Informed Search: A* Algorithm

CE417: Introduction to Artificial Intelligence
Sharif University of Technology
Fall 2023

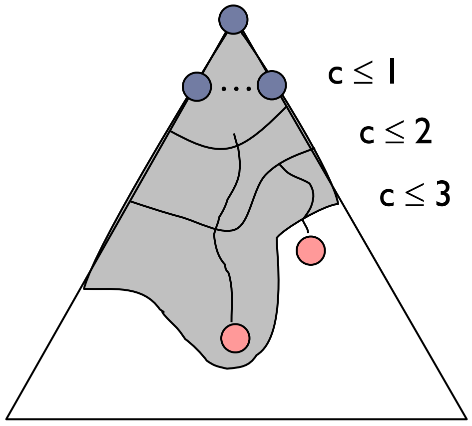
Soleymani

Most slides have been adopted from Klein and Abdeel, CS188, UC Berkeley.

Uninformed search

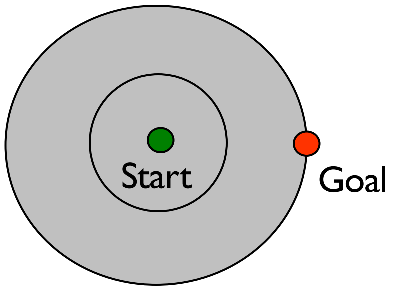
Uniform Cost Search

- Strategy: expand lowest path cost



- The good: UCS is complete and optimal!

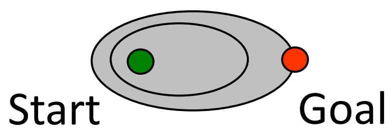
- The bad:
  - Explores options in every "direction"
  - No information about goal location



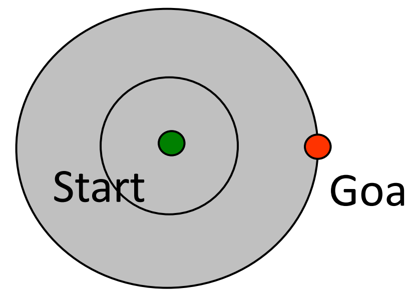
UCS example

What we would like to have happen

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

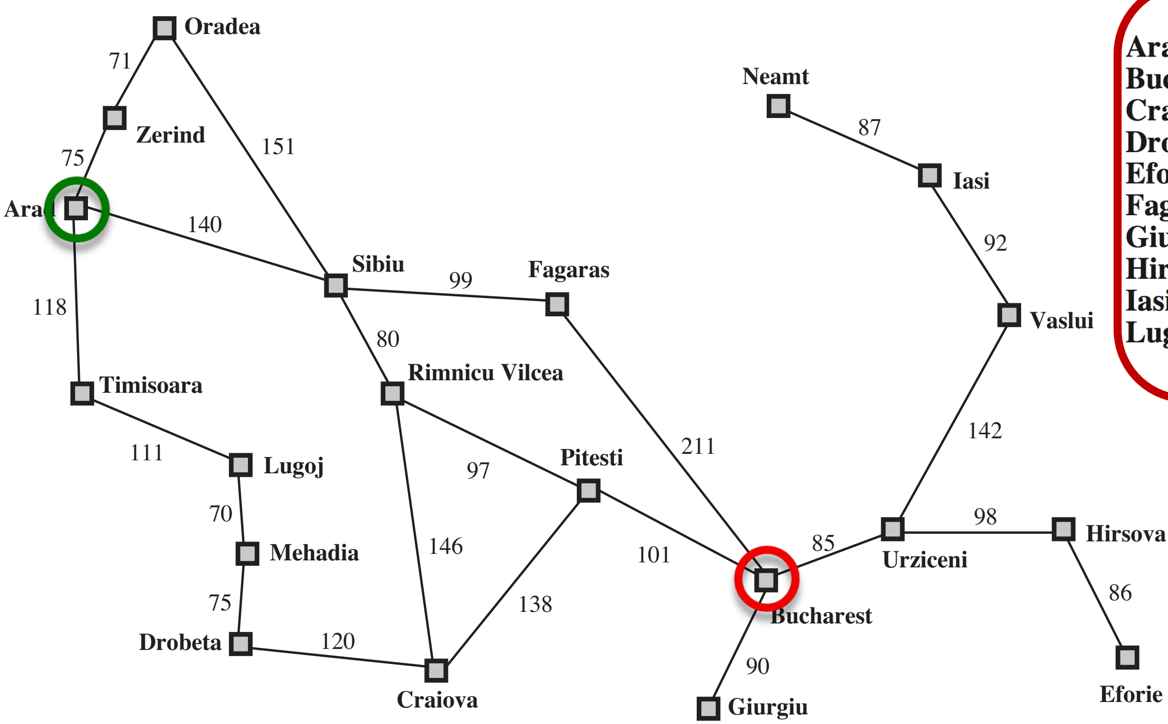


Informed



Uninformed

Example: Route-finding in Romania

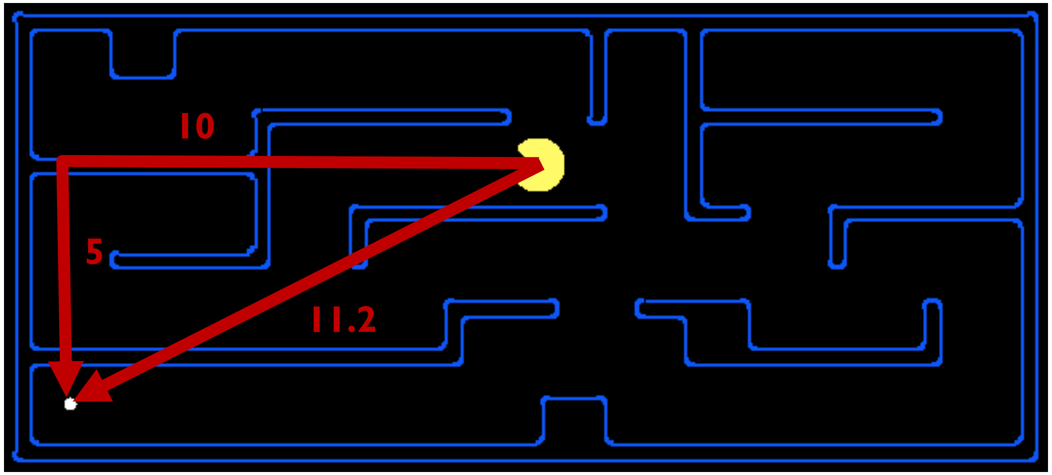


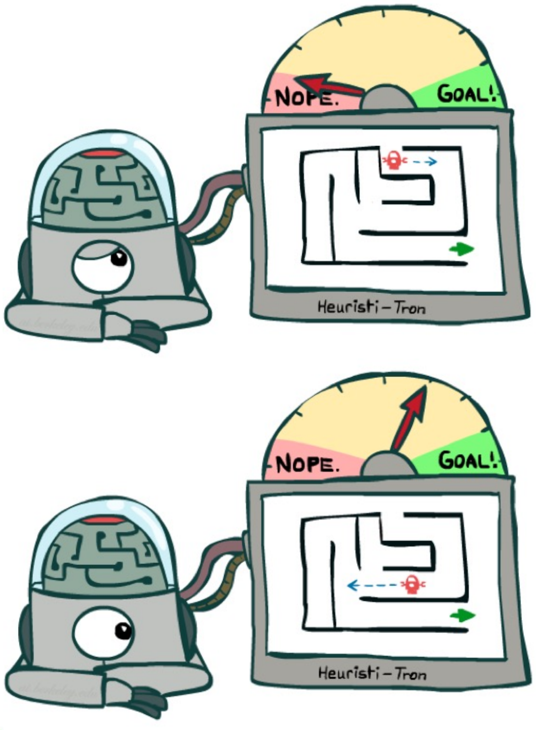
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

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

Search heuristics

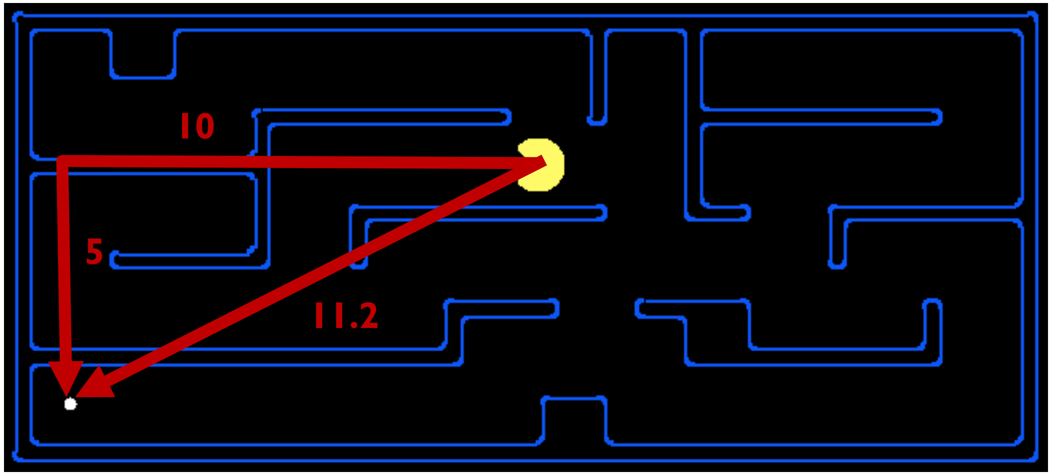
- A heuristic is:
  - A function that *estimates* how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance for pathing

