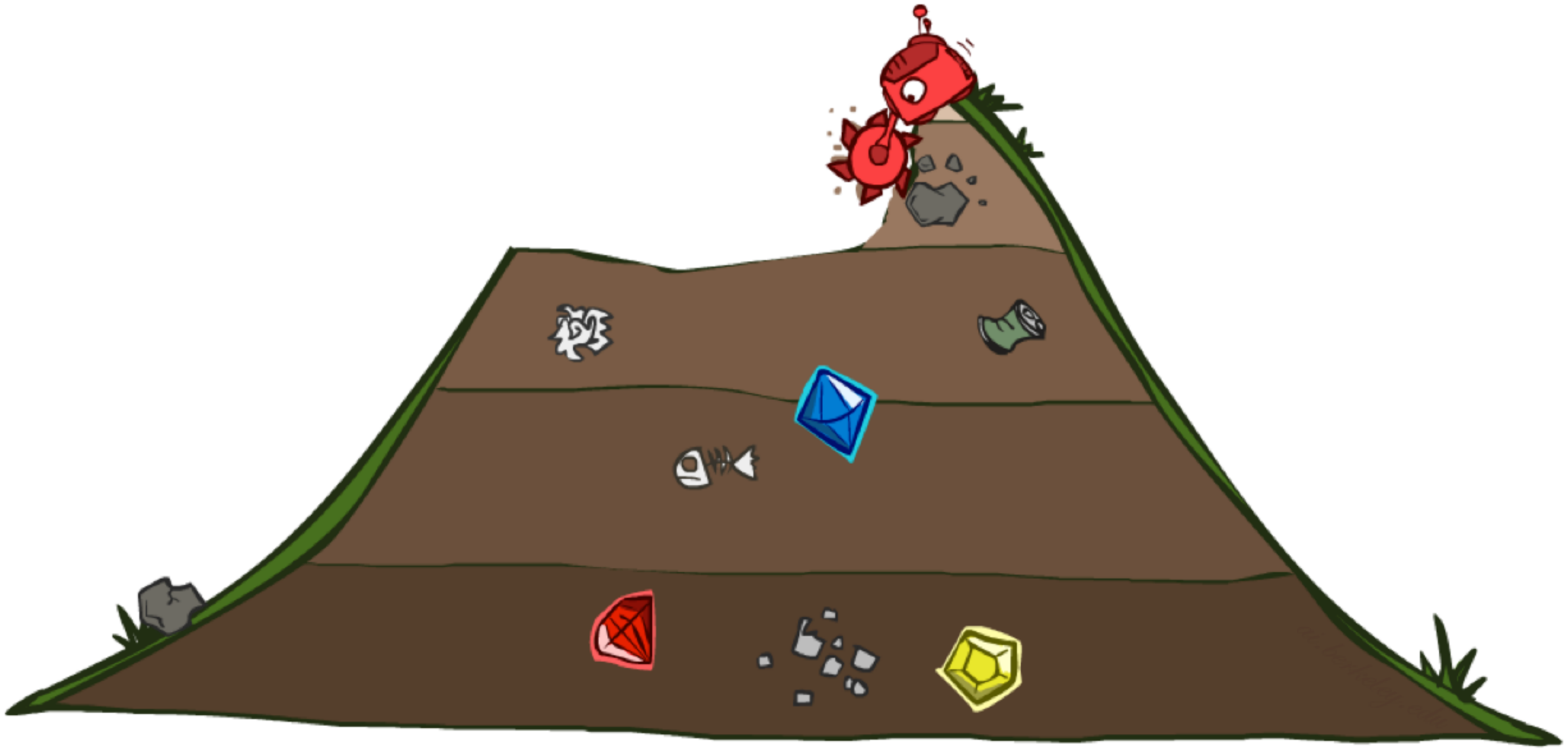
Uniform Cost Search

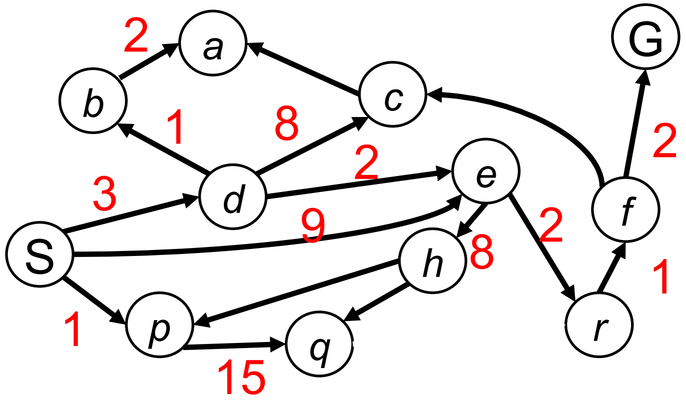


Uniform Cost Search

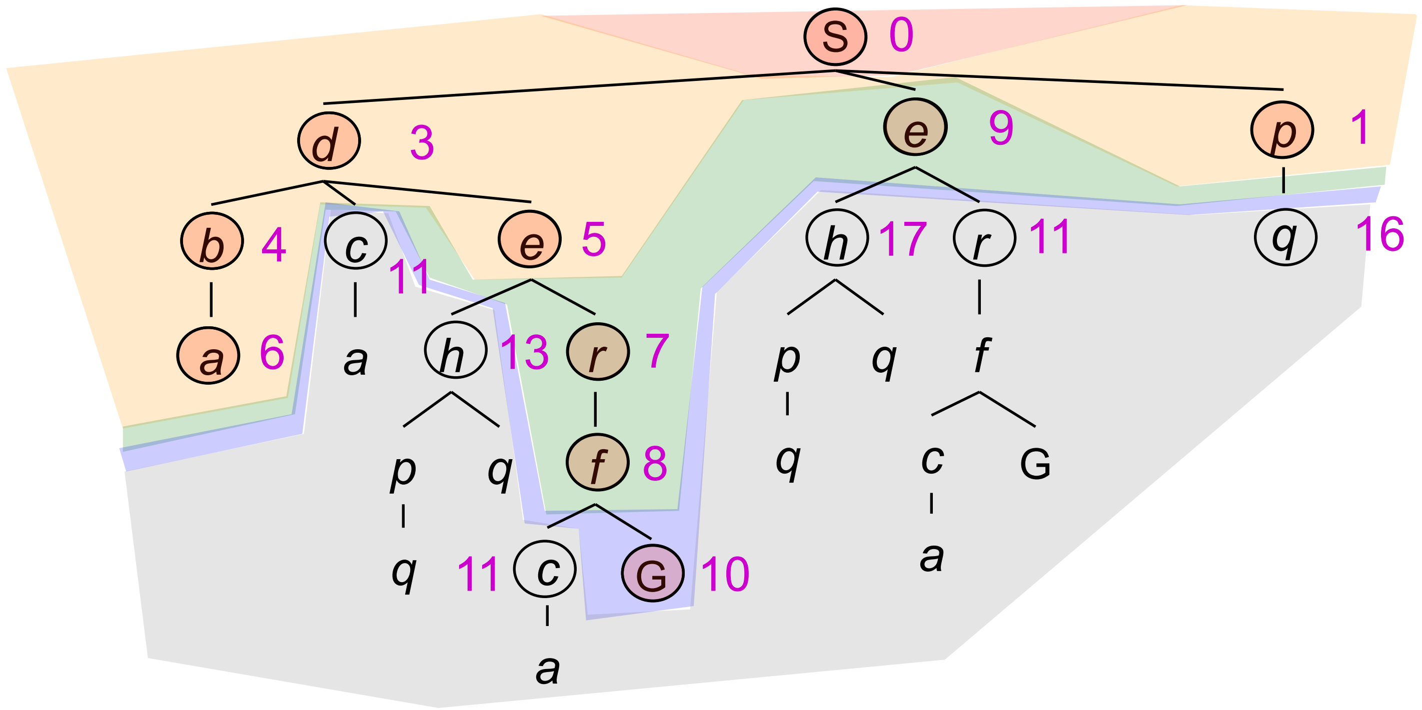
g(n) = cost from root to n

Strategy: expand lowest g(n)

Frontier is a priority queue sorted by g(n)

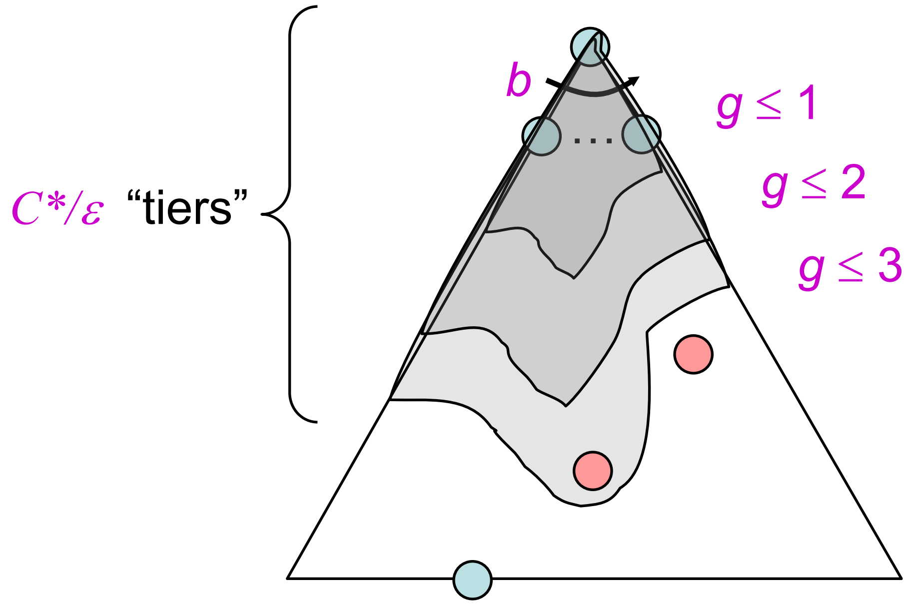


Cost contours



Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Expands all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)
- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$
- Is it complete?
  - Assuming C^* is finite and $\varepsilon > 0$, yes!
- Is it optimal?
  - Yes! (Proof next lecture via A*)



Summary

- Assume known, discrete, observable, deterministic, atomic
- Search problems defined by $\mathcal{S}$, s_0, $\mathcal{A}(s)$, $Result(s,a)$, $G(s)$, $c(s,a,s')$
- Search algorithms find action sequences that reach goal states
  - Optimal => minimum-cost
- Search algorithm properties:
  - Depth-first: incomplete, suboptimal, space-efficient
  - Breadth-first: complete, (sub)optimal, space-prohibitive
  - Iterative deepening: complete, (sub)optimal, space-efficient
  - Uniform-cost: complete, optimal, space-prohibitive

