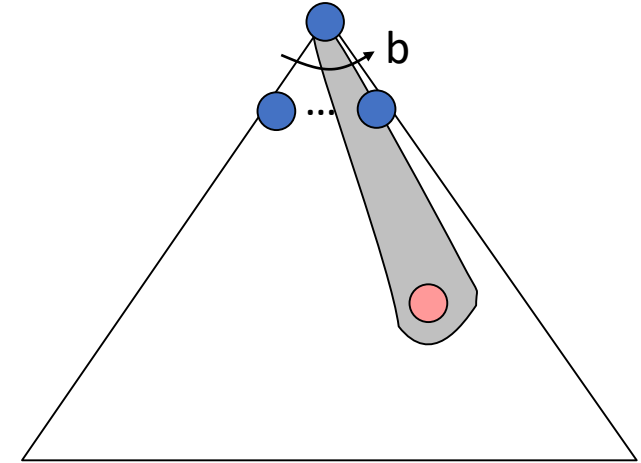


Greedy Search

Strategy: expand a node that *seems* closest to a goal state, according to h

Problem 1: it chooses a node even if it's at the end of a very long and winding road

Problem 2: it takes h literally even if it's completely wrong



Demo Contours Greedy (Empty)

Demo Contours Greedy (Pacman Small Maze)

A* Search



A* Search



UCS



Greedy

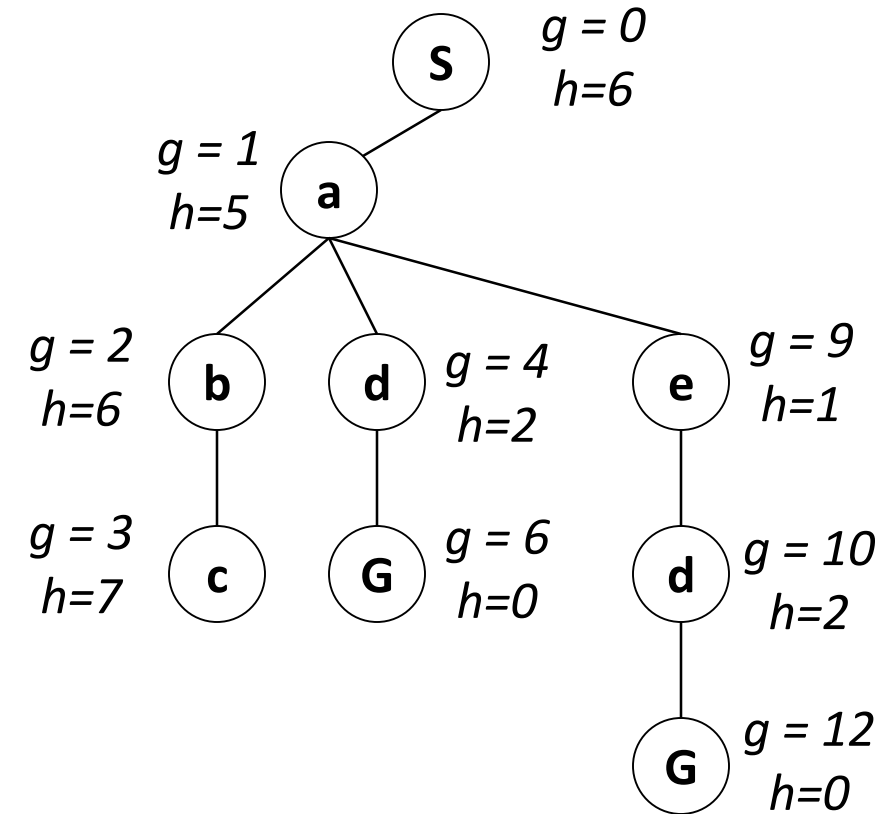
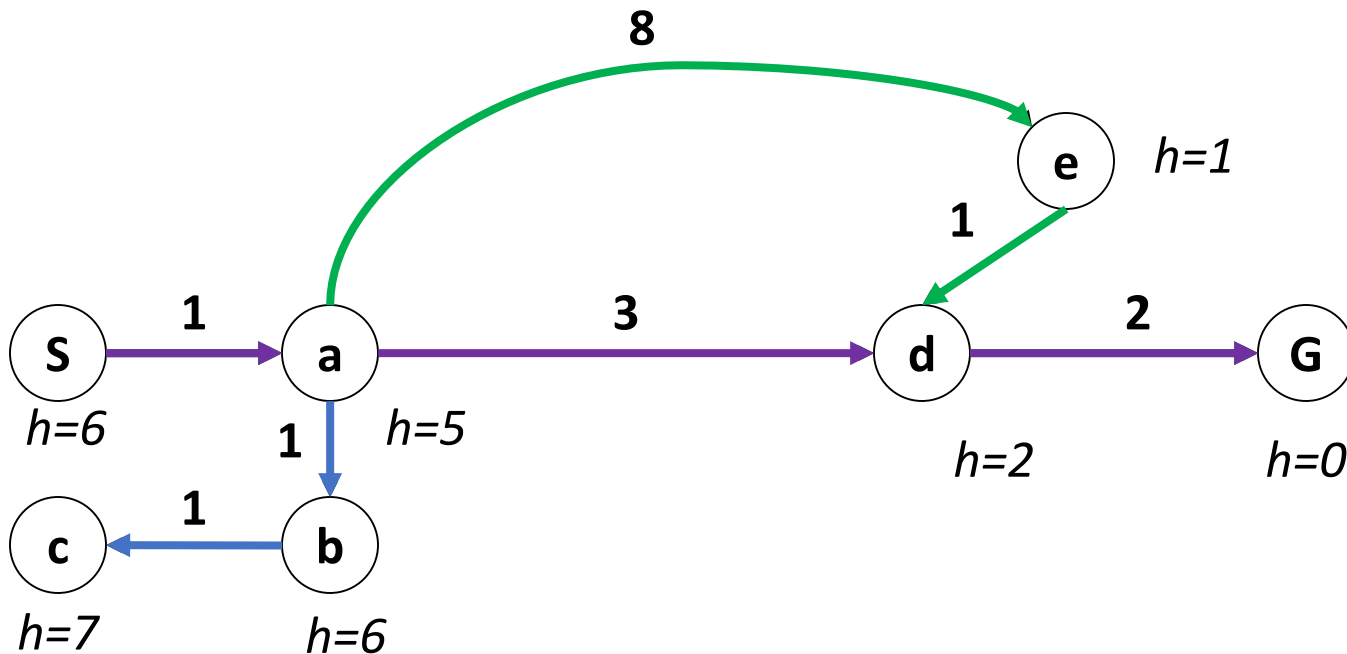