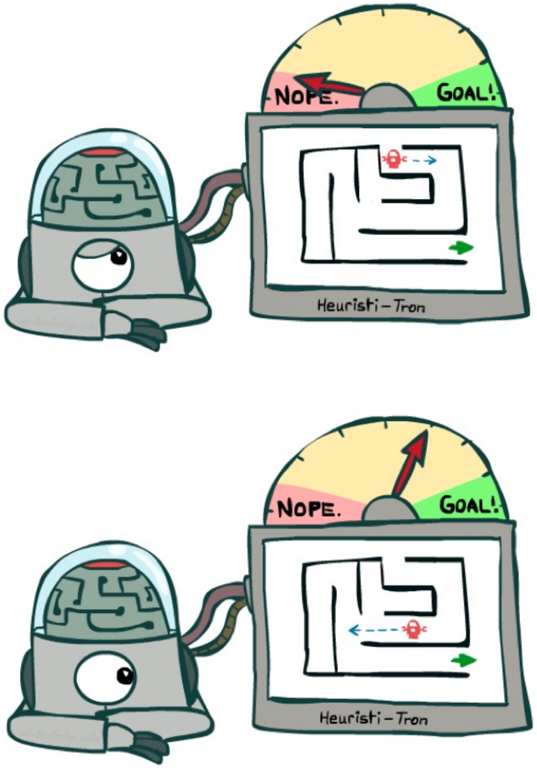




Example: Pathing in pacman

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





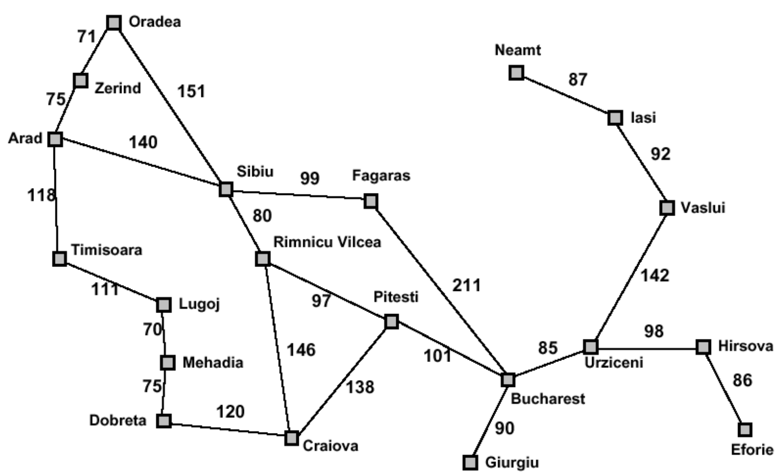
Greedy Best First Search

Greedy Best First Search

- Priority queue based on $h(n)$
  - e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest

- Greedy search expands the node that appears to be closest to goal

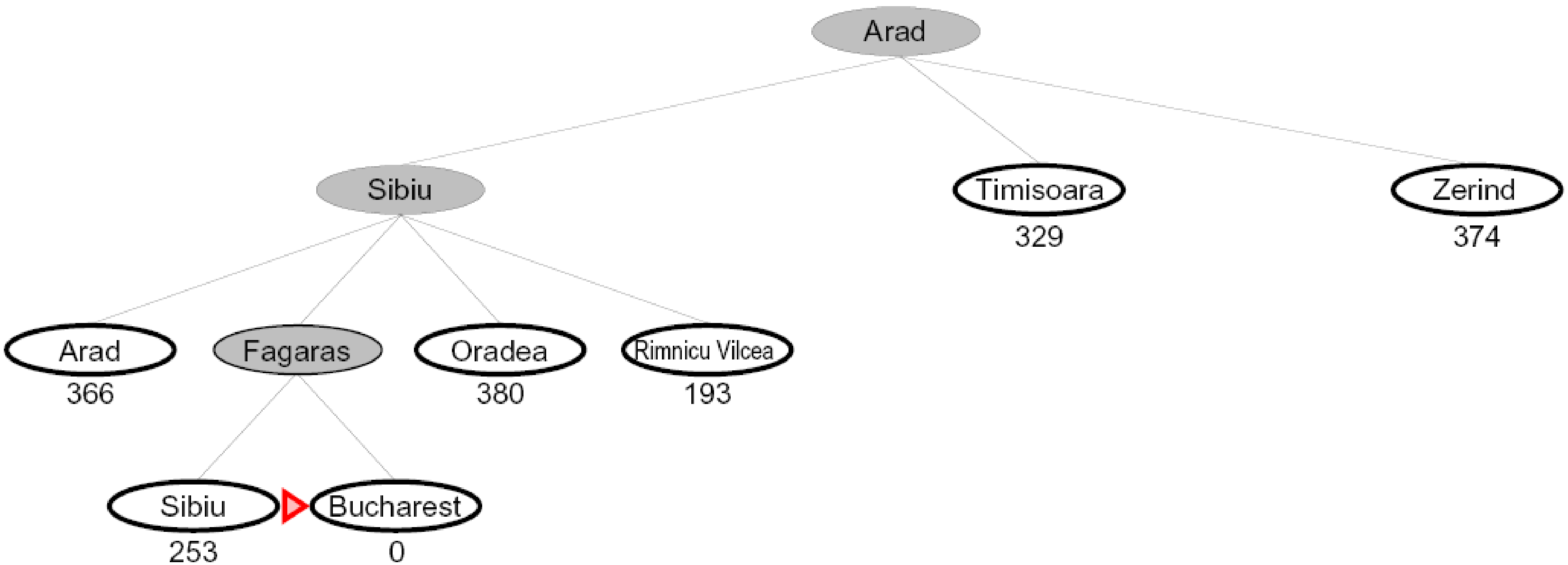
Greedy

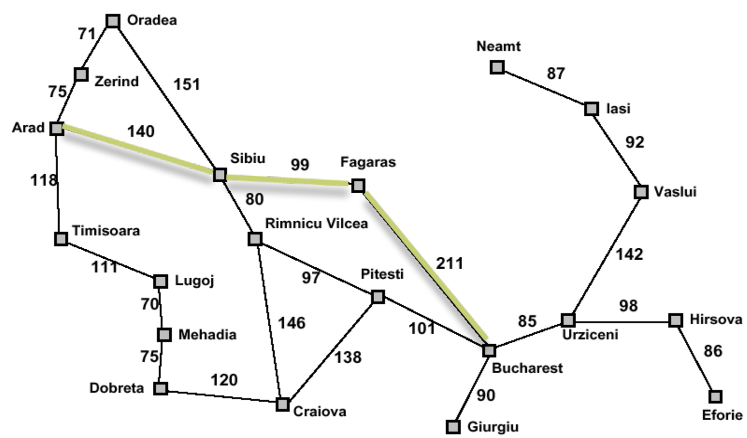


Straight–line distance to Bucharest

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

Greedy Best First Search



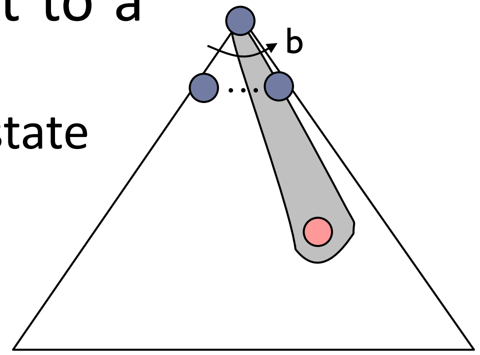


Straight–line distance to Bucharest

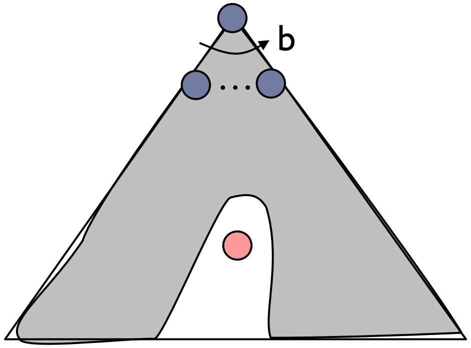
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Best First Search

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



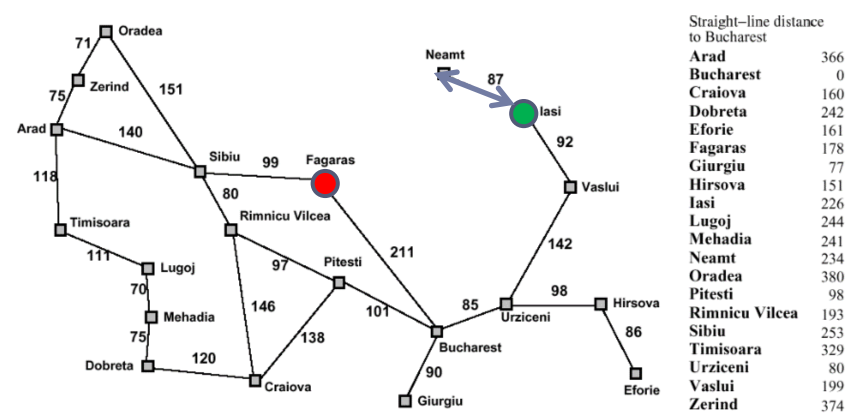
- A common case:
  - Best-first takes you straight to the (wrong) goal



- Worst-case: like a badly-guided DFS

Properties of greedy search

- Complete? No
  - Similar to DFS, only graph search version is complete in finite spaces
  - Infinite loops, e.g., (Iasi to Fagaras) Iasi → Neamt → Iasi → Neamt
- Time
  - $O(b^m)$, but a good heuristic can give dramatic improvement
- Space
  - $O(b^m)$: keeps all nodes in memory
- Optimal? No



Video of demo contours greedy (Empty)

Video of demo contours greedy (Pacman small maze)

