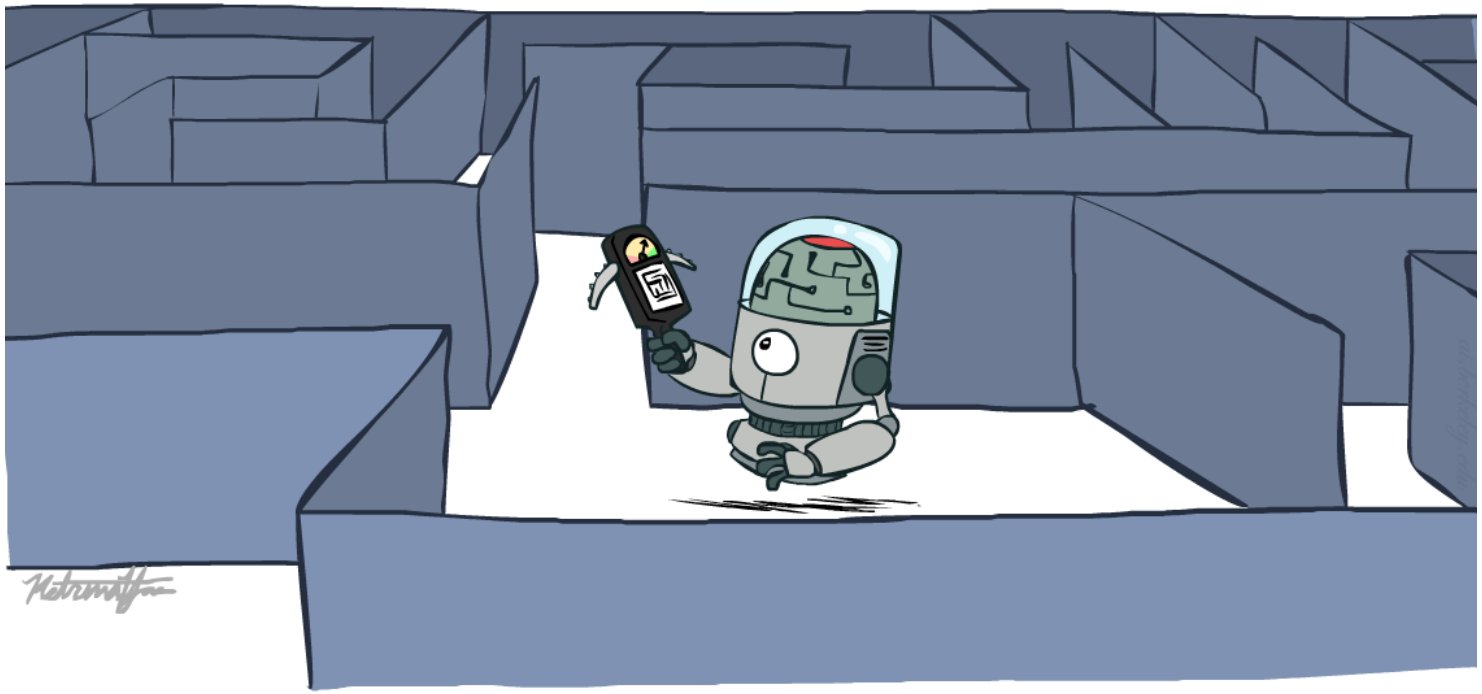
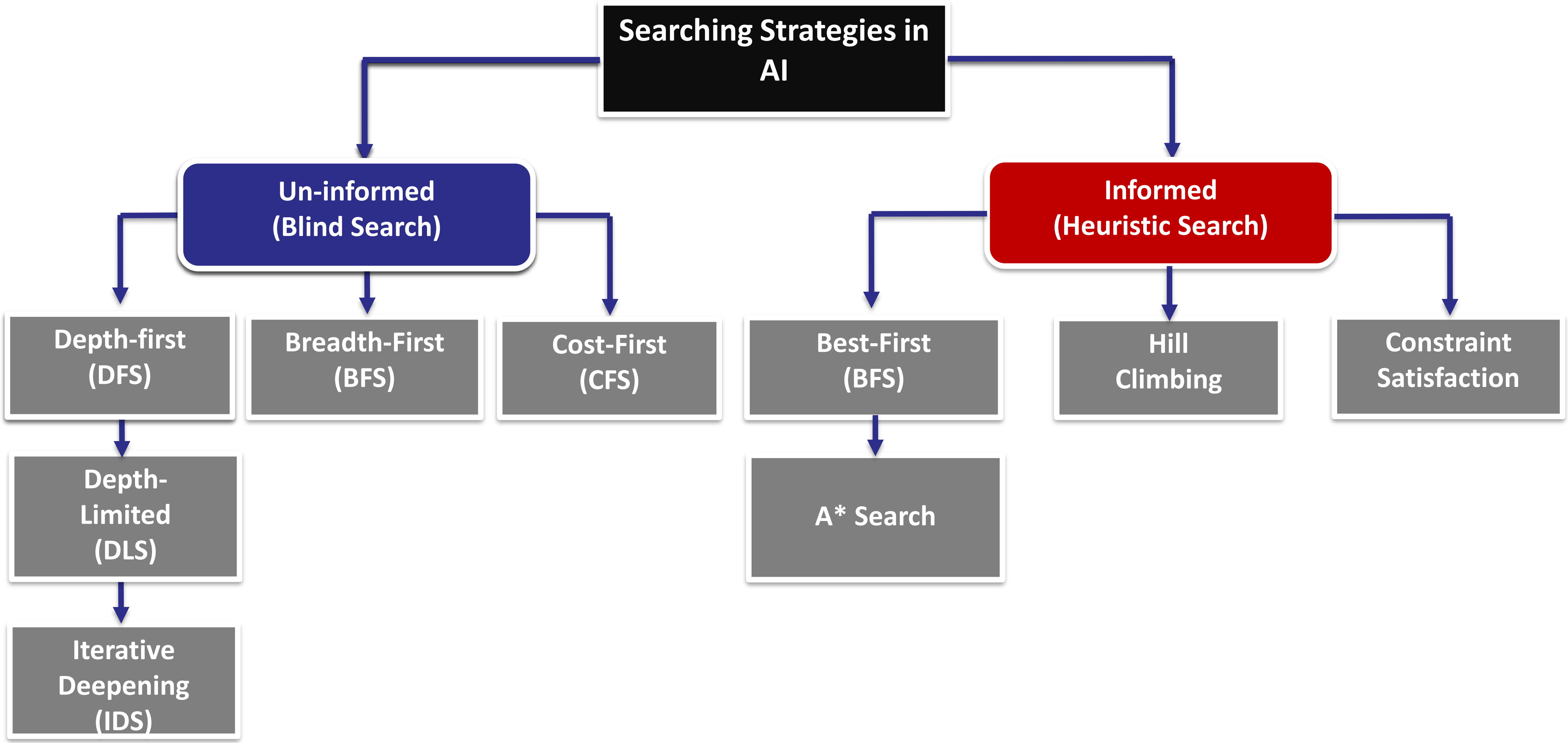
Artificial Intelligence

Informed Search



AI searching Strategies



A very large number of AI problems are formulated as search problems.

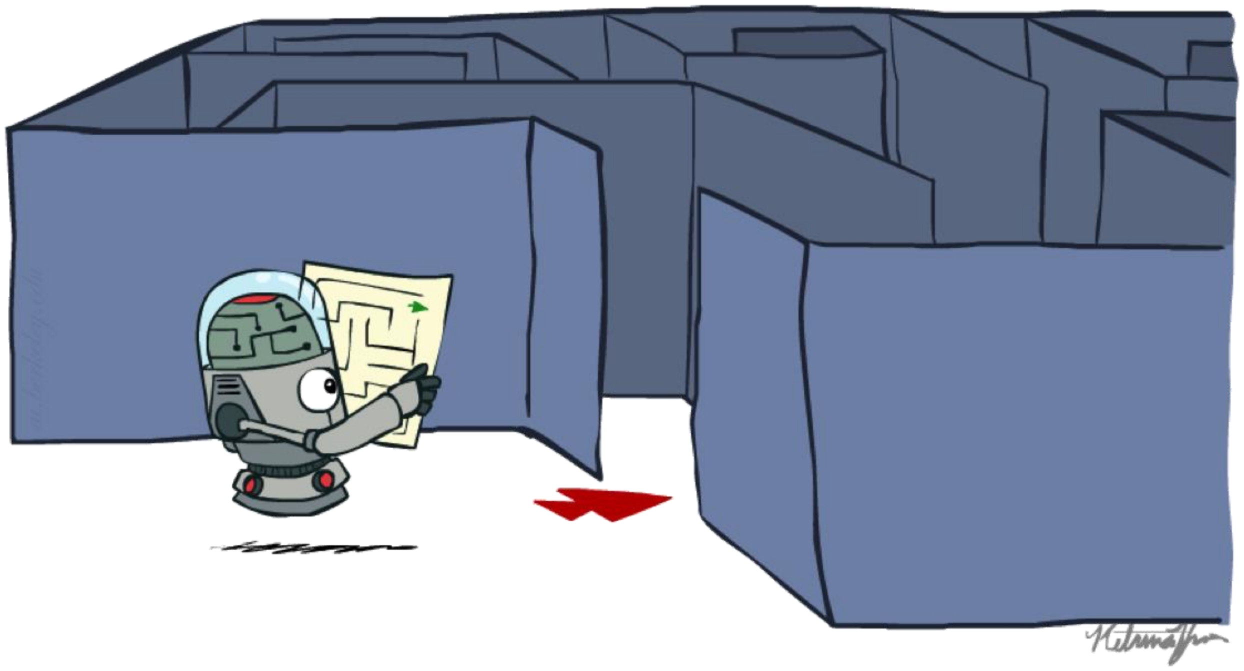
Today

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

- Graph Search

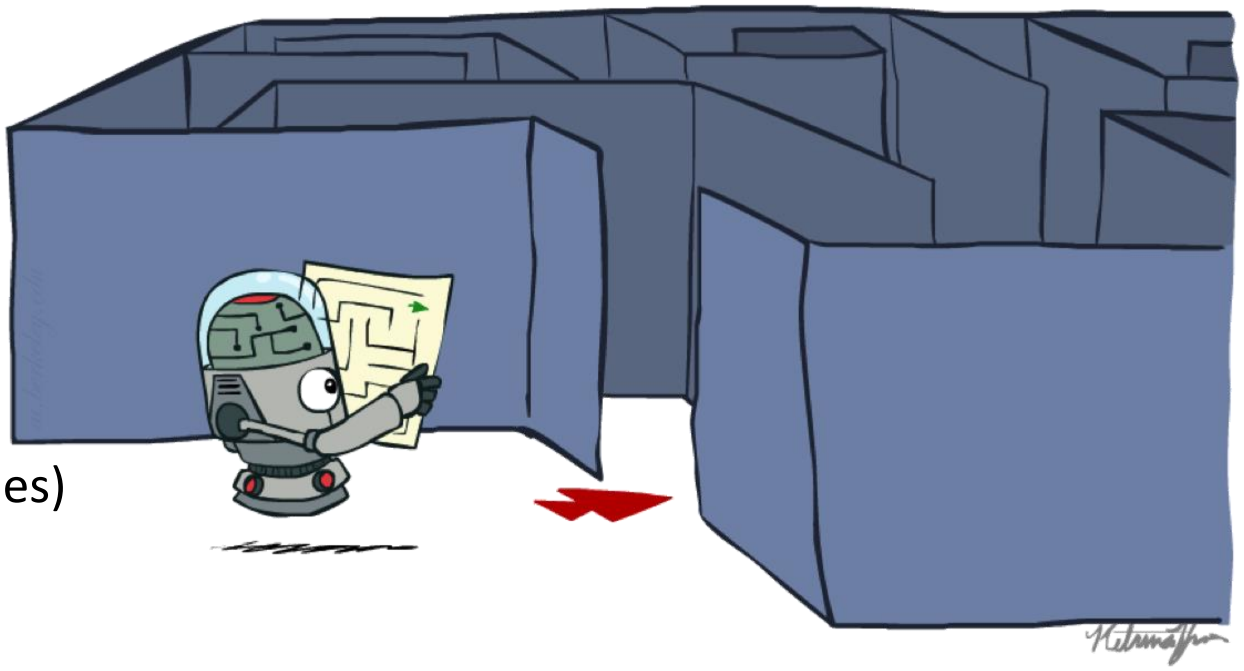
Recap: Search

1. s_0 ← sense/read initial state
2. GOAL? ← select/read goal test
3. Succ ← read successor function
4. solution ← **search**(Space, s_0, GOAL?)
5. perform(solution)

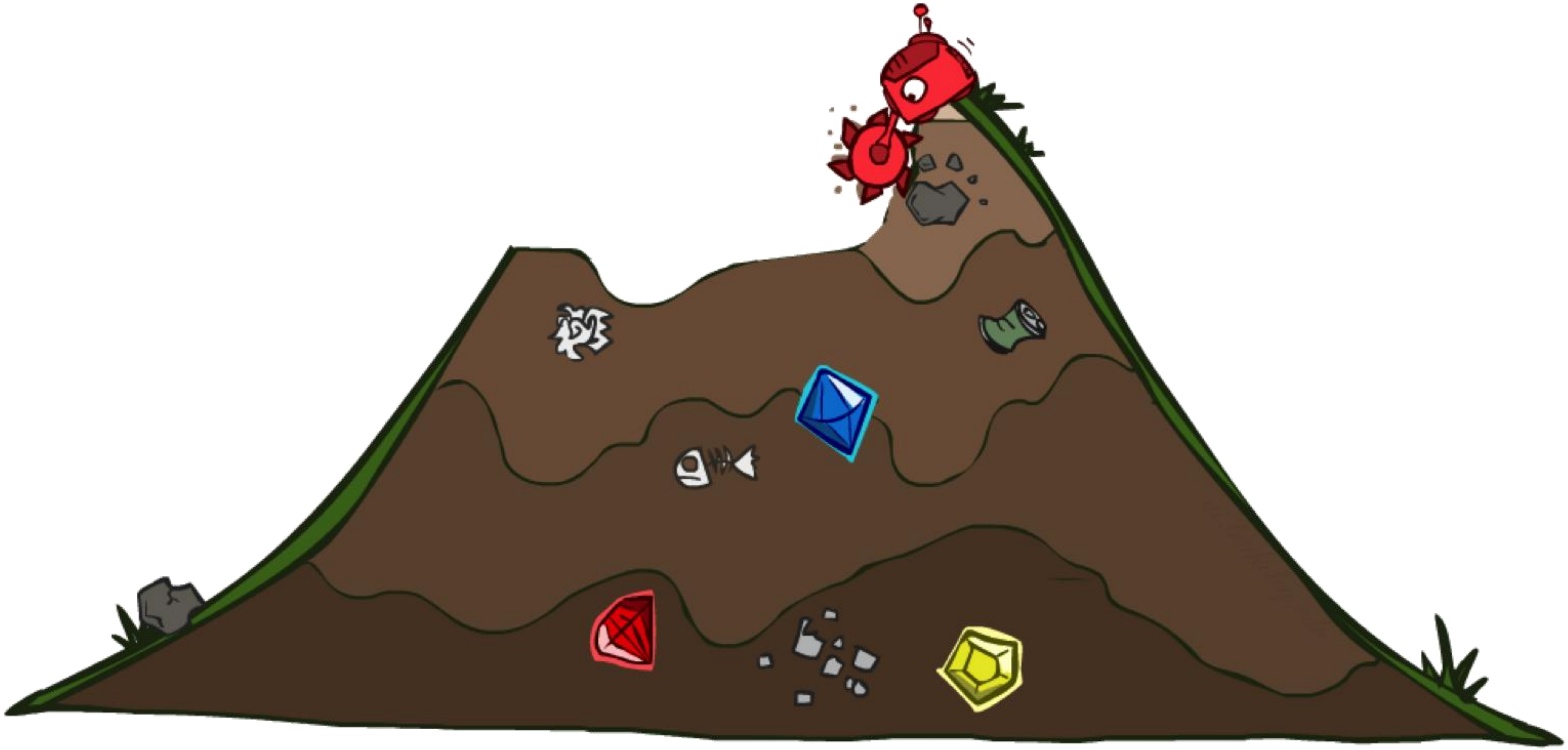


Recap: Search

- 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



Uninformed Search

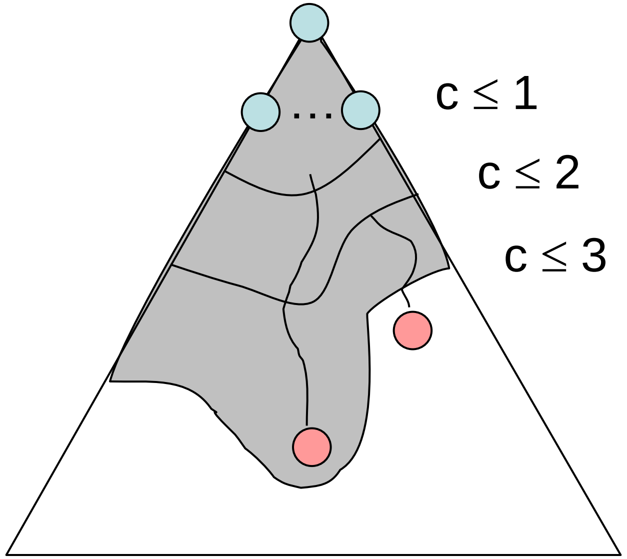


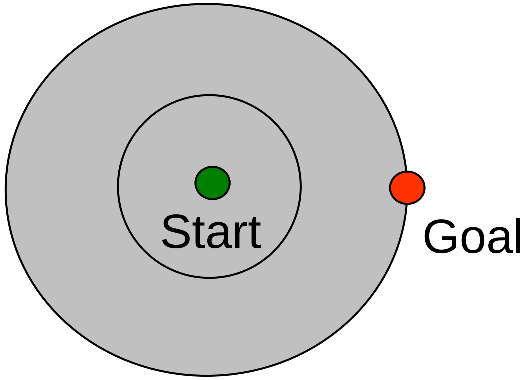
Uniform Cost Search

- Strategy: expand lowest path cost

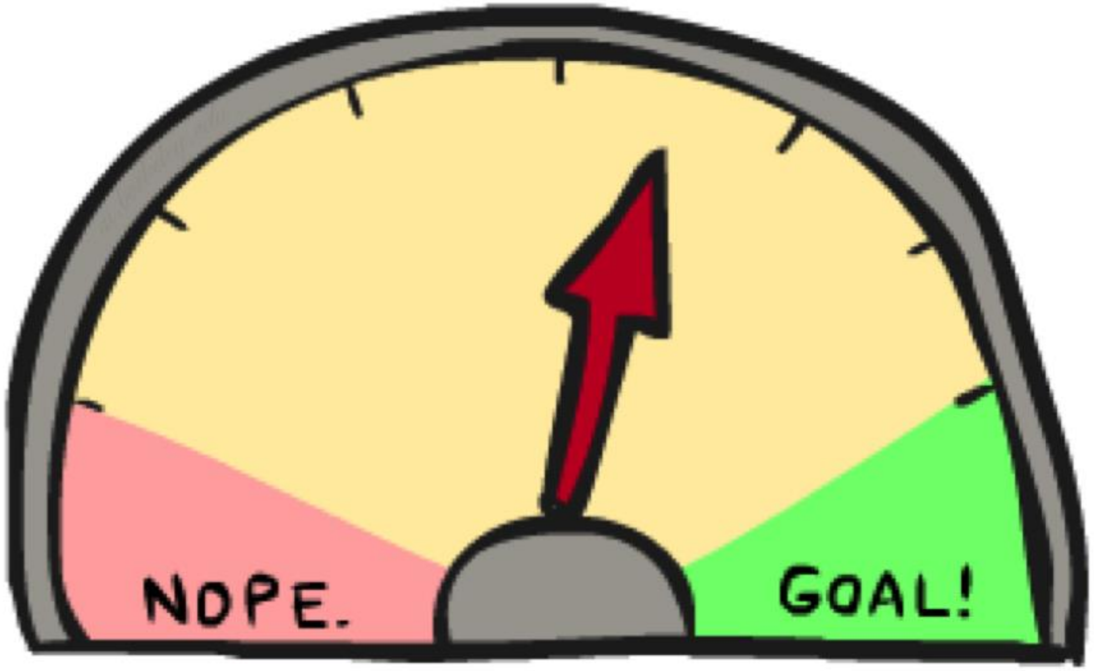
- The good: UCS is complete and optimal!