A*

Combining UCS and Greedy

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

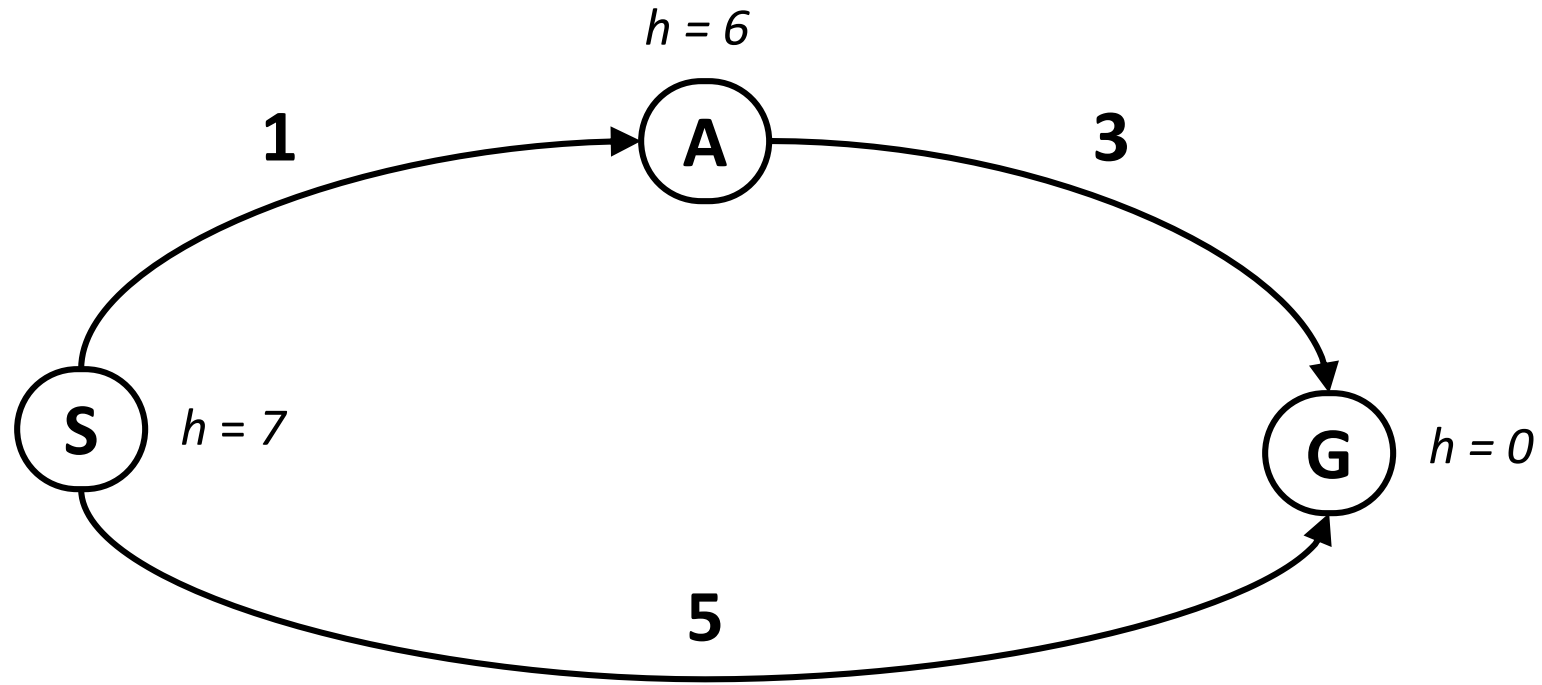
Greedy orders by goal proximity, or *forward cost* $h(n)$



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

Example: Teg Grenager

Is A* Optimal?



What went wrong?

Actual bad goal cost < **estimated** good goal cost

We need estimates to be less than actual costs!

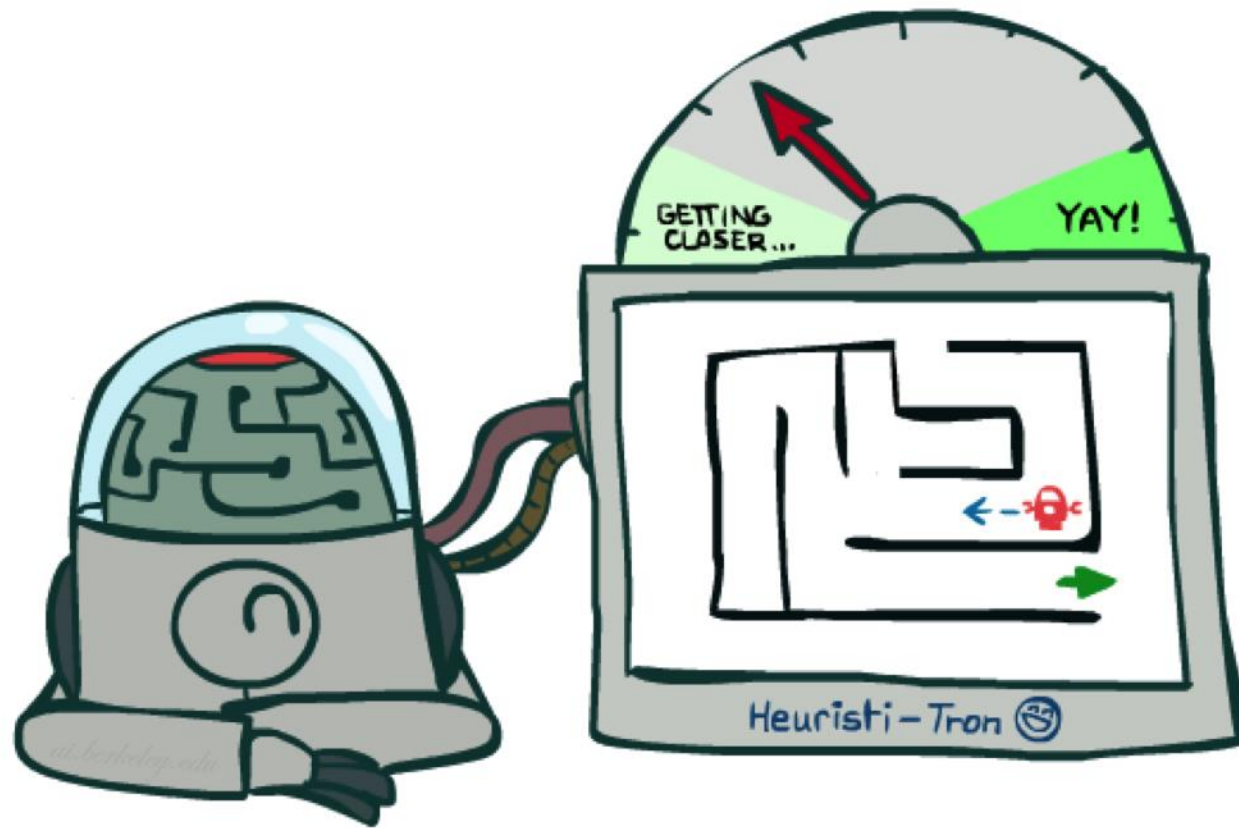
The Price is Wrong...

Closest bid without going over...



