Search

Lecture Attendance: http://bit.ly/3GEMoKZ

Search

Lecture Attendance: http://bit.ly/3GEMoKZ

State space graph.

Search

Lecture Attendance: http://bit.ly/3GEMoKZ

State space graph.

Given a task, model of world appropriate to task.

Search

Lecture Attendance: http://bit.ly/3GEMoKZ

State space graph.

Given a task, model of world appropriate to task.

Search Tree.

Search

Lecture Attendance: http://bit.ly/3GEMoKZ

State space graph.

Given a task, model of world appropriate to task.

Search Tree.

Representation of all plans

Search

Lecture Attendance: http://bit.ly/3GEMoKZ

State space graph.

Given a task, model of world appropriate to task.

Search Tree.

Representation of all plans
(also paths through state space graph.)

Search

Lecture Attendance: http://bit.ly/3GEMoKZ

State space graph.

Given a task, model of world appropriate to task.

Search Tree.

Representation of all plans
(also paths through state space graph.)

Uninformed Search.

Search

Lecture Attendance: http://bit.ly/3GEMoKZ

State space graph.

Given a task, model of world appropriate to task.

Search Tree.

Representation of all plans
(also paths through state space graph.)

Uninformed Search.

Depth First Search.

Search

Lecture Attendance: http://bit.ly/3GEMoKZ

State space graph.

Given a task, model of world appropriate to task.

Search Tree.

Representation of all plans
(also paths through state space graph.)

Uninformed Search.

Depth First Search.

Good space bound, “leftmost” exploration seems bad.

Search

Lecture Attendance: http://bit.ly/3GEMoKZ

State space graph.

Given a task, model of world appropriate to task.

Search Tree.

Representation of all plans
(also paths through state space graph.)

Uninformed Search.

Depth First Search.

Good space bound, “leftmost” exploration seems bad.

Breadth First Search.

Search

Lecture Attendance: http://bit.ly/3GEMoKZ

State space graph.

Given a task, model of world appropriate to task.

Search Tree.

Representation of all plans
(also paths through state space graph.)

Uninformed Search.

Depth First Search.

Good space bound, “leftmost” exploration seems bad.

Breadth First Search.

Bad space bound, finds plan with fewest number of actions.

Search

Lecture Attendance: http://bit.ly/3GEMoKZ

State space graph.

Given a task, model of world appropriate to task.

Search Tree.

Representation of all plans
(also paths through state space graph.)

Uninformed Search.

Depth First Search.

Good space bound, “leftmost” exploration seems bad.

Breadth First Search.

Bad space bound, finds plan with fewest number of actions.

Uniform Cost Search.

Search

Lecture Attendance: http://bit.ly/3GEMoKZ

State space graph.

Given a task, model of world appropriate to task.

Search Tree.

Representation of all plans
(also paths through state space graph.)

Uninformed Search.

Depth First Search.

Good space bound, “leftmost” exploration seems bad.

Breadth First Search.

Bad space bound, finds plan with fewest number of actions.

Uniform Cost Search.

Bad space bounds, finds optimal plan.

Search

Lecture Attendance: http://bit.ly/3GEMoKZ

State space graph.

Given a task, model of world appropriate to task.

Search Tree.

Representation of all plans
(also paths through state space graph.)

Uninformed Search.

Depth First Search.

Good space bound, “leftmost” exploration seems bad.

Breadth First Search.

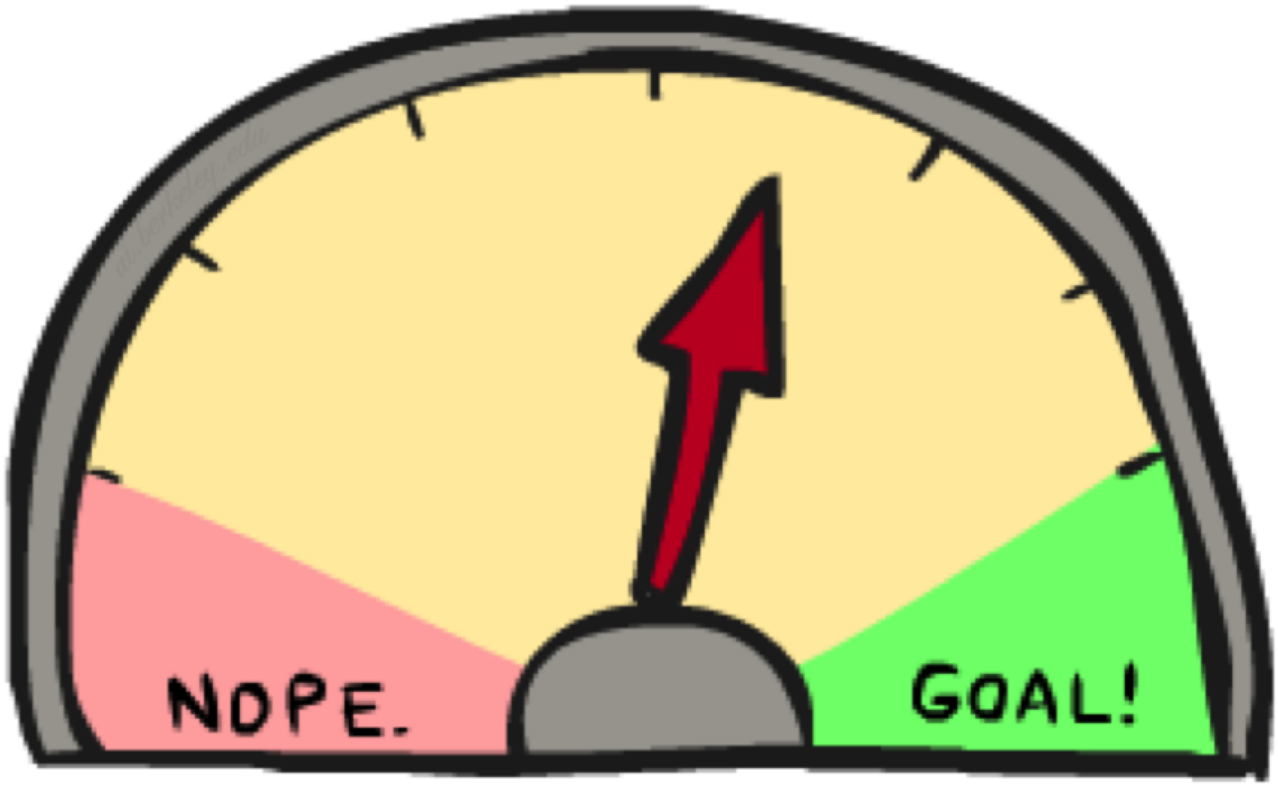
Bad space bound, finds plan with fewest number of actions.

Uniform Cost Search.

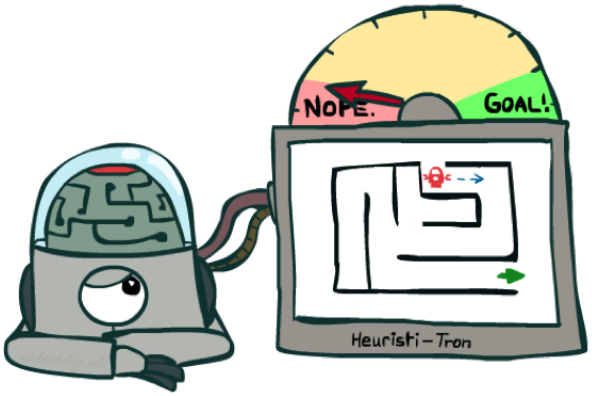
Bad space bounds, finds optimal plan.

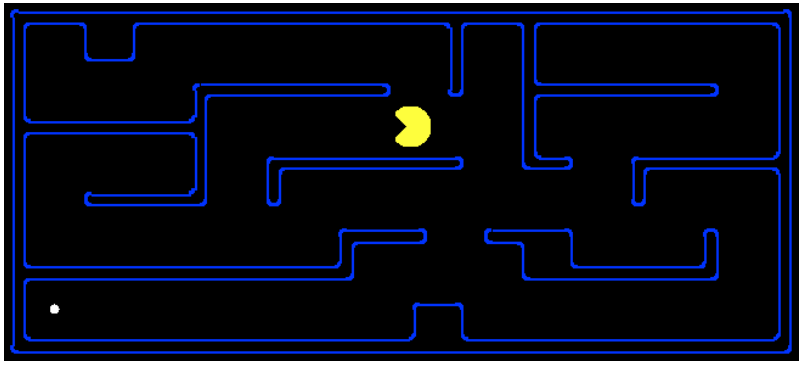
Demo: L3D1

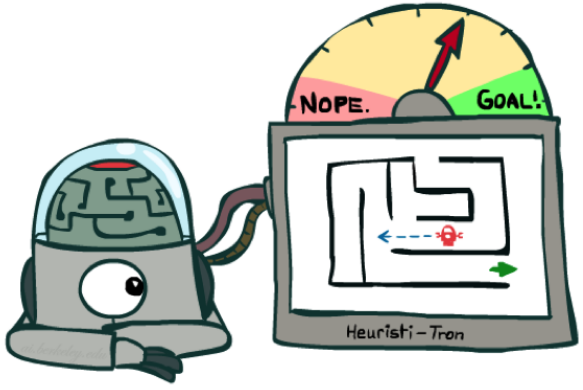
Informed Search



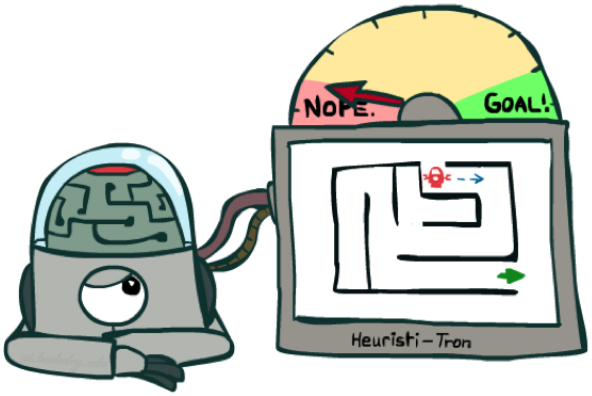
Search Heuristics

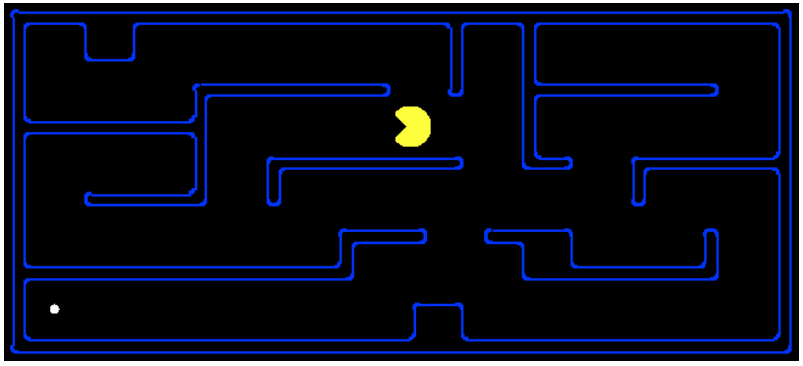


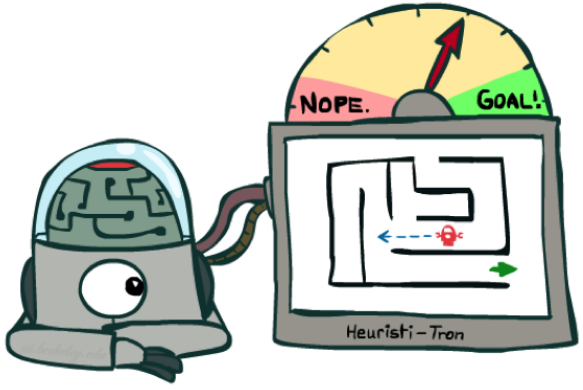




Search Heuristics



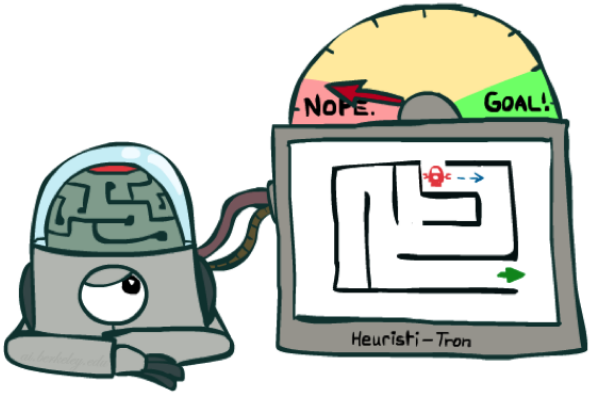


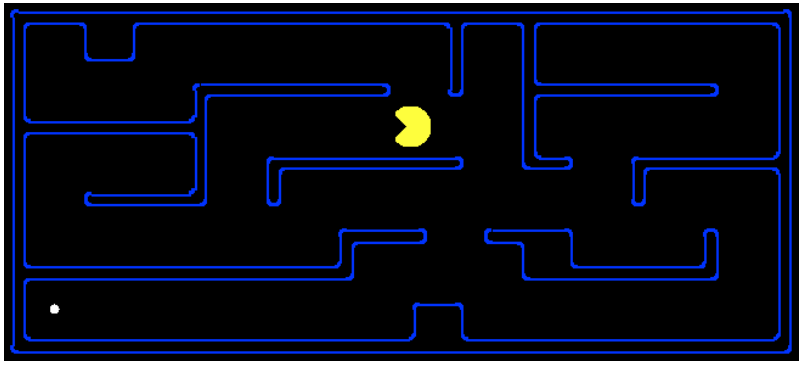


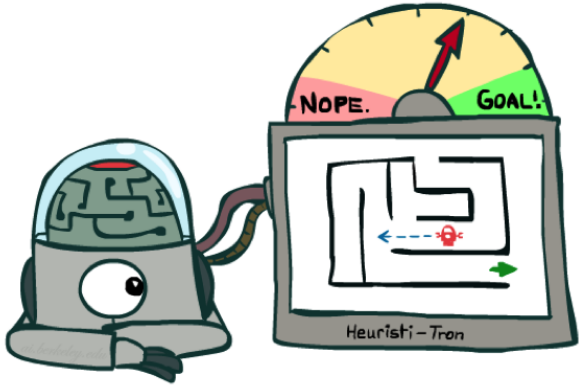
A heuristic is:

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem

Search Heuristics



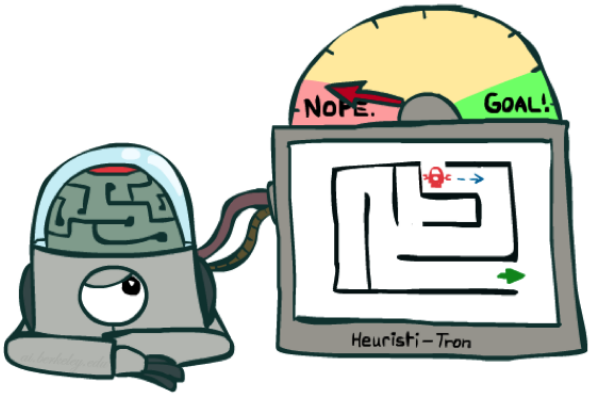


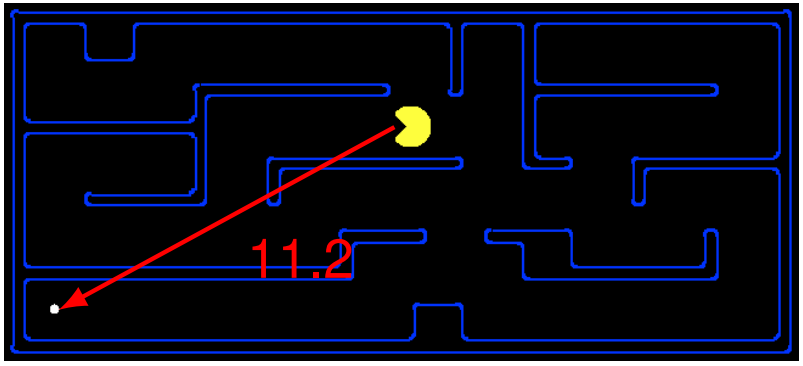


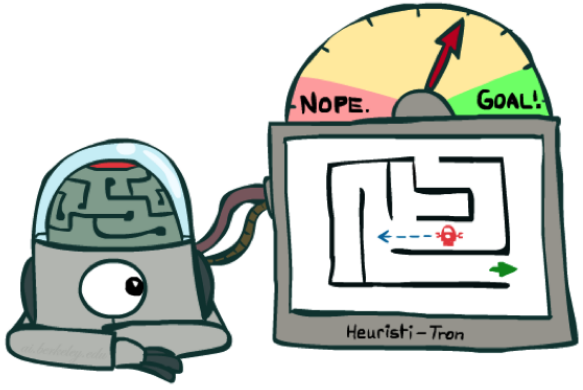
A heuristic is:

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- Examples:

Search Heuristics



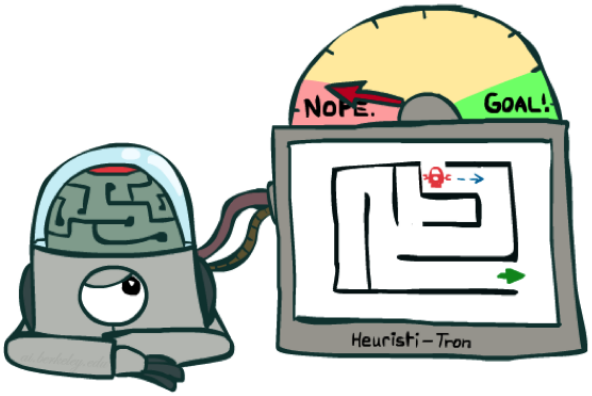


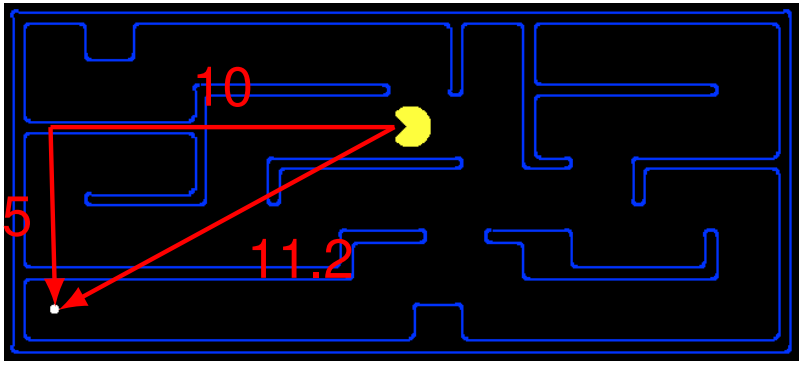


A heuristic is:

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

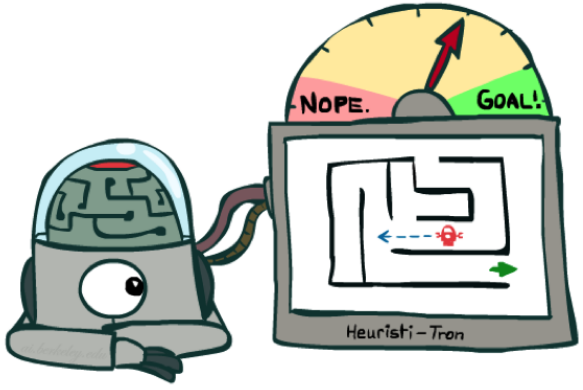
Search Heuristics



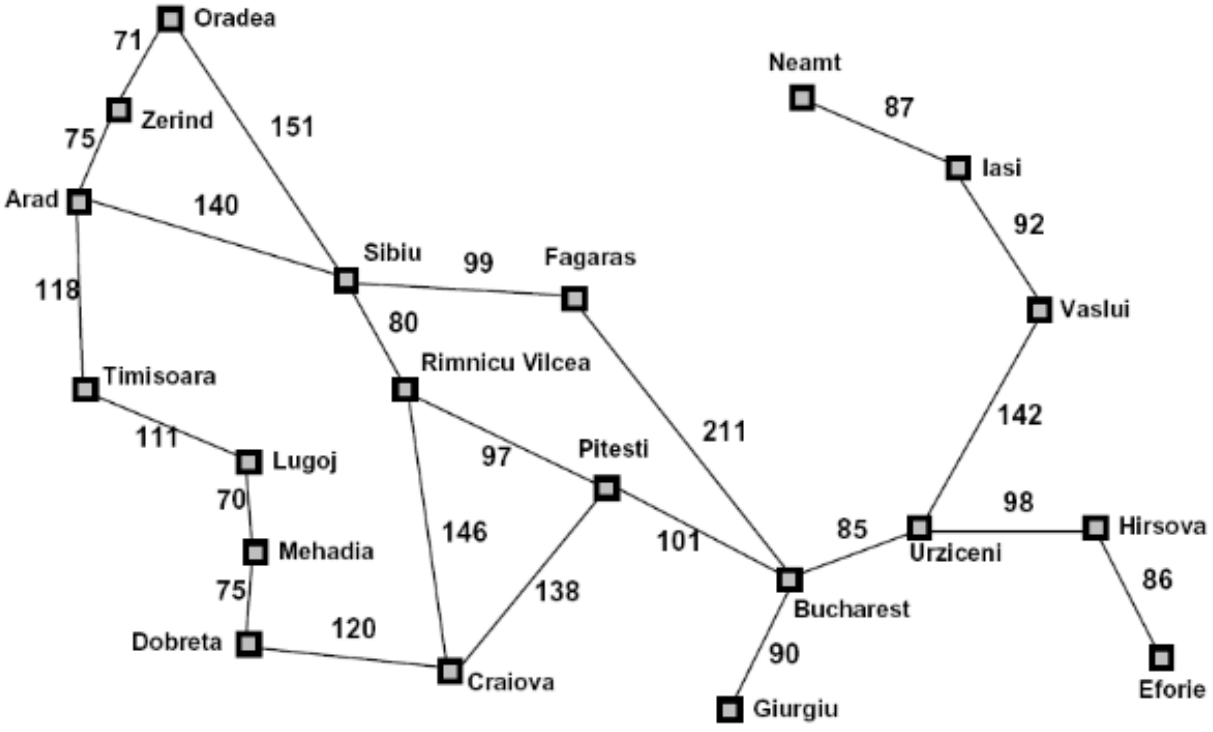


A heuristic is:

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

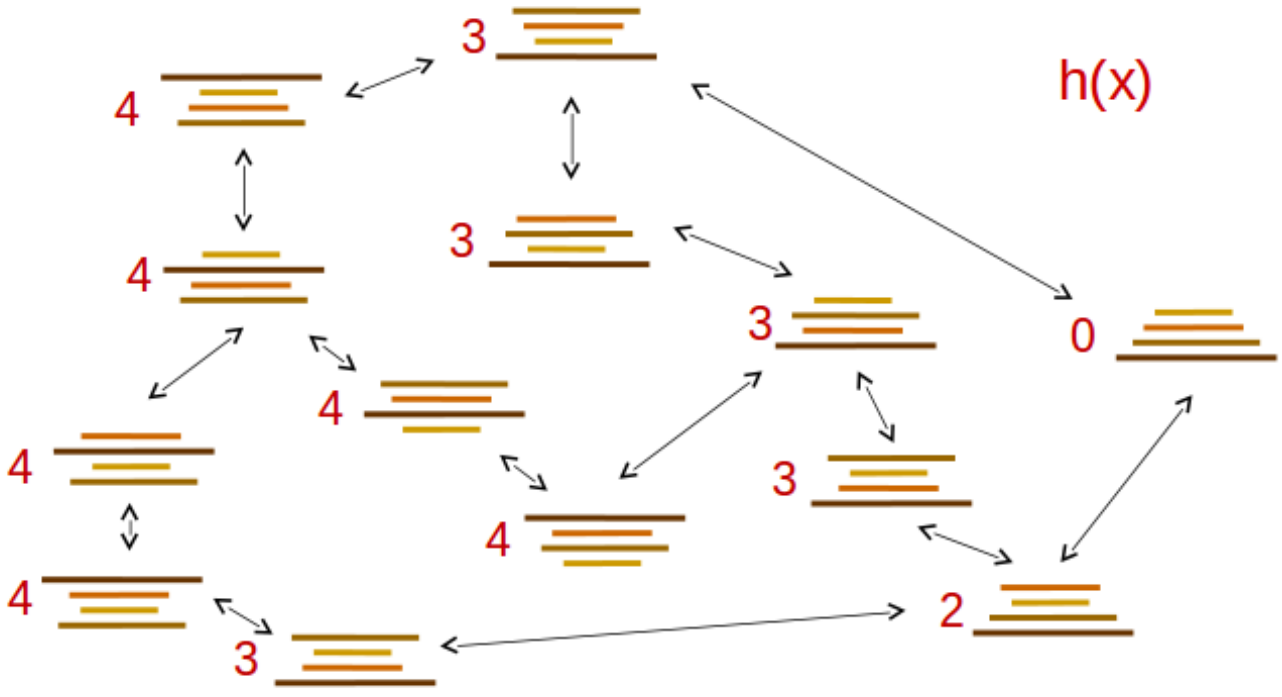


Example: Heuristic Function



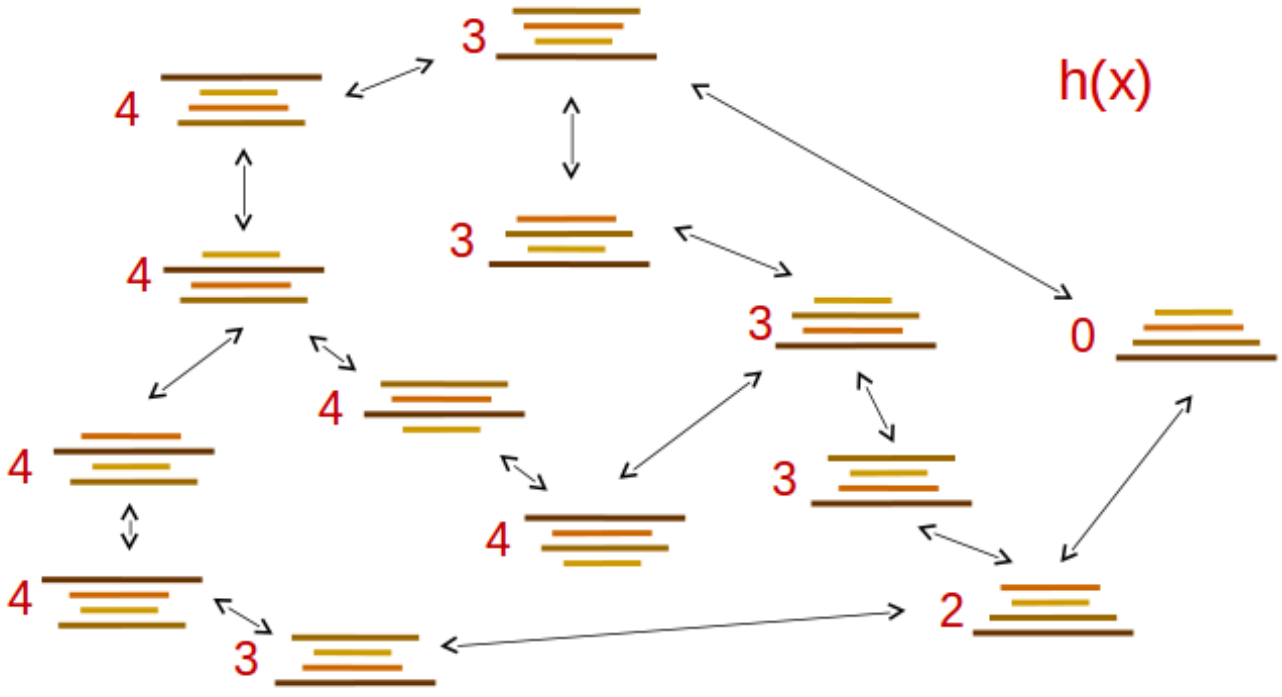
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Example: Heuristic Function



