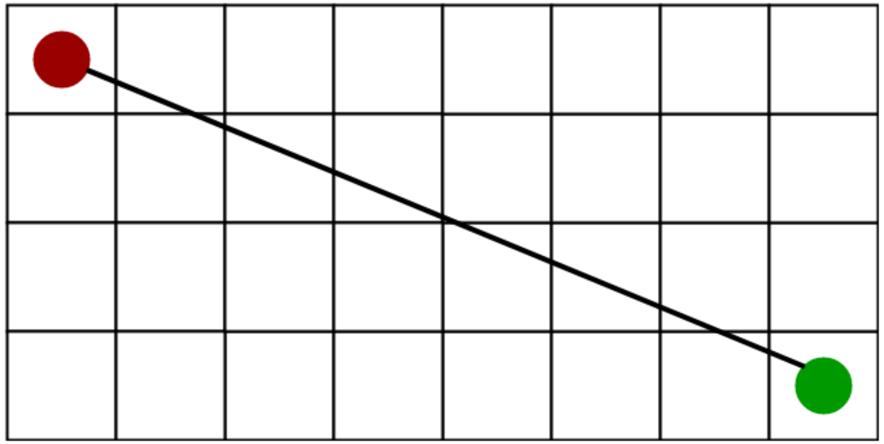
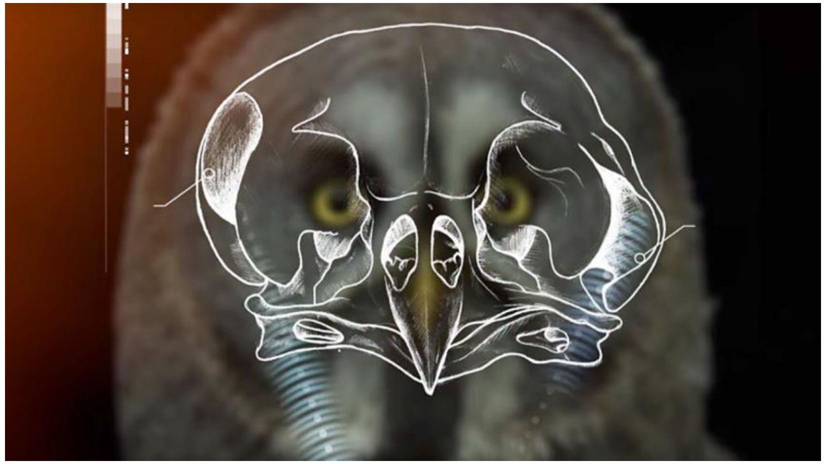
CS 6511: Artificial Intelligence

Informed Search





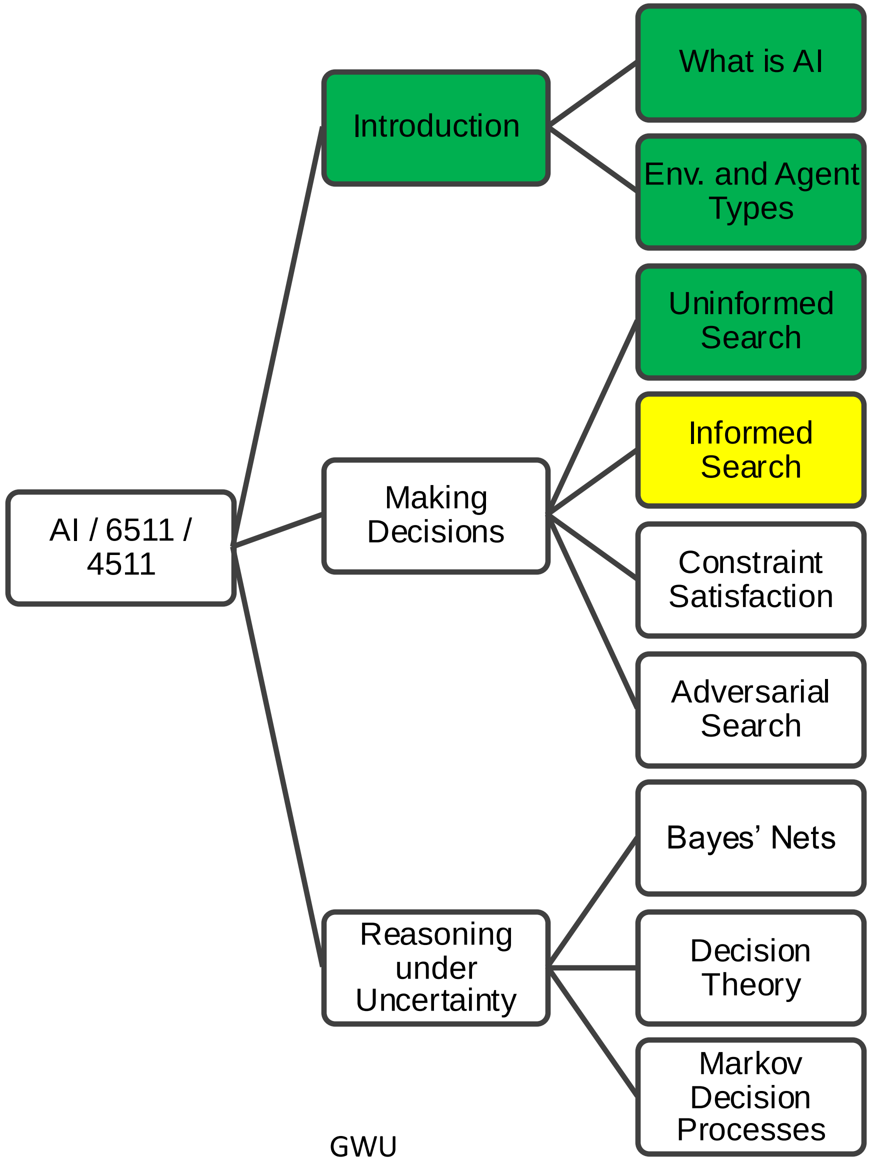
Amrinder Arora

[Original version of these slides was created by Dan Klein and Pieter Abbeel for Intro to AI at UC Berkeley. http://ai.berkeley.edu]

Course Outline

Other AI Topics (Not included in this class):

- Robotics
- NLP
- Machine Learning
- Big Data
- AR/VR
- Speech Synthesis

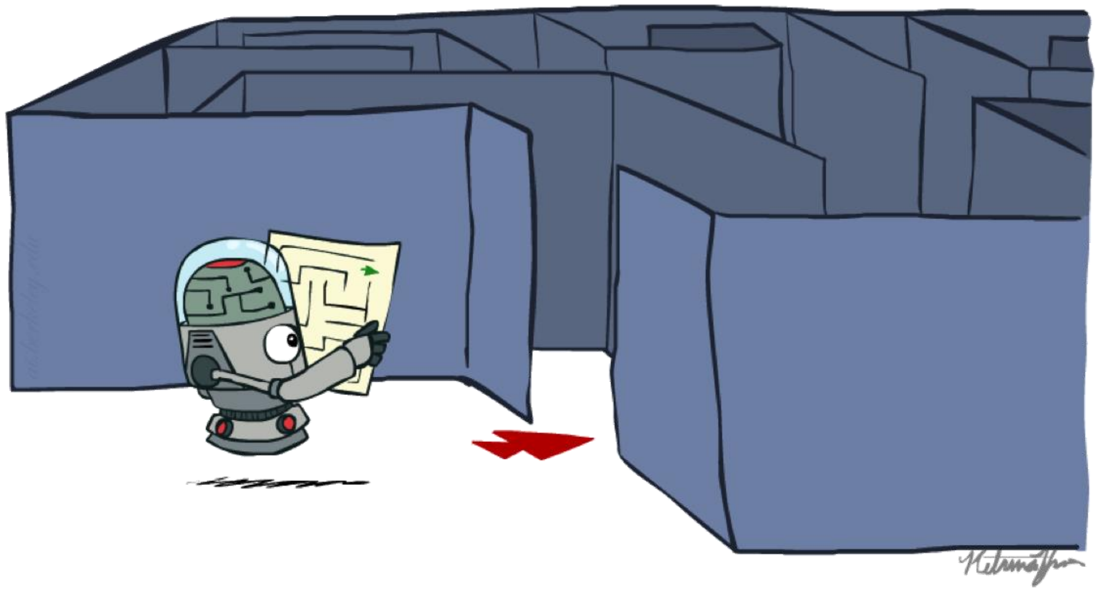


Key Learning Objectives

- Informed Search
  - Concept of “Direction”
    - In physical space
    - In logical solution space
  - 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

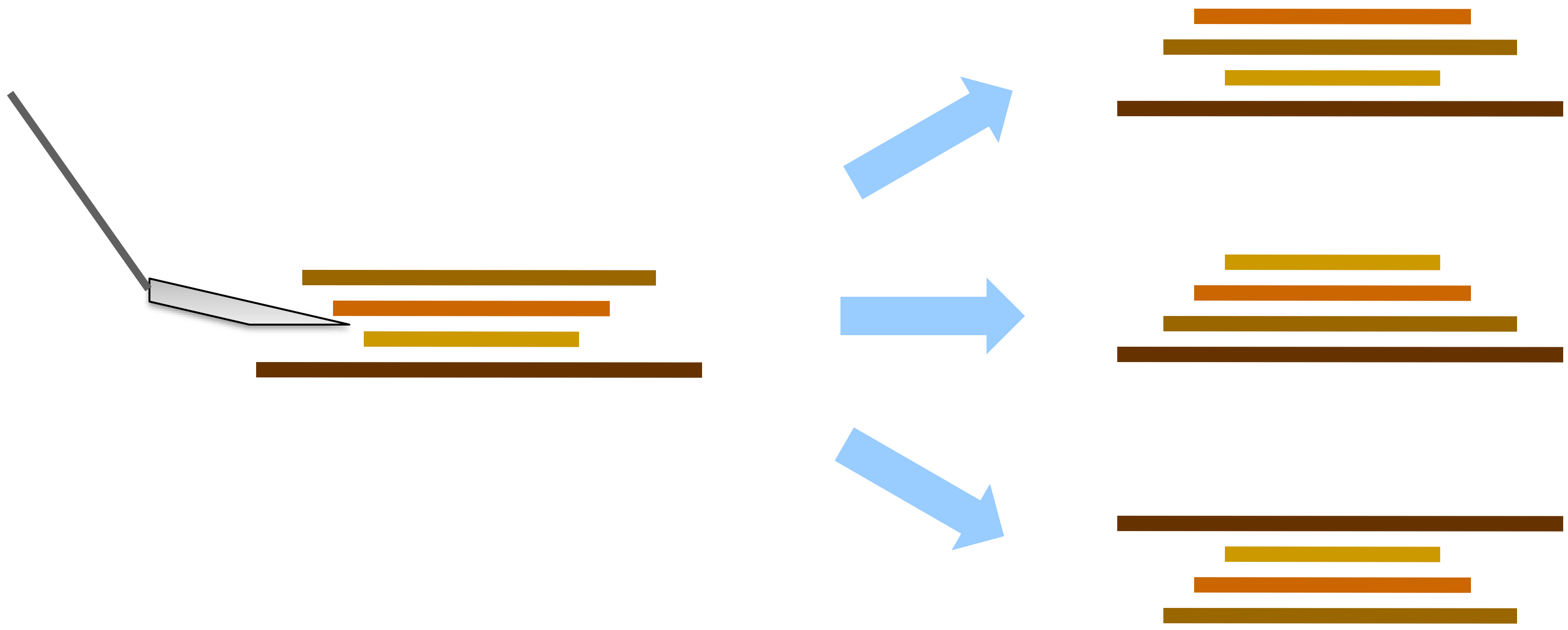


- Search tree:
  - Nodes: represent plans for reaching states
  - Plans have costs (sum of action costs)

- Search algorithm:
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: finds least-cost plans

- Which is smaller:
  - 4 is the branching factor (b): number of successors per node
  - m is the depth (possibly max depth)
  - If m is 10: 40 (DFS) or 4^10 (BFS)
  - If m is 20: 80 (DFS) or 4^20 (BFS)
  - If m is 30: 120 (DFS) or 4^30 (BFS)