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

Admissible Heuristics



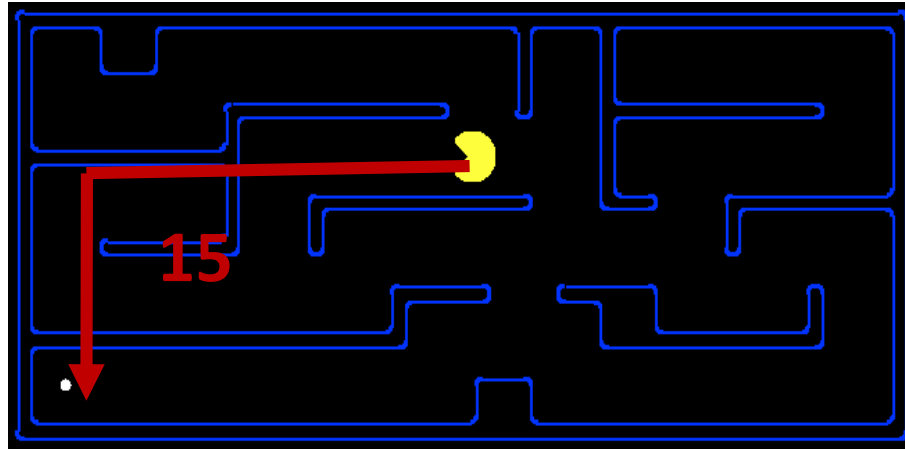
Admissible Heuristics

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

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

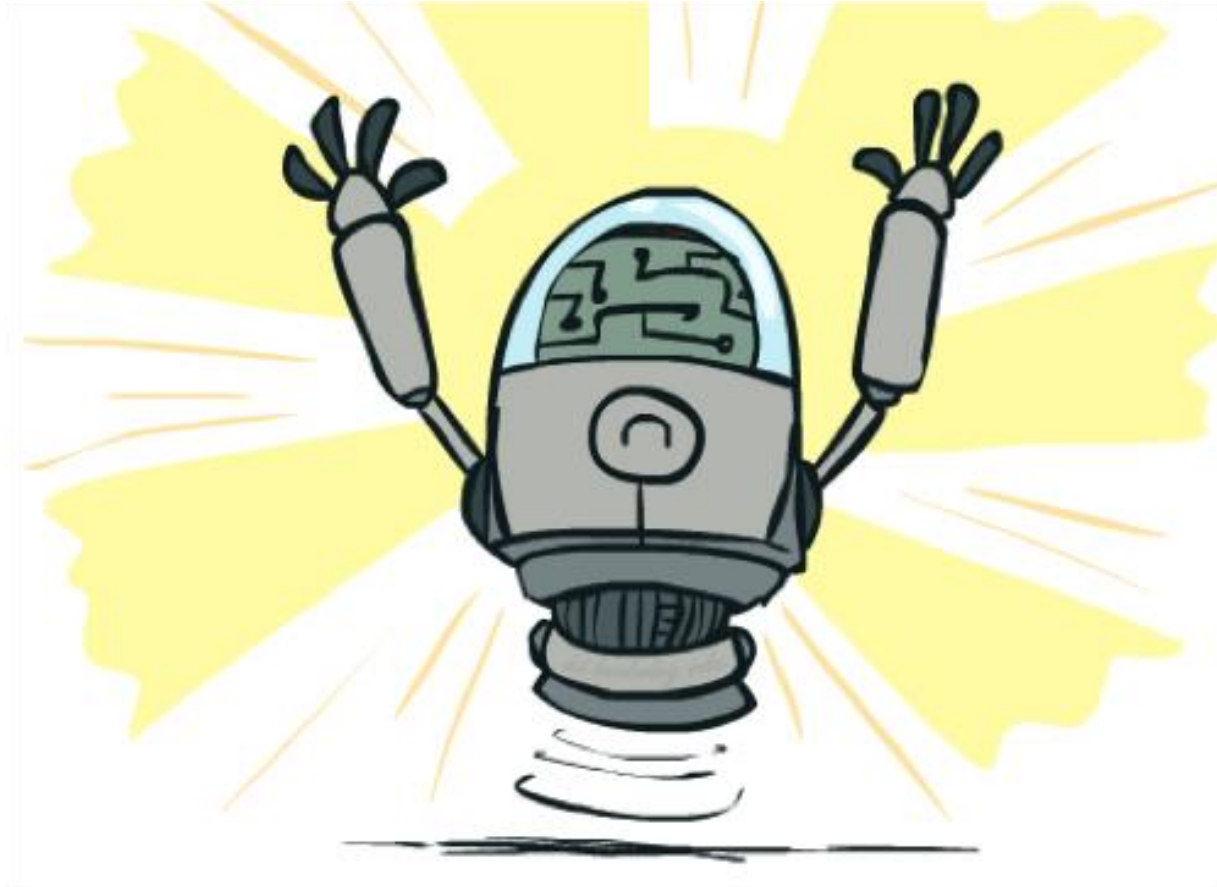
where $h^*(n)$ is the true cost to a nearest goal

Example:



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

Optimality of A* Tree Search



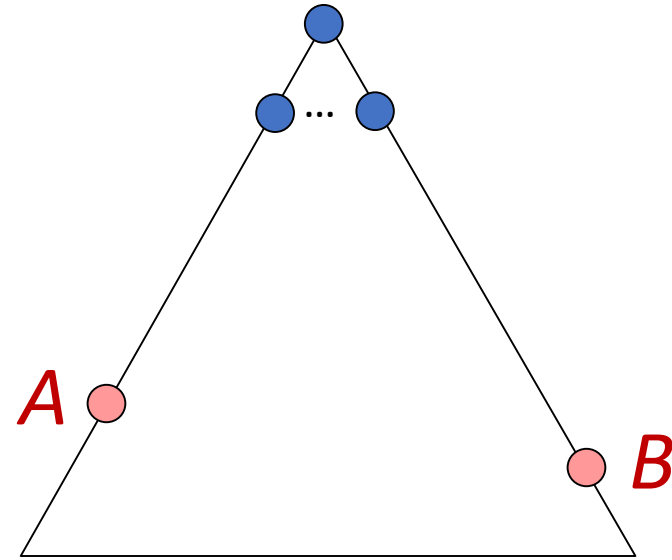
Optimality of A* Tree Search

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

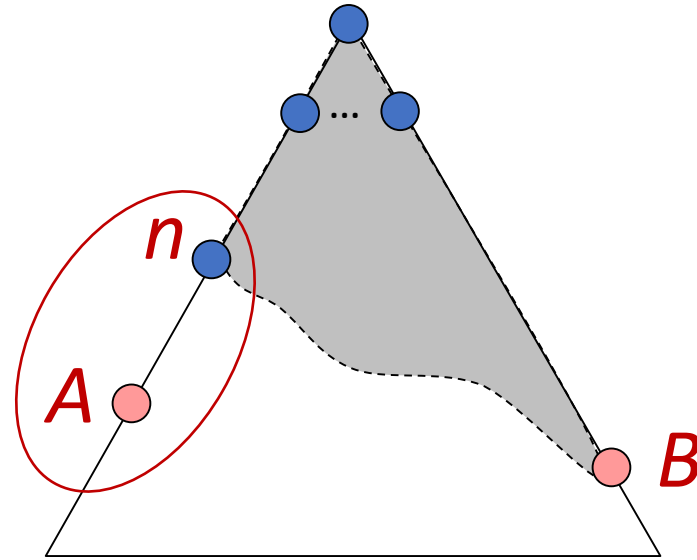
Proof:

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. $f(n)$ is less than or equal to $f(A)$



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

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

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

Definition of f -cost

Admissibility of h

$h = 0$ at a goal

Optimality of A* Tree Search: Blocking

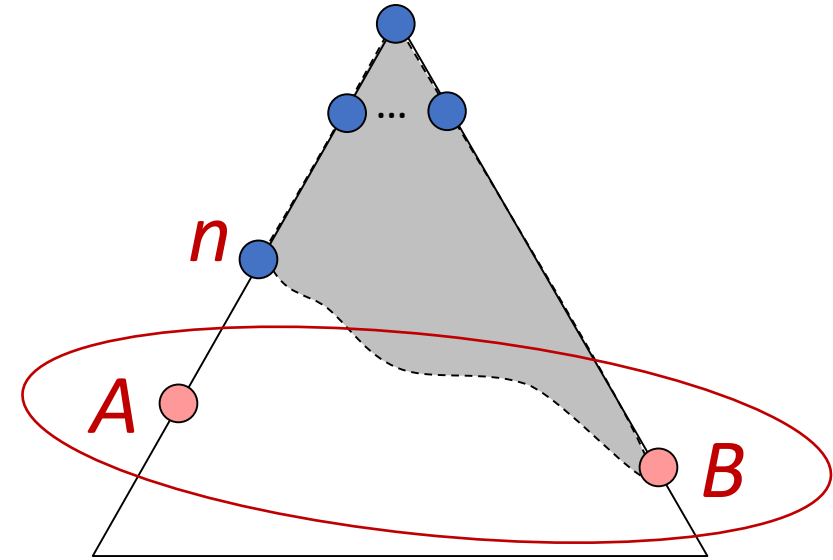
Proof:

Imagine B is on the 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 $f(B)$



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

Suboptimality of B
 $h = 0$ at a goal

Optimality of A* Tree Search: Blocking

Proof:

