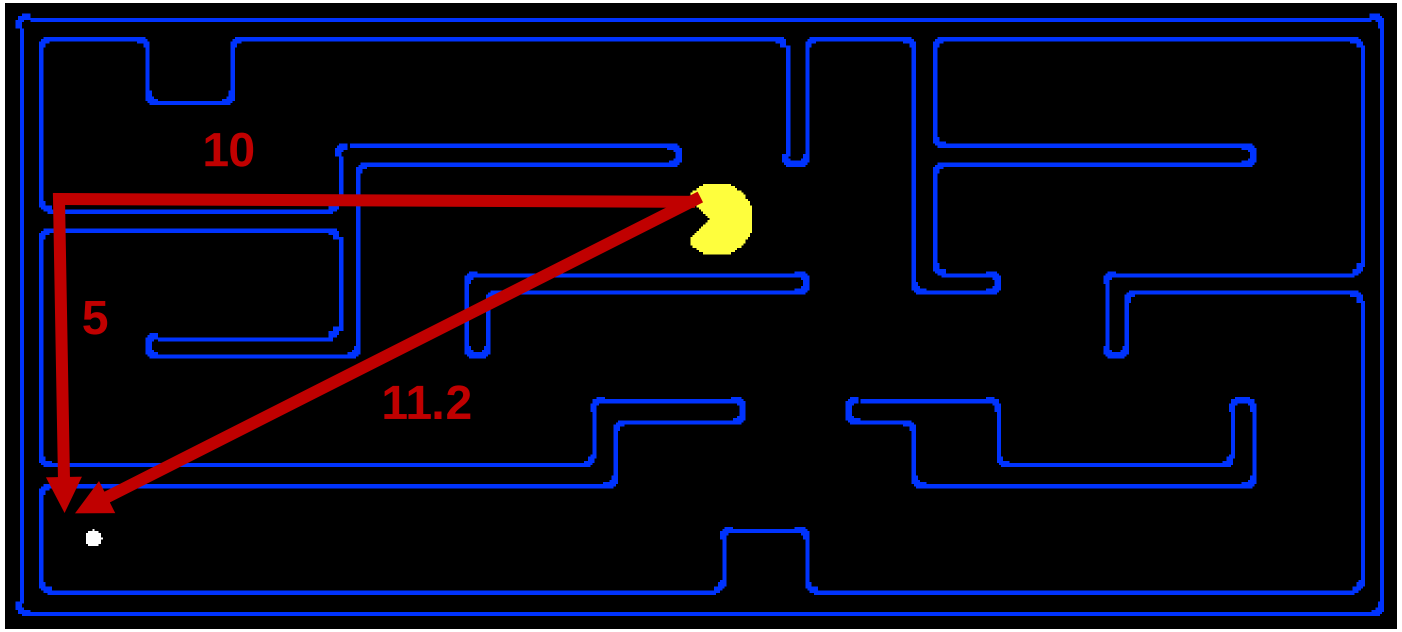
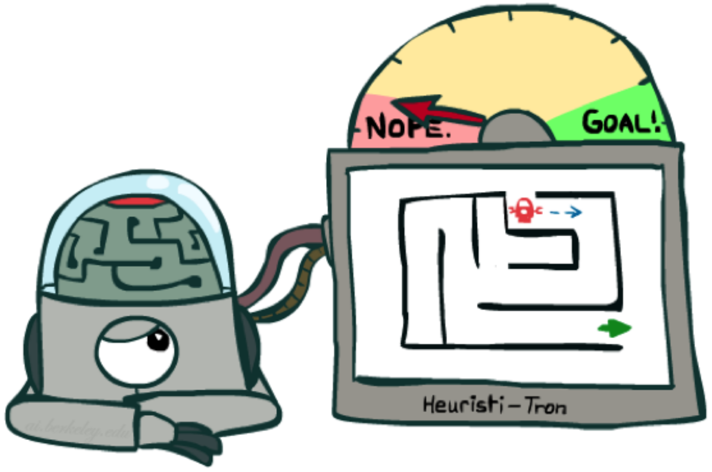
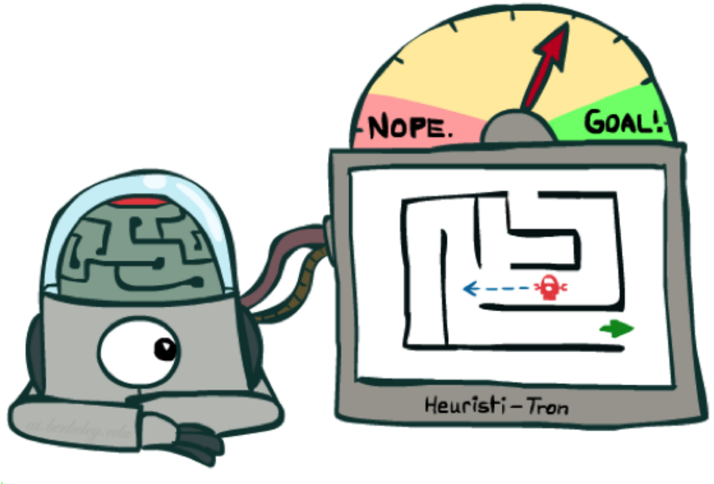
Search Heuristics

- 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



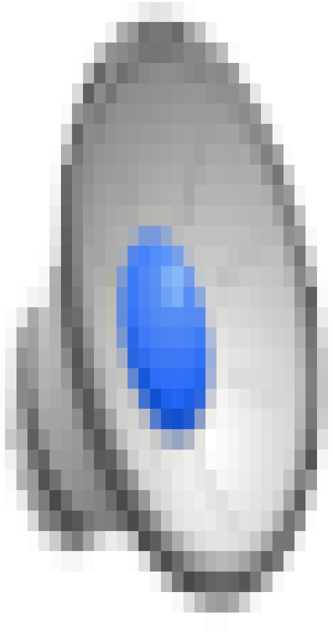




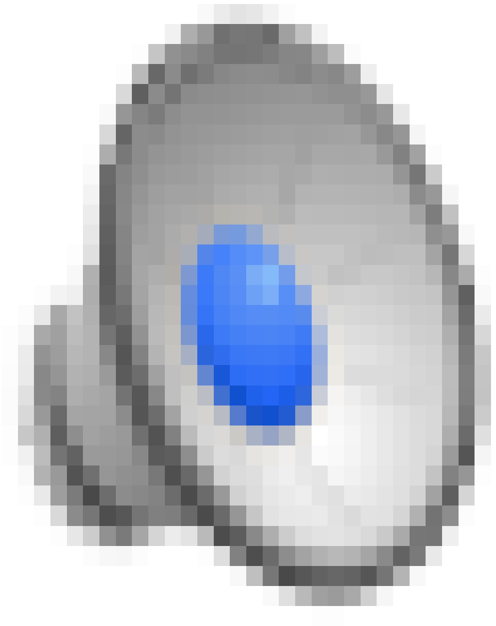
Greedy best-first search

- **Greedy (best-first) search** tries to expand the node that is closest to the goal, because this is likely to lead to a solution quickly
- Thus, the evaluation function is $f(n) = h(n)$
- E.g., in minimizing road distances, a heuristic lower bound for distances of cities is their straight-line (Euclidean) distance
- Greedy search ignores the cost of the path that has already been traversed to reach n
- Therefore, the solution given is not necessarily optimal

