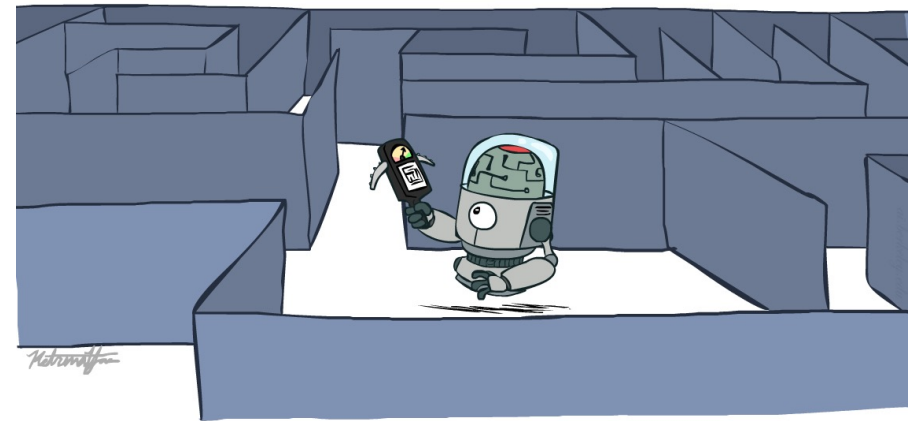


Artificial Intelligence - INFOF311

Informed search



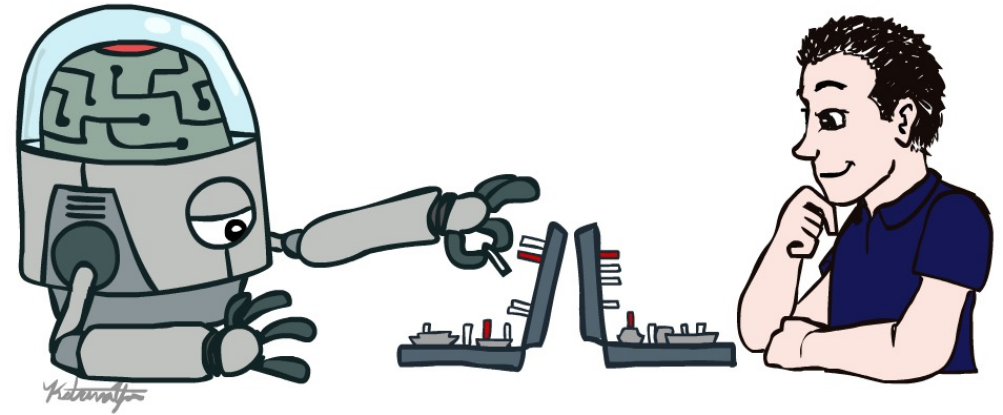
Instructor : Tom Lenaerts

Acknowledgement

We thank Stuart Russell for his generosity in allowing us to use the slide set of the UC Berkeley Course CS188, Introduction to Artificial Intelligence. These slides were created by Dan Klein, Pieter Abbeel and Anca Dragan for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at <http://ai.berkeley.edu>.

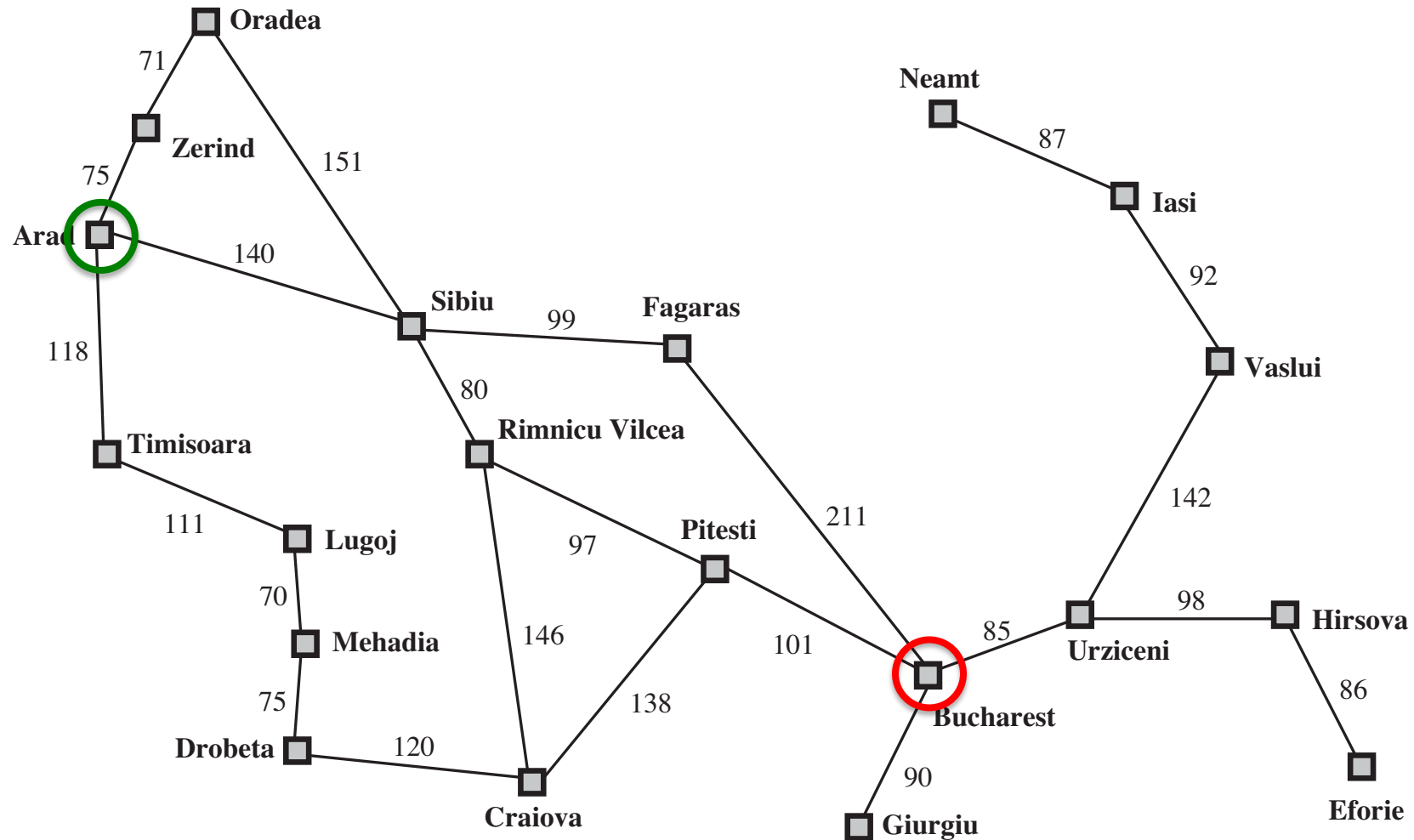


Center for
Human-Compatible
Artificial
Intelligence



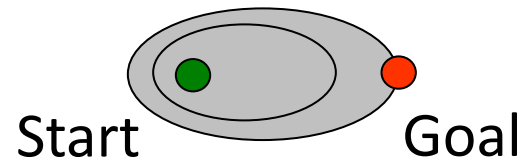
The slides for INFOF311 are slightly modified versions of the slides of the spring and summer CS188 sessions in 2021 and 2022

Example: route-finding in Romania

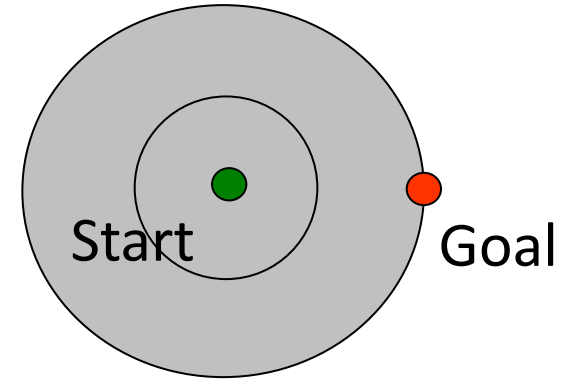


What we would like to have happen

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



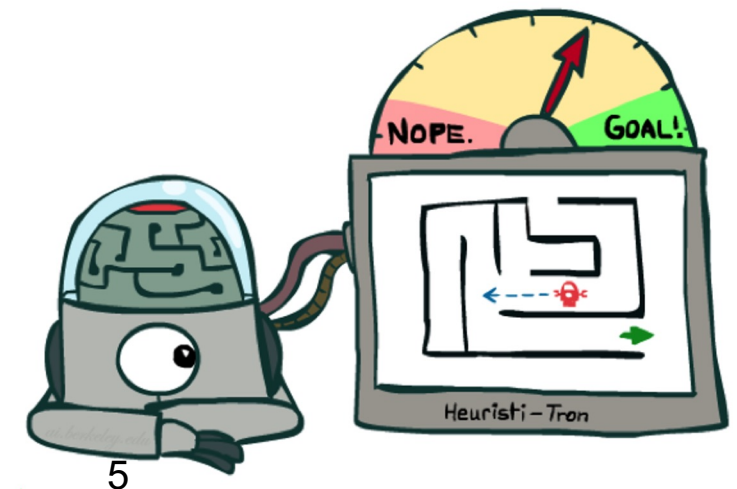
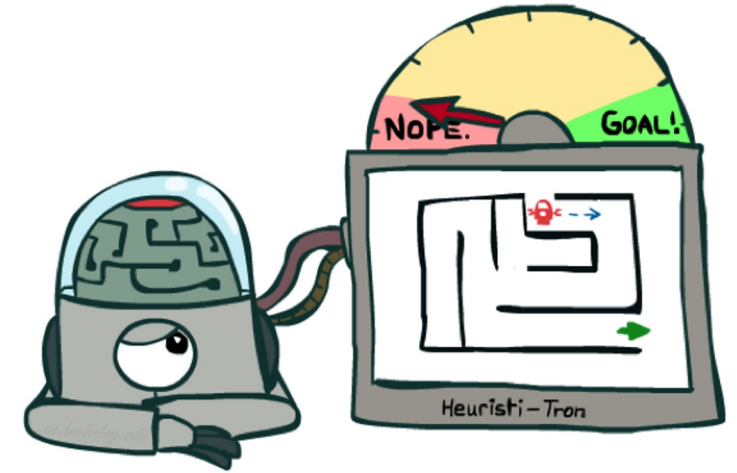
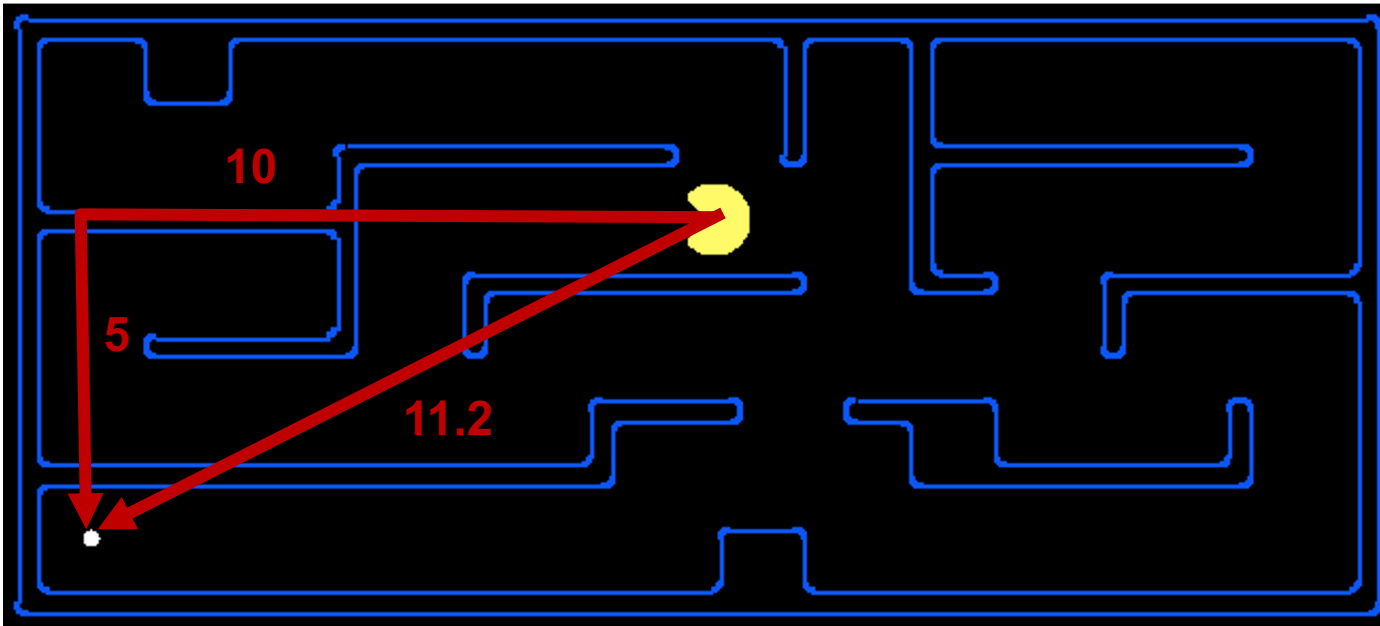
Informed