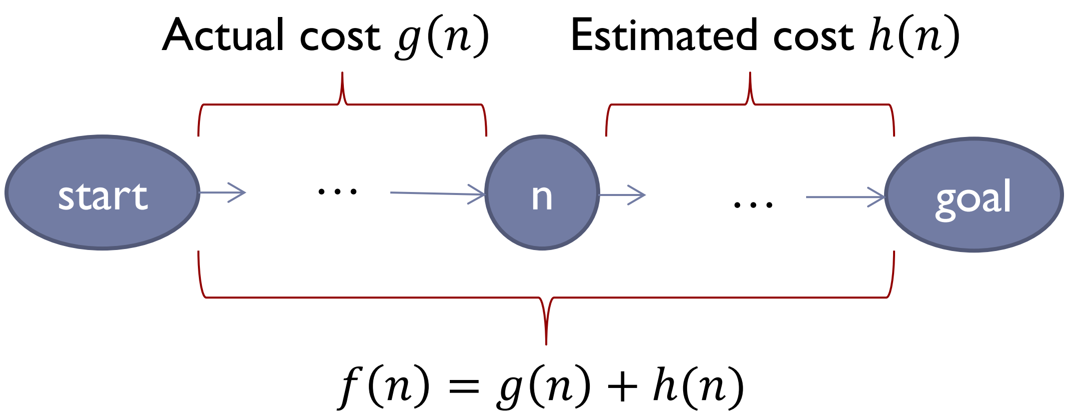
A* search

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

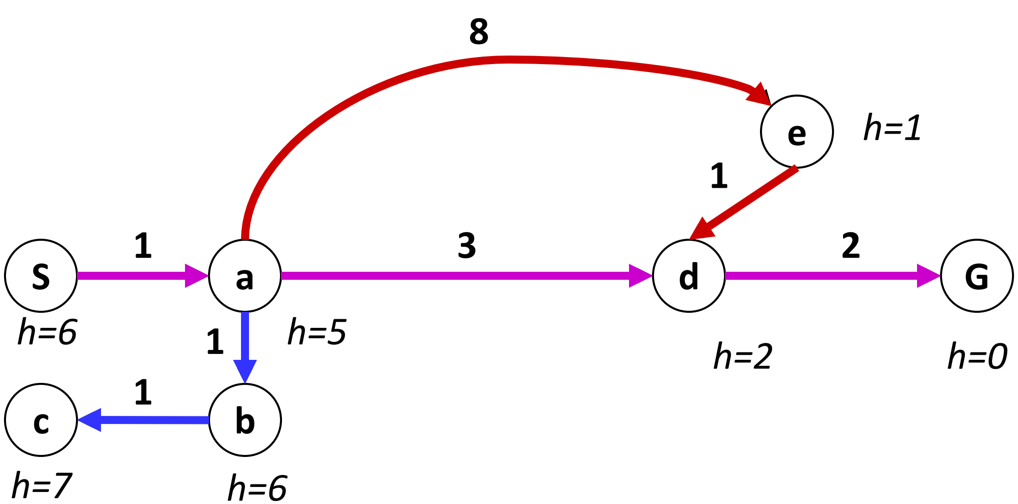
A* search

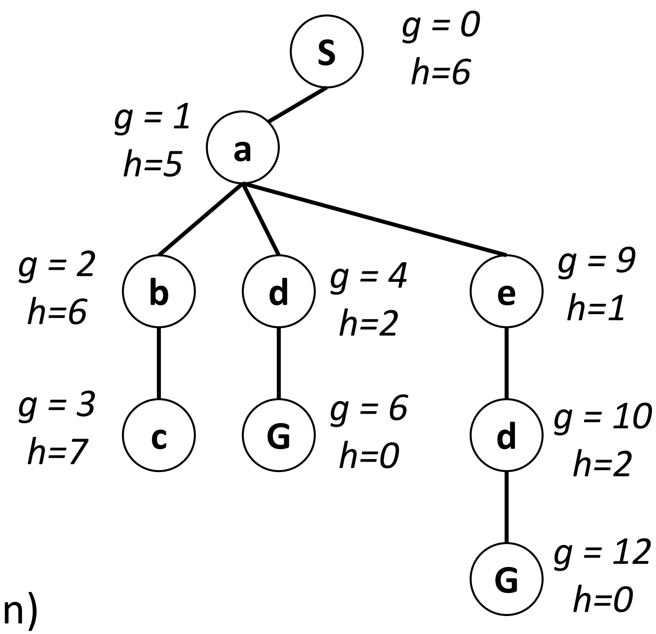
- Idea: minimizing the total estimated solution cost
- Evaluation function for priority $\underline{f(n) = g(n) + h(n)}$
  - $g(n) =$ cost so far to reach n
  - $h(n) =$ estimated cost of the cheapest path from n to goal
  - So, $f(n) =$ estimated total cost of path through n to goal



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)



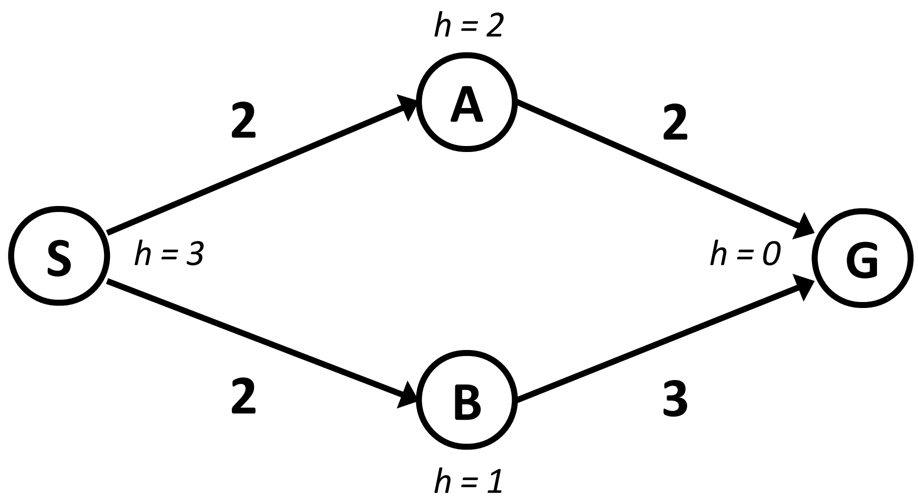


- A* Search orders by the sum: f(n) = g(n) + h(n)

Example: Teg Grenager

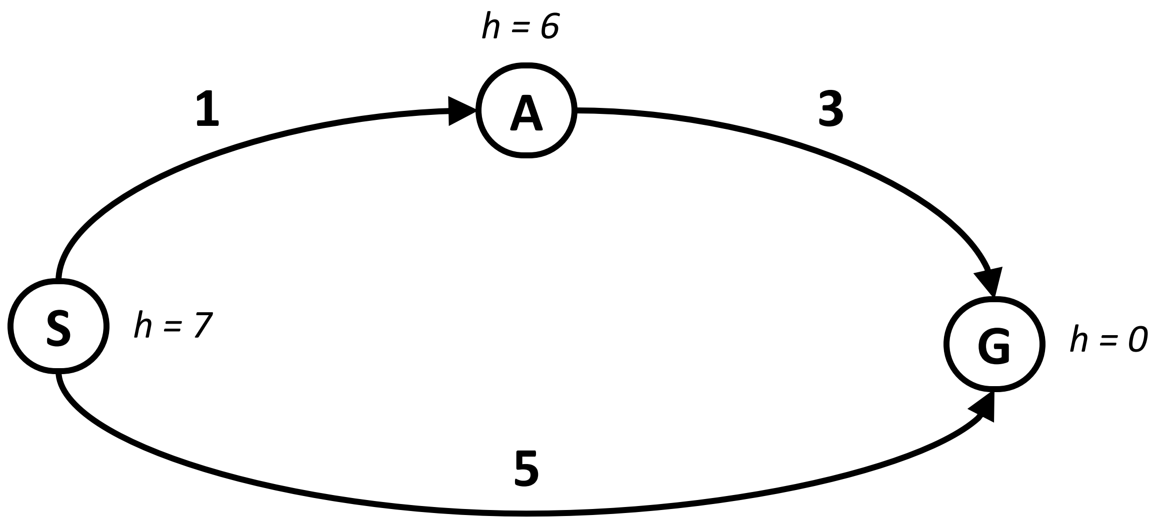
A* termination

- Should we stop when we enqueue a goal?



- No: only stop when we dequeue a goal

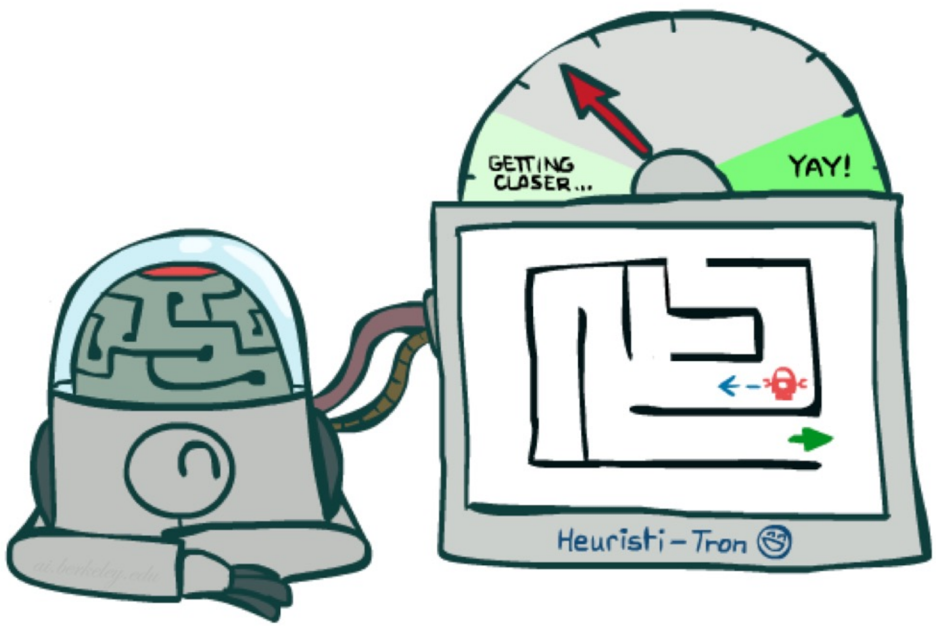
Is A* Optimal?



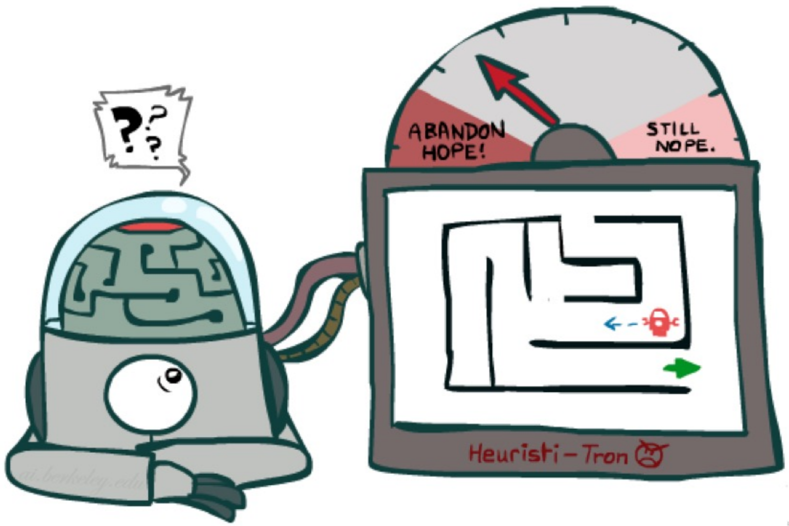
What went wrong?