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



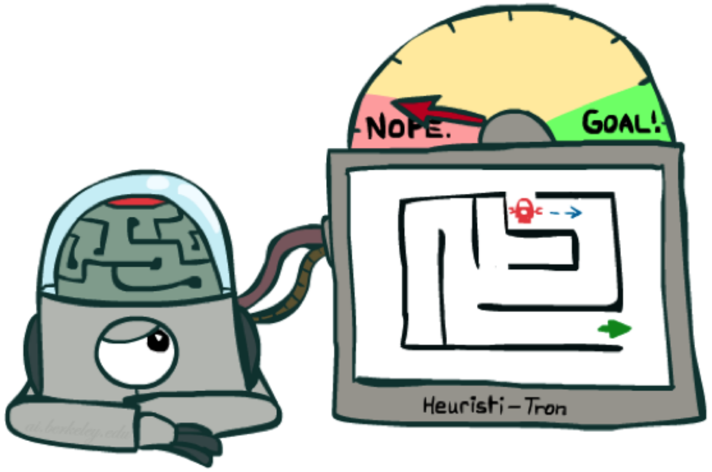


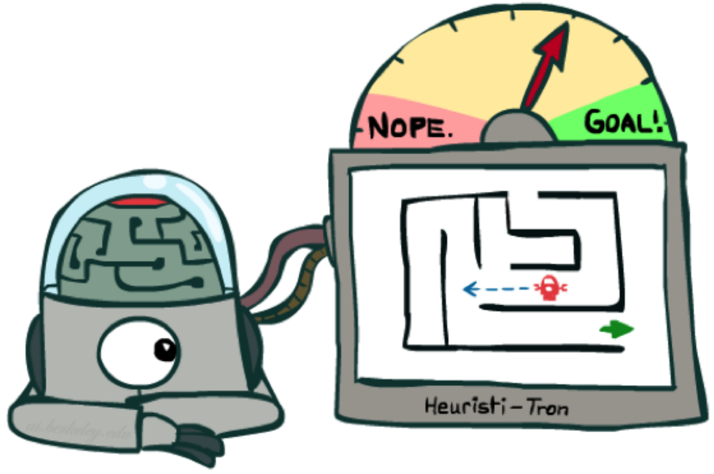
Informed Search



Search Heuristics

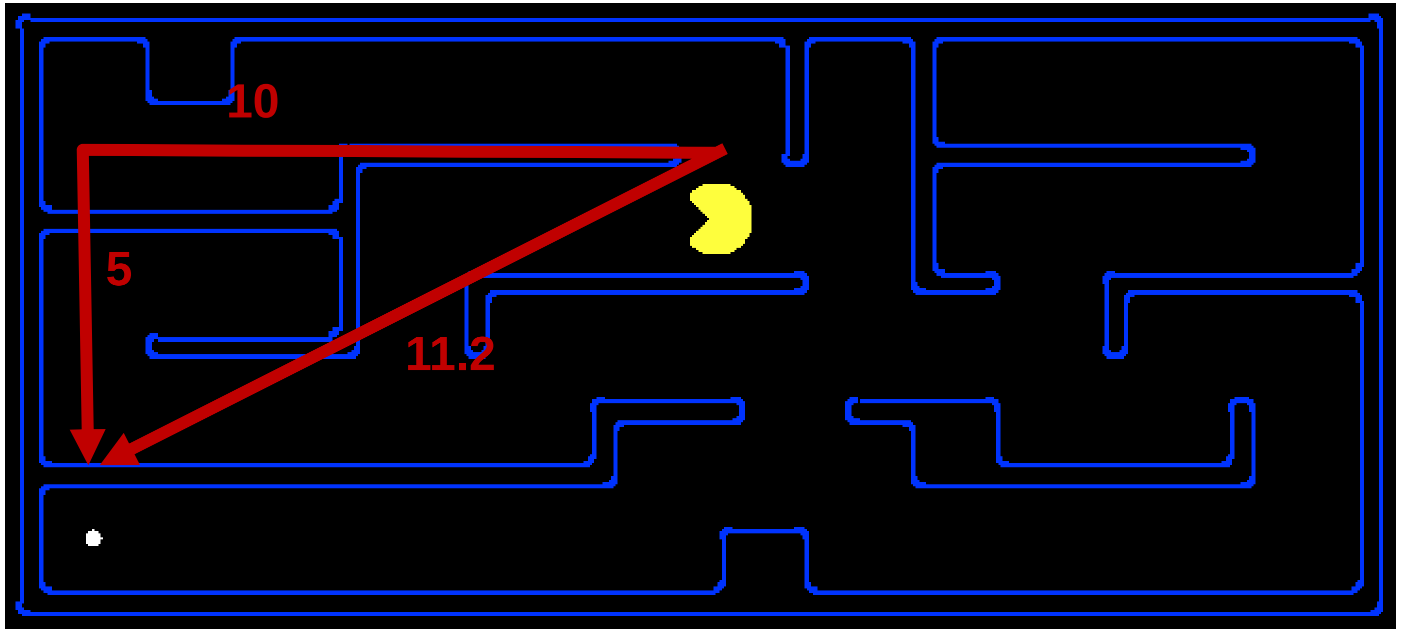
- A heuristic is:
  - A function that *estimates* how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance for pathing



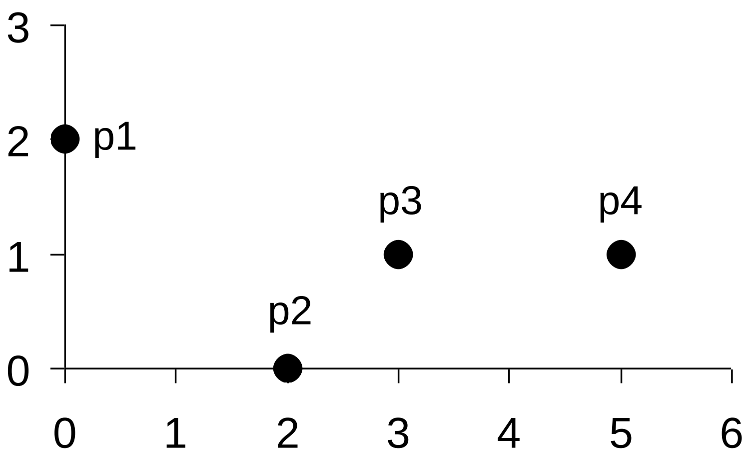


Distance

- Manhattan Distance (L_2 Norm): $d(x,y) = |x_{goal} - x_n| + |\mathrm{y_{goal}} - \mathrm{y_n}|$
- Euclidean Distance (L1 Norm): $d(x,y) = \sqrt{(x_{goal} - x_n)^2 + (y_{goal} - y_n)^2}$
- Chebychev Distance (L_∞ Norm): $d(x,y) = \max_{\forall i}\{|x_{goal} - x_n|, |\mathrm{y_{goal}} - \mathrm{y_n}|\}$



L_1, L_2, L_∞ Examples



point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

L1	p1	p2	p3	p4
p1	0	4	4	6
p2	4	0	2	4
p3	4	2	0	2
p4	6	4	2	0

L2	p1	p2	p3	p4
p1	0	2.828	3.162	5.099
p2	2.828	0	1.414	3.162
p3	3.162	1.414	0	2
p4	5.099	3.162	2	0

L_∞	p1	p2	p3	p4
p1	0	2	3	5
p2	2	0	1	3
p3	3	1	0	2
p4	5	3	2	0

Evaluation Function

- It exploits state description to estimate how “good” each search node is
- An evaluation function f maps each node N of the search tree to a real number $f(N) \geq 0$
  [Traditionally, f(N) is an estimated cost; so, the smaller f(N), the more promising N]

How to construct f?

- Typically, f(N) estimates:
  - either the cost of a path from N to a goal

    Then f(N) = h(N) → Greedy best-search
  - or the cost of a solution path through N

    Then f(N) = g(N) + h(N), where → A-Star Saarch
    - g(N) is the cost of the path from the initial node to N
    - h(N) is an estimate of the cost of a path from N to a goal node

- But there are no limitations on f. Any function of your choice is acceptable. But will it help the search algorithm?

Greedy Search

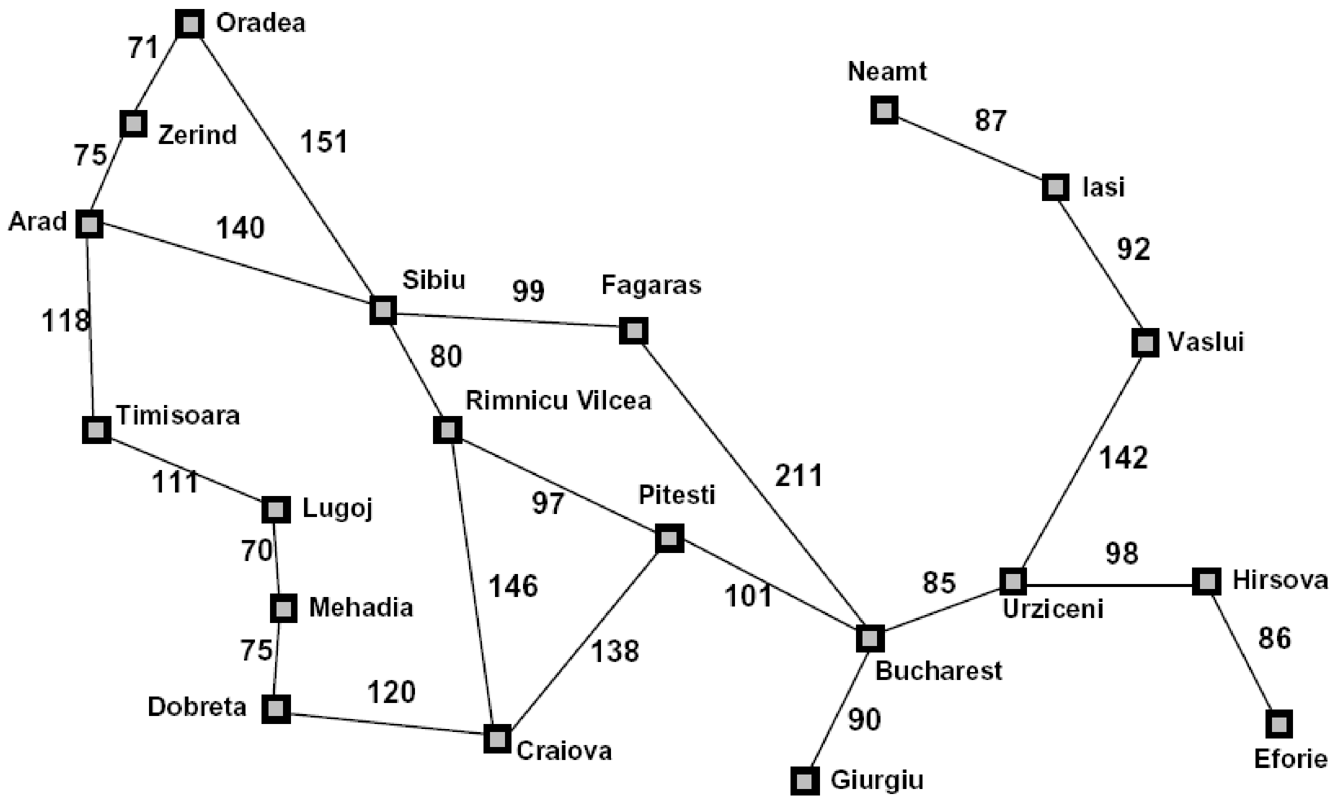
Best-First Search

- An evaluation function f maps each node N of the search tree to a real number

$$f(N) = h(N) \geq 0$$

- Best-first search sorts the FRINGE in increasing f
  [Arbitrary order is assumed among nodes with equal f]

Example: Heuristic Function



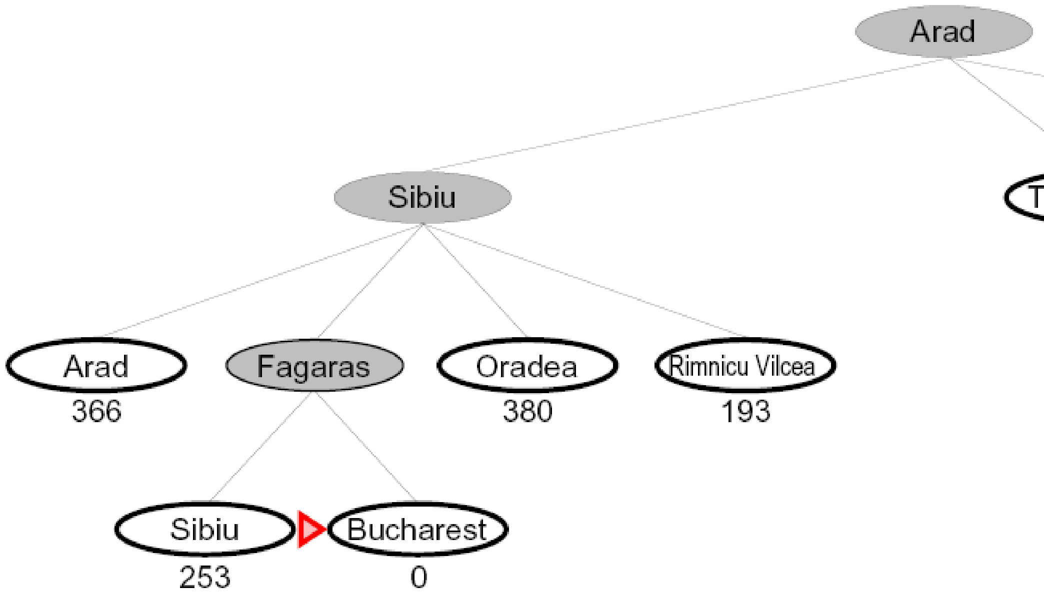
Straight–line distance to Bucharest

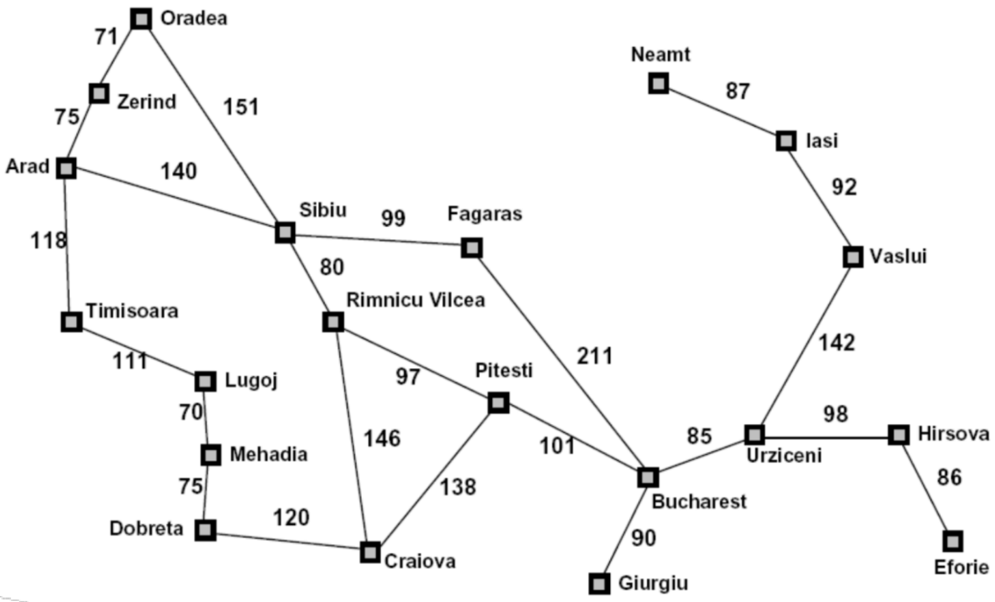
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

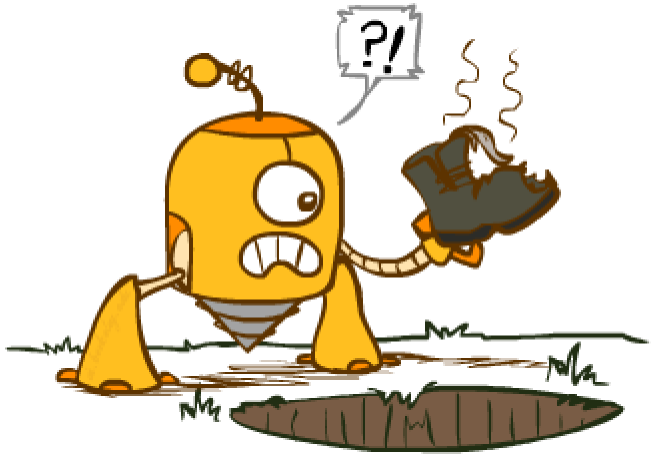
Greedy Search

- Expand the node that seems closest…



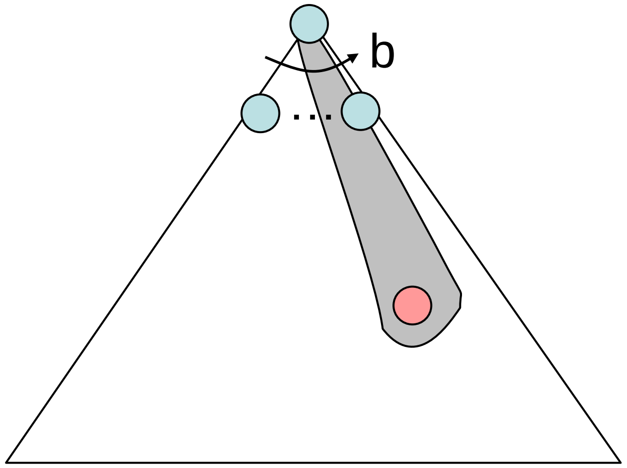


- What can go wrong?