Bonus Search Algo Summary

Search	Frontier	Completeness	Optimality	Time	Space
DFS (Depth-First)	stack	tree search - no (cycle) graph search < yes (finite), no (infinite)	no	$O(b^m)$	$O(bm)$
BFS (Breadth-First)	queue	yes	no (except when all edge costs same)	$O(b^s)$	$O(b^s)$
Iterative Deepening (BFS result w/ modified DFS algo)	stack (same as DFS)	yes (same as BFS)	no (same as BFS)	$O(b^s)$ (same as BFS)	$O(bs)$ (same as DFS but w/ shortest solution length)
UCS (Uniform Cost)	heap-based PQ (backward cost)	yes (assuming positive edge costs and $\epsilon > 0$)	yes (assuming positive edge costs and $\epsilon > 0$)	$O(b^{C^*/\epsilon})$	$O(b^{C^*/\epsilon})$

b = branching factor (assume finite)

m = max depth of search tree

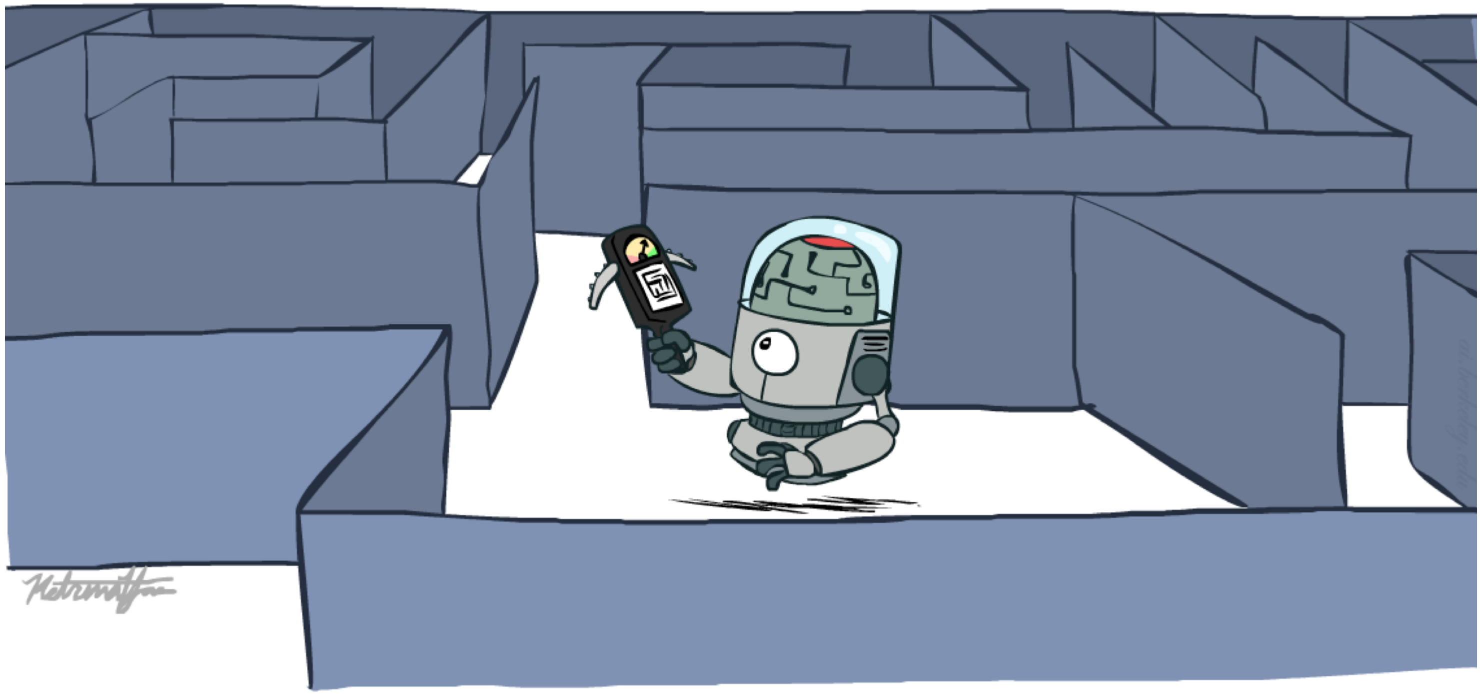
s = smallest depth of solution (assume finite)

C^* = cost of optimal solution (assume finite)

ϵ = minum cost between 2 nodes

CS 188: Artificial Intelligence

Informed Search

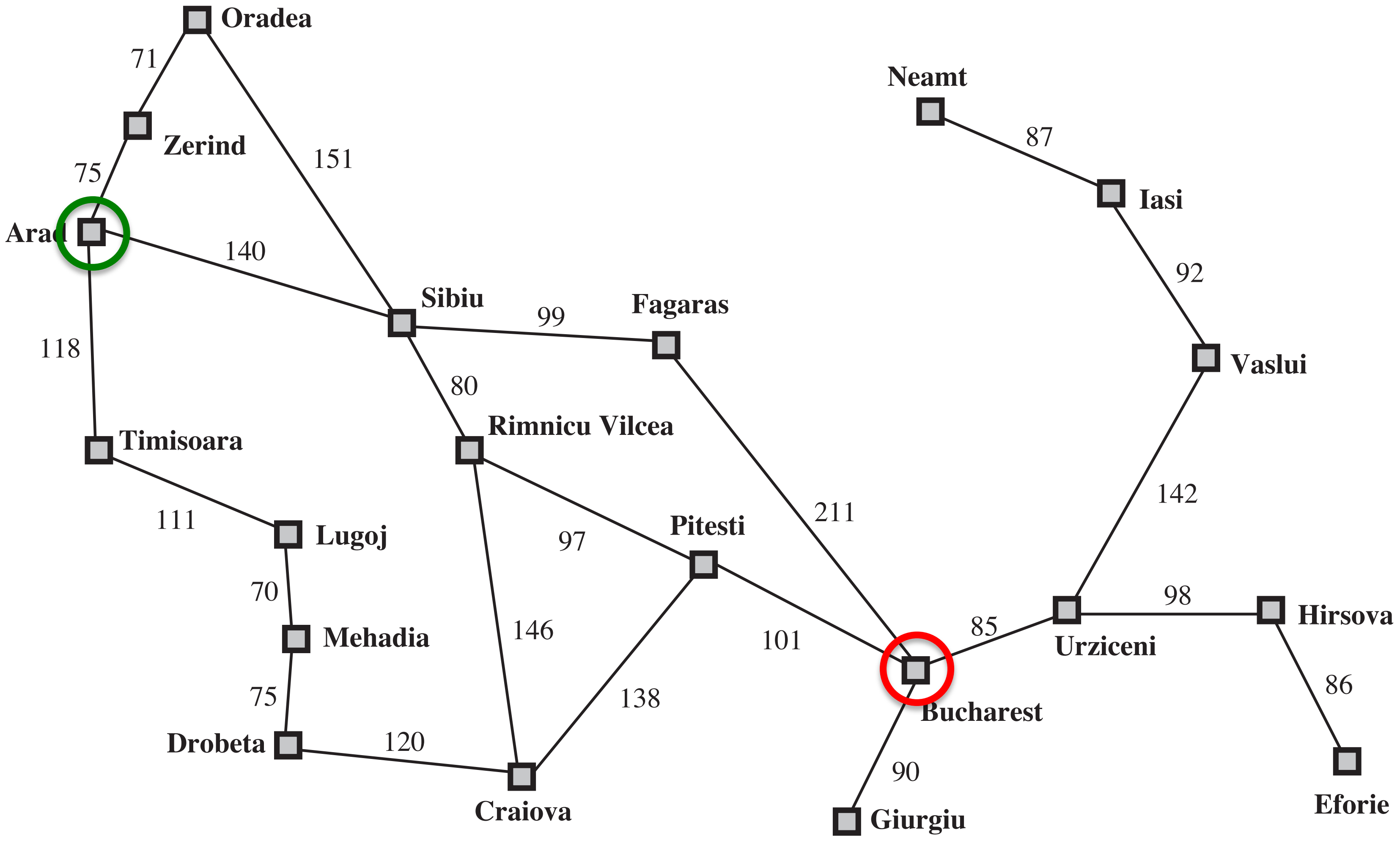


Instructors: Angela Liu and Yanlai Yang

University of California, Berkeley

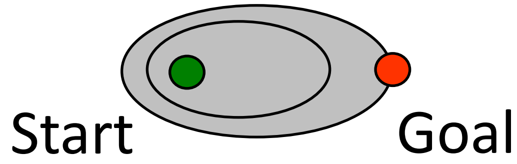
[slides adapted from Stuart Russel, Dawn Song]