Heuristic:

Example: Heuristic Function



Heuristic: the number of the largest pancake that is still out of place

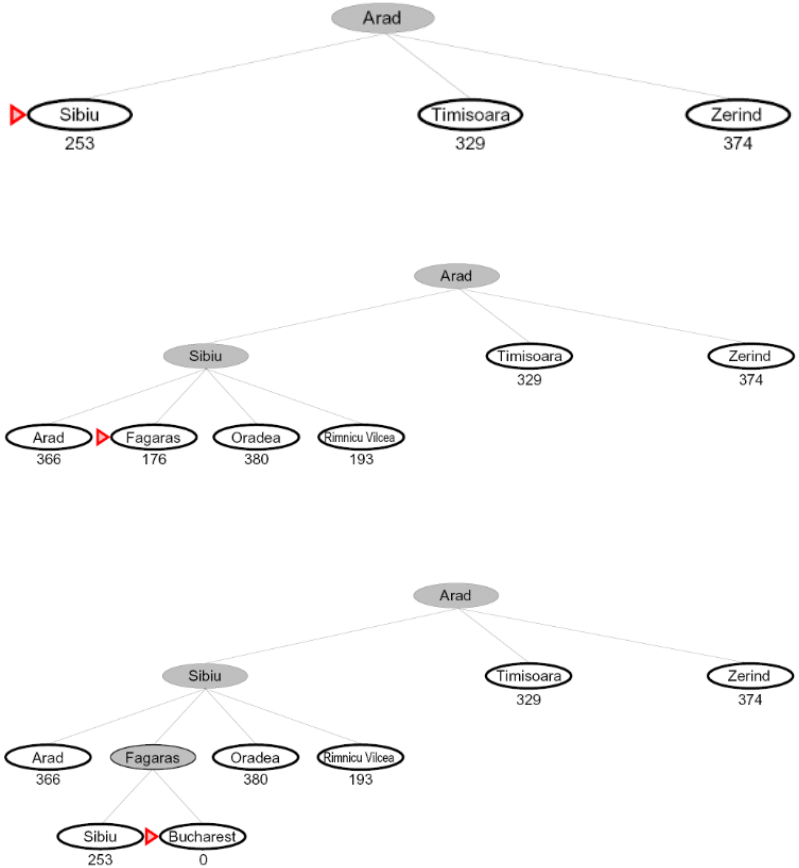
Greedy Search

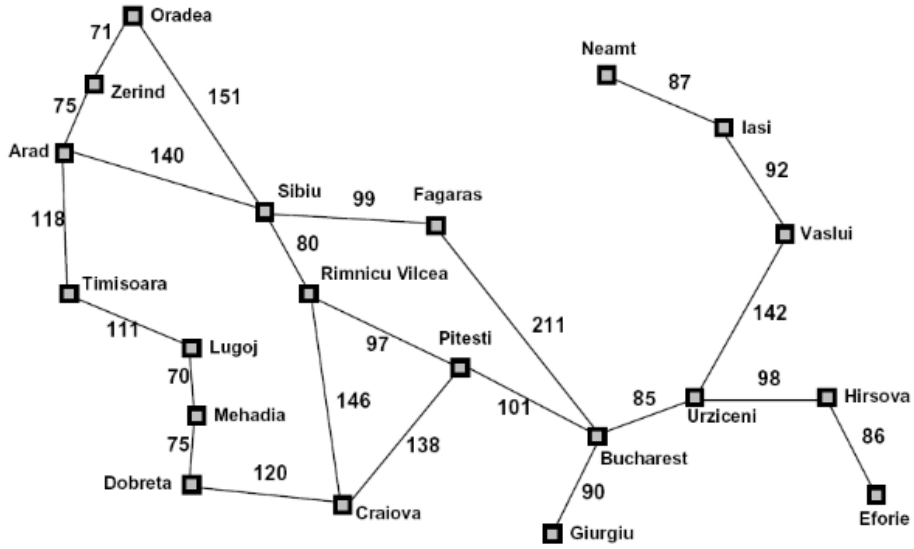
Greedy Search

Expand the node that seems closest

Greedy Search

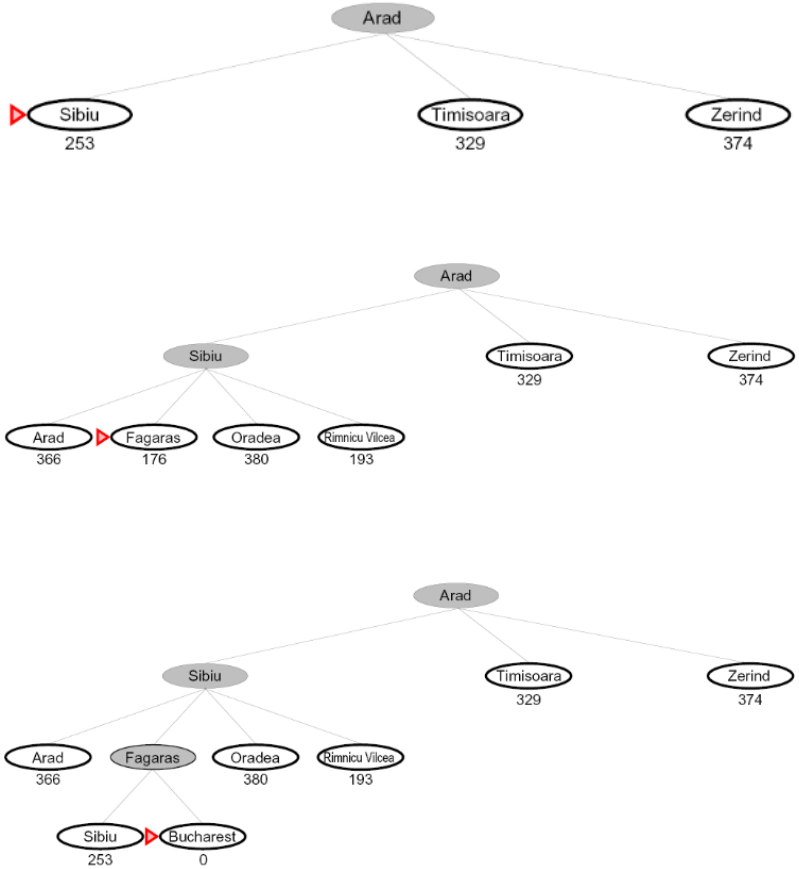
Expand the node that seems closest to the goal?

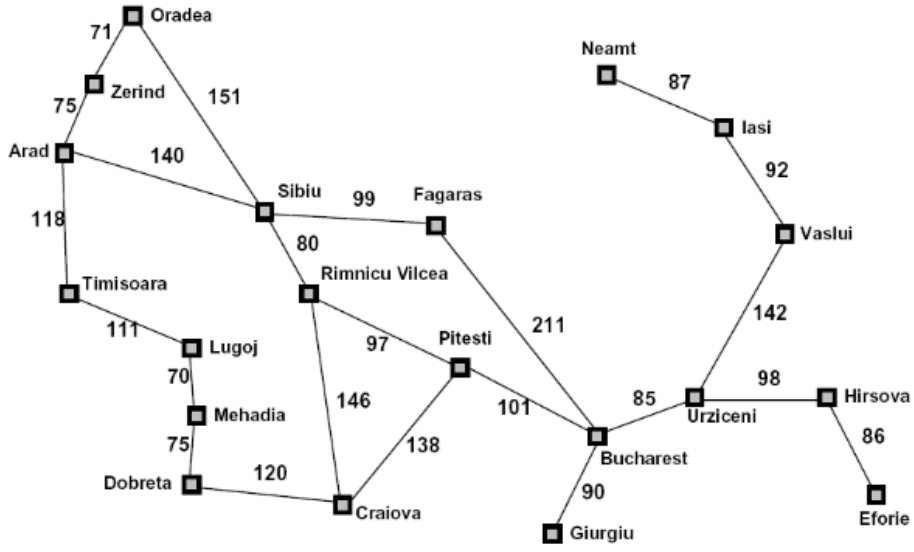




Greedy Search

Expand the node that seems closest to the goal?

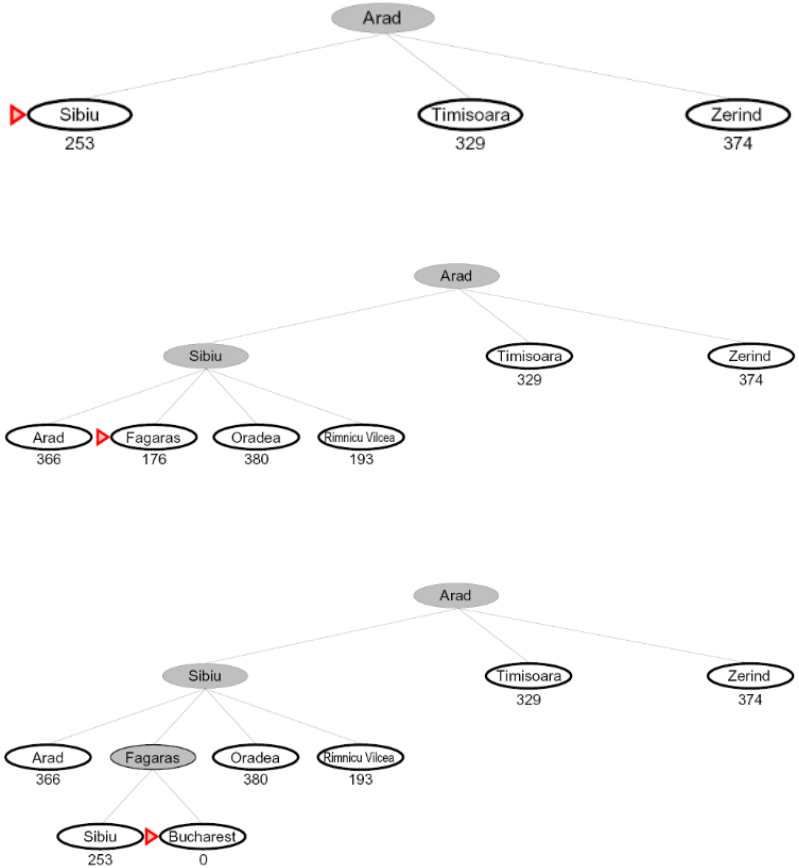


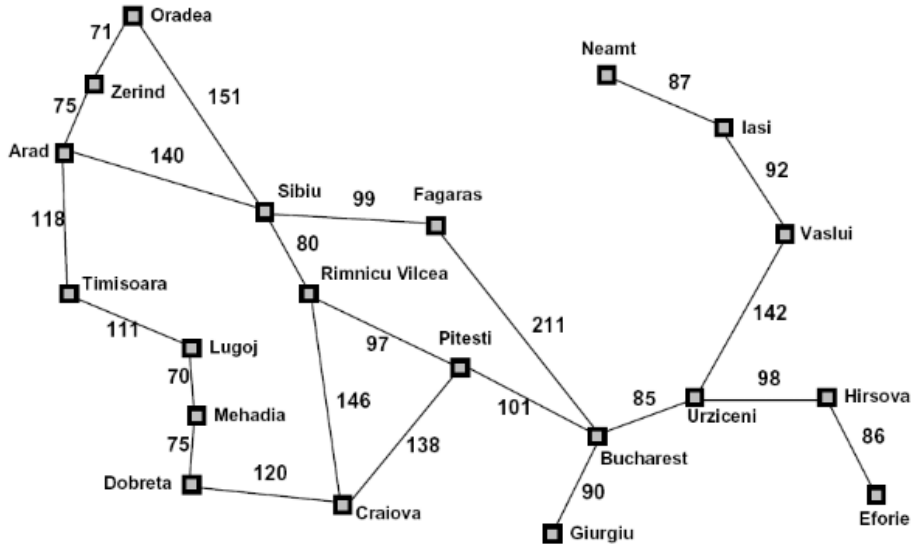


What can go wrong?

Greedy Search

Expand the node that seems closest to the goal?



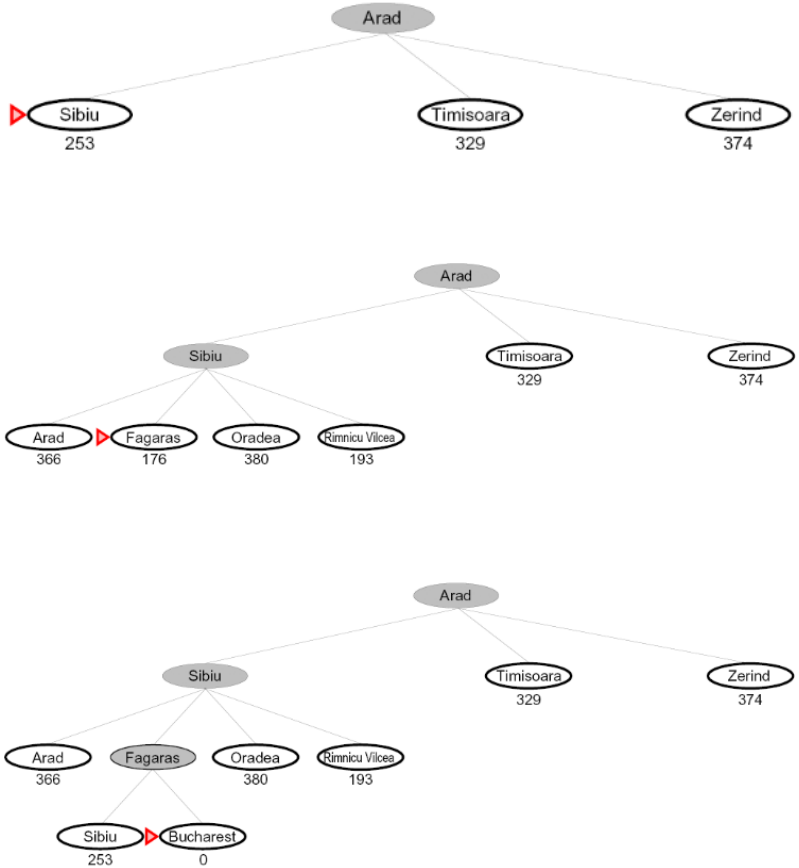


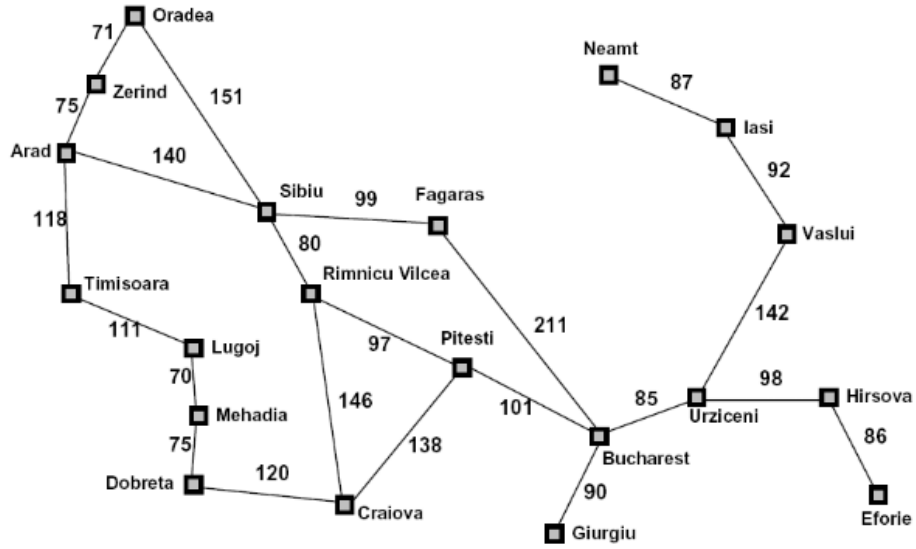
What can go wrong?

Greedy: 140 + 99 + 211 =

Greedy Search

Expand the node that seems closest to the goal?





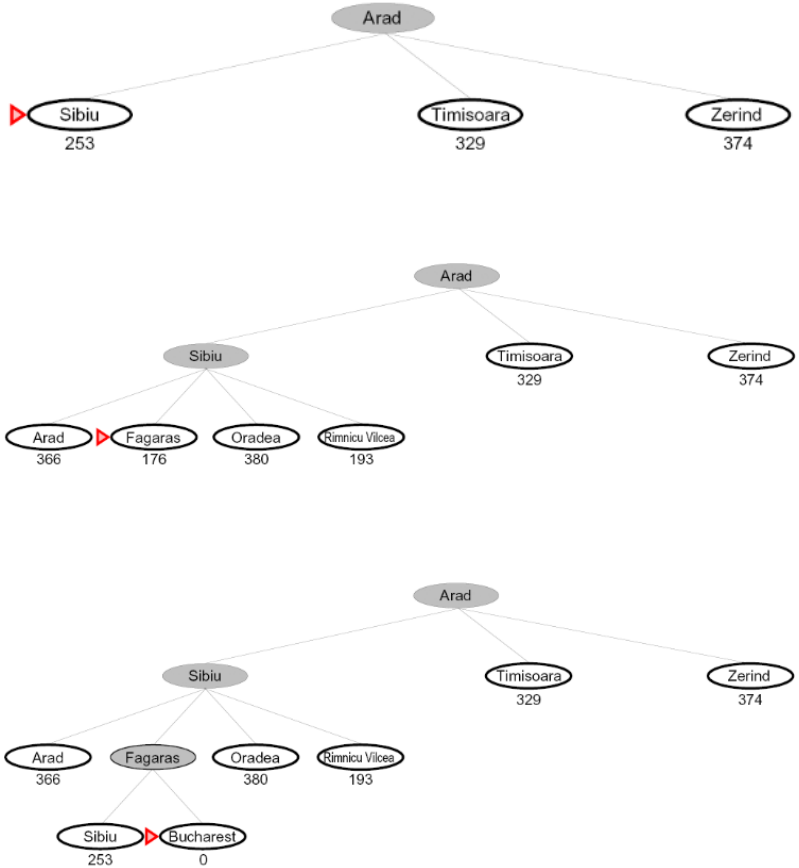
What can go wrong?

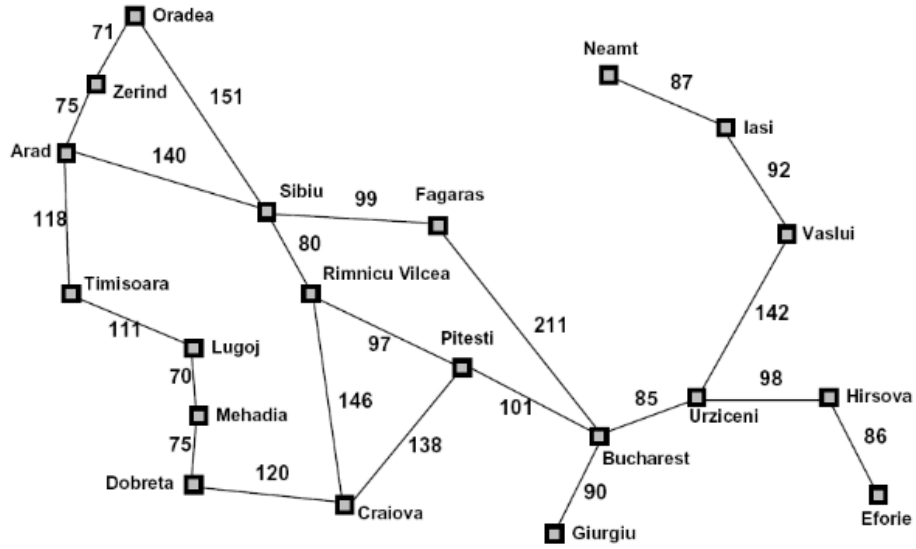
Greedy: 140 + 99 + 211 = 450

Better: 140+80+97+101 =

Greedy Search

Expand the node that seems closest to the goal?





What can go wrong?

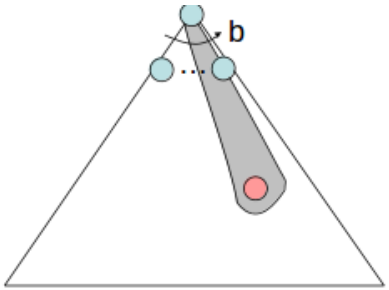
Greedy: 140 + 99 + 211 = 450

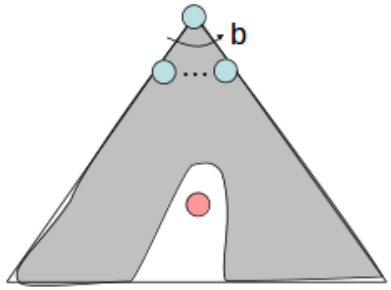
Better: 140+80+97+101 = 418

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



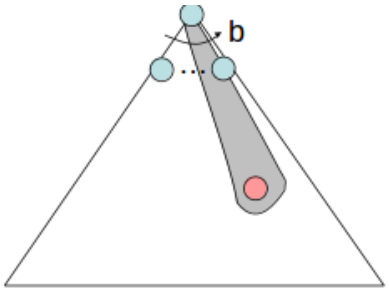


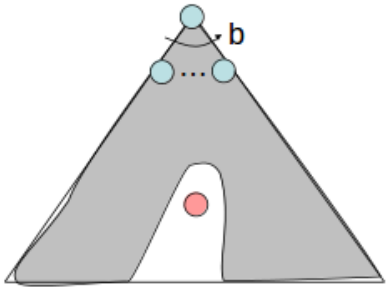
Wildest dream:

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





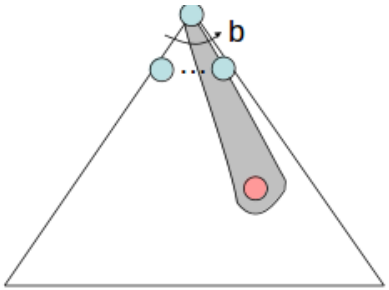
Wildest dream:

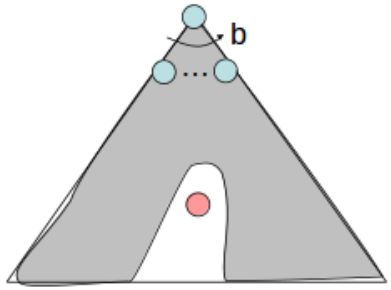
Perfect heuristic.