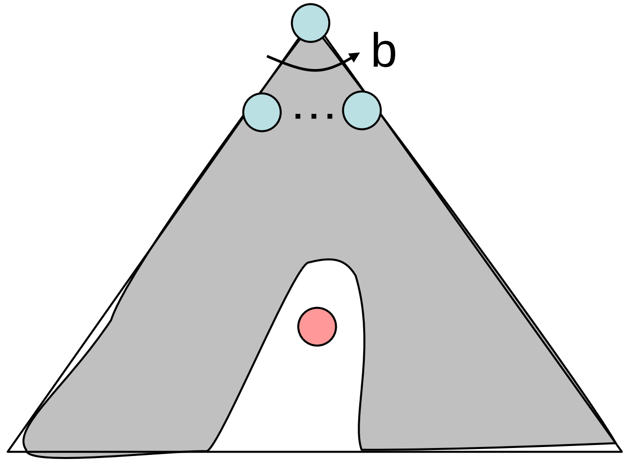
Greedy Search

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



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

- Worst-case: like a badly-guided DFS



A* Search

A* Search (most popular algorithm in AI)

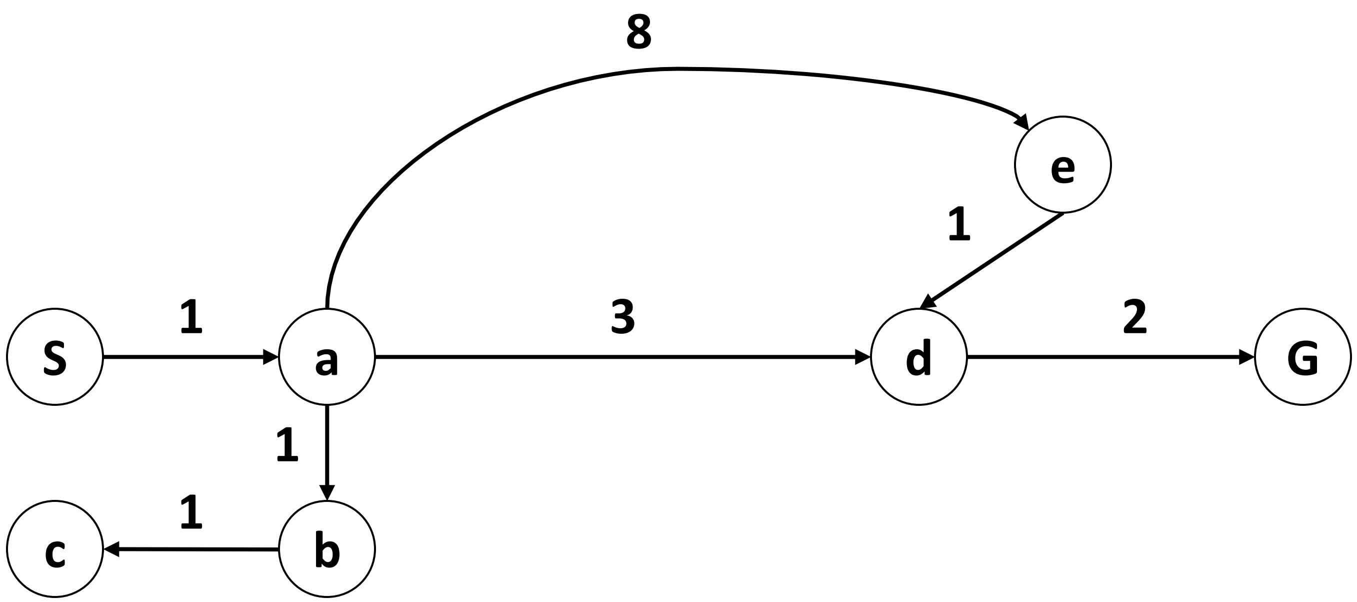
1) $f(N) = g(N) + h(N)$, where:
   - $g(N)$ = cost of best path found so far to N
   - $h(N)$ = **admissible** heuristic function

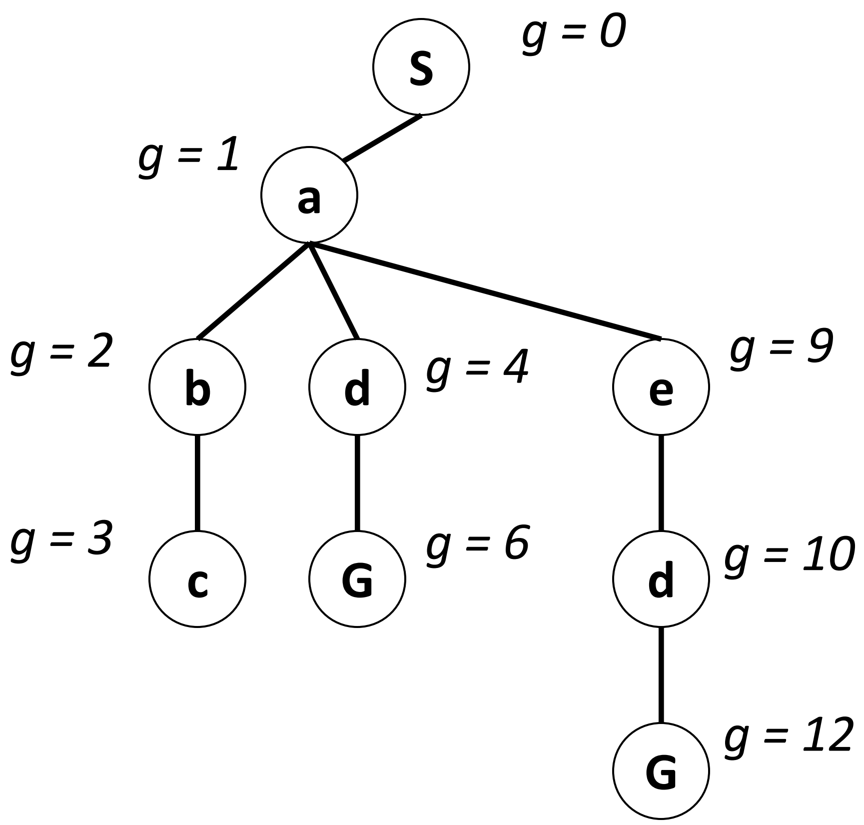
2) for all arcs: $c(N,N') \geq \boldsymbol{\varepsilon} > 0$

➔ Best-first search is then called A* search

Combining UCS and Greedy

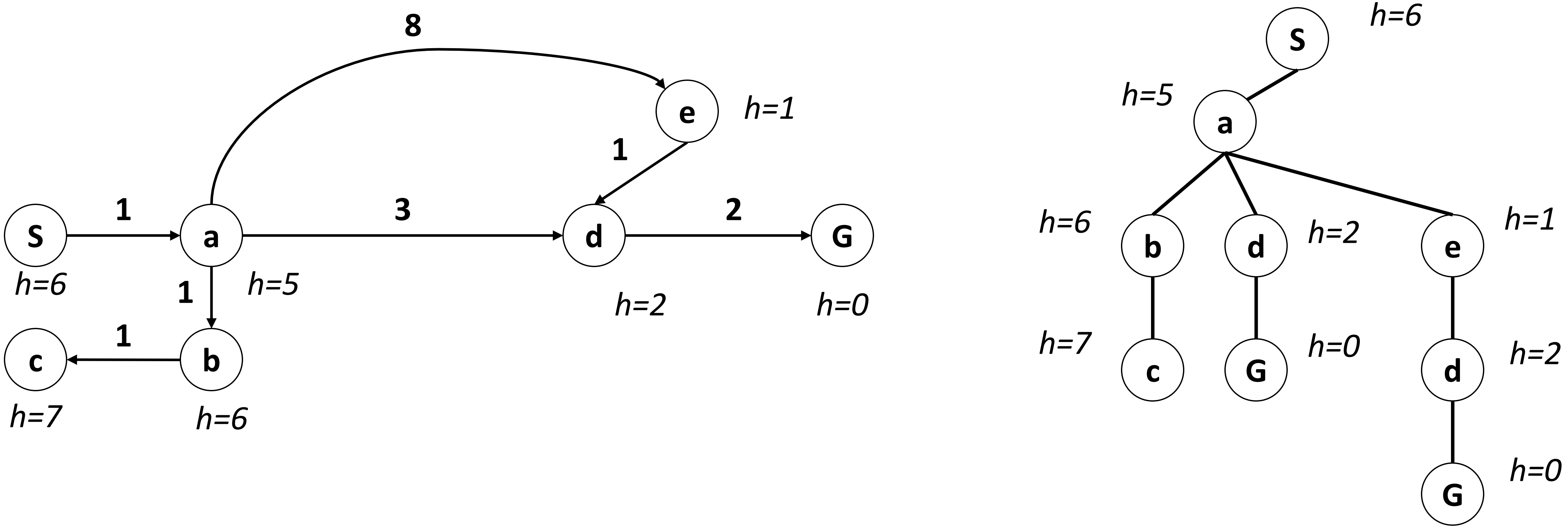
- Uniform-cost orders by path cost, or *backward cost* g(n)





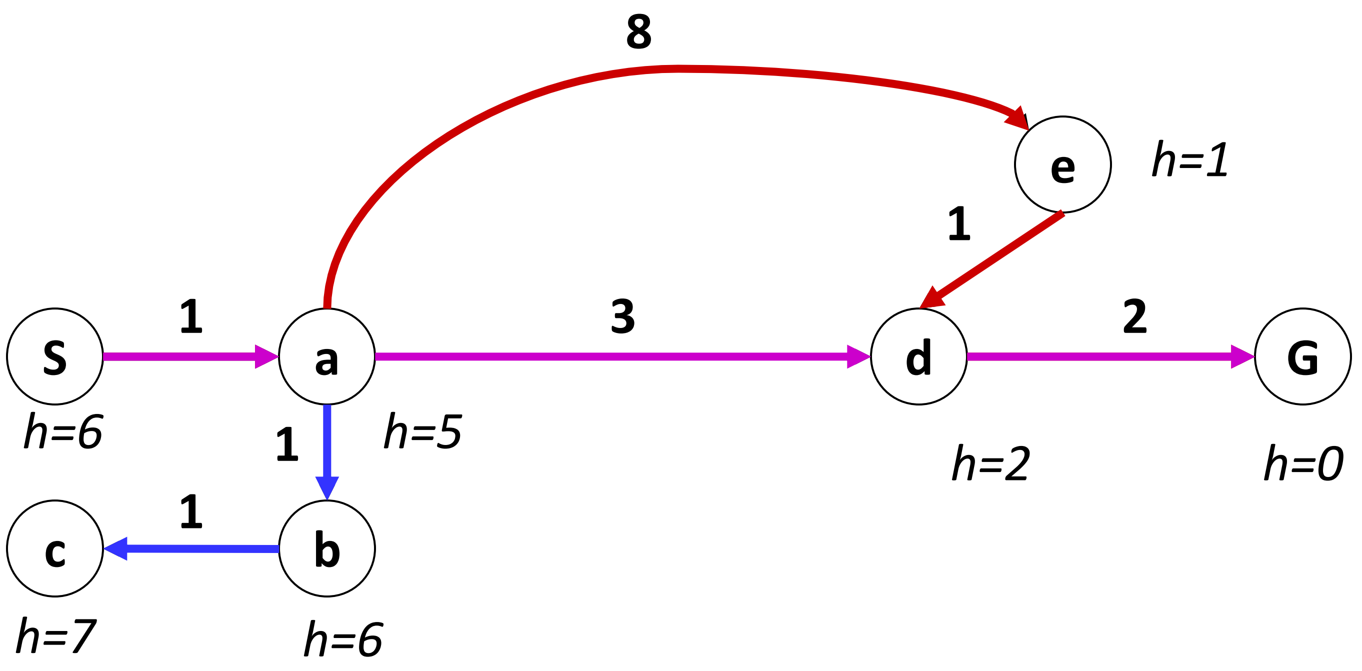
Combining UCS and Greedy

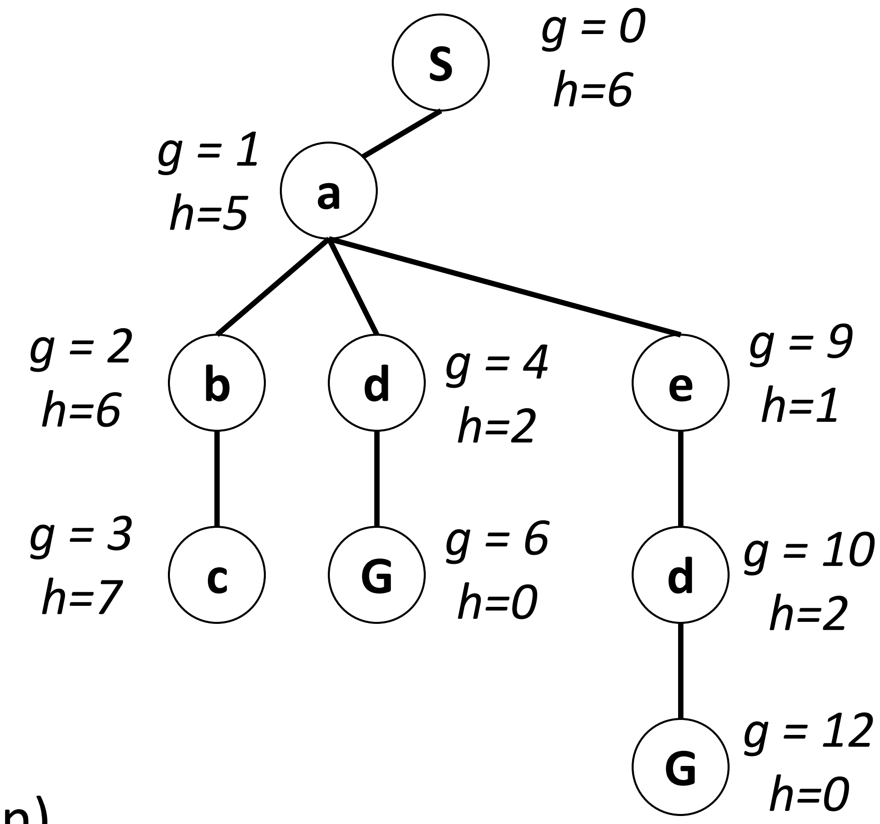
- Greedy orders by goal proximity, or *forward cost* h(n)



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)

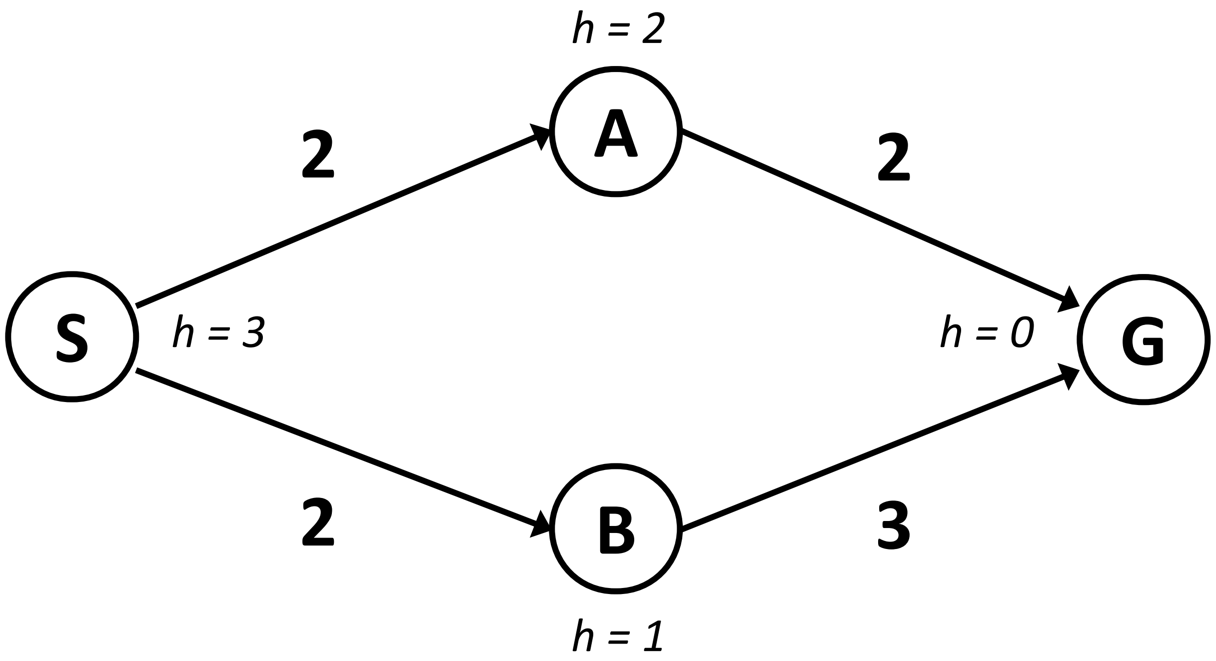




- A* Search orders by the sum: f(n) = g(n) + h(n)

When should A* terminate?

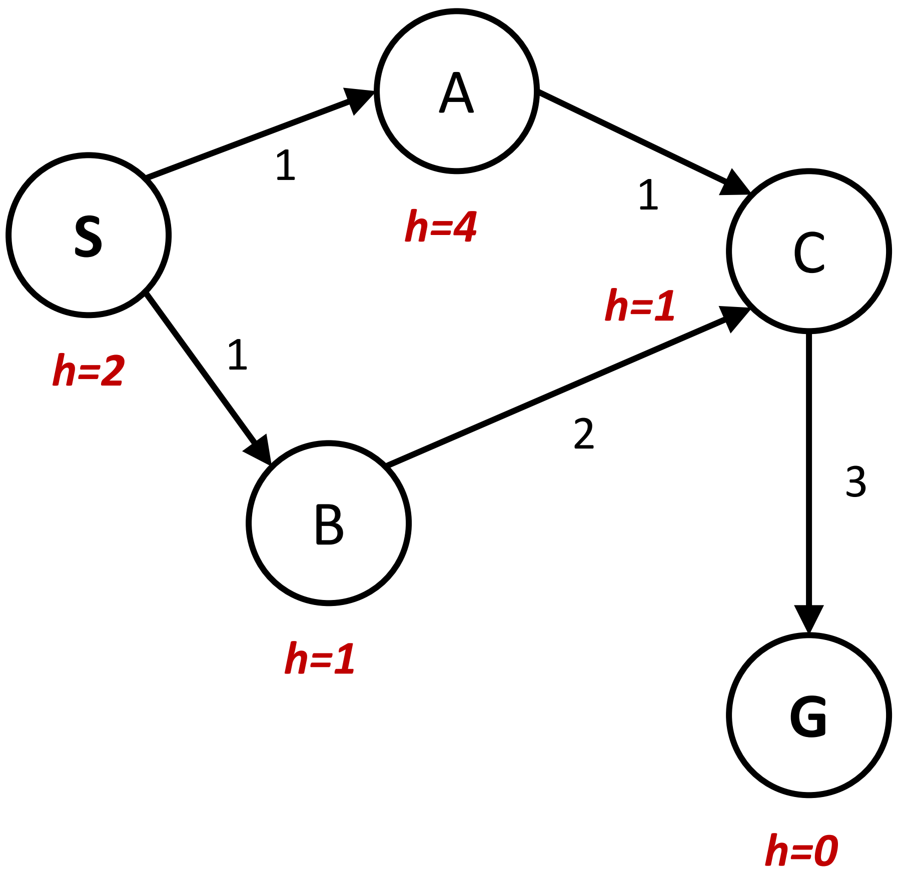
- Should we stop when we enqueue a goal?



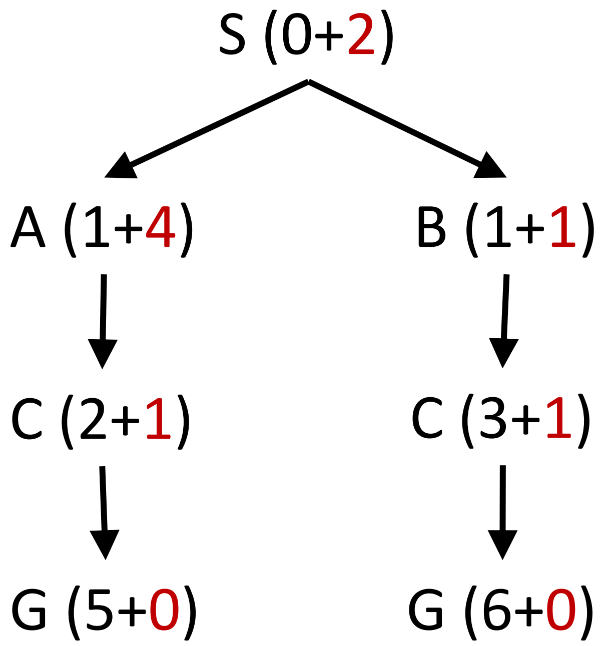
- No: only stop when we dequeue a goal

A* Graph Search Gone Wrong?

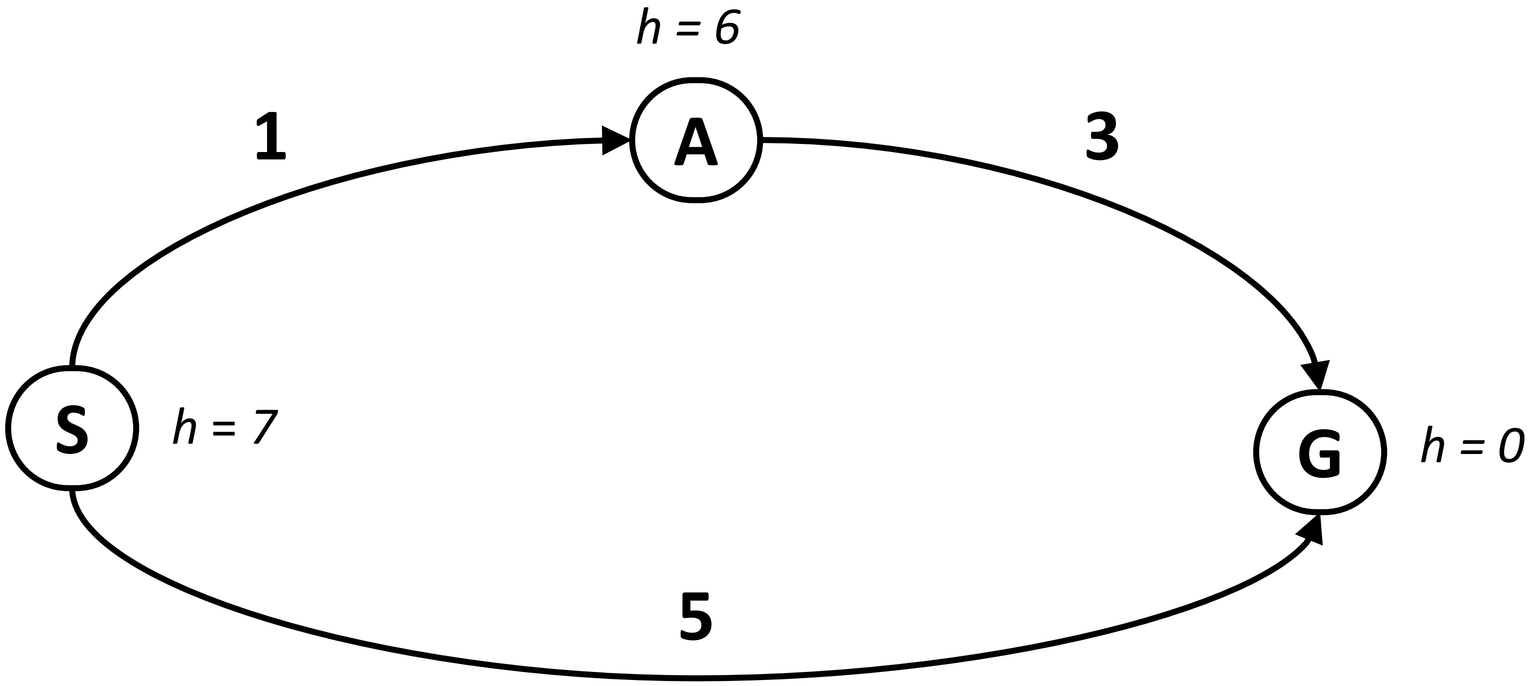
State space graph



Search tree



Is A* Optimal?