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

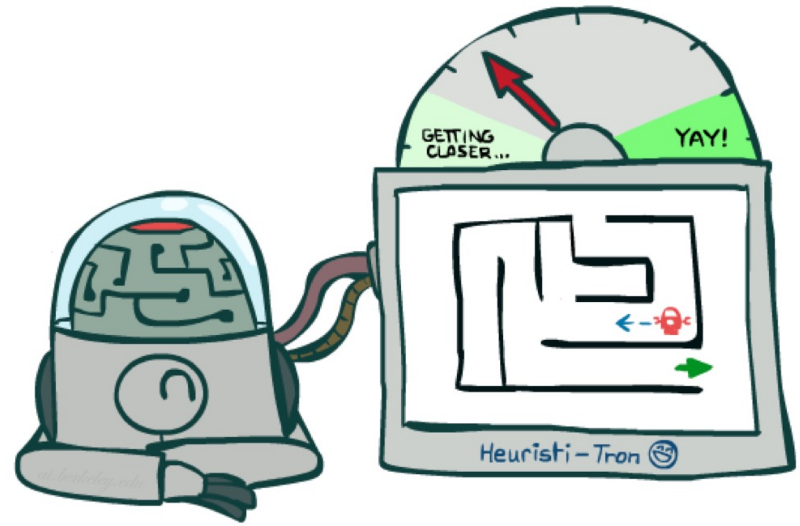
Admissible Heuristics



Idea: Admissibility



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



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

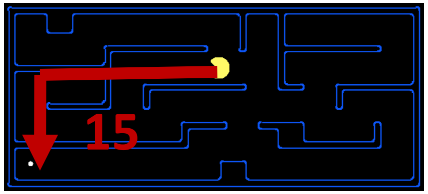
Admissible Heuristics

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

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

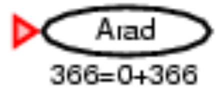
- where $h^*(n)$ is the true cost to a nearest goal

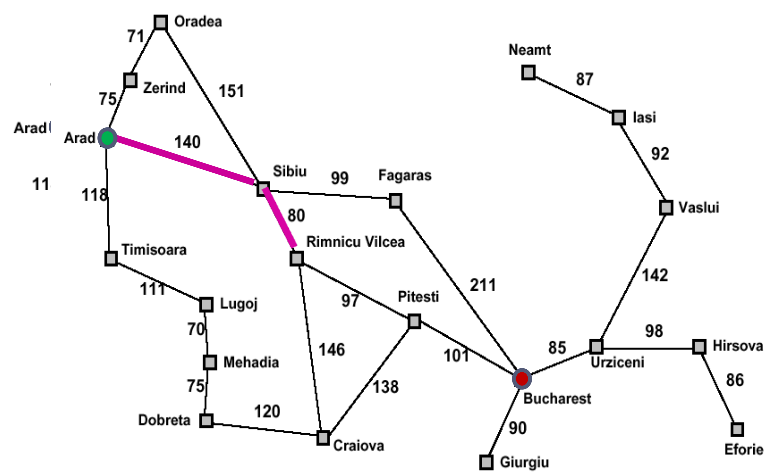
- Examples:



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

A* search: example

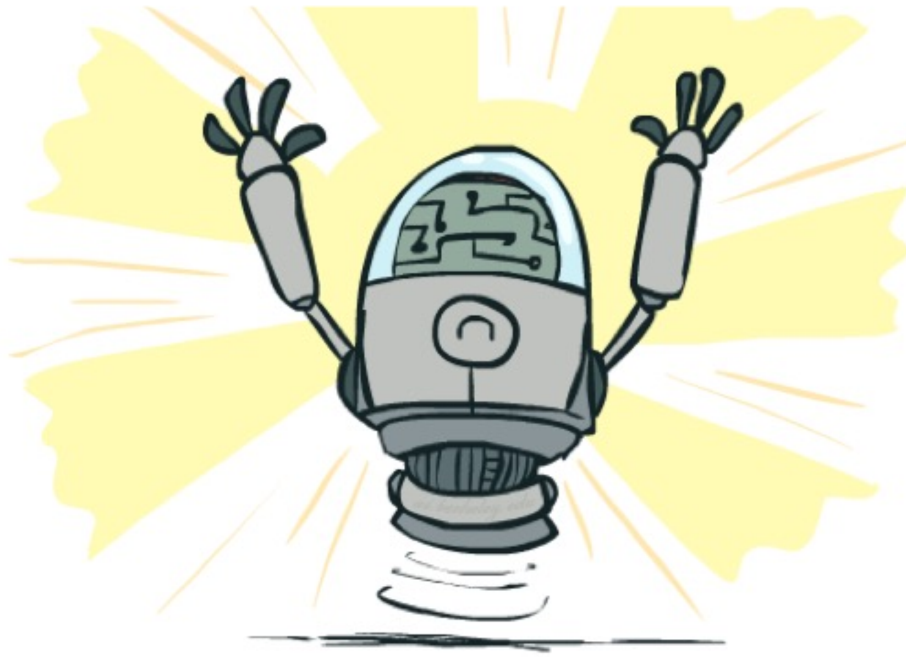




Straight−line distance to Bucharest

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

Optimality of A* Tree Search



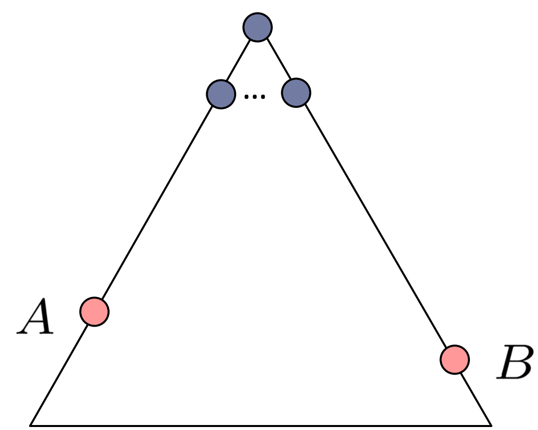
Optimality of A* Tree Search

Assume:

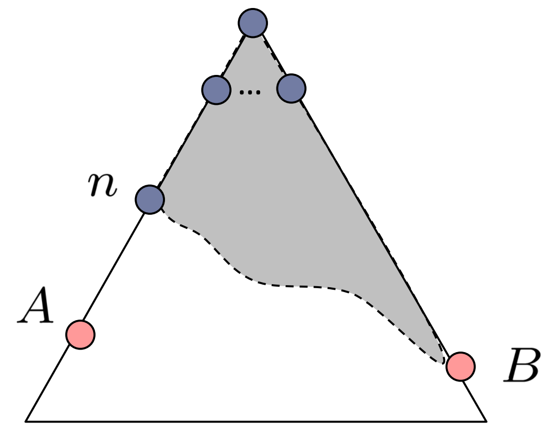
- A is an optimal goal node
- B is a suboptimal goal node
- h is admissible

Claim:

- A will exit the frontier before B



Optimality of A* Tree Search: Blocking



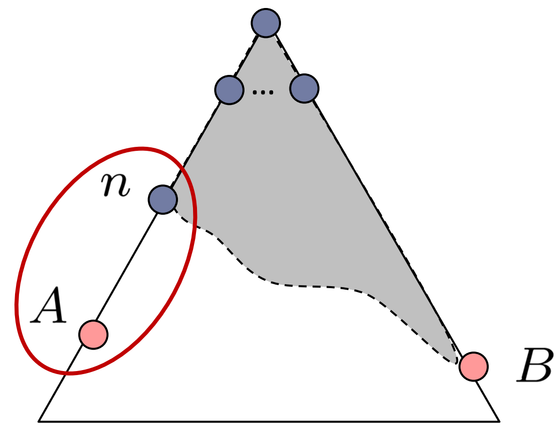
Proof:

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

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!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)

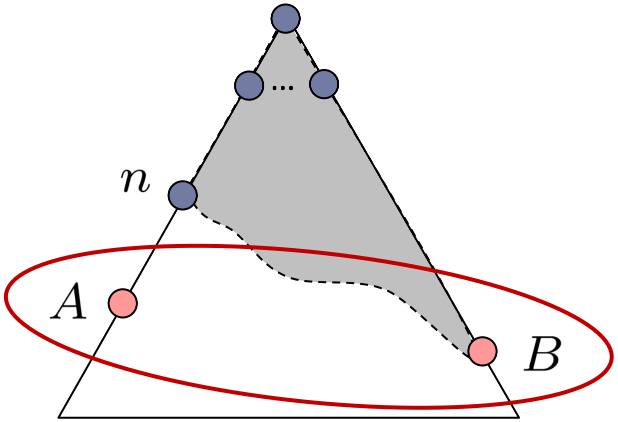


$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \leq g(n) + h^*(n)$	Admissibility of h
$f(n) \leq g(A)$	$g(A) = g(n) + h^*(n)$
$g(A) = f(A)$	h = 0 at a goal

Optimality of A* Tree Search: Blocking

Proof:

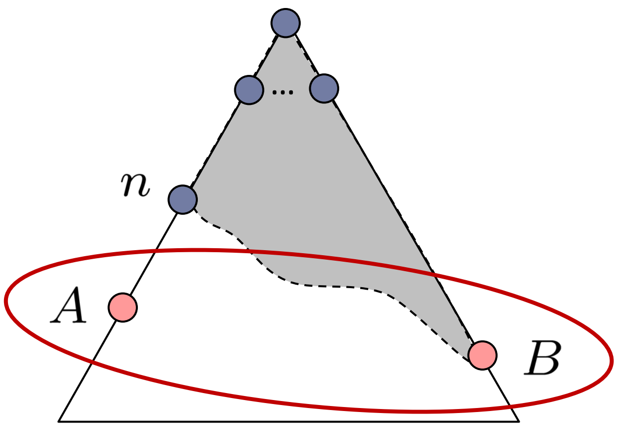
- Imagine B is on the frontier
- Some ancestor *n* of A is on the frontier, too (maybe A!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)



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!)
- 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)



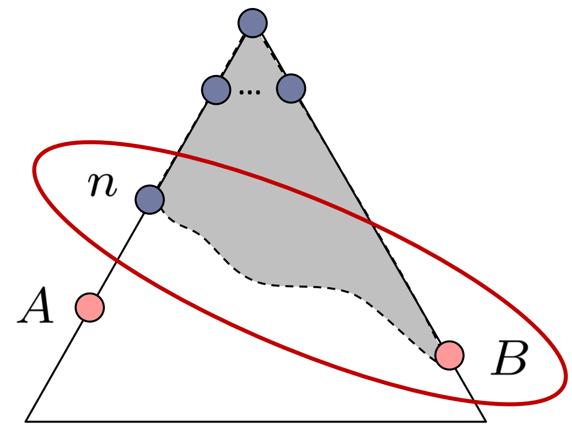
$g(A) < g(B)$ B is suboptimal

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

Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the frontier
- Some ancestor *n* of A is on the frontier, 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