Imagine B is on the frontier

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

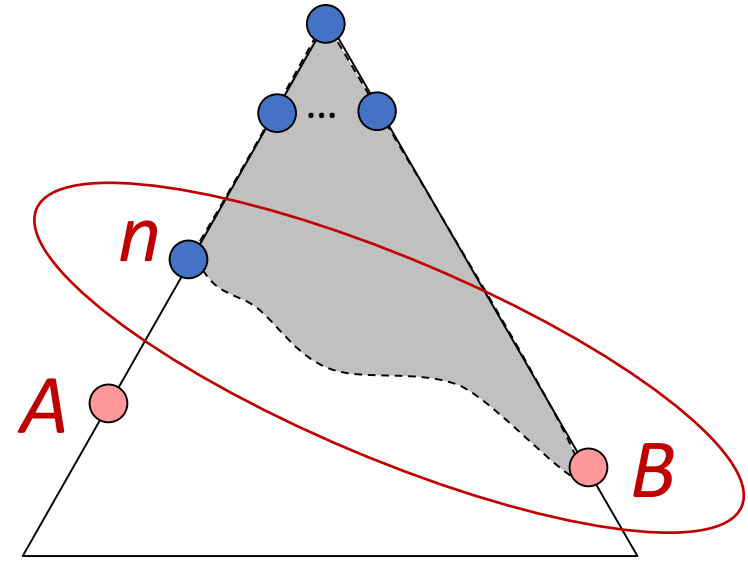
Claim: n will be explored before B

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

All ancestors of A are explored before B

A is explored before B

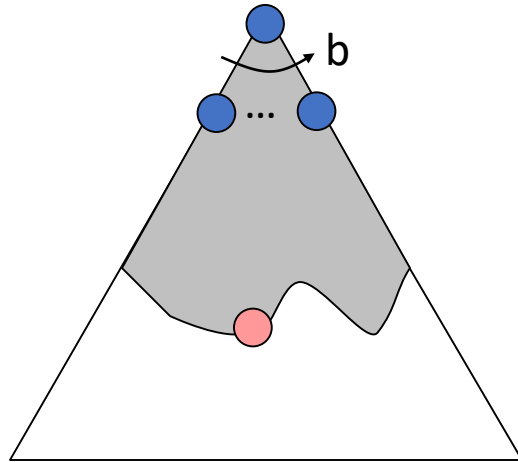
A* search is optimal



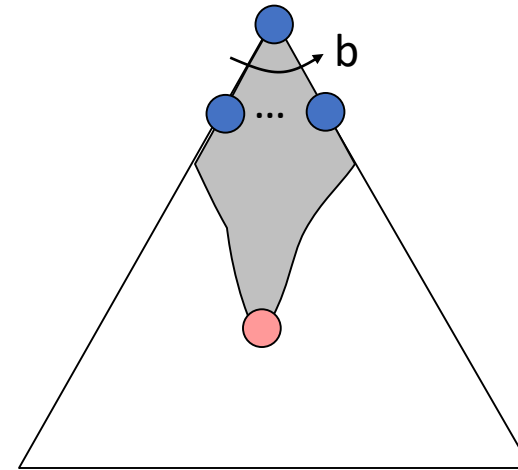
$$f(n) \leq f(A) < f(B)$$

Properties of A^*

Uniform-Cost

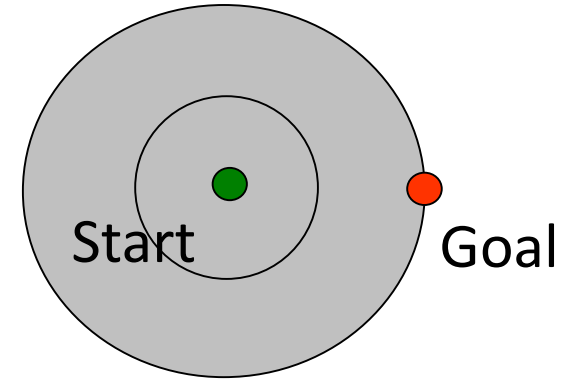


A^*

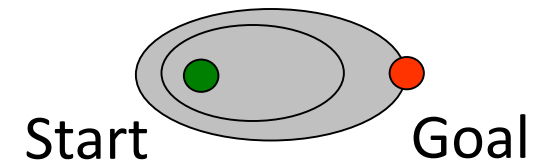


UCS vs A* Contours

Uniform-cost expands equally in all “directions”



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



Demo Contours (Empty) -- UCS

Demo Contours (Empty) – A^*

Demo Contours (Pacman Small Maze) – A*

Comparison



Greedy



Uniform Cost



A*

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 as **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**

A* Applications

Pathing / routing problems

Resource planning problems

Robot motion planning

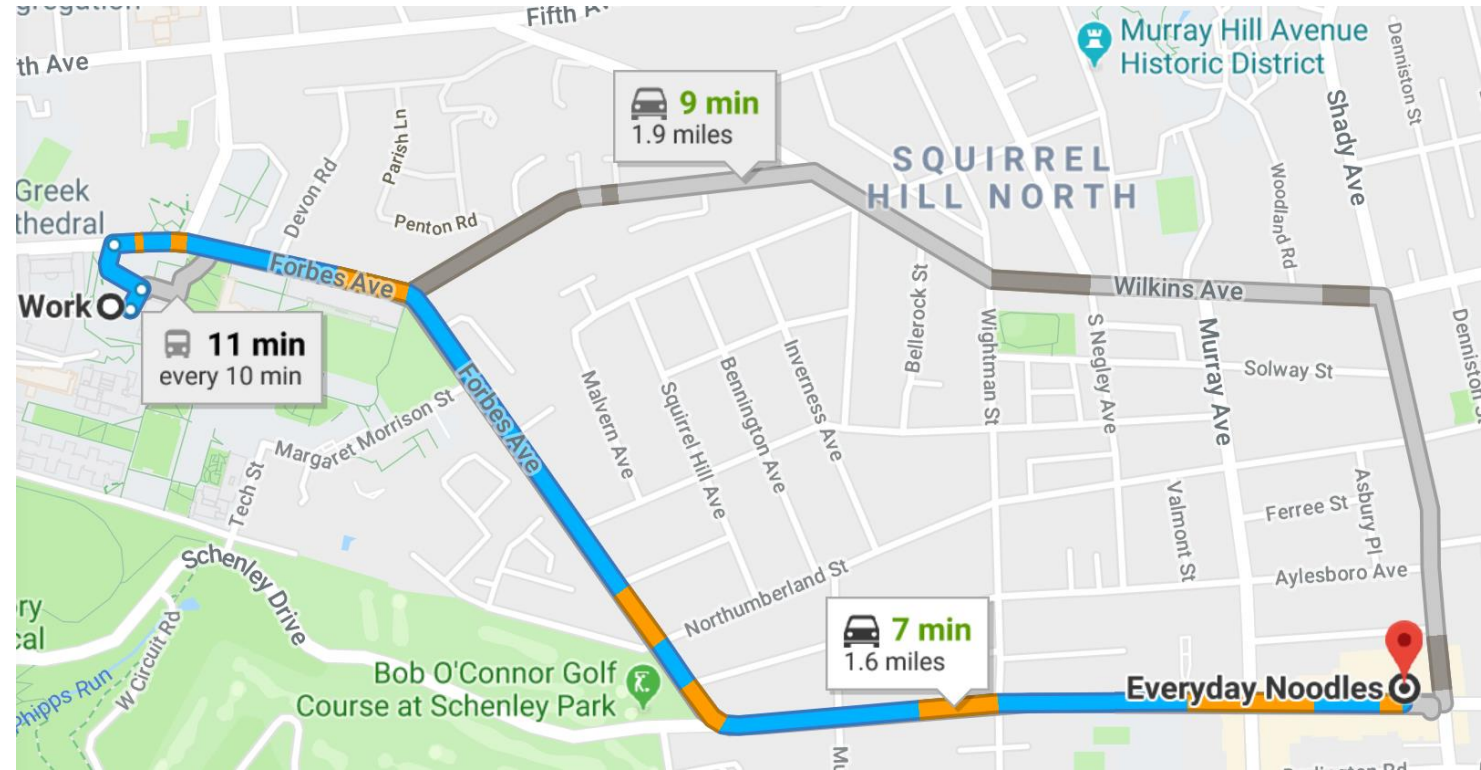
Language analysis

Video games

Machine translation

Speech recognition

...