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

Heuristic Function

- The **heuristic function** $h(N) \geq 0$ estimates the cost to go from STATE(N) to a goal state

  Its value is **independent of the current search tree**; it depends only on STATE(N) and the goal test GOAL?

- Example:

5		8
4	2	1
7	3	6

STATE(N)

1	2	3
4	5	6
7	8	

Goal state

$h_1(N)$ = number of misplaced numbered tiles = 6

[Why is it an estimate of the distance to the goal?]

Other Examples

5		8
4	2	1
7	3	6

STATE(N)

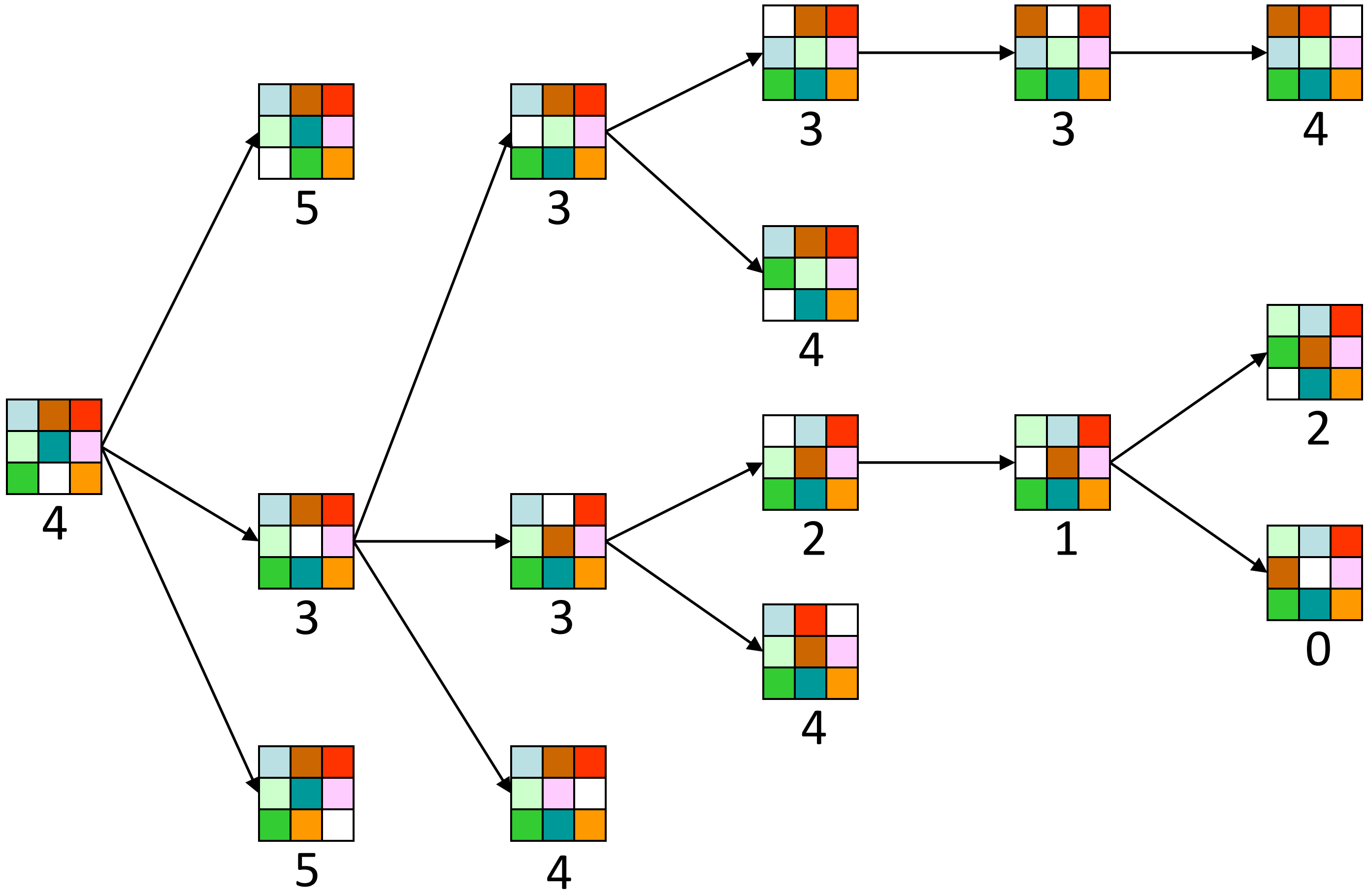
1	2	3
4	5	6
7	8	

Goal state

- $h_1(N)$ = number of misplaced numbered tiles = 6
- $h_2(N)$ = sum of the (Manhattan) distance of every numbered tile to its goal position
  $= 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1 = 13$
- $h_3(N)$ = sum of permutation inversions
  $= n_5 + n_8 + n_4 + n_2 + n_1 + n_7 + n_3 + n_6$
  $= 4 + 6 + 3 + 1 + 0 + 2 + 0 + 0$
  $= 16$

8-Puzzle

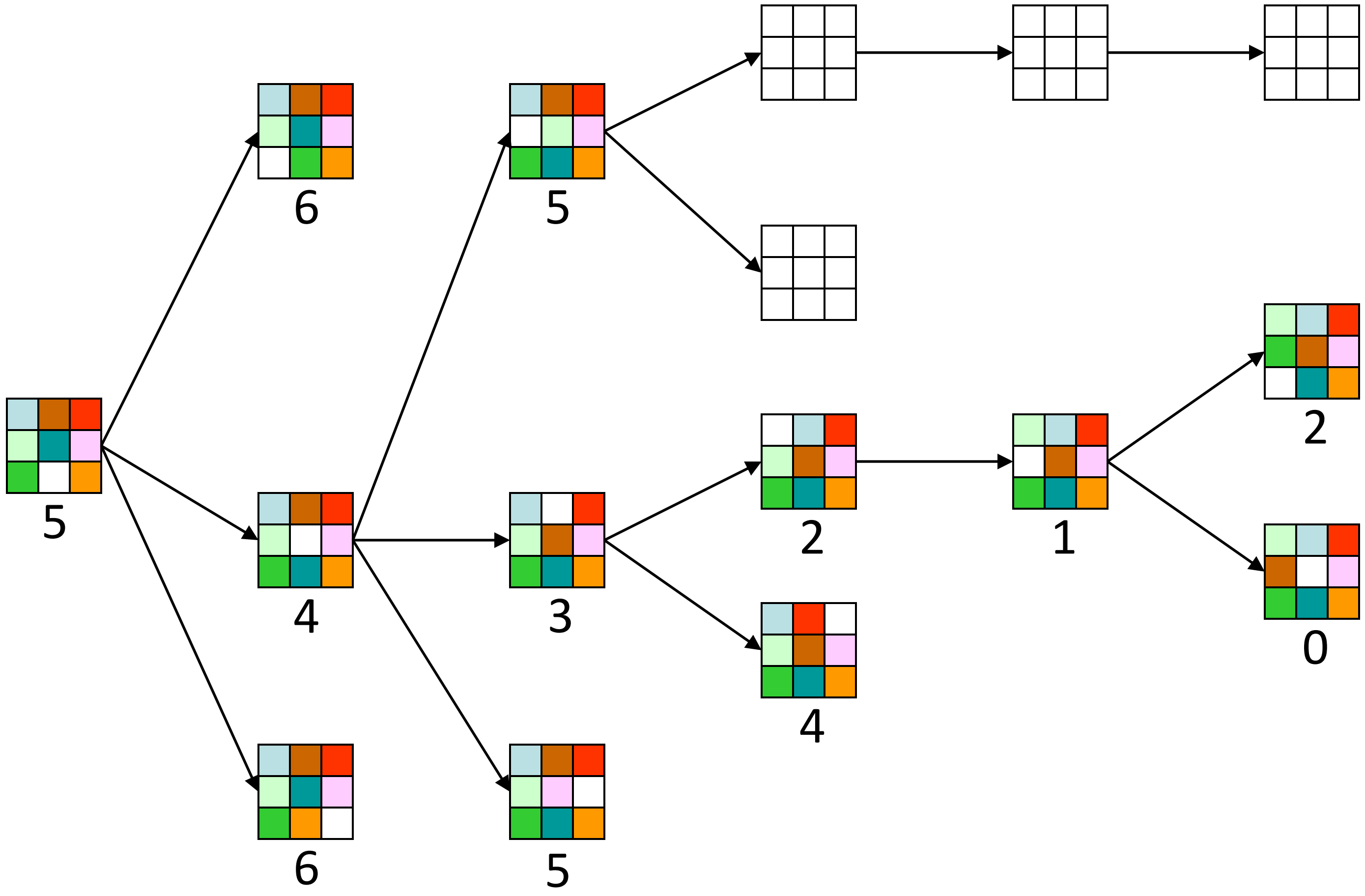
f(N) = h(N) = number of misplaced numbered tiles



The white tile is the empty tile

8-Puzzle

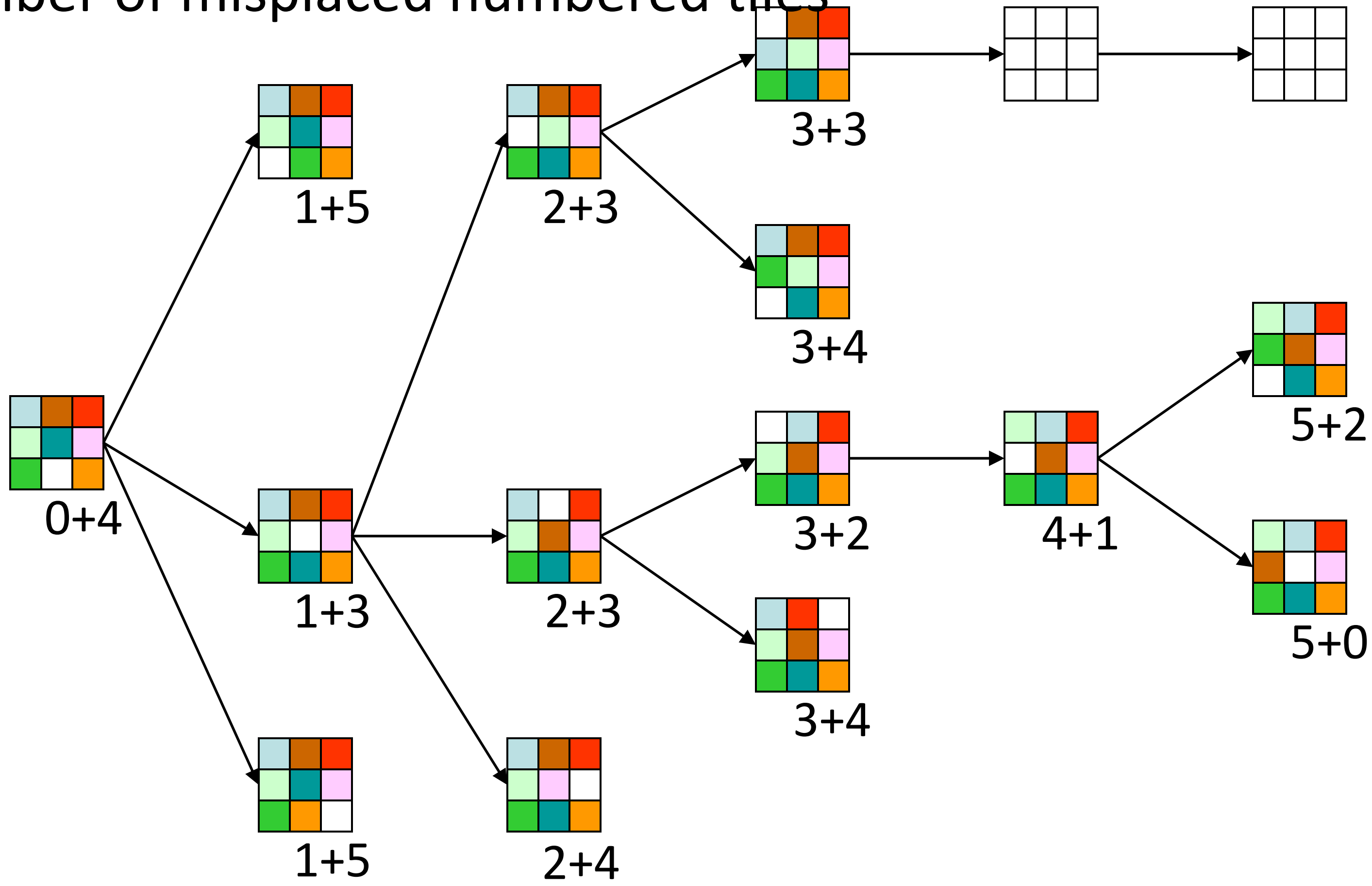
f(N) = h(N) = Σ distances of numbered tiles to their goals



8-Puzzle

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

with $h(N)$ = number of misplaced numbered tiles



Admissible Heuristic

- Let $h^*(N)$ be the cost of the optimal path from N to a goal node
- The heuristic function $h(N)$ is admissible if:

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

G is a goal node → $h(G) = 0$

- An admissible heuristic function is always optimistic !

8-Puzzle Heuristics

5		8
4	2	1
7	3	6

STATE(N)

1	2	3
4	5	6
7	8	

Goal state

- h1(N) = number of misplaced tiles = 6

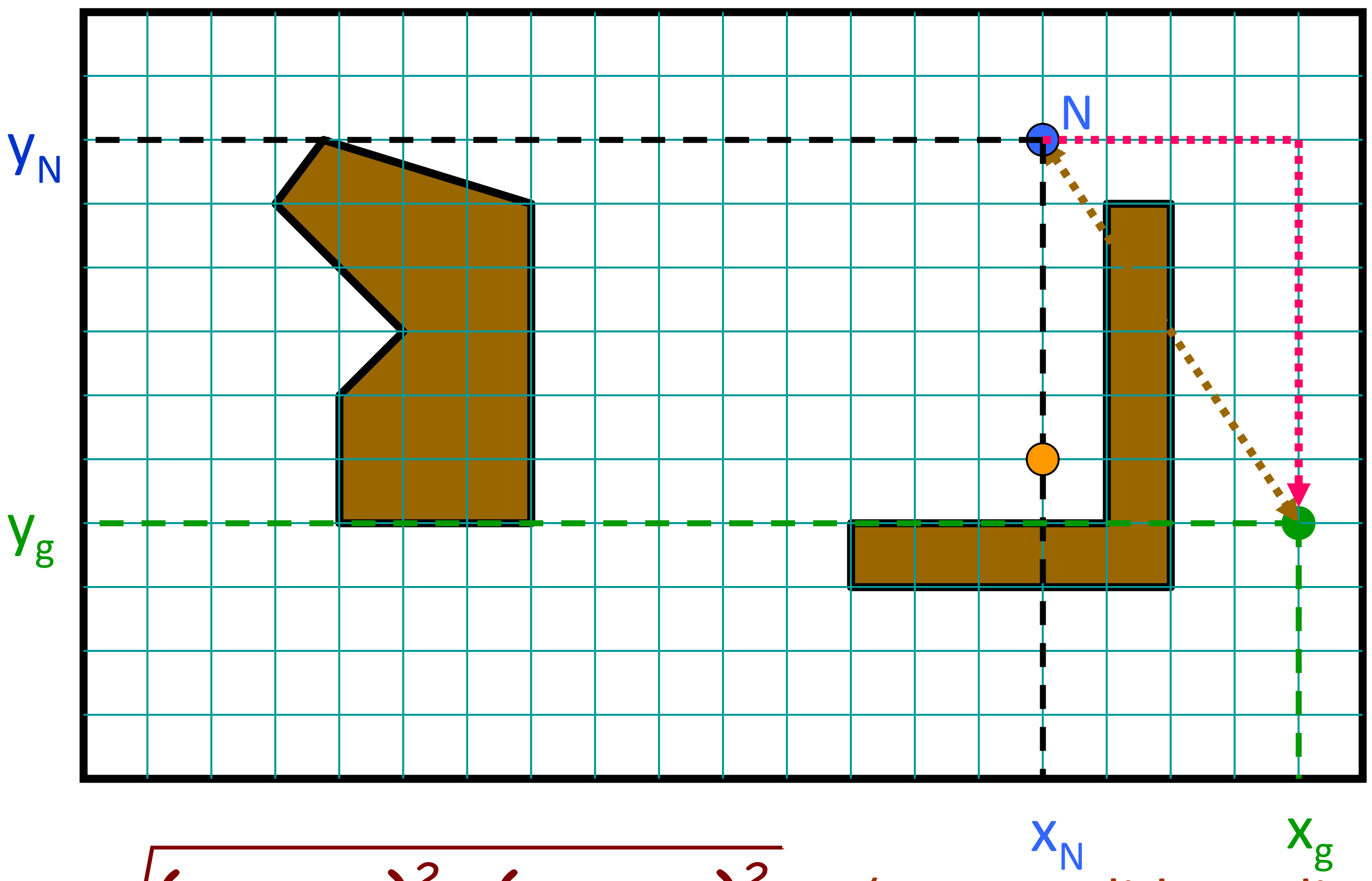
  is admissible

- h2(N) = sum of the (Manhattan) distances of every tile to its goal position

  = 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1 = 13

  is admissible

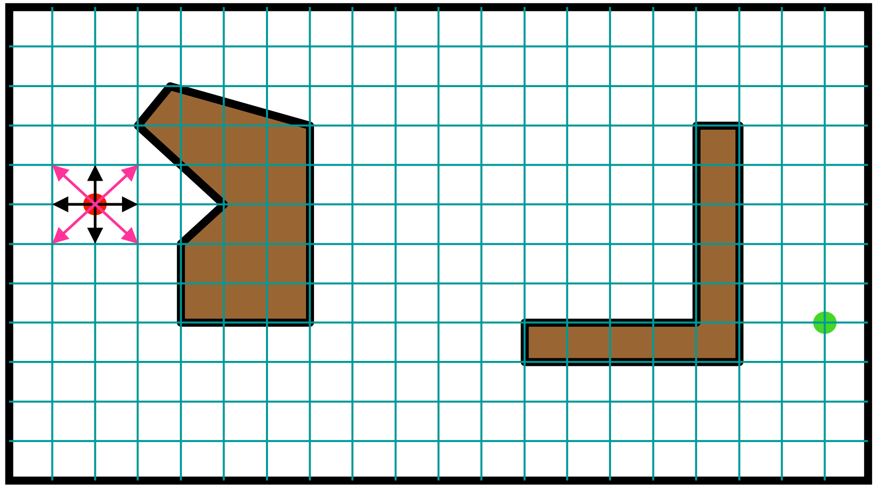
Robot Navigation



$$h_1(N) = \sqrt{(x_N - x_g)^2 + (y_N - y_g)^2}$$ (L_2 or Euclidean distance)

$$h_2(N) = |x_N - x_g| + |y_N - y_g|$$ (L_1 or Manhattan distance)

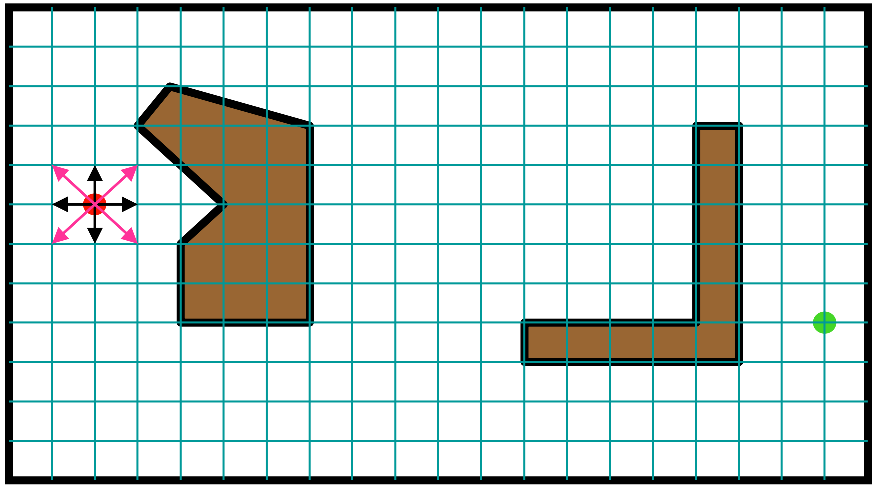
Robot Navigation Heuristics



Cost of one horizontal/vertical step = 1
Cost of one diagonal step = $\sqrt{2}$

$$h_1(N) = \sqrt{(x_N - x_g)^2 + (y_N - y_g)^2}$$ is admissible

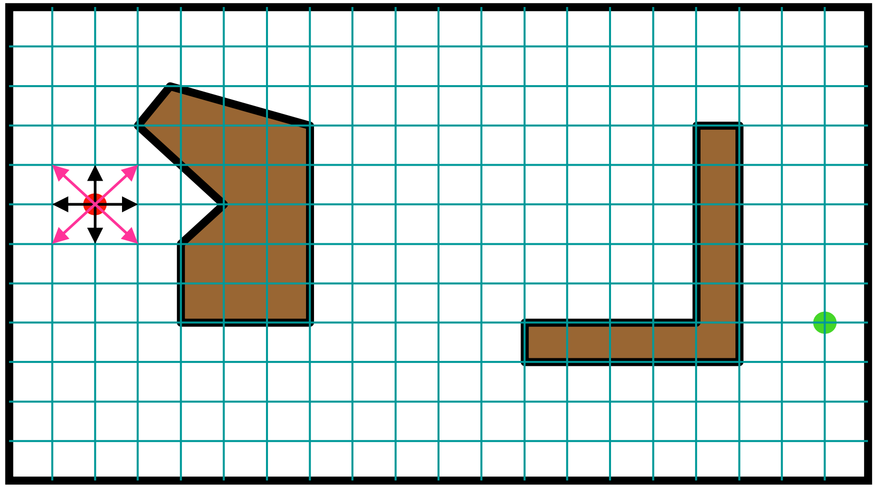
Robot Navigation Heuristics



Cost of one horizontal/vertical step = 1
Cost of one diagonal step = $\sqrt{2}$

$h_2(N) = |x_N - x_g| + |y_N - y_g|$ is ???