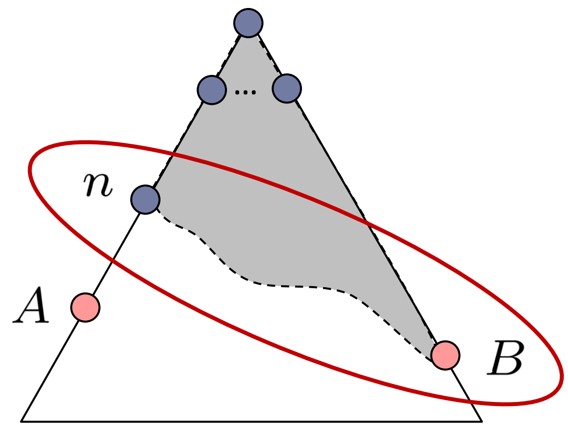


$$f(n) \leq f(A) < f(B)$$

Optimality of A* Tree Search: Blocking

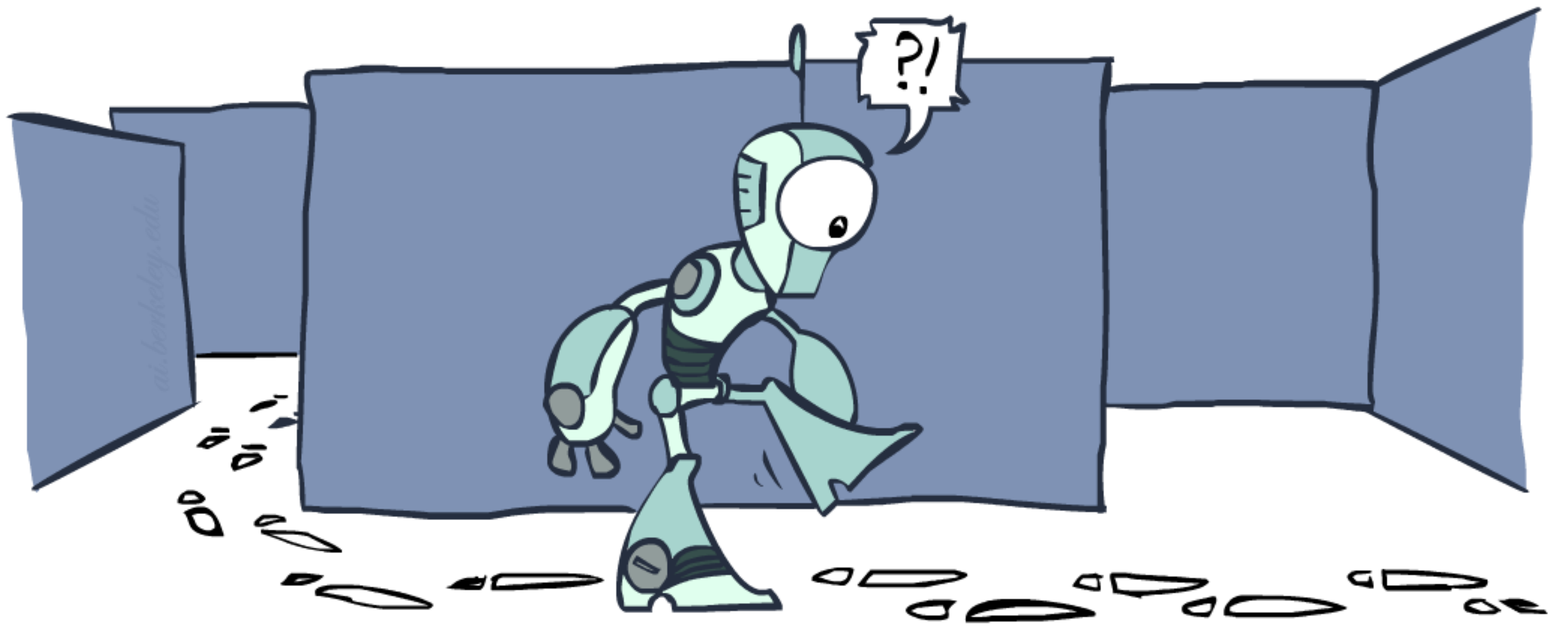
Proof:

- Imagine B is on the frontier
- Some ancestor *n* of A is on the frontier, too (maybe A!)
- 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



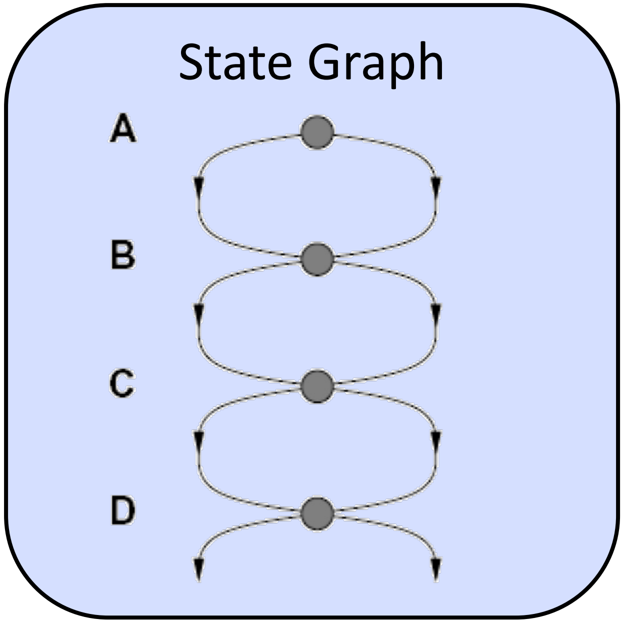
$$f(n) \leq f(A) < f(B)$$

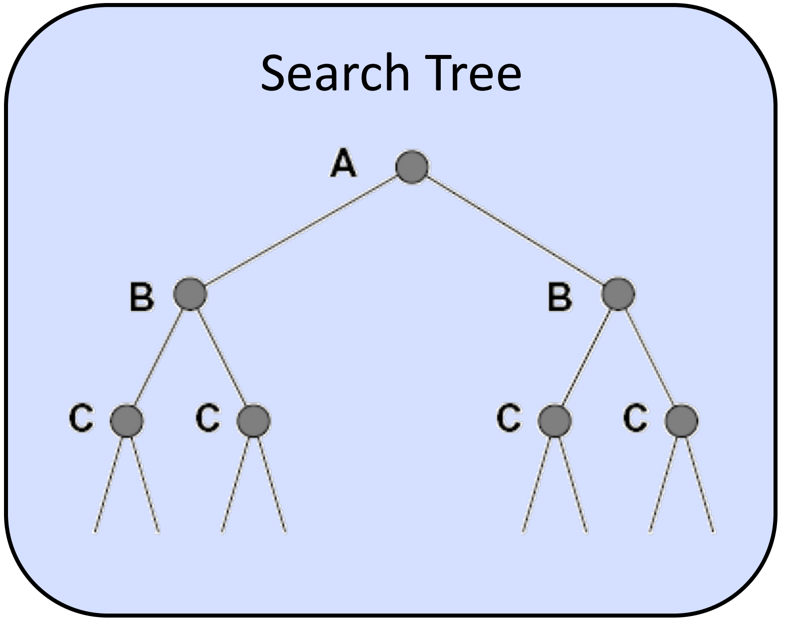
Graph Search



Tree Search: Extra Work!

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



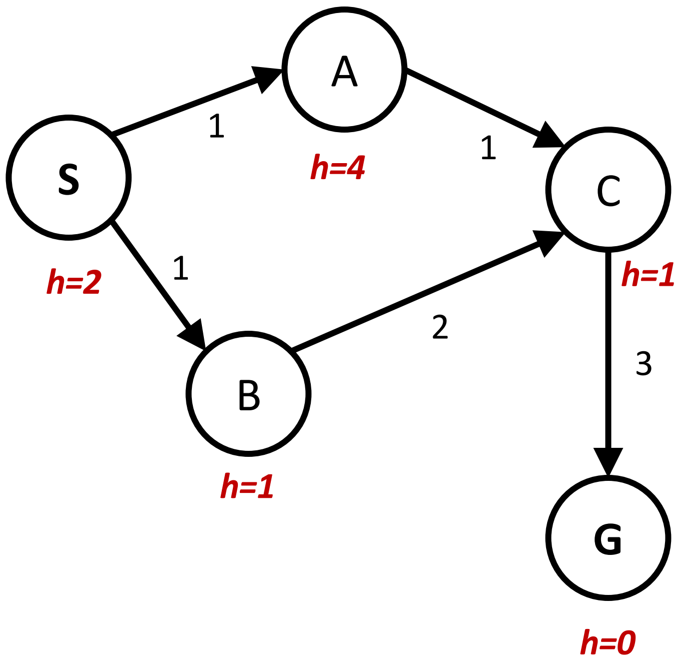


Graph Search

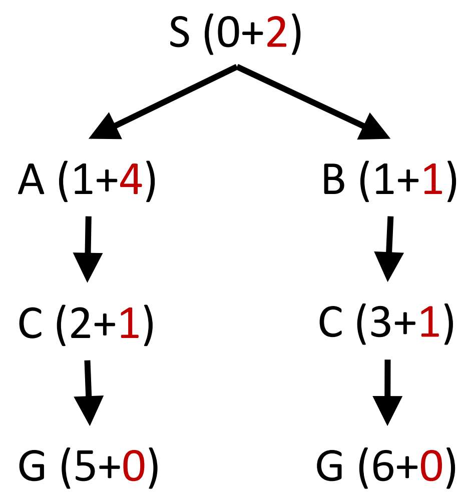
- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states ("closed set")
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to closed set
- Important: 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



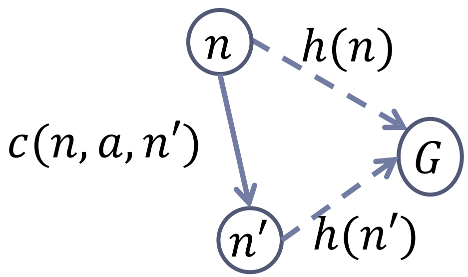
Simple check against expanded set blocks C

Fancy check allows new C if cheaper than old but requires recalculating C's descendants

Conditions for optimality of A*

- Admissibility: $h(n)$ be a lower bound on the cost to reach goal
  - Condition for optimality of TREE−SEARCH version of A*

- Consistency (monotonicity): $h(n) \leq c(n,a,n') + h(n')$
  - Condition for optimality of GRAPH−SEARCH version of A*



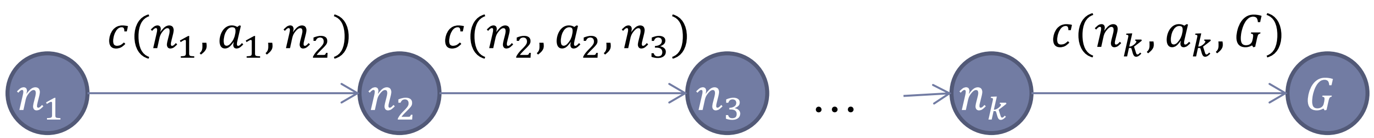
for every node n and every successor n' generated by any action a

$$h(n) \leq c(n,a,n') + h(n')$$

$c(n,a,n')$: cost of generating n' by applying action to n

Consistency implies admissibility

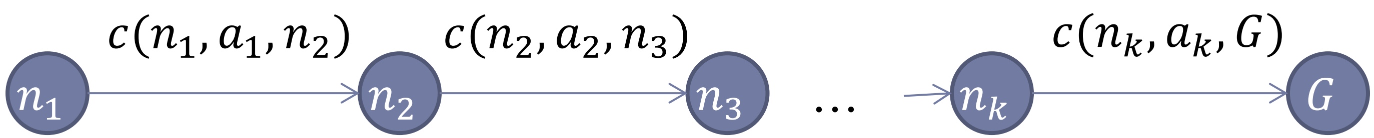
- Consistency ⇒ Admissblity
  - All consistent heuristic functions are admissible
  - Nonetheless, most admissible heuristics are also consistent



$$
\begin{aligned}
h(n_1) &\le c(n_1, a_1, n_2) + h(n_2) \\
&\le c(n_1, a_1, n_2) + c(n_2, a_2, n_3) + h(n_3) \\
&\dots \\
&\le \textstyle\sum_{i=1}^{k} c(n_i, a_i, n_{i+1}) + h(\mathrm{G})
\end{aligned}
$$