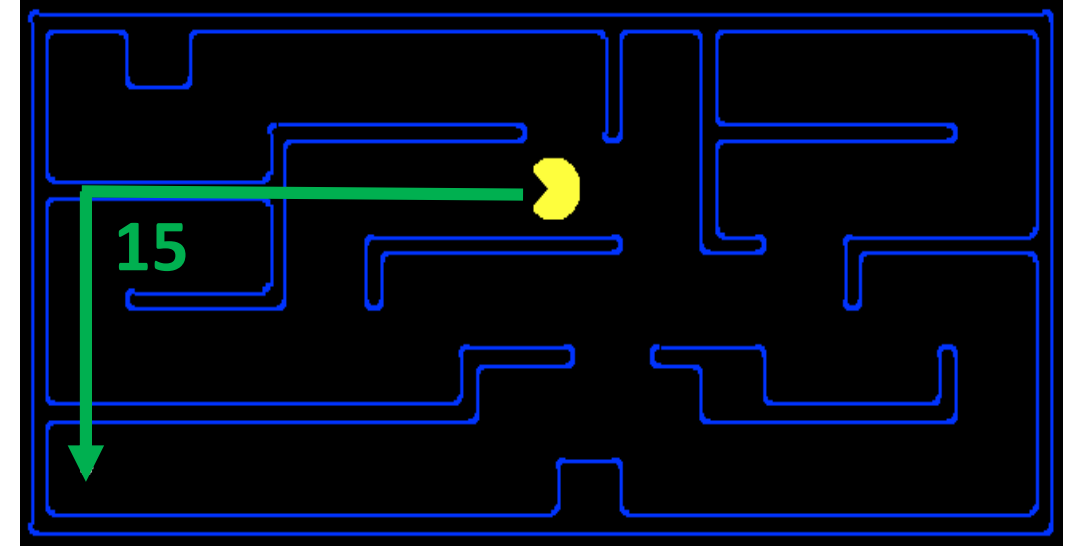
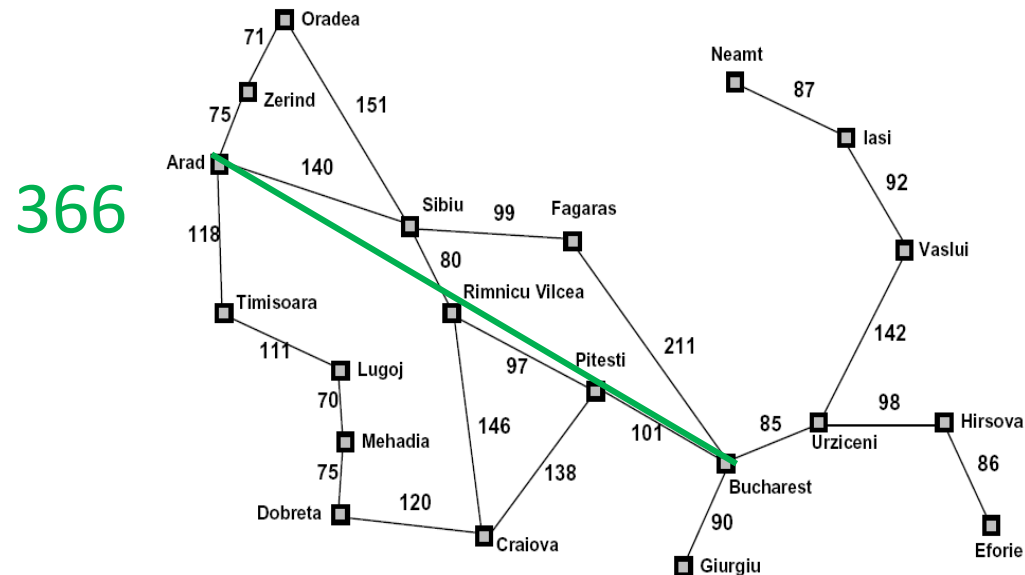
Creating Heuristics



Creating Admissible Heuristics

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

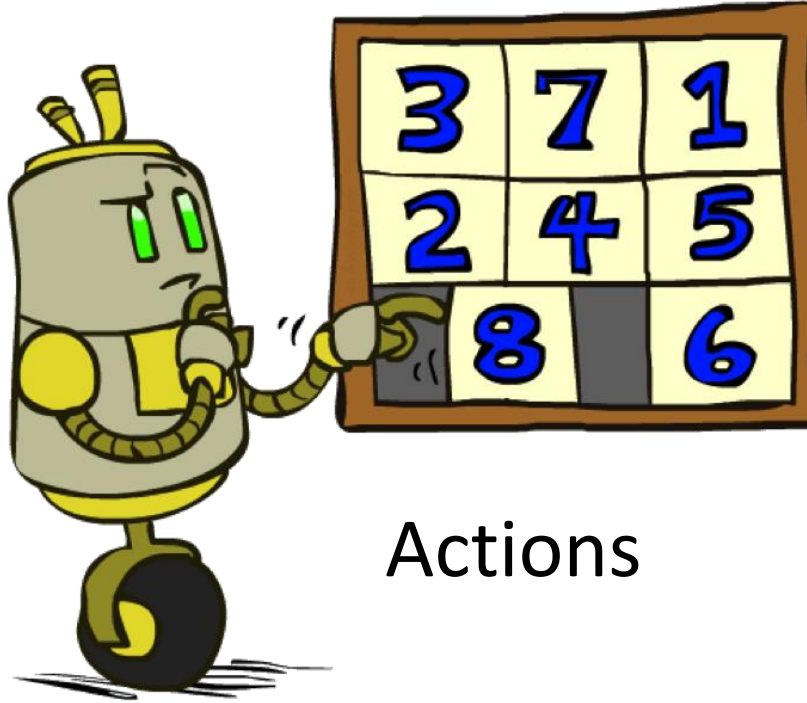
Often, admissible heuristics are solutions to *relaxed problems*, where new actions are available



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?

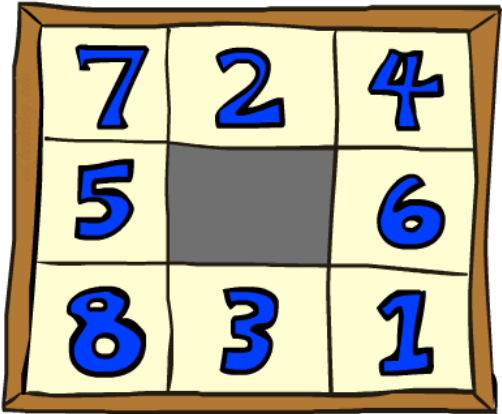
8 Puzzle I

Heuristic: Number of tiles misplaced

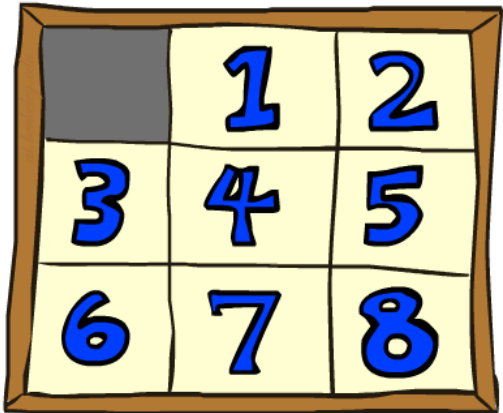
Why is it admissible?