Robot Navigation Heuristics



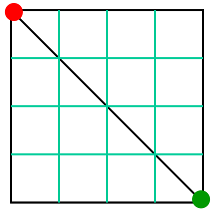
Cost of one horizontal/vertical step = 1

Cost of one diagonal step = $\sqrt{2}$

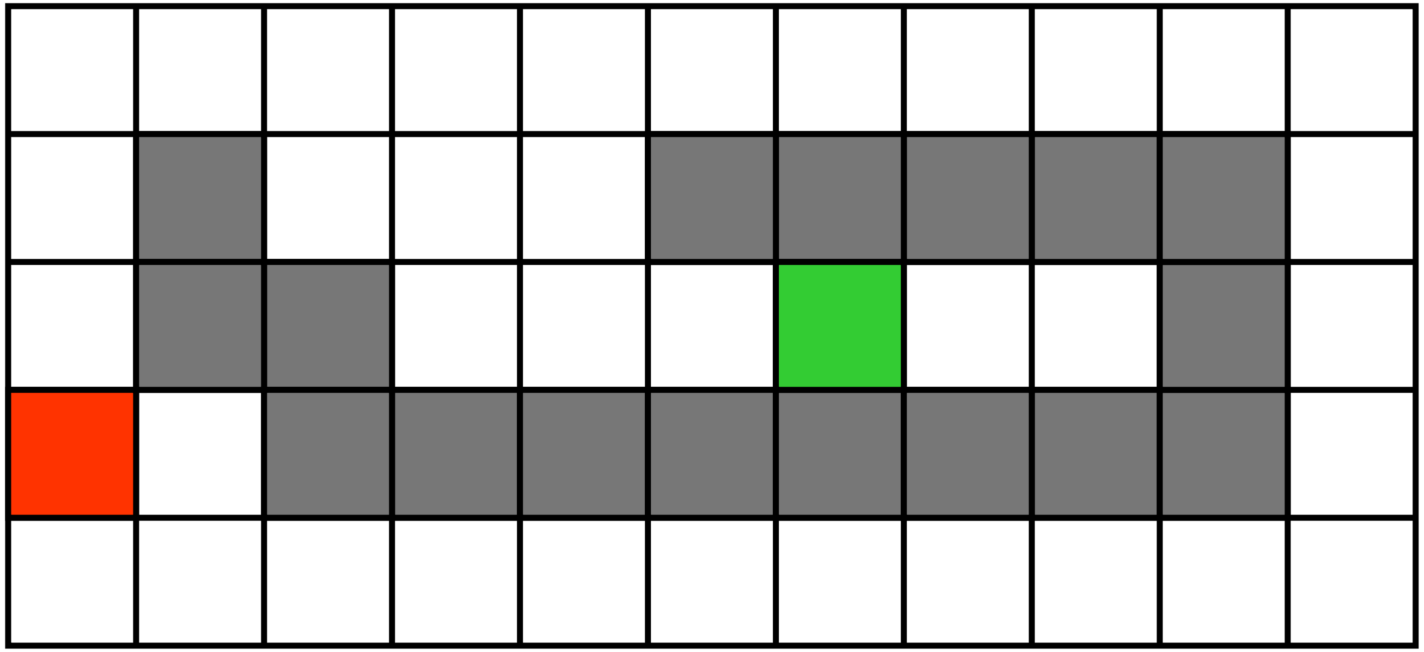
$h_2(N) = |x_N - x_g| + |y_N - y_g|$ is admissible if moving along diagonals is not allowed, and not admissible otherwise

$h^*(I) = 4\sqrt{2}$

$h_2(I) = 8$



Robot Navigation



Robot Navigation

f(N) = h(N), with h(N) = Manhattan distance to the goal
(not A*)

8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

Robot Navigation

f(N) = h(N), with h(N) = Manhattan distance to the goal
(not A*)

8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

Robot Navigation

f(N) = g(N)+h(N), with h(N) = Manhattan distance to goal

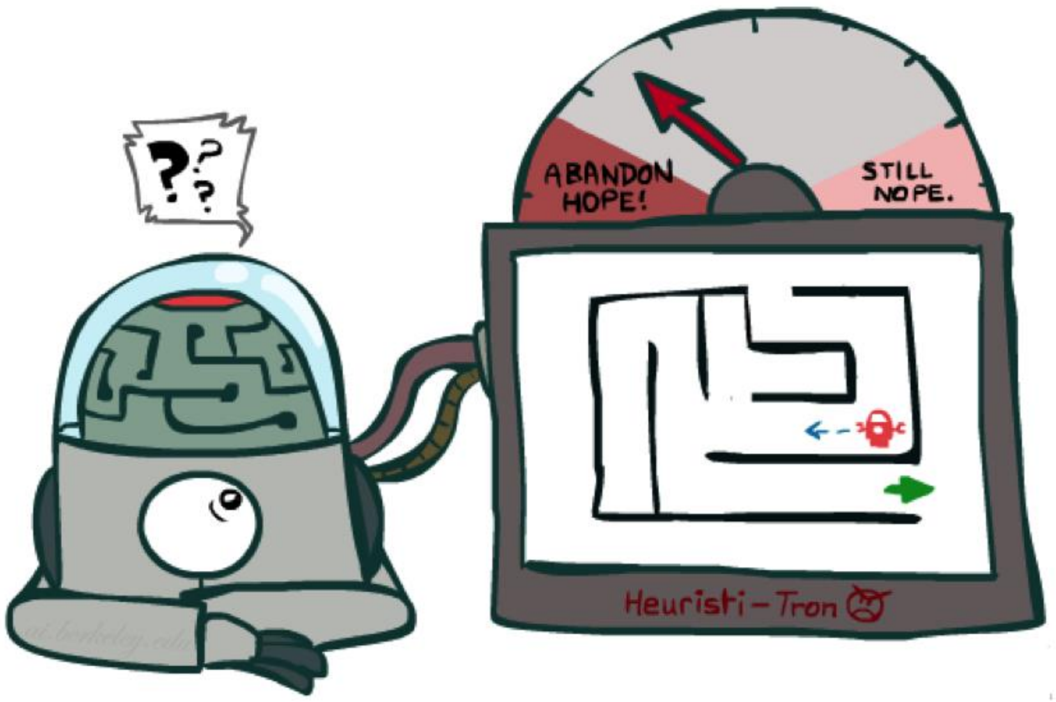
(A*)

8+3	7+4	6+3	5+6	4+7	3+8	2+9	3+10	4	5	6
7+2		5+6	4+7	3+8						5
6+1			3	2+9	1+10	0+11	1	2		4
7+0	6+1									5
8+1	7+2	6+3	5+4	4+5	3+6	2+7	3+8	4	5	6

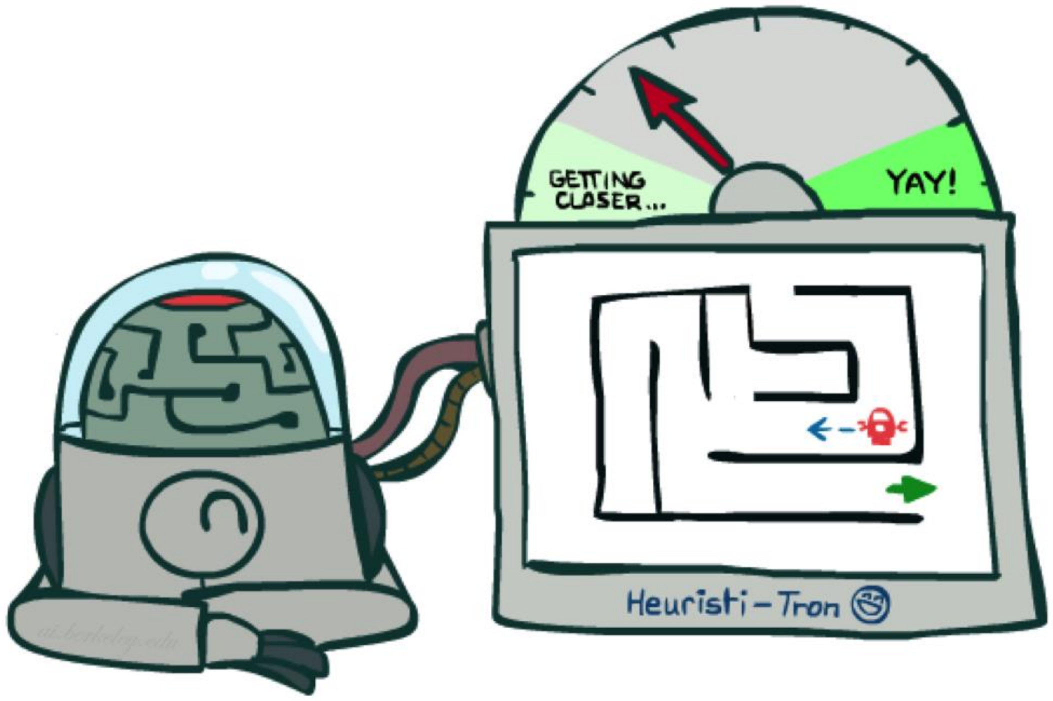
How to create an admissible h?

- An admissible heuristic can usually be seen as the cost of an optimal solution to a relaxed problem (one obtained by removing constraints)
- In robot navigation:
  - The Manhattan distance corresponds to removing the obstacles
  - The Euclidean distance corresponds to removing both the obstacles and the constraint that the robot moves on a grid
- More on this topic later

Idea: Admissibility



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



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

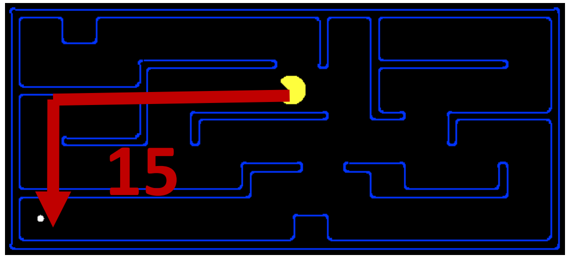
Admissible Heuristics

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

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

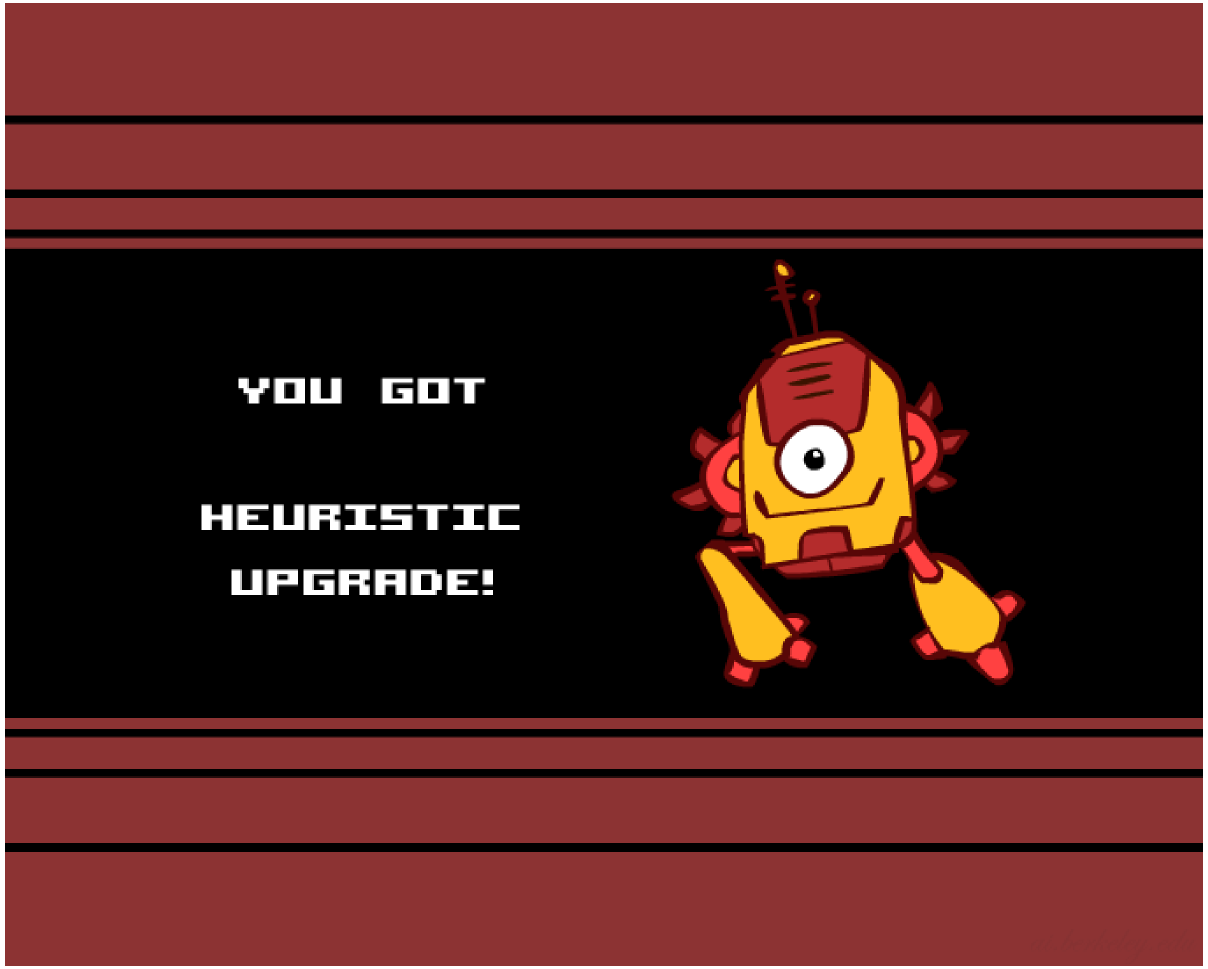
  where $h^*(n)$ is the true cost to a nearest goal

- Examples:



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

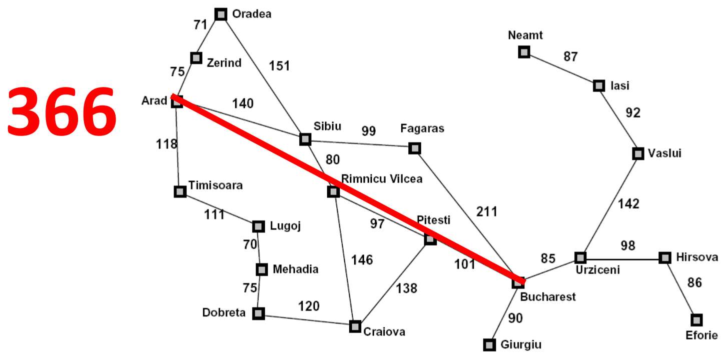
Creating Heuristics

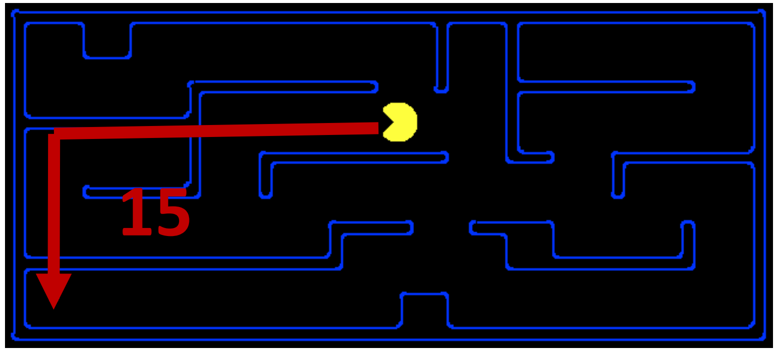


Creating Admissible Heuristics

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

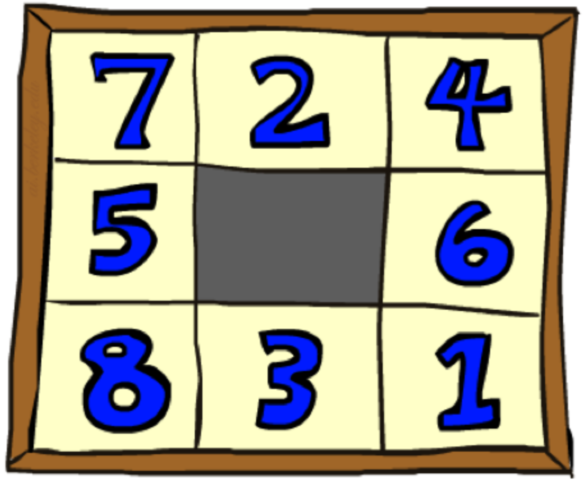
- Often, admissible heuristics are solutions to *relaxed problems,* where new actions are available