Demo Contours Greedy (Empty)



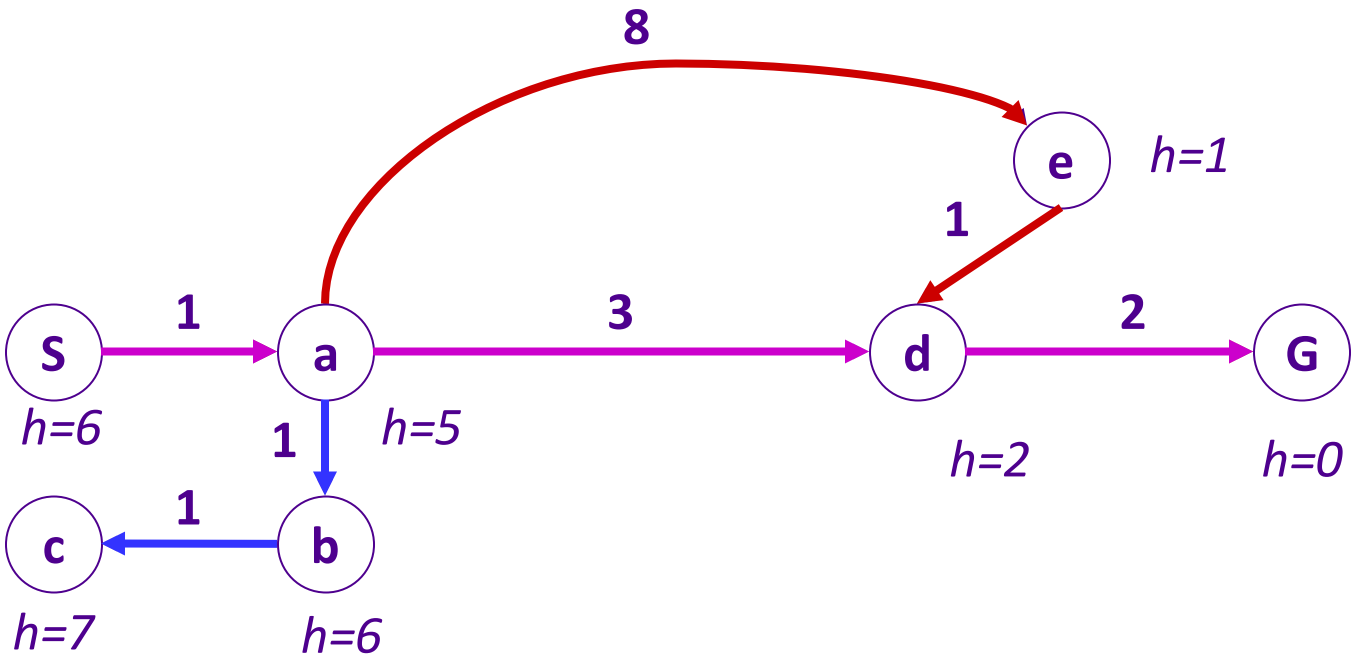
Demo Contours Greedy (Pacman Small Maze)

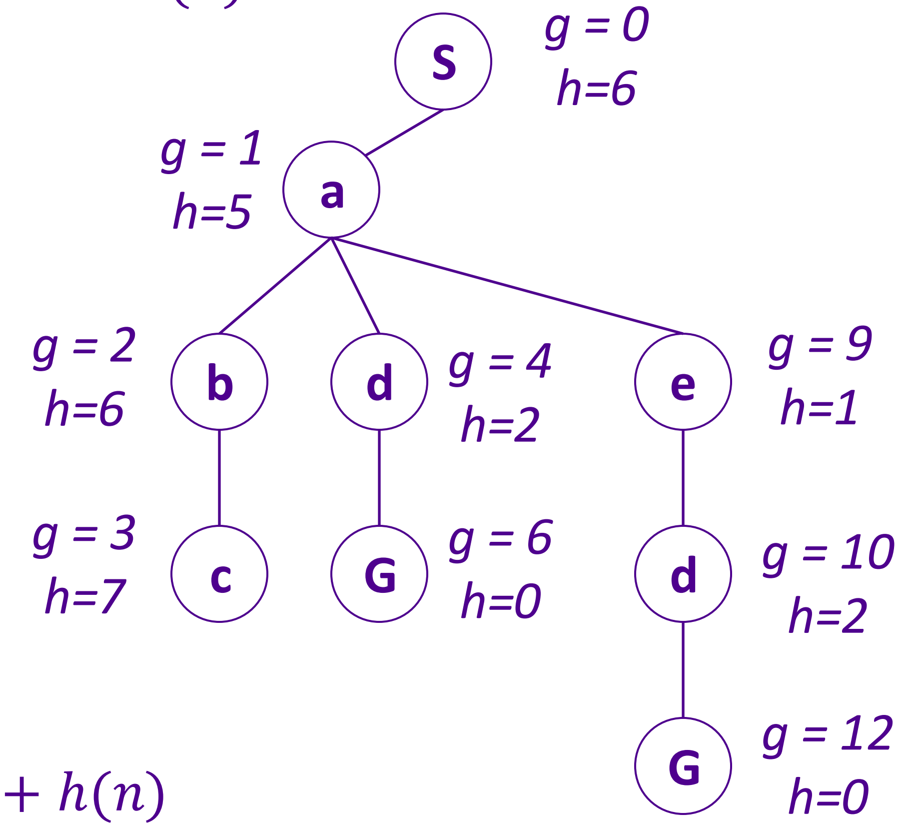


- Because greedy search can start down an infinite path and never return to try other possibilities, it is incomplete
- Because of its greediness the search makes choices that can lead to a dead end; then one backs up in the search tree to the deepest unexpanded node
- Greedy search resembles DFS in the way it prefers to follow a single path all the way to the goal, but will back up when it hits a dead end
- The worst-case time and space complexity is $O(b^m)$
- The quality of the heuristic function determines the practical usability of greedy search