Example: Pancake Problem



Cost: Number of pancakes flipped

Example Run

4	2	3	5	4	1	2	3	1
3	5	1	1	2	3	3	2	2
1	1	5	3	3	2	1	1	3
5	3	2	2	1	4	4	4	4
2	4	4	4	5	5	5	5	5
	5	4	3	5	4	3	2	3

Total cost = 29

Example Run

- 1
- 2
- 3
- 4
- 5

- 1
- 2
- 3
- 4
- 5
- 0

Total cost = 0

Example Run

- 5
- 4
- 3
- 2
- 1

- 1
- 2
- 3
- 4
- 5

- 5

Total cost = 5

Example Run

4	2	3	5	4	1	2	3	1
3	5	1	1	2	3	3	2	2
1	1	5	3	3	2	1	1	3
5	3	2	2	1	4	4	4	4
2	4	4	4	5	5	5	5	5
	5	4	3	5	4	3	2	3

Total cost = 29

Example: Pancake Problem

BOUNDS FOR SORTING BY PREFIX REVERSAL

William H. GATES

Microsoft, Albuquerque, New Mexico

Christos H. PAPADIMITRIOU*†

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

Received 18 January 1978
Revised 28 August 1978

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

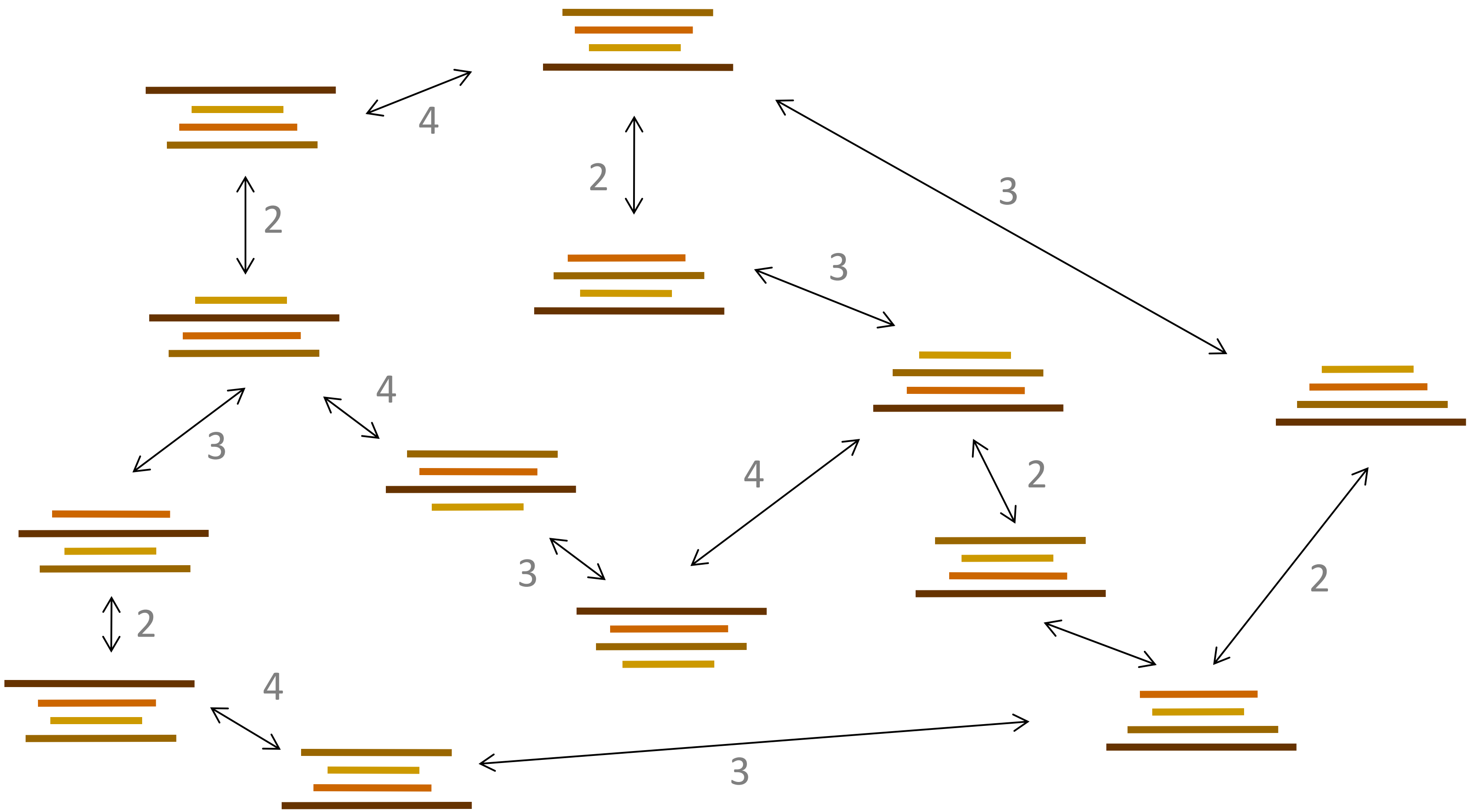
- 8, 5, 4, 10, 1, 2, 7, 3, 9, 6
- 10, 4, 5, 8, …
- 6, 9, 3, 7, 2, 1, 8, 5, 4, 10
- 9, 6, 3, 7, 2, 1, 8, 5, 4, 10
- 4, 5, 8, 1, 2, 7, 3, 6, 9, 10

Longest "Shortest" Path

- Can be confusing at times! Let us make sure we understand it.
- Concept is also used in graphs
  - Shortest path between each pair = distance between *that* pair
  - Longest distances across *all* pairs = diameter of the graph
  - In other words, it is the *longest* distance, when taken over all pairs.
- f(sigma) is the corresponding "distance"
- f(n) is the corresponding "diameter"
- Their result, from 1978, is that $f(n) <= 5n/3 + 5/3$.
- Best result, in 2011: 18/11 n

Example: Pancake Problem

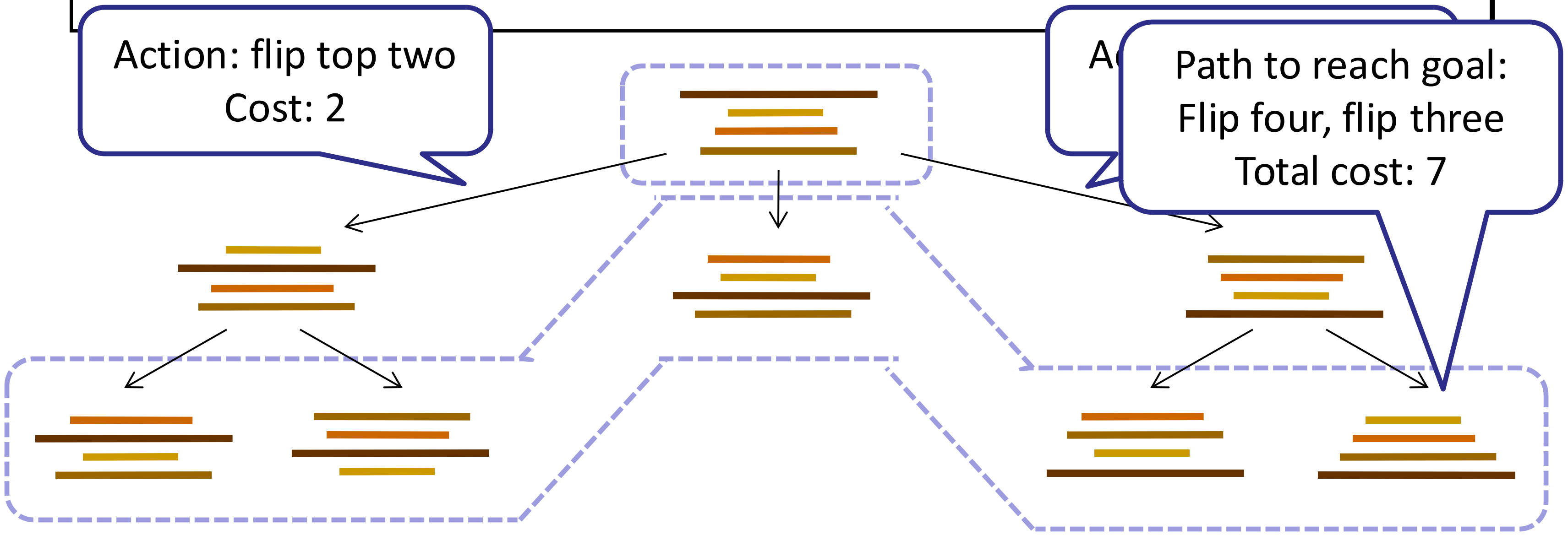
State space graph with costs as weights



General Tree Search

> You know exactly where you came from and how you got there, but you have no idea where you're going. But, you'll know it when you see it.

function TREE-SEARCH(*problem, strategy*) **returns** a solution, or failure
  initialize the search tree using the initial state of *problem*
  loop do
    if there are no candidates for expansion **then return** failure
    choose a leaf node for expansion according to *strategy*
    if the node contains a goal state **then return** the corresponding solution
    else expand the node and add the resulting nodes to the search tree
  end

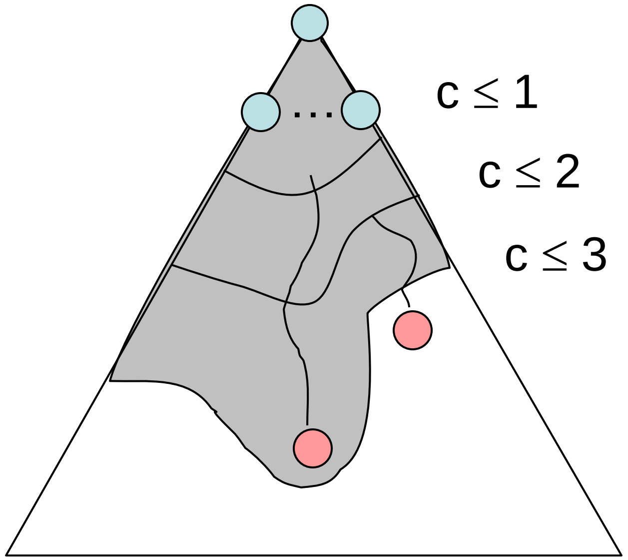


Uninformed Search

- Uninformed search is **structured** search, but does not have any way of knowing which way to go. It is an improvement in terms of endless looping, but not intelligent.

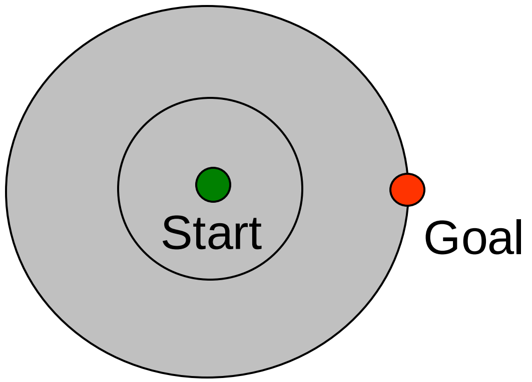
Uniform Cost Search

- Strategy: expand lowest path cost (BFS, but considers weights/costs of edges)



- The good: UCS is complete and optimal!