Consistency implies admissibility

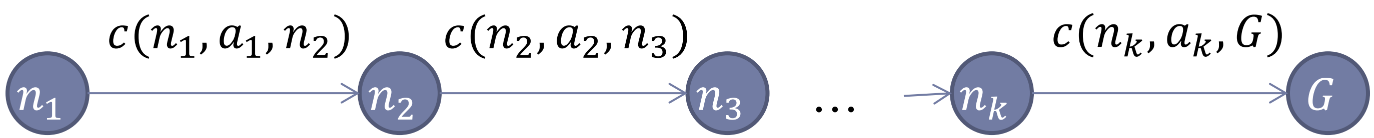
- Consistency ⇒ Admissblity
  - All consistent heuristic functions are admissible
  - Nonetheless, most admissible heuristics are also consistent



$$
\begin{aligned}
h(n_1) &\leq c(n_1, a_1, n_2) + h(n_2) \\
&\leq c(n_1, a_1, n_2) + c(n_2, a_2, n_3) + h(n_3) \\
&\dots \\
&\leq \sum_{i=1}^{k} c(n_i, a_i, n_{i+1}) + \underbrace{h(G)}_{0}
\end{aligned}
$$

Consistency implies admissibility

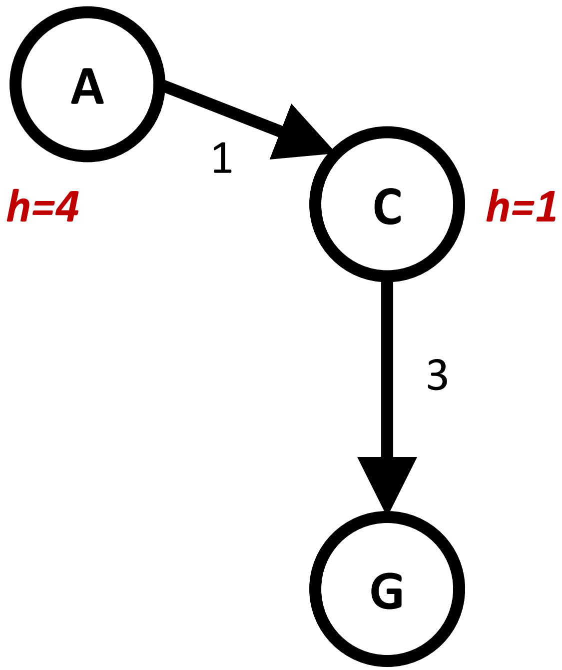
- Consistency ⇒ Admissblity
  - All consistent heuristic functions are admissible
  - Nonetheless, most admissible heuristics are also consistent



$$
\begin{aligned}
h(n_1) &\leq c(n_1, a_1, n_2) + h(n_2) \\
&\leq c(n_1, a_1, n_2) + c(n_2, a_2, n_3) + h(n_3) \\
&\dots \\
&\leq \textstyle\sum_{i=1}^{k} c(n_i, a_i, n_{i+1}) + h(G) \; \mathbf{0}
\end{aligned}
$$

$\Rightarrow h(n_1) \leq$ cost of (every) path from n_1 to goal
$\leq$ cost of optimal path from n_1 to goal

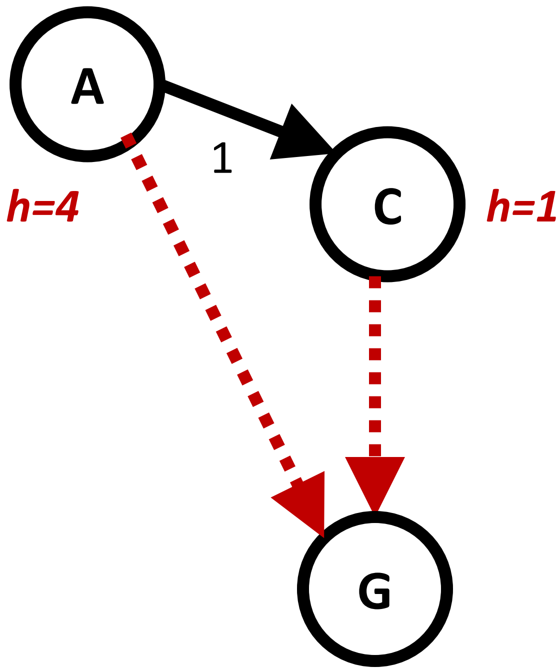
Consistency of Heuristics



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

    h(A) ≤ actual cost from A to G

Consistency of Heuristics



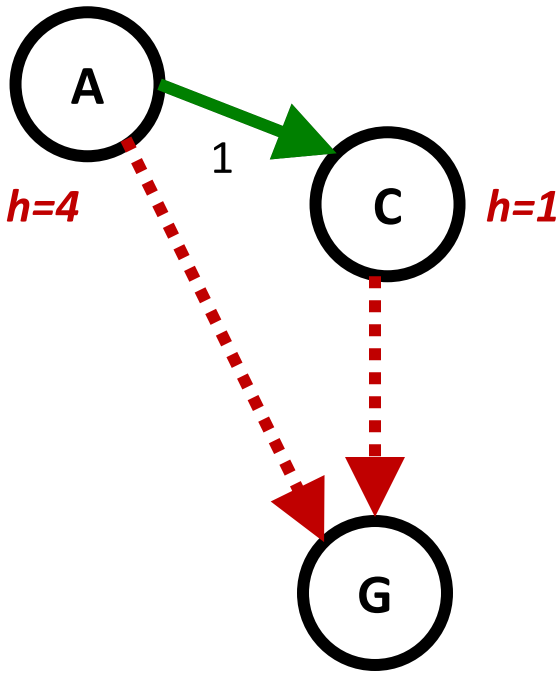
- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal

    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)

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

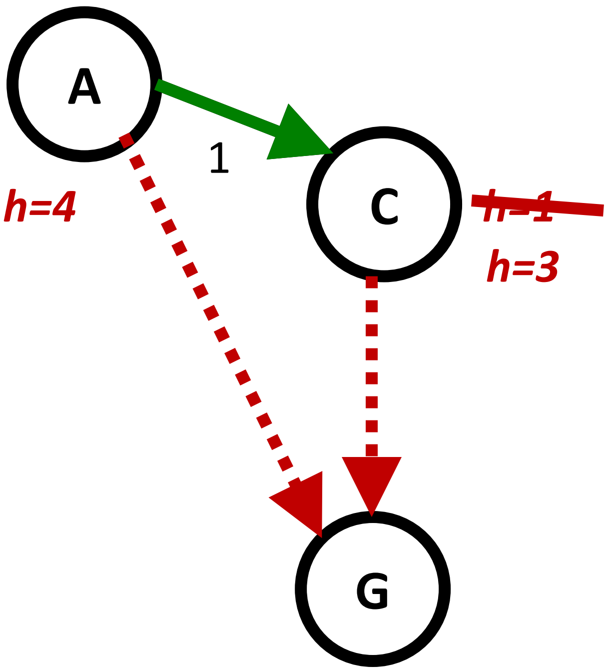
  - Consistency: heuristic “arc” cost ≤ actual cost for each arc

    h(A) – h(C) ≤ cost(A to C)

    or h(A) ≤ c(A,C) + h(C) (triangle inequality)

    Note: h* *necessarily* satisfies triangle inequality

Consistency of Heuristics



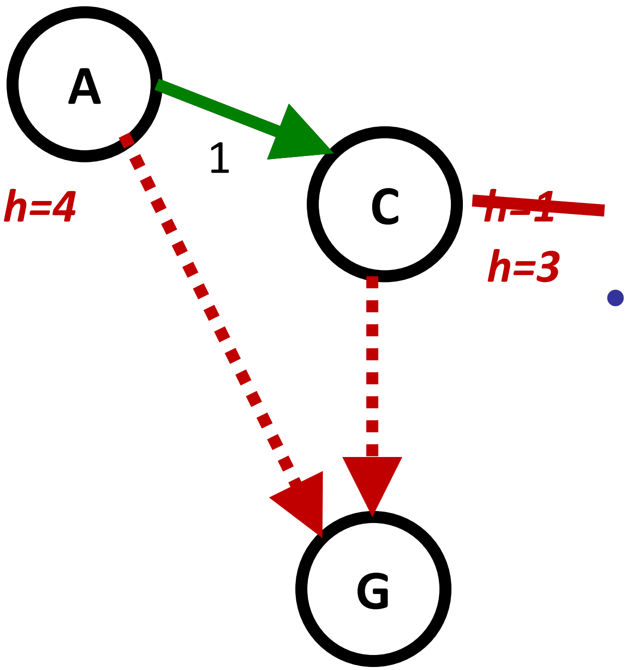
- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal

    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)

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)

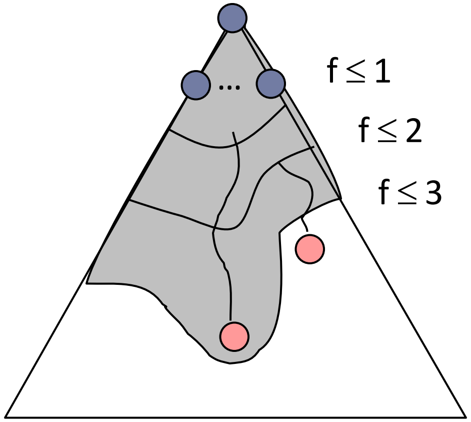
    => g(A) + h(A) ≤ g(A) + c(A,C) + h(C)

    => f(A) ≤ g(C)+h(C)=f(C)

  - A* graph search is optimal

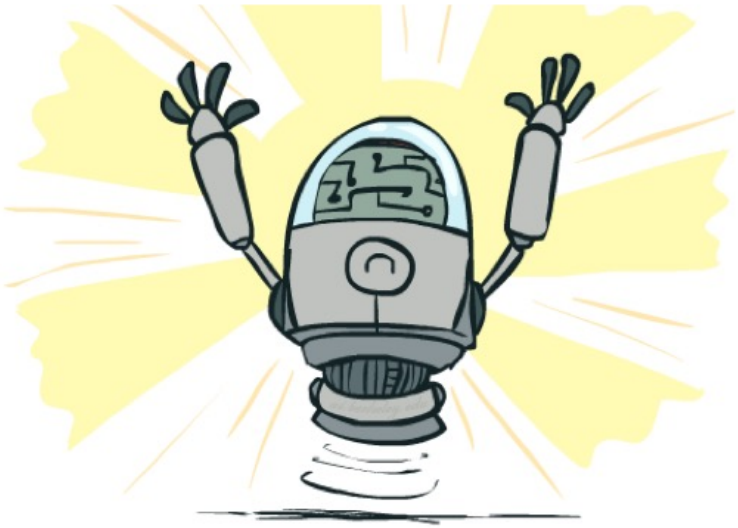
Optimality of A* Graph Search

- Sketch: consider what A* does with a consistent heuristic:
  - Fact 1: 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