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

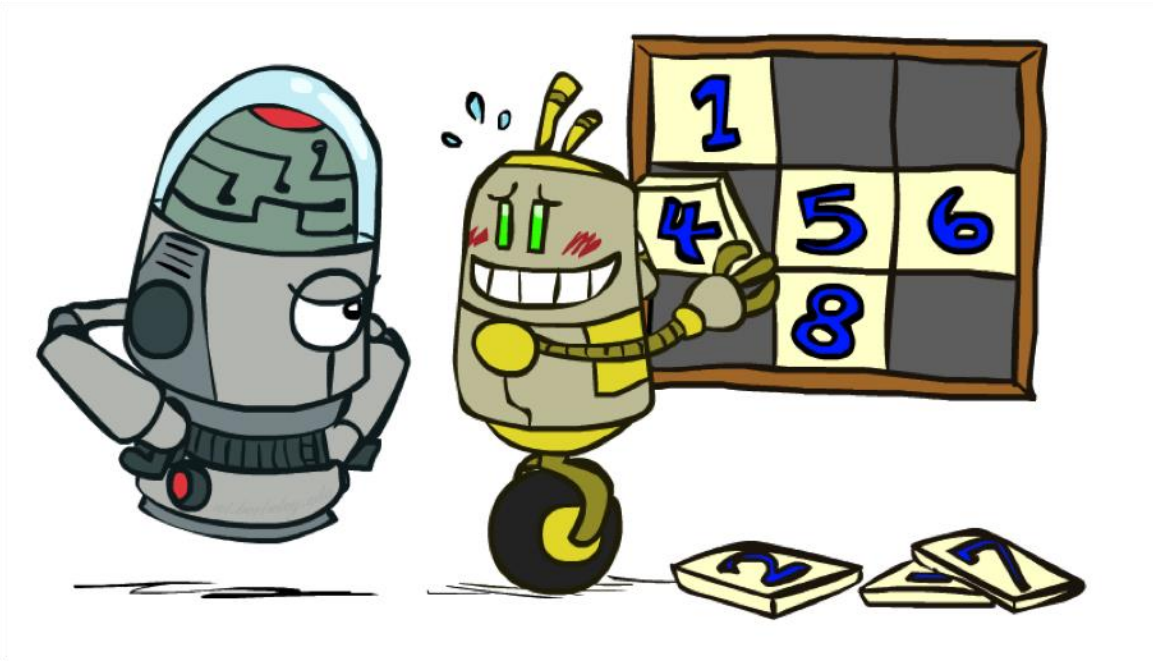
This is a *relaxed-problem* heuristic



Start State



Goal State

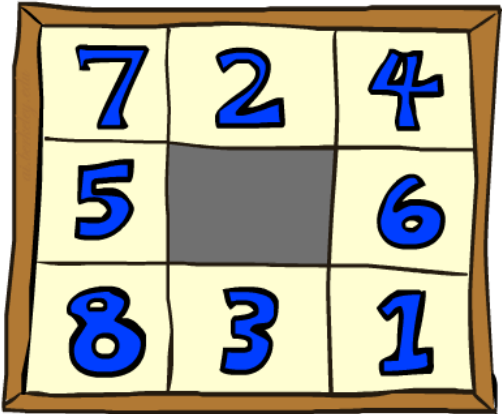


Average nodes expanded when the optimal path has...			
	...4 steps	...8 steps	...12 steps
UCS	112	6,300	3.6×10^6
A*TILES	13	39	227

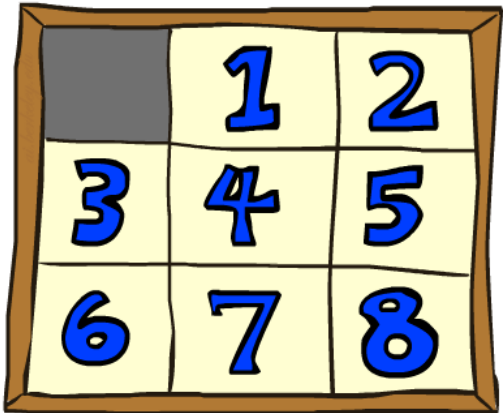
Statistics from Andrew Moore

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

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
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!

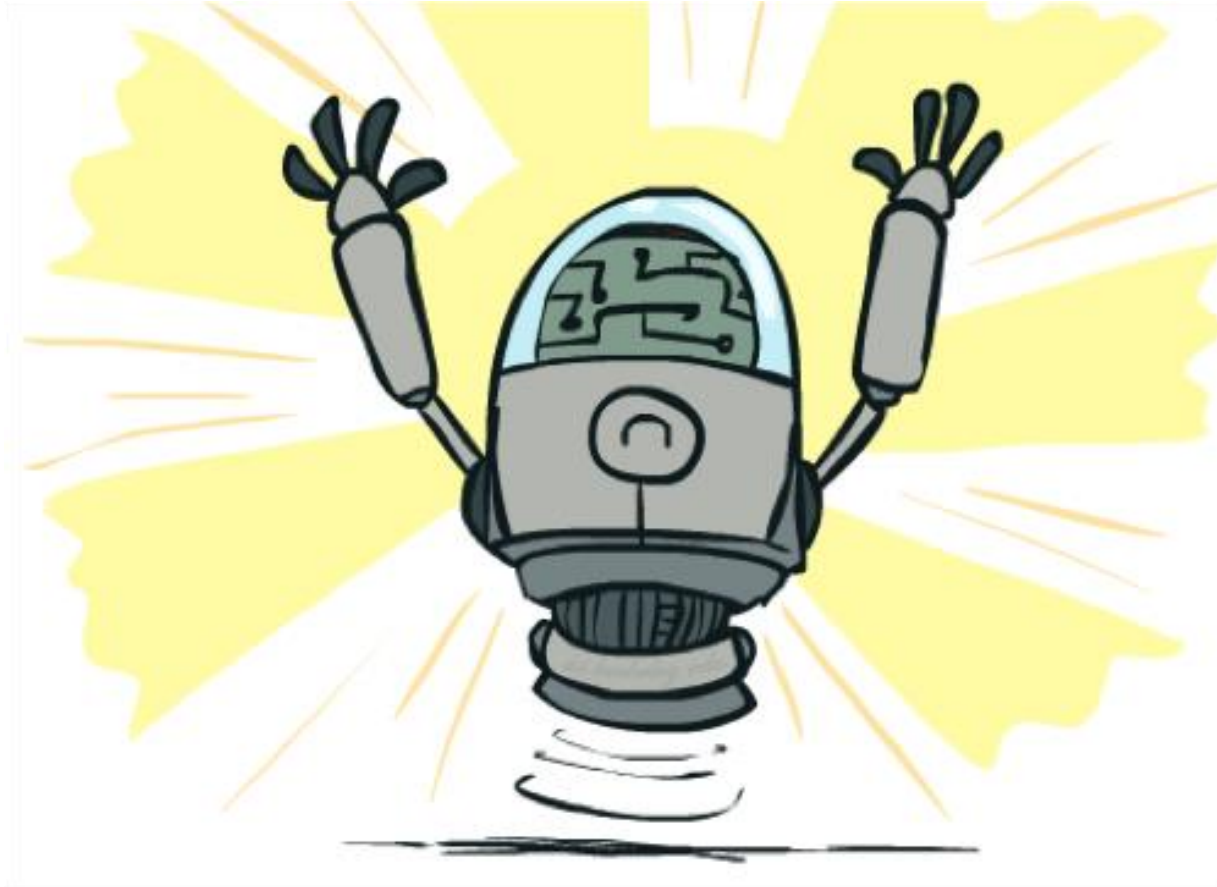
What if we have two heuristics, neither dominates the other?