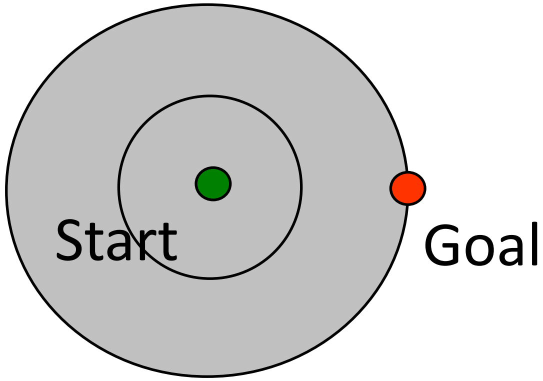
Example: route-finding in Romania



What we would like to have happen

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



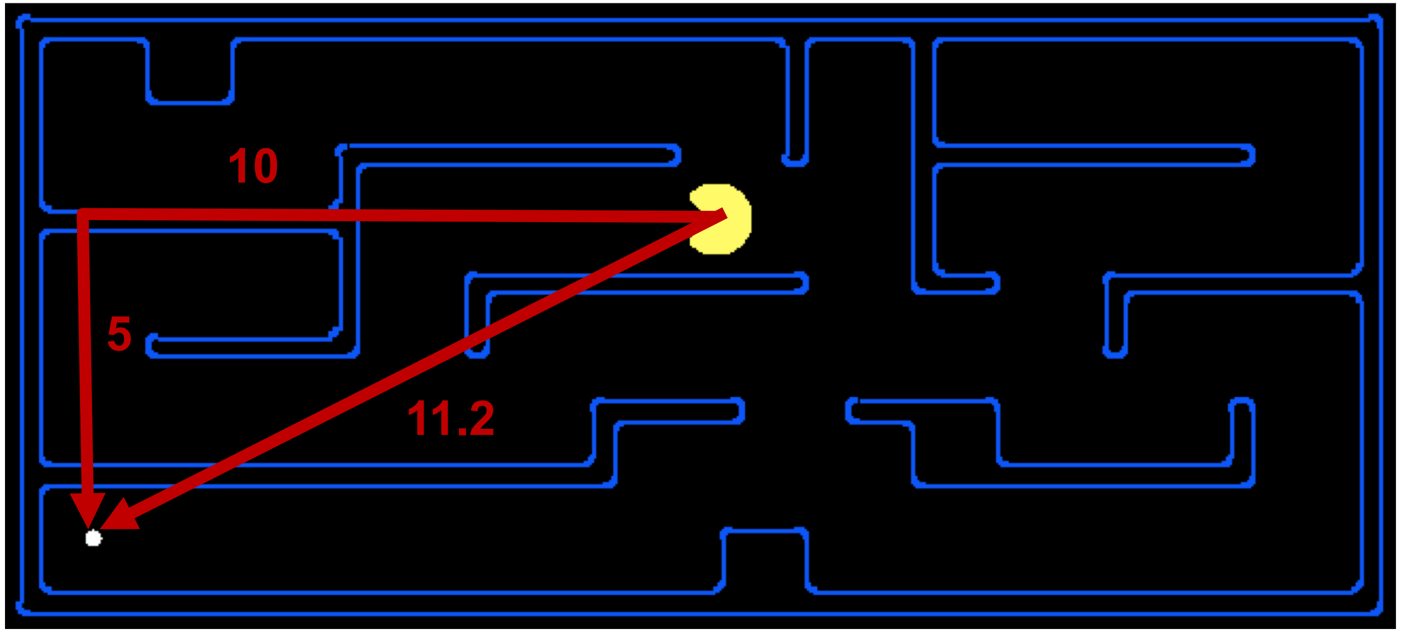


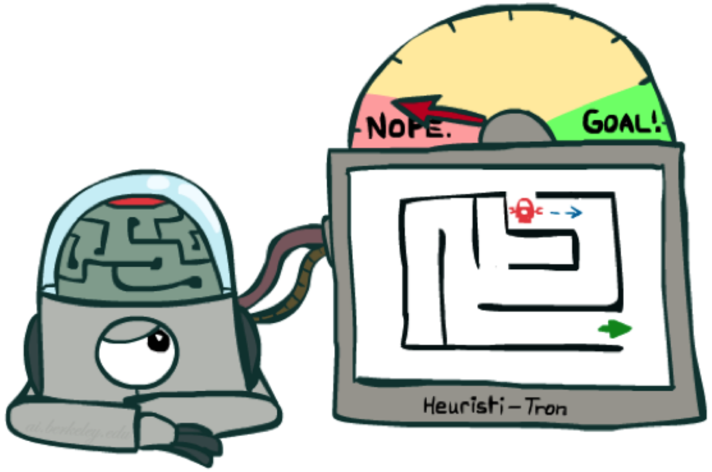
Informed

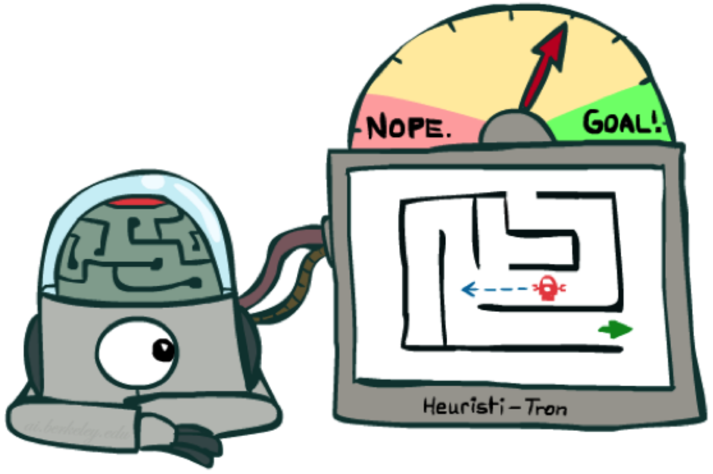
Uninformed

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



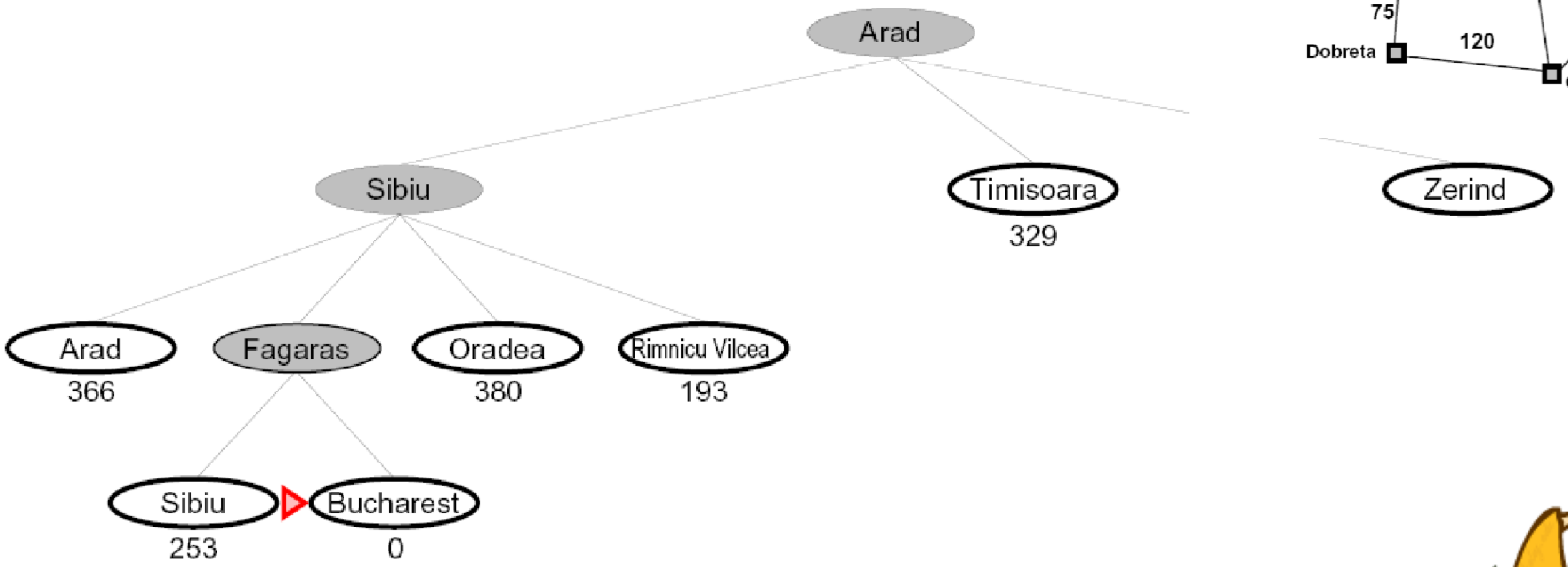


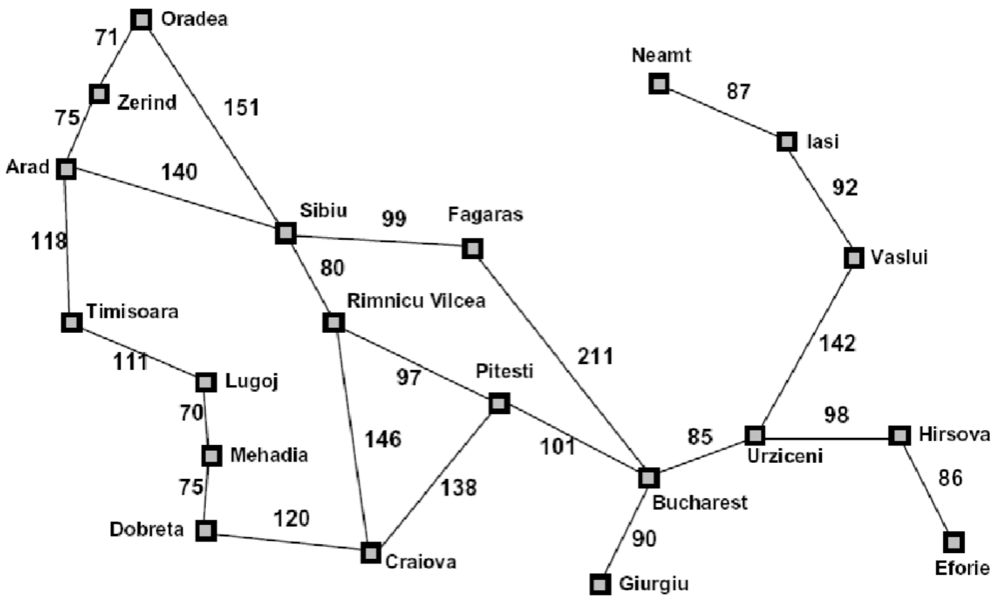


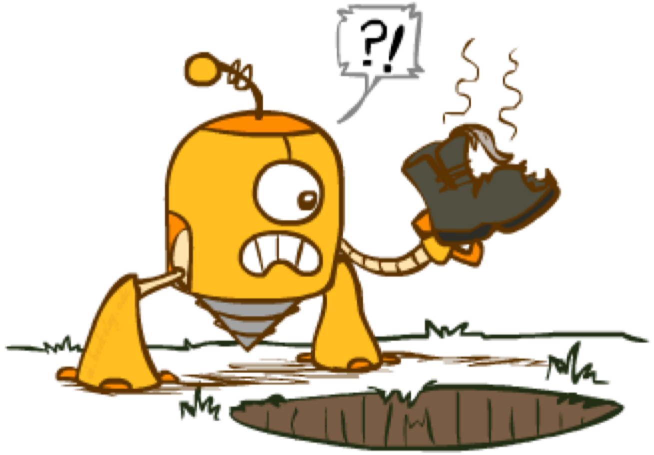
Greedy Search

Greedy Search

- Expand the node that seems closest…



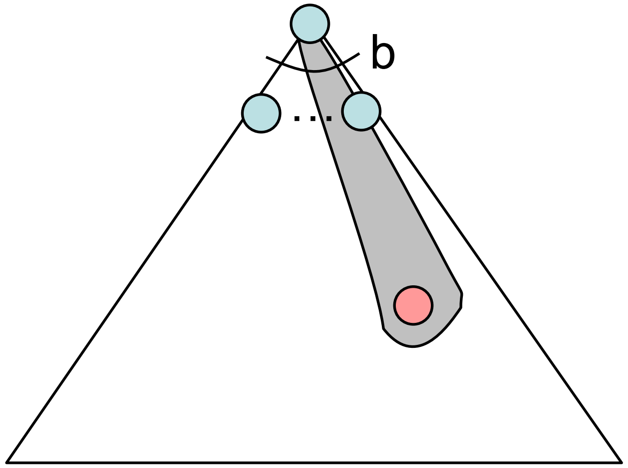




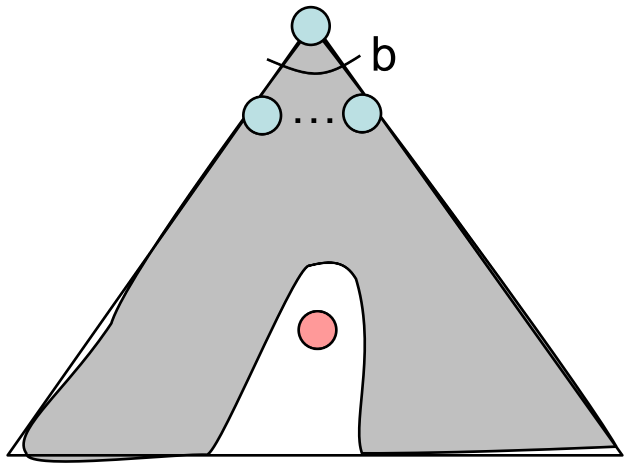
- Is it optimal?
    - No. Resulting path to Bucharest is not the shortest!

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

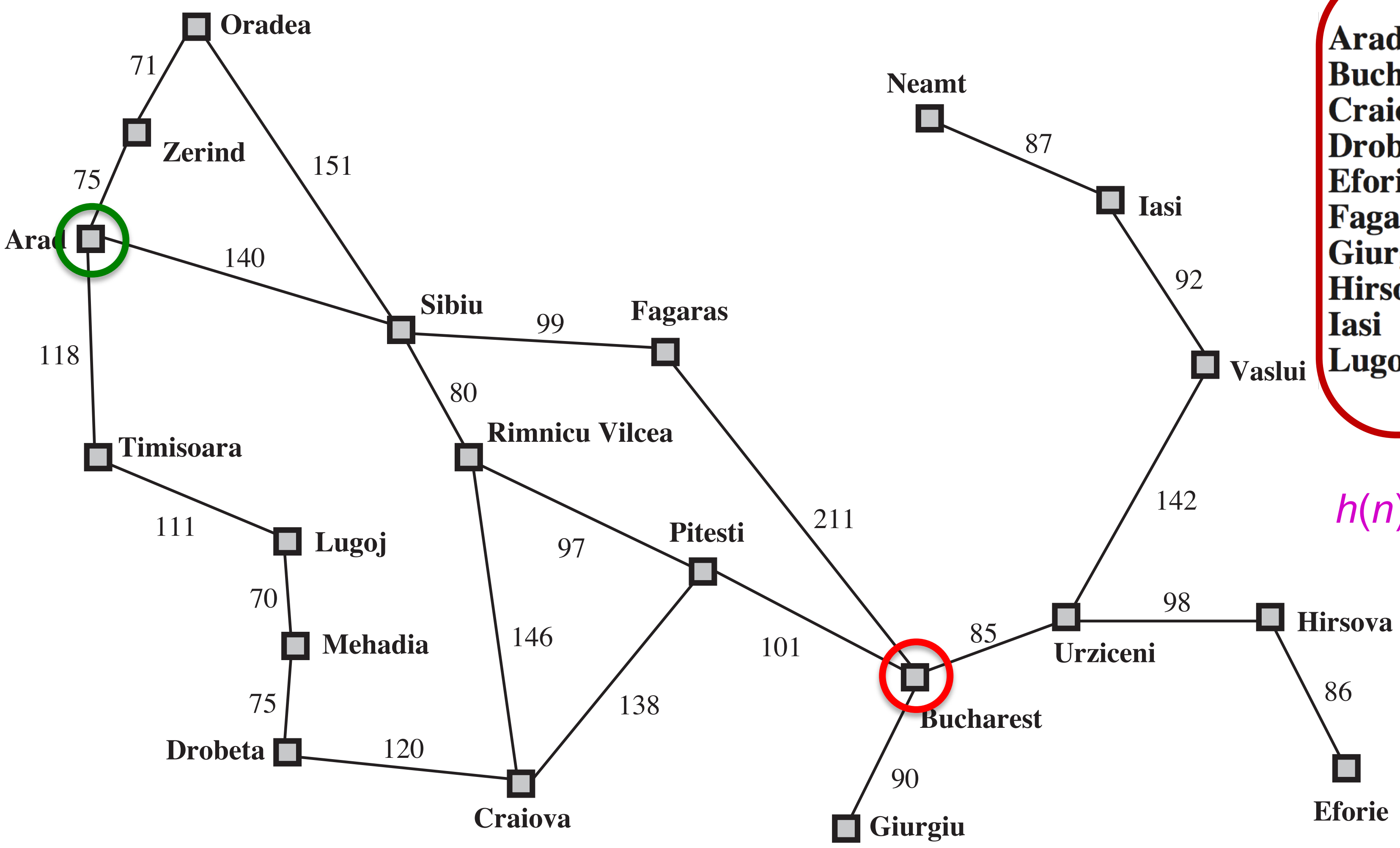
[Demo: contours greedy empty (L3D1)]
[Demo: contours greedy pacman small maze (L3D4)]