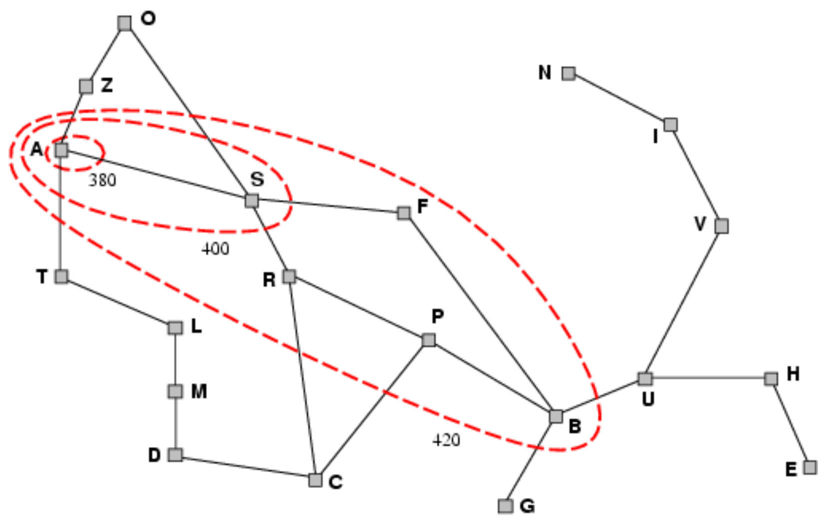
Admissible vs. Consistent (Tree vs. Graph Search)

- Consistent heuristic: When selecting a node for expansion, the path with the lowest cost to that node has been found

- When an admissible heuristic is not consistent, a node will need repeated expansion, every time a new best (so-far) cost is achieved for it.

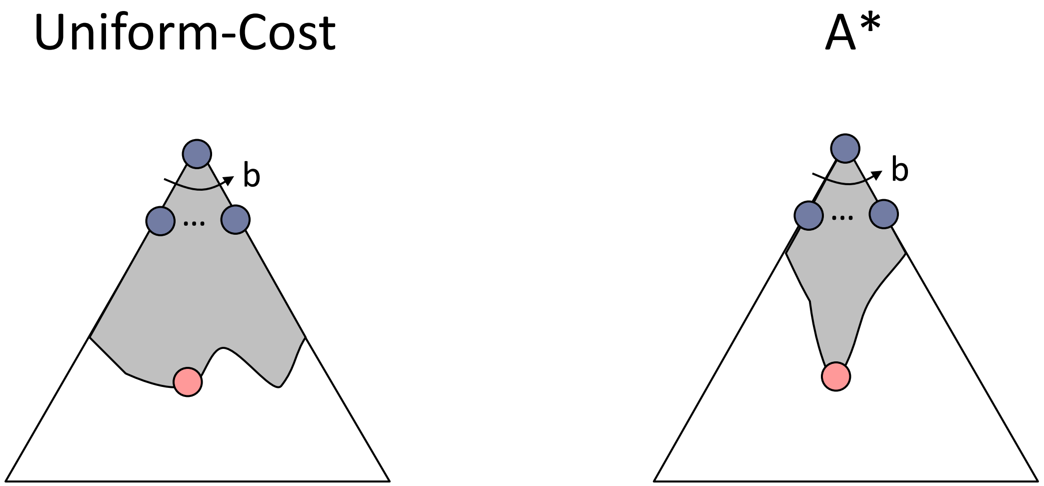
Contours in the state space

- A* (using `GRAPH-SEARCH`) expands nodes in order of increasing f value
- Gradually adds "f-contours" of nodes
  - Contour i has all nodes with $f = f_i$ where $f_i < f_{i+1}$



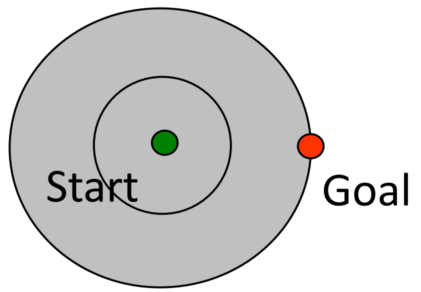
A* expands all nodes with f(n) < C*
A* expands some nodes with f(n) = C* (nodes on the goal contour)
A* expands no nodes with f(n) > C* ⟹ pruning

Properties of A*

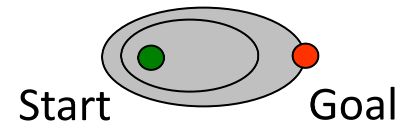


UCS vs A* Contours

- Uniform-cost (A* using $h(n)$ = 0) expands equally in all "directions"



- A* expands mainly toward the goal, but does hedge its bets to ensure optimality
  - More accurate heuristics stretched toward the goal (more narrowly focused around the optimal path)



States are points in 2-D Euclidean space.
g(n)=distance from start
h(n)=estimate of distance from goal

Video of Demo Contours (Empty) -- UCS

Video of Demo Contours (Empty) -- Greedy

Video of Demo Contours (Empty) – A*

Video of Demo Contours (Pacman Small Maze) – A*

Comparison

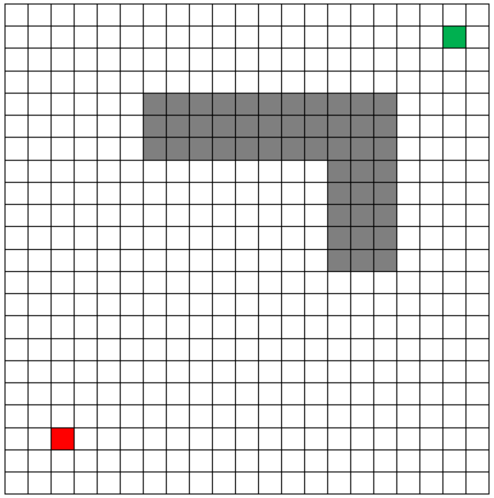
Greedy (h)

Uniform Cost (g)

A* (g+h)

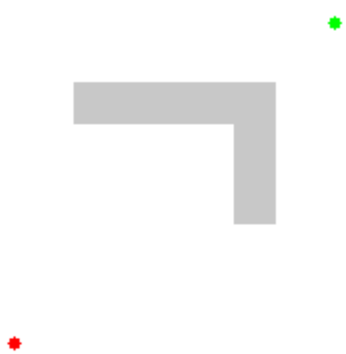
Robot navigation example

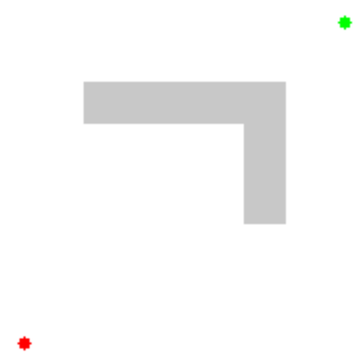
- Initial state? Red cell
- States? Cells on rectangular grid (except to obstacle)
- Actions? Move to one of 8 neighbors (if it is not obstacle)
- Goal test? Green cell
- Path cost? Action cost is the Euclidean length of movement



A* vs. UCS: Robot navigation example

- Heuristic: Euclidean distance to goal
- Expanded nodes: filled circles in red & green
  - Color indicating g value (red: lower, green: higher)
- Frontier: empty nodes with blue boundary
- Nodes falling inside the obstacle are discarded



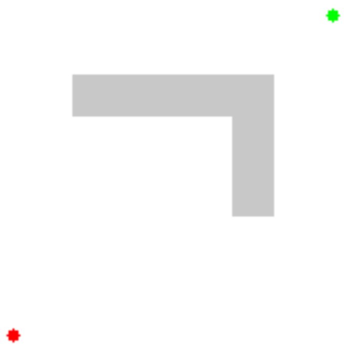


Adopted from: http://en.wikipedia.org/wiki/A*_search_algorithm

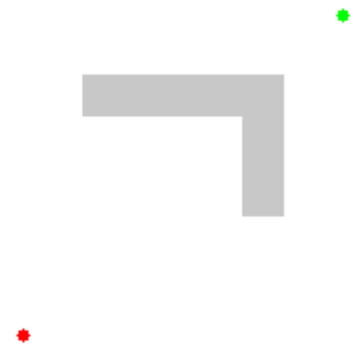
Robot navigation: Admissible heuristic

- Is Manhattan $d_M(x, y) = |x_1 - y_1| + |x_2 - y_2|$ distance an admissible heuristic for previous example?

A*: Inadmissible heuristic



$h = 5 * h\_SLD$



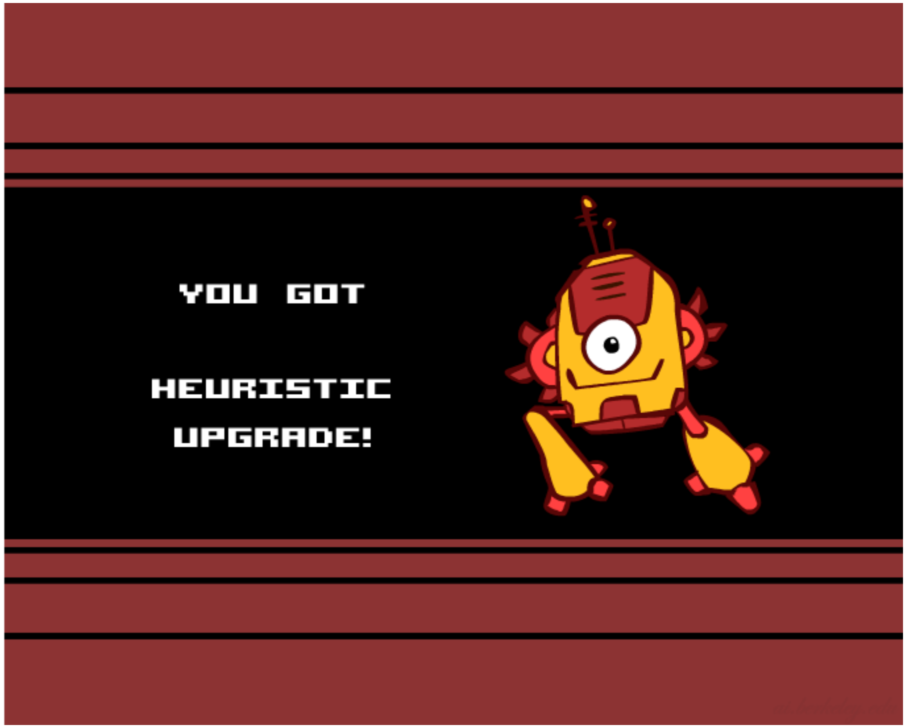
$h = h\_SLD$

Adopted from: http://en.wikipedia.org/wiki/A*_search_algorithm

A*: Summary

- A* orders nodes in the queue by $f(n) = g(n) + h(n)$
  - A* uses both backward costs and (estimates of) forward costs
- A* is optimal for trees/graphs with admissible/consistent heuristics
- Heuristic design is key: often use relaxed problems