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



Informed Search

- Concept of “direction”

- Which node may be better to explore (which one is **estimated** to be closer to a goal node)

Key Idea

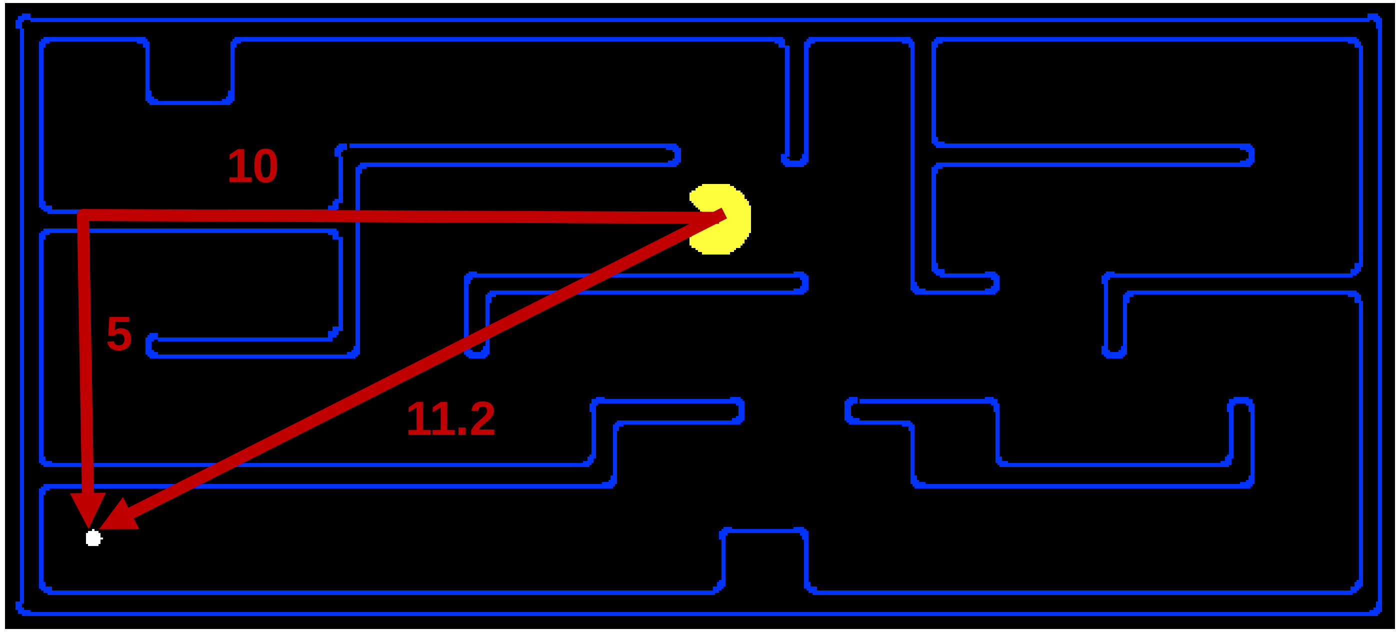
DIRECTION (WHICH NODE TO EXPAND)

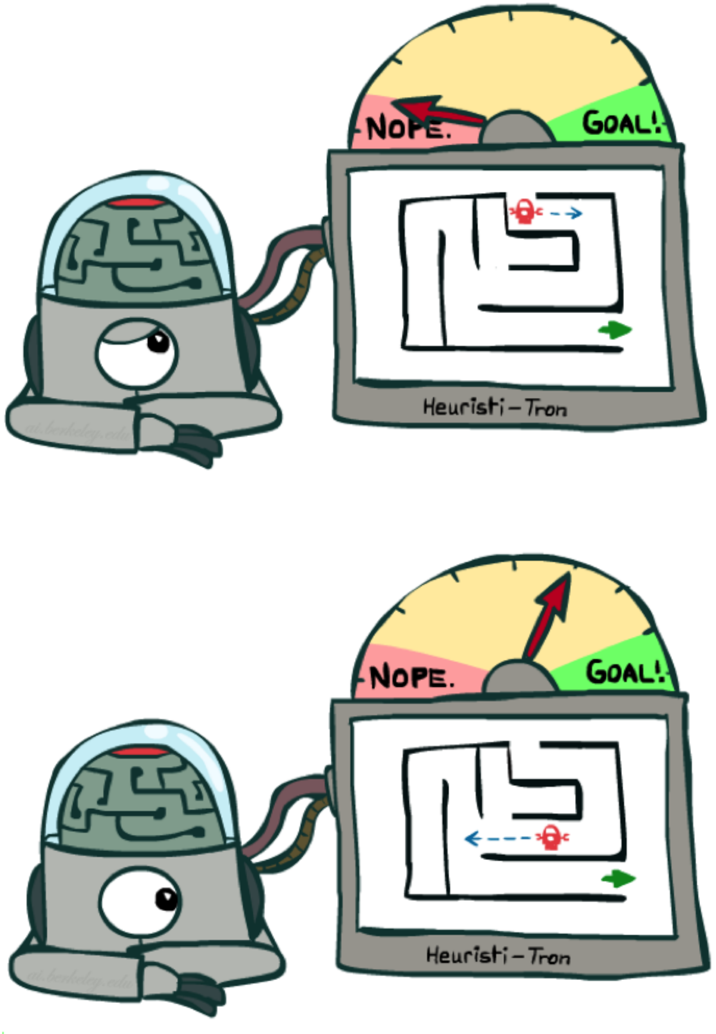
How do we use Direction/Information

- We “hear” the direction where the sound is coming from, based on the distance (the time) it takes for the sound to reach the two ears.
  - “Studies of barn owls offer insight into just how the brain combines acoustic signals from two sides of the head into a single spatial perception”
  - https://www.scientificamerican.com/article/listening-with-two-ears-2006-09/

Search Heuristics

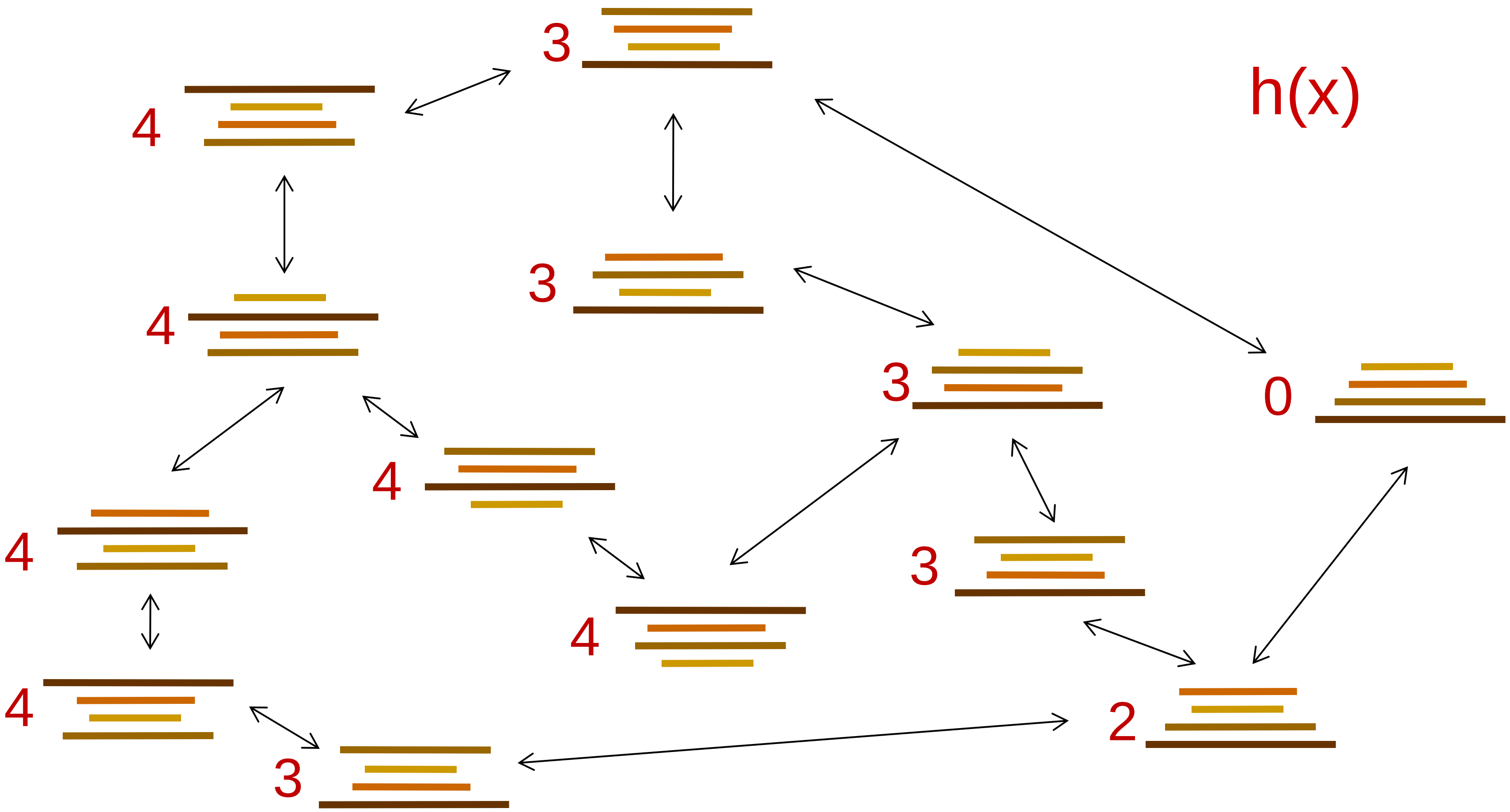
- A heuristic is:
  - A function that *estimates* how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance for pathing