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





Wildest dream:

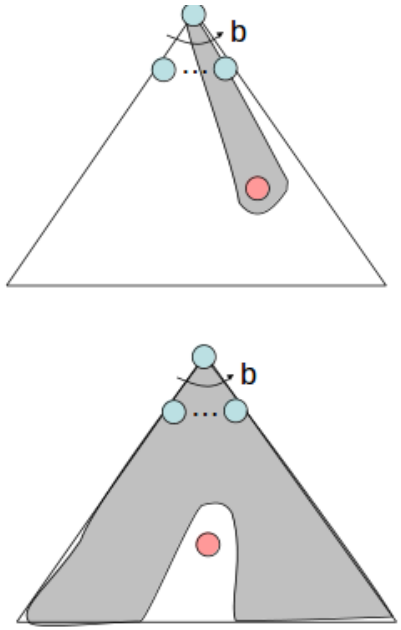
Perfect heuristic. A common case:

Best-first takes you straight to the (wrong) goal

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



Wildest dream:

Perfect heuristic. A common case:

Best-first takes you straight to the (wrong) goal

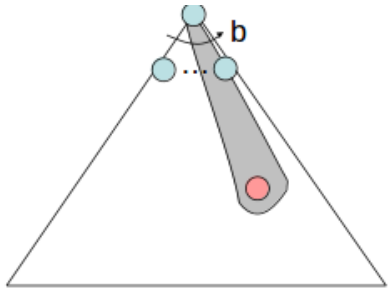
Worst-case:

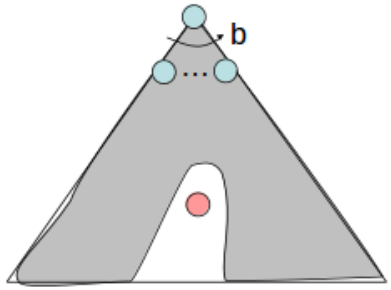
Like a badly-guided DFS

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





Wildest dream:

Perfect heuristic. A common case:

Best-first takes you straight to the (wrong) goal

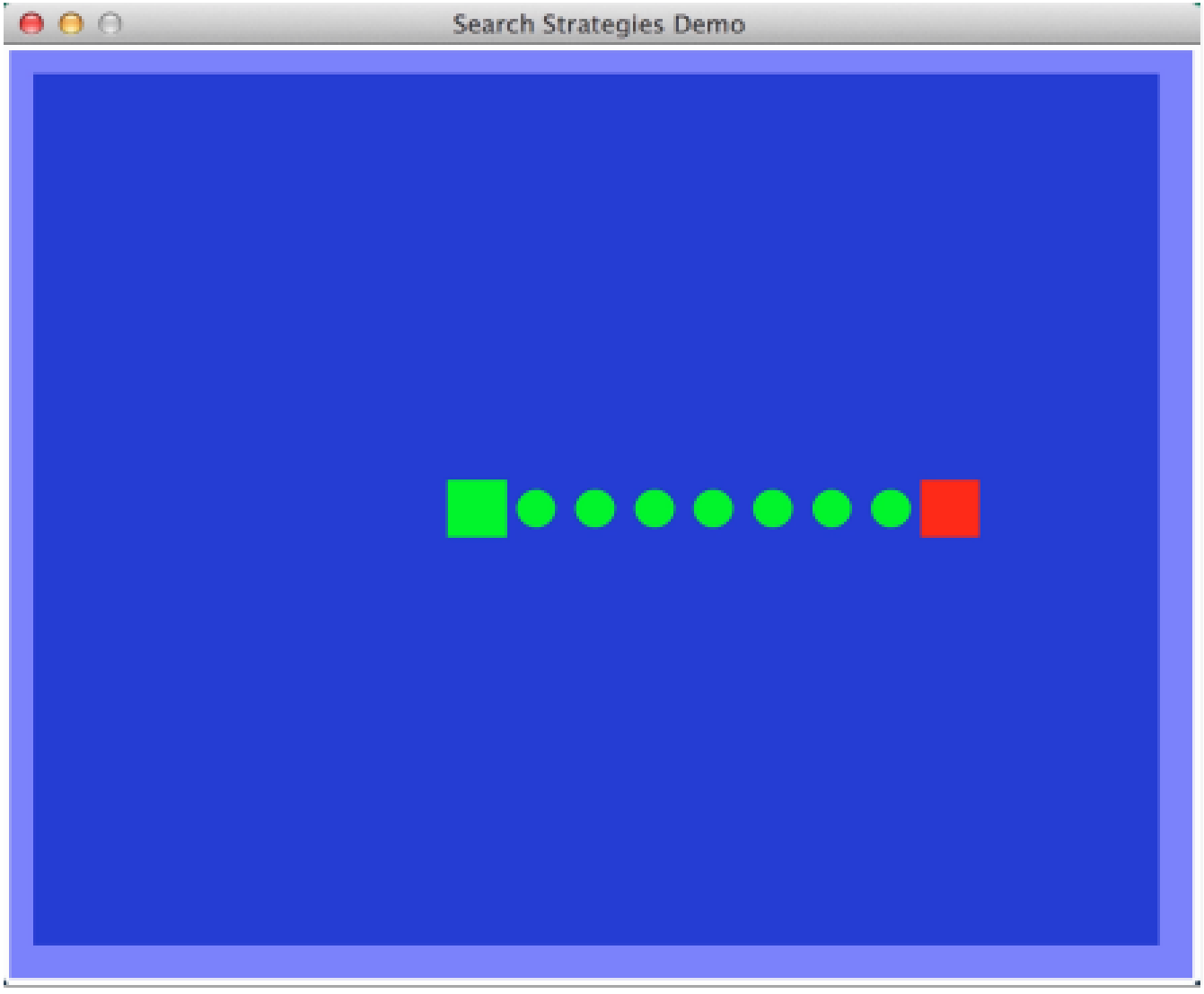
Worst-case:

Like a badly-guided DFS

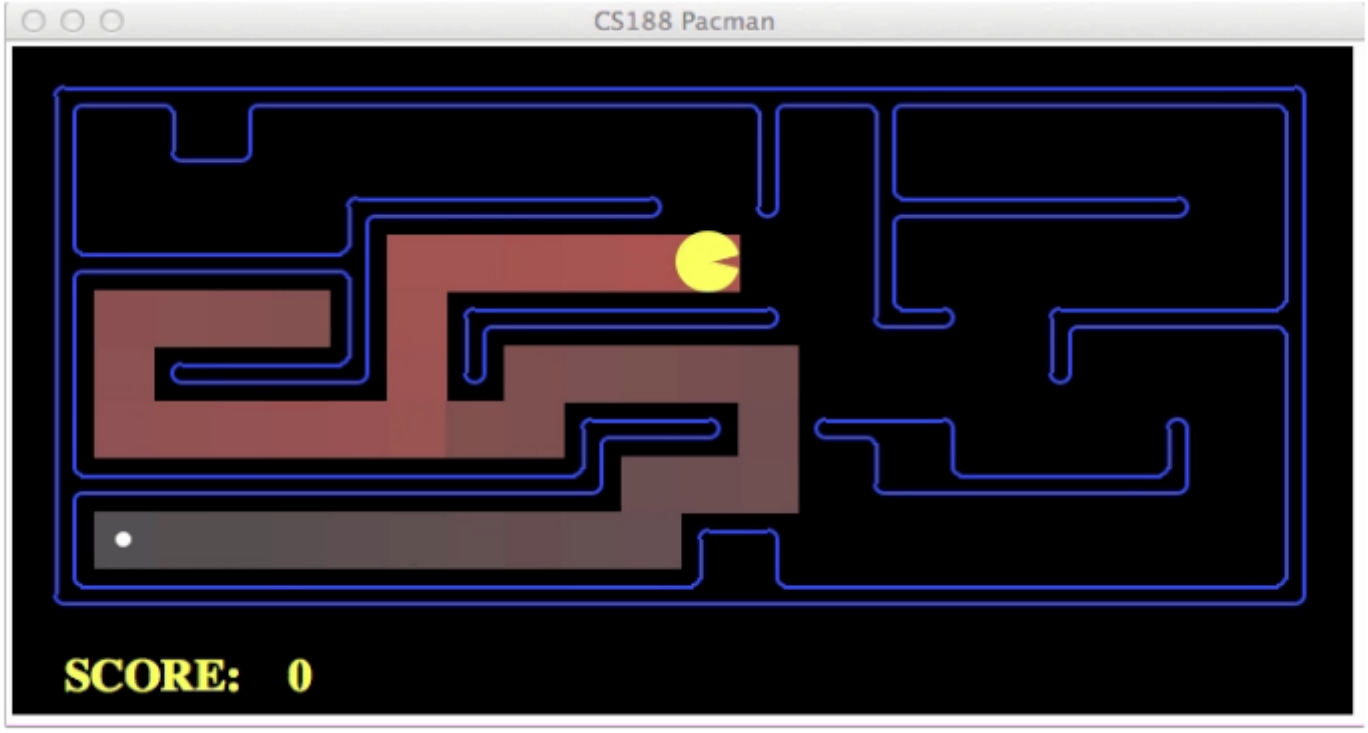
[Democontours greedy empty (L3D3)]

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

Video of Demo Contours Greedy (Empty)



Video of Demo Contours Greedy (pacman/small)



A* Search

A* Search

UCS

Greedy

A* Search

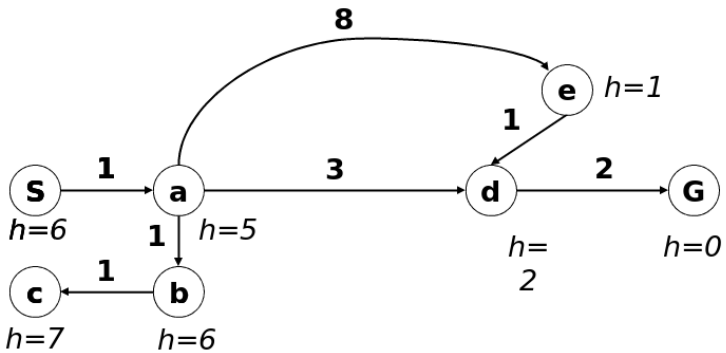
UCS

Greedy

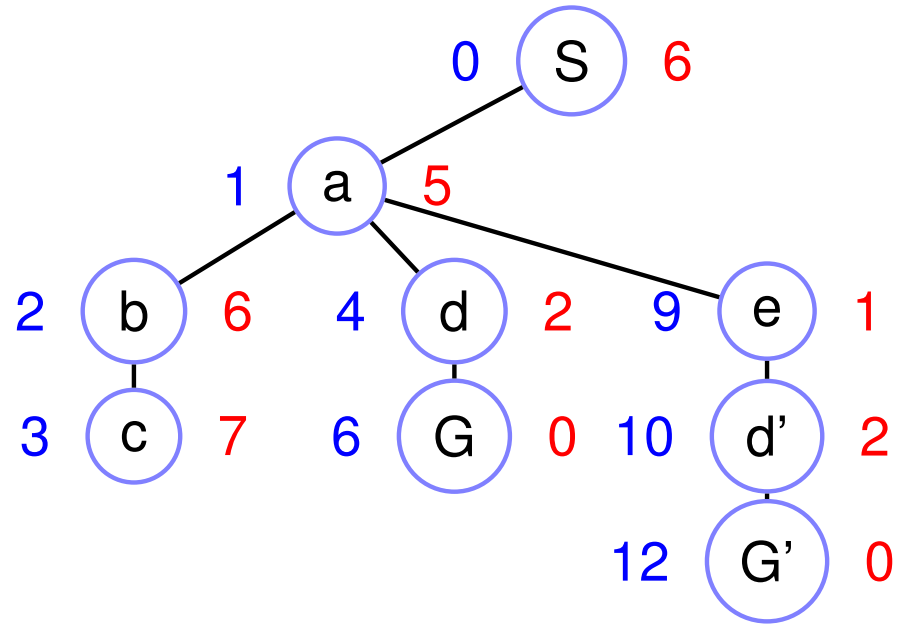
A*

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

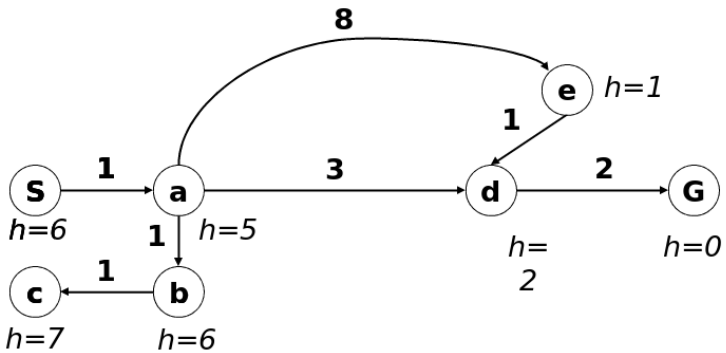


Blue $g(n)$, Red $h(n)$.

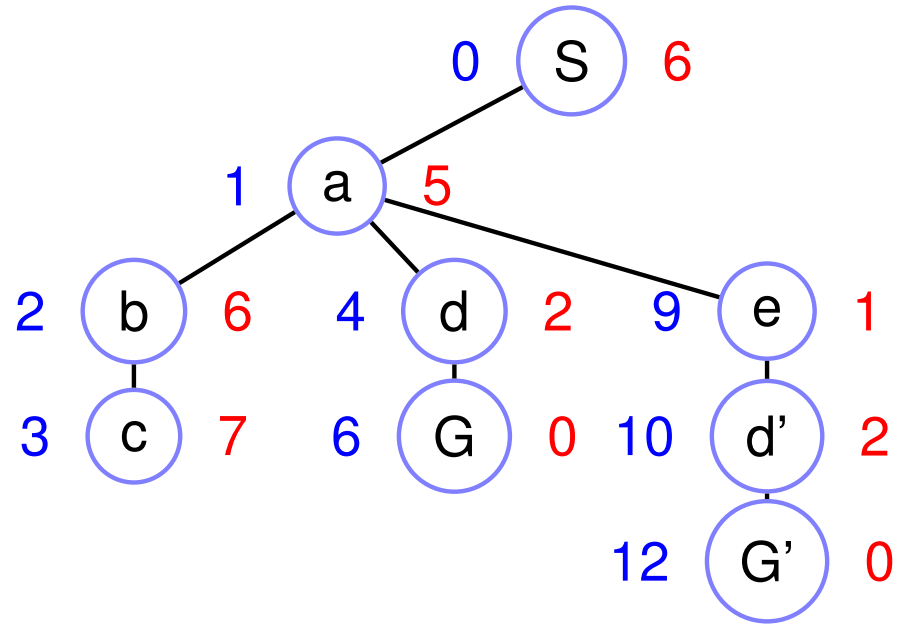


Combining UCS and Greedy

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

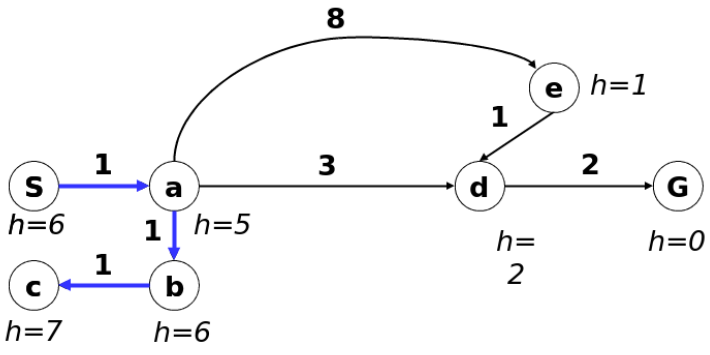


Blue $g(n)$, Red $h(n)$.

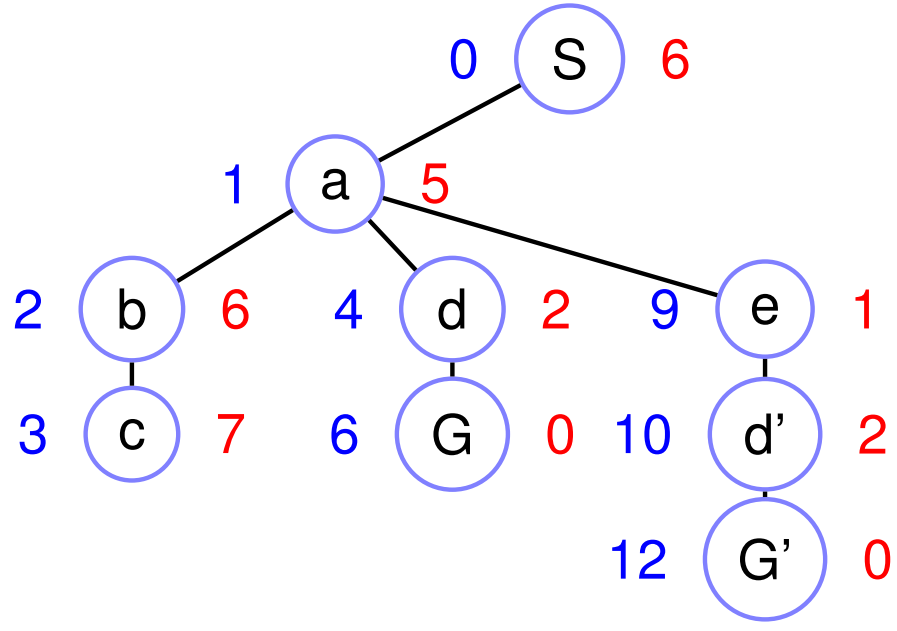


Combining UCS and Greedy

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



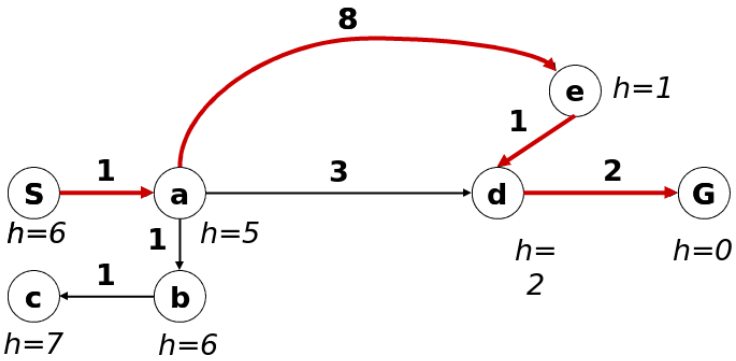
Blue $g(n)$, Red $h(n)$.



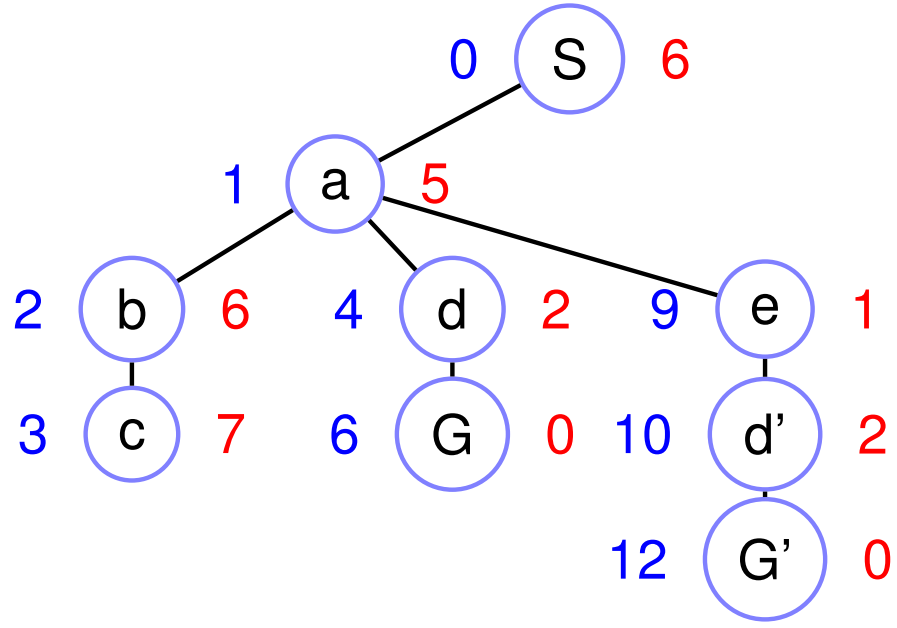
Uniform Cost Search.

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



Blue $g(n)$, Red $h(n)$.

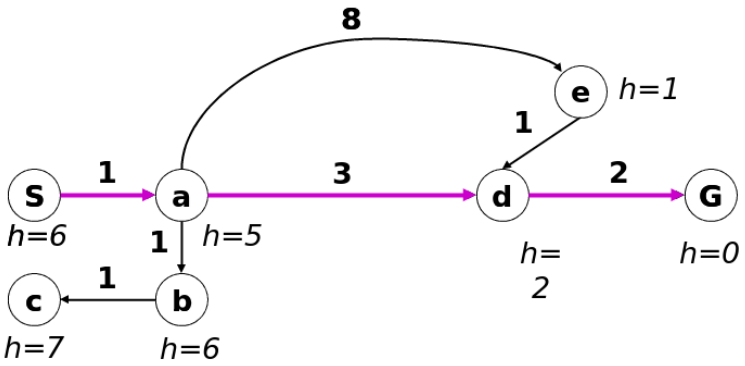


Uniform Cost Search.

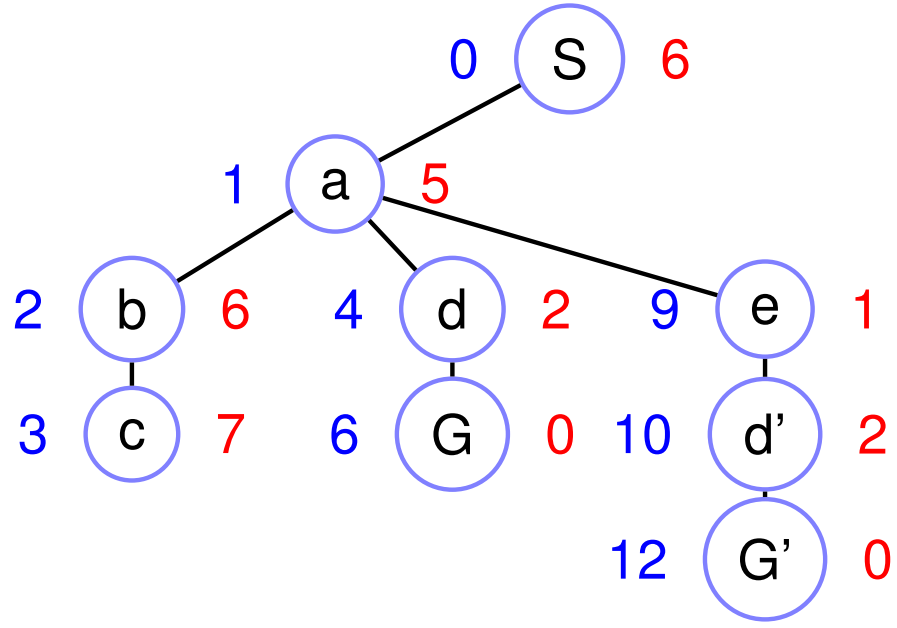
Greedy

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



Blue $g(n)$, Red $h(n)$.



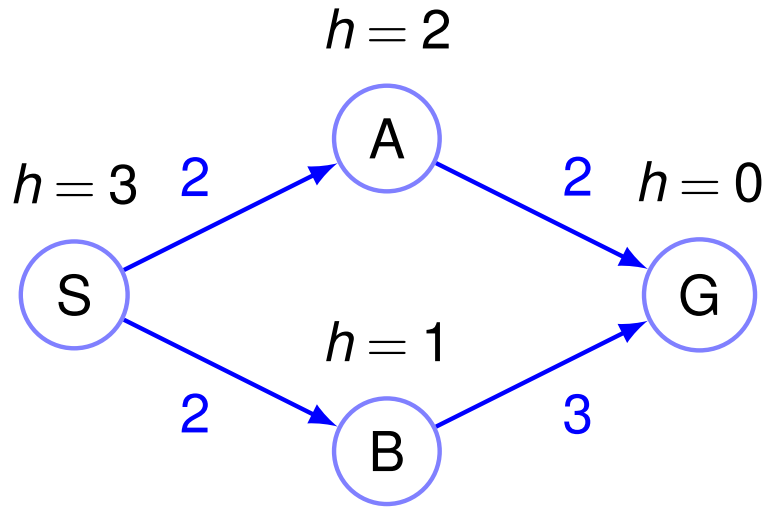
Uniform Cost Search.

Greedy

Astar.

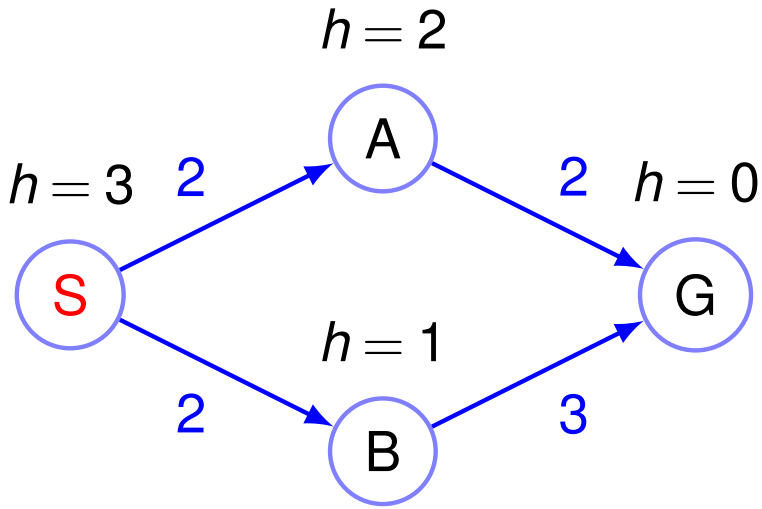
When should A* terminate?

Should we stop when we enqueue a goal?



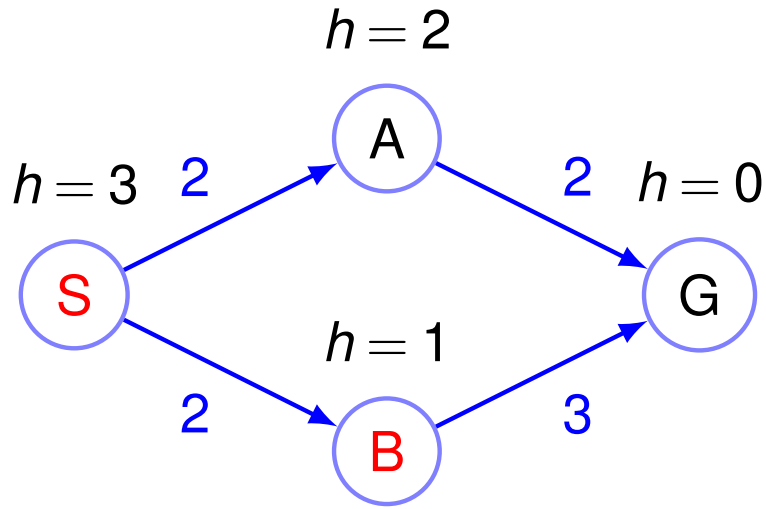
When should A* terminate?

Should we stop when we enqueue a goal?



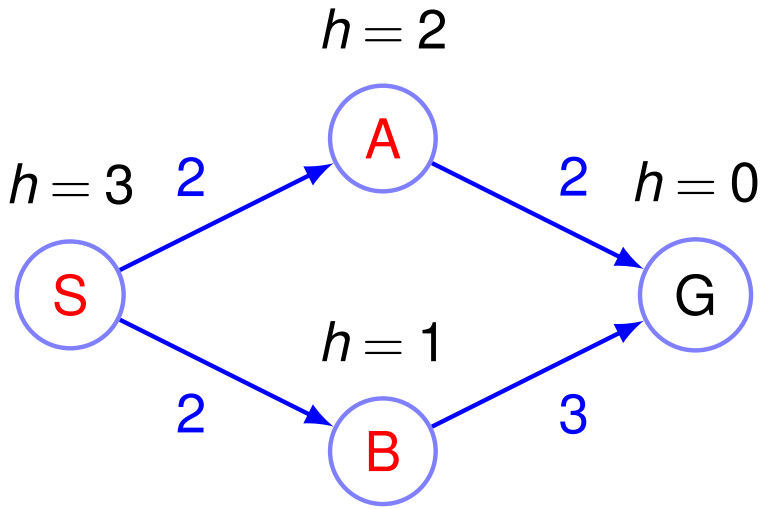
When should A* terminate?

Should we stop when we enqueue a goal?



When should A* terminate?