Combining UCS and Greedy

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





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

Example: Teg Grenager

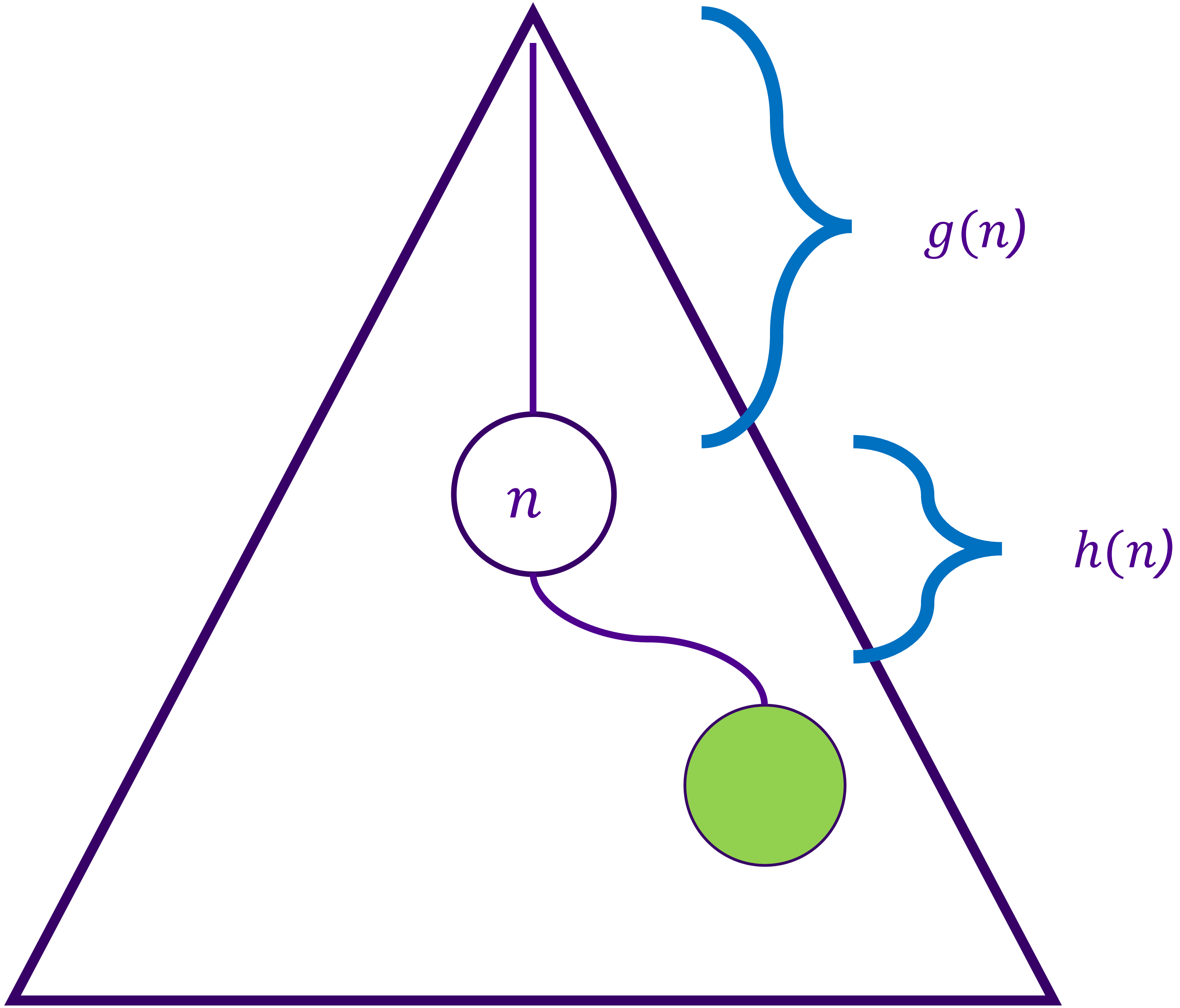
A* search

- A* combines the value of
  - the heuristic function $h(n)$ and
  - the cost to reach the node n, $g(n)$
- Evaluation function $f(n)$ thus estimates the cost of the cheapest solution through n
- A* tries the node with the lowest $f(n)$ value first
- This leads to both complete and optimal search algorithm, provided that $h(n)$ satisfies certain conditions
  - $h(n)$ is ***admissible*** if it never overestimates the cost to reach the goal

The actual cost that has been paid to reach node n

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

An estimate of the cost that still needs to be paid to reach the goal from node n