Heuristic Function: Example 1

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

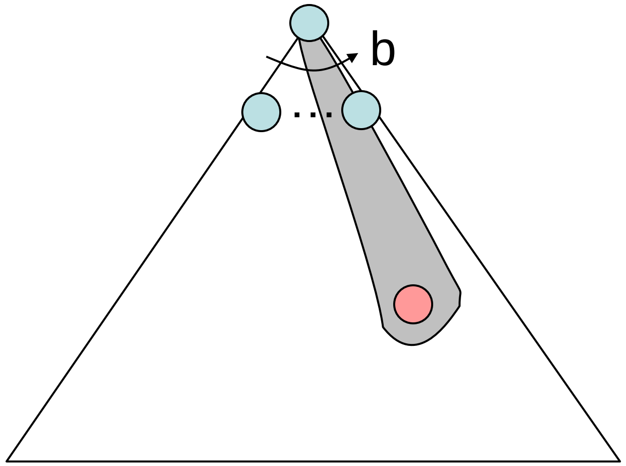


Heuristic Function: Example 2

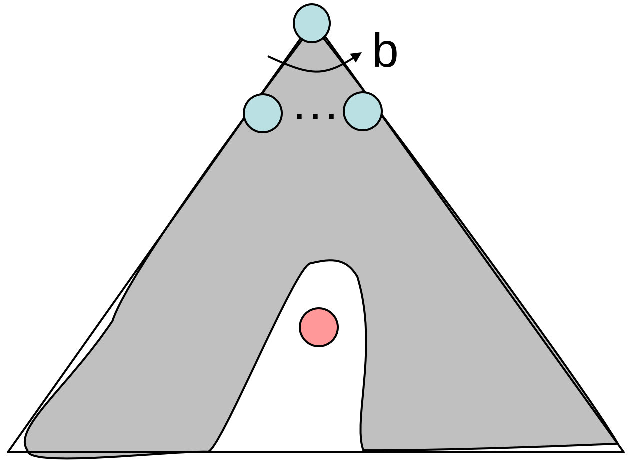
- Euclidean (Flying) distance

Greedy (Informed) Search

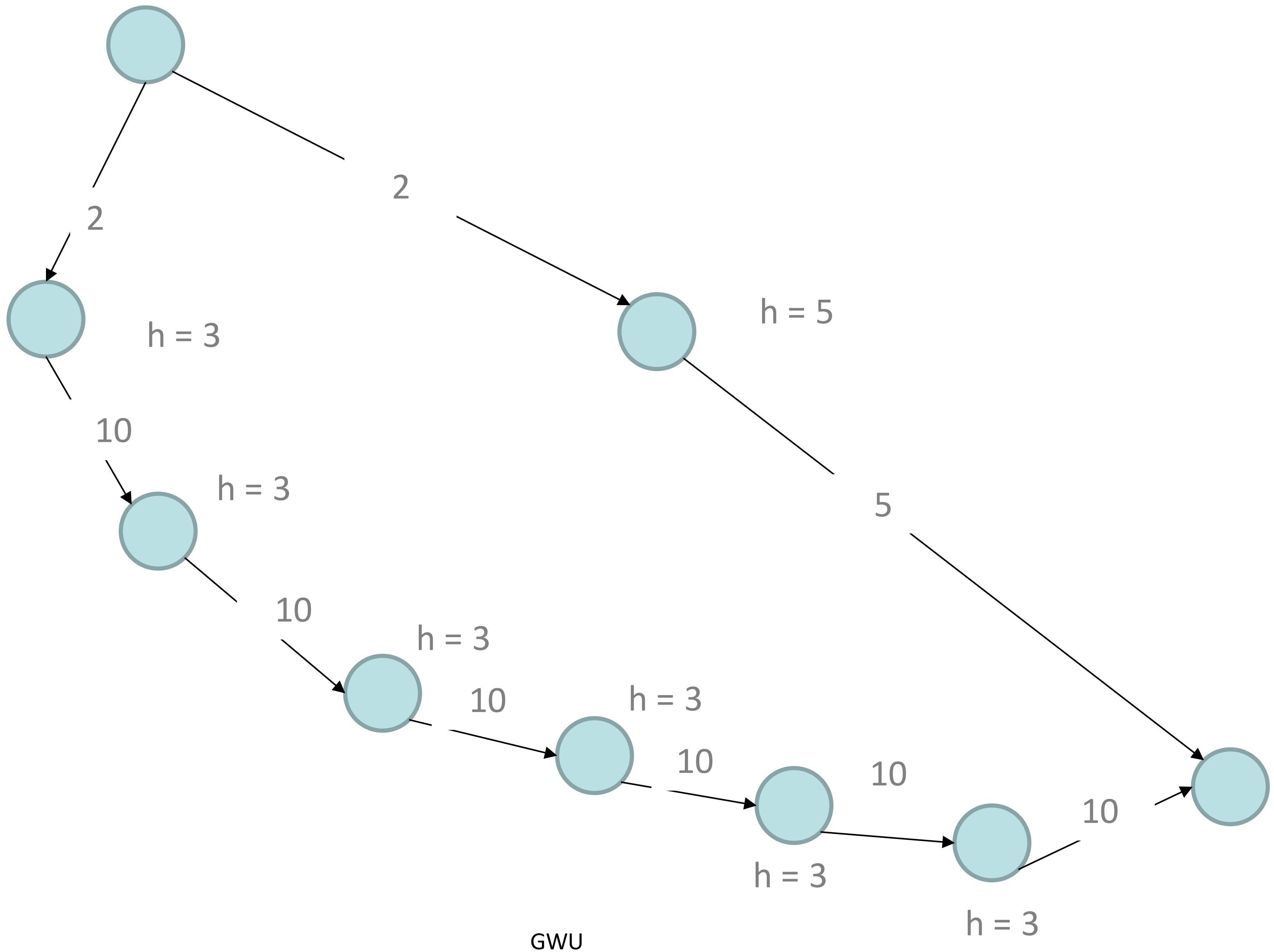
- Strategy: expand a node that you think is closest to a goal state
  - Heuristic: estimate of distance to nearest goal for each state



- A common case:
  - Best-first takes you straight to the (wrong) goal

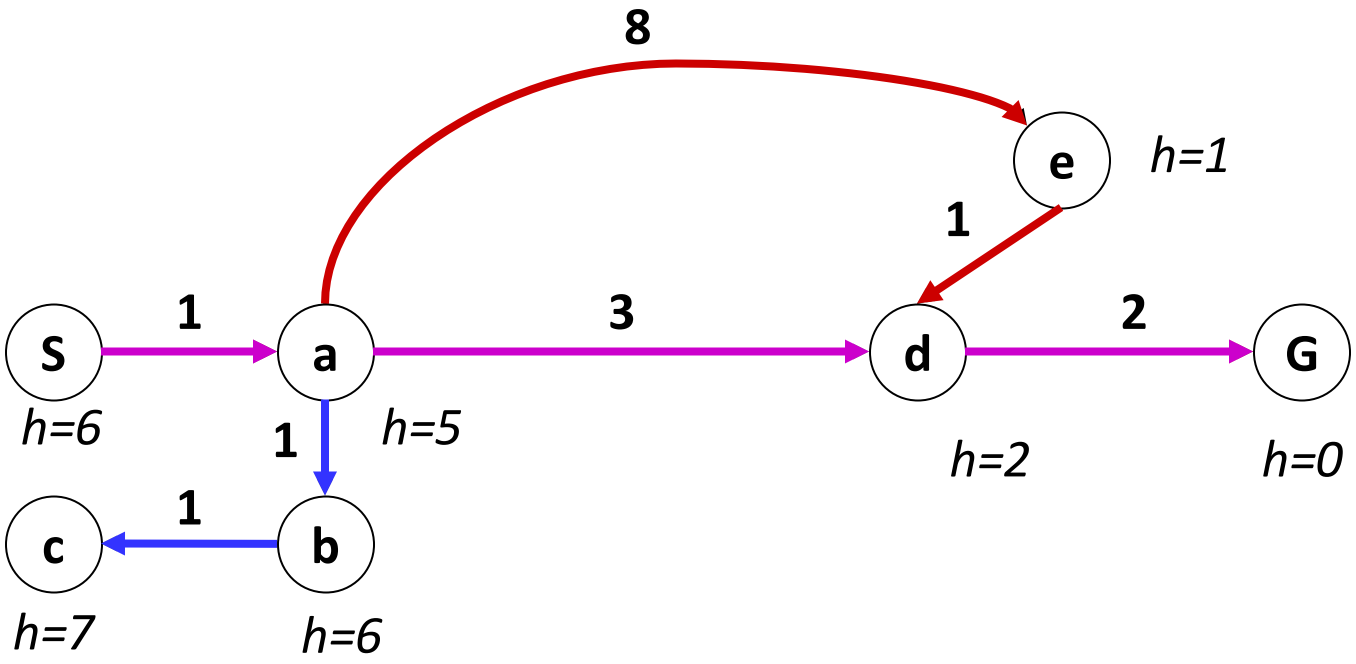


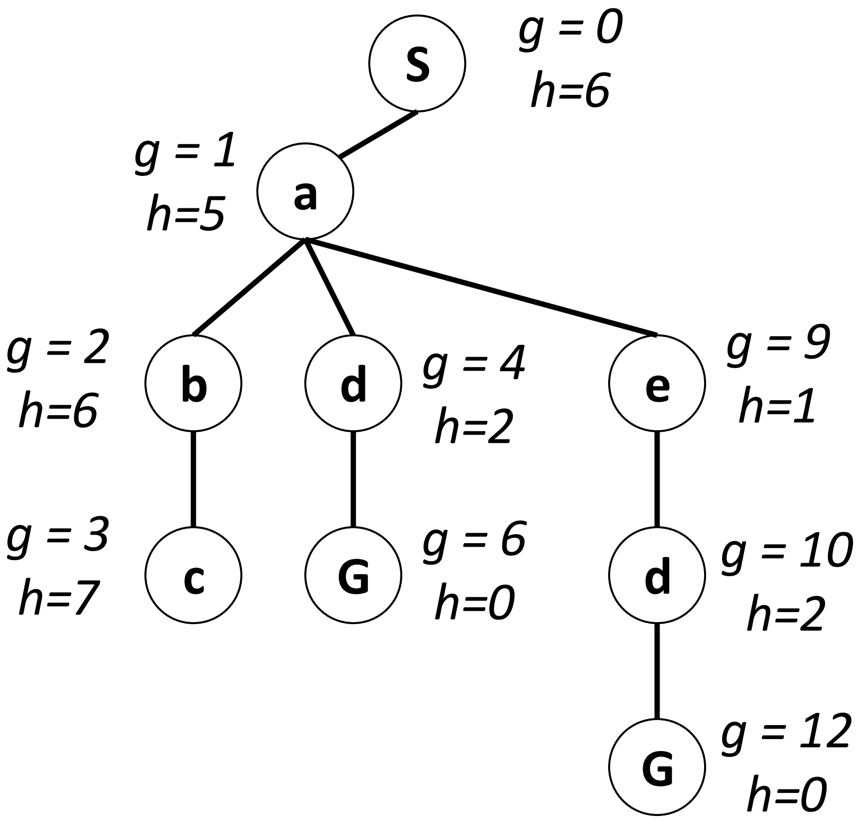
- Worst-case: like a badly-guided DFS


2
2
h = 3
h = 5
10
h = 3
5
10
h = 3
10
h = 3
10
10
10
h = 3
h = 3

Comparing UCS and Greedy

- Uniform-cost orders by path cost, or *backward cost* g(n)
- Greedy orders by goal proximity, or *forward cost* h(n)





Pros and Cons (and What to Do)

- Neither approach is bad
- UCS doesn't take direction into account
- Greedy doesn't take the past covered distance into account.

- **We should combine the two ideas.**

Key Idea

A* - NEW ALGORITHM

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

- *Why is this a good strategy?*

Key Idea

ADMISSIBILITY (OF HEURISTICS)

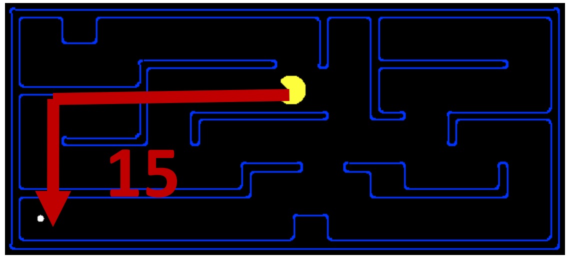
Admissible Heuristics

- 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, that is, h(n) does not over estimate the cost/distance.

- Examples:



4

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

Consistent Heuristics vs. Admissible Heuristics

- Consistency: heuristic “arc” cost ≤ actual cost for each arc

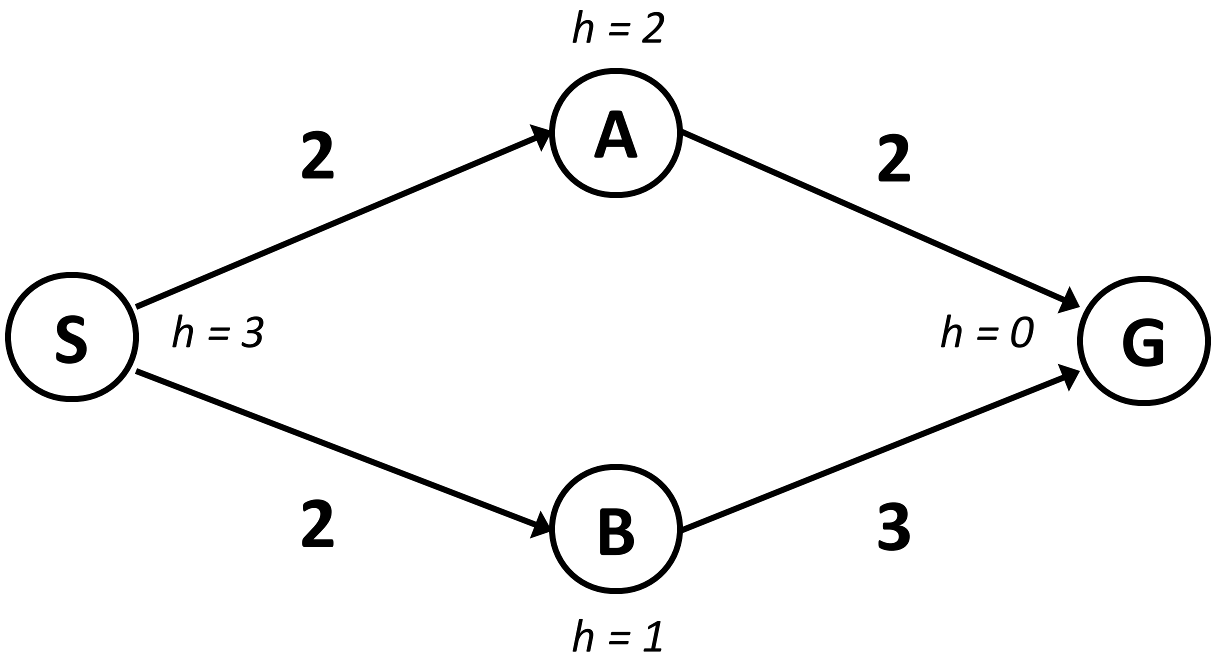
$$h(A) \le cost(A \text{ to } B) + h(B)$$

$$d(A,G) \le c(A,B) + d(B,G)$$

- Consistent = Monotone
- All consistent heuristics are admissible.
- Reverse is not true.
  - That means, there can be heuristics that ARE admissible, but ARE NOT consistent.

When should A* terminate?

- Should we stop when we enqueue a goal?