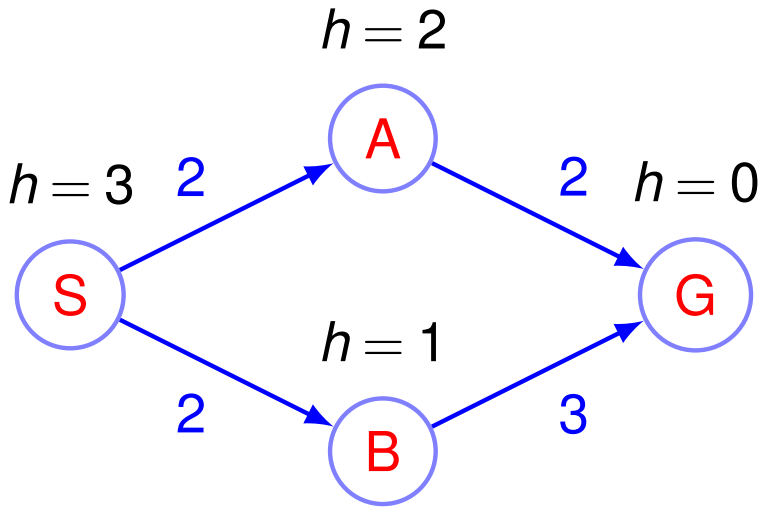
Should we stop when we enqueue a goal?



G in queue when B expanded.

When should A* terminate?

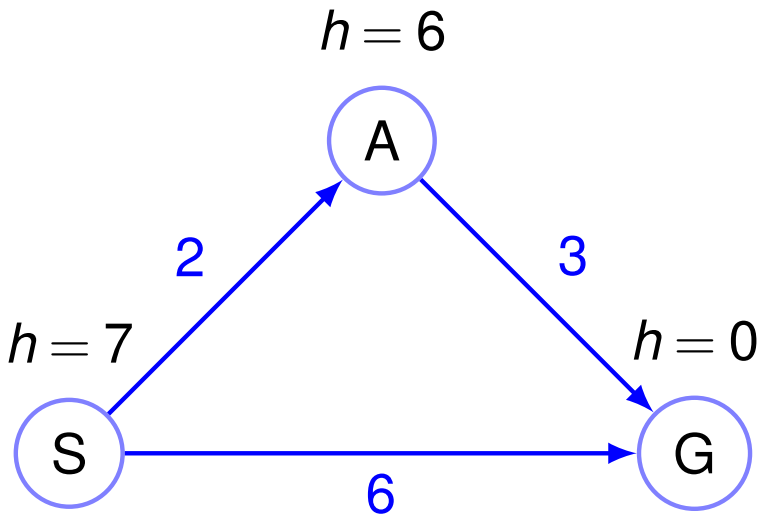
Should we stop when we enqueue a goal?



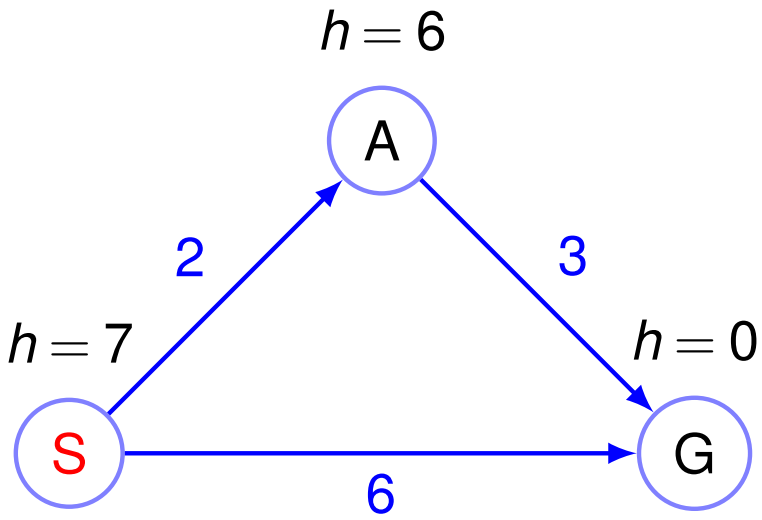
G in queue when B expanded.

No: only stop when we dequeue a goal.

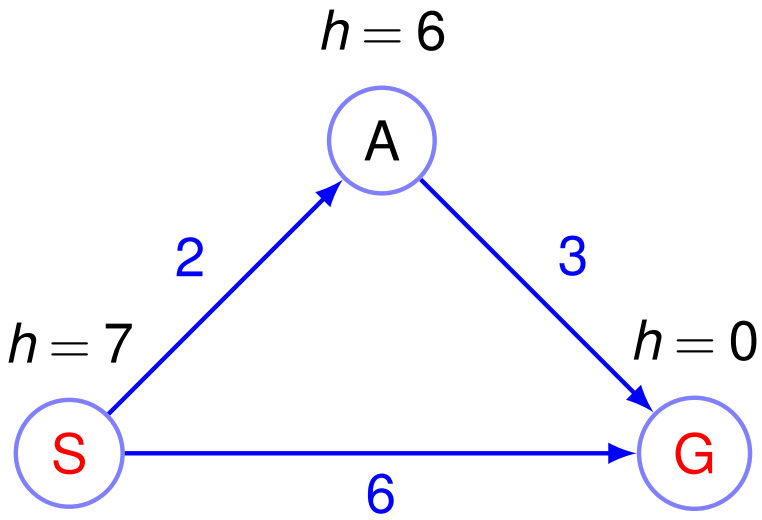
Is A* Optimal?



Is A* Optimal?

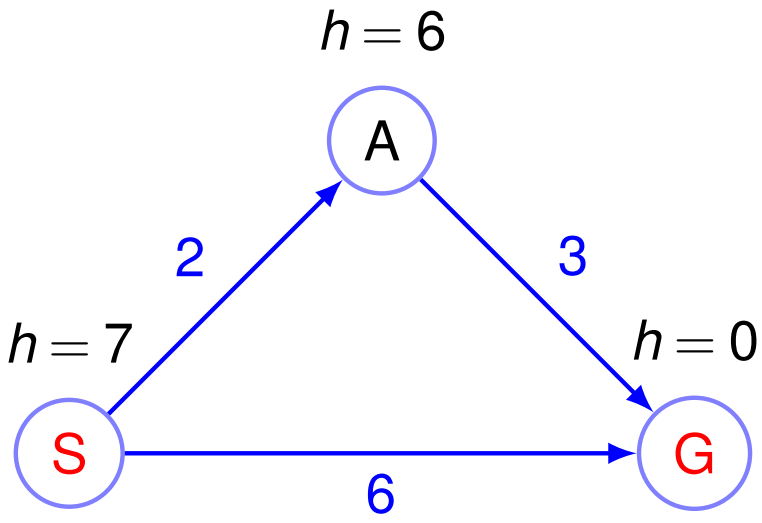


Is A* Optimal?



What goes wrong?

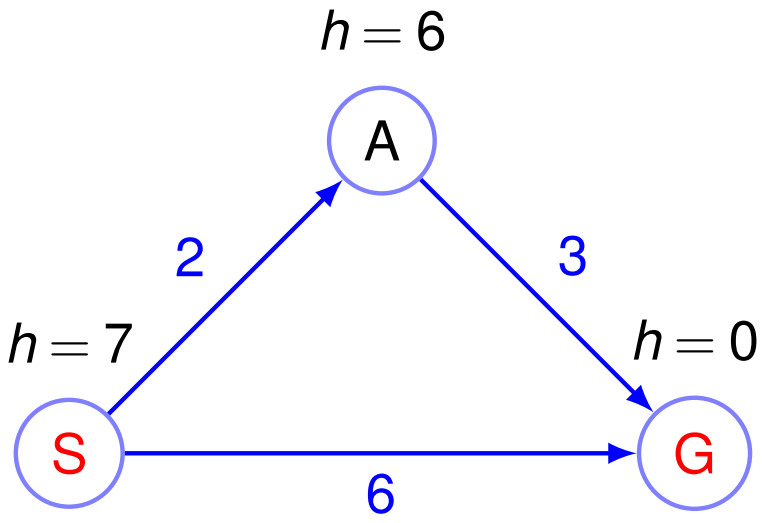
Is A* Optimal?



What goes wrong?

Actual goal cost $<$ estimated goal cost

Is A* Optimal?

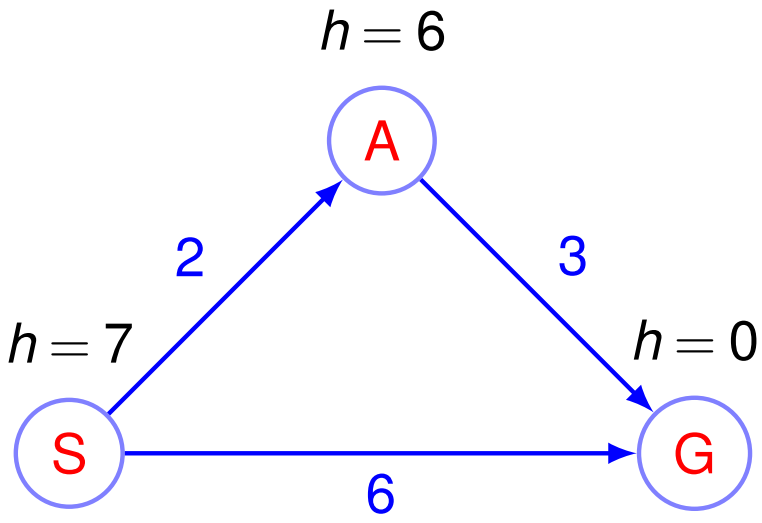


What goes wrong?

Actual goal cost $<$ estimated goal cost

We need estimates to be less than actual costs!

Is A* Optimal?

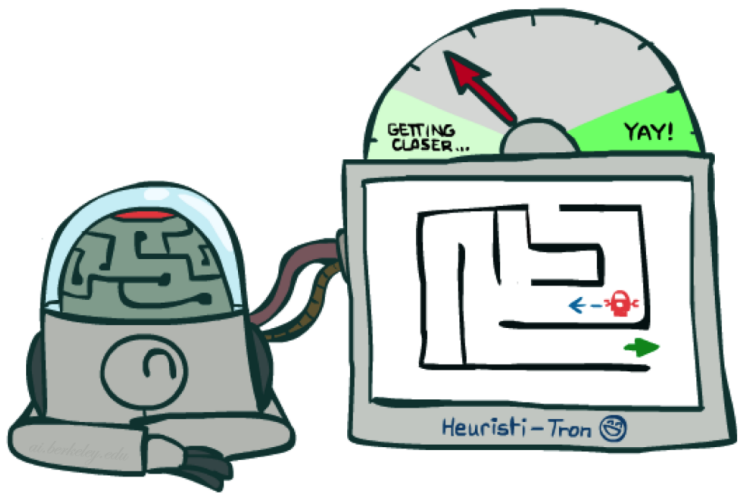


What goes wrong?

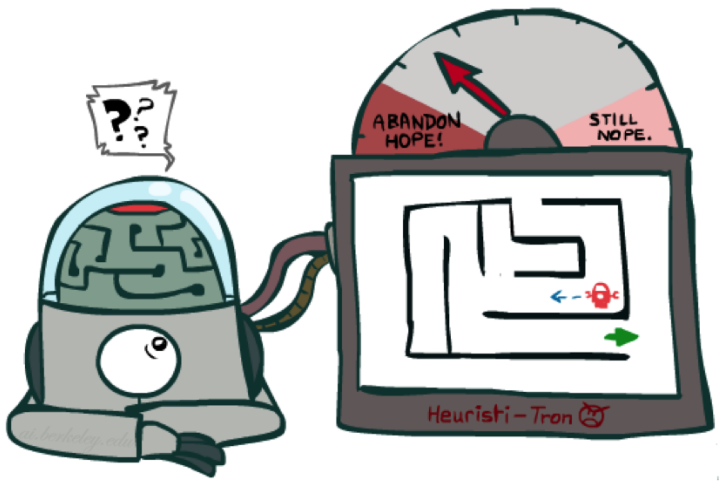
Actual goal cost $<$ estimated goal cost

We need estimates to be less than actual costs!
Lower bounds for actual costs.

Admissible Heuristics

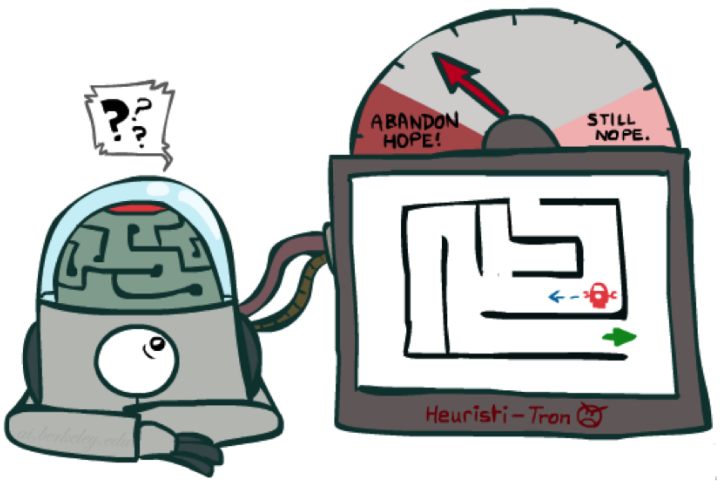


Idea: Admissibility

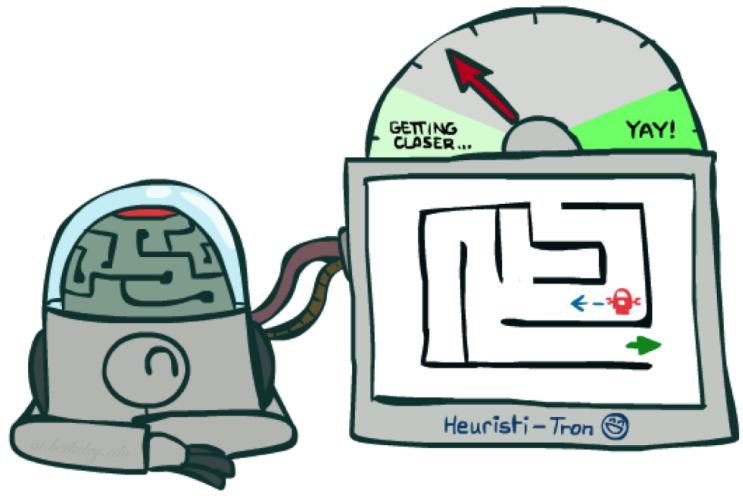


Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe

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

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

Admissible Heuristics

A heuristic h is admissible (optimistic) if:

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

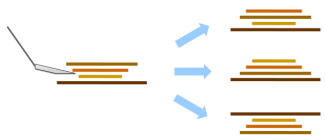
where $h^*(n)$ is the true cost to a nearest goal Note: Coming up with admissible heuristics is most of what's involved in using A* in practice.
Examples:

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

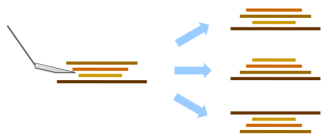


Admissible Heuristics

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



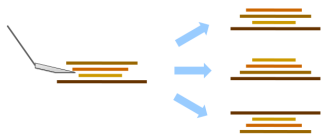
Optimize: number of flips.

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



Optimize: number of flips.

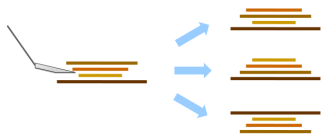
Largest out of place pancake.

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



Optimize: number of flips.

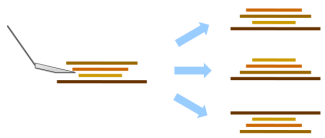
Largest out of place pancake.

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



Optimize: number of flips.

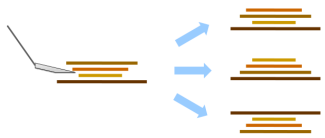
Largest out of place pancake.
Admissible?

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



Optimize: number of flips.

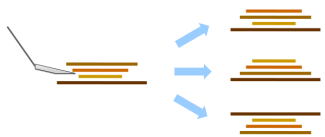
Largest out of place pancake.
Admissible?
No. For number of flips.
Yes. For height of stack.

Admissible Heuristics

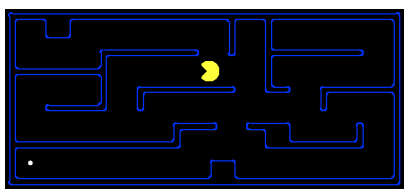
A heuristic h is admissible (optimistic) if:

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

where $h^*(n)$ is the true cost to a nearest goal Note: Coming up with admissible heuristics is most of what's involved in using A* in practice.
Examples:



Optimize: number of flips.



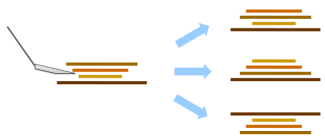
Largest out of place pancake.
Admissible?
No. For number of flips.
Yes. For height of stack.

Admissible Heuristics

A heuristic h is admissible (optimistic) if:

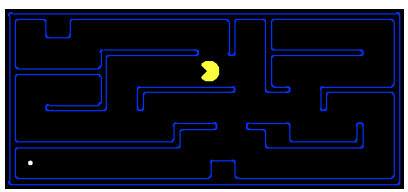
$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal Note: Coming up with admissible heuristics is most of what's involved in using A* in practice.
Examples:



Optimize: number of flips.

Largest out of place pancake.
Admissible?
No. For number of flips.
Yes. For height of stack.



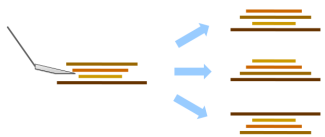
Manhattan distance.

Admissible Heuristics

A heuristic h is admissible (optimistic) if:

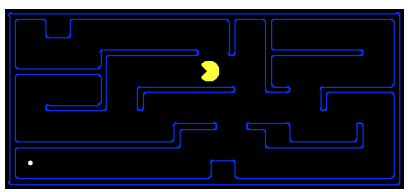
$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal Note: Coming up with admissible heuristics is most of what's involved in using A* in practice.
Examples:



Optimize: number of flips.

Largest out of place pancake.
Admissible?
No. For number of flips.
Yes. For height of stack.



Manhattan distance.

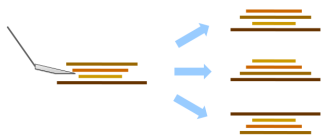
Admissible?

Admissible Heuristics

A heuristic h is admissible (optimistic) if:

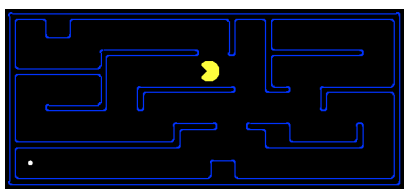
$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal Note: Coming up with admissible heuristics is most of what's involved in using A* in practice.
Examples:



Optimize: number of flips.

Largest out of place pancake.
Admissible?
No. For number of flips.
Yes. For height of stack.



Manhattan distance.

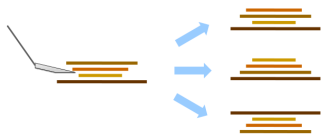
Admissible? Yes!

Admissible Heuristics

A heuristic h is admissible (optimistic) if:

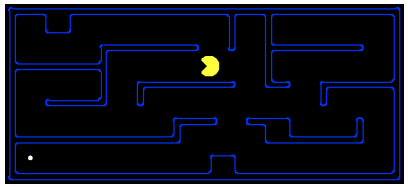
$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal Note: Coming up with admissible heuristics is most of what's involved in using A* in practice.
Examples:



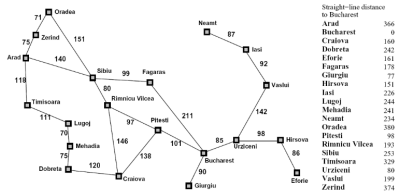
Optimize: number of flips.

Largest out of place pancake.
Admissible?
No. For number of flips.
Yes. For height of stack.



Manhattan distance.

Admissible? Yes!

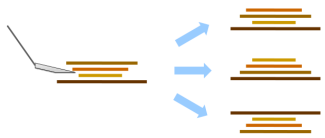


Admissible Heuristics

A heuristic h is admissible (optimistic) if:

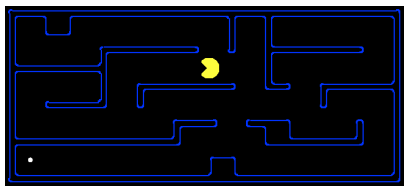
$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal Note: Coming up with admissible heuristics is most of what's involved in using A* in practice.
Examples:



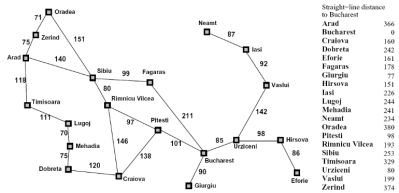
Optimize: number of flips.

Largest out of place pancake.
Admissible?
No. For number of flips.
Yes. For height of stack.



Manhattan distance.

Admissible? Yes!



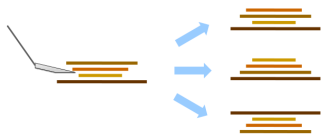
Euclidean Distance.

Admissible Heuristics

A heuristic h is admissible (optimistic) if:

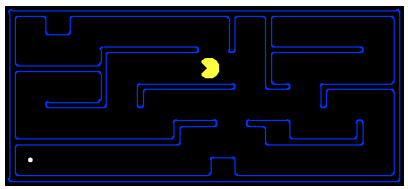
$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal Note: Coming up with admissible heuristics is most of what's involved in using A* in practice.
Examples:



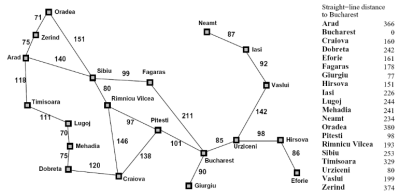
Optimize: number of flips.

Largest out of place pancake.
Admissible?
No. For number of flips.
Yes. For height of stack.



Manhattan distance.

Admissible? Yes!



Euclidean Distance.

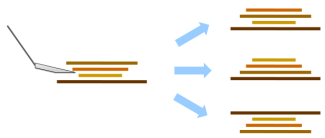
Admissible?

Admissible Heuristics

A heuristic h is admissible (optimistic) if:

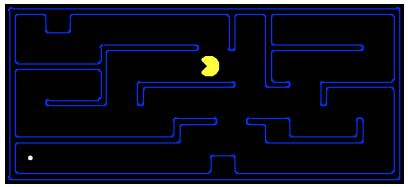
$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal Note: Coming up with admissible heuristics is most of what's involved in using A* in practice.
Examples:



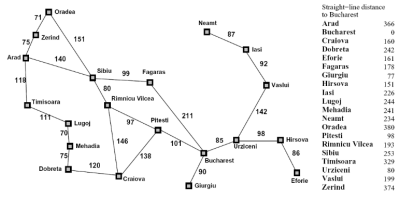
Optimize: number of flips.

Largest out of place pancake.
Admissible?
No. For number of flips.
Yes. For height of stack.



Manhattan distance.

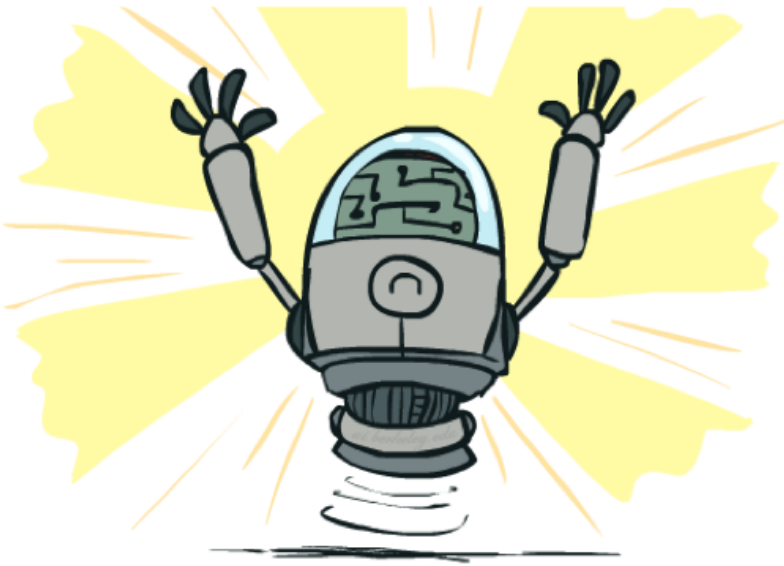
Admissible? Yes!



Euclidean Distance.

Admissible? Yes.

Optimality of A* Tree Search



Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Optimality of A* Tree Search: Blocking

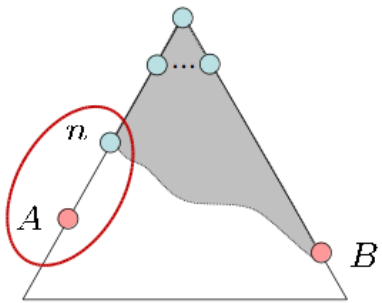
Claim: Goal A is expanded before B with $g(B) > g(A)$.

Proof:

Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Proof:

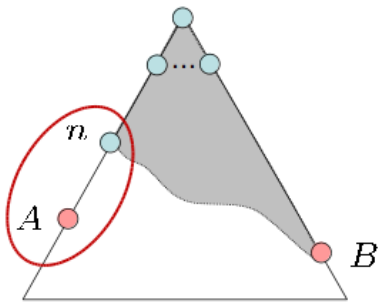


Definition of f-cost: $g(n) + h(n)$.

Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Proof:

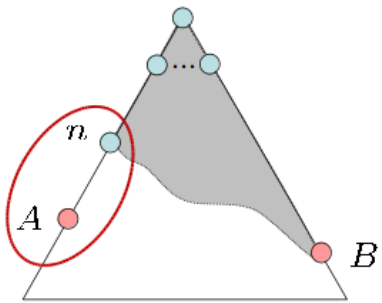


Definition of f-cost: $g(n) + h(n)$.
Admissibility of $h(n) \leq h^*(n)$

Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Proof:

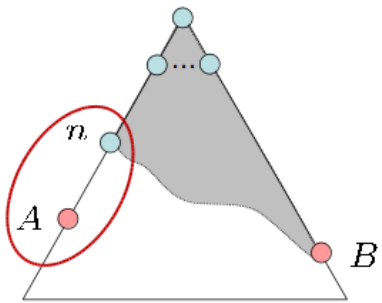


Definition of f-cost: $g(n) + h(n)$.
Admissibility of $h(n) \leq h^*(n)$
$\Longrightarrow$ h = 0 at a goal

Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Proof:

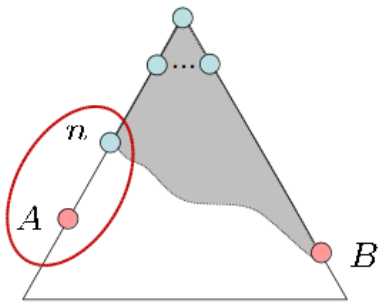


Definition of f-cost: $g(n) + h(n)$.
Admissibility of $h(n) \leq h^*(n)$
$\Longrightarrow$ h = 0 at a goal
Imagine B is on the fringe.

Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Proof:



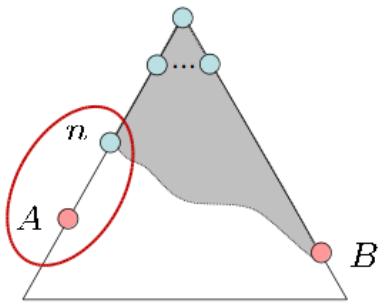
Definition of f-cost: $g(n) + h(n)$.
Admissibility of $h(n) \leq h^*(n)$
$\Longrightarrow$ h = 0 at a goal
Imagine B is on the fringe.

Some ancestor n of A is on the fringe, too (maybe A!)

Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Proof:



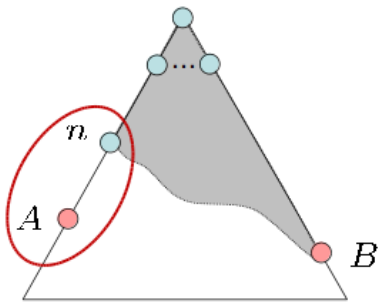
Definition of f-cost: $g(n) + h(n)$.
Admissibility of $h(n) \leq h^*(n)$
$\Longrightarrow$ h = 0 at a goal
Imagine B is on the fringe.

Some ancestor n of A is on the fringe, too (maybe A!)

Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Proof:



Definition of f-cost: $g(n) + h(n)$.
Admissibility of $h(n) \leq h^*(n)$
$\Longrightarrow$ h = 0 at a goal
Imagine B is on the fringe.

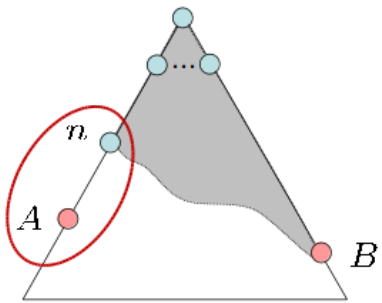
Some ancestor n of A is on the fringe, too (maybe A!)

Claim: n will be expanded before B

Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Proof:



Definition of f-cost: $g(n) + h(n)$.
Admissibility of $h(n) \leq h^*(n)$
$\Longrightarrow$ h = 0 at a goal
Imagine B is on the fringe.

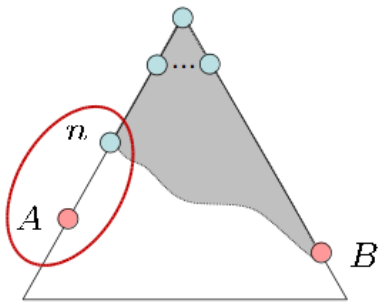
Some ancestor n of A is on the fringe, too (maybe A!)

Claim: n will be expanded before B
f(n) is less or equal to f(A)

Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Proof:



Definition of f-cost: $g(n) + h(n)$.
Admissibility of $h(n) \leq h^*(n)$
$\Longrightarrow$ h = 0 at a goal
Imagine B is on the fringe.

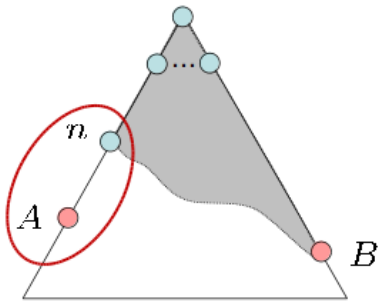
Some ancestor n of A is on the fringe, too (maybe A!)

Claim: n will be expanded before B
f(n) is less or equal to f(A)
since $f(n) = g(n) + h(n) < f(A)$,
f(A) is less than f(B)

Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Proof:



Definition of f-cost: $g(n) + h(n)$.
Admissibility of $h(n) \leq h^*(n)$
$\Longrightarrow$ h = 0 at a goal
Imagine B is on the fringe.

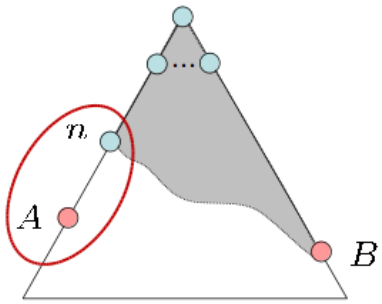
Some ancestor n of A is on the fringe, too (maybe A!)

Claim: n will be expanded before B
f(n) is less or equal to f(A)
since $f(n) = g(n) + h(n) < f(A)$,
f(A) is less than f(B)
since $g(A) + 0 < g(B) + h(B)$

Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Proof:



Definition of f-cost: $g(n) + h(n)$.
Admissibility of $h(n) \leq h^*(n)$
$\Longrightarrow$ h = 0 at a goal
Imagine B is on the fringe.

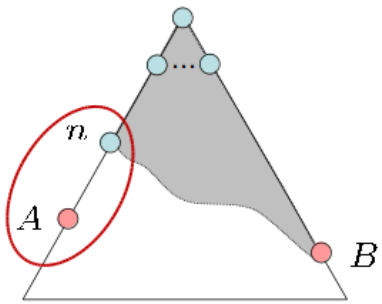
Some ancestor n of A is on the fringe, too (maybe A!)

Claim: n will be expanded before B
f(n) is less or equal to f(A)
since $f(n) = g(n) + h(n) < f(A)$,
f(A) is less than f(B)
since $g(A) + 0 < g(B) + h(B)$

Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Proof:



Definition of f-cost: $g(n) + h(n)$.
Admissibility of $h(n) \leq h^*(n)$
$\Longrightarrow$ h = 0 at a goal
Imagine B is on the fringe.

Some ancestor n of A is on the fringe, too (maybe A!)

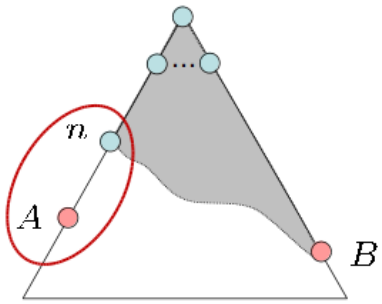
Claim: n will be expanded before B
f(n) is less or equal to f(A)
since $f(n) = g(n) + h(n) < f(A)$,
f(A) is less than f(B)
since $g(A) + 0 < g(B) + h(B)$

All ancestors of A expand before B.

Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Proof:



Definition of f-cost: $g(n) + h(n)$.
Admissibility of $h(n) \leq h^*(n)$
$\Longrightarrow$ h = 0 at a goal
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
f(n) is less or equal to f(A)
since $f(n) = g(n) + h(n) < f(A)$,
f(A) is less than f(B)
since $g(A) + 0 < g(B) + h(B)$

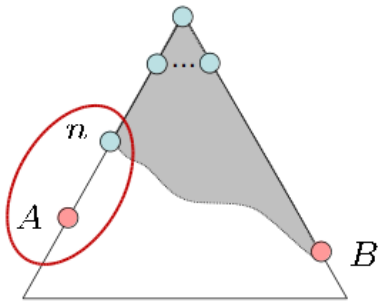
All ancestors of A expand before B.

A expands before B.

Optimality of A* Tree Search: Blocking

Claim: Goal A is expanded before B with $g(B) > g(A)$.

Proof:



Definition of f-cost: $g(n) + h(n)$.
Admissibility of $h(n) \leq h^*(n)$
$\Longrightarrow$ h = 0 at a goal
Imagine B is on the fringe.

Some ancestor n of A is on the fringe, too (maybe A!)

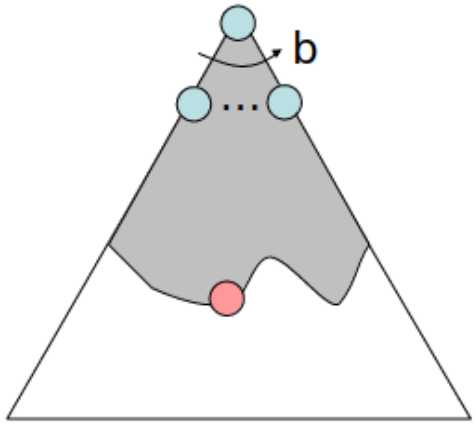
Claim: n will be expanded before B
f(n) is less or equal to f(A)
since $f(n) = g(n) + h(n) < f(A)$,
f(A) is less than f(B)
since $g(A) + 0 < g(B) + h(B)$

All ancestors of A expand before B.

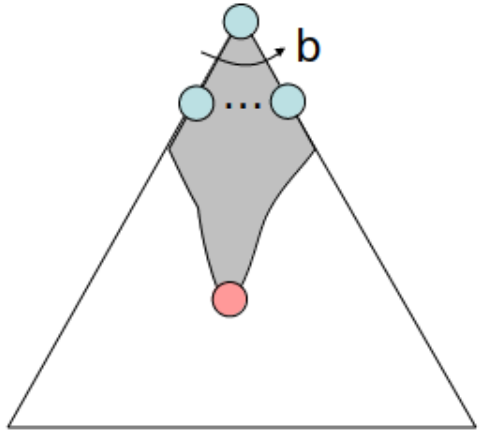
A expands before B. $\rightarrow$ A* search is optimal.

Properties of A*

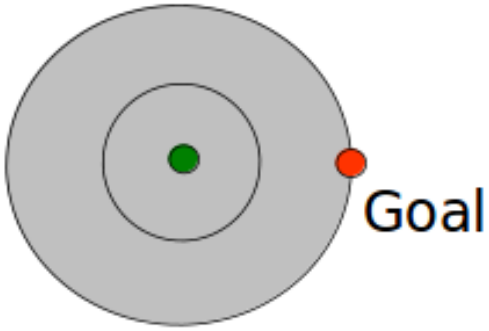
UCS



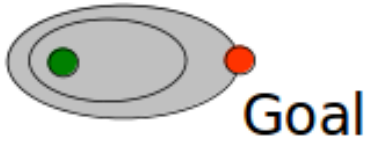
A*



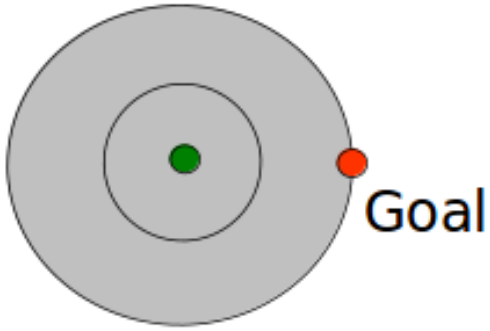
UCS vs A* Contours



Uniform-cost expands equally in all "directions".

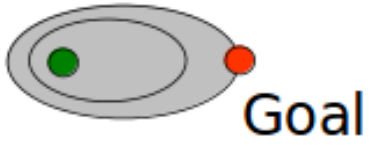


UCS vs A* Contours



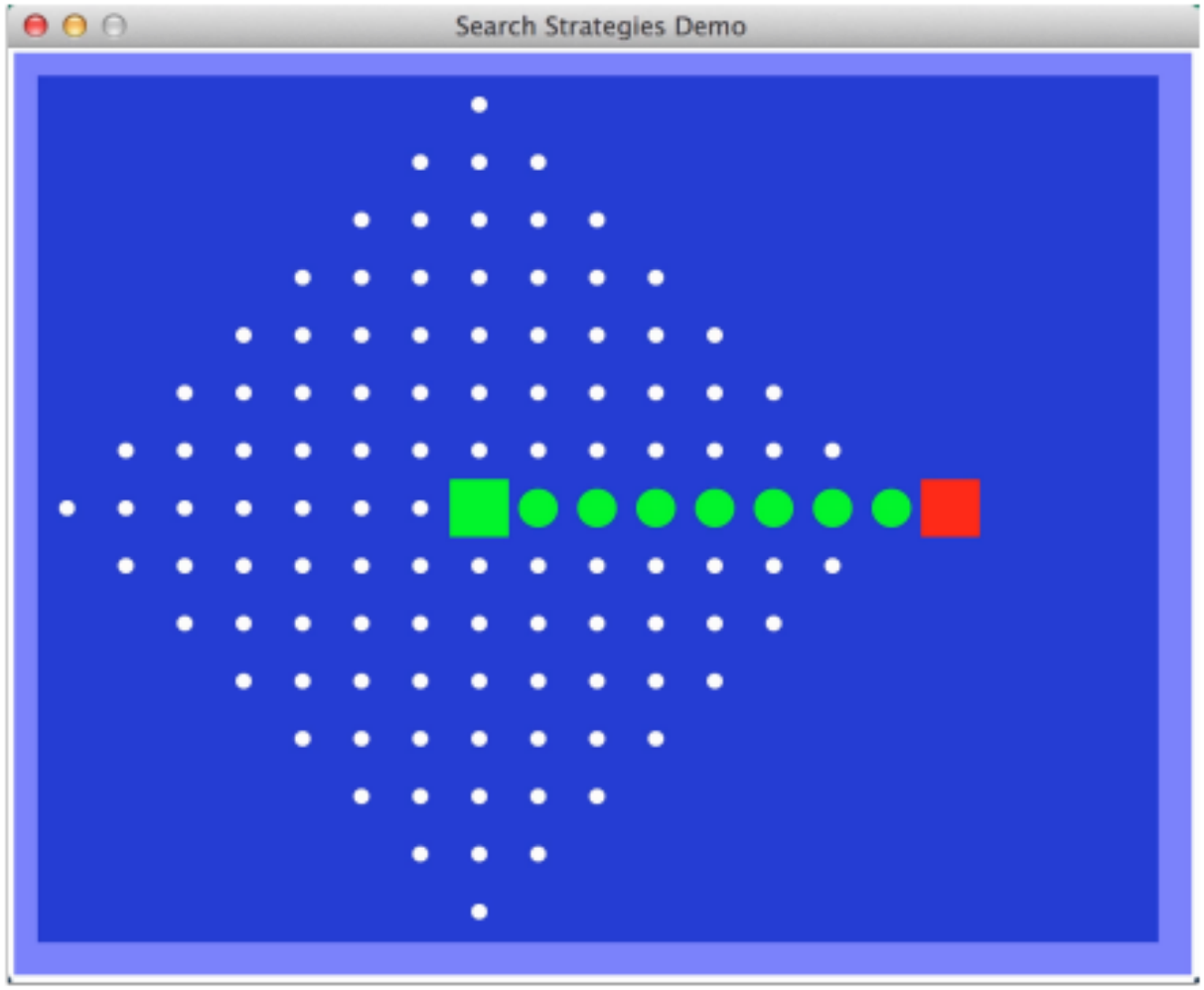
Uniform-cost expands equally in all "directions".

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

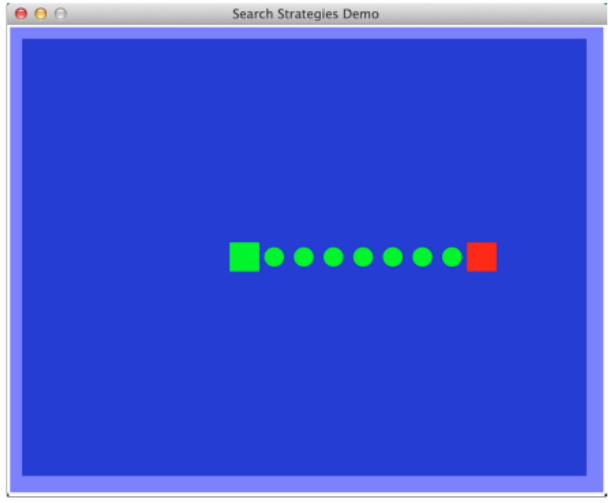


[Demo: contours UCS / greedy / A* empty (L3D1)]
[Demo: contours A* pacmansmall maze (L3D5)]

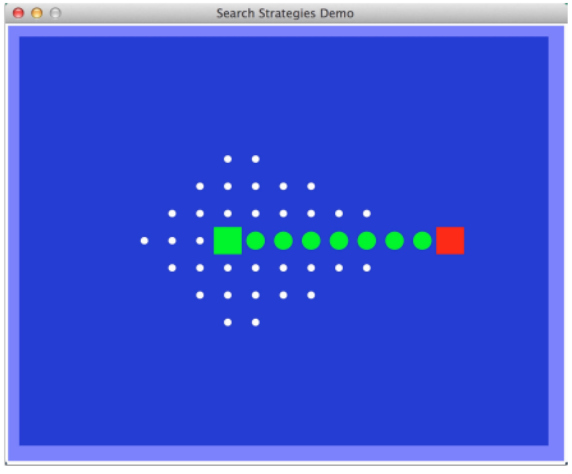
Video of Demo Contours (Empty) – UCS



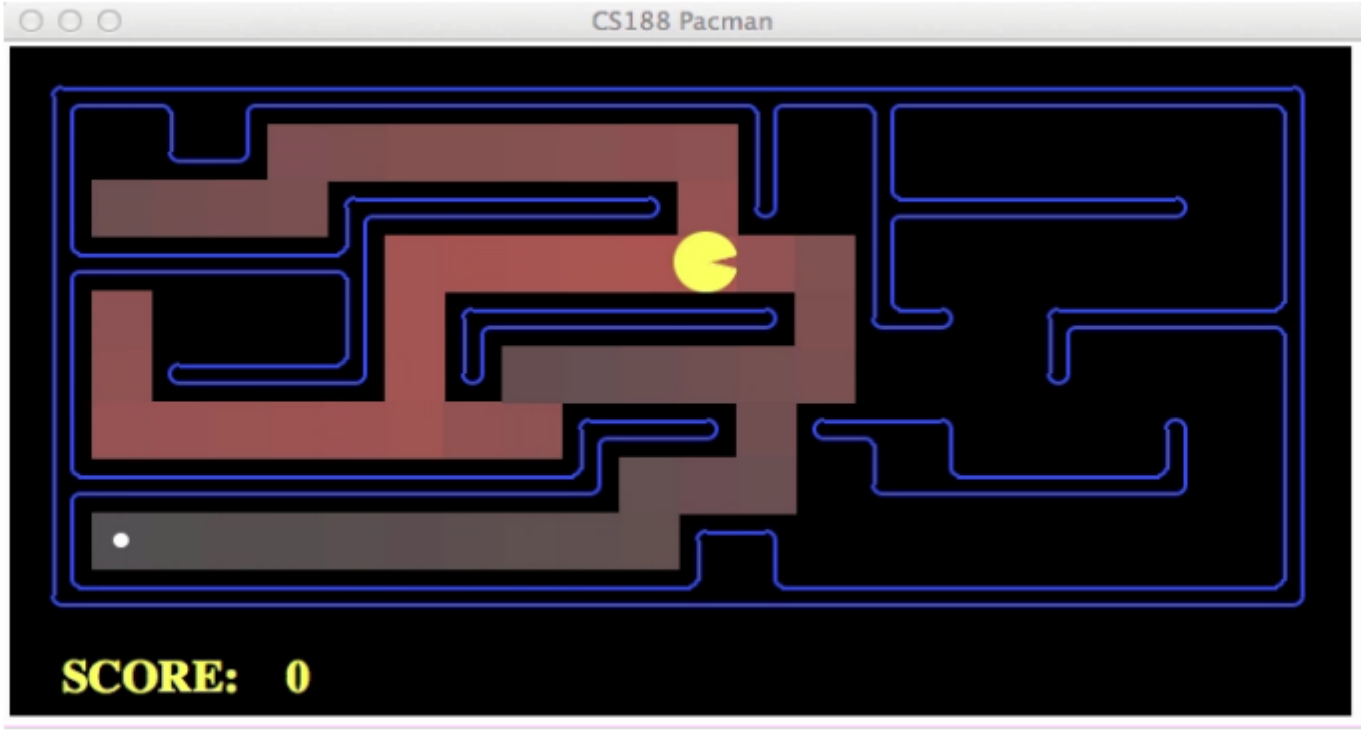
Video of Demo Contours (Empty) – Greedy



Video of Demo Contours (Empty) – A*

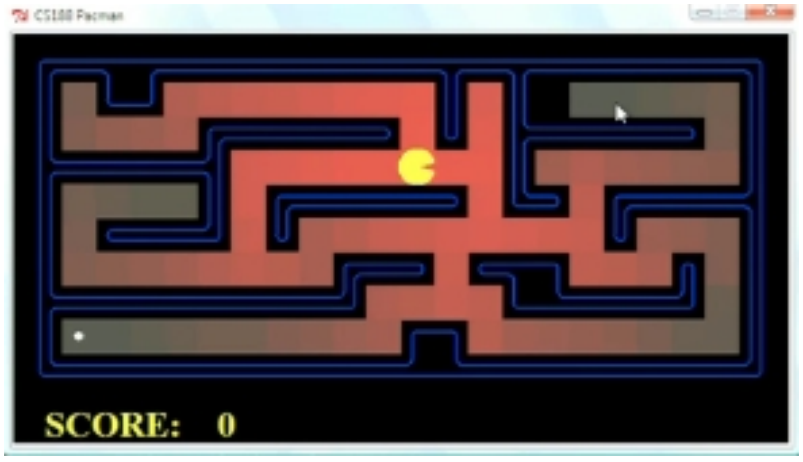


Video of Demo Contours: Pacman A*

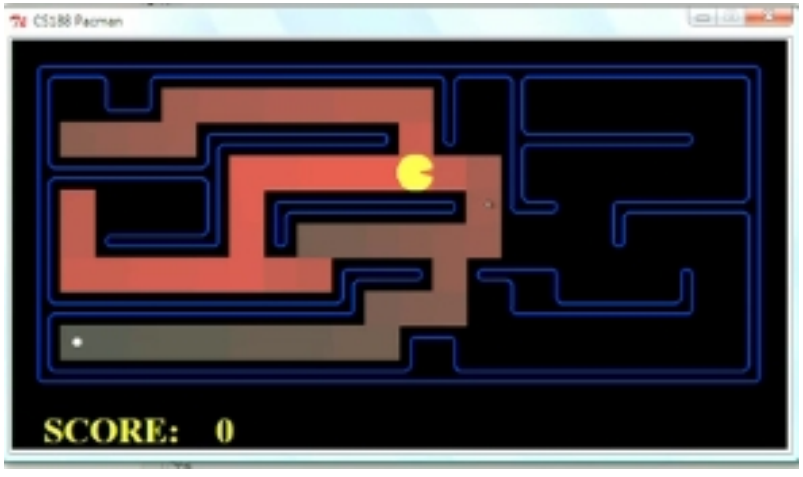


Comparison

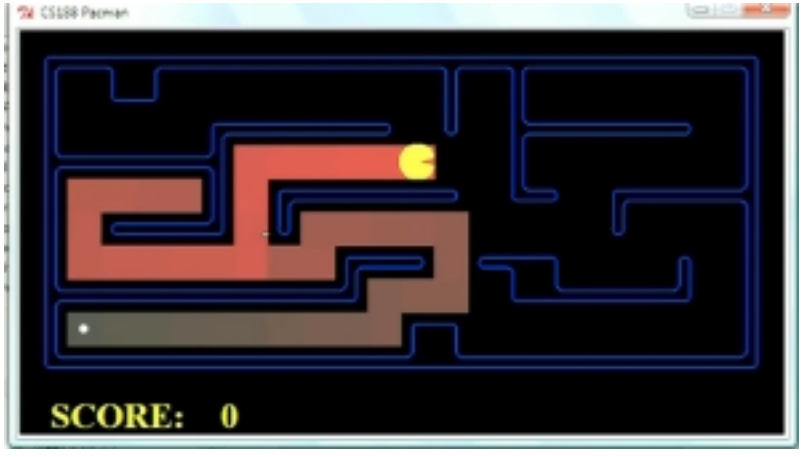
Uniform Cost



Greedy



A*



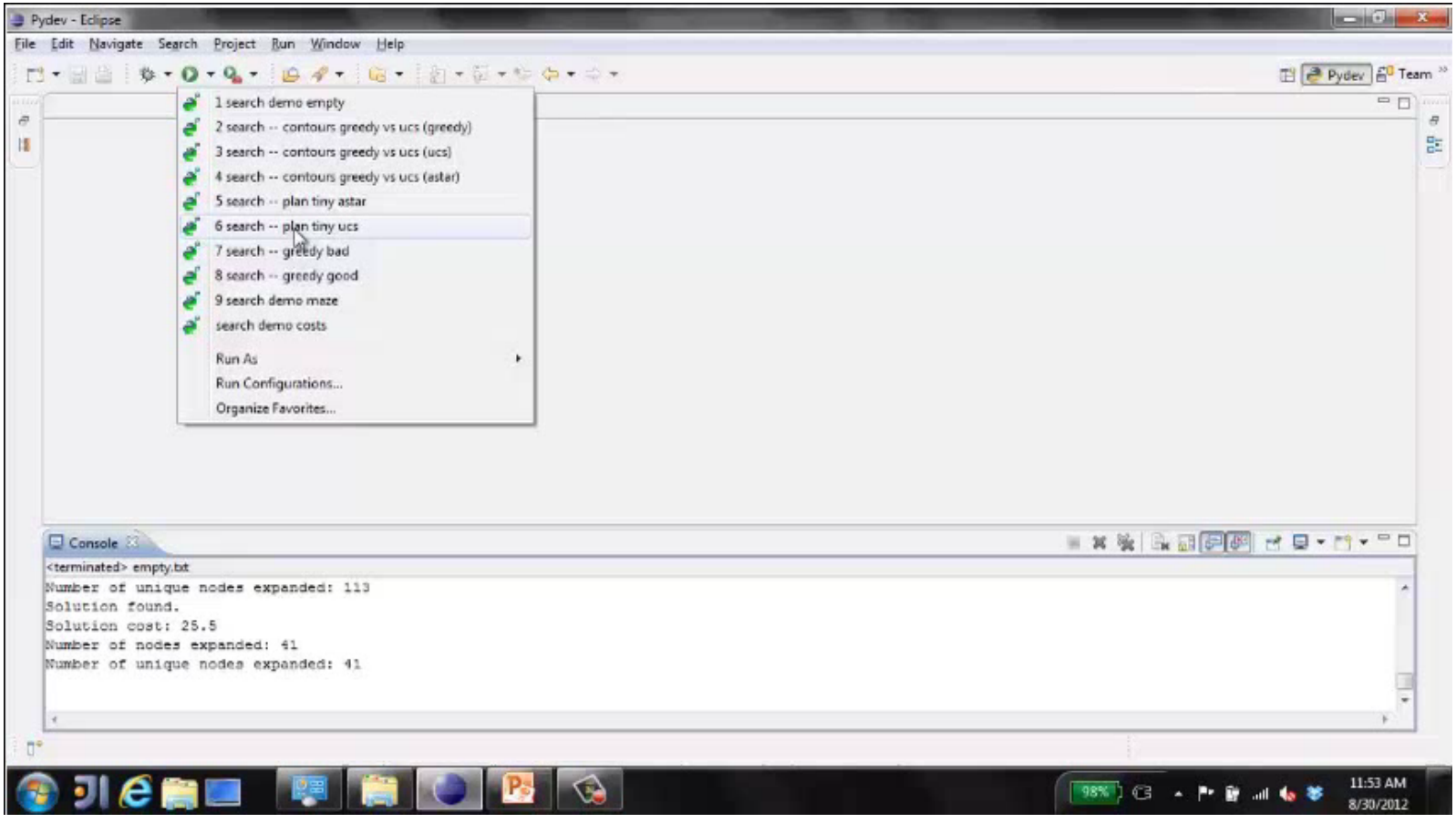
A* Applications

```
G attack: false G defense: false
[3553:WorkerLoiterAndHarassController16] INFO - Scared of a visible Terran Marine.   40 f:    0 n:    0 s:
Global(D) t:    40 f:    0 n:    0 s:
```