$g(n)$
n
$h(n)$

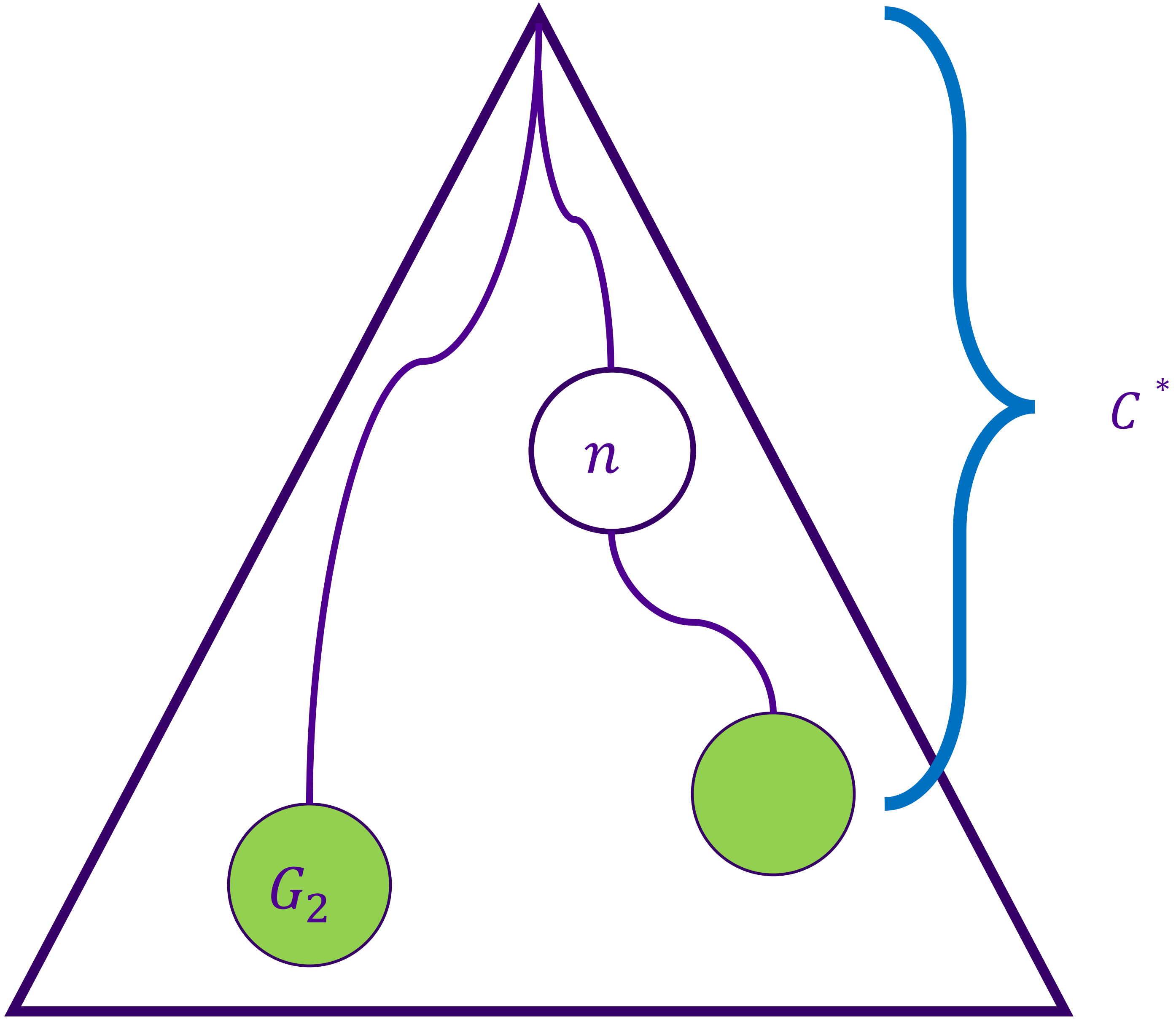
Optimality of A*

Provided that $h(n)$ is admissible, then in tree search A* gives the optimal solution

- Let G_2 be a suboptimal goal node generated to the tree
- Let C^* be the cost of the optimal solution
- Because G_2 is a goal node, it holds that $h(G_2) = 0$,

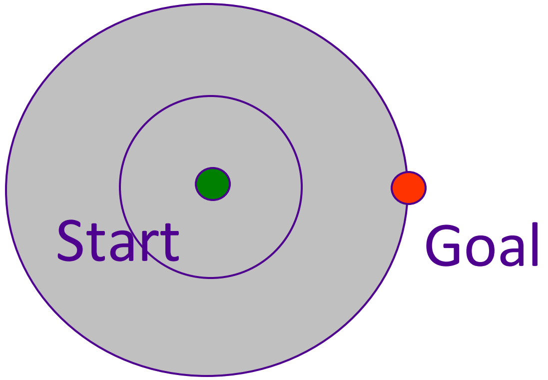
and we know that $f(G_2) = g(G_2) > C^*$

- On the other hand, if a solution exists, there must exist a node n that is on the optimal solution path in the tree
- Because $h(n)$ does not overestimate the cost of completing the solution path, $f(n) = g(n) + h(n) \leq C^*$
- We have shown that $f(n) \leq C^* < f(G_2)$, so G_2 will not be expanded and A* must return an optimal solution

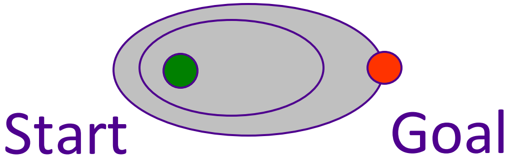

n
G_2
C^*

UCS vs A* Contours

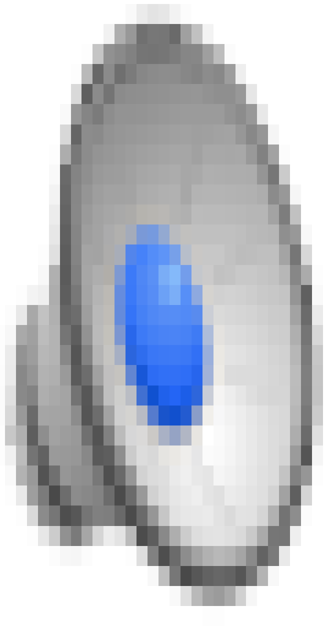
- Uniform-cost expands equally in all “directions”



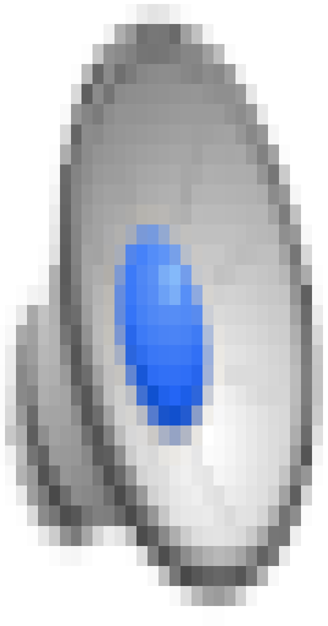
- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