- No: only stop when we dequeue a goal

Key Idea

OPTIMALITY OF A* TREE SEARCH

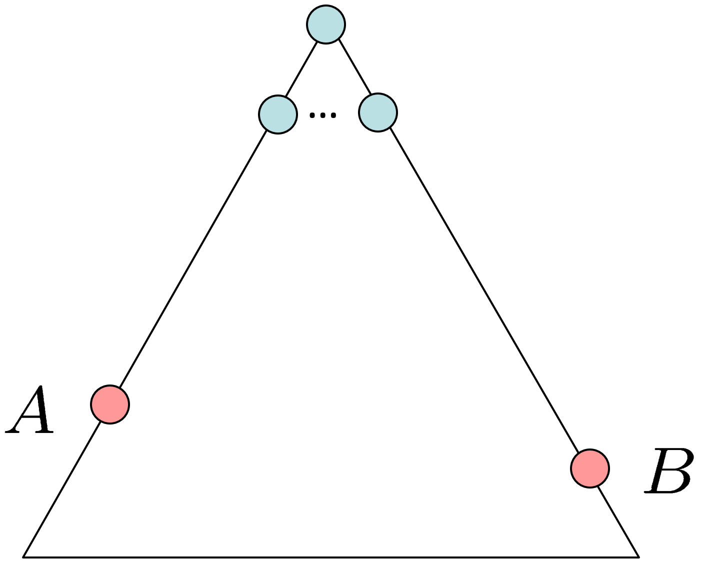
Optimality of A* Tree Search

Assume:

- A is an optimal goal node
- B is a suboptimal goal node
- h(x) is an admissible heuristic

Claim:

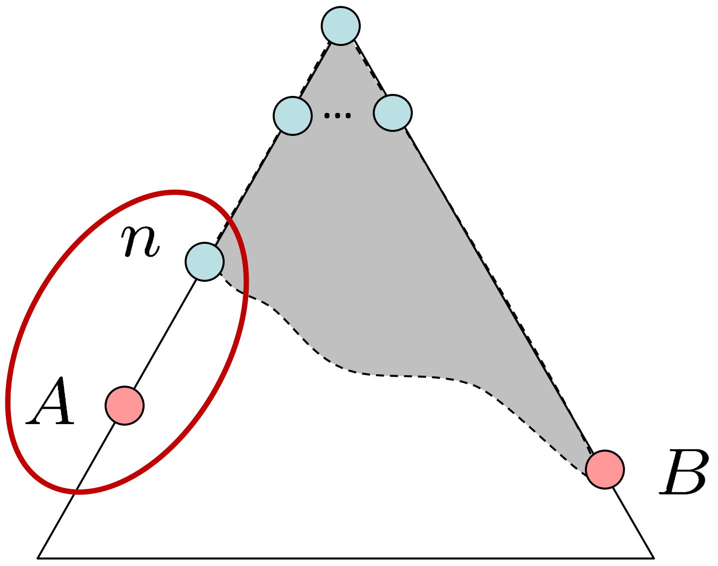
- A will exit the fringe before B



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)

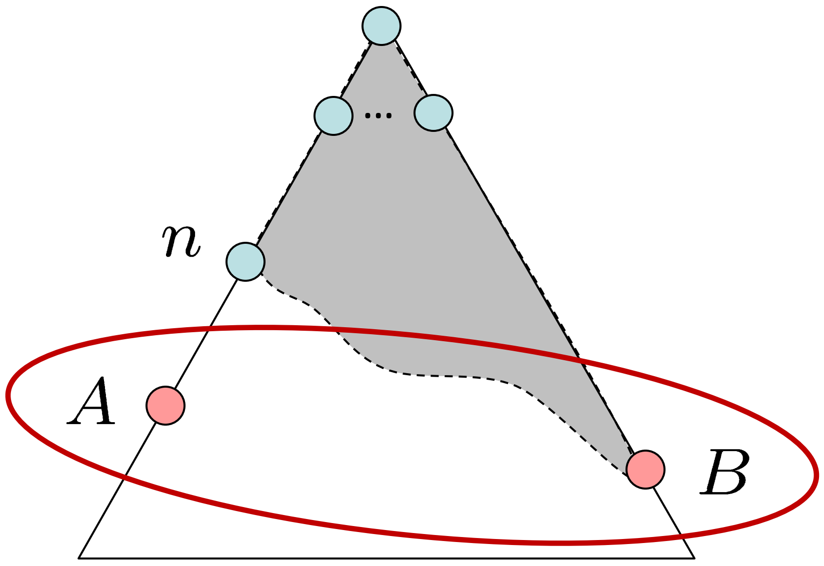


$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \leq g(A)$	Admissibility of h
$g(A) = f(A)$	h = 0 at a goal

Optimality of A* Tree Search: Blocking

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)



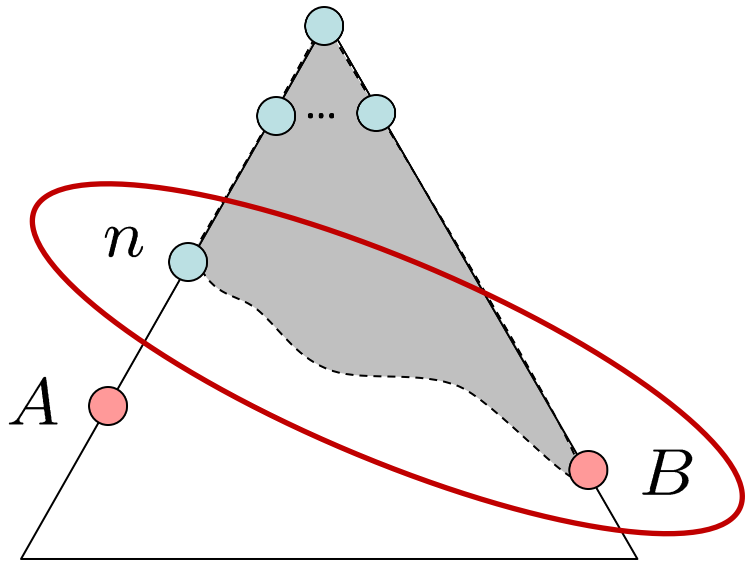
$g(A) < g(B)$ B is suboptimal

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

Optimality of A* Tree Search: Blocking

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

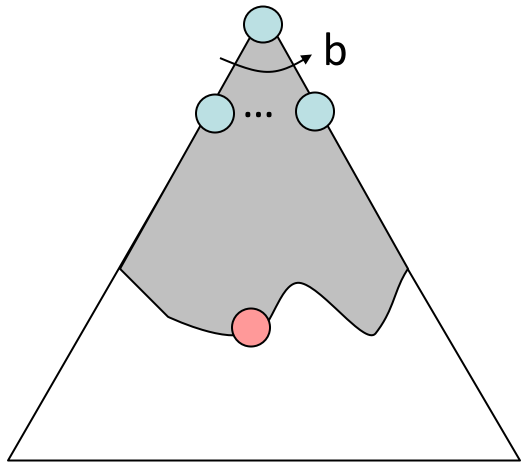


$$f(n) \leq f(A) < f(B)$$

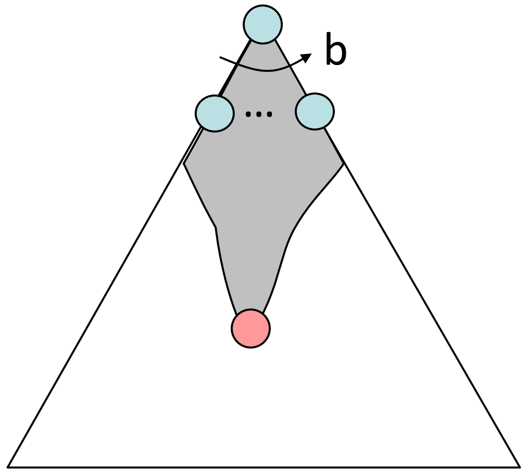
PROPERTIES OF A* TREE SEARCH

Properties of A*

Uniform-Cost

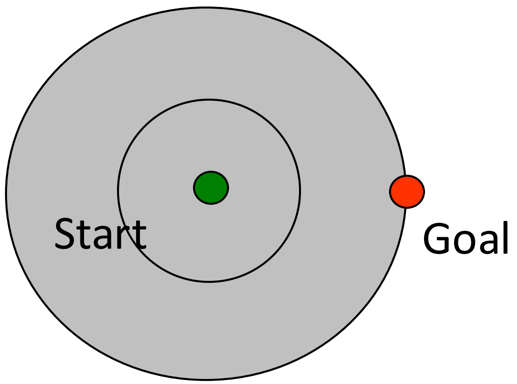


A*

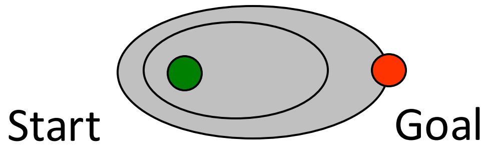


UCS vs. A* Contours

- Uniform-cost expands equally in all "directions"