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

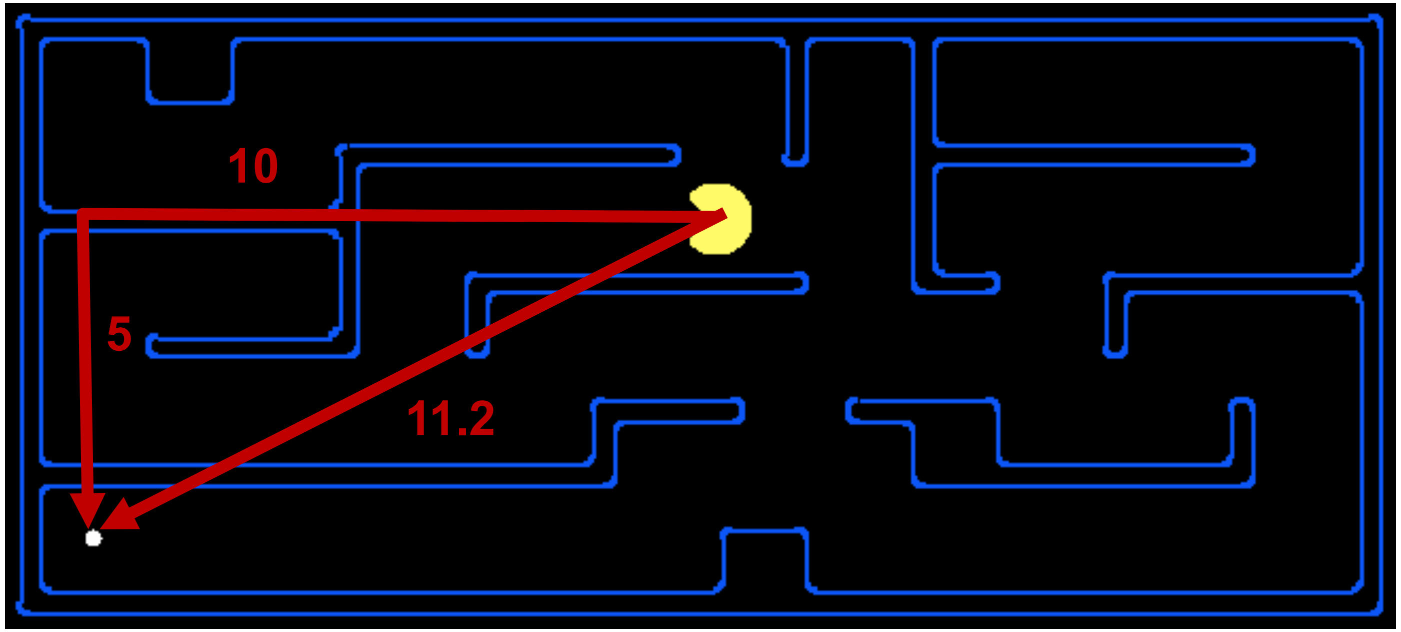


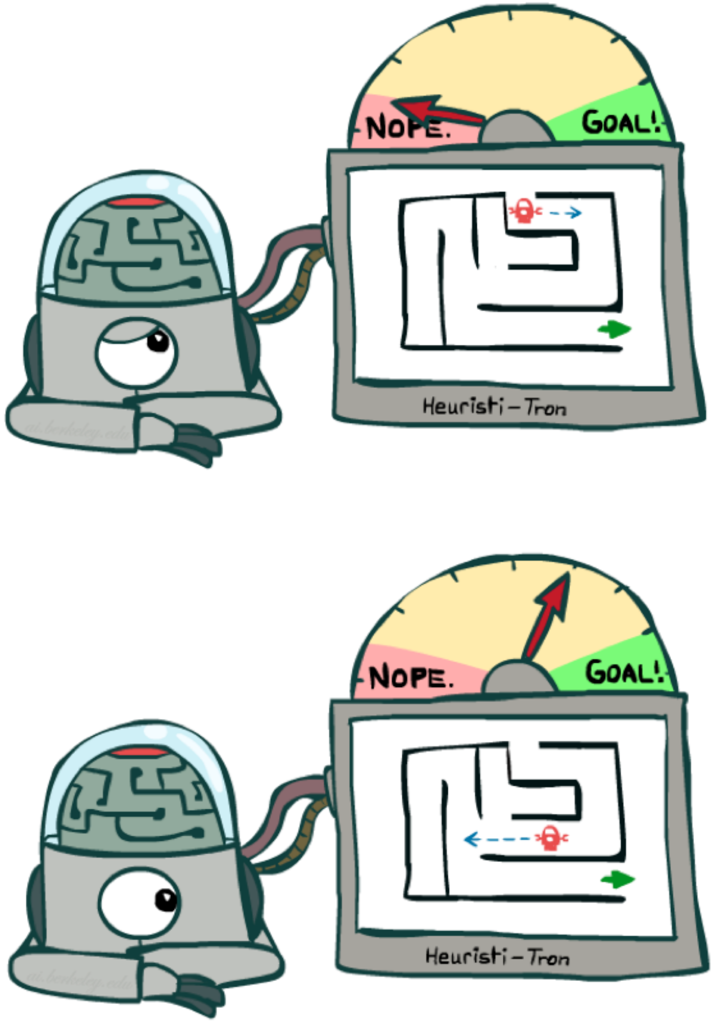
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

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

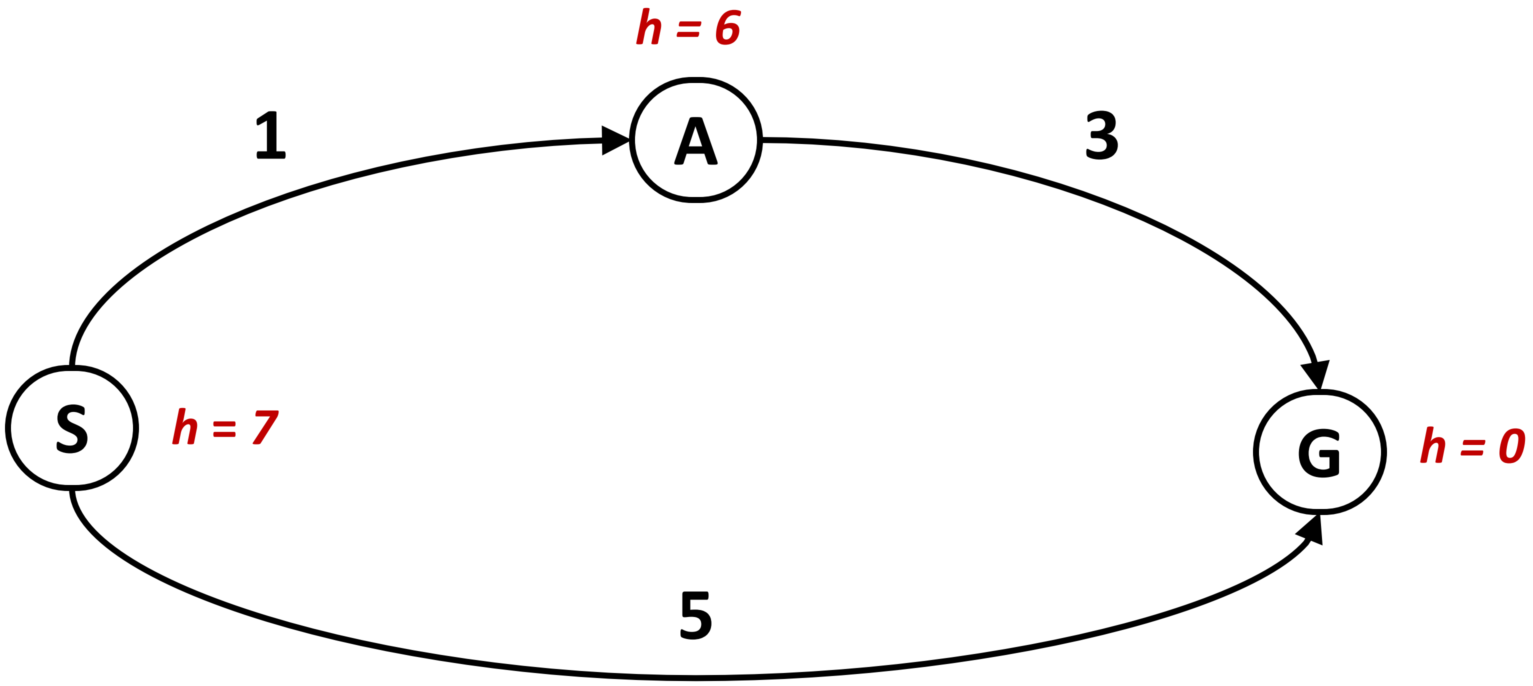
Example: pathing in Pacman

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





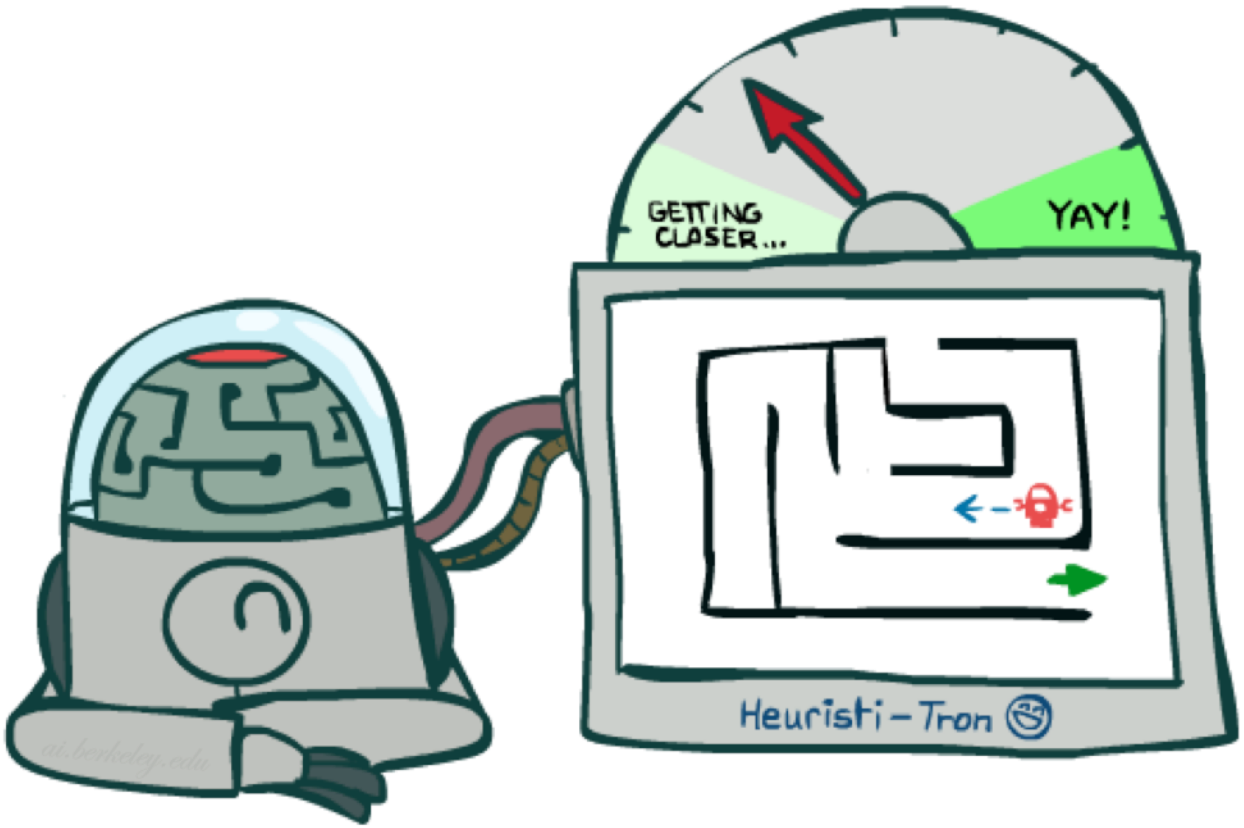
Is A* Optimal?