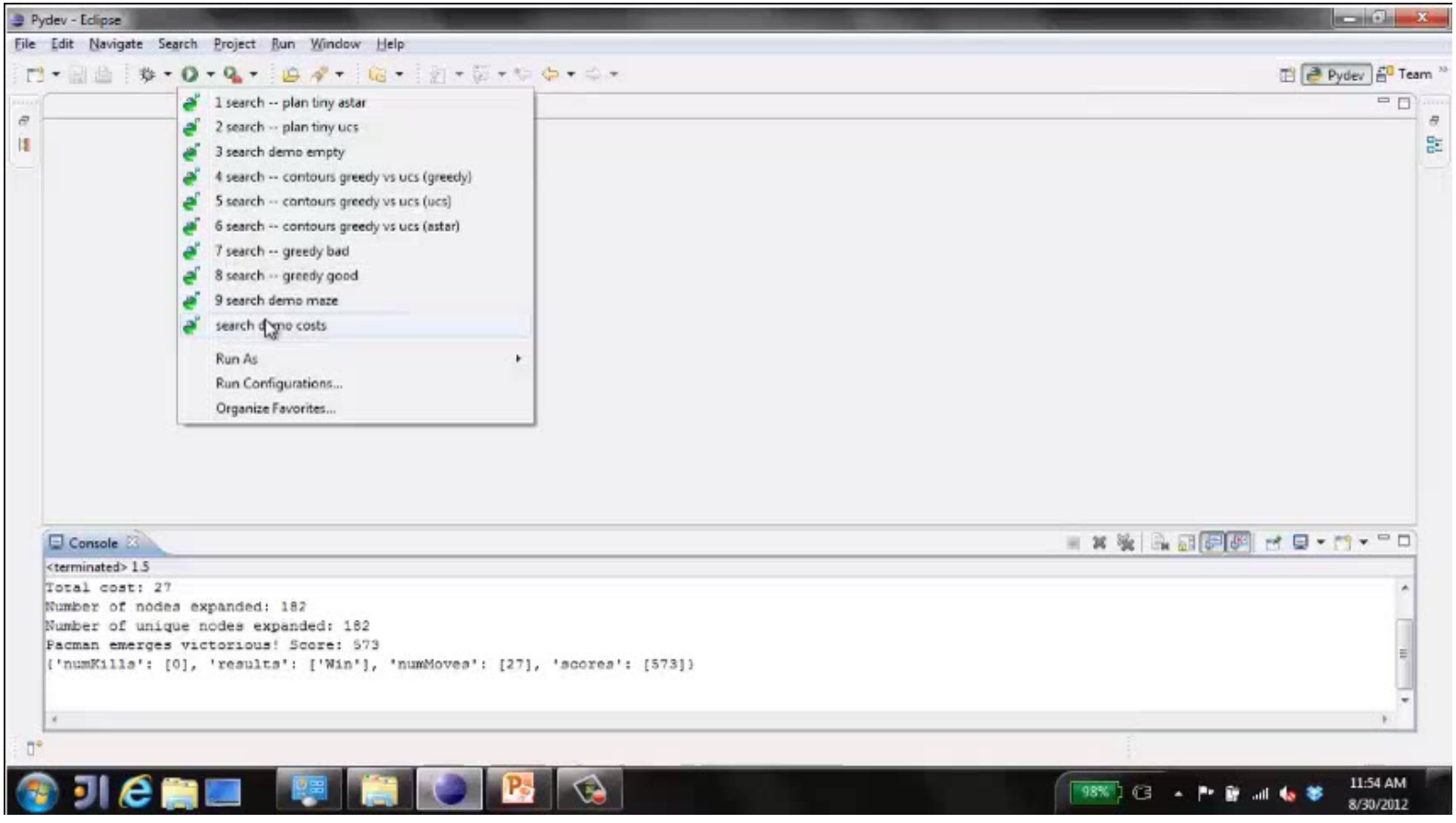
A* Applications

Video games
Pathing / routing problems
Resource planning problems
Robot motion planning
Language analysis
Machine translation
Speech recognition
[Demo: UCS / A * pacman tiny maze (L3D6,L3D7)]
[Demo: guess algorithm Empty Shallow/Deep (L3D8)]

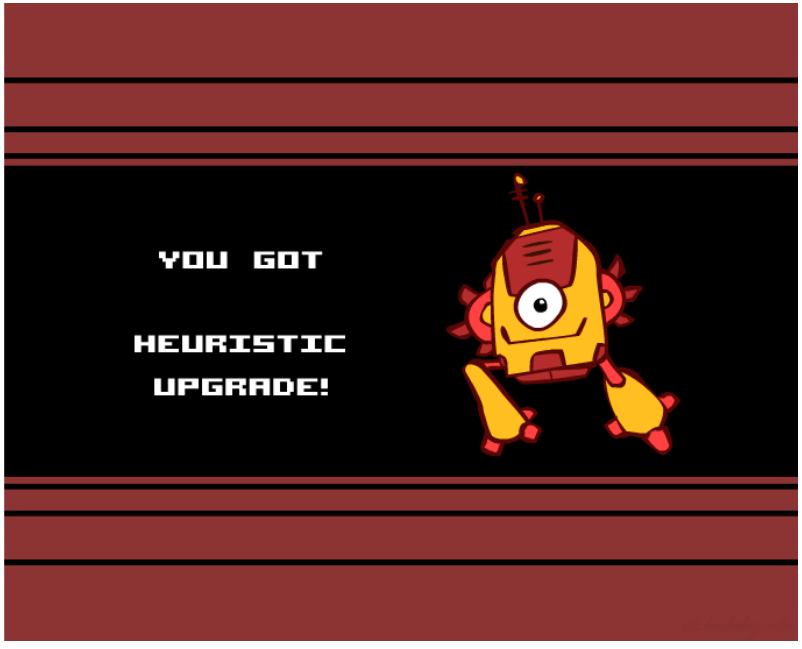
Video of Demo Pacman (Tiny Maze) – UCS / A*



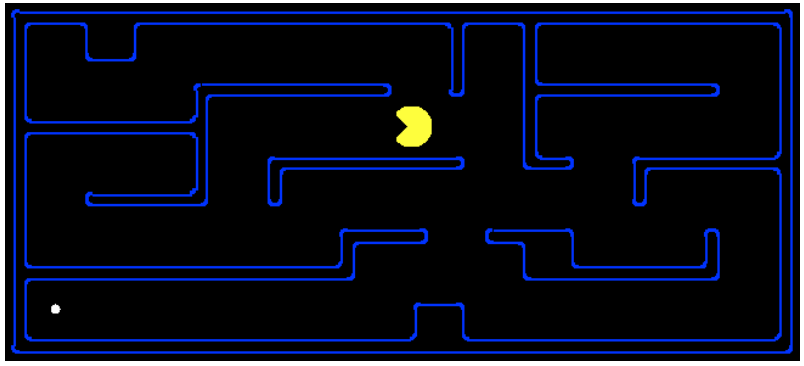
Video: Demo Water Shallow/Deep – Guess Algorithm



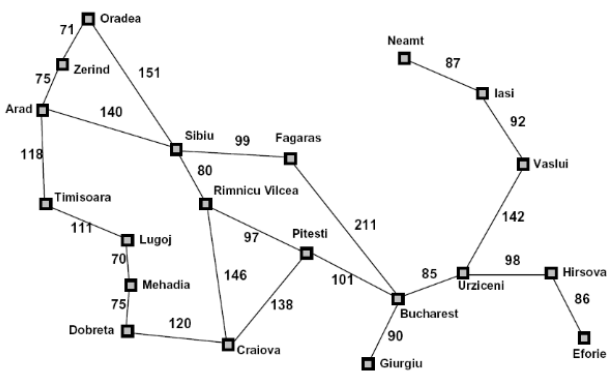
Creating Heuristics



Creating Admissible Heuristics



Manhattan Distance: 15

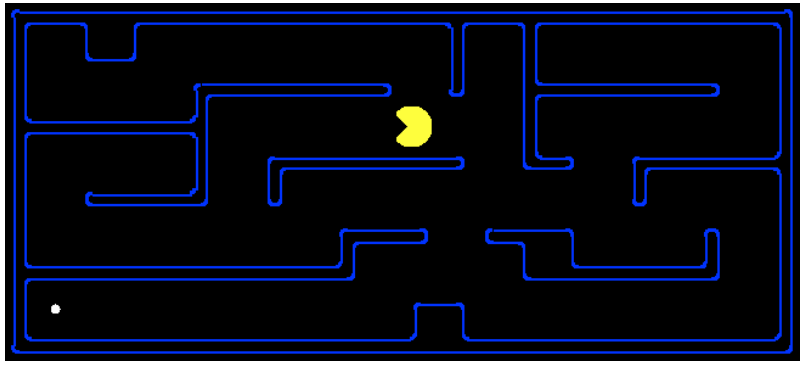


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

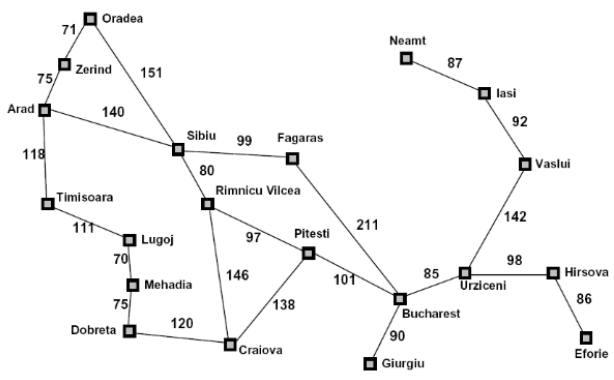
Straight Line Distance: 366

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

Creating Admissible Heuristics



Manhattan Distance: 15



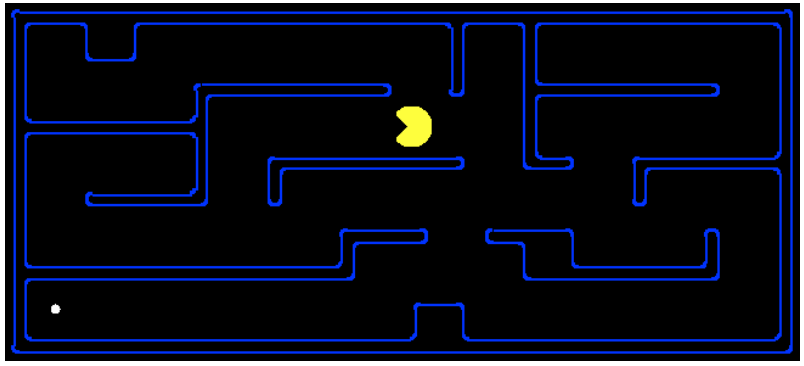
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Straight Line Distance: 366

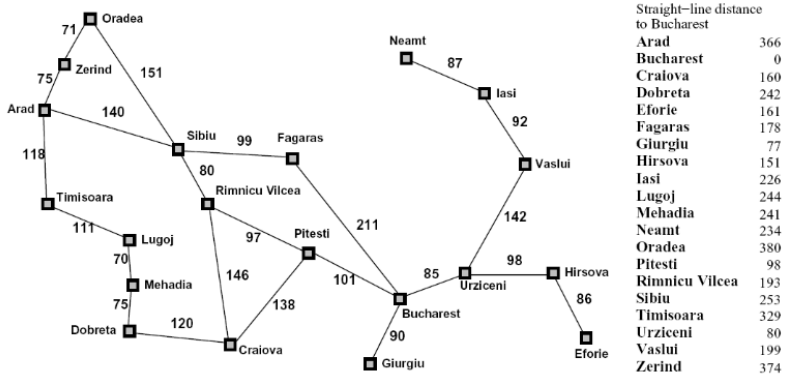
Most of the work in solving hard search problems optimally is in coming up with admissible heuristics

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

Creating Admissible Heuristics



Manhattan Distance: 15



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

Straight Line Distance: 366

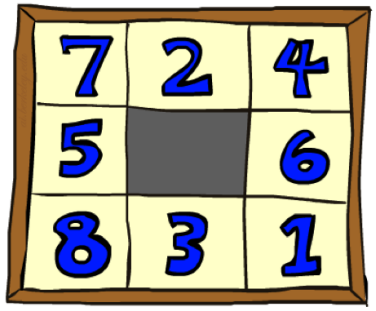
Most of the work in solving hard search problems optimally is in coming up with admissible heuristics

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

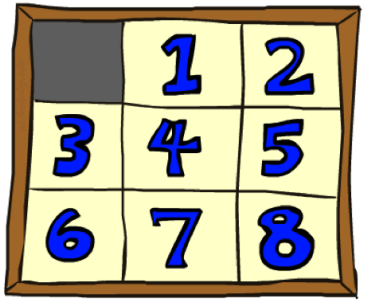
Inadmissible heuristics are often useful too.

Example: 8 Puzzle

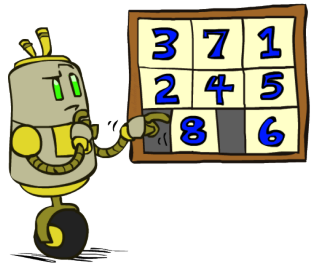
Start State



Goal State

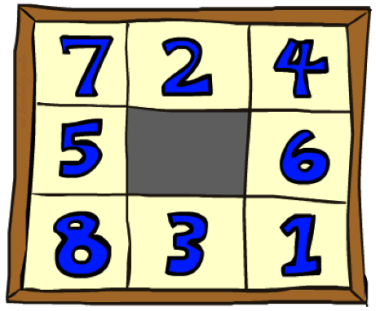


Actions

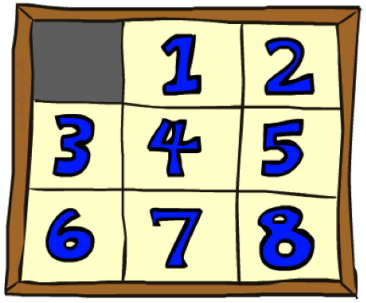


Example: 8 Puzzle

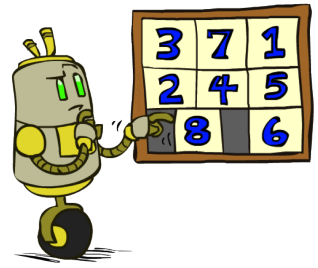
Start State



Goal State



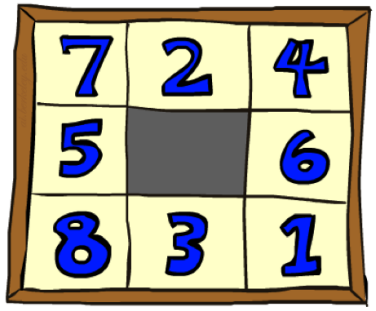
Actions



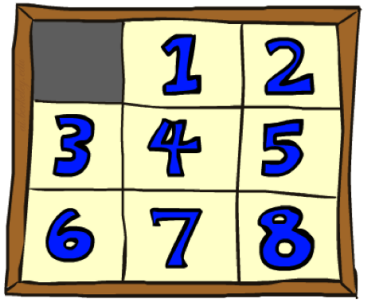
What are the states?

Example: 8 Puzzle

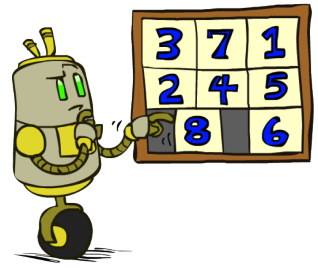
Start State



Goal State



Actions

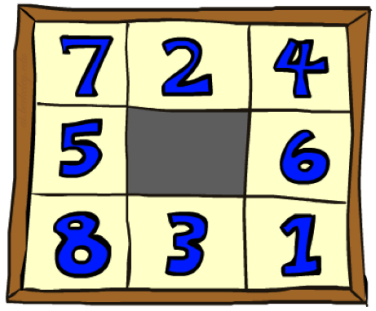


What are the states?

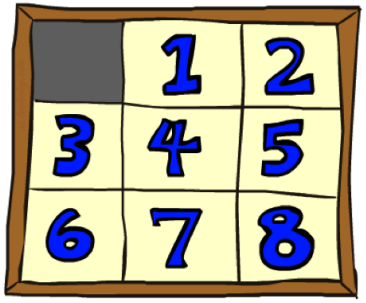
How many states?

Example: 8 Puzzle

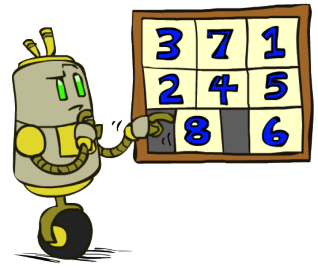
Start State



Goal State



Actions



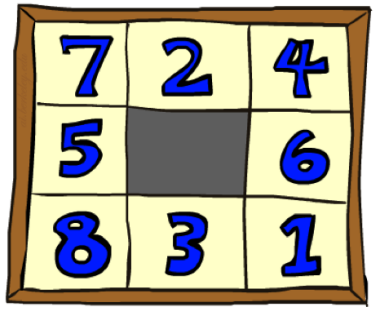
What are the states?

How many states?

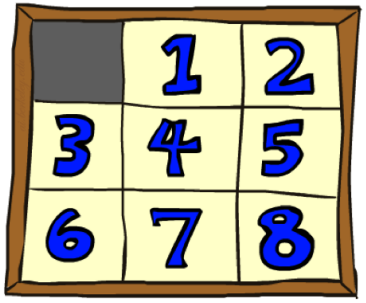
What are the actions?

Example: 8 Puzzle

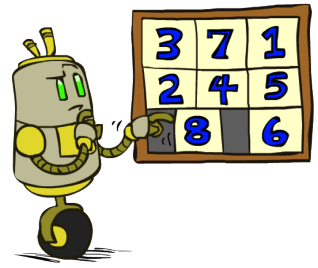
Start State



Goal State



Actions



What are the states?

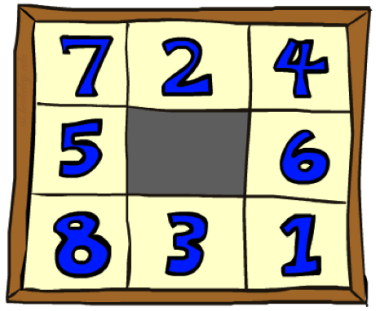
How many states?

What are the actions?

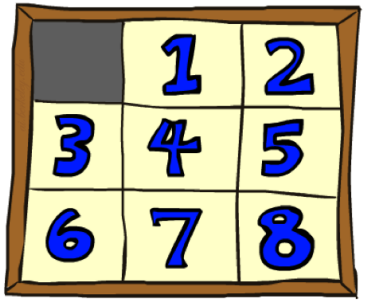
How many successors from the start state?

Example: 8 Puzzle

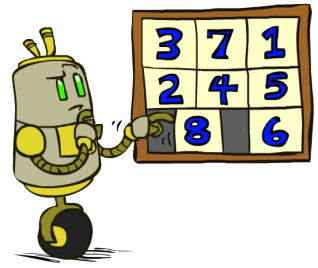
Start State



Goal State



Actions



What are the states?

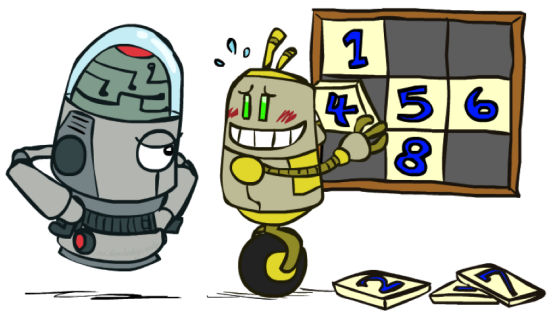
How many states?

What are the actions?

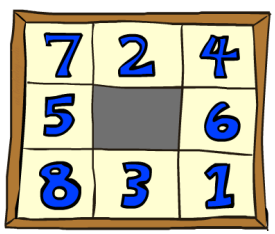
How many successors from the start state?

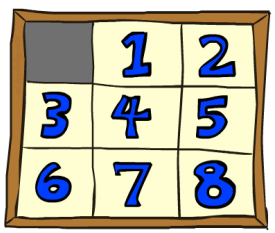
What should the costs be?

8 Puzzle I

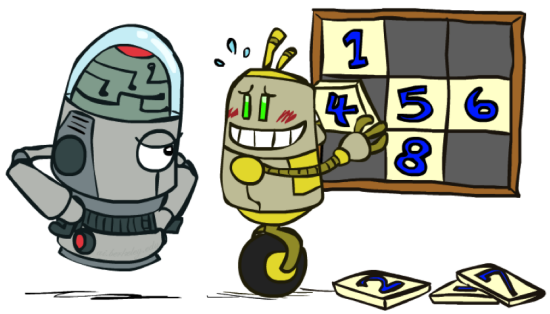


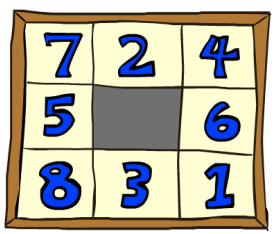
Heuristic: Number of tiles misplaced

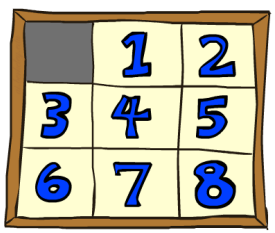




8 Puzzle I



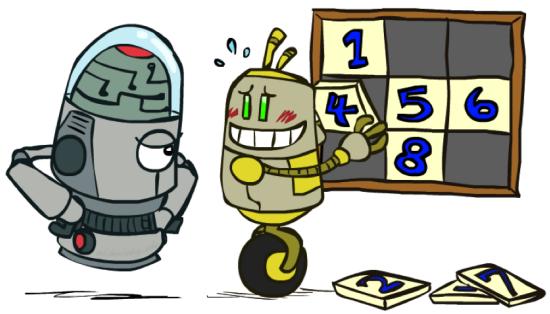


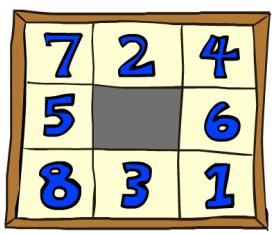


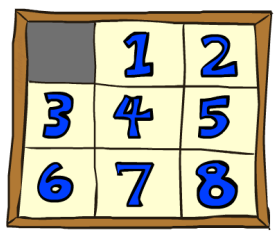
Heuristic: Number of tiles misplaced

Why is it admissible? h(start) =

8 Puzzle I





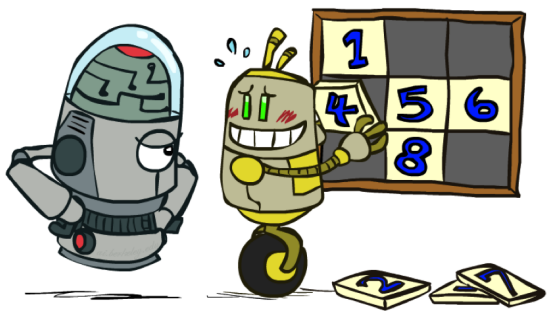


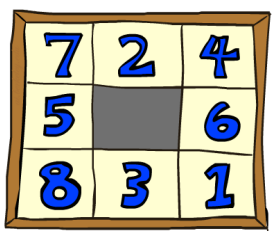
Heuristic: Number of tiles misplaced

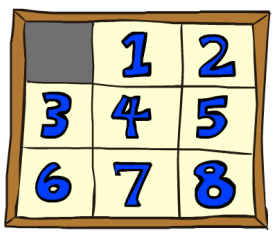
Why is it admissible? h(start) =

This is a relaxed-problem heuristic

8 Puzzle I







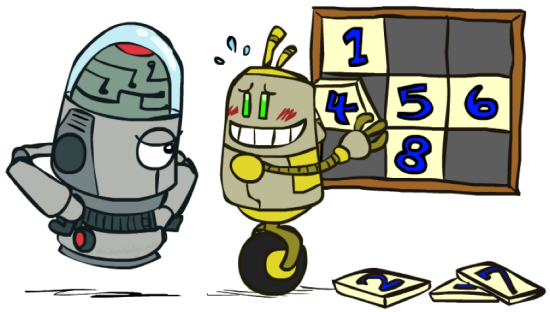
Heuristic: Number of tiles misplaced

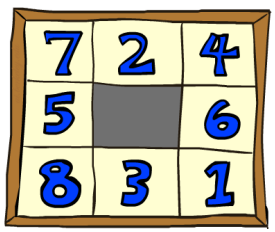
Why is it admissible? h(start) =

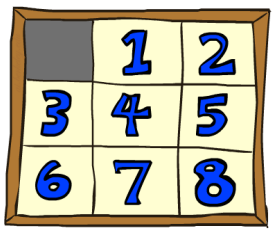
This is a relaxed-problem heuristic

Average nodes expanded when the optimal path has

8 Puzzle I







Heuristic: Number of tiles misplaced

Why is it admissible? h(start) =

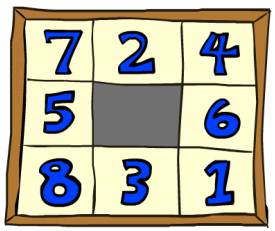
This is a relaxed-problem heuristic

Average nodes expanded when the optimal path has

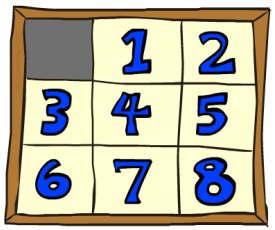
	4 steps	8 steps	12 steps.
UCS	112	6300	3.6×10^6
TILES	13	39	227

8 Puzzle II

Easier 8-puzzle: tile could slide any direction at any time, ignoring other tiles.