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



Comparison

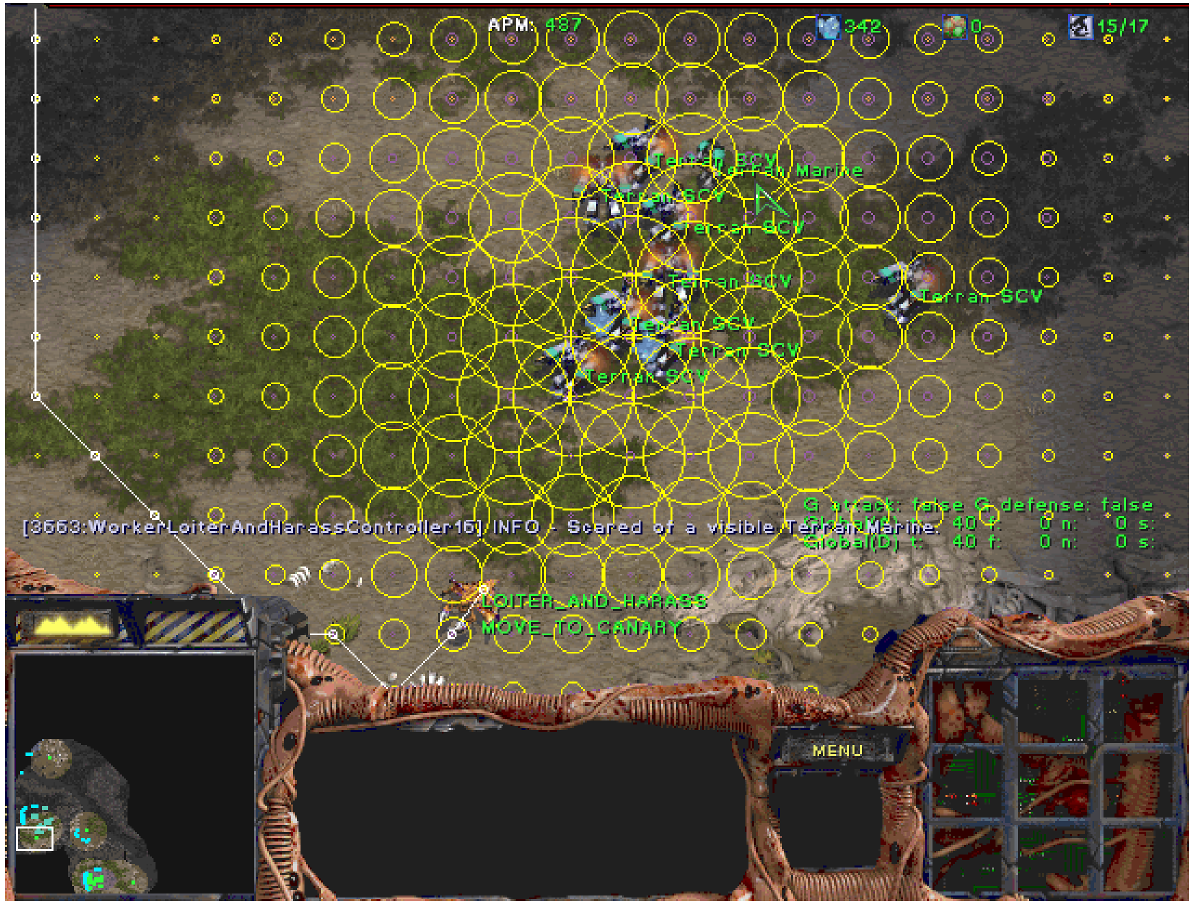
Greedy

Uniform Cost

A*

A* Applications

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



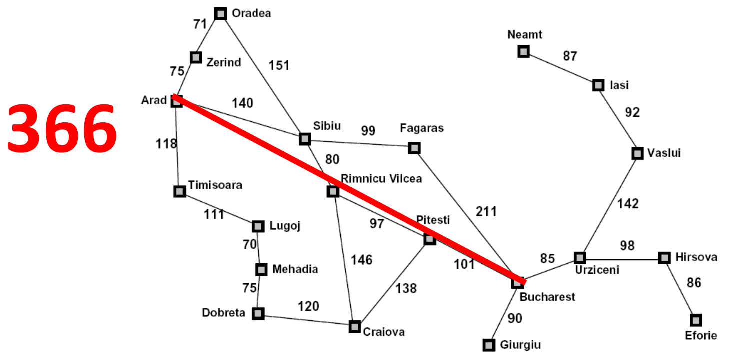
Creating Heuristics

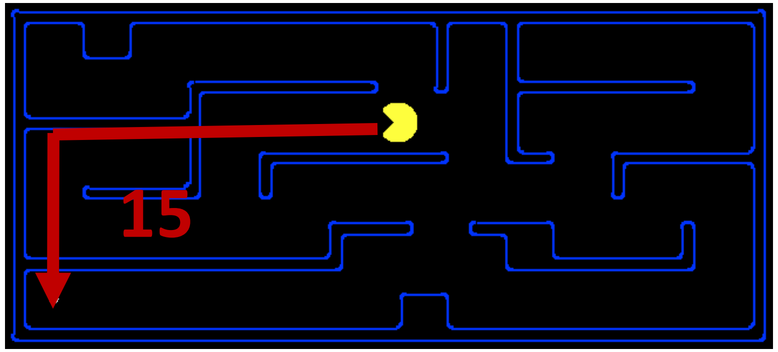
- The main point of improvement!!

Creating Admissible Heuristics

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

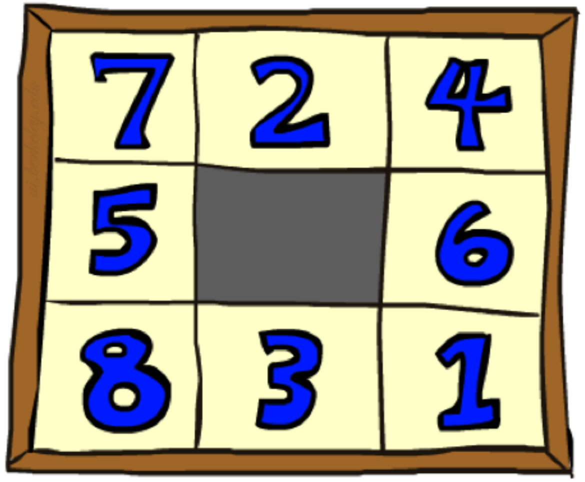
- Often, admissible heuristics are solutions to *relaxed problems,* where new actions are available



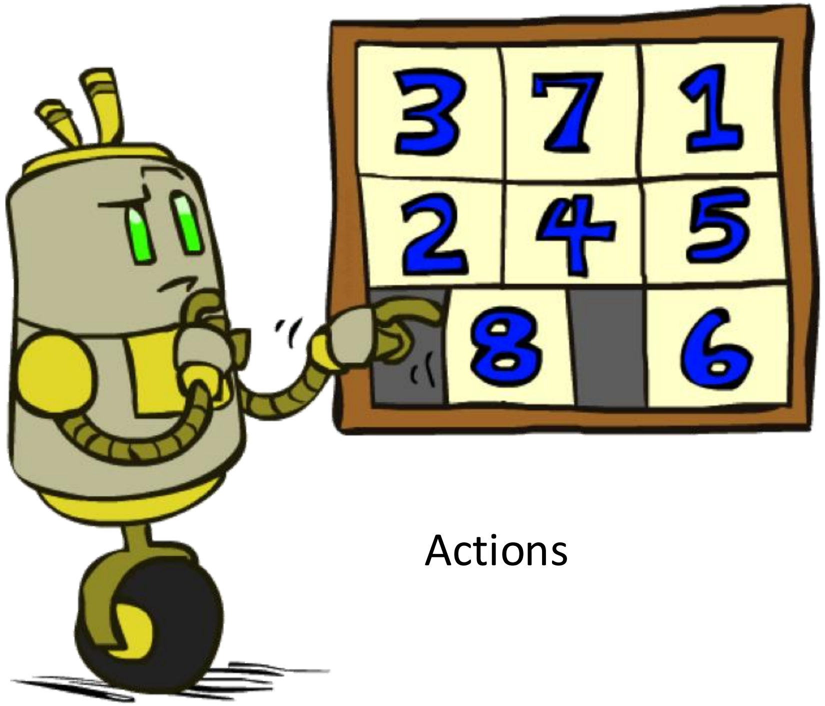


- Inadmissible heuristics are often useful too

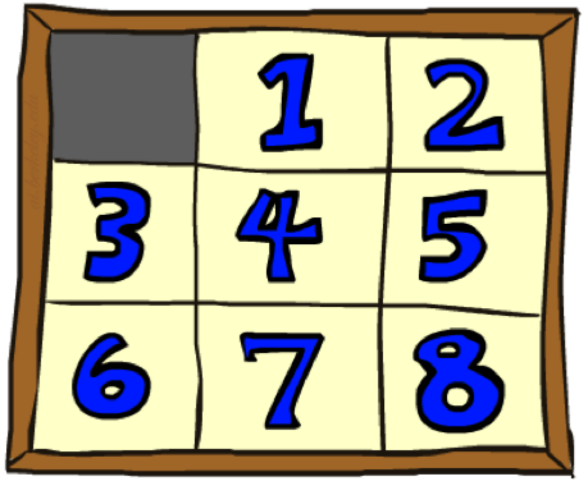
Example: 8 Puzzle



Start State



Actions

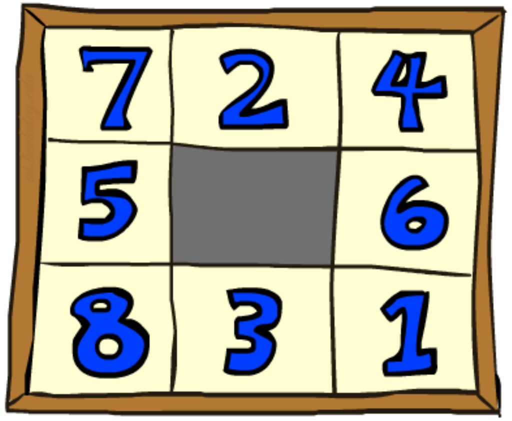


Goal State

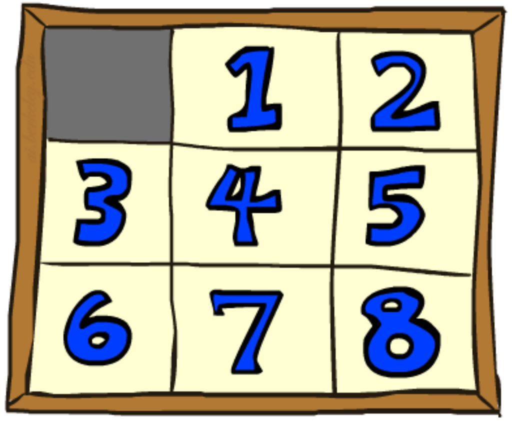
- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

8 Puzzle I

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



Start State

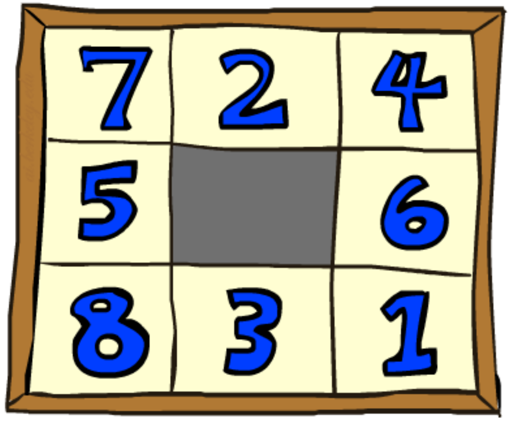


Goal State

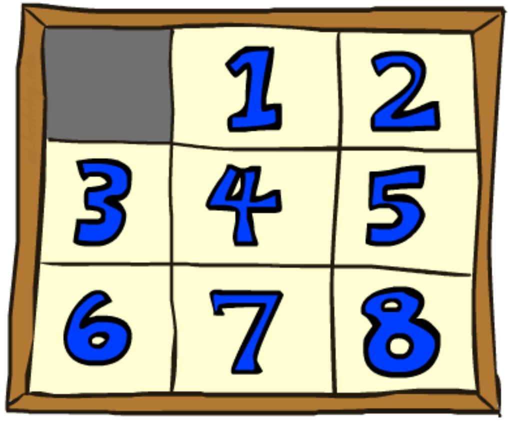
	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
A* TILES	13	39	227

8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
- Total *Manhattan* distance
- Why is it admissible?
- h(start) = 3 + 1 + 2 + … = 18



Start State

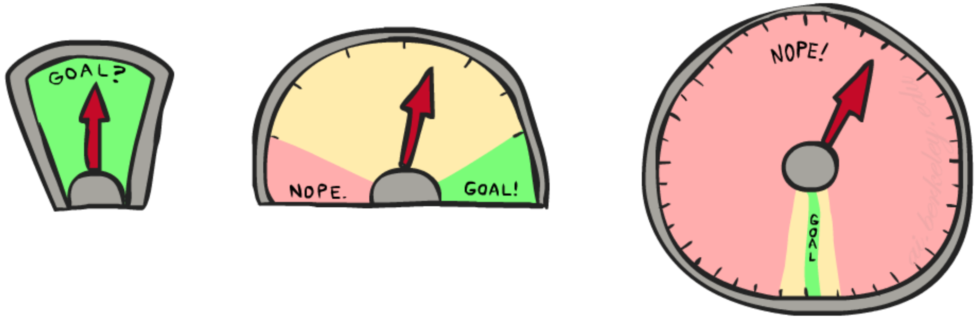


Goal State

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

8 Puzzle III

- How about using the *actual cost* as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?



- With A*: a trade-off between quality of estimate and work per node
  - As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

Semi-Lattice of Heuristics

Trivial Heuristics, Dominance

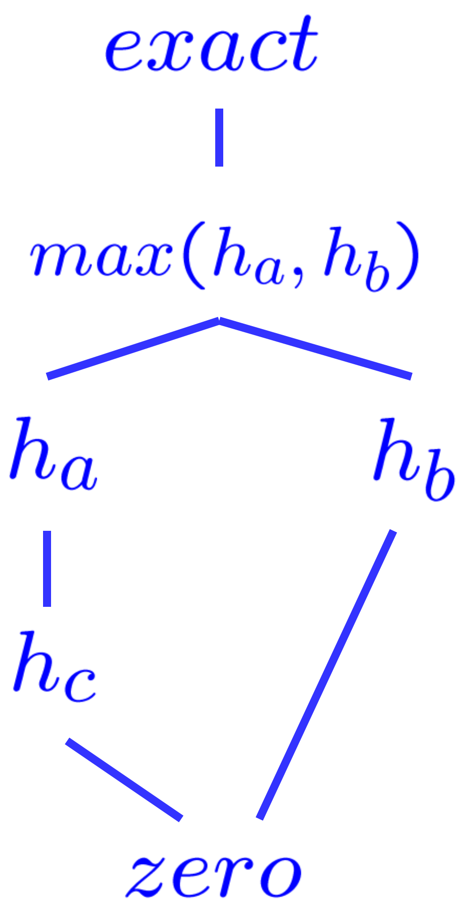
- Dominance: $h_a \geq h_c$ if

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

- Heuristics form a semi-lattice:
  - Max of admissible heuristics is admissible

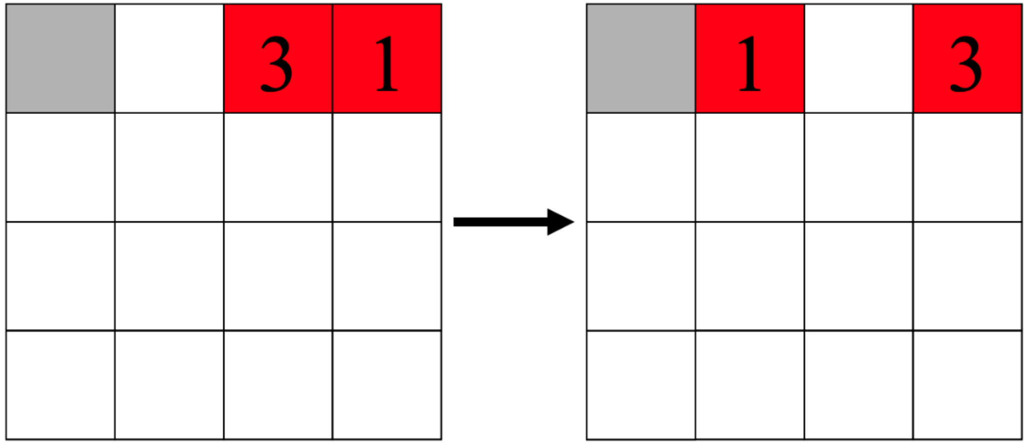
$$h(n) = max(h_a(n), h_b(n))$$

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



8 Puzzle, Beyond Manhattan Distance

- Even if using A* (or other algorithms such as IDA*) along with a heuristic such as Manhattan Distance, larger puzzles, such as 24 puzzles are still untenable.
- This is because Manhattan Distance does not take into account linear conflicts. For example:



- Manhattan Distance is 4, but tiles 1 and 3 interfere with each other.
- [Hansson, Mayer, and Yung, 1991] show that given two tiles in their goal row, but reversed in position, additional vertical moves can be added to Manhattan distance.
- So, 4 + 2 = 6 in this case.