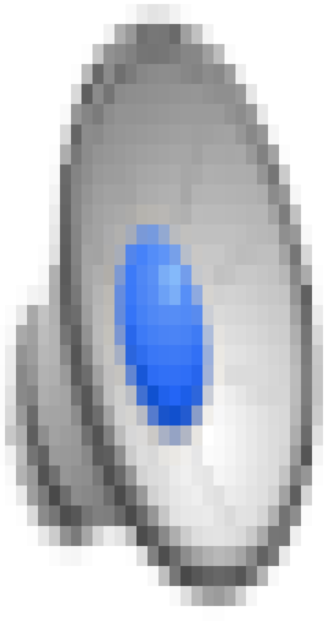
Demo Contours (Empty) -- UCS



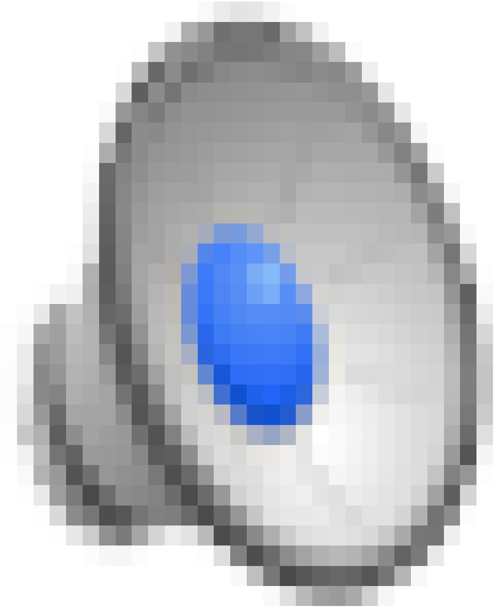
Demo Contours (Empty) -- Greedy



Demo Contours (Empty) – A*



Demo Contours (Pacman Small Maze) – A*



Comparison

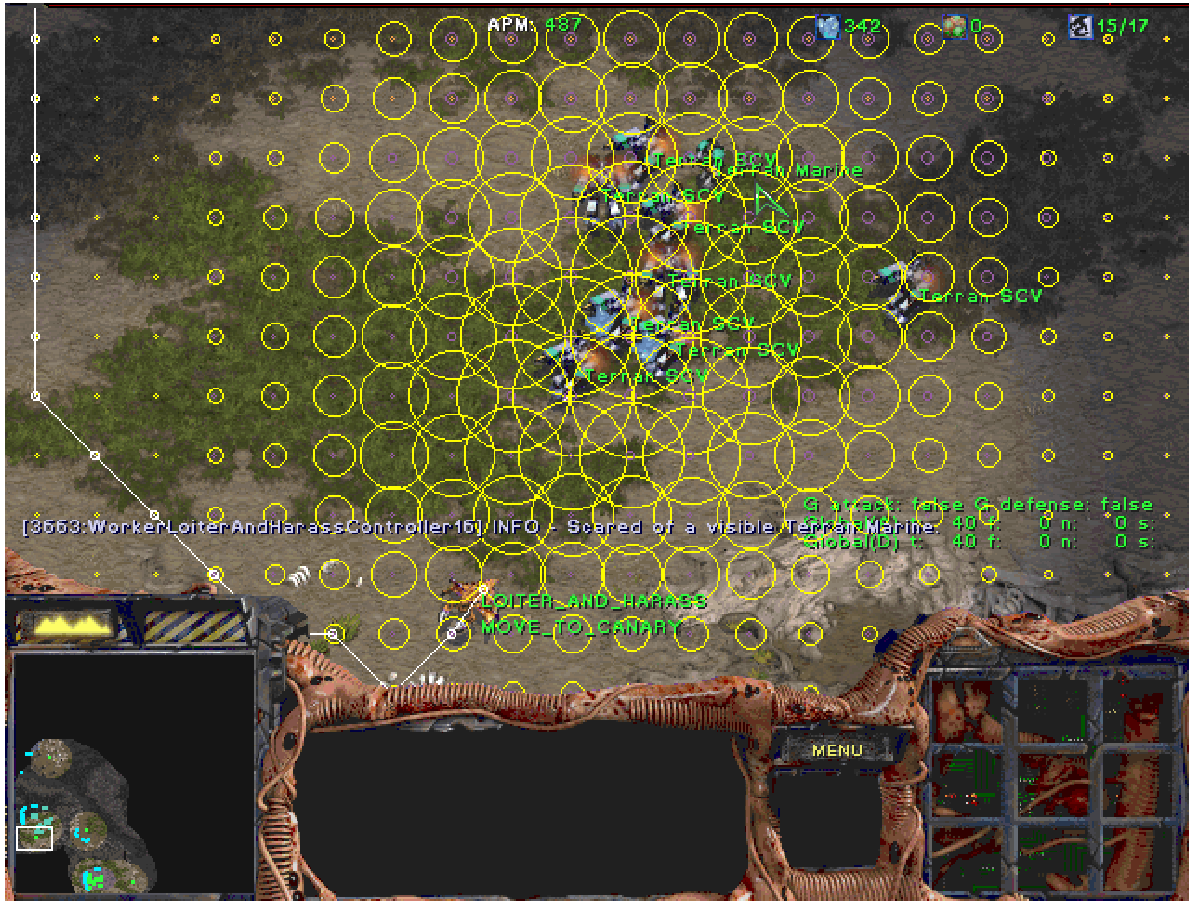
Greedy

Uniform Cost

A*

A* Applications

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



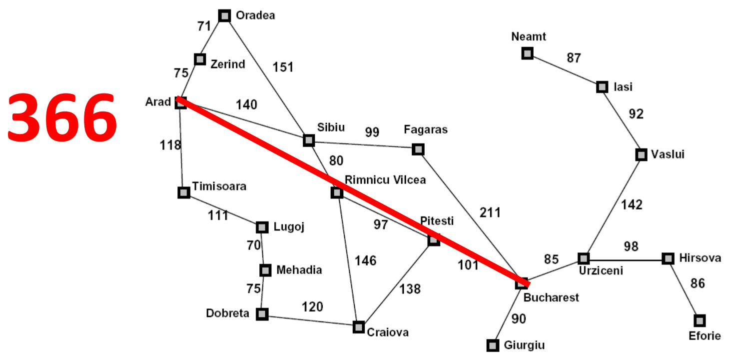
Memory-bounded heuristic search

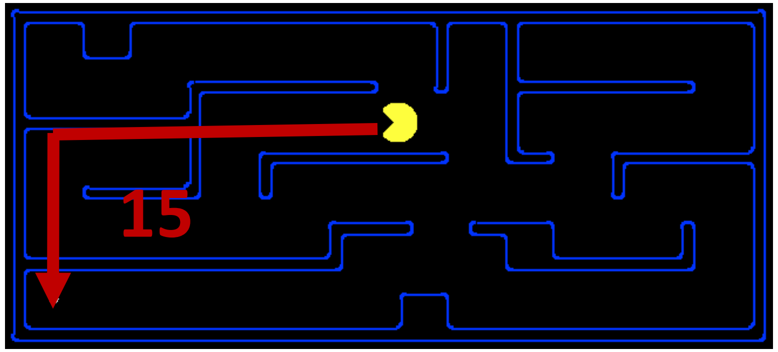
- Once again, the main drawback of search is not computation time, but rather space consumption
- Therefore, several memory-bounded variants of A* developed
- IDA* (Iterative Deepening A*) adapts the idea of iterative deepening
- The cutoff used is the f-cost ($g + h$) rather than the depth
- At each iteration the cutoff value is the smallest f-cost of any node that exceeded the cutoff on the previous iteration
- Subsequent more modern algorithms carry out more complex pruning

Creating Admissible Heuristics

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

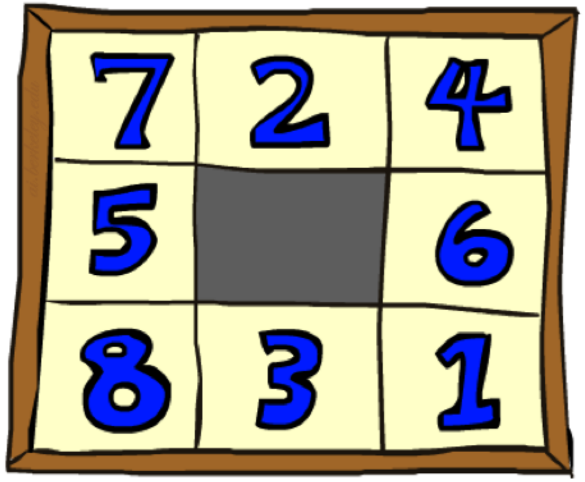
- Often, admissible heuristics are solutions to *relaxed problems,* where new actions are available