- Form a new heuristic by taking the max of both:

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

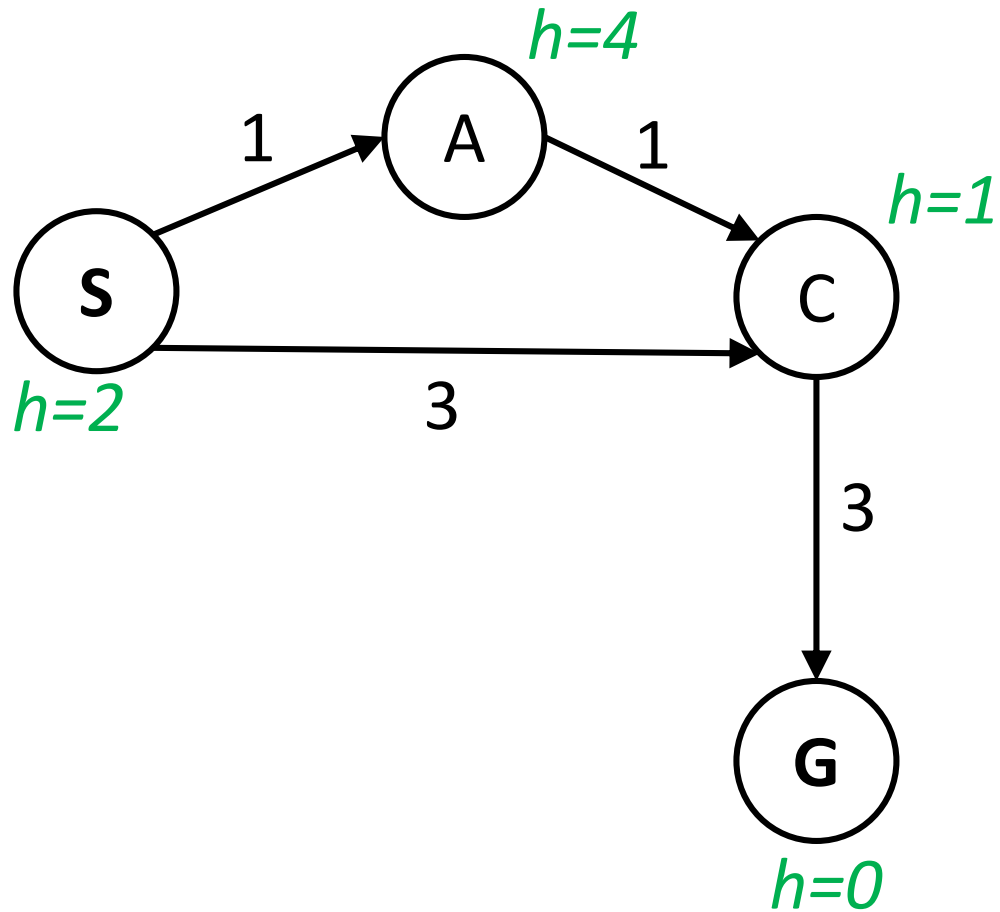
- Max of admissible heuristics is admissible and dominates both!

Optimality of A* Graph Search

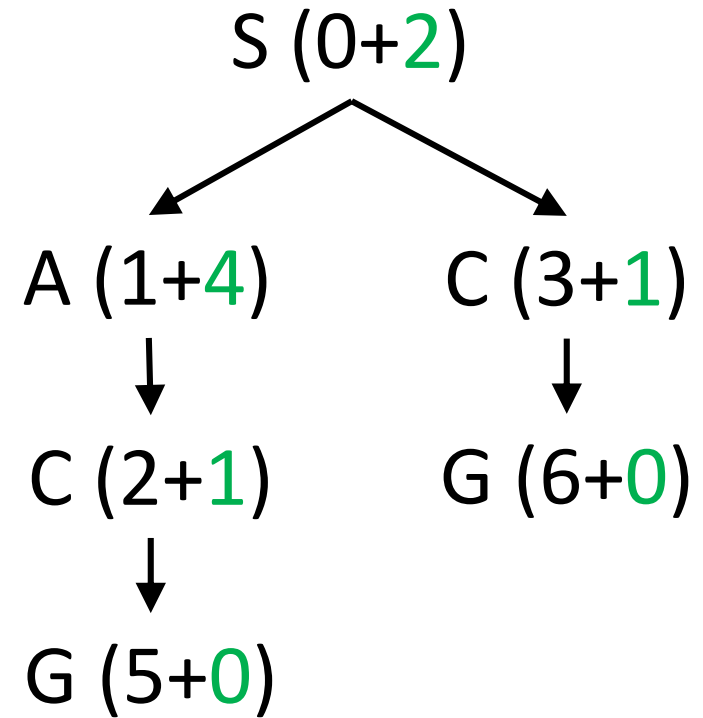


A* Tree Search

State space graph

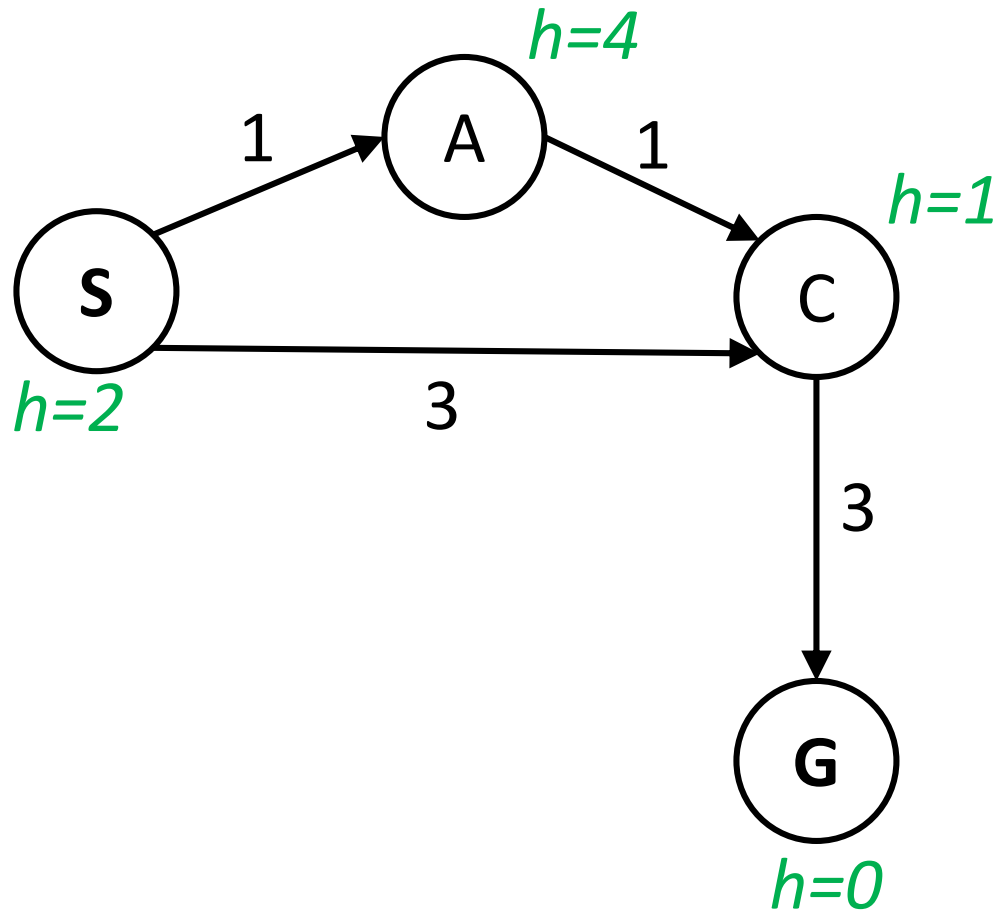


Search tree

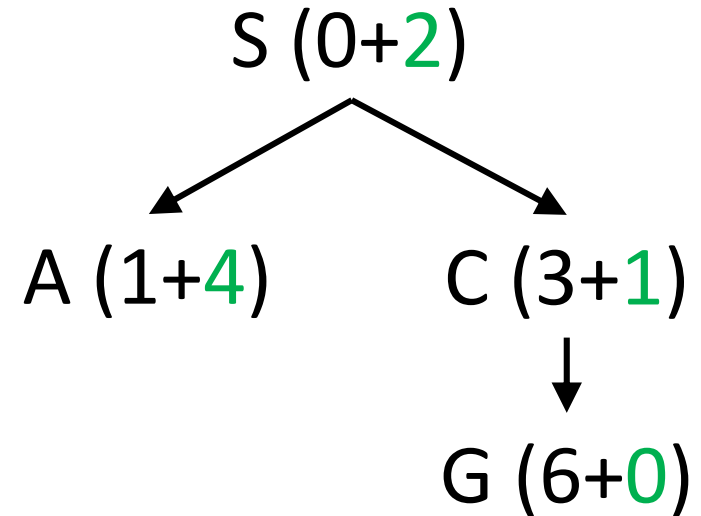


A* Graph Search Gone Wrong?