What went wrong?

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

Admissible Heuristics



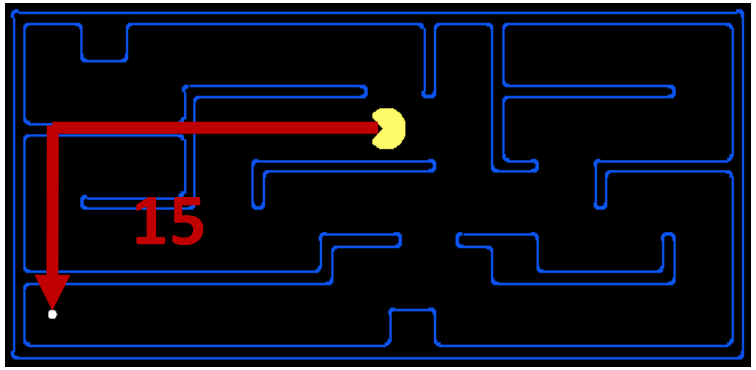
Admissible Heuristics

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

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

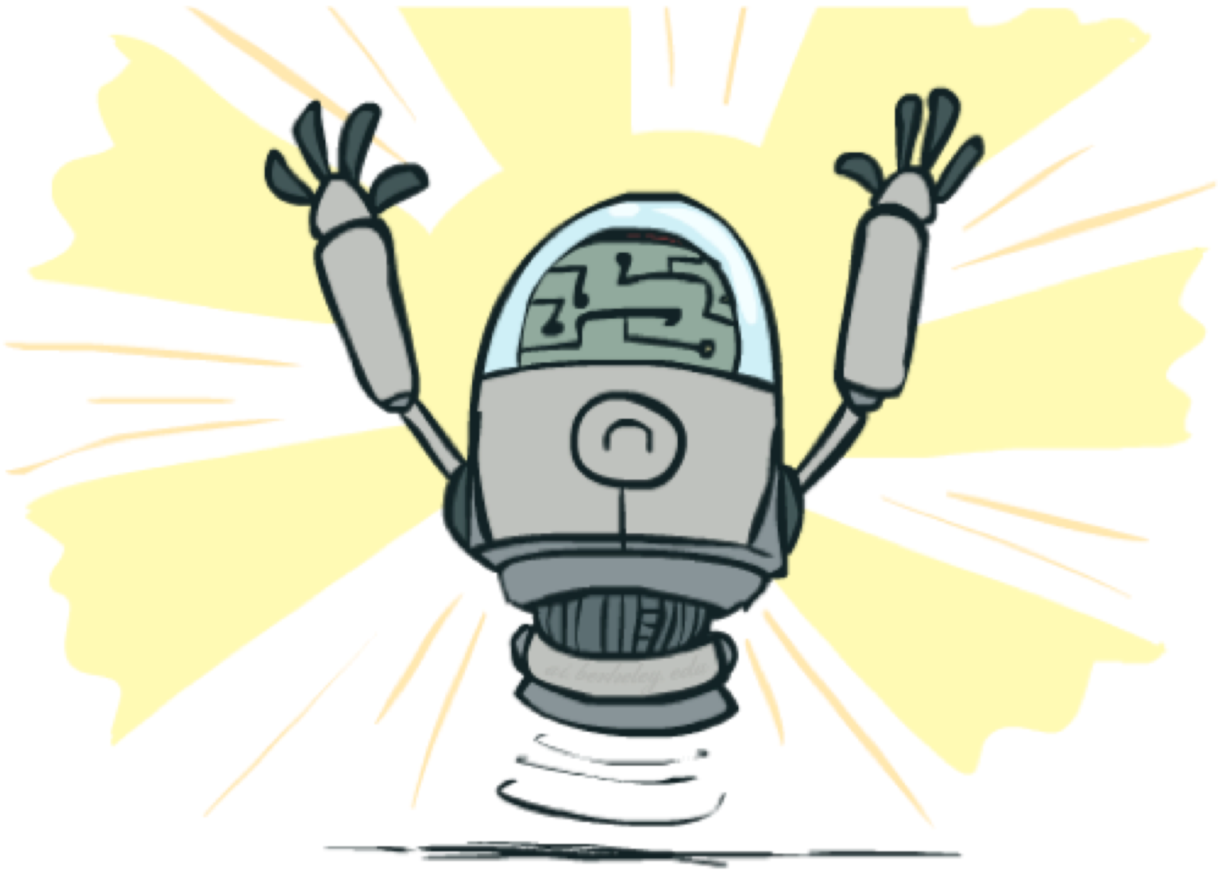
  where $h^*(n)$ is the true cost to a nearest goal

- Example:



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

Optimality of A* Tree Search



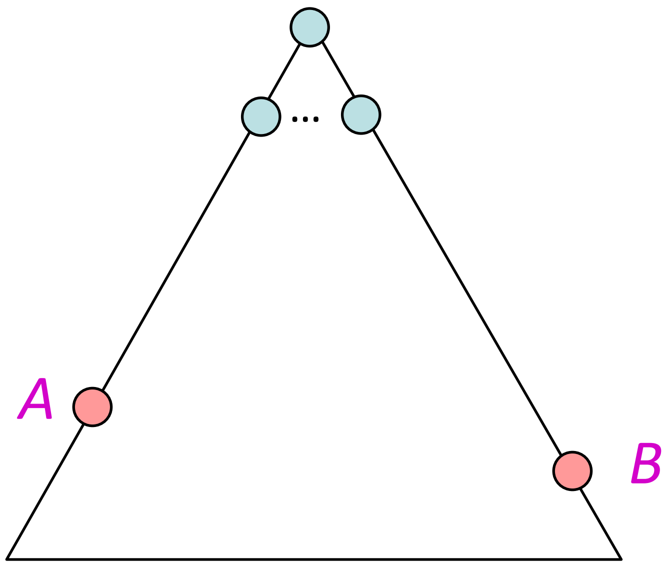
Optimality of A* Tree Search

Assume:

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

Claim:

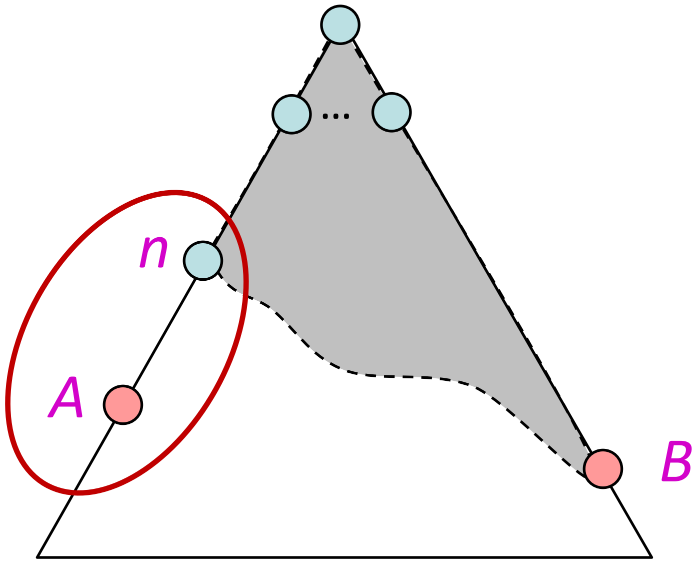
- *A* will be chosen for expansion before *B*



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

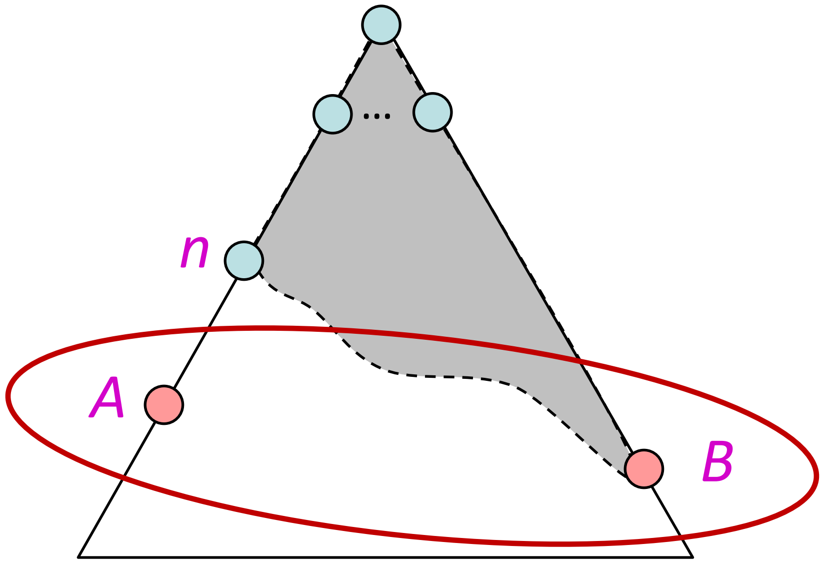


$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \leq g(A)$	Admissibility of h
$g(A) = f(A)$	$h = 0$ at a goal

Optimality of A* Tree Search: Blocking

Proof:

- Imagine *B* is on the 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. $f(A) < f(B)$



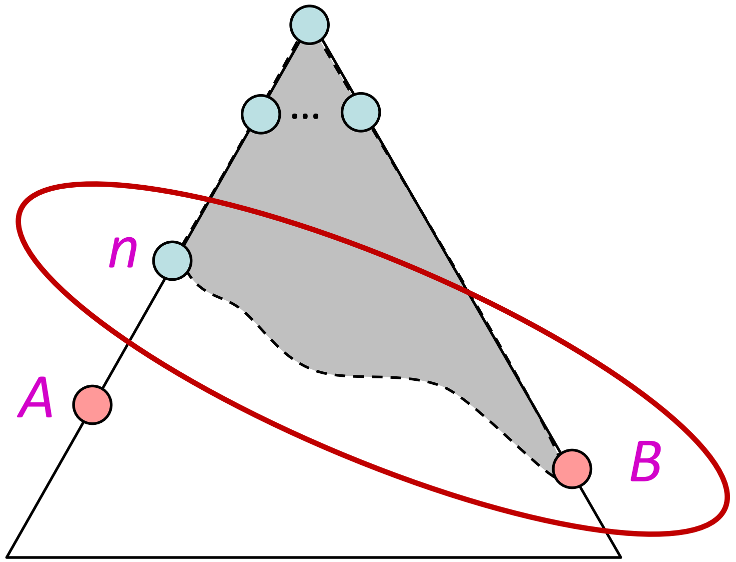
$g(A) < g(B)$ Suboptimality of *B*

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

Optimality of A* Tree Search: Blocking

Proof:

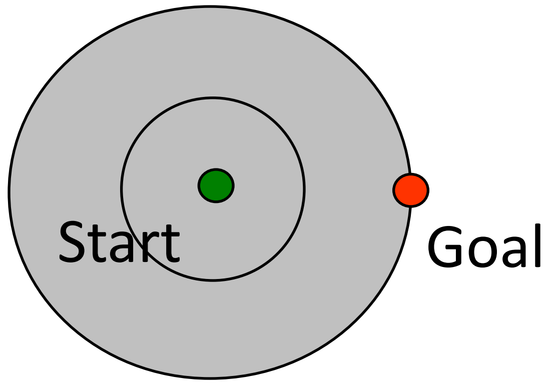
- Imagine *B* is on the frontier
- Some ancestor *n* of *A* is on the frontier, too (maybe *A* itself!)
- Claim: *n* will be expanded before *B*
  1. $f(n) \leq f(A)$
  2. $f(A) < f(B)$
  3. *n* is expanded before *B*
- All ancestors of *A* are expanded before *B*
- *A* is expanded before *B*
- **A*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