Start State

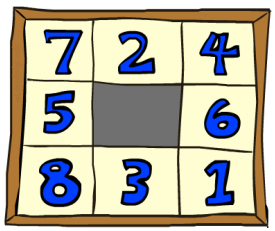


Goal State

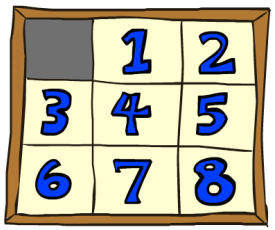
8 Puzzle II

Easier 8-puzzle: tile could slide any direction at any time, ignoring other tiles.

Total Manhattan distance

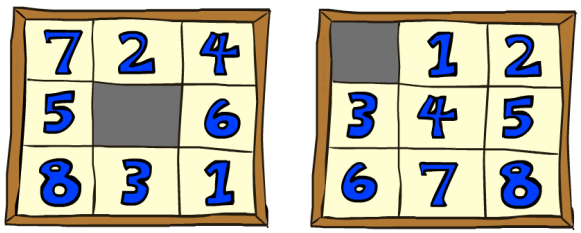


Start State



Goal State

8 Puzzle II



Start State

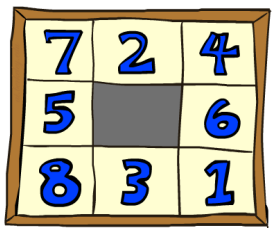
Goal State

Easier 8-puzzle: tile could slide any direction at any time, ignoring other tiles.

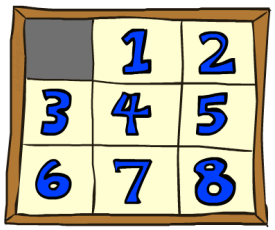
Total Manhattan distance

Why is it admissible?

8 Puzzle II



Start State



Goal State

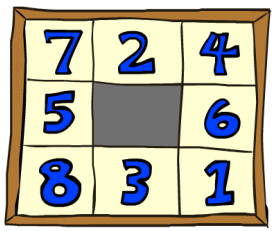
Easier 8-puzzle: tile could slide any direction at any time, ignoring other tiles.

Total Manhattan distance

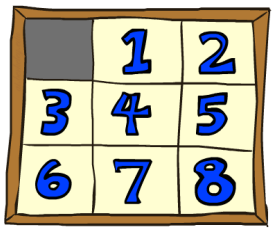
Why is it admissible?

$h(start) = 3 + 1 + 2 + ... = 18.$

8 Puzzle II



Start State



Goal State

Easier 8-puzzle: tile could slide any direction at any time, ignoring other tiles.

Total Manhattan distance

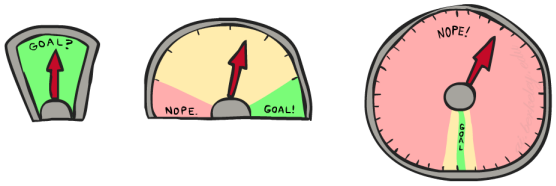
Why is it admissible?

$h(start) = 3 + 1 + 2 + ... = 18.$

Average nodes expanded when the optimal path has:

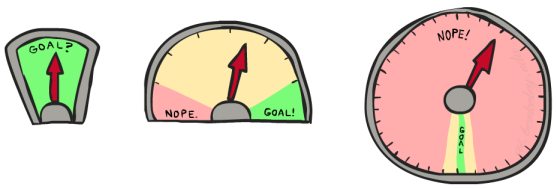
	4 steps	8 steps	12 steps.
TILES	13	39	227
MANHATTAN	12	25	73

8 Puzzle III



How about using the actual cost as a heuristic?

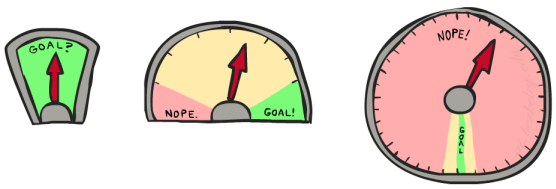
8 Puzzle III



How about using the actual cost as a heuristic?

Would it be admissible?

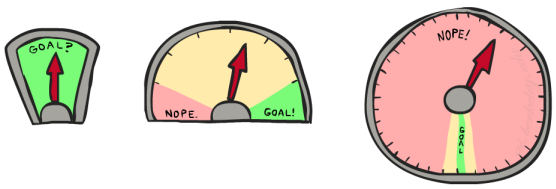
8 Puzzle III



How about using the actual cost as a heuristic?

- Would it be admissible?
- Would we save on nodes expanded?

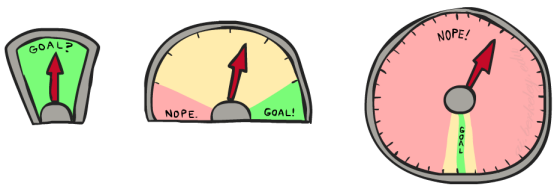
8 Puzzle III



How about using the actual cost as a heuristic?

- Would it be admissible?
- Would we save on nodes expanded?
- What's wrong with it?

8 Puzzle III



How about using the actual cost as a heuristic?

Would it be admissible?

Would we save on nodes expanded?

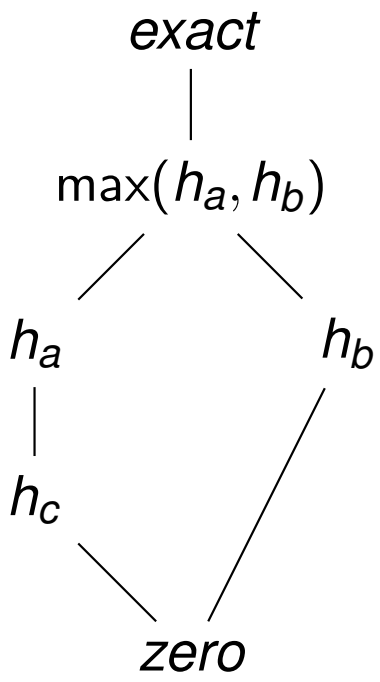
What's wrong with it?

With A*: a trade-off between quality of estimate and work per node

- As heuristics get closer to the true cost, expand fewer nodes, but more work per node to compute the heuristic itself.

Semi-Lattice of Heuristics

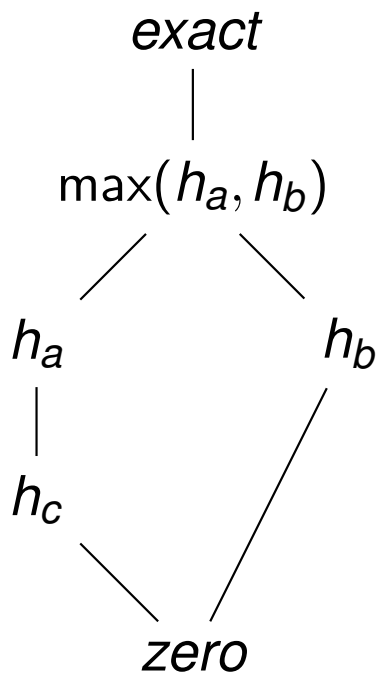
Trivial Heuristics, Dominance



Dominance: $h_a \geq h_c$ if

$$\forall n : h_a(n) \geq h_c(n)$$

Trivial Heuristics, Dominance

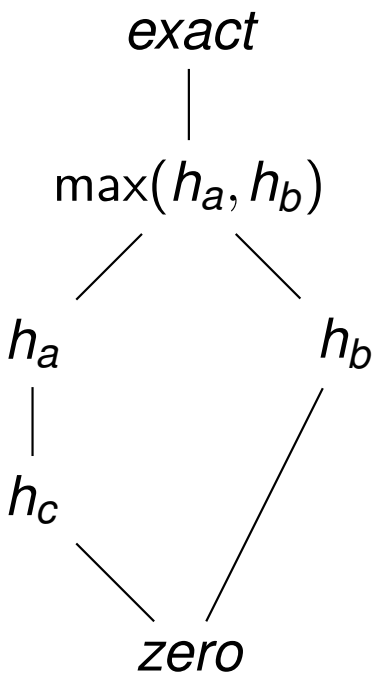


Dominance: $h_a \geq h_c$ if

$$\forall n : h_a(n) \geq h_c(n)$$

Heuristics form a semi-lattice:

Trivial Heuristics, Dominance

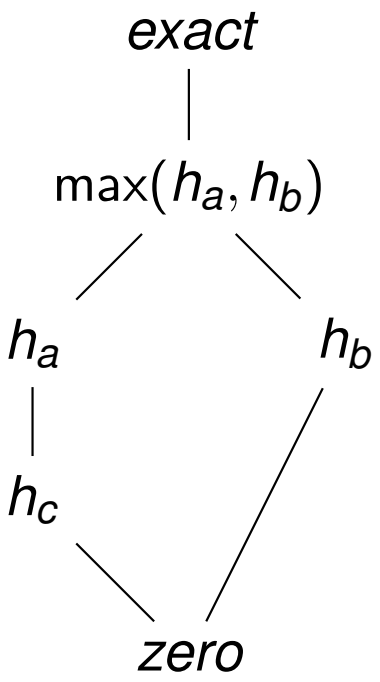


Dominance: $h_a \geq h_c$ if

$$\forall n : h_a(n) \geq h_c(n)$$

Heuristics form a semi-lattice:
Max of admissible heuristics is admissible.

Trivial Heuristics, Dominance



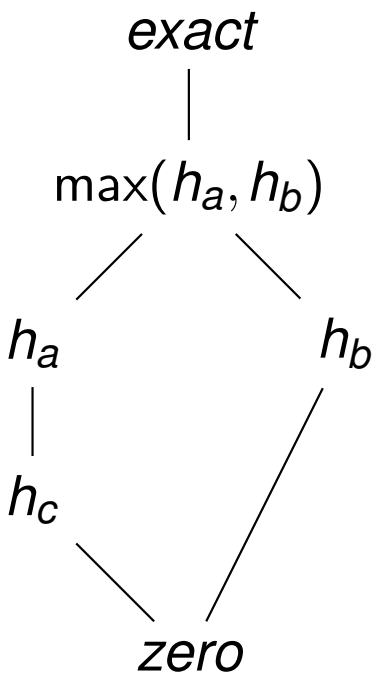
Dominance: $h_a \geq h_c$ if

$$\forall n : h_a(n) \geq h_c(n)$$

Heuristics form a semi-lattice:
Max of admissible heuristics is admissible.

Trivial heuristics

Trivial Heuristics, Dominance



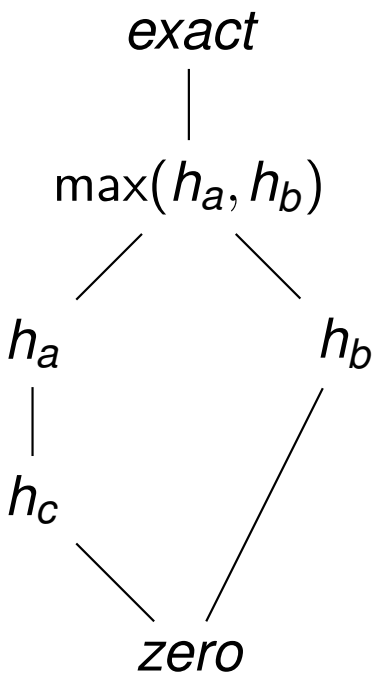
Dominance: $h_a \geq h_c$ if

$$\forall n : h_a(n) \geq h_c(n)$$

Heuristics form a semi-lattice:
Max of admissible heuristics is admissible.

Trivial heuristics
Bottom of lattice is the zero heuristic.

Trivial Heuristics, Dominance



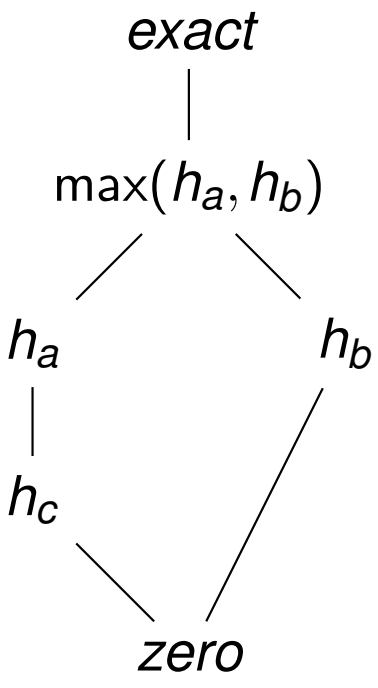
Dominance: $h_a \geq h_c$ if

$$\forall n : h_a(n) \geq h_c(n)$$

Heuristics form a semi-lattice:
Max of admissible heuristics is admissible.

Trivial heuristics
Bottom of lattice is the zero heuristic.
(what does this give us?)

Trivial Heuristics, Dominance



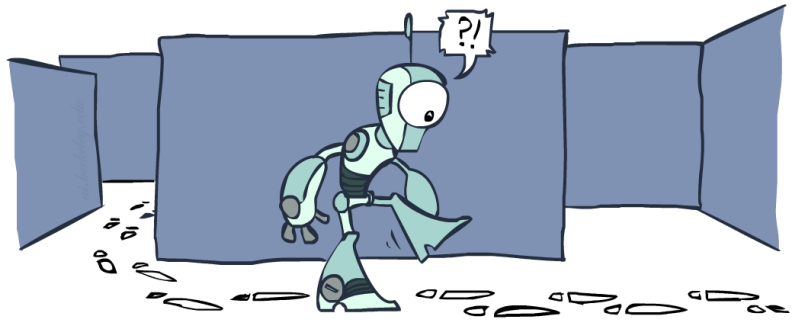
Dominance: $h_a \geq h_c$ if

$$\forall n : h_a(n) \geq h_c(n)$$

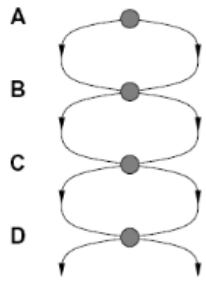
Heuristics form a semi-lattice:
Max of admissible heuristics is admissible.

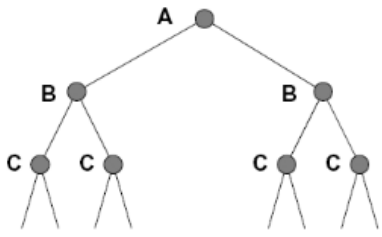
Trivial heuristics
Bottom of lattice is the zero heuristic.
(what does this give us?)
Top of lattice is the exact heuristic

Graph Search



Tree Search: Extra Work!

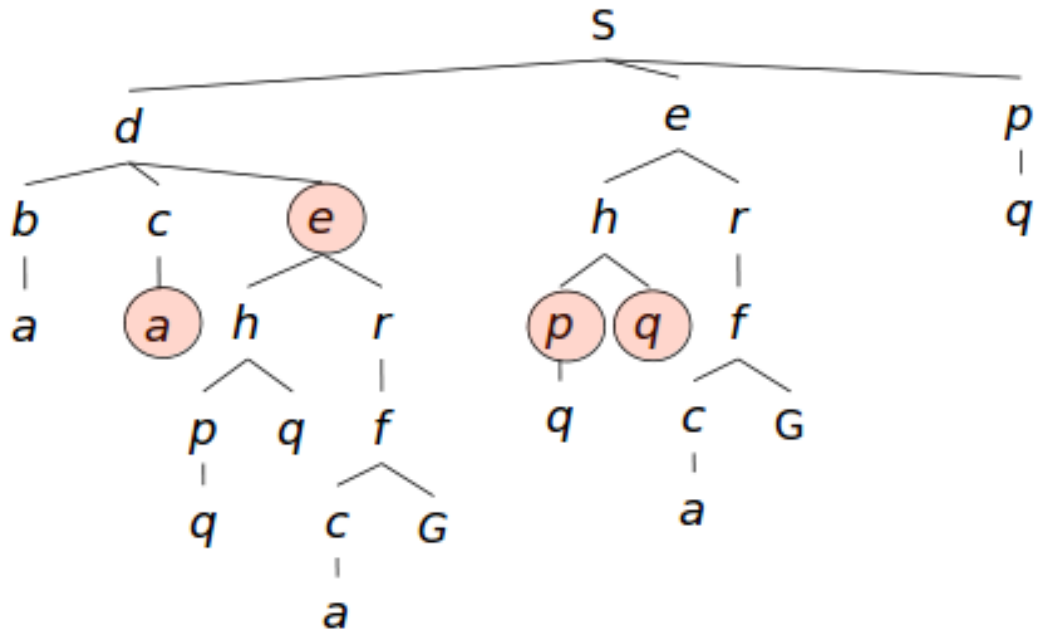




Search Tree
State Graph

Failure to detect repeated states can cause exponentially more work.

Graph Search



In BFS, for example, we shouldn't bother expanding the circled nodes (why?)

Graph Search

Idea: never expand a state twice

Graph Search

Idea: never expand a state twice

How to implement:

Graph Search

Idea: never expand a state twice

How to implement:

Tree search + set of expanded states (“closed set”)

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...

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,

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 if state was never been expanded before

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 if state was never been expanded before
If yes skip it, else add to closed set and expand.

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 if state was never been expanded before
If yes skip it, else add to closed set and expand.

Important: store the closed set as a set, not a list

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 if state was never been expanded before
If yes skip it, else add to closed set and expand.

Important: store the closed set as a set, not a list

Can graph search wreck completeness?

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 if state was never been expanded before
If yes skip it, else add to closed set and expand.

Important: store the closed set as a set, not a list

Can graph search wreck completeness? Why/why not?

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 if state was never been expanded before
If yes skip it, else add to closed set and expand.

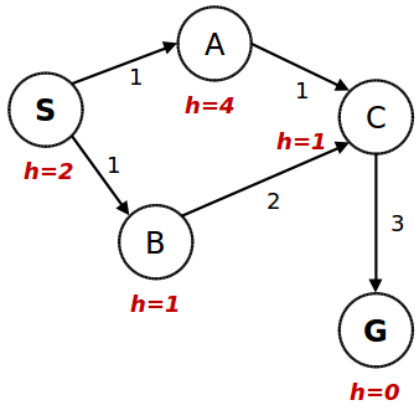
Important: store the closed set as a set, not a list

Can graph search wreck completeness? Why/why not?

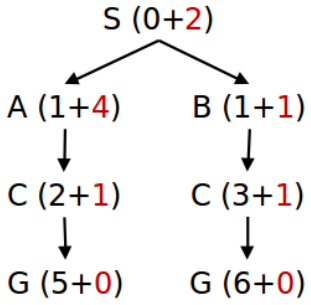
How about optimality?

A* Graph Search Gone Wrong?

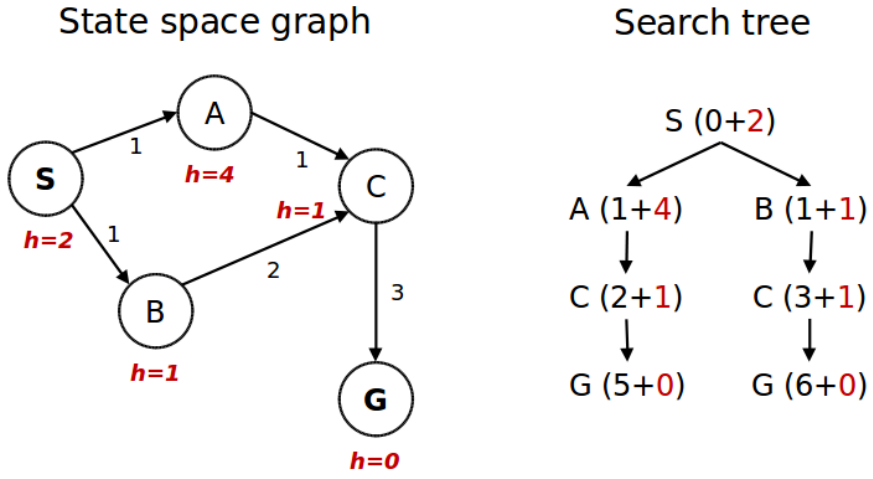
State space graph



Search tree



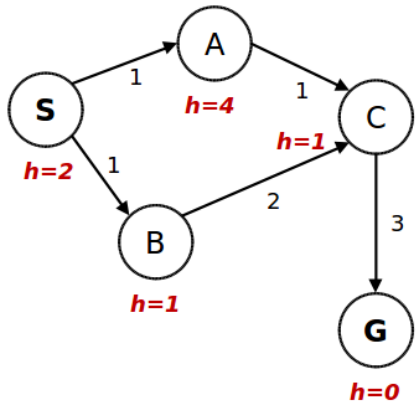
A* Graph Search Gone Wrong?



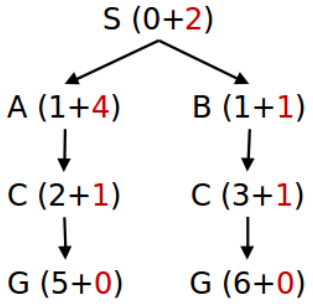
Is $h(\cdot)$ admissible?

A* Graph Search Gone Wrong?

State space graph

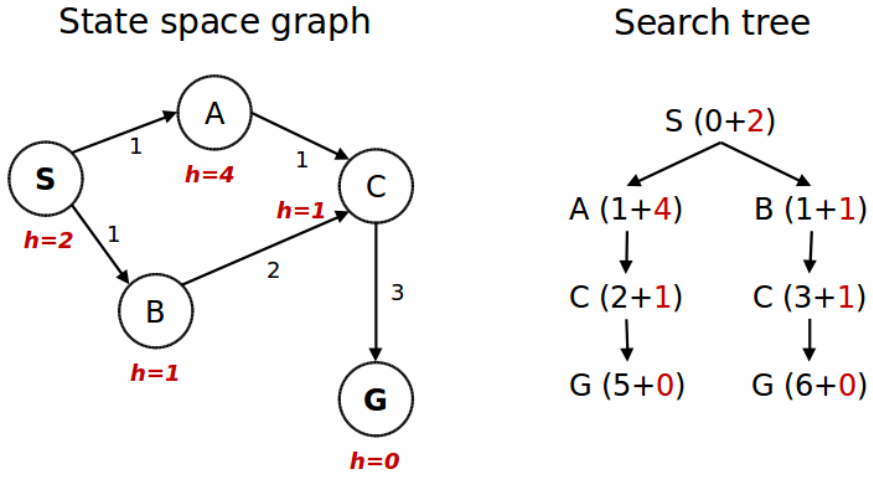


Search tree



Is $h(\cdot)$ admissible? Yes.

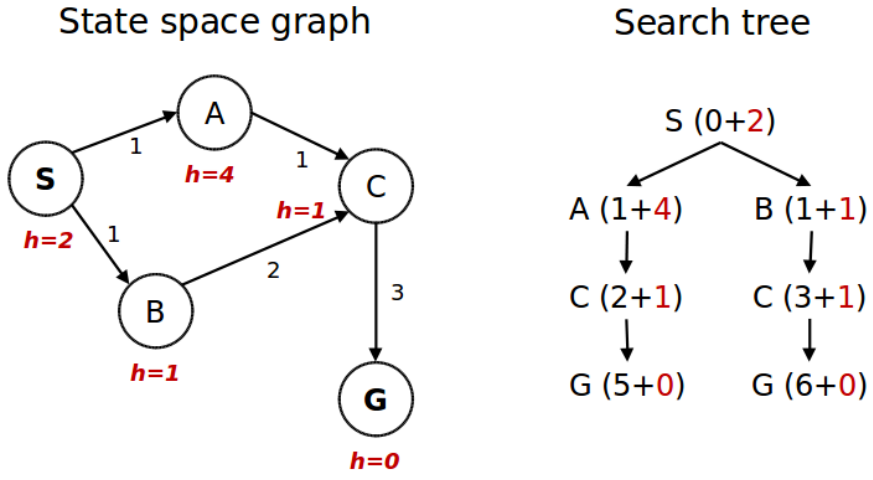
A* Graph Search Gone Wrong?



Is $h(\cdot)$ admissible? Yes.

Will exploring w.r..t $h(B) + g(n)$ be optimal?

A* Graph Search Gone Wrong?

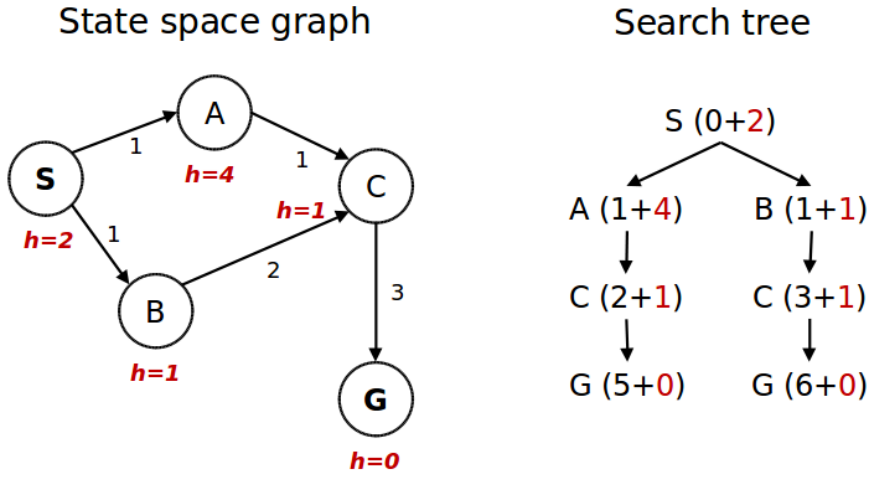


Is $h(\cdot)$ admissible? Yes.

Will exploring w.r..t $h(B) + g(n)$ be optimal?

Expand S.

A* Graph Search Gone Wrong?