Uninformed

Search Heuristics

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

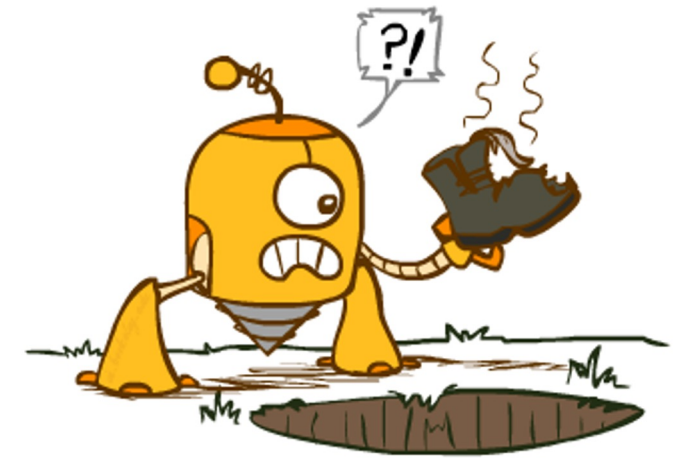
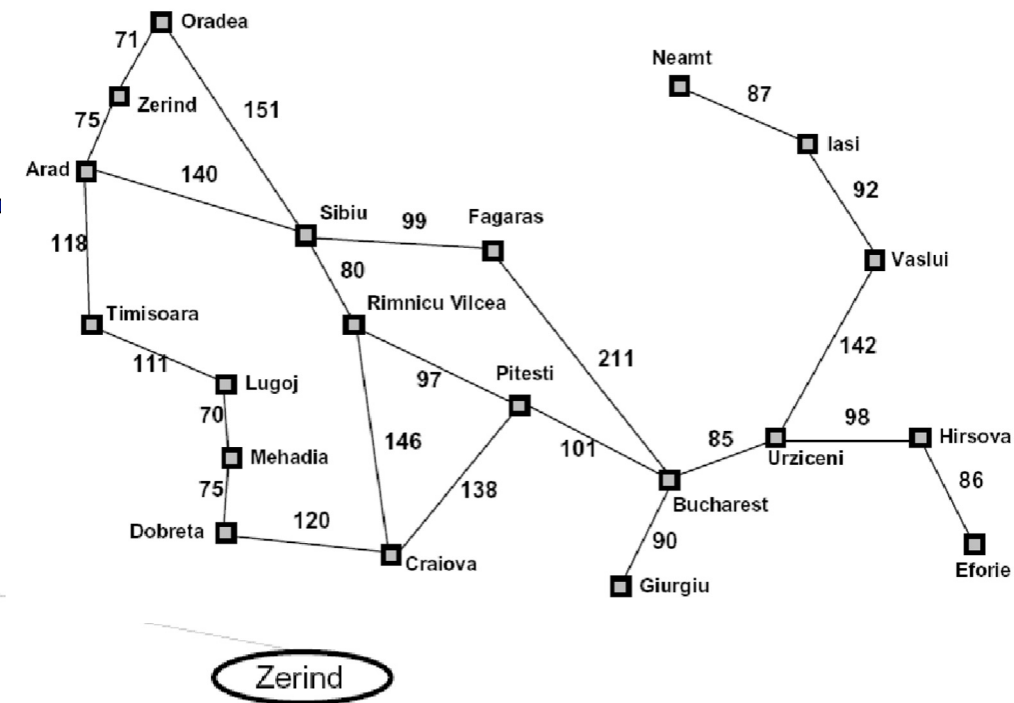
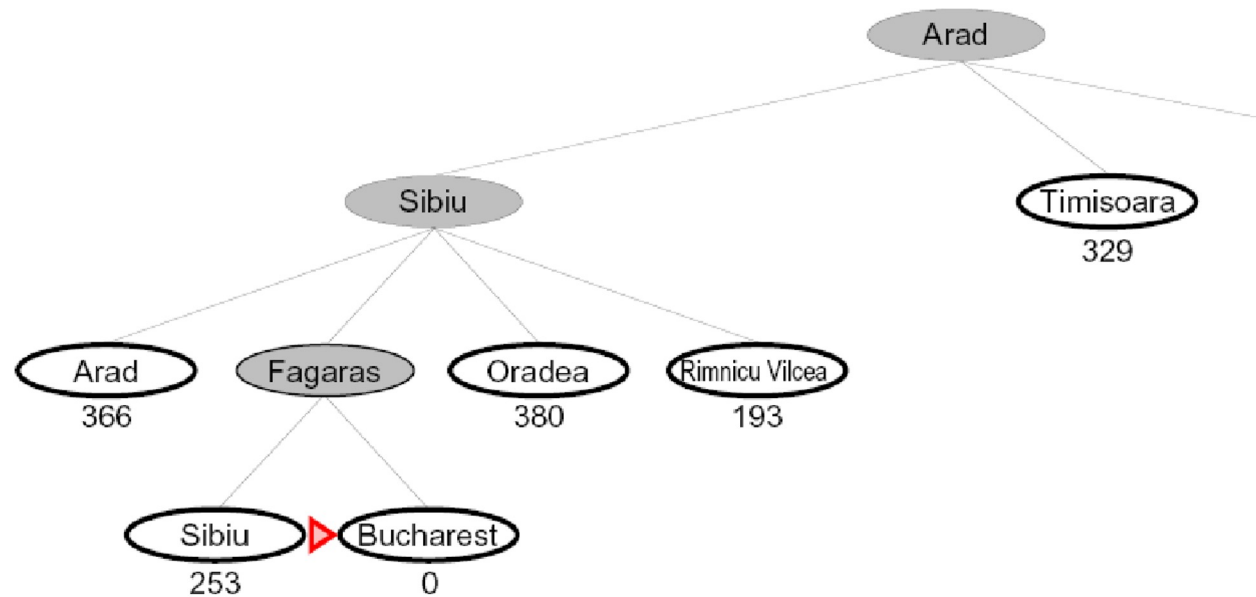


Greedy Search



Greedy Search

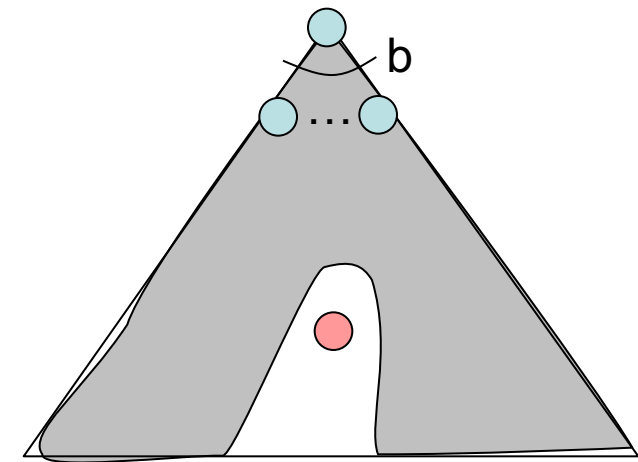
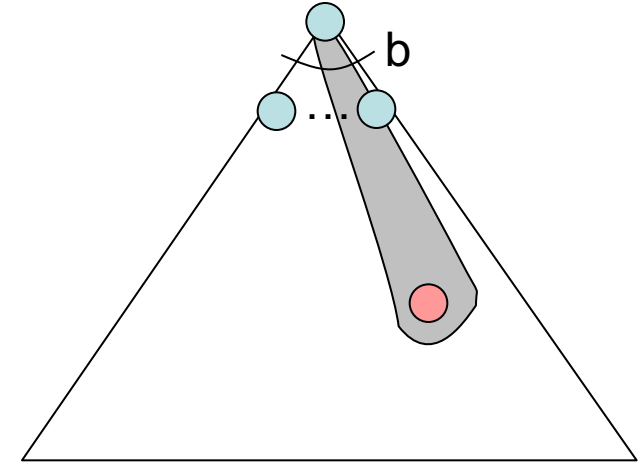
- Expand the node that seems closest...



- Is it optimal?
 - No. Resulting path to Bucharest is not the shortest!