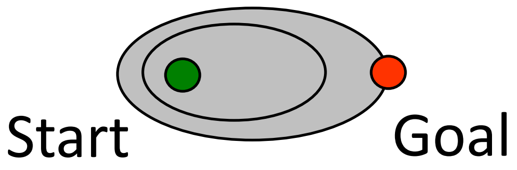
$f(n) \leq f(A) < f(B)$

UCS vs A* Contours

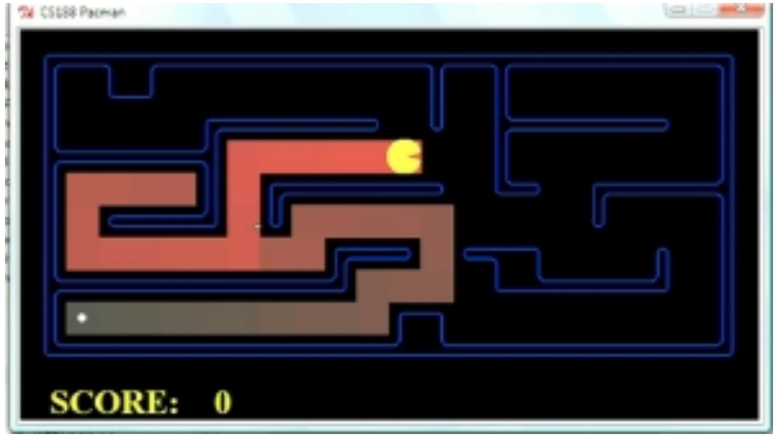
- Uniform-cost expands equally in all "directions"

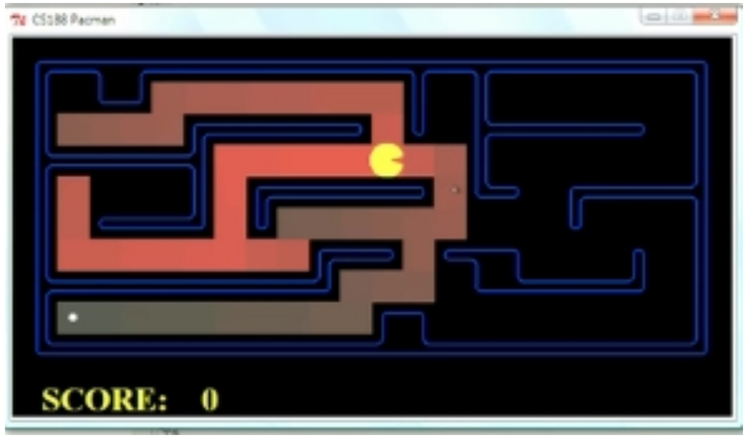


- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



Comparison





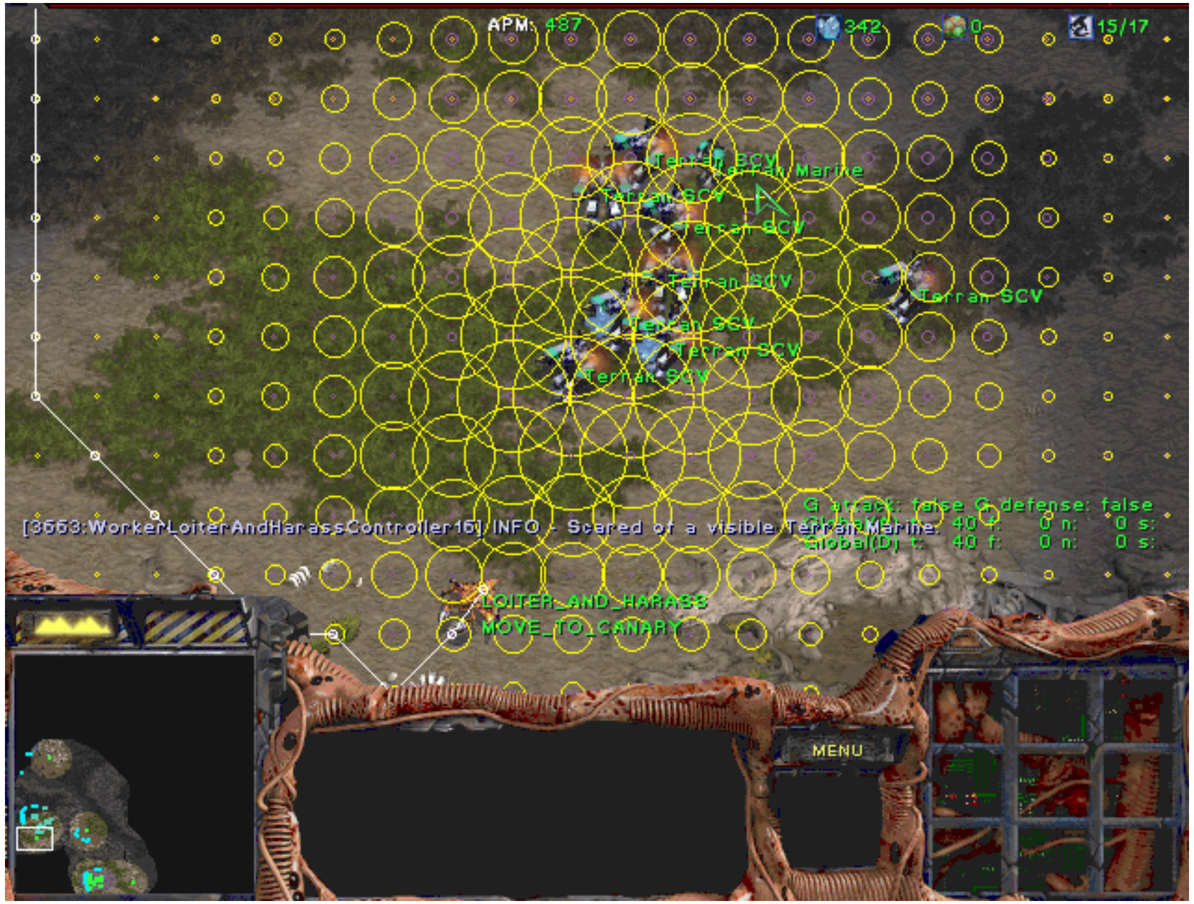
Greedy (h)

Uniform Cost (g)

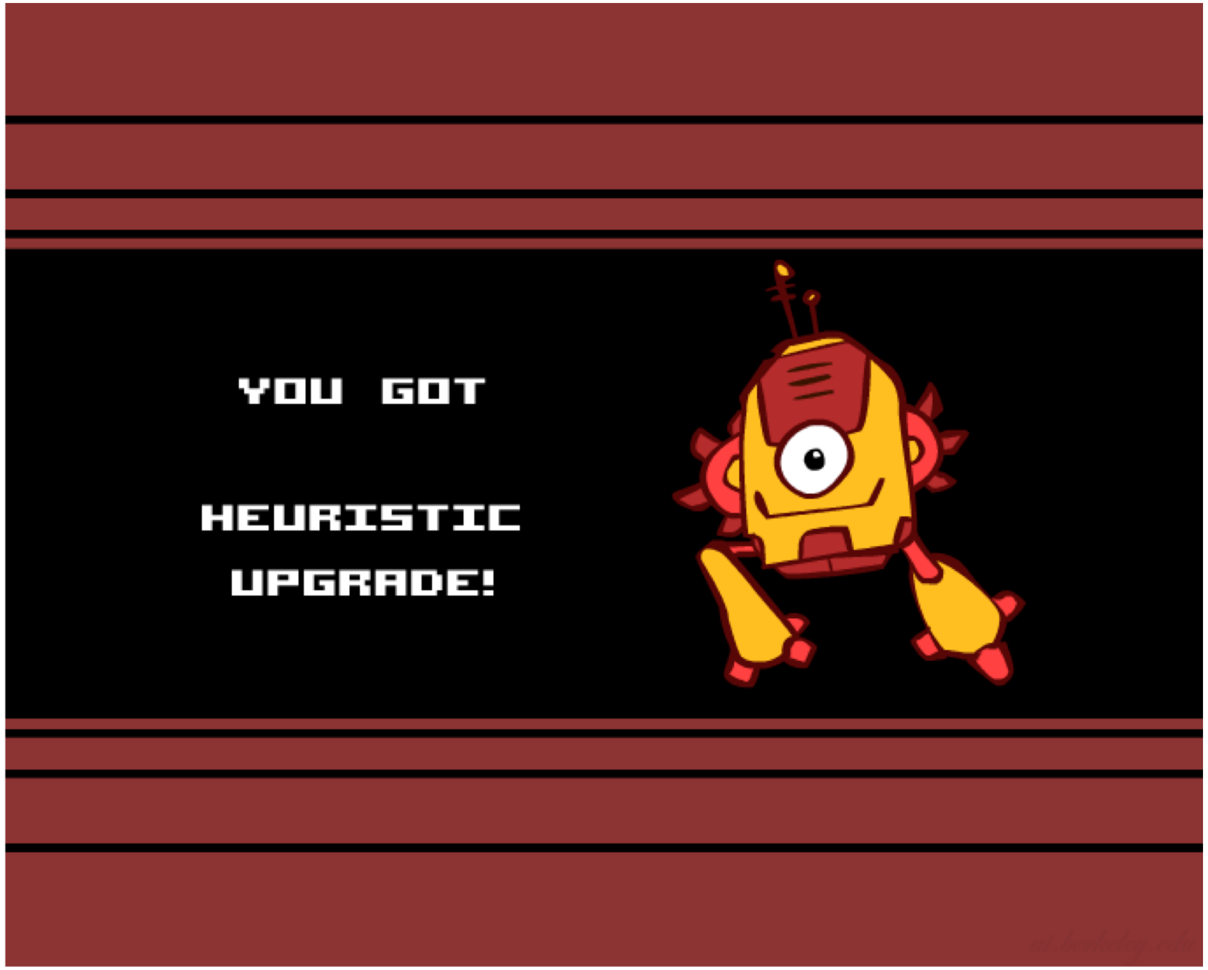
A* (g+h)