8 Puzzle, Beyond Manhattan Distance

- So, using A* and using the New Heuristic (Manhattan + Vertical Moves), larger puzzles, such as 24 puzzles are still untenable.
- We need to use a pattern database.
- A pattern database is a complete set of such positions, with associated number of moves
- A 7-tile pattern database for the Fifteen Puzzle contains 519 million entries.

Using Pattern Database and Semi-Lattice

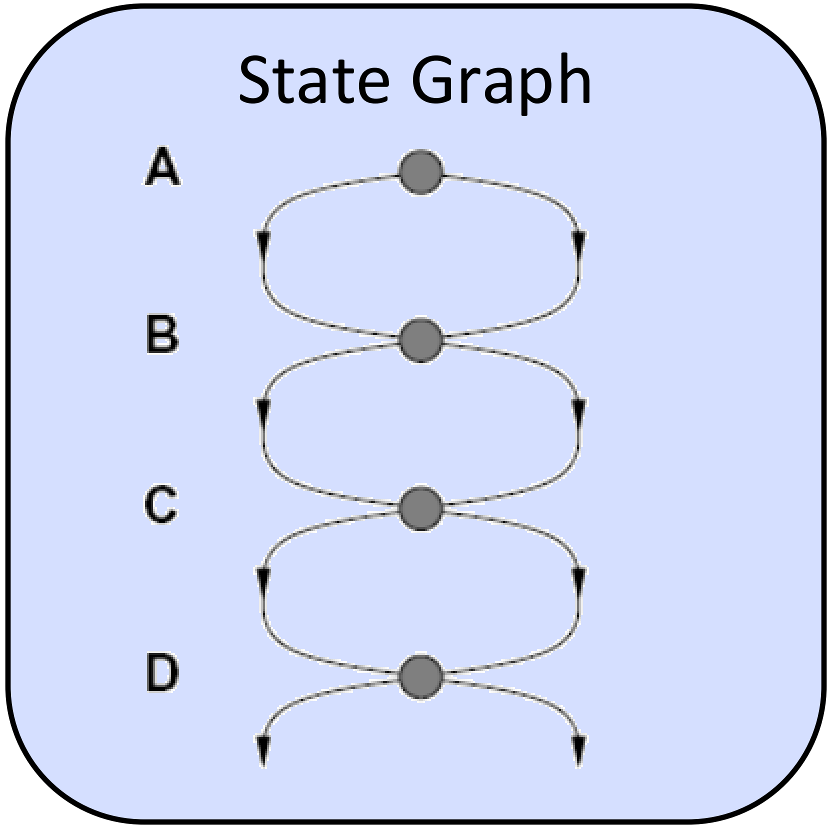
- From a given 15-puzzle we may recognize two different patterns using two different sets of tiles.
- If one pattern suggests a distance of 20 and the other pattern suggests a distance of 30, we can take the maximum (30).

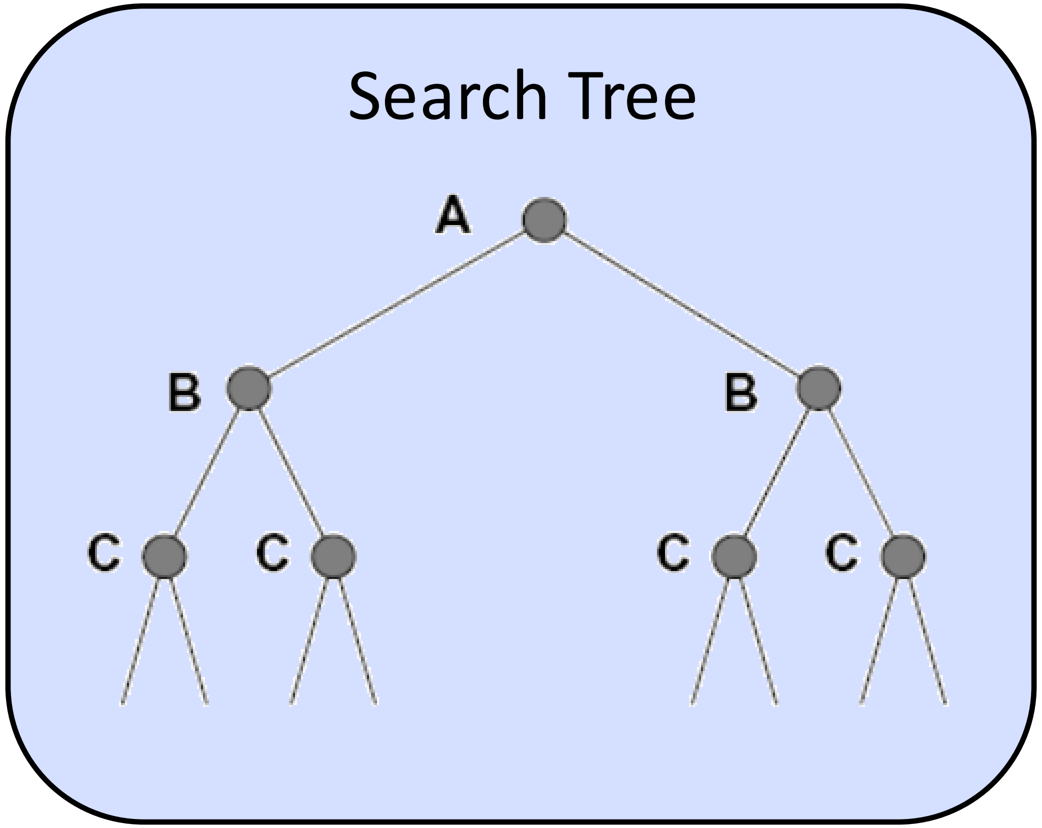
That magical time, when you realize, I have been here!

GRAPH SEARCH

Tree Search: Extra Work!

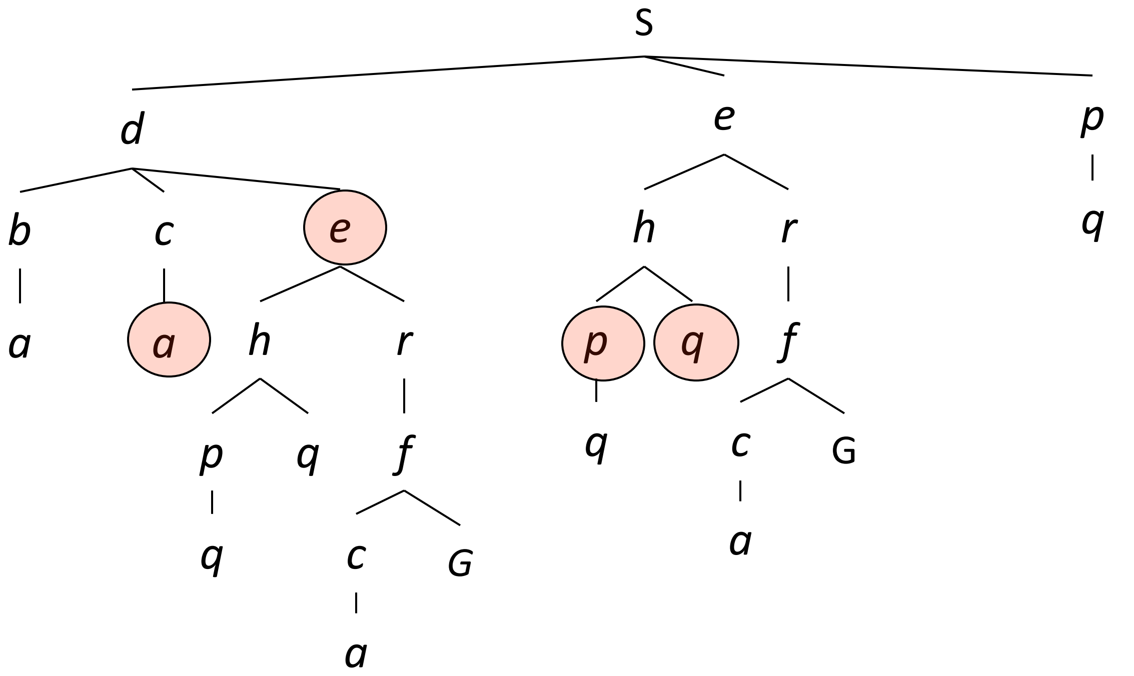
- Failure to detect repeated states can cause exponentially more work.





Graph Search

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

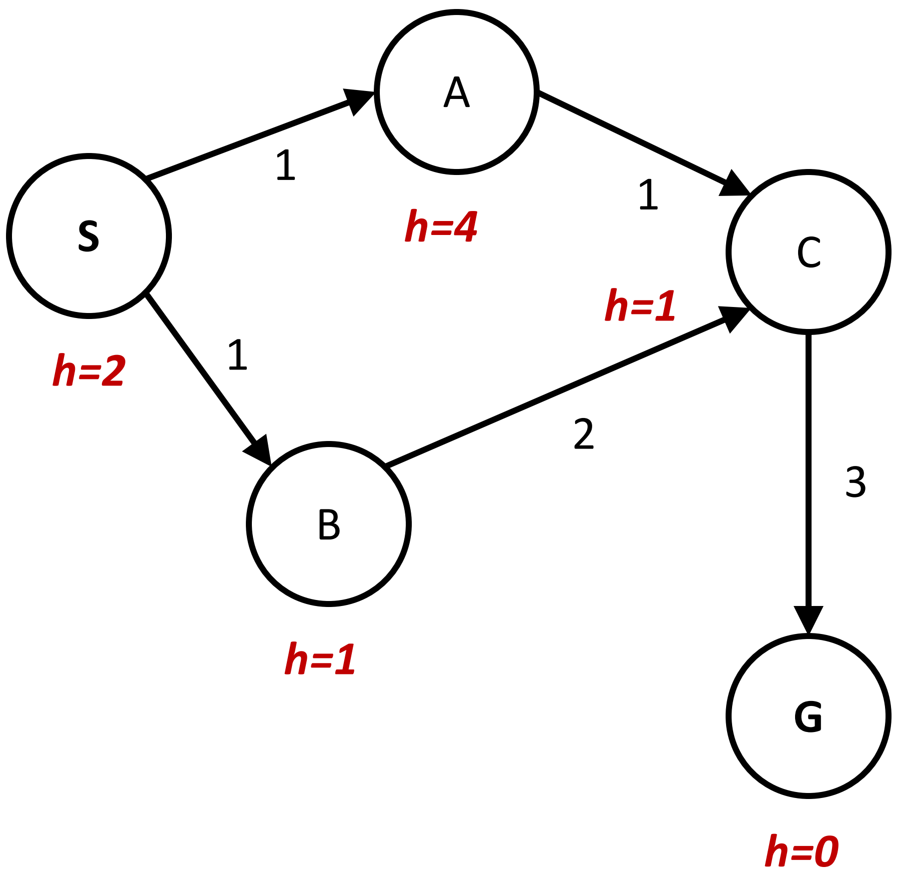


Graph Search

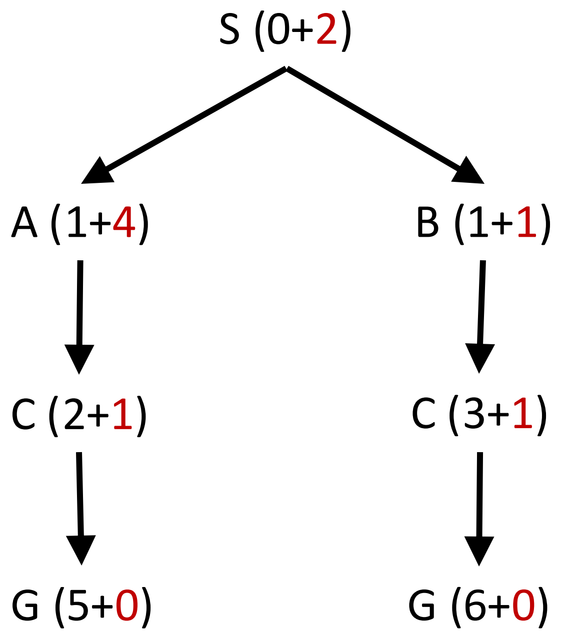
- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states (“closed set”)
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to closed set
- Important: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

A* Graph Search Gone Wrong?