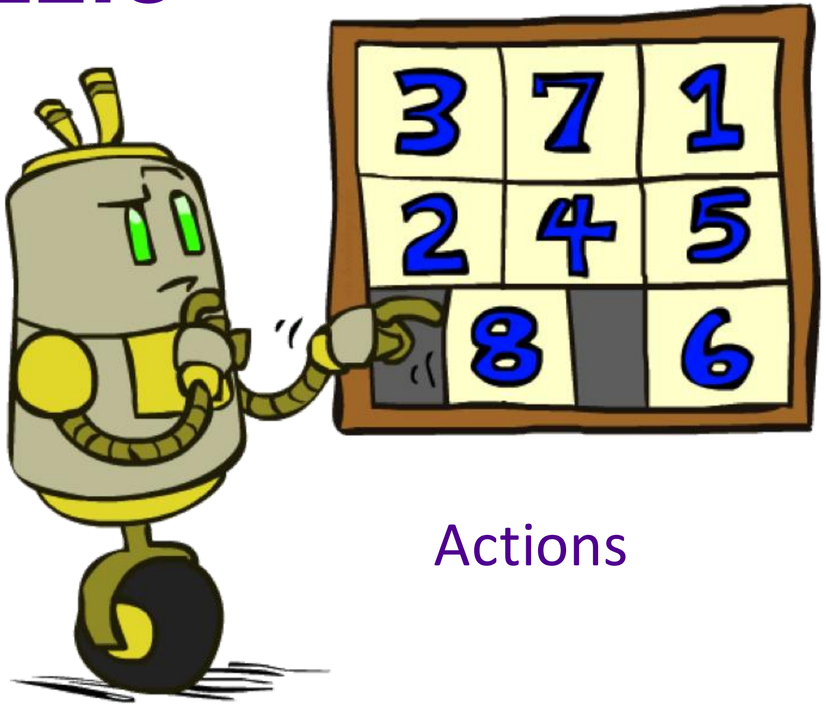


- Inadmissible heuristics are often useful too

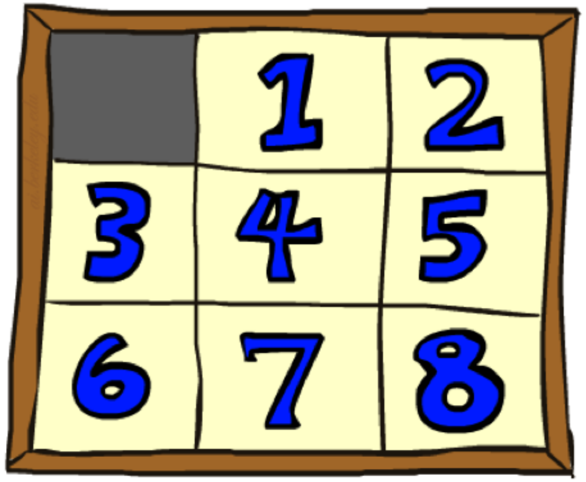
Example: 8 Puzzle



Start State



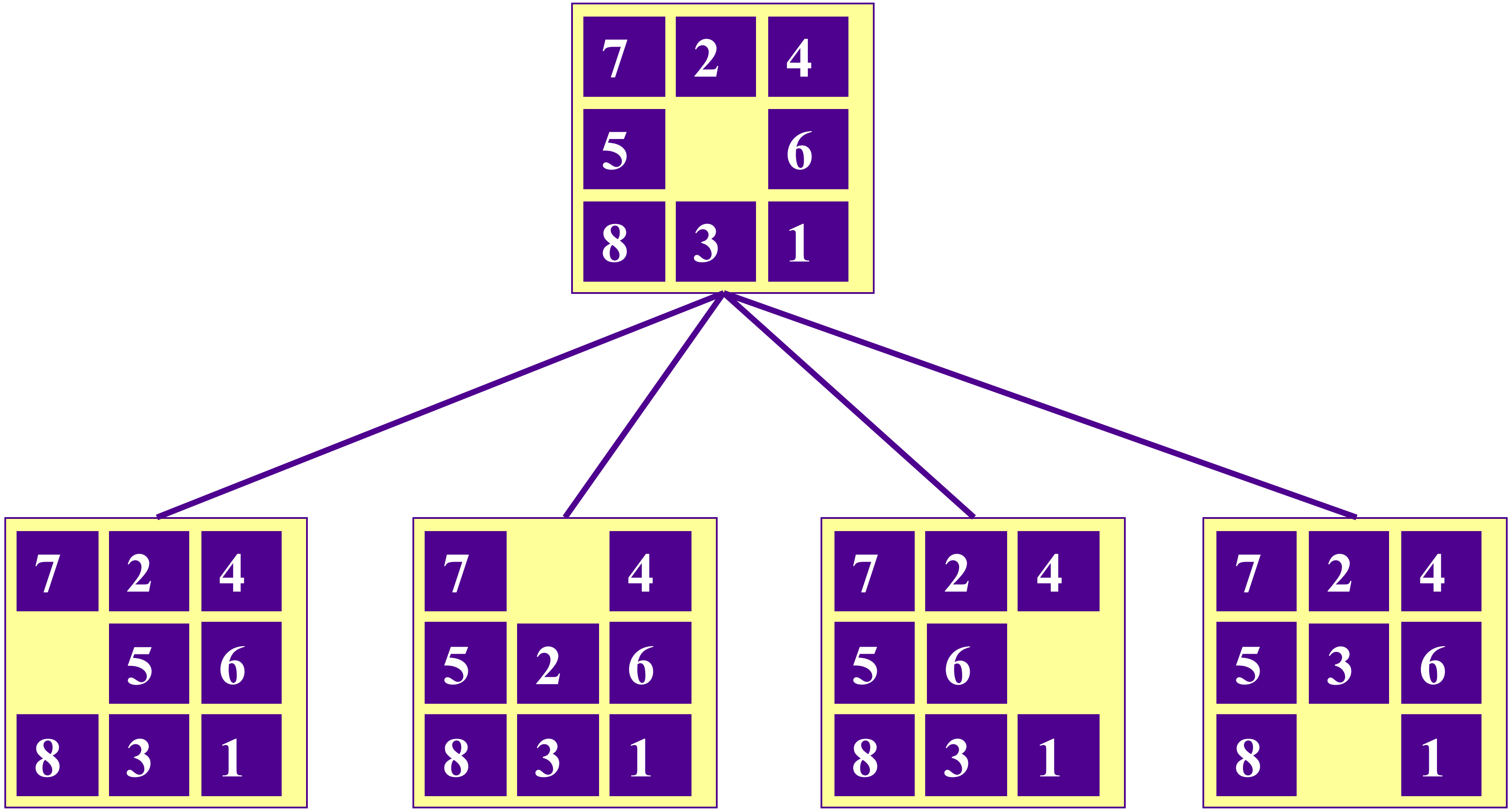
Actions



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

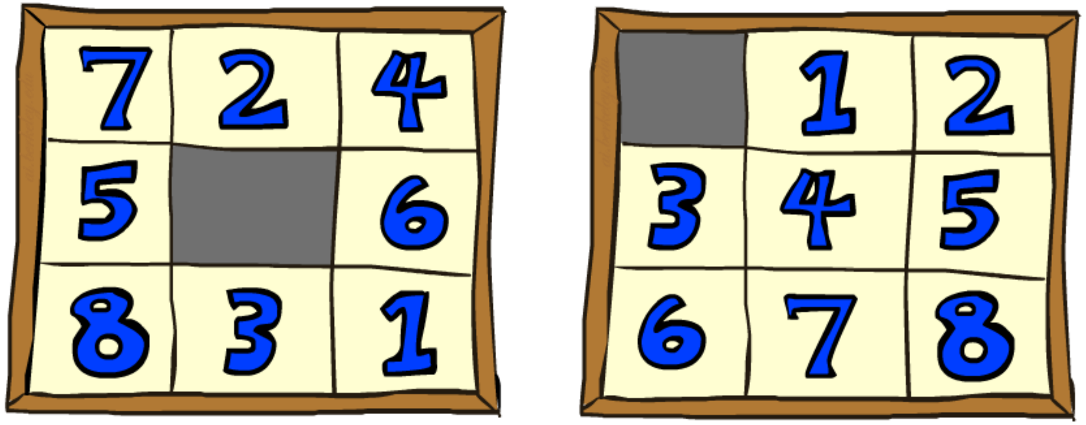


Heuristic functions

- In 8-puzzle we can define the following heuristic functions, which never overestimate:
  - h_1: the number of misplaced tiles: any tile that is out of place must be moved at least once to obtain the desired configuration
  - h_2: The sum of **Manhattan distances** of tiles from their goal position: the tiles need to be transported to their goal positions to reach the desired configuration

- In the initial configuration all tiles are out of their place: $h_1(s_1) = 8$
- The value of the second heuristic for the example is:

$$3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$$



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

TILES = h_1
MANHATTAN = h_2

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

Relaxation & heuristics

- We can study **relaxed** problems from which some restrictions of the original problem have been removed