A* Applications

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



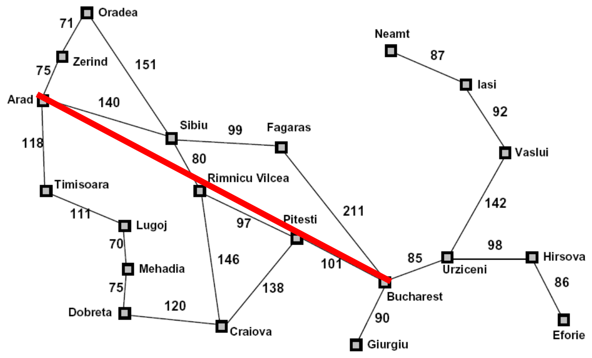
Creating Heuristics

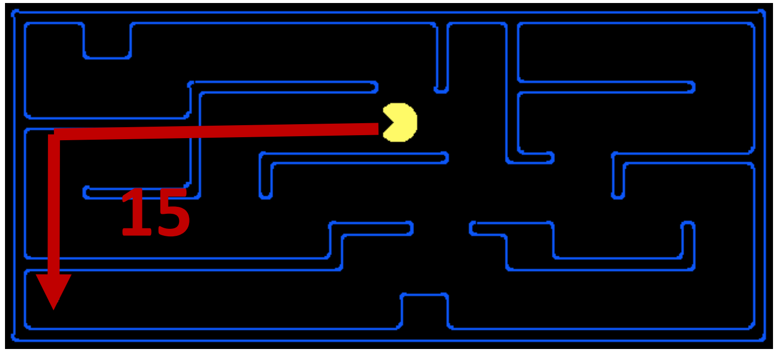


Creating Admissible Heuristics

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

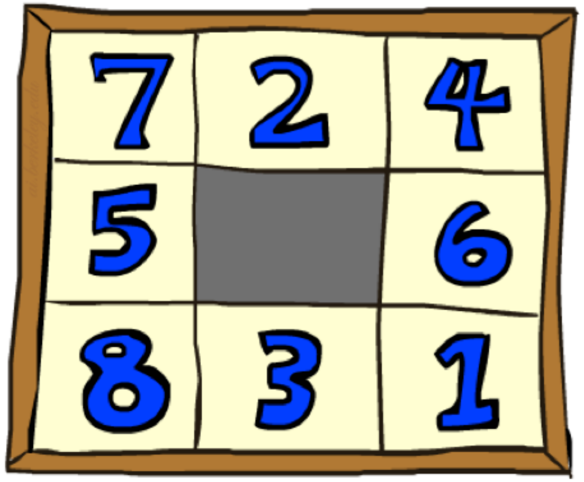
366



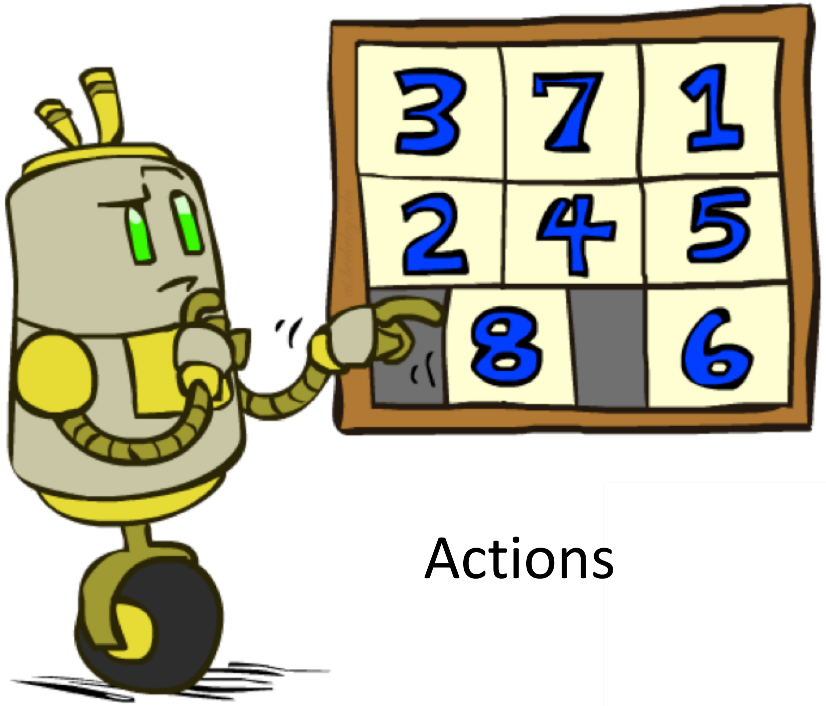


- Problem P_2 is a relaxed version of P_1 if $\mathcal{A}_2(s) \supseteq \mathcal{A}_1(s)$ for every s
- Theorem: $h_2^*(s) \leq h_1^*(s)$ for every s, so $h_2^*(s)$ is admissible for P_1

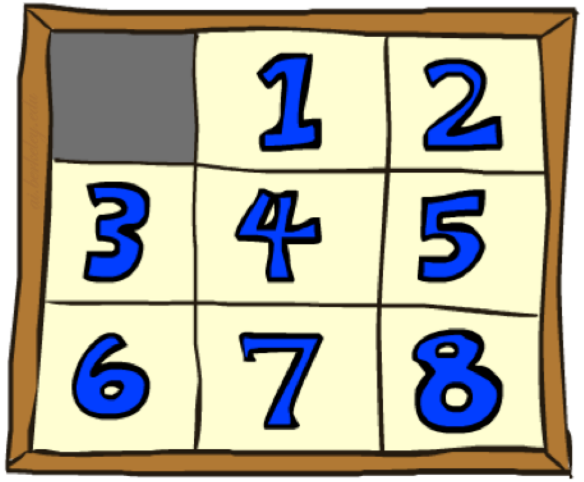
Example: 8 Puzzle



Start State



Actions

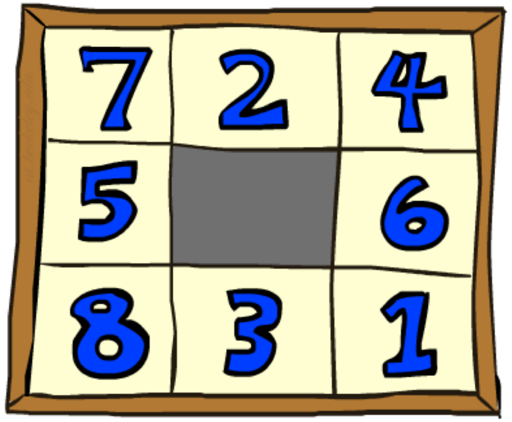


Goal State

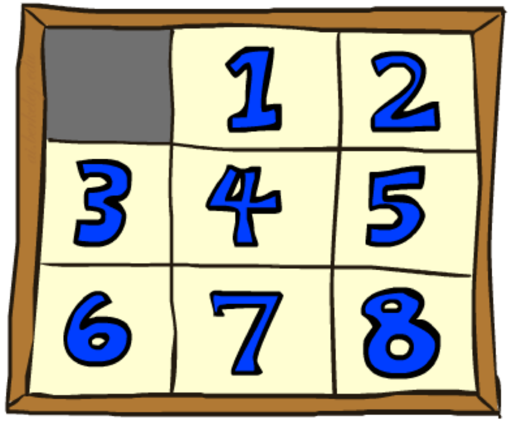
- What are the states?
- How many states?
- What are the actions?
- What are the step costs?

8 Puzzle I

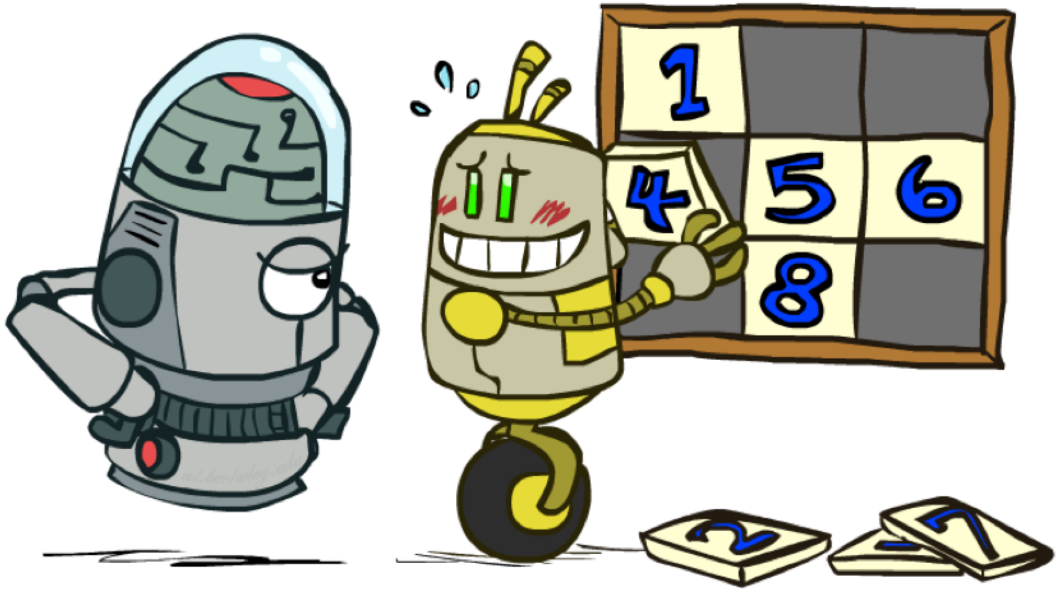
- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8



Start State



Goal State

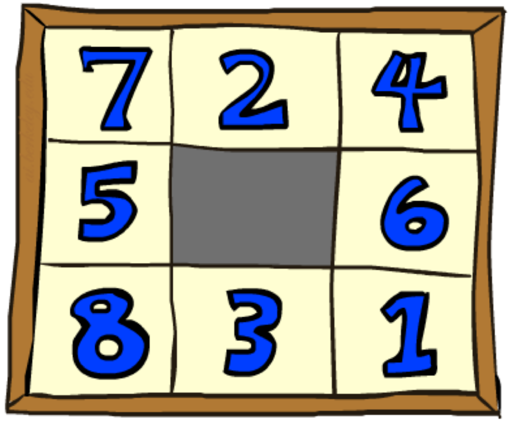


	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
A*TILES	13	39	227

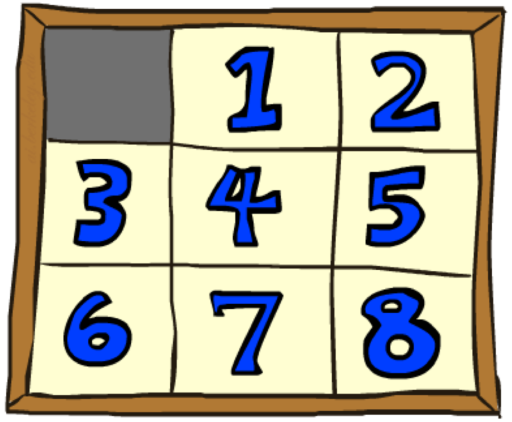
Statistics from Andrew Moore

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
A*TILES	13	39	227
A*MANHATTAN	12	25	73

Combining heuristics

- Dominance: $h_1 \geq h_2$ if

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

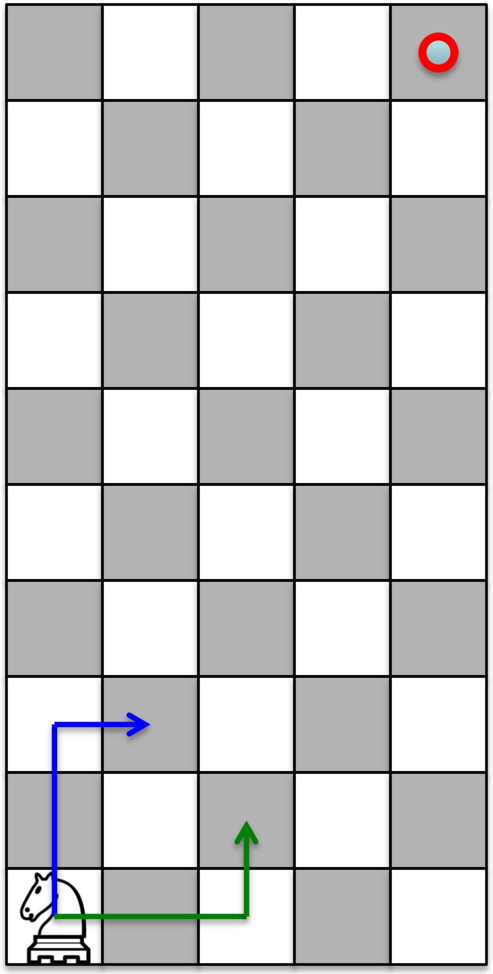
  - Roughly speaking, larger is better as long as both are admissible
  - The zero heuristic is pretty bad (what does A* do with h=0?)
  - The exact heuristic is pretty good, but usually too expensive!
- What if we have two heuristics, neither dominates the other?
  - Form a new heuristic by taking the max of both:

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

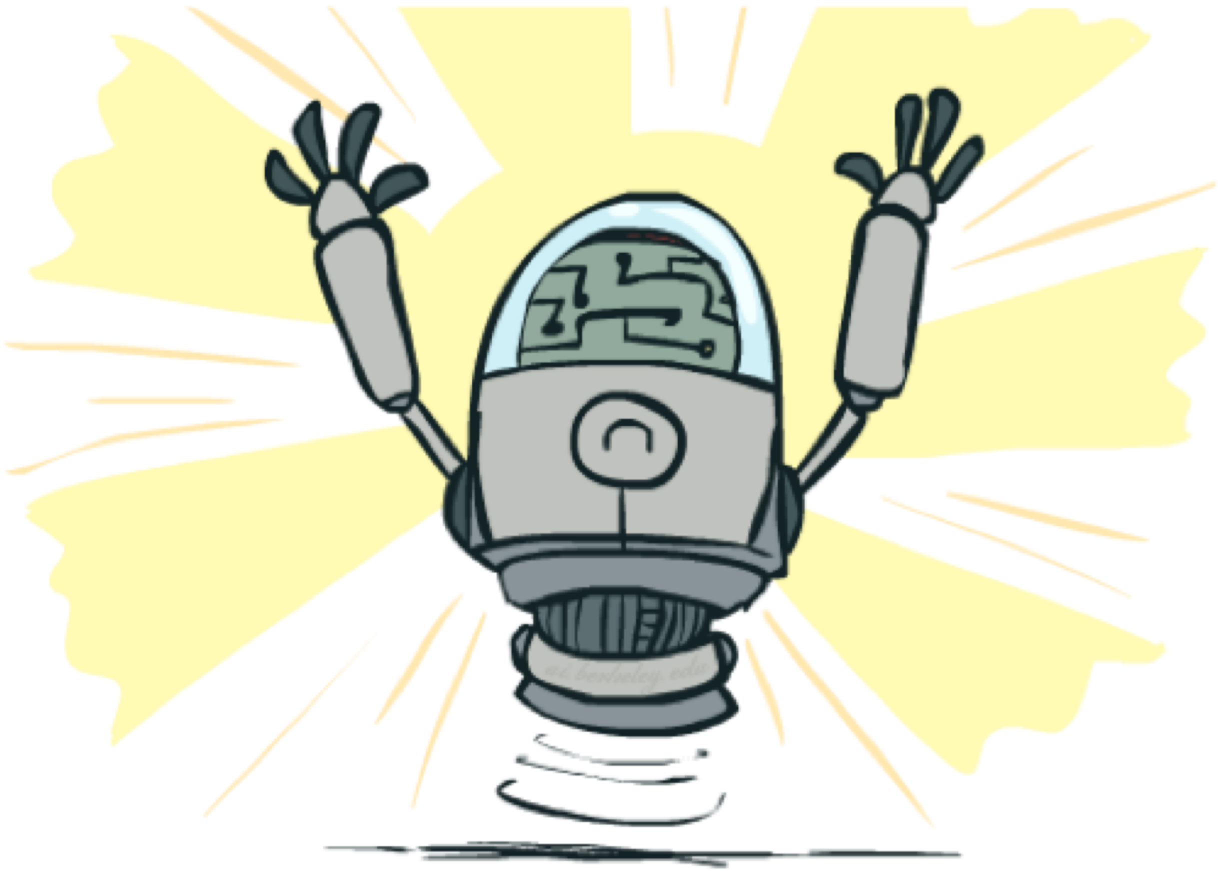
  - Max of admissible heuristics is admissible and dominates both!