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



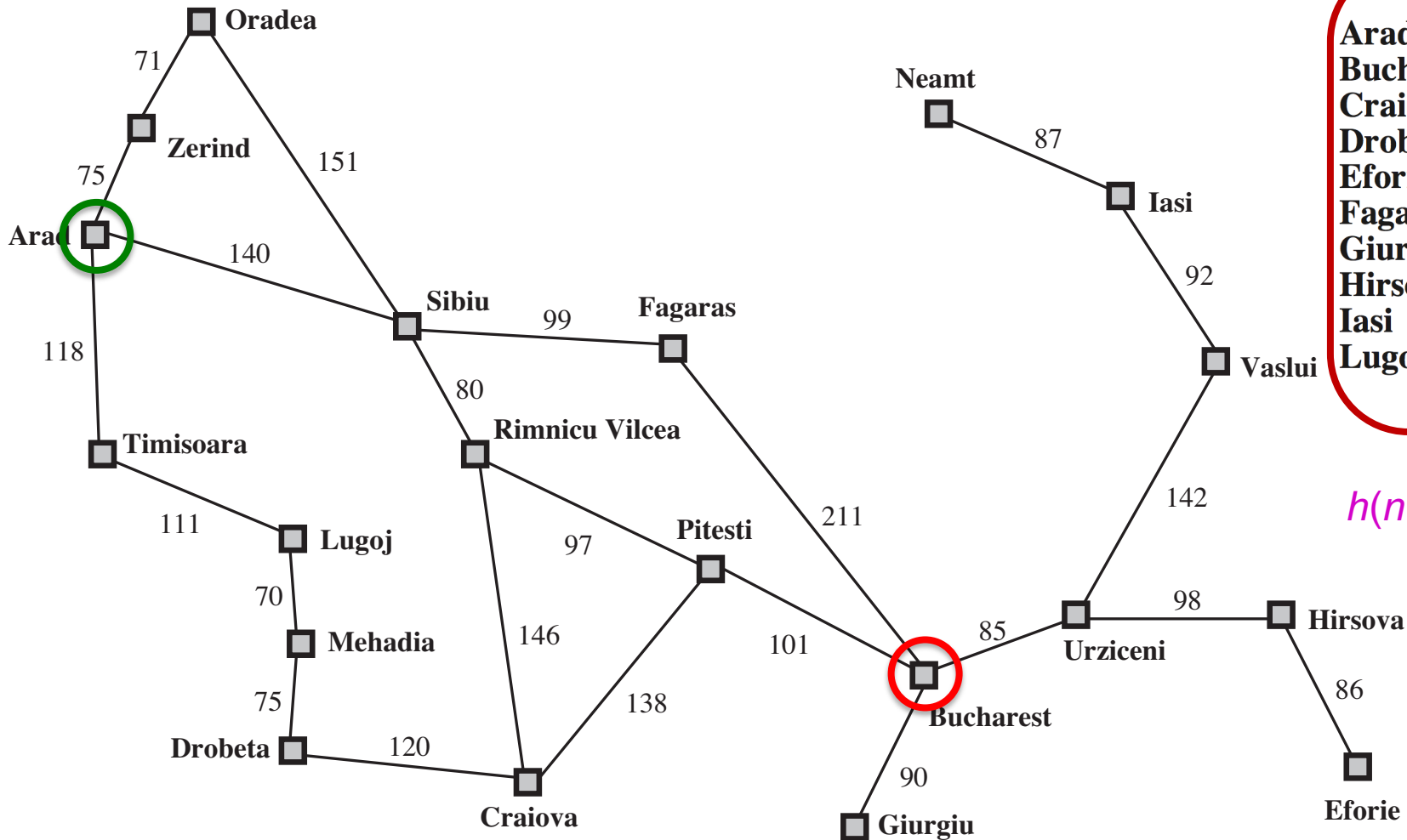
A* Search



A*: the core idea

- Expand a node n most likely to be on an optimal path
- Expand a node n s.t. the cost of the best solution through n is optimal
- Expand a node n with lowest value of $g(n) + h^*(n)$
 - $g(n)$ is the cost from root to n
 - $h^*(n)$ is the optimal cost from n to the closest goal
- We seldom know $h^*(n)$ but might have a heuristic approximation $h(n)$
- A* = tree search with priority queue ordered by $f(n) = g(n) + h(n)$

Example: route-finding in Romania

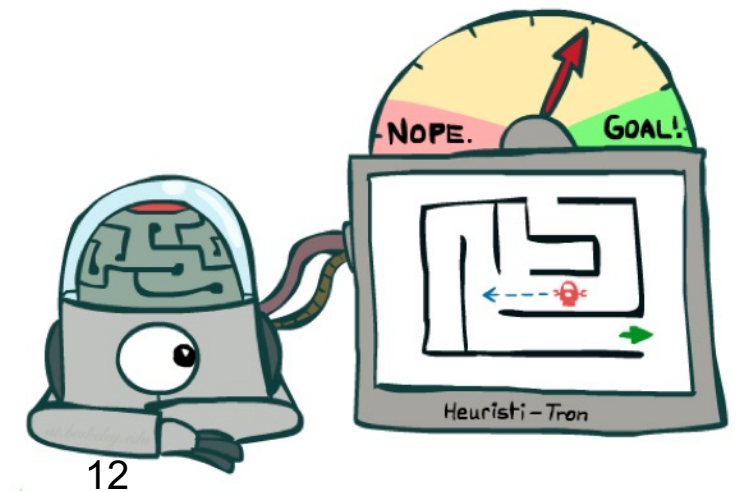
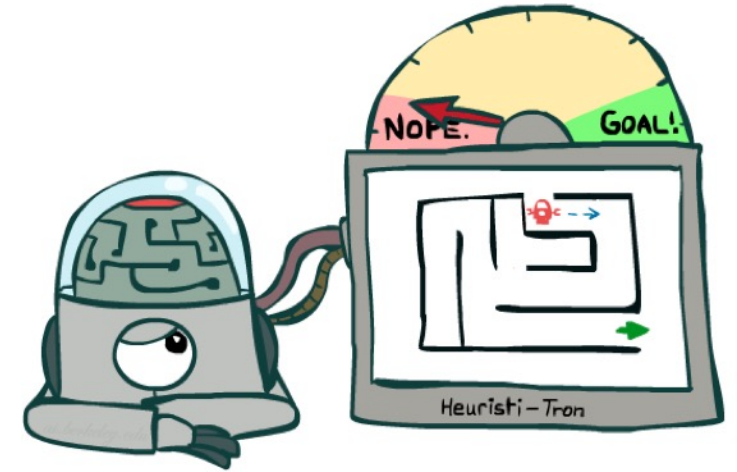
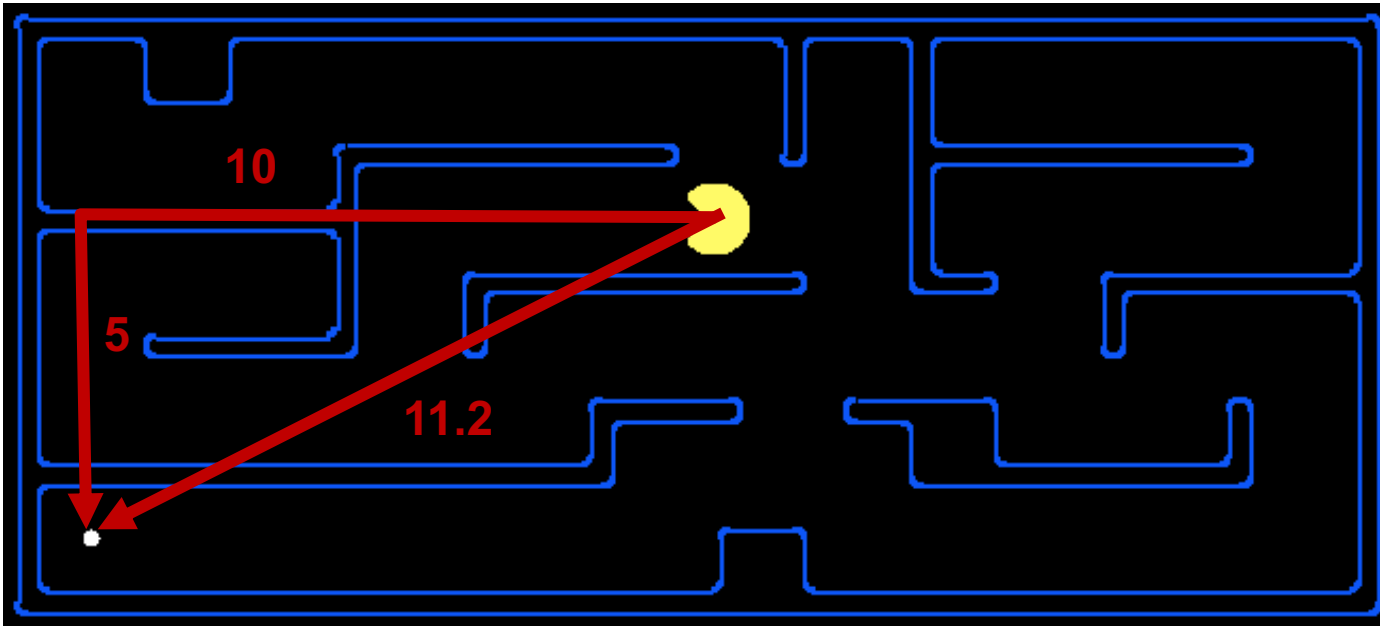


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

$h(n)$ = straight-line distance to Bucharest

Example: pathing in Pacman

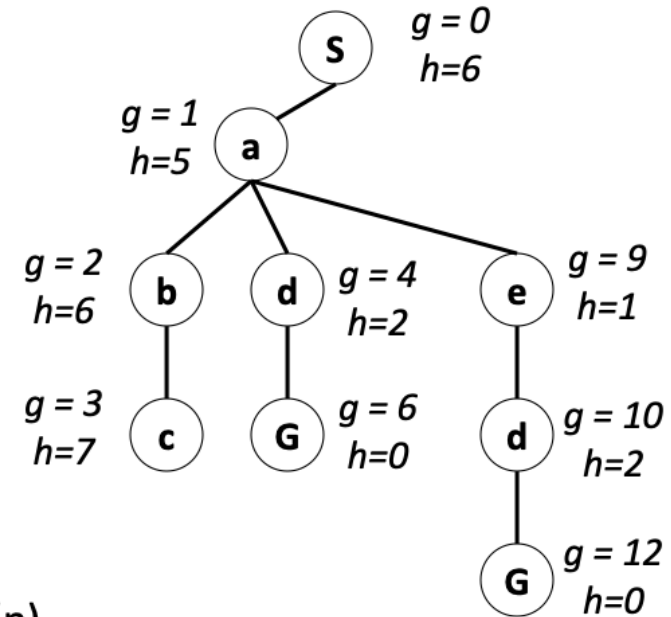
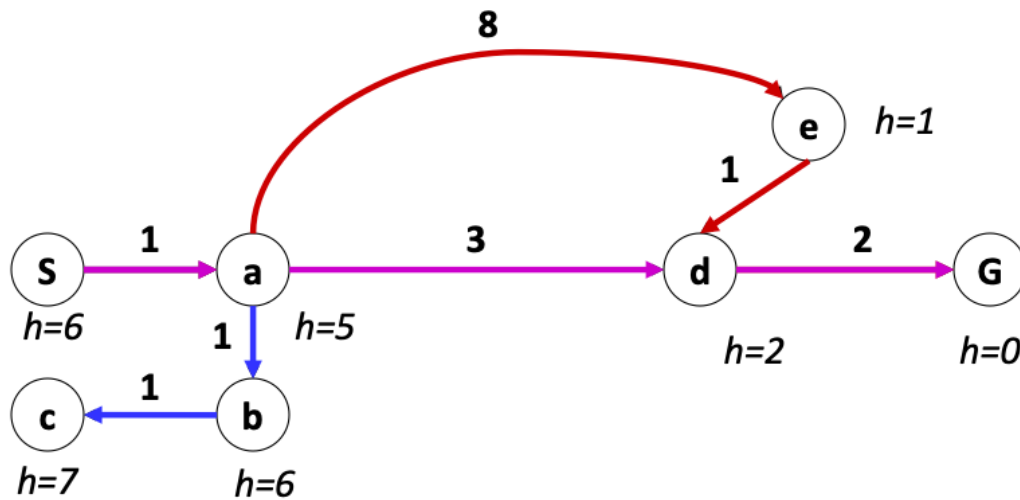
- $h(n)$ = Manhattan distance = $|\Delta x| + |\Delta y|$
- Is Manhattan better than straight-line distance?