Creating Heuristics



Relaxed problem

- Relaxed problem: Problem with fewer restrictions on the actions

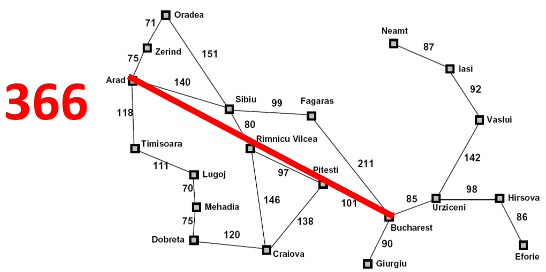
- Optimal solution to the relaxed problem may be computed easily (without search)

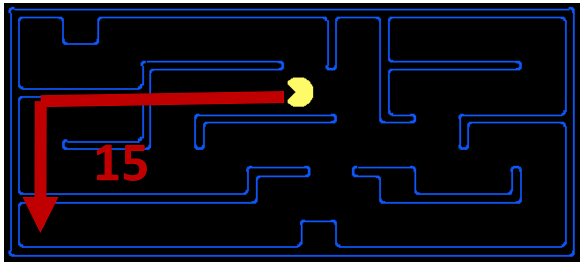
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
  - The optimal solution is the shortest path in the super-graph of the state-space.

Creating Admissible Heuristics

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

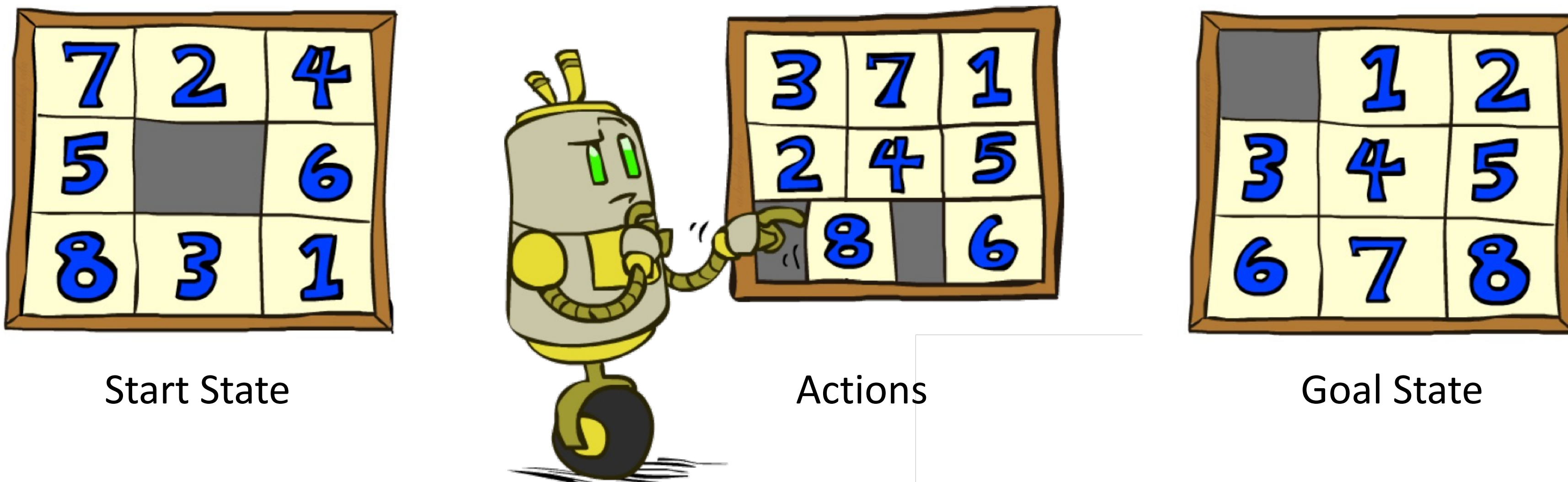
- Often, admissible heuristics are solutions to *relaxed problems,* where new actions are available





- 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
- Inadmissible heuristics are often useful too

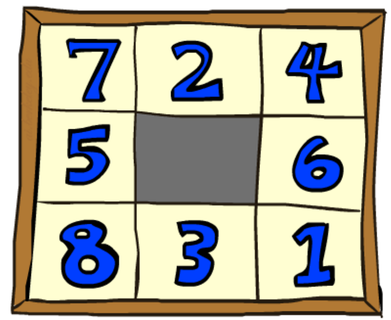
Example: 8 Puzzle



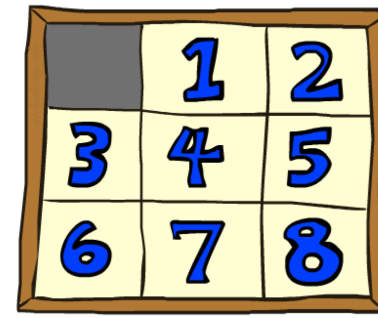
- What are the states?
- How many states?
- What are the actions?
- What are the step costs?

8 Puzzle I

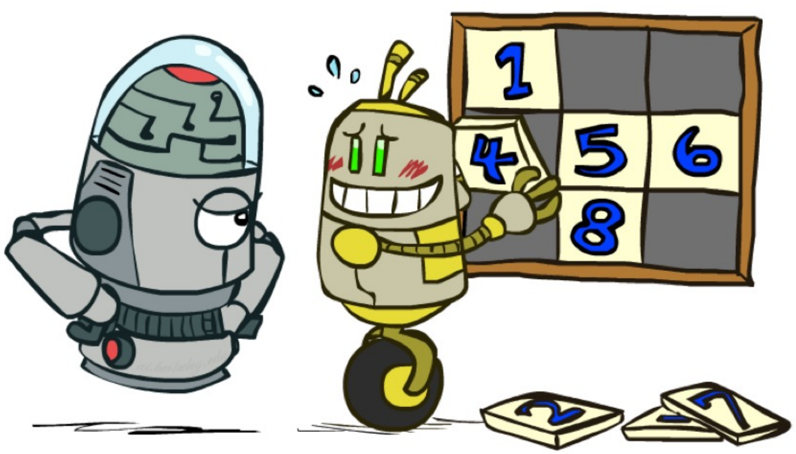
- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8



Start State



Goal State

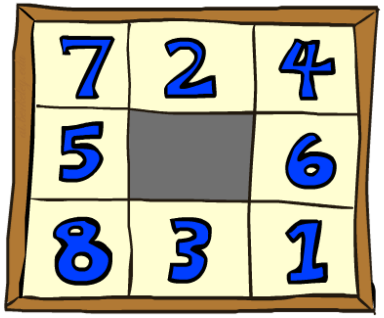


	Average nodes expanded when the optimal path has...		
	...4 steps	...8 steps	...12 steps
UCS	112	6,300	3.6×10^6
A*TILES	13	39	227

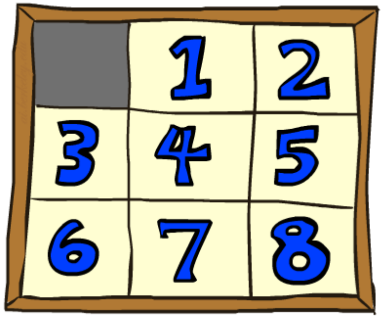
Statistics from Andrew Moore

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= sum of Manhattan distance of tiles from their target position
- 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

Relaxed problem: 8-puzzle

- 8-Puzzle: move a tile from square A to B if A is adjacent (left, right, above, below) to B and B is blank
  - Relaxed problems
    1. can move from A to B if A is adjacent to B (ignore whether or not position is blank)
    2. can move from A to B if B is blank (ignore adjacency)
    3. can move from A to B (ignore both conditions)
- Admissible heuristics for original problem ($h_1(n)$ and $h_2(n)$) are optimal path costs for relaxed problems
  - First case: a tile can move to any adjacent square $\Rightarrow h_2(n)$
  - Third case: a tile can move anywhere $\Rightarrow h_1(n)$

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!

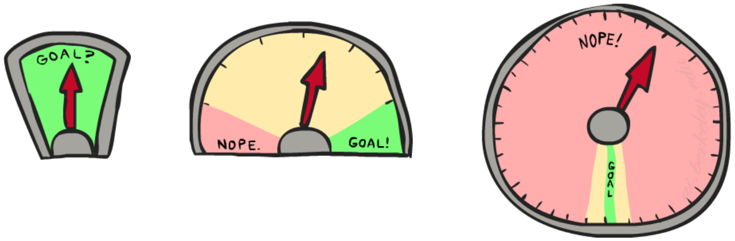
Heuristic quality

- If $\forall n, h_2(n) \geq h_1(n)$ (both admissible)
 then h_2 dominates h_1 and it is better for search

- Surely expanded nodes: $f(n) < C^* \Rightarrow h(n) < C^* - g(n)$
  - If $h_2(n) \geq h_1(n)$ then every node expanded for h_2 will also be surely expanded with h_1 (h_1 may also causes some more node expansion)

8 Puzzle: heuristic

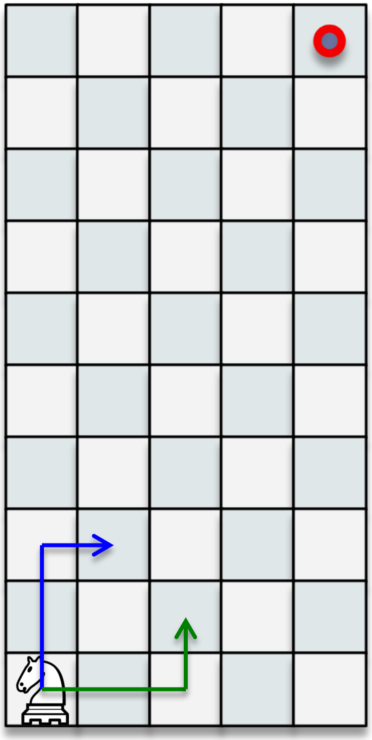
- How about using the *actual cost* as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?



- 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

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!



A* Applications

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

Memory bounded methods

- 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 IDS, except it uses an *f*-limit instead of a depth limit
  - 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
  - SMA* uses *all available memory* for the queue, minimizing thrashing