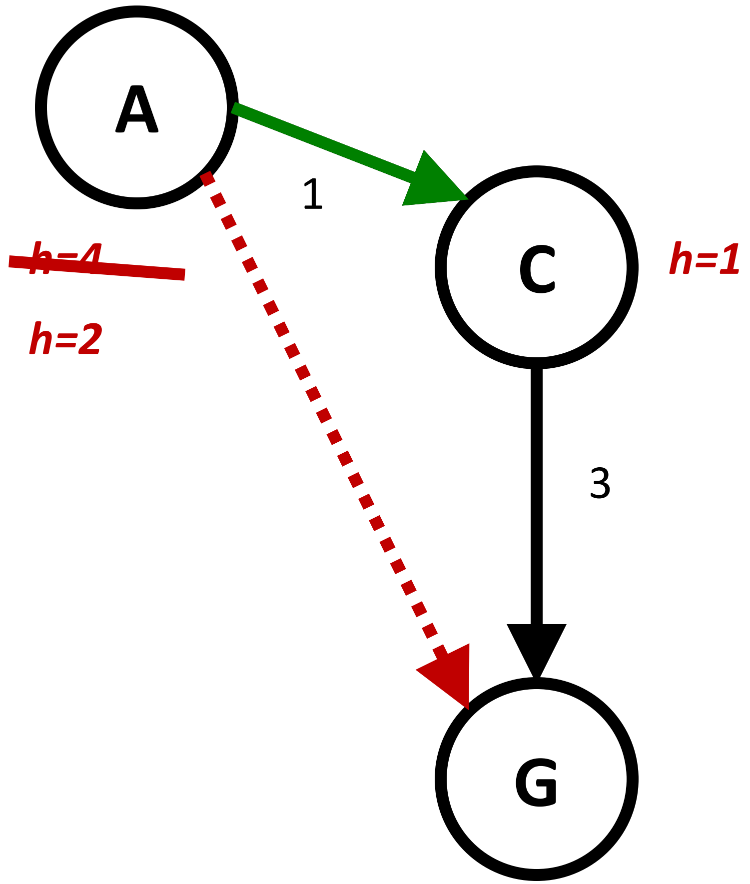
State space graph



Search tree



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)

- Consequences of consistency:
  - The f value along a path never decreases

    h(A) ≤ cost(A to C) + h(C)

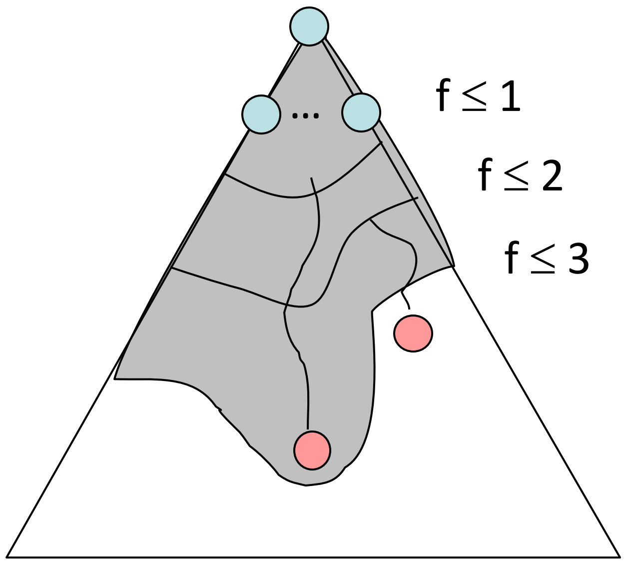
  - A* graph search is optimal

Optimality of A* Graph Search

- Why do we need consistency in this case?
- Why is admissibility not sufficient?

Optimality of A* Graph Search

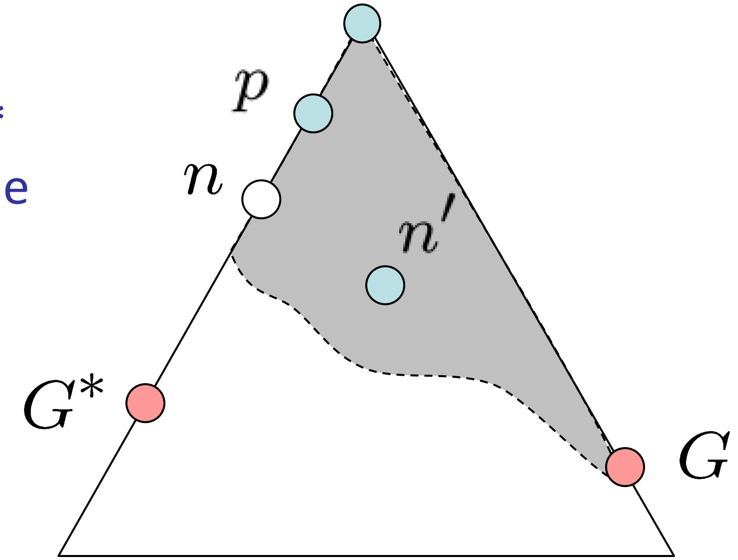
- Sketch: consider what A* does with a consistent heuristic:
  - Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)
  - Fact 2: For every state s, nodes that reach s optimally are expanded before nodes that reach s suboptimally
  - Result: A* graph search is optimal



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!



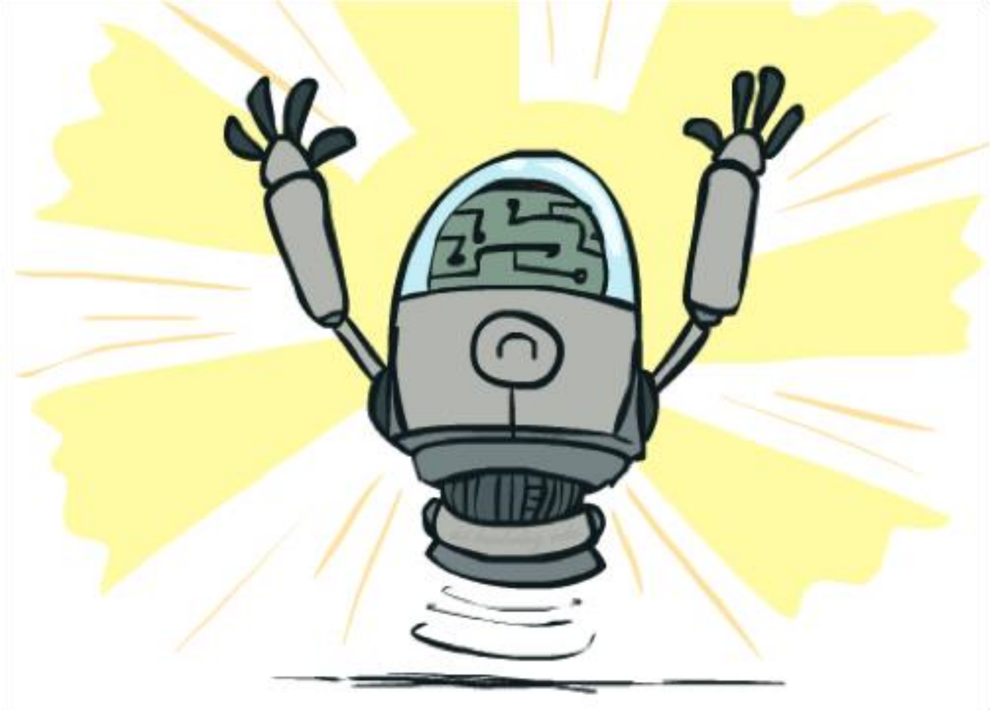
Optimality

- 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)

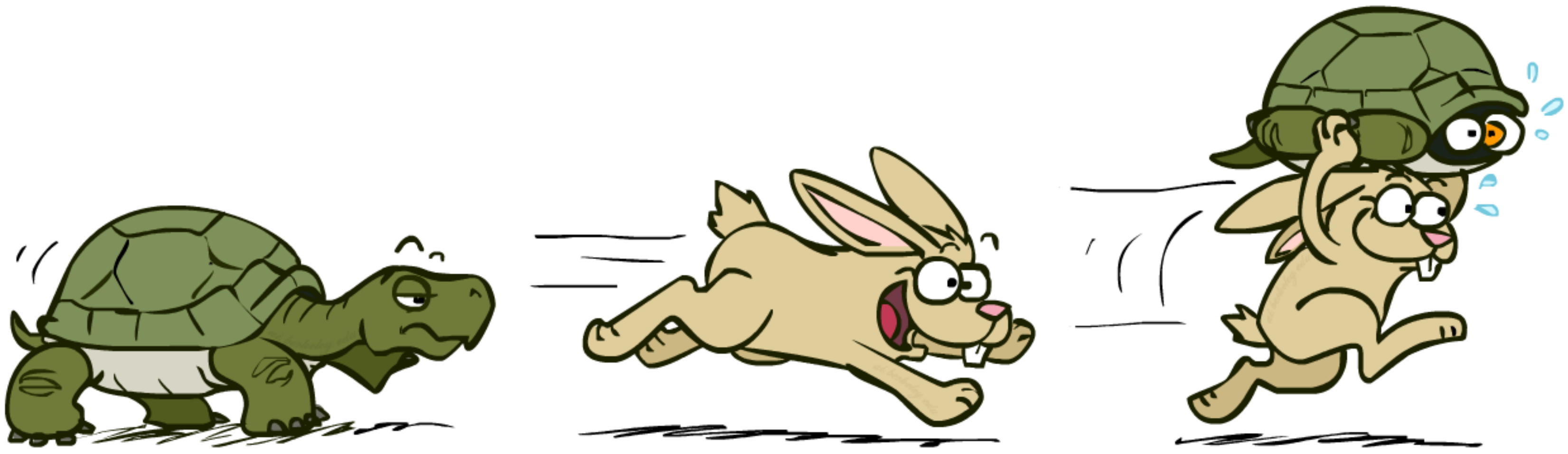
- Consistency implies admissibility

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



A*: Summary

- 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



Tree Search Pseudo-Code

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

Graph Search Pseudo-Code

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

10 AI Commandments

1. "No model is perfect, but some models are useful"
   General AI and ML
2. "The algorithms that forget their history are doomed to repeat it."
   Graph Search vs. Tree Search
3. "Ask not what the state can do for you, ask what you can do in that state."
   Successor function concept in search problems
4. **"Your direction is more important than your speed"**
   Informed search vs. uninformed search

That's It!

WE JUST WRAPPED UP INFORMED SEARCH! IT SEEMS, WE HAVE FOUND THE DIRECTION IN LIFE.