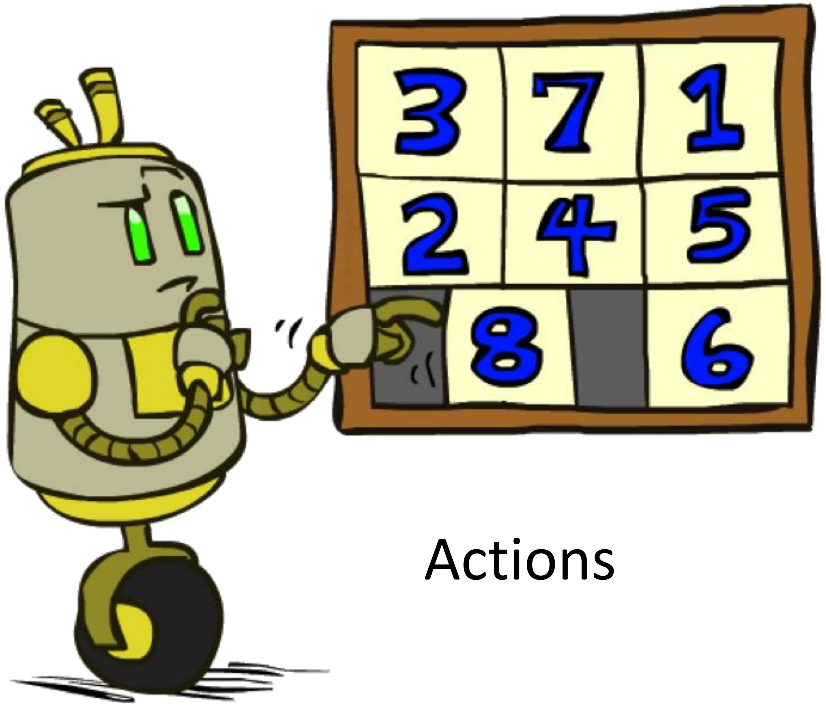


- Inadmissible heuristics are often useful too

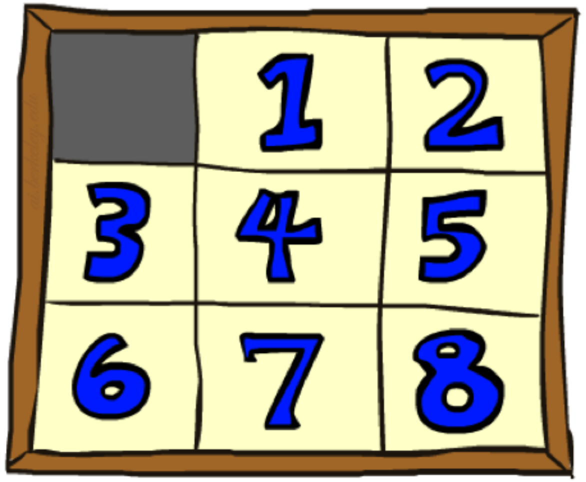
Example: 8 Puzzle



Start State



Actions

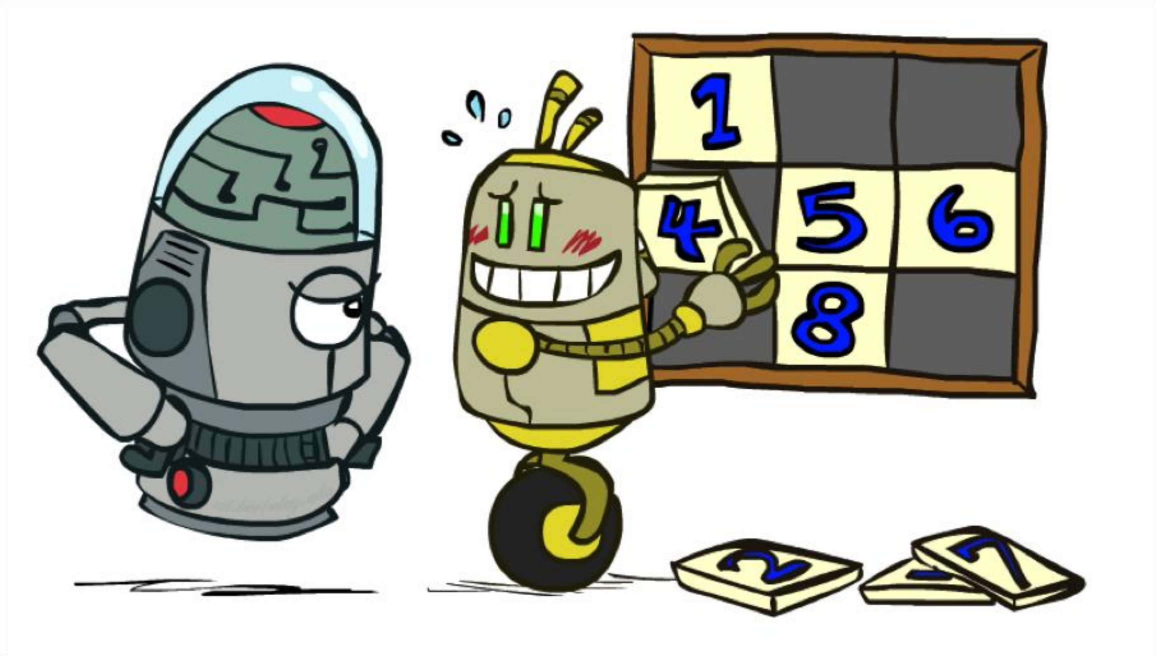


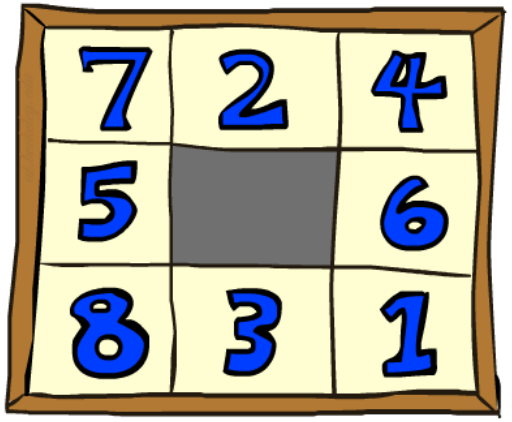
Goal State

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

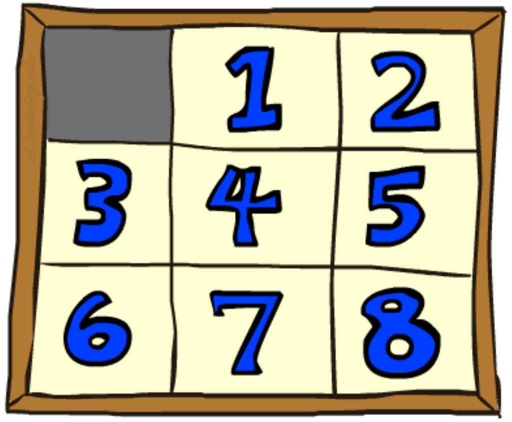
8 Puzzle I

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





Start State



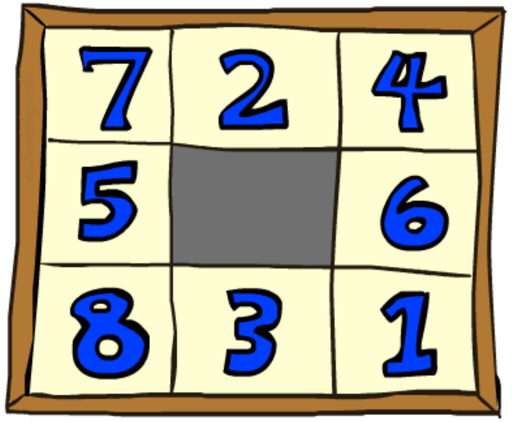
Goal State

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

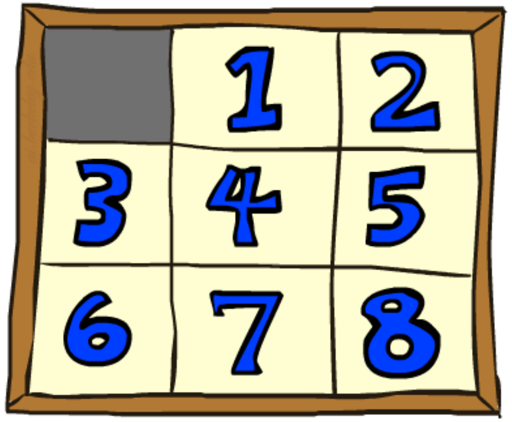
Statistics from Andrew Moore

8 Puzzle II

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



Start State

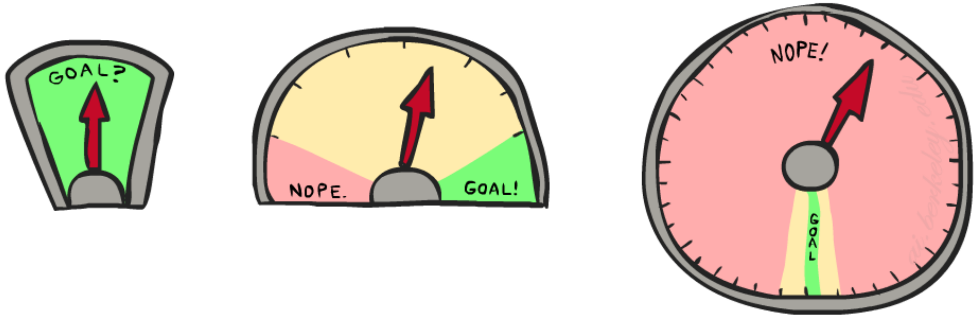


Goal State

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

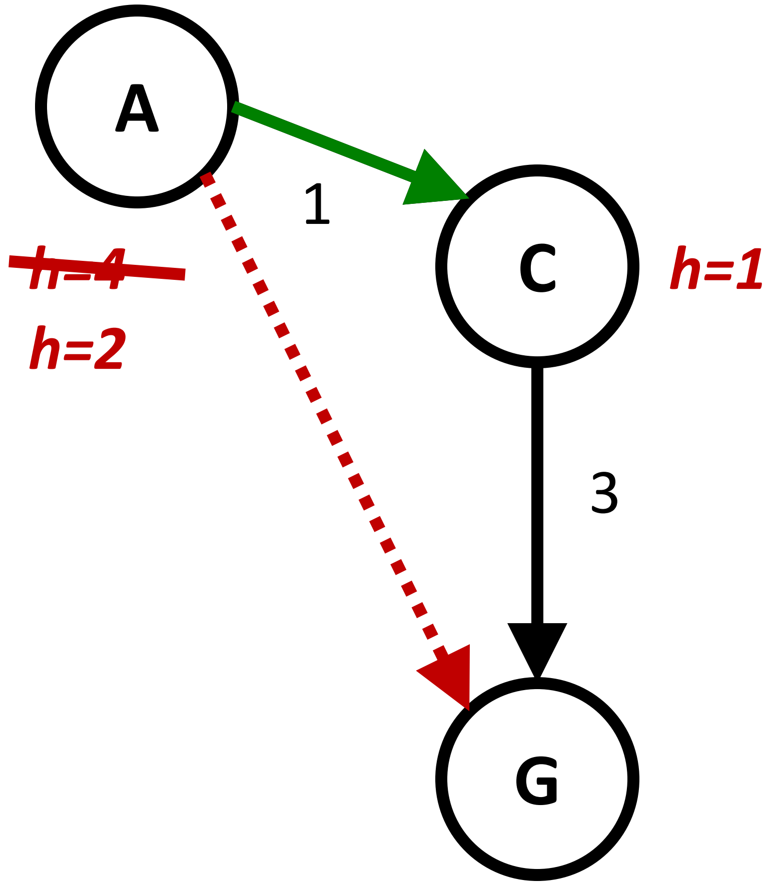
8 Puzzle III

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



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

Consistency of Heuristics



- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal

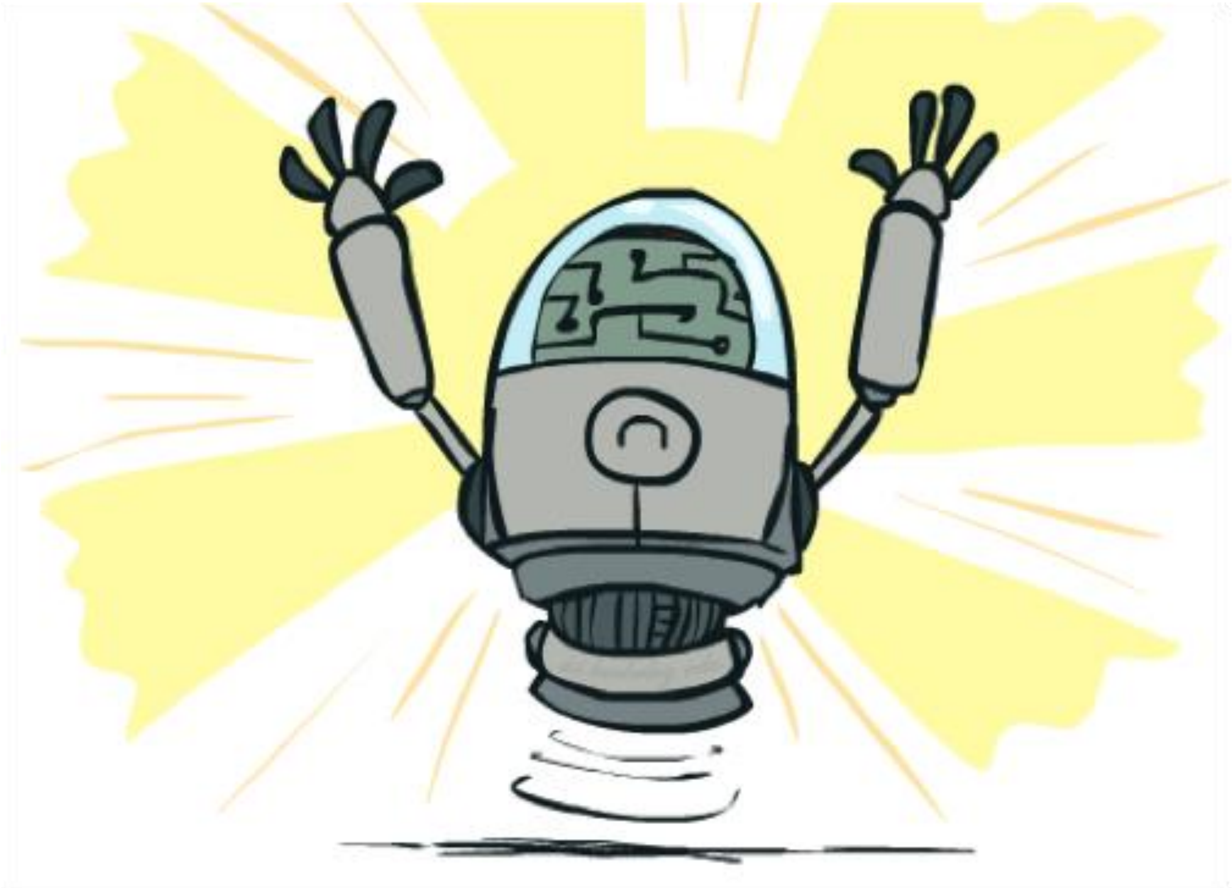
    h(A) ≤ actual cost from A to G
  - Consistency: heuristic "arc" cost ≤ actual cost for each arc

    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* Tree Search



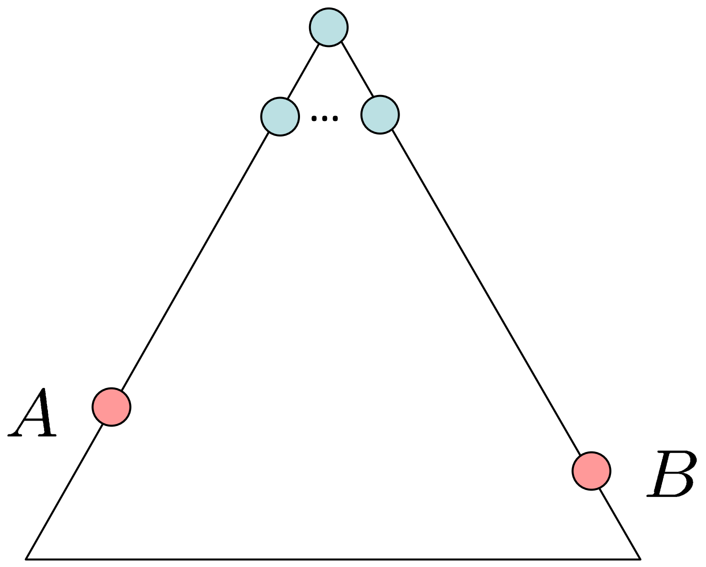
Optimality of A* Tree Search

Assume:

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

Claim:

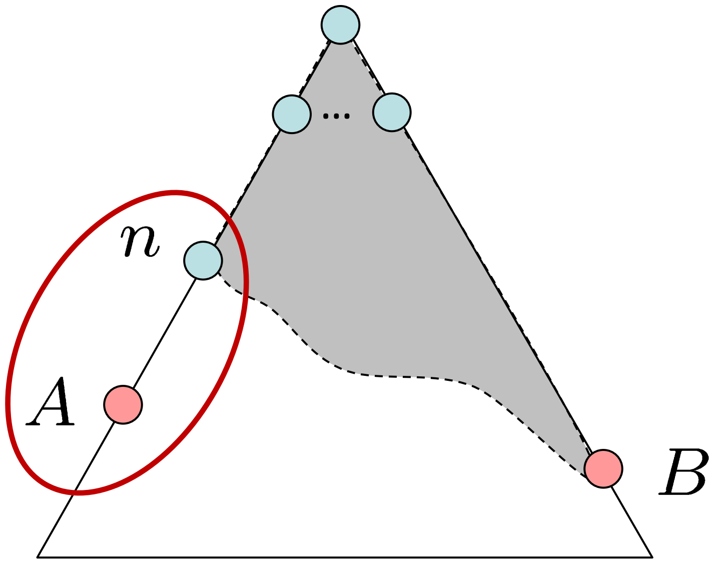
- A will exit the fringe before B



Optimality of A* Tree Search: Blocking

Proof:

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

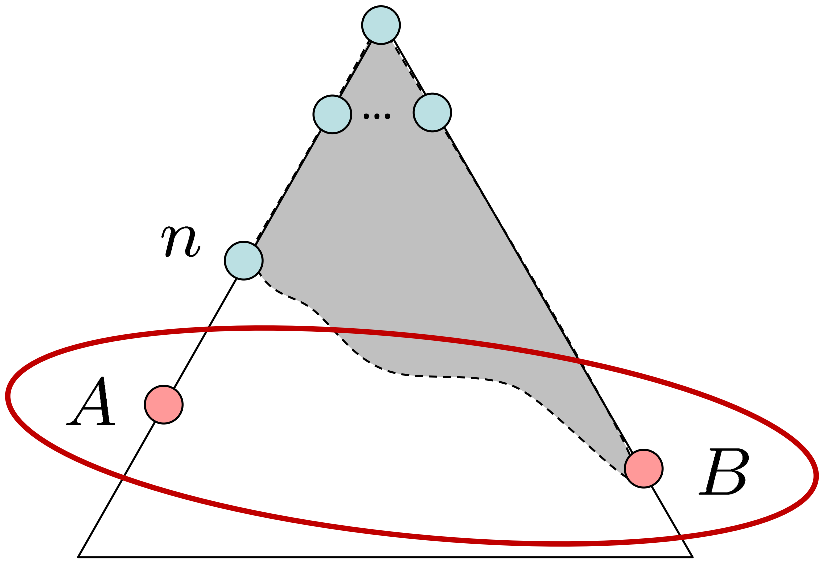


$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \leq g(A)$	Admissibility of h
$g(A) = f(A)$	h = 0 at a goal

Optimality of A* Tree Search: Blocking

Proof:

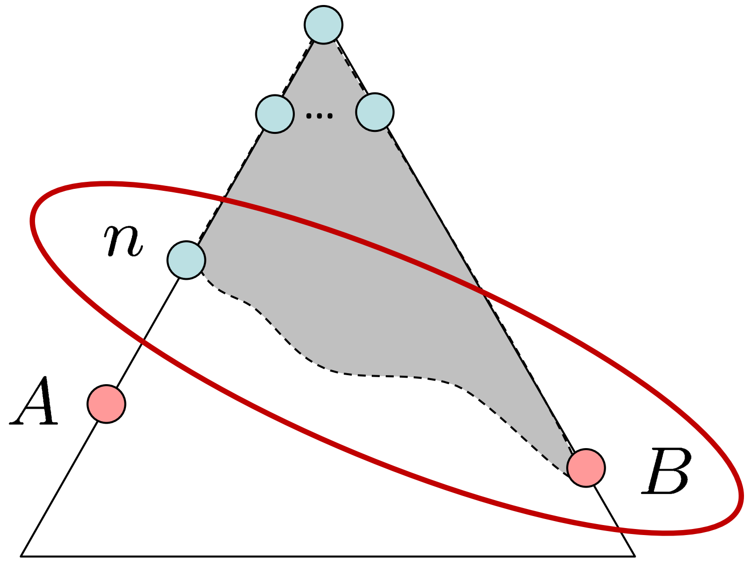
- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too (maybe A!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)