A* combines UCS and greedy

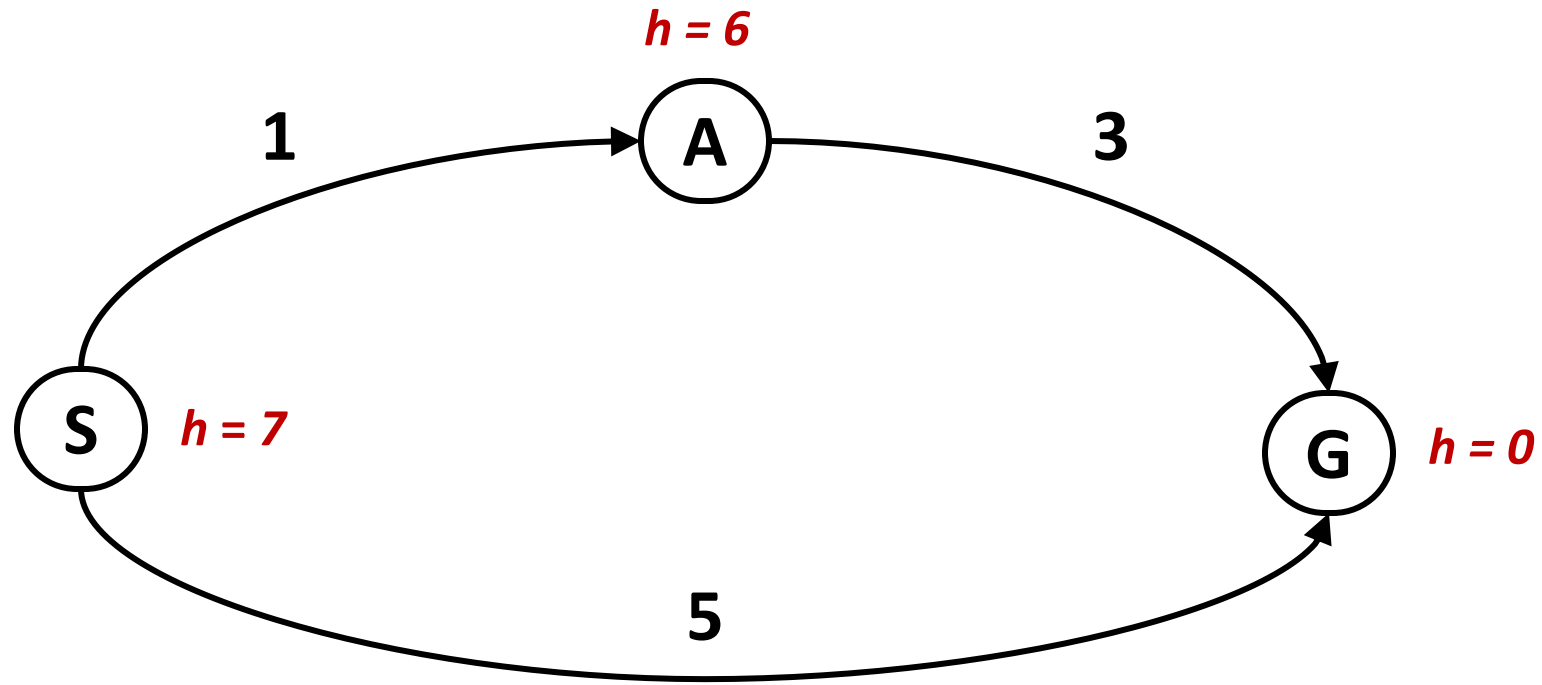
Uniform Cost orders by path cost, or *backward cost* $g(n)$

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



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

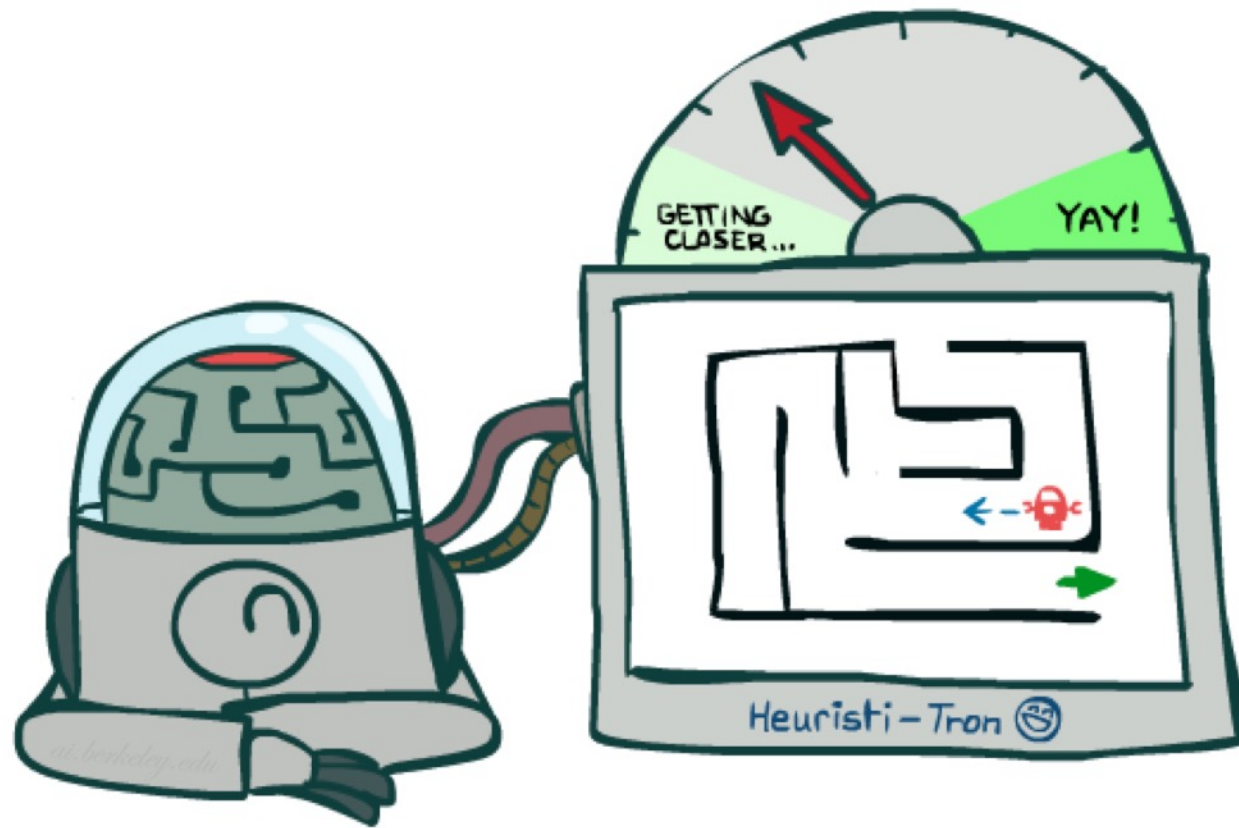
Is A* Optimal?



What went wrong?

- **Actual** bad solution cost < **estimated** good solution cost
- We need estimates to be less than actual costs!