Example: Knight's moves

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

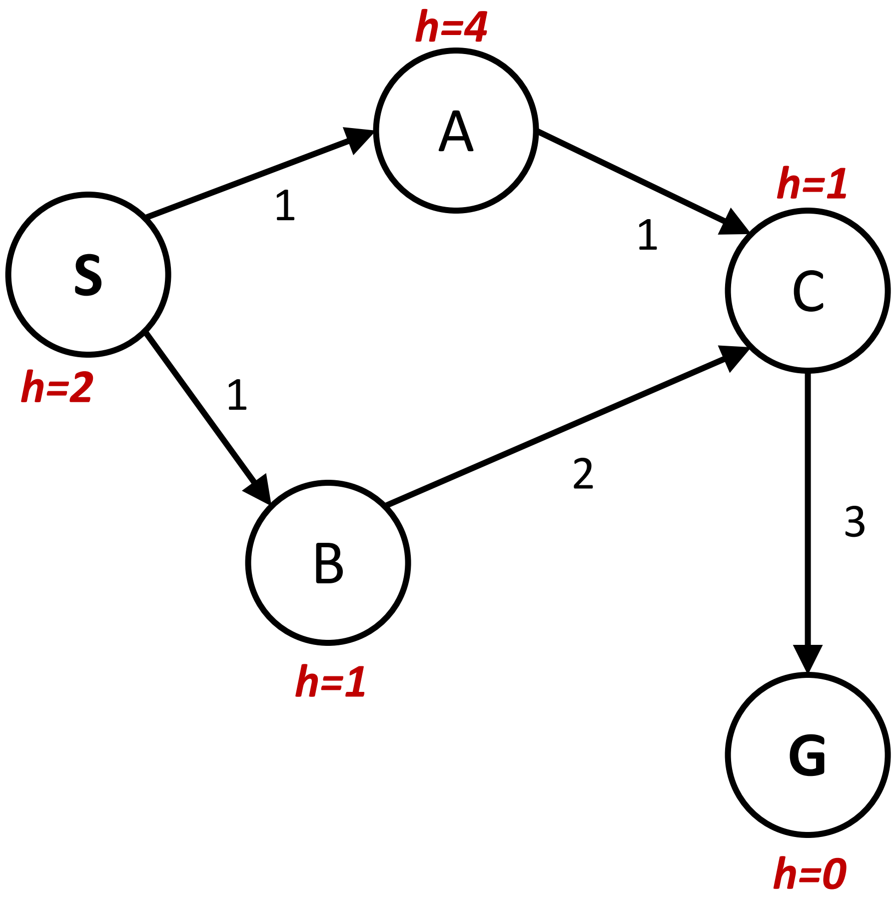


Optimality of A* Graph Search

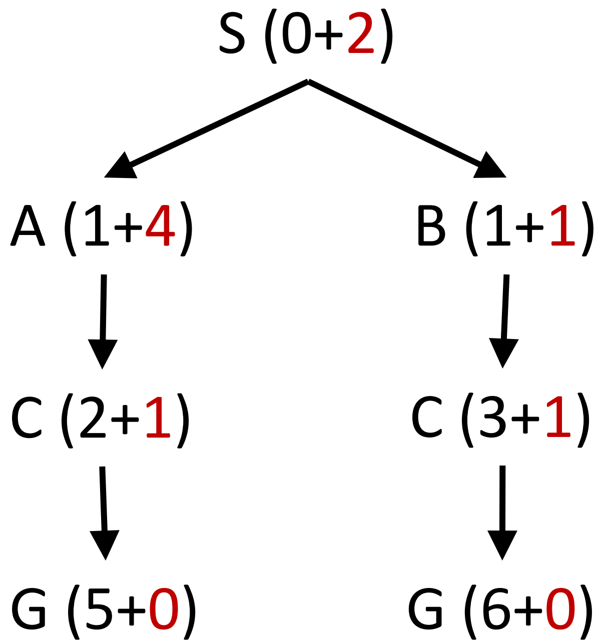


A* Graph Search Gone Wrong?

State space graph

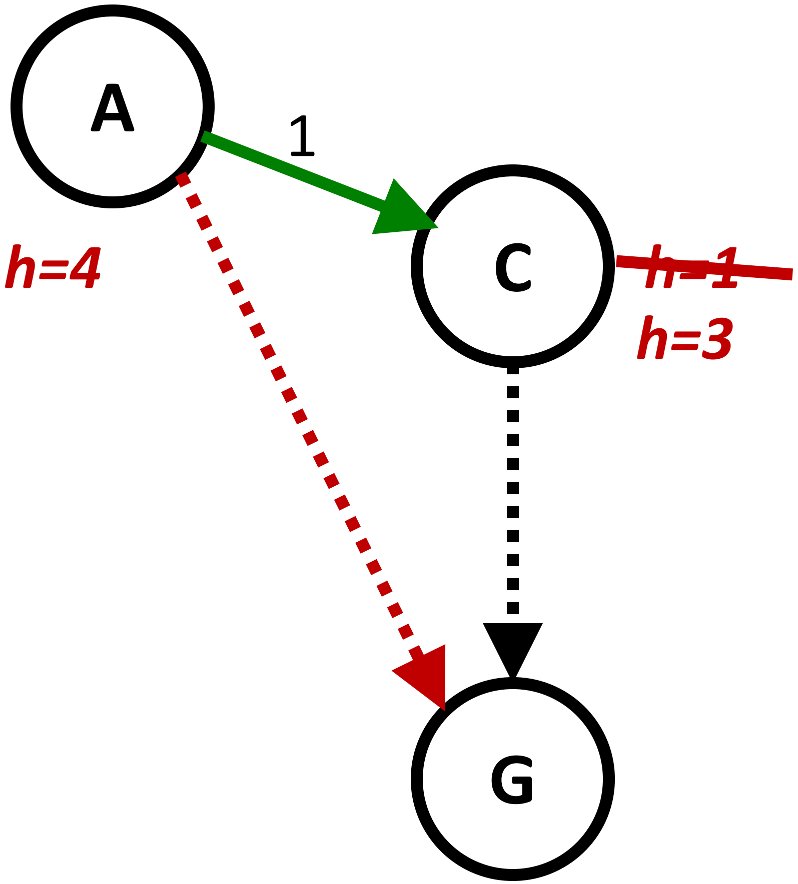


Search tree



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 ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal

    $h(A) \leq h^*(A)$
  - Consistency: heuristic "arc" cost ≤ actual cost for each arc

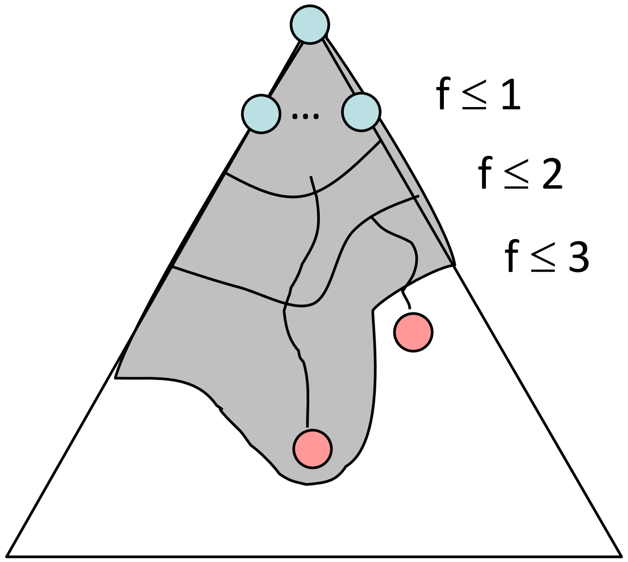
    $h(A) - h(C) \leq c(A,C)$

    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:

    $h(A) \leq c(A,C) + h(C) \quad \Rightarrow g(A) + h(A) \leq g(A) + c(A,C) + h(C)$
  - A* graph search is optimal

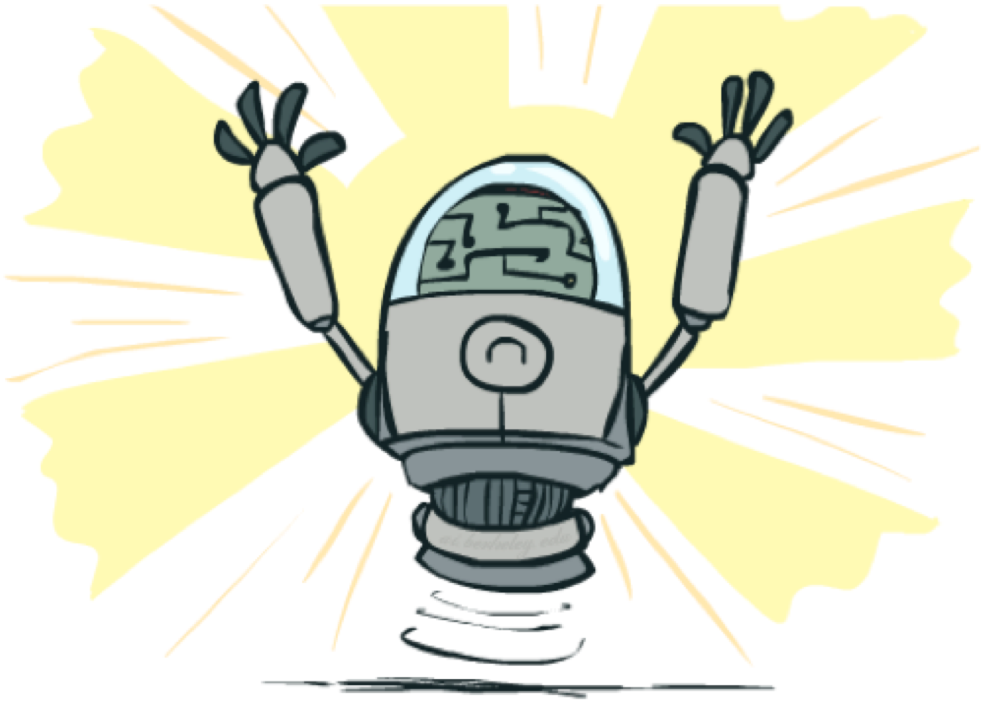
Optimality of A* Graph Search

- 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

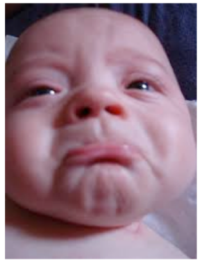


Optimality

- 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