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

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

Optimality of A* Tree Search: Blocking

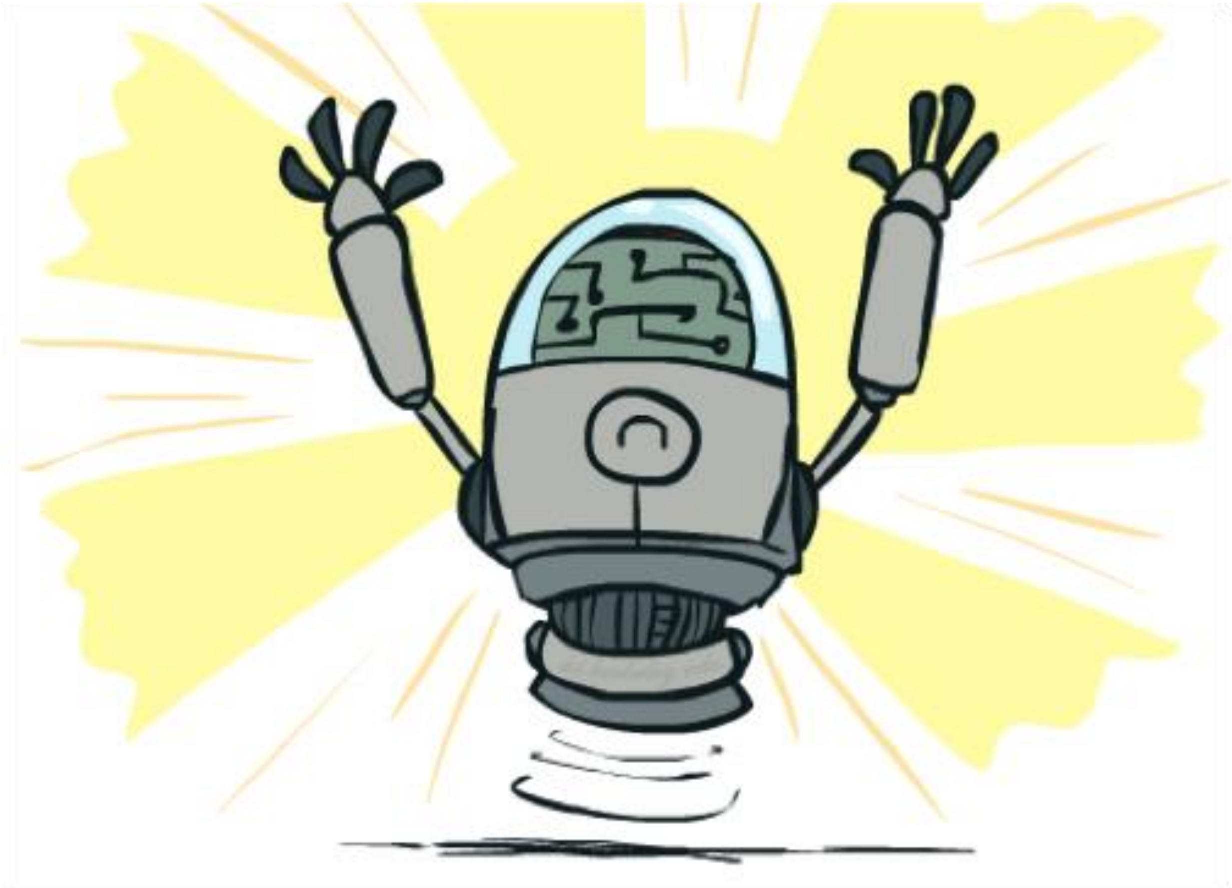


Proof:

- 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

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

Optimality of A* Graph Search



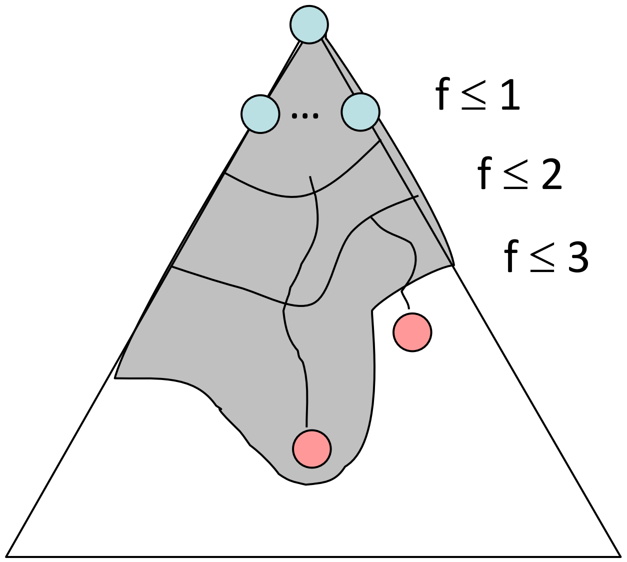
Optimality of A* Graph Search

- Sketch: consider what A* does with a consistent heuristic:

  - Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)

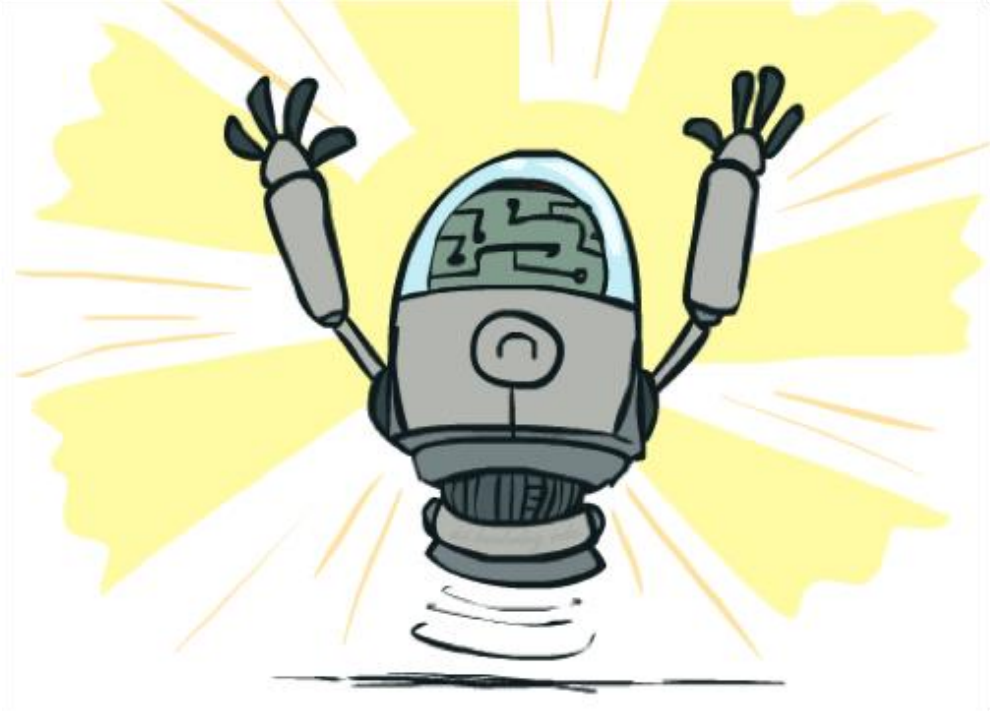
  - Fact 2: For every state s, nodes that reach s optimally are expanded before nodes that reach s suboptimally

  - Result: A* graph search is optimal



Optimality

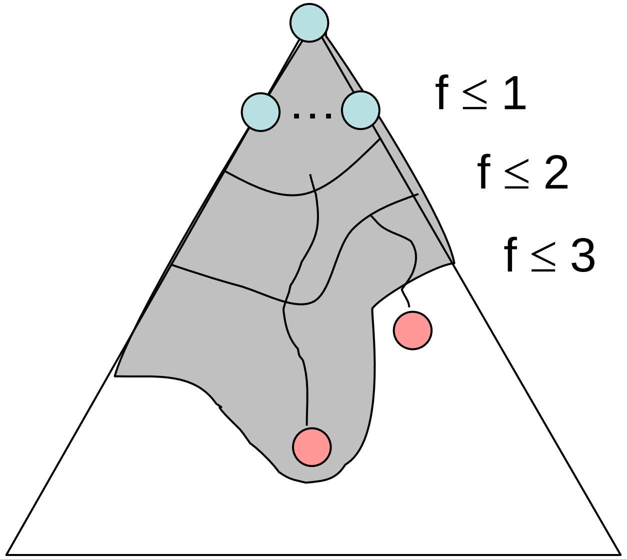
- Tree search:
  - A* is optimal if heuristic is admissible
  - UCS is a special case (h = 0)
- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal (h = 0 is consistent)
- Consistency implies admissibility
- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems



Optimality of A* Graph Search

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

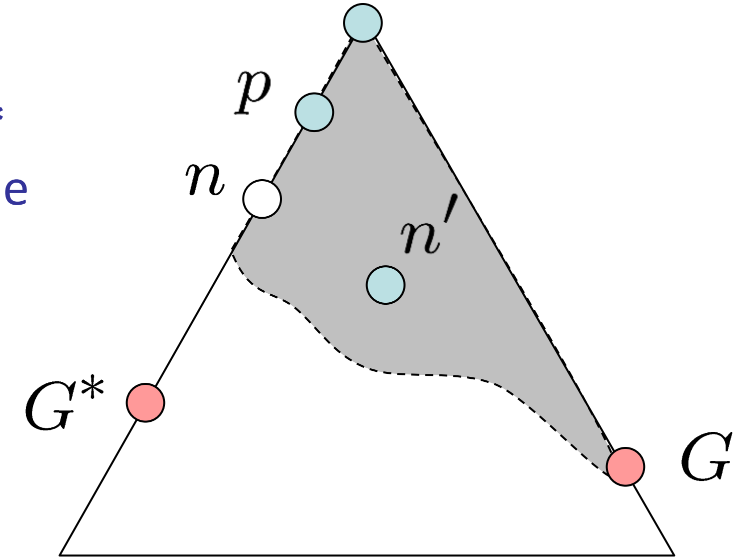
There's a problem with this argument. What are we assuming is true?



Optimality of A* Graph Search

Proof:

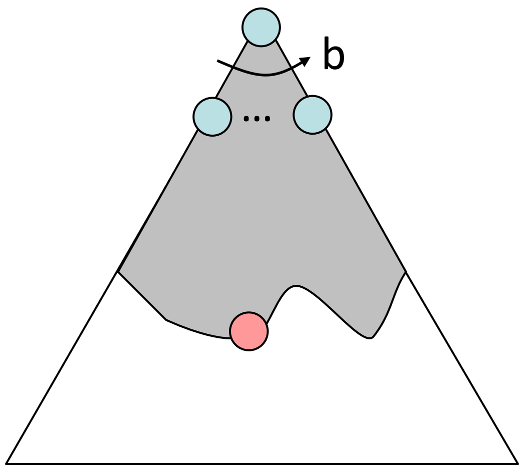
- New possible problem: some *n* on path to G* isn't in queue when we need it, because some worse *n'* for the same state dequeued and expanded first (disaster!)
- Take the highest such *n* in tree
- Let *p* be the ancestor of *n* that was on the queue when *n'* was popped
- *f(p) < f(n)* because of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



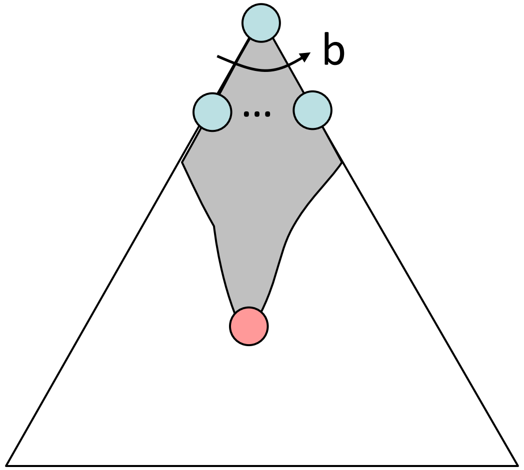
Properties of A*

Properties of A*

Uniform-Cost

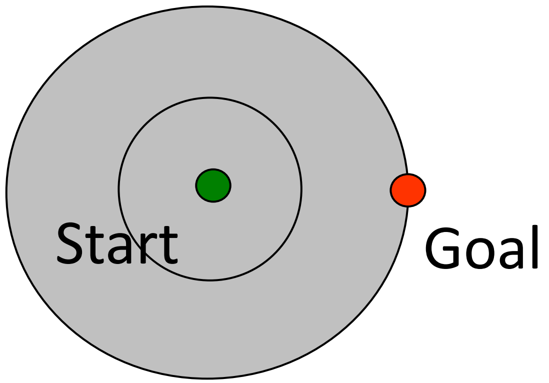


A*

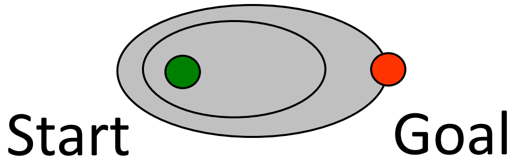


UCS vs A* Contours

- Uniform-cost expands equally in all "directions"



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



Comparison

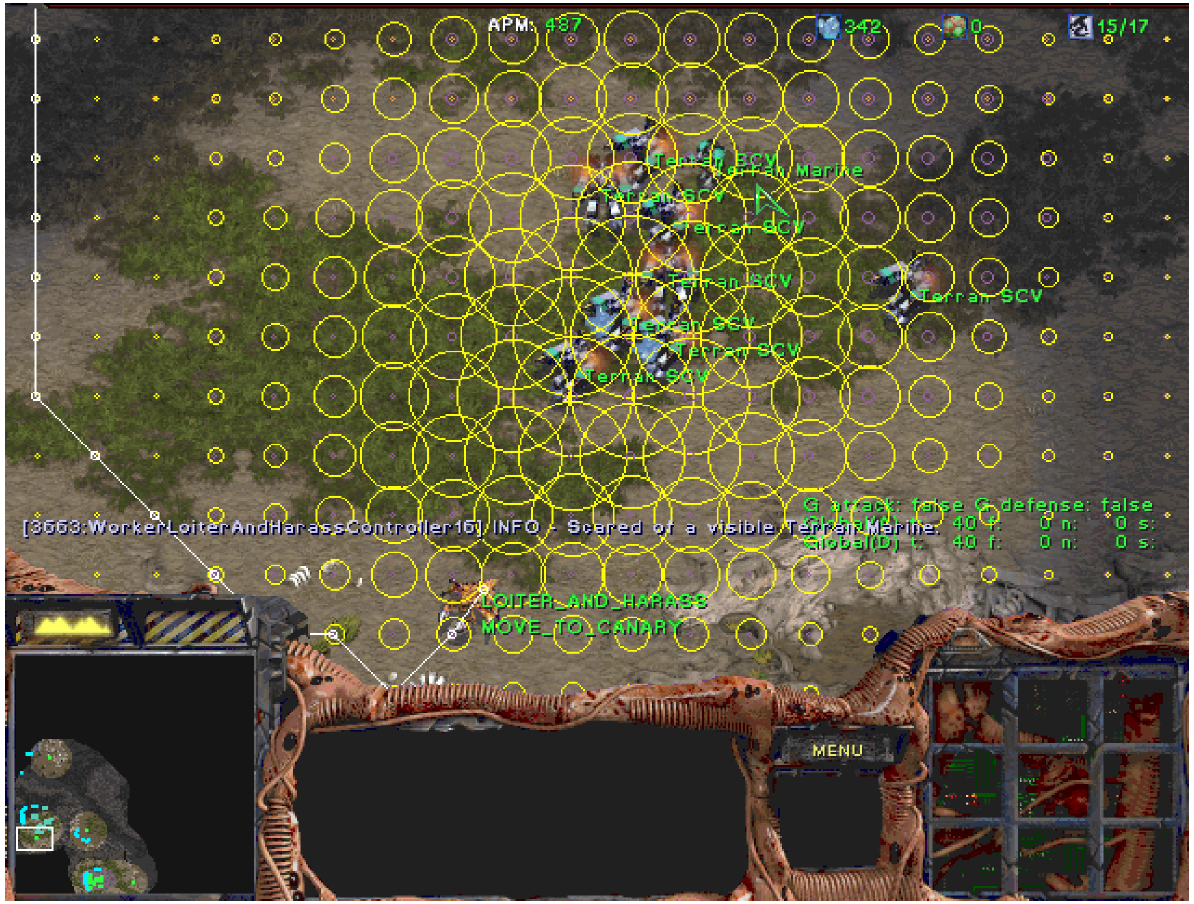
Greedy

Uniform Cost

A*

A* Applications

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- …



Iterative Deepening A* (IDA*)

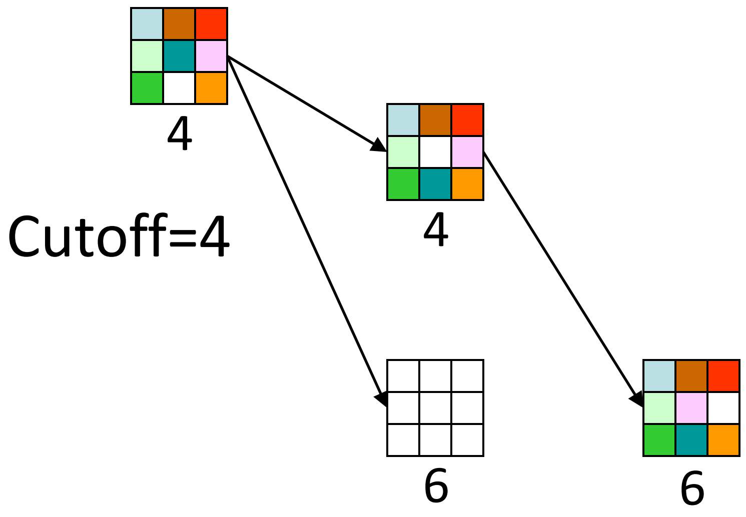
- Idea: Reduce memory requirement of A* by applying cutoff on values of f
- Consistent heuristic function h
- Algorithm IDA*:
  1. Initialize cutoff to f(initial-node)
  2. Repeat:
     a. Perform depth-first search by expanding all nodes N such that $f(N) \leq$ cutoff
     b. Reset cutoff to smallest value f of non-expanded (leaf) nodes

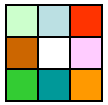
8-Puzzle

$f(N) = g(N) + h(N)$
with $h(N)$ = number of misplaced tiles

8-Puzzle

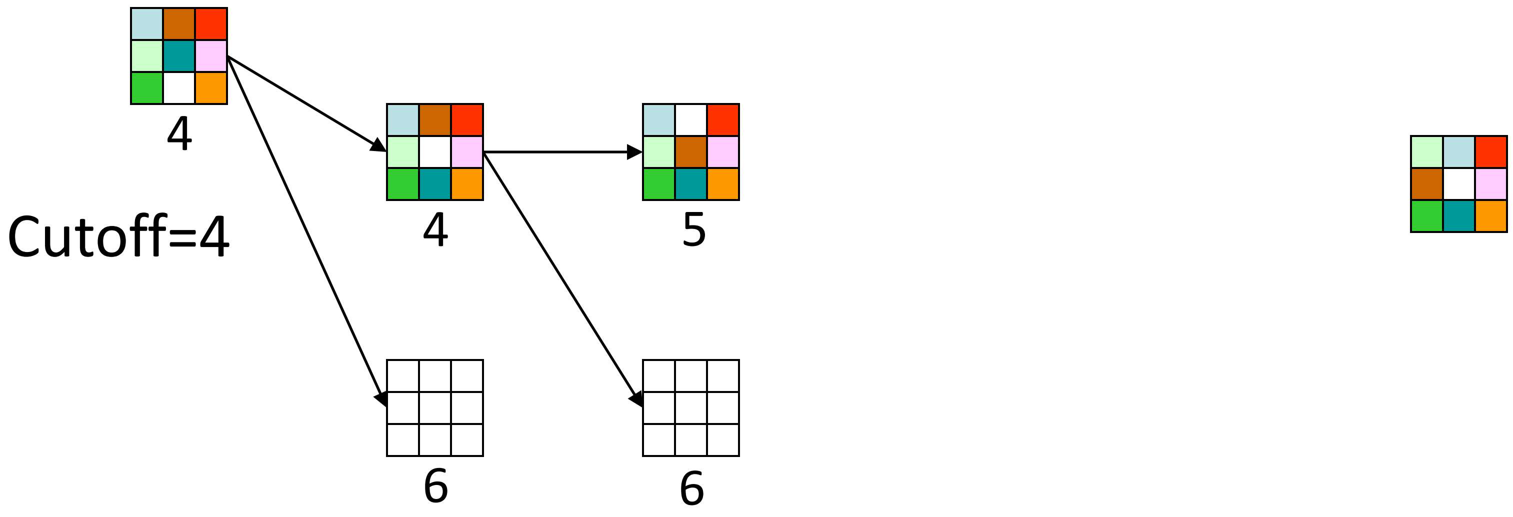
f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles





8-Puzzle

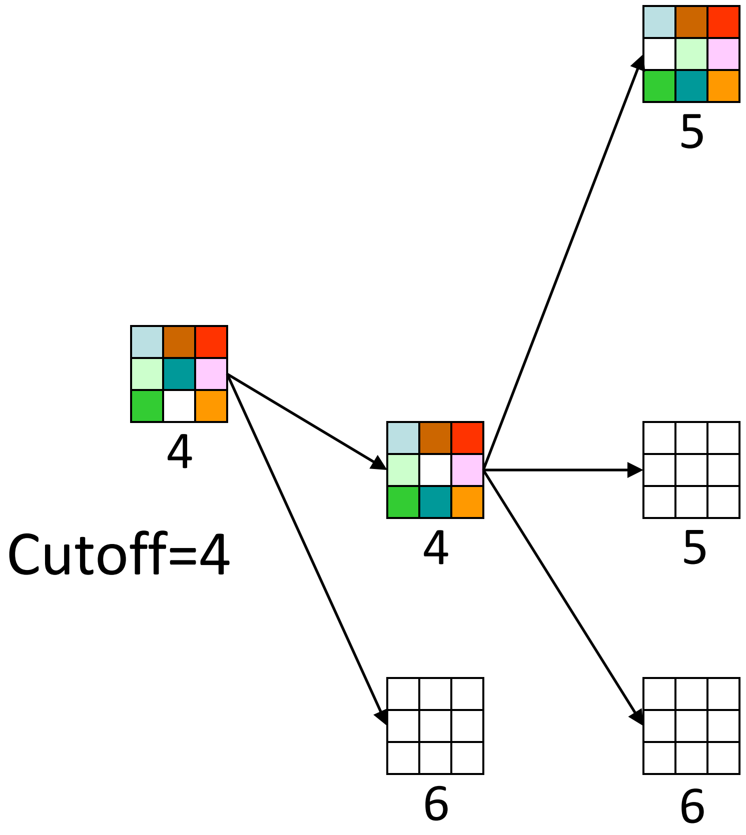
f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles

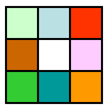


8-Puzzle

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

with $h(N)$ = number of misplaced tiles

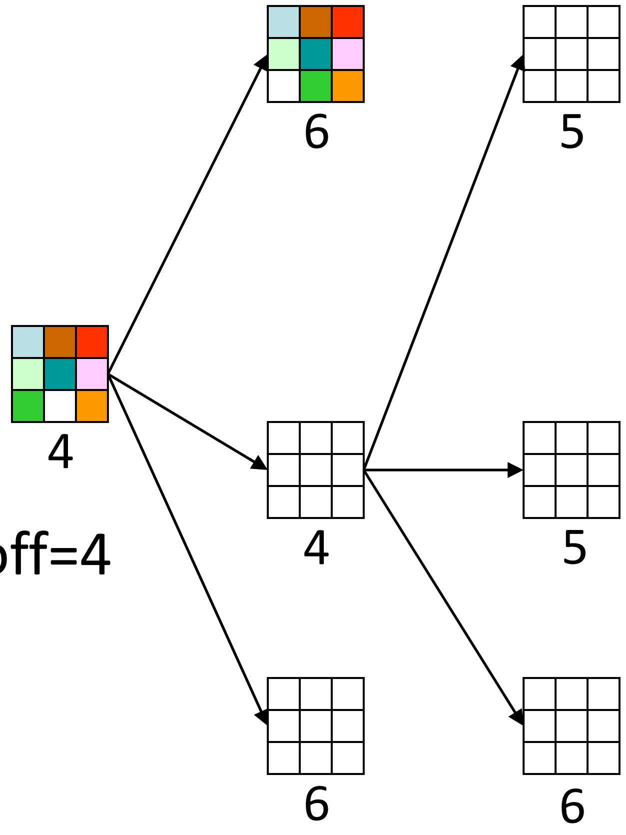




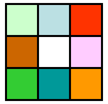
8-Puzzle

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

with $h(N)$ = number of misplaced tiles



Cutoff=4

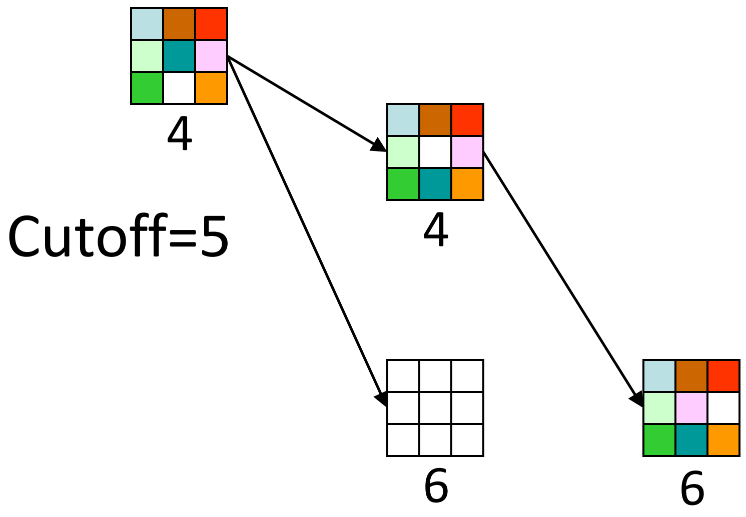


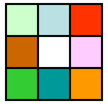
8-Puzzle

f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles

8-Puzzle

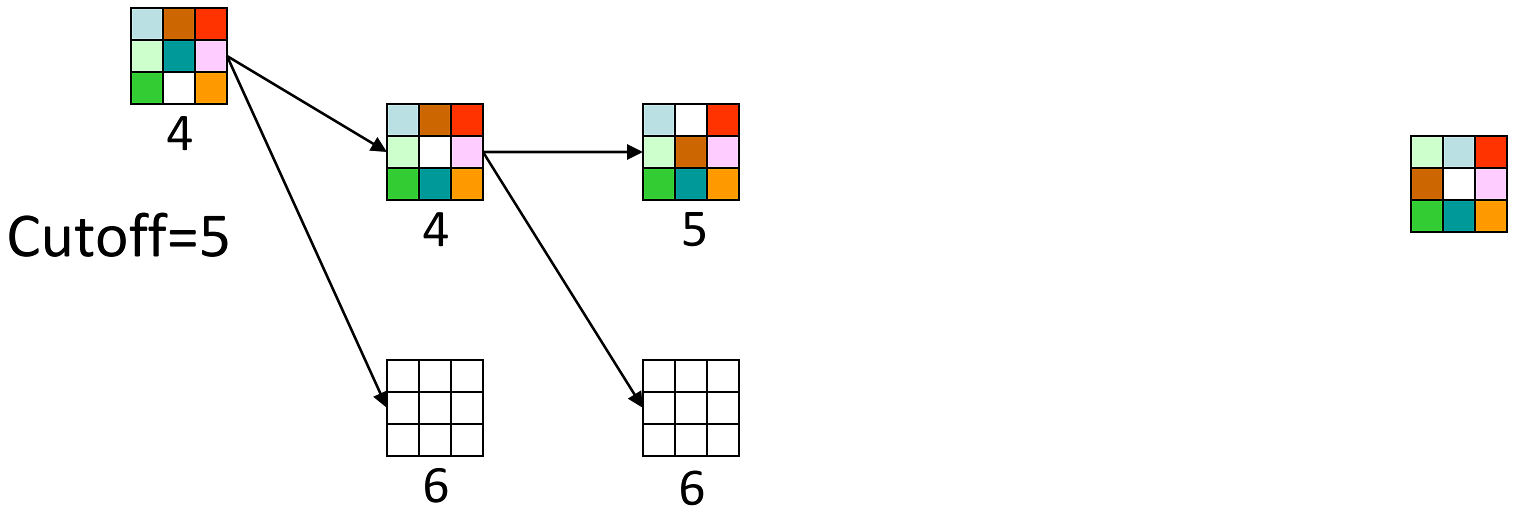
f(N) = g(N) + h(N)
 with h(N) = number of misplaced tiles





8-Puzzle

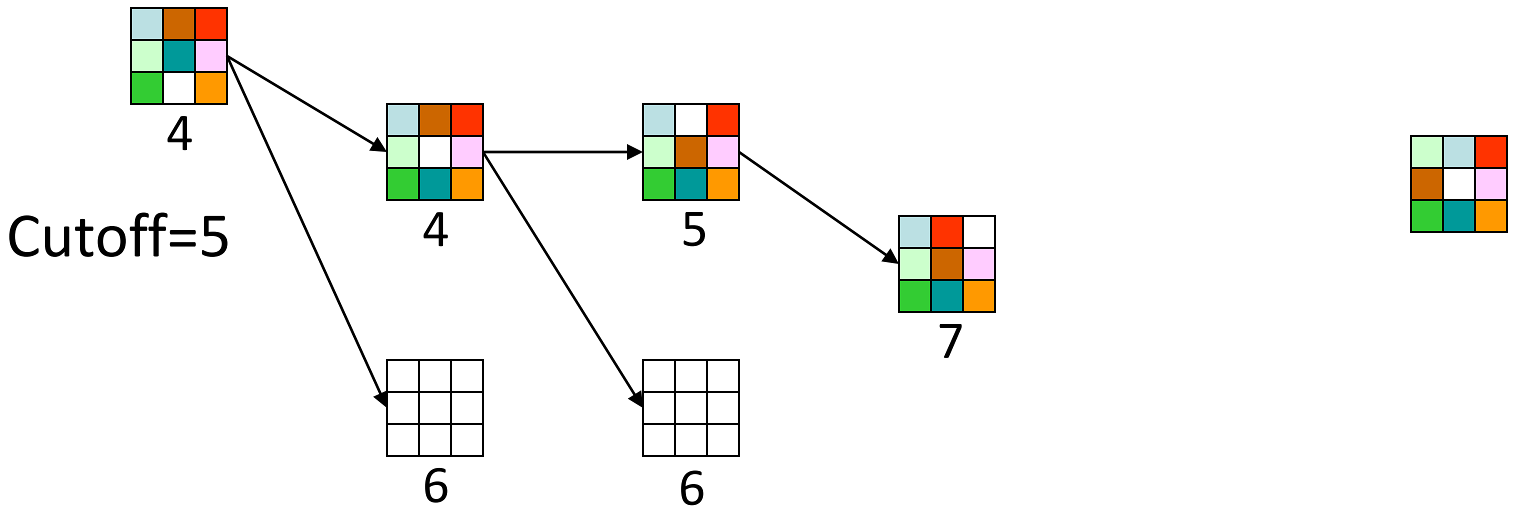
f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles



8-Puzzle

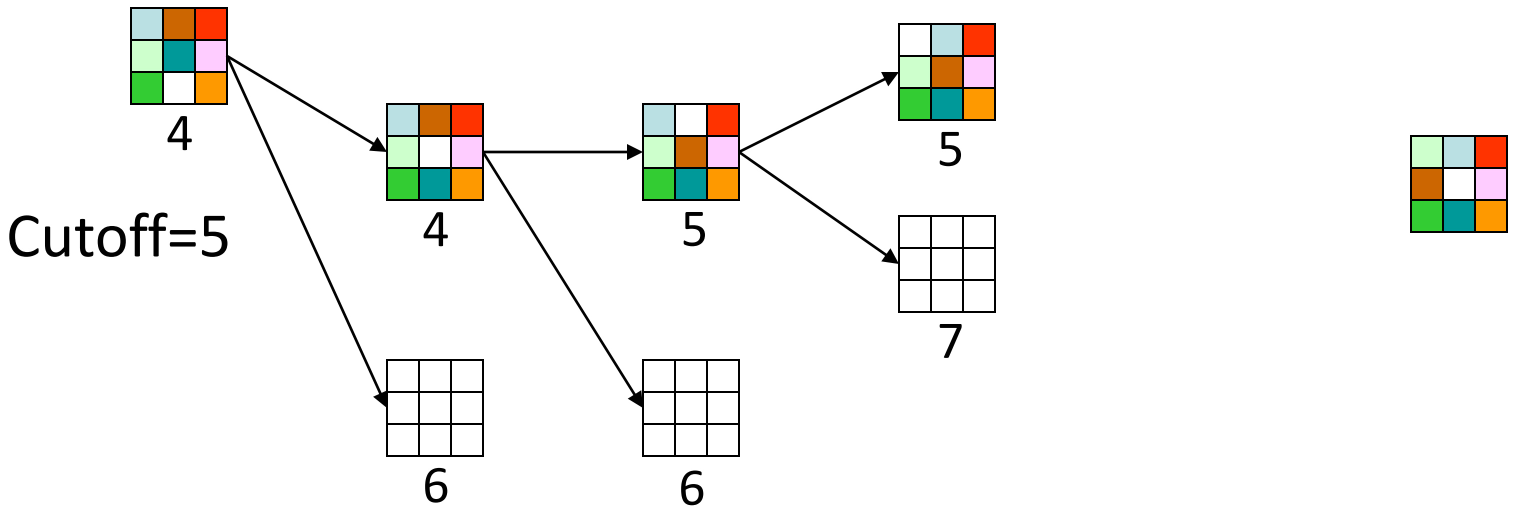
$f(N) = g(N) + h(N)$

with $h(N)$ = number of misplaced tiles



8-Puzzle

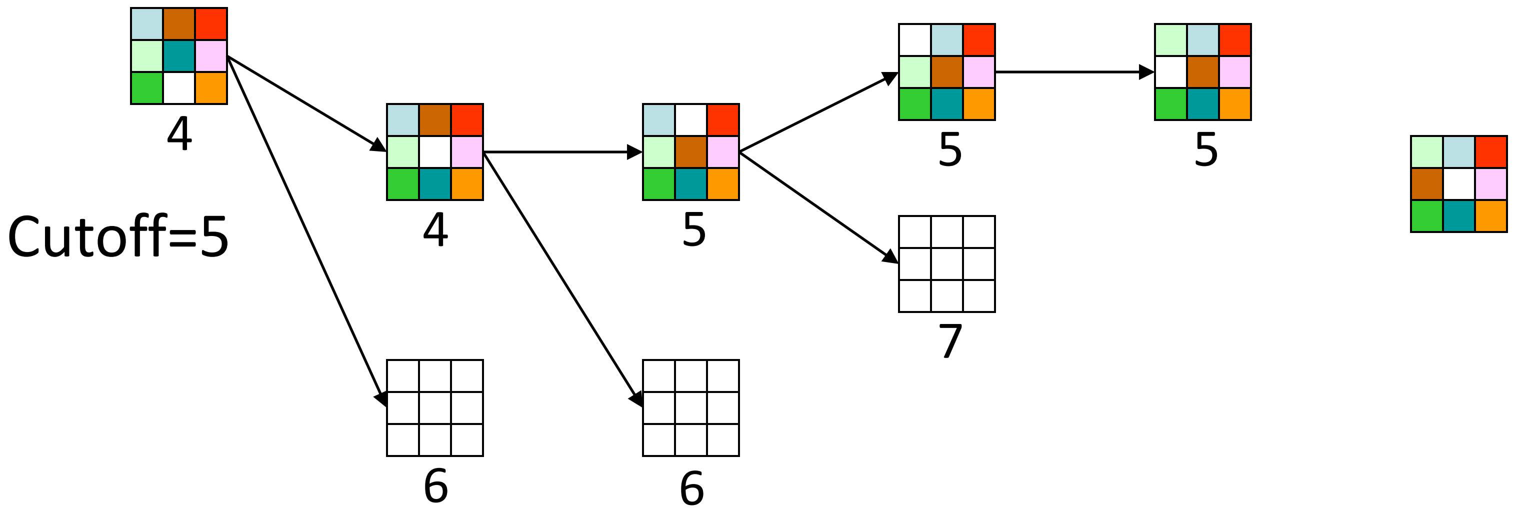
$f(N) = g(N) + h(N)$
with $h(N)$ = number of misplaced tiles



8-Puzzle

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

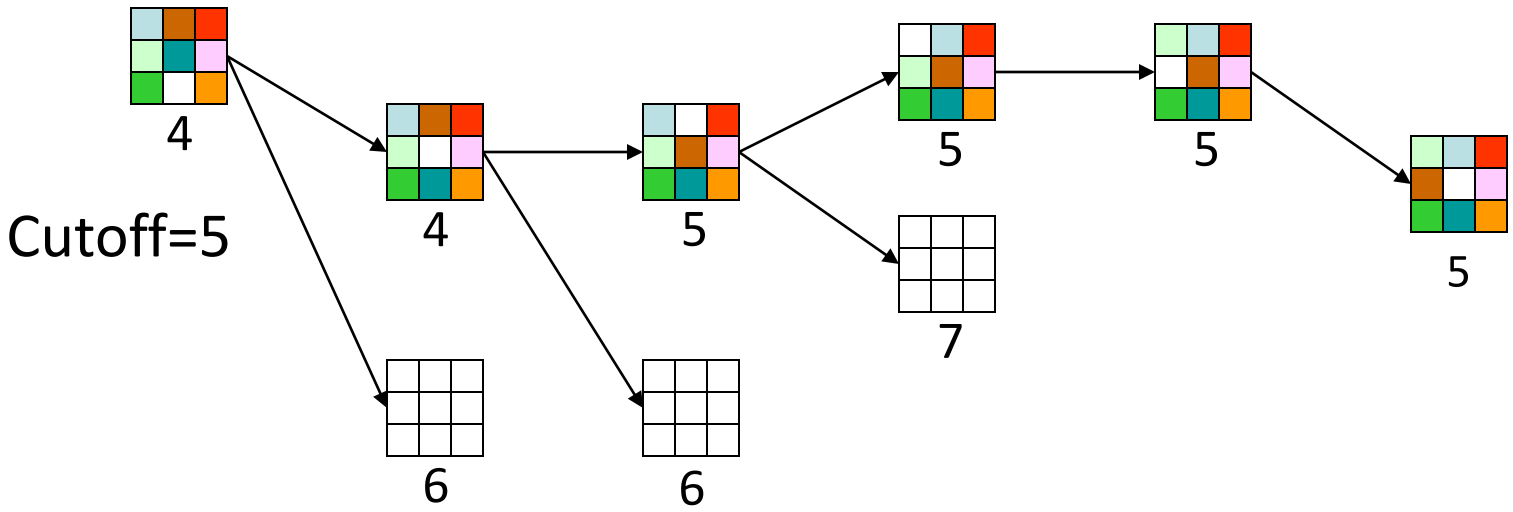
with $h(N)$ = number of misplaced tiles



8-Puzzle

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

with $h(N)$ = number of misplaced tiles



Advantages/Drawbacks of IDA*

- Advantages:
  - Still complete and optimal
  - Requires less memory than A*
  - Avoid the overhead to sort the fringe
- Drawbacks:
  - Can’t avoid revisiting states not on the current path
  - Available memory is poorly used
    (→ memory-bounded search, see R&N p. 101-104)

Tree Search Pseudo-Code

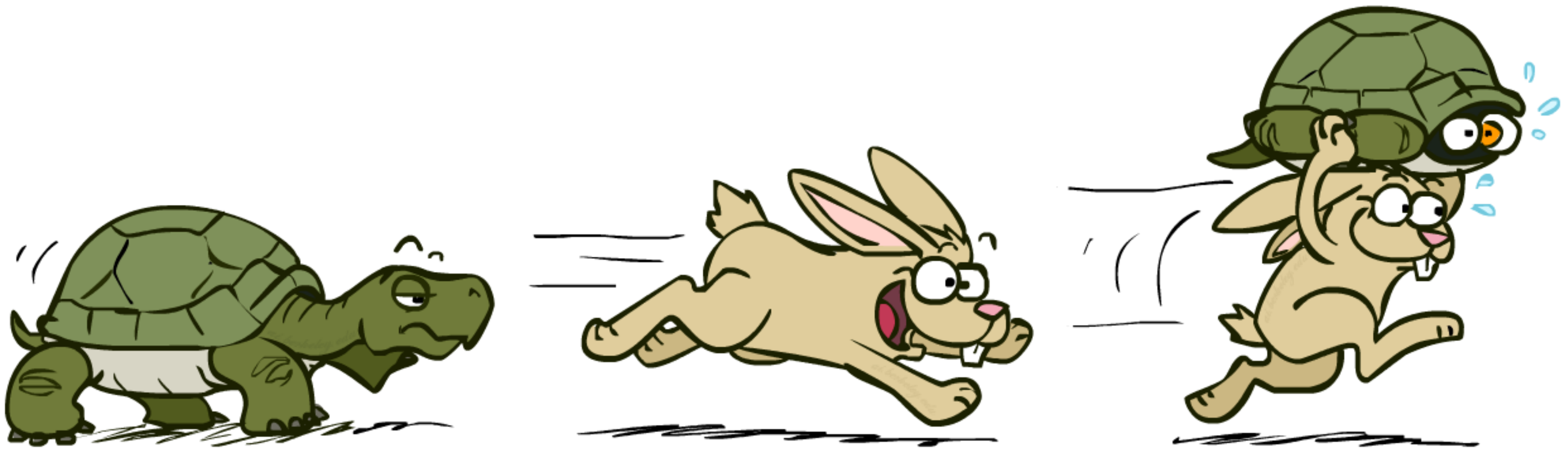
```
function TREE-SEARCH(problem, fringe) return a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        for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

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



A*: Summary