Admissible Heuristics



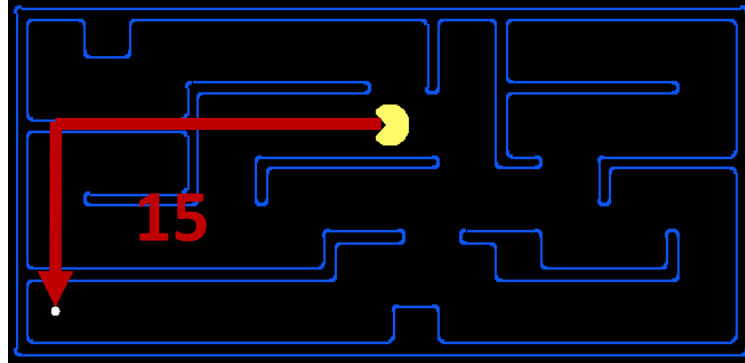
Admissible Heuristics

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

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

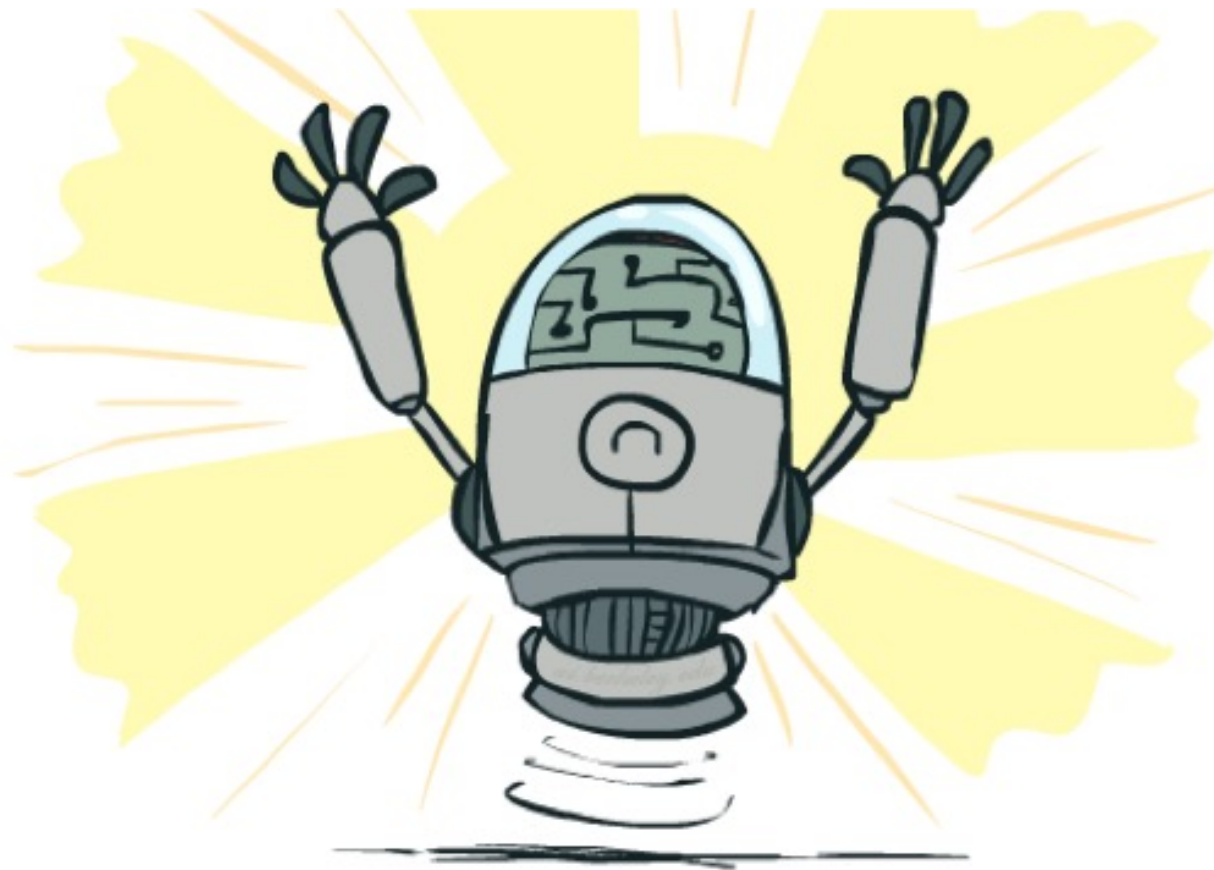
where $h^*(n)$ is the true cost to a nearest goal

- Example:



- Finding good, cheap admissible heuristics is the key to success

Optimality of A* Tree Search



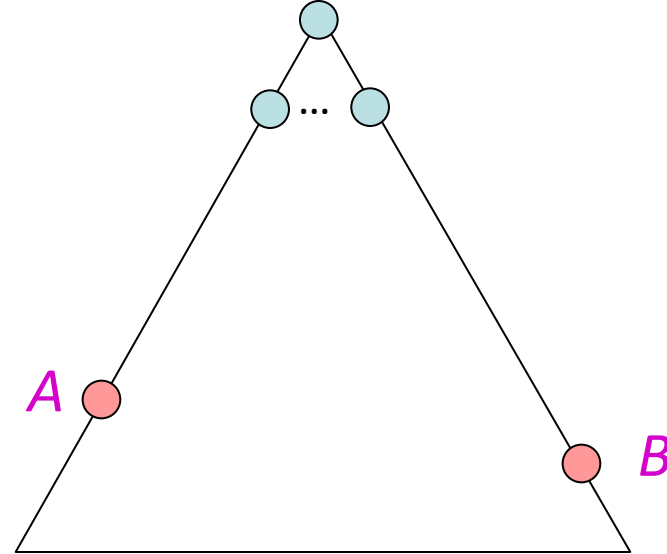
Optimality of A* Tree Search

Assume:

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

Claim:

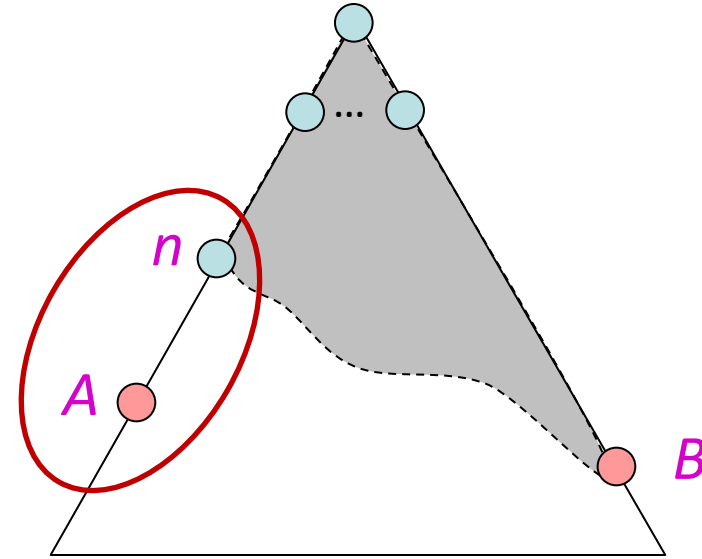
- A will be chosen for expansion before B



Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the frontier
- Some ancestor n of A is on the frontier, too (maybe A itself!)
- Claim: n will be expanded before B
 1. $f(n) \leq f(A)$



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

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

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

Definition of f -cost

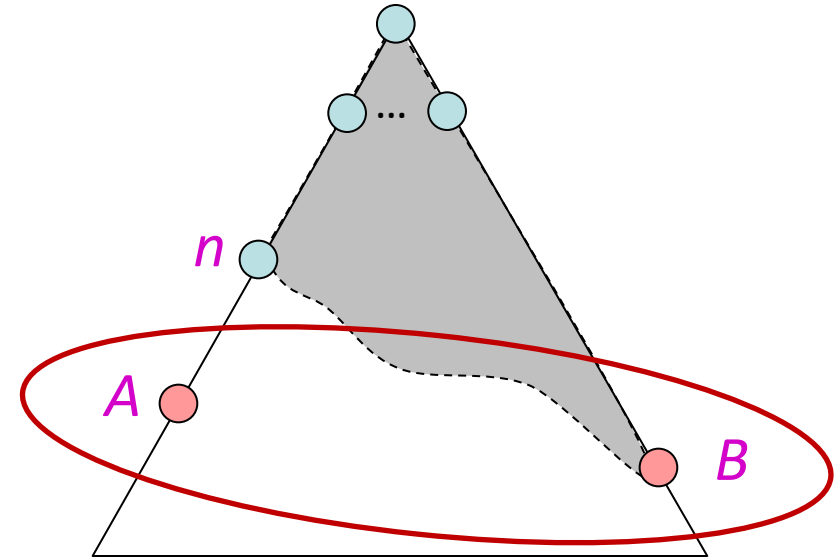
Admissibility of h

$h = 0$ 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 A itself!)
- Claim: n will be expanded before B
 1. $f(n) \leq f(A)$
 2. $f(A) < f(B)$



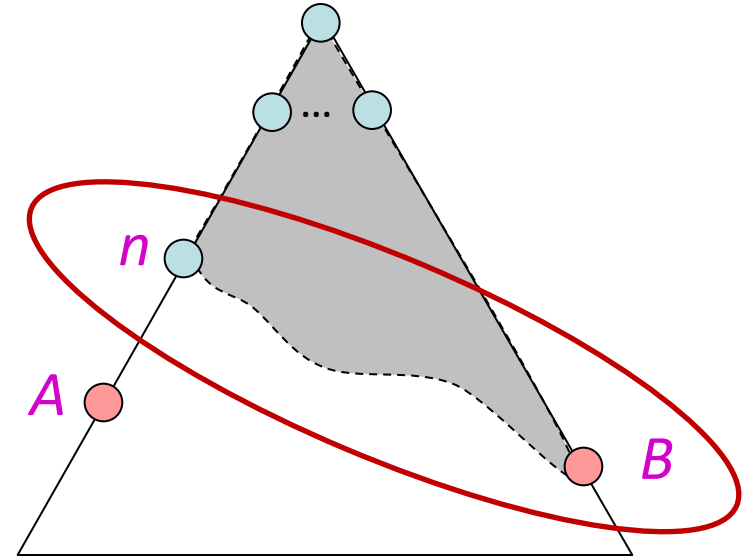
$$g(A) < g(B)$$
$$f(A) < f(B)$$

Suboptimality of B
 $h = 0$ at a goal

Optimality of A* Tree Search: Blocking

Proof:

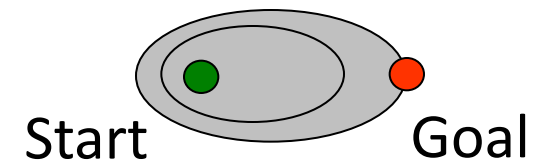
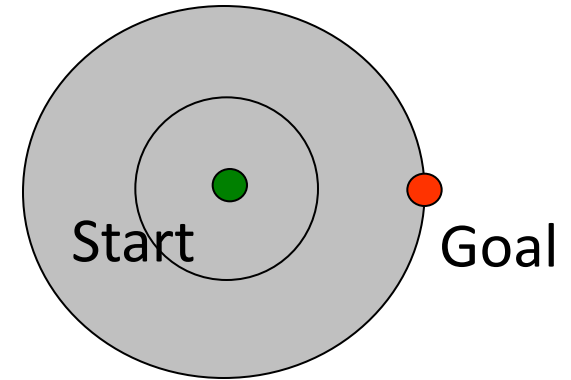
- Imagine B is on the frontier
- Some ancestor n of A is on the frontier, too (maybe A itself!)
- Claim: n will be expanded before B
 1. $f(n) \leq f(A)$
 2. $f(A) < f(B)$
 3. n is expanded before B
- All ancestors of A are expanded before B
- A is expanded before B
- **A* tree search is optimal**



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

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 (h)



Uniform Cost (g)