- The cost of an optimal solution to a relaxed problem is an **admissible heuristic** for the original problem (does not over-estimate)
- The optimal solution in the original problem is, by definition, also a solution in the relaxed problem
- E.g., heuristic h_1 for the 8-puzzle gives perfectly accurate path length for a simplified version of the puzzle, where a tile can move anywhere
- Similarly, h_2 gives an optimal solution to a relaxed 8-puzzle, where tiles can move also to occupied squares

- If a collection of admissible heuristics is available for a problem, and none of them dominates any of the others, we can use the composite function

$$h(n) = \max\{\, h_1(n), \ldots, h_m(n) \,\}$$

- The composite function dominates all of its component functions and is **consistent** if none of the components overestimates
- One way of relaxing problems is to study subproblems

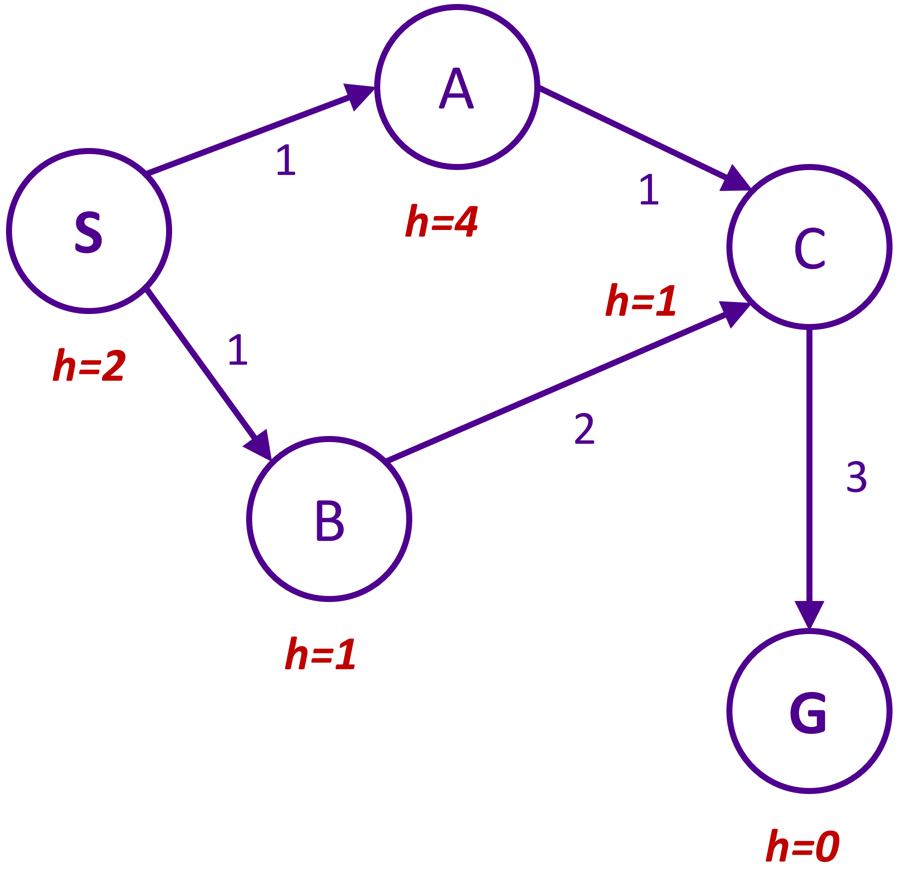
- E.g., in 8-puzzle we could study only four tiles at a time and let the other tiles wander to any position
- By combining the heuristics concerning distinct tiles into one composite function yields a heuristic function, that is much more efficient than the Manhattan distance