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

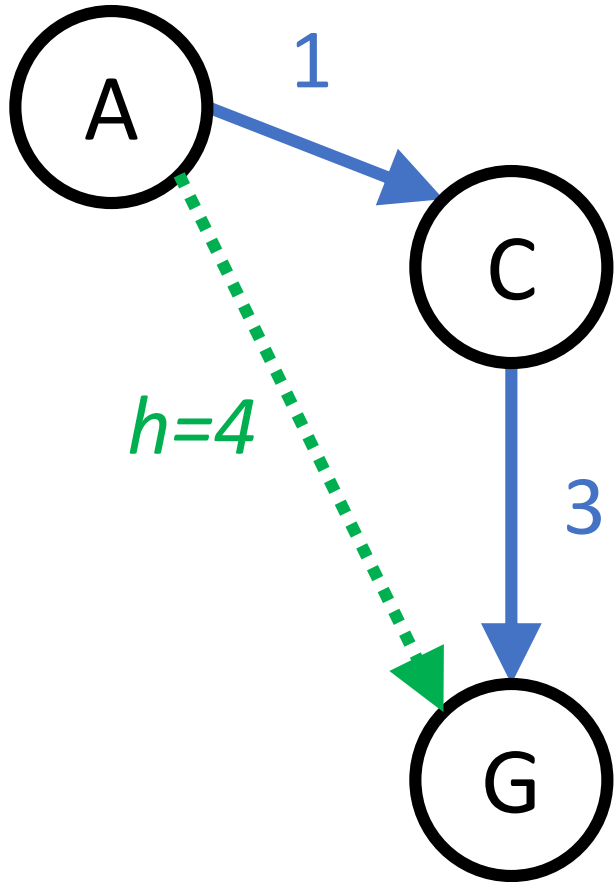
Admissibility of Heuristics

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

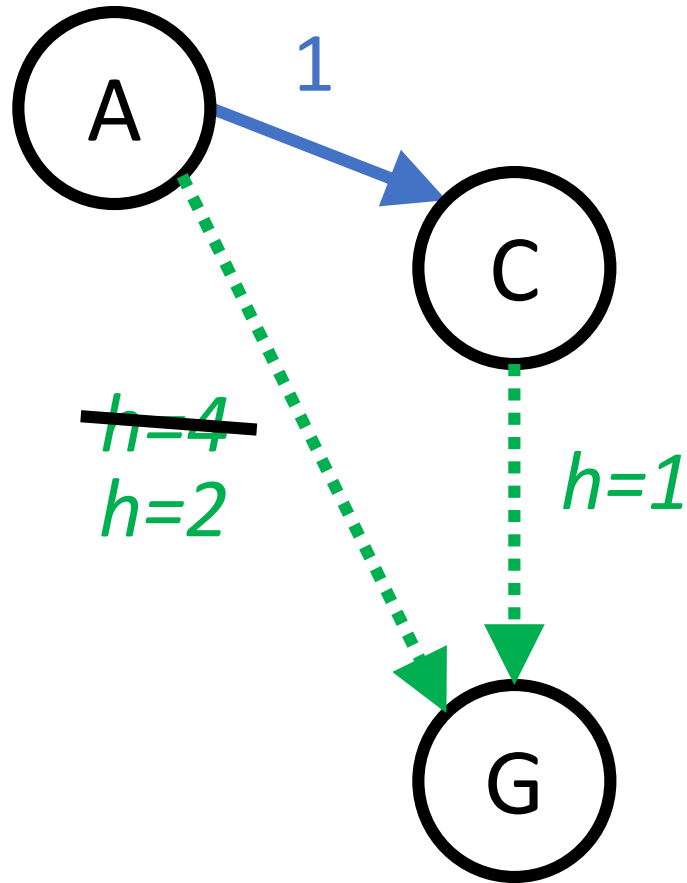
■ Admissibility:

heuristic value $\leq$ actual cost to goal

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



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

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

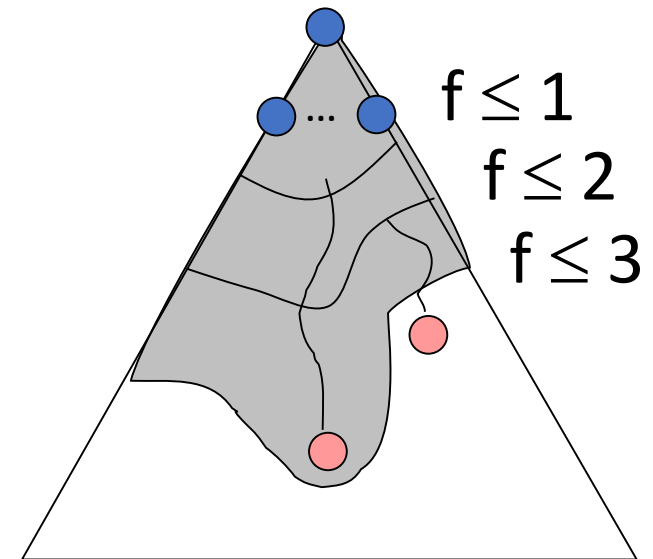
Consequences of consistency:

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

Optimality 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 explored before nodes that reach **s** suboptimally
- Result: A* graph search is optimal



Optimality

Tree search:

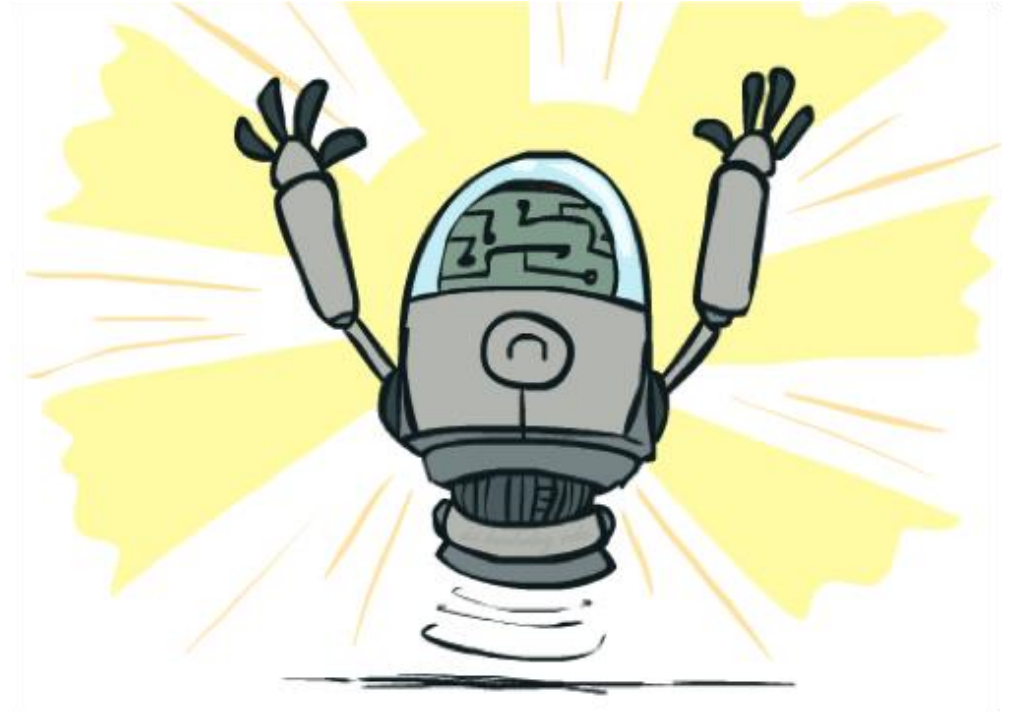
- A* is optimal if heuristic is admissible
- UCS is a special case ($h = 0$)

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



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