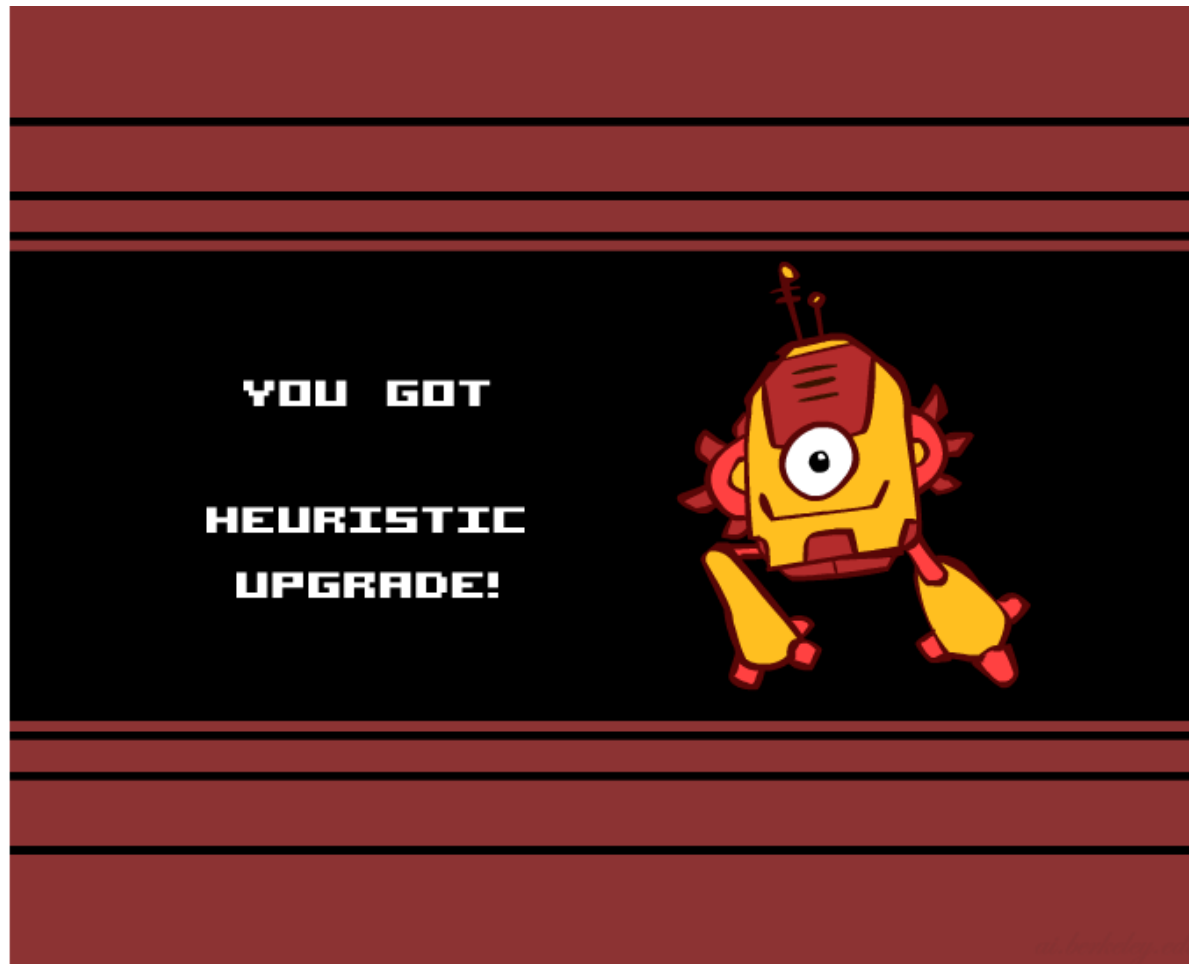
A^* (g+h)

A* Applications

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- Protein design
- Chemical synthesis
- ...

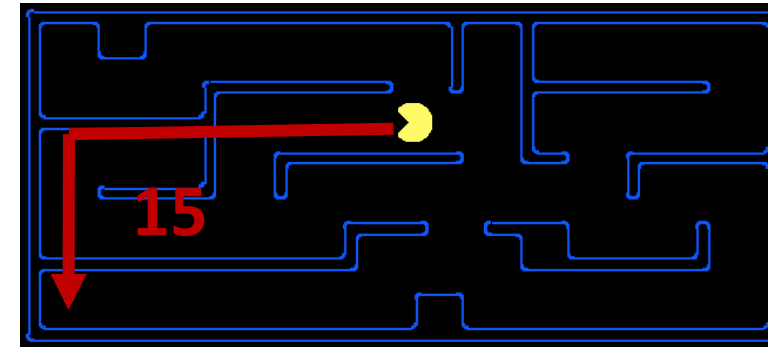
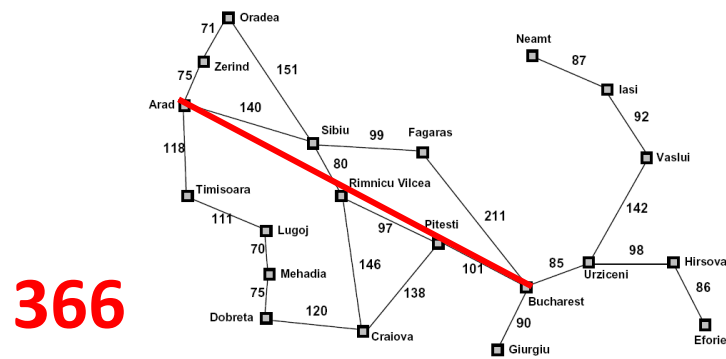


Creating Heuristics



Creating Admissible Heuristics

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

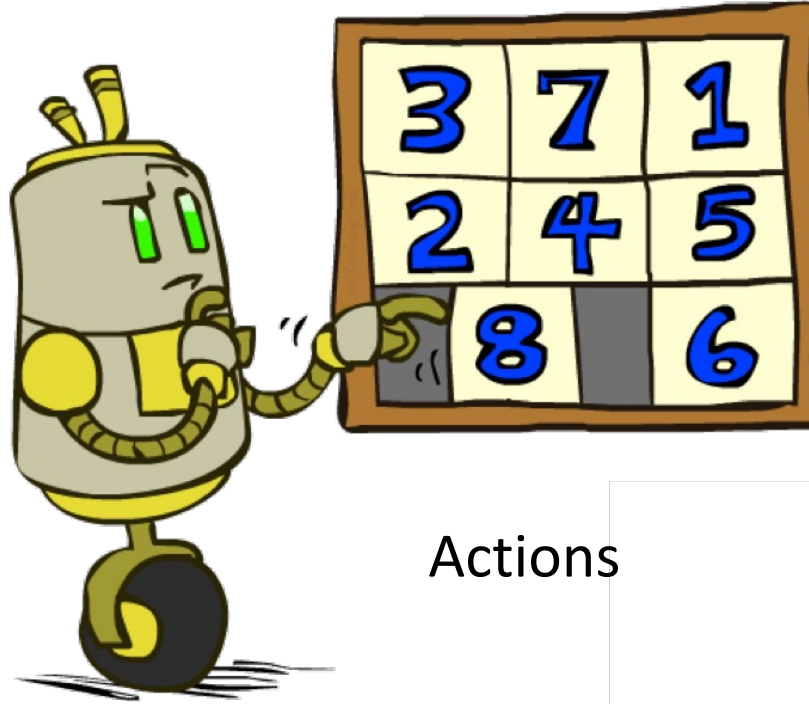


- Problem P_2 is a relaxed version of P_1 if $\mathcal{A}_2(s) \supseteq \mathcal{A}_1(s)$ for every s
- Theorem: $h_2^*(s) \leq h_1^*(s)$ for every s , so $h_2^*(s)$ is admissible for P_1

Example: 8 Puzzle

7	2	4
5		6
8	3	1

Start State



Actions

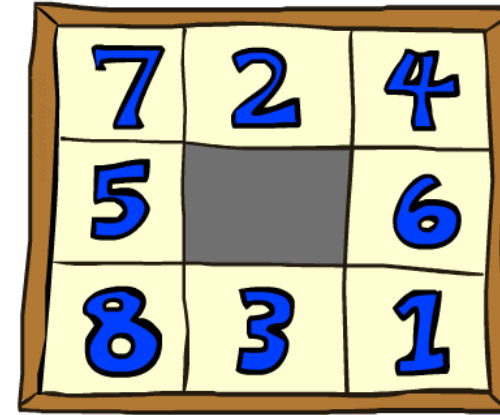
	1	2
3	4	5
6	7	8

Goal State

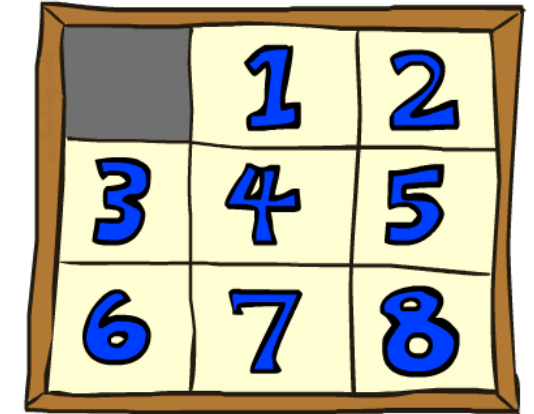
- What are the states?
- How many states?
- What are the actions?
- What are the step costs?

8 Puzzle I

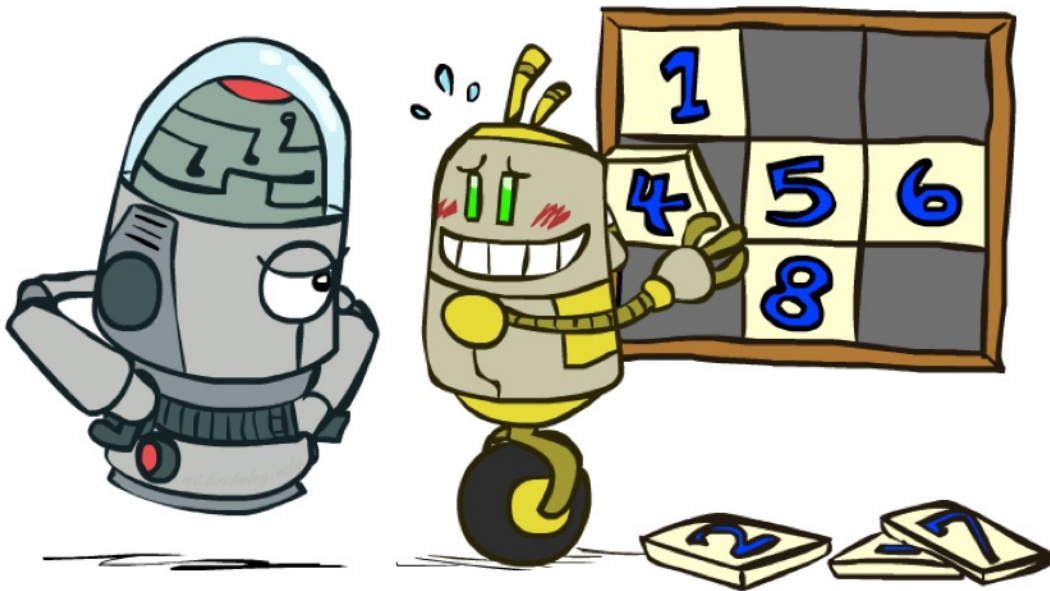
- Heuristic: Number of tiles misplaced
- Why is it admissible?
- $h(\text{start}) = 8$



Start State



Goal State

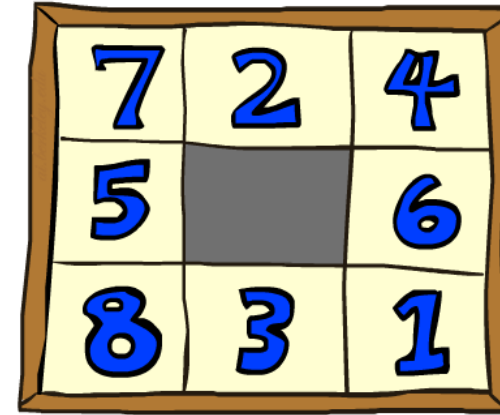


Average nodes expanded when the optimal path has...