*	2	4
*		*
*	3	1

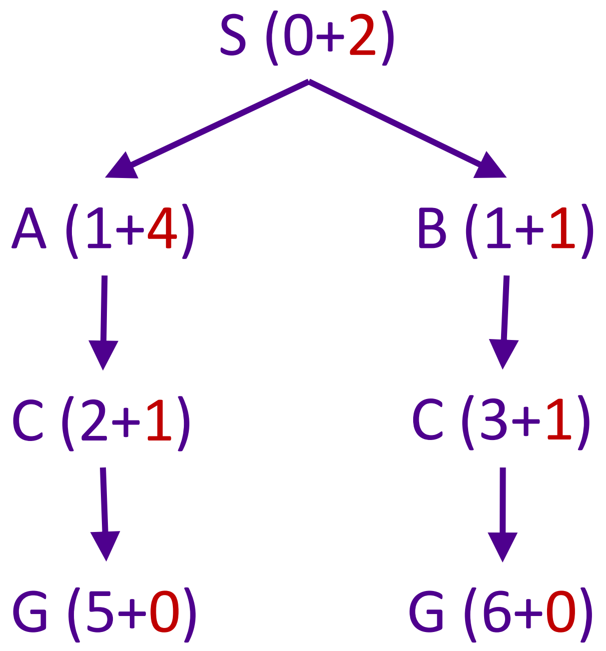
	1	2
3	4	*
*	*	*

A* Graph Search Gone Wrong

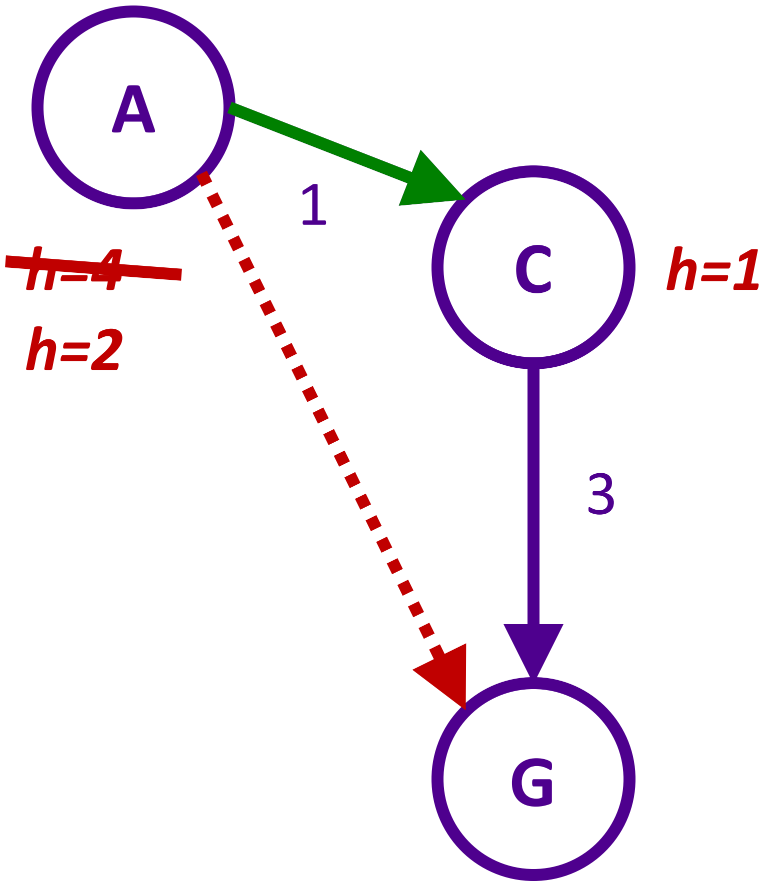
State space graph



Search tree



Consistency of Heuristics



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

    $h(A) \leq$ actual cost from A to G
  - **Consistency**: heuristic “arc” cost ≤ actual cost for each arc

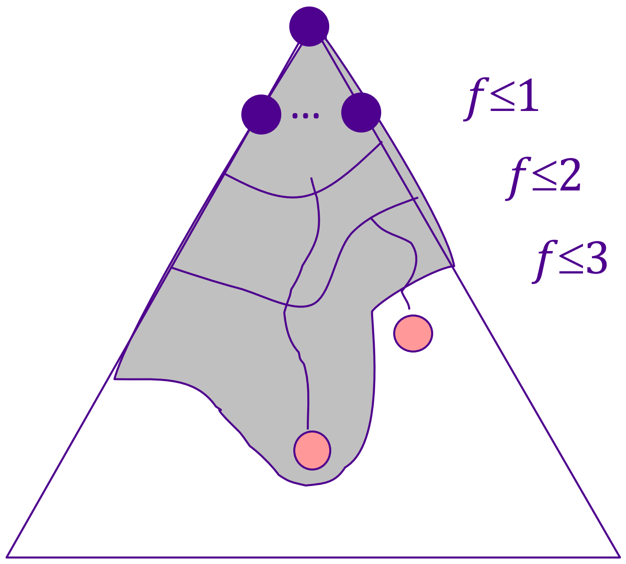
    $h(A) - h(C) \leq \text{cost}(A \text{ to } C)$

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

    $h(A) \leq \text{cost}(A \text{ to } C) + h(C)$
  - A* graph search is optimal

Optimality of A* Graph Search

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



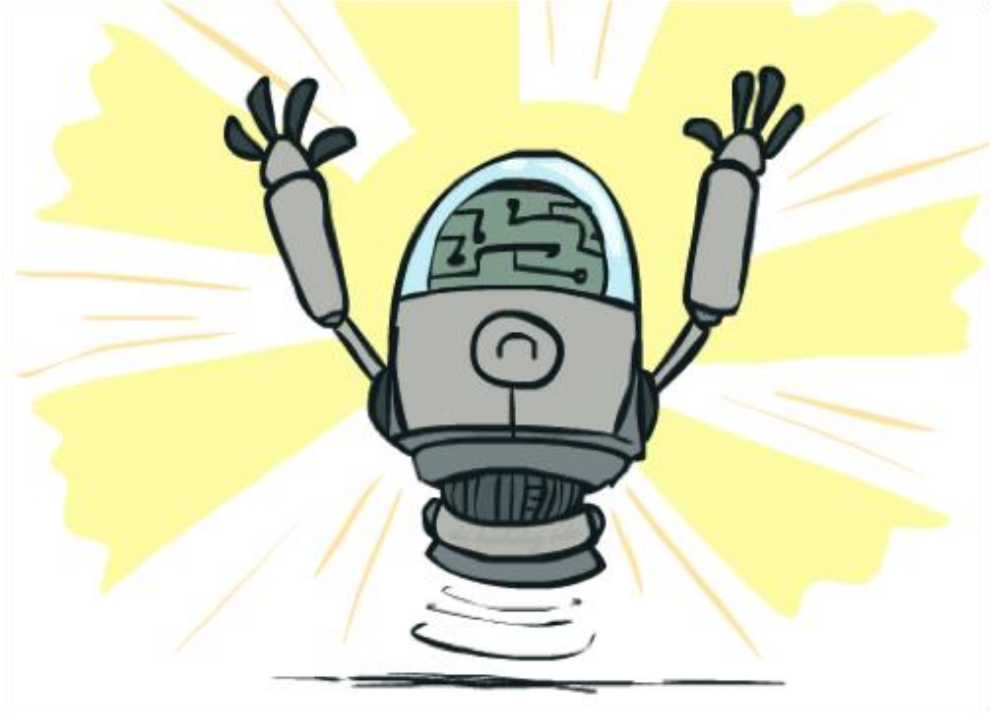
Optimality

- 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