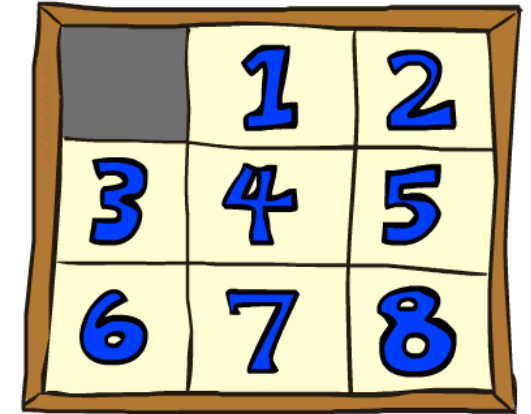
	...4 steps	...8 steps	...12 steps
UCS	112	6,300	3.6×10^6
A*TILES	13	39	227

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



Start State



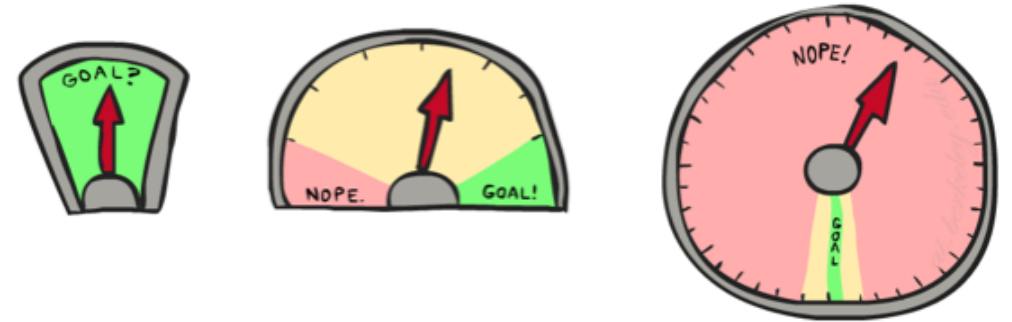
Goal State

Average nodes expanded when the optimal path has...			
	...4 steps	...8 steps	...12 steps
A* _{TILES}	13	39	227
A* _{MANHATTAN}	12 ₂₉	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

Combining heuristics

- Dominance: $h_1 \geq h_2$ if

$$\forall n \ h_1(n) \geq h_2(n)$$

- 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?

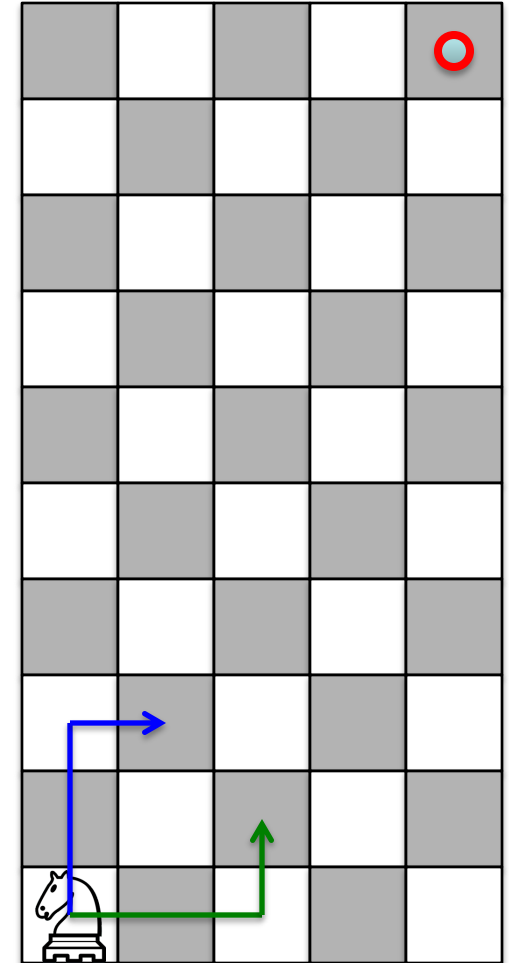
- Form a new heuristic by taking the max of both:

$$h(n) = \max(h_1(n), h_2(n))$$

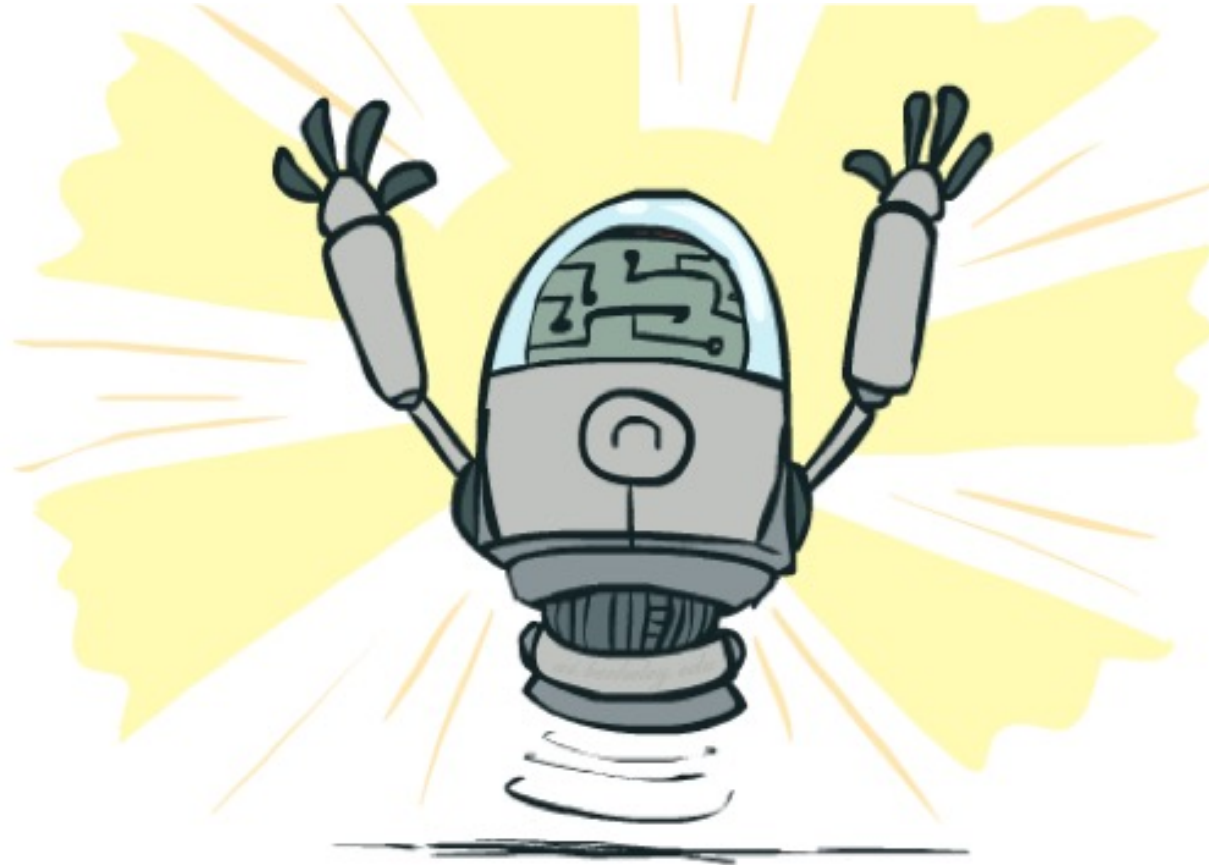
- Max of admissible heuristics is admissible and dominates both!

Example: Knight's moves

- Minimum number of knight's moves to get from A to B?
 - $h_1 = (\text{Manhattan distance})/3$
 - $h_1' = h_1$ rounded up to correct parity (even if A, B same color, odd otherwise)
 - $h_2 = (\text{Euclidean distance})/\sqrt{5}$ (rounded up to correct parity)
 - $h_3 = (\max \text{ x or y shift})/2$ (rounded up to correct parity)
- $h(n) = \max(h_1'(n), h_2(n), h_3(n))$ is admissible!



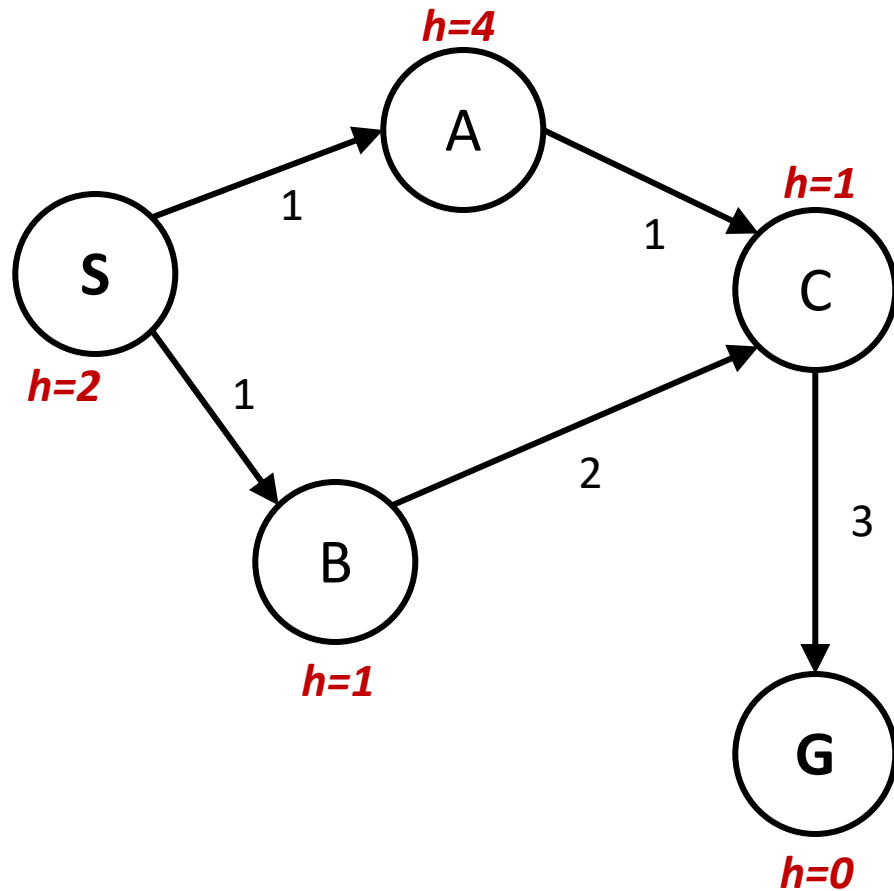
Optimality of A* Graph Search



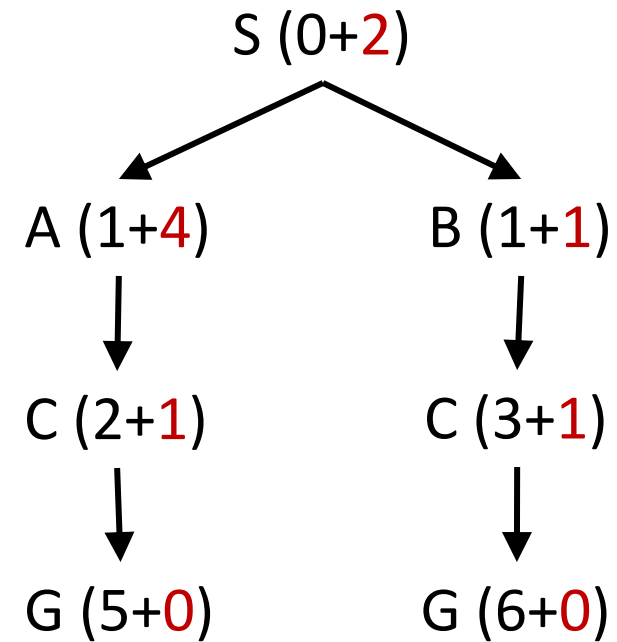
This part is a bit fiddly,
sorry about that

A* Graph Search Gone Wrong?

State space graph

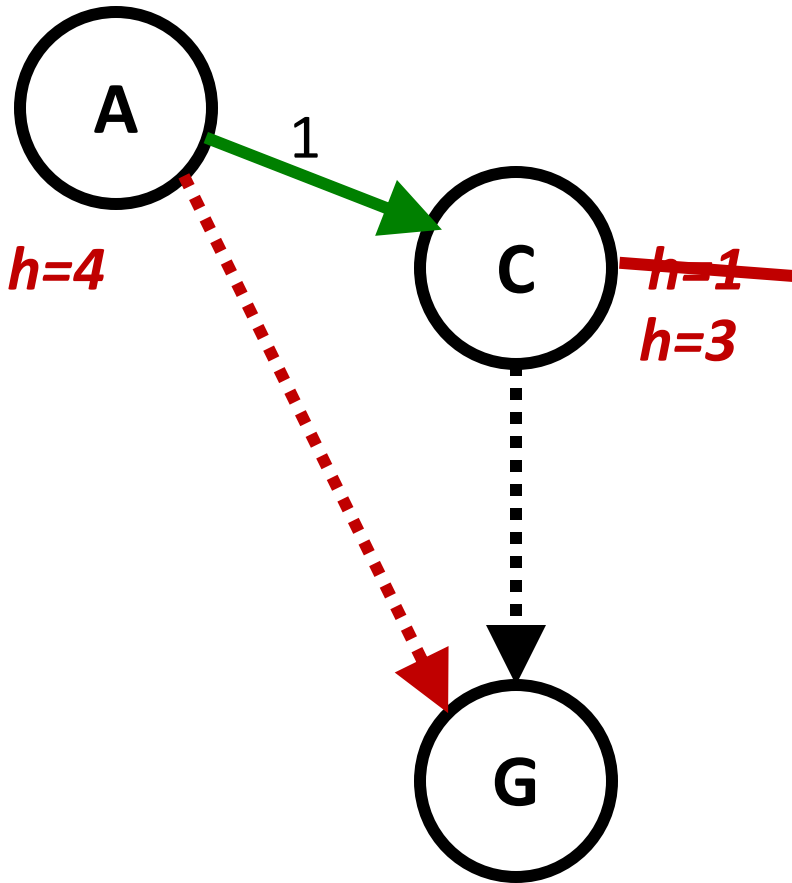


Search tree



Simple check against expanded set blocks C
Fancy check allows new C if cheaper than old
but requires recalculating C's descendants

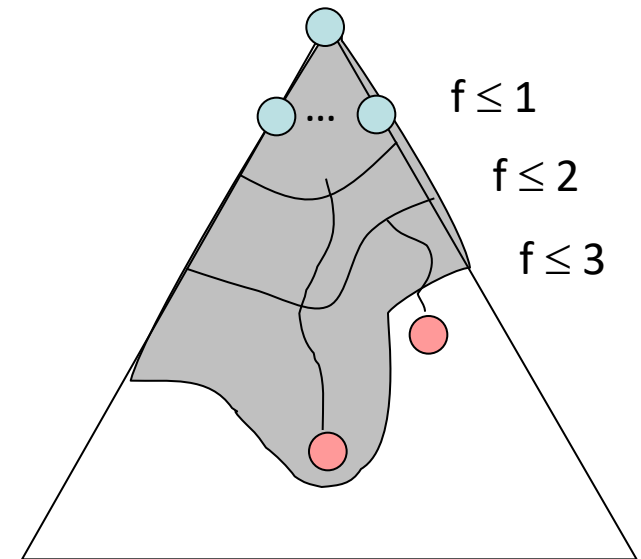
Consistency of Heuristics



- Main idea: estimated heuristic costs \leq actual costs
 - Admissibility: heuristic cost \leq actual cost to goal
$$h(A) \leq h^*(A)$$
 - Consistency: heuristic “arc” cost \leq actual cost for each arc
$$h(A) - h(C) \leq c(A,C)$$
or $h(A) \leq c(A,C) + h(C)$ (triangle inequality)
 - Note: h^* necessarily satisfies triangle inequality
- Consequences of consistency:
 - The f value along a path never decreases:
$$h(A) \leq c(A,C) + h(C) \Rightarrow g(A) + h(A) \leq g(A) + c(A,C) + h(C)$$
 - A* graph search is optimal

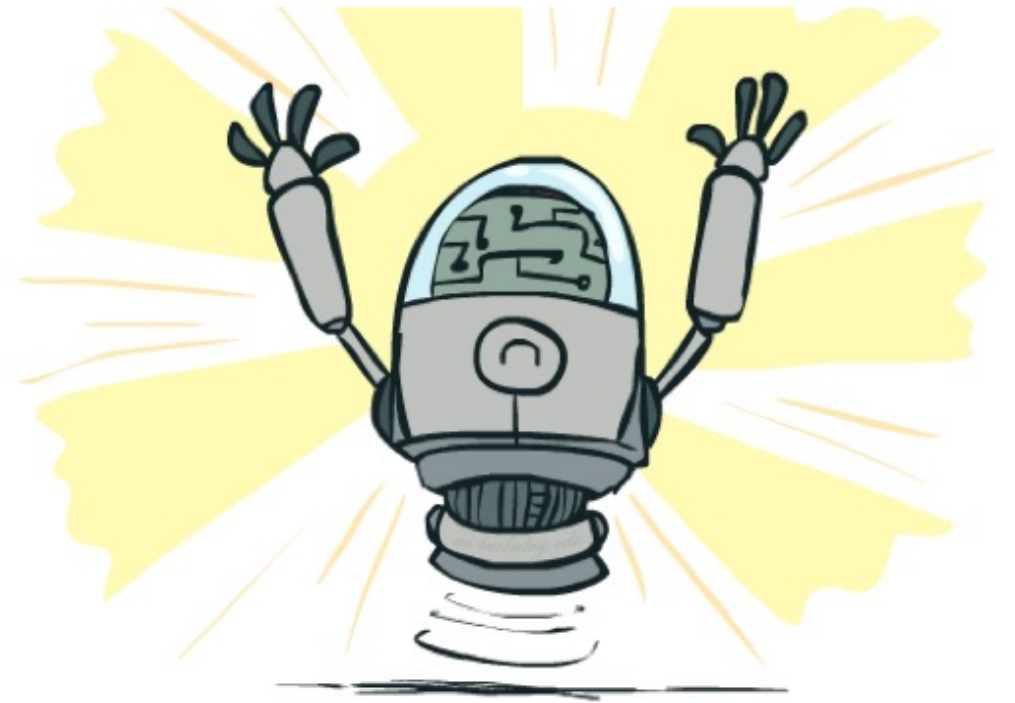
Optimality of A* Graph Search

- 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
- Graph search:
 - A^* optimal if heuristic is consistent
- Consistency implies admissibility
- Most natural admissible heuristics tend to be consistent, especially if from relaxed problems



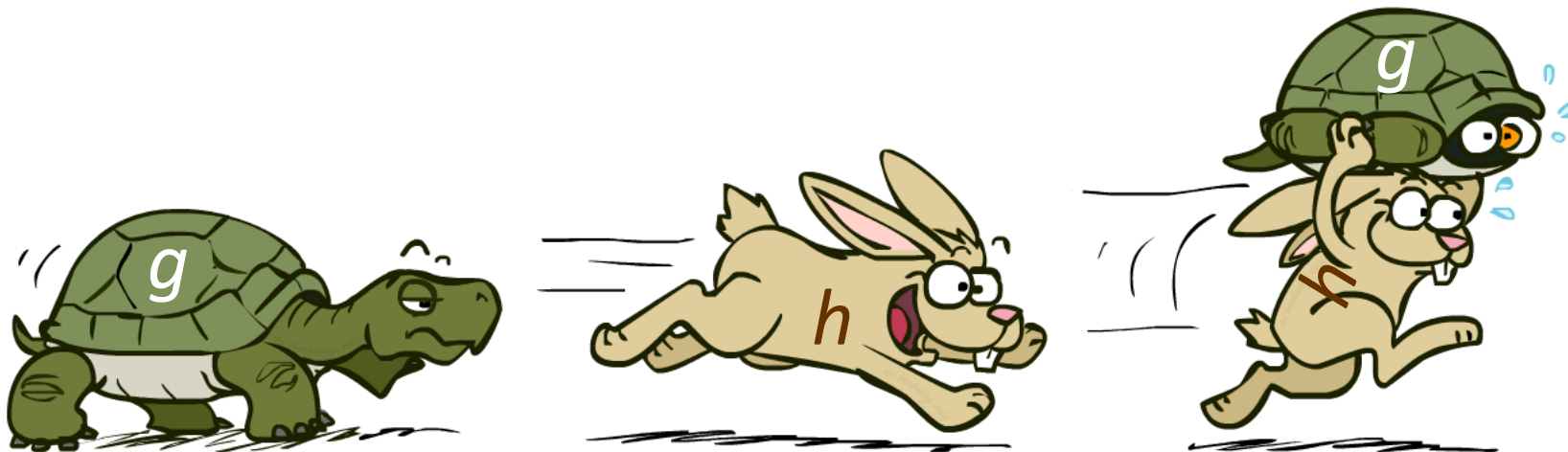
But...

- A* keeps the entire explored region in memory
- => will run out of space before you get bored waiting for the answer
- There are variants that use less memory (Section 3.5.5):
 - IDA* works like iterative deepening, except it uses an f -limit instead of a depth limit
 - On each iteration, remember the smallest f -value that exceeds the current limit, use as new limit
 - Very inefficient when f is real-valued and each node has a unique value
 - RBFS is a recursive depth-first search that uses an f -limit = the f -value of the best alternative path available from any ancestor of the current node
 - When the limit is exceeded, the recursion unwinds but remembers the best reachable f -value on that branch
 - SMA* uses *all available memory* for the queue, minimizing thrashing
 - When full, drop worst node on the queue but remember its value in the parent



A*: Summary

- A* orders nodes in the queue by $f(n) = g(n) + h(n)$
- A* is optimal for trees/graphs 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
```