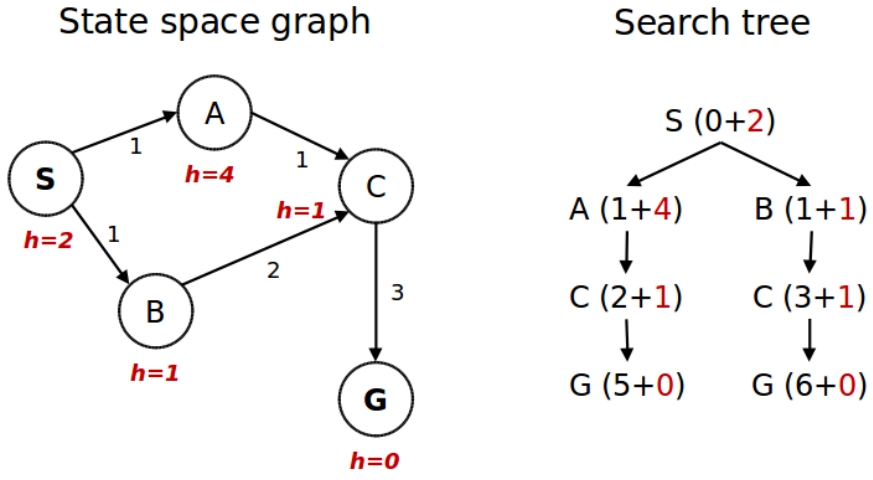
Is $h(\cdot)$ admissible? Yes.

Will exploring w.r..t $h(B) + g(n)$ be optimal?

Expand S.

A and B in fringe!

A* Graph Search Gone Wrong?



Is $h(\cdot)$ admissible? Yes.

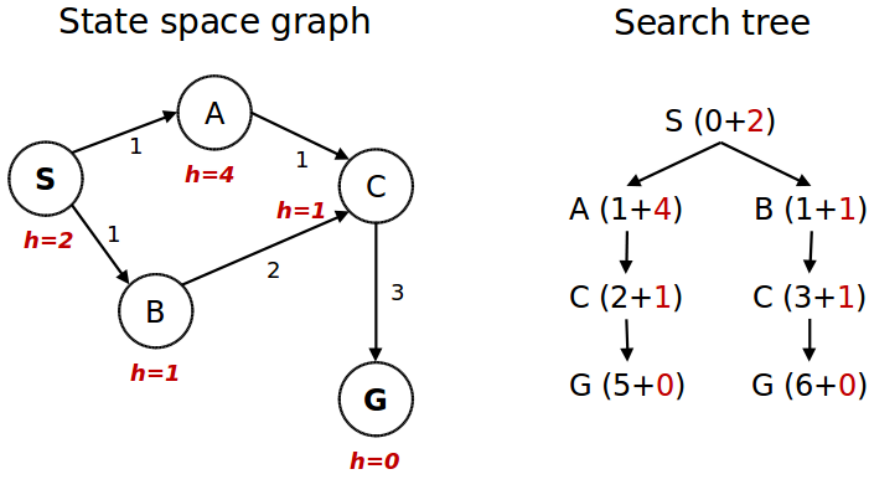
Will exploring w.r..t $h(B)+g(n)$ be optimal?

Expand S.

A and B in fringe!

Expands B, since $h(B)+g(B)=2<5=h(A)+g(A)$.

A* Graph Search Gone Wrong?



Is $h(\cdot)$ admissible? Yes.

Will exploring w.r..t $h(B)+g(n)$ be optimal?

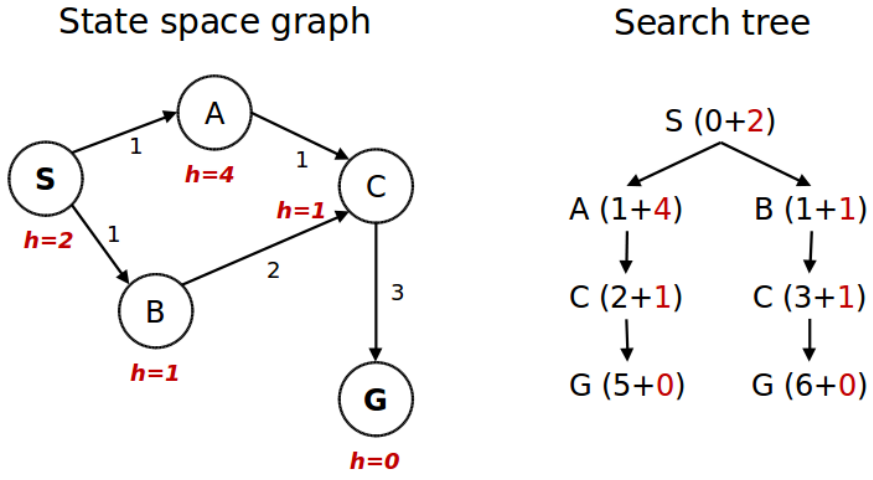
Expand S.

A and B in fringe!

Expands B, since $h(B)+g(B)=2<5=h(A)+g(A)$.

C in fringe with key, $3+h(C)=4$.

A* Graph Search Gone Wrong?



Is $h(\cdot)$ admissible? Yes.

Will exploring w.r..t $h(B)+g(n)$ be optimal?

Expand S.

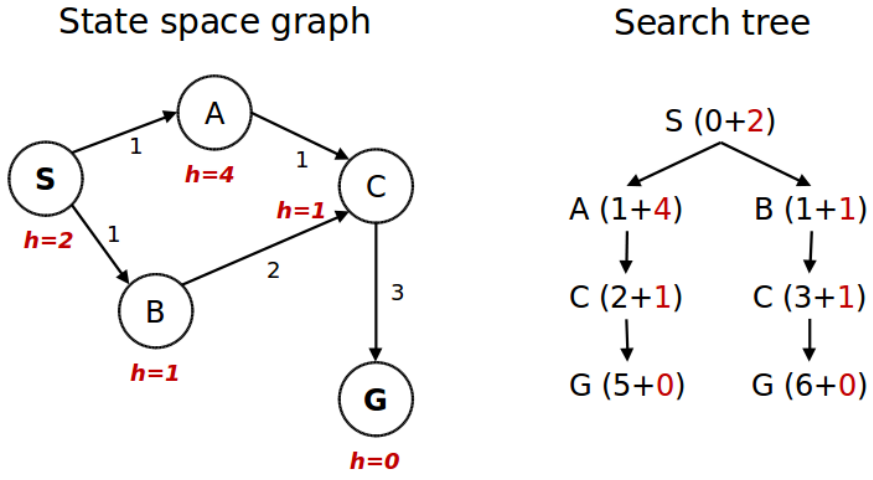
A and B in fringe!

Expands B, since $h(B)+g(B)=2<5=h(A)+g(A)$.

C in fringe with key, $3+h(C)=4$.

G in fringe with key, 5.

A* Graph Search Gone Wrong?



Is $h(\cdot)$ admissible? Yes.

Will exploring w.r..t $h(B)+g(n)$ be optimal?

Expand S.

A and B in fringe!

Expands B, since $h(B)+g(B)=2<5=h(A)+g(A)$.

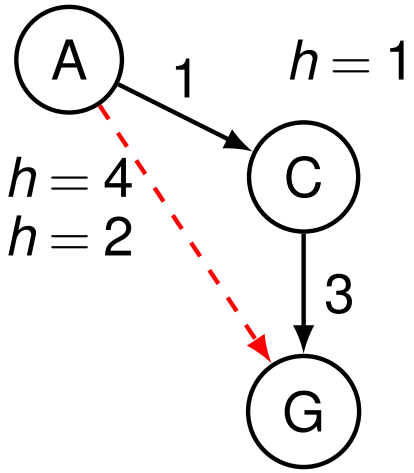
C in fringe with key, $3+h(C)=4$.

G in fringe with key, 5.

Could have been there in 4.

Consistency of Heuristics

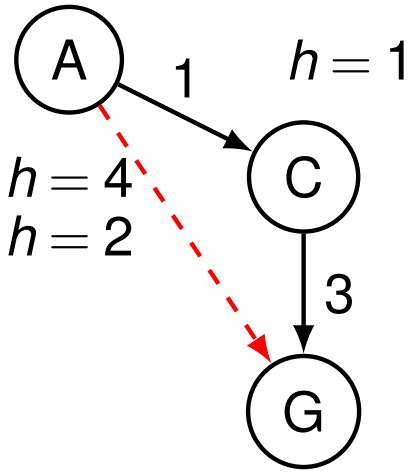
Main idea: est. heuristic costs $\leq$ actual costs



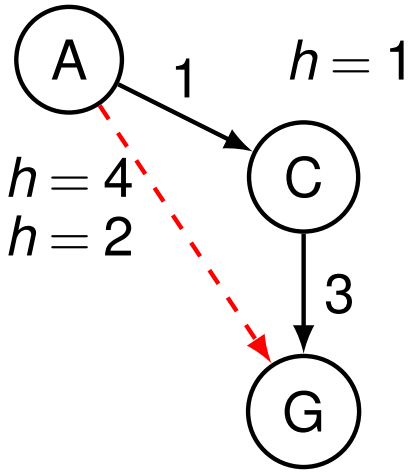
Consistency of Heuristics

Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.



Consistency of Heuristics

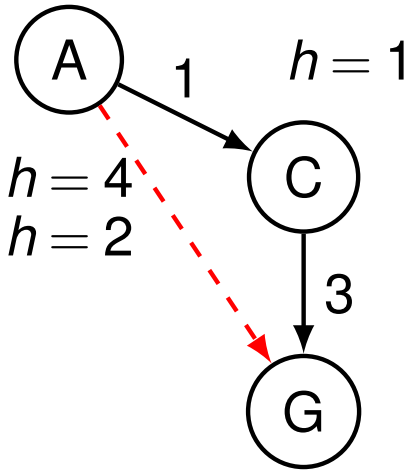


Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

Consistency of Heuristics



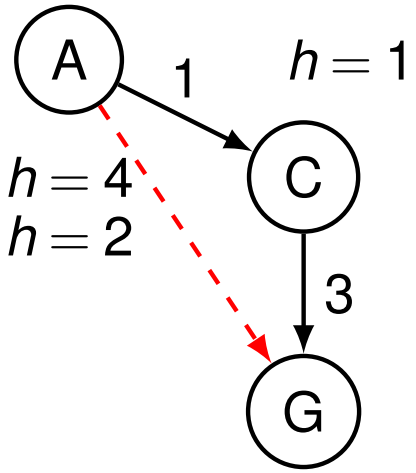
Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

Consistency of Heuristics



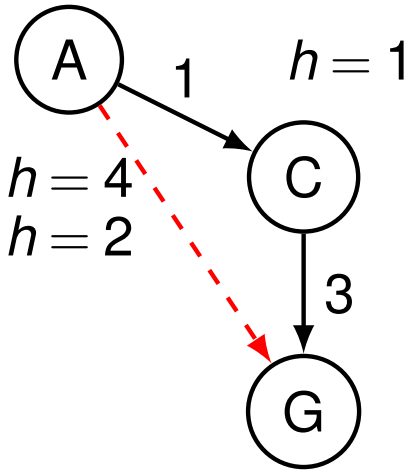
Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs

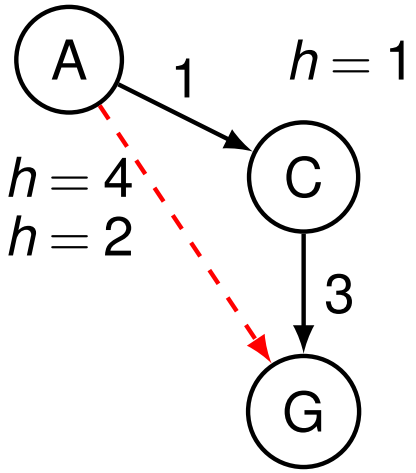
Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

Consistent $\Longrightarrow$ admissible?

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs

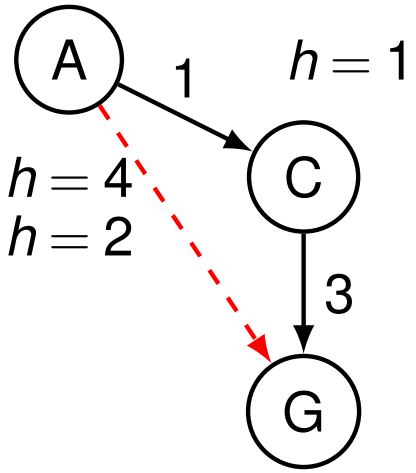
Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

Consistent $\Longrightarrow$ admissible? Yes?

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs

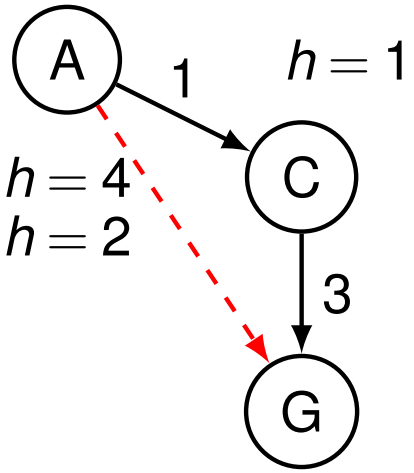
Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

Consistent $\Longrightarrow$ admissible? Yes? No?

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

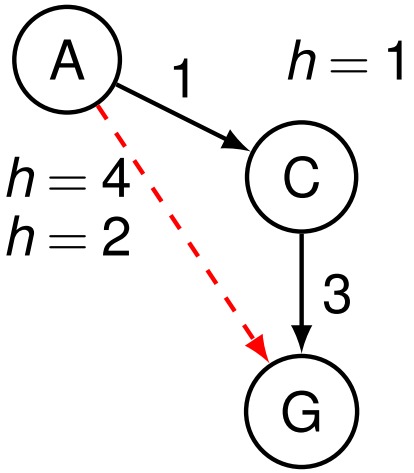
Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

Consistent $\implies$ admissible? Yes? No?

Consistent:

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

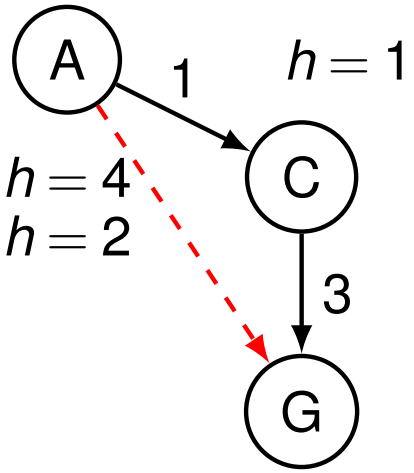
Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

Consistent $\Longrightarrow$ admissible? Yes? No?

Consistent: f value along a path never decreases

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

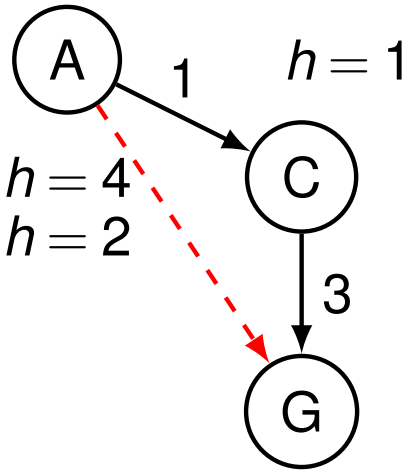
heuristic "arc" cost $\leq$ actual arc cost

Consistent $\Longrightarrow$ admissible? Yes? No?

Consistent: f value along a path never decreases

Admissible:

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

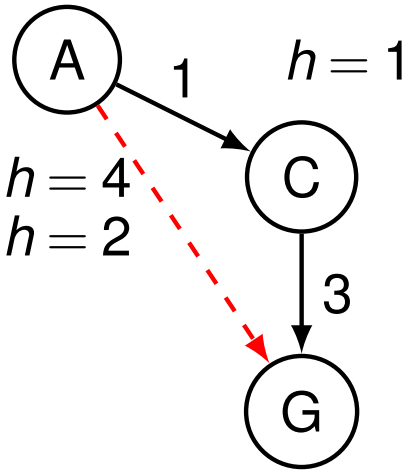
Consistent $\implies$ admissible? Yes? No?

Consistent: f value along a path never decreases

Admissible:

$f(C) = h(C) + 1 = 3$.

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

Consistent $\Longrightarrow$ admissible? Yes? No?

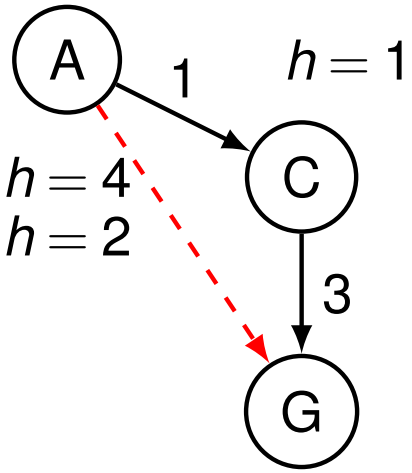
Consistent: f value along a path never decreases

Admissible:

$f(C) = h(C) + 1 = 3$.

$f(A) = h(A) = 4$.

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

Consistent $\Longrightarrow$ admissible? Yes? No?

Consistent: f value along a path never decreases

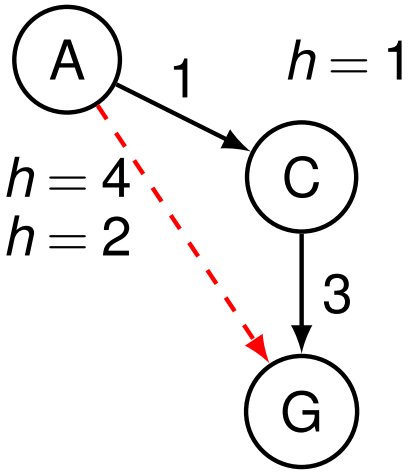
Admissible:

$f(C) = h(C) + 1 = 3$.

$f(A) = h(A) = 4$.

Consistent:

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

Consistent $\implies$ admissible? Yes? No?

Consistent: f value along a path never decreases

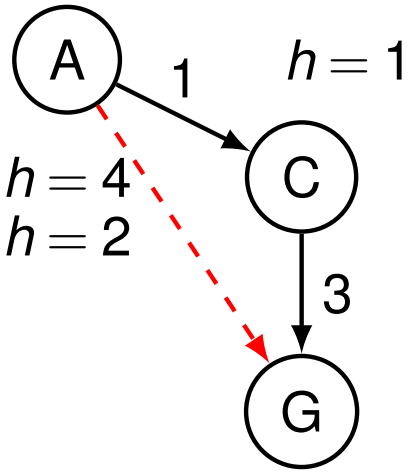
Admissible:

$f(C) = h(C) + 1 = 3$.

$f(A) = h(A) = 4$.

Consistent: $f(A) = 2$

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

Consistent $\implies$ admissible? Yes? No?

Consistent: f value along a path never decreases

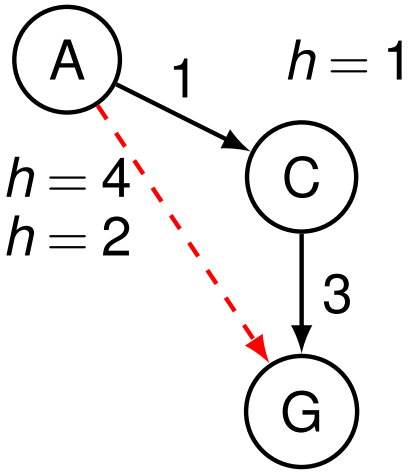
Admissible:

$f(C) = h(C) + 1 = 3$.

$f(A) = h(A) = 4$.

Consistent: $f(A) = 2 < 3 = f(C)$.

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

Consistent $\implies$ admissible? Yes? No?

Consistent: f value along a path never decreases

Admissible:

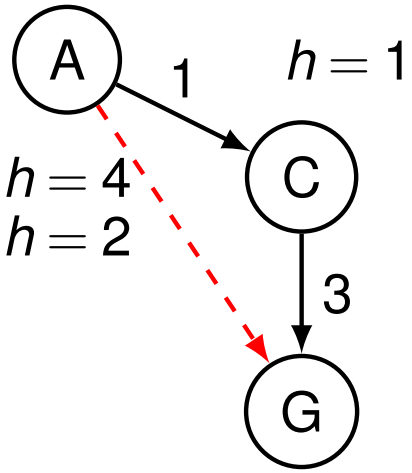
$f(C) = h(C) + 1 = 3$.

$f(A) = h(A) = 4$.

Consistent: $f(A) = 2 < 3 = f(C)$.

Claim: If y is expanded due to x, $f(y) \geq f(x)$.

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

Consistent $\implies$ admissible? Yes? No?

Consistent: f value along a path never decreases

Admissible:

$f(C) = h(C) + 1 = 3$.

$f(A) = h(A) = 4$.

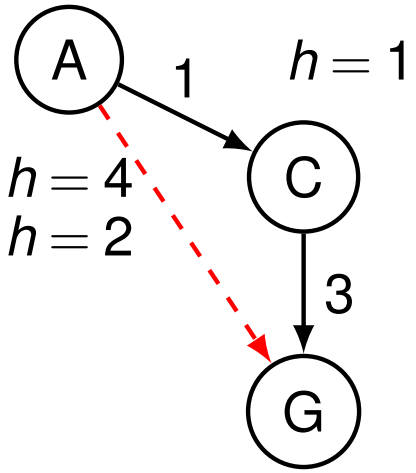
Consistent: $f(A) = 2 < 3 = f(C)$.

Claim: If y is expanded due to x, $f(y) \geq f(x)$.

Proof:

$f(y) = g(x) + cost(x, y) + h(y)$

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

Consistent $\implies$ admissible? Yes? No?

Consistent: f value along a path never decreases

Admissible:

$f(C) = h(C) + 1 = 3$.

$f(A) = h(A) = 4$.

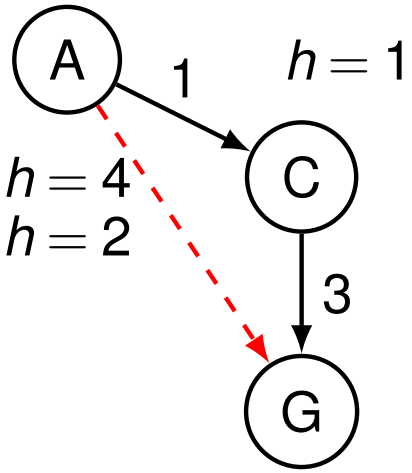
Consistent: $f(A) = 2 < 3 = f(C)$.

Claim: If y is expanded due to x, $f(y) \geq f(x)$.

Proof:

$$f(y) = g(x) + cost(x, y) + h(y)$$
$$\geq g(x) + h(x) - h(y) + h(y)$$

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

Consistent $\implies$ admissible? Yes? No?

Consistent: f value along a path never decreases

Admissible:

$f(C) = h(C) + 1 = 3$.

$f(A) = h(A) = 4$.

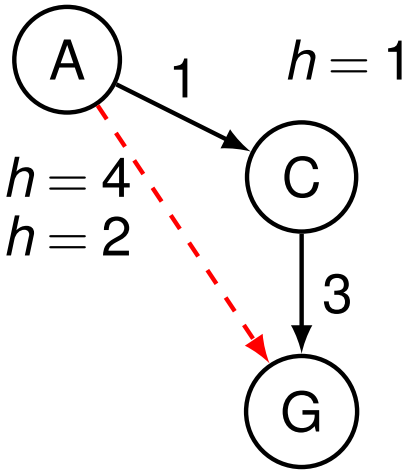
Consistent: $f(A) = 2 < 3 = f(C)$.

Claim: If y is expanded due to x, $f(y) \geq f(x)$.

Proof:

$$f(y) = g(x) + cost(x, y) + h(y)$$
$$\geq g(x) + h(x) - h(y) + h(y) = g(x) + h(x)$$

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

Consistent $\implies$ admissible? Yes? No?

Consistent: f value along a path never decreases

Admissible:

$f(C) = h(C) + 1 = 3$.

$f(A) = h(A) = 4$.

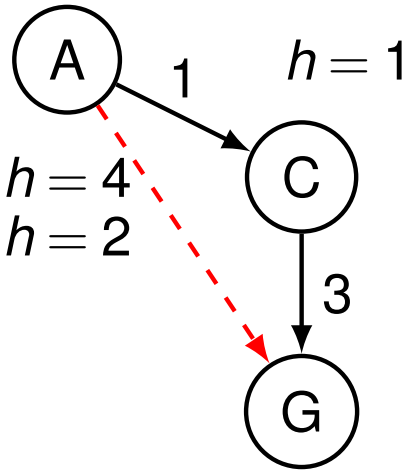
Consistent: $f(A) = 2 < 3 = f(C)$.

Claim: If y is expanded due to x, $f(y) \geq f(x)$.

Proof:

$$f(y) = g(x) + cost(x, y) + h(y)$$
$$\geq g(x) + h(x) - h(y) + h(y) = g(x) + h(x) = f(x)$$

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

Consistent $\implies$ admissible? Yes? No?

Consistent: f value along a path never decreases

Admissible:

$f(C) = h(C) + 1 = 3$.

$f(A) = h(A) = 4$.

Consistent: $f(A) = 2 < 3 = f(C)$.

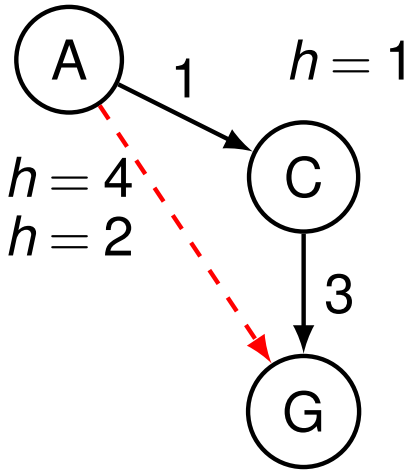
Claim: If y is expanded due to x, $f(y) \geq f(x)$.

Proof:

$$f(y) = g(x) + cost(x, y) + h(y)$$
$$\geq g(x) + h(x) - h(y) + h(y) = g(x) + h(x) = f(x)$$

□

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs

Admissibility: $h(x) \leq$ cost to goal.

Consistency: $h(x) - h(y) \leq cost(x, y)$.

heuristic "arc" cost $\leq$ actual arc cost

Consistent $\implies$ admissible? Yes? No?

Consistent: f value along a path never decreases

Admissible:

$f(C) = h(C) + 1 = 3$.

$f(A) = h(A) = 4$.

Consistent: $f(A) = 2 < 3 = f(C)$.

Claim: If y is expanded due to x, $f(y) \geq f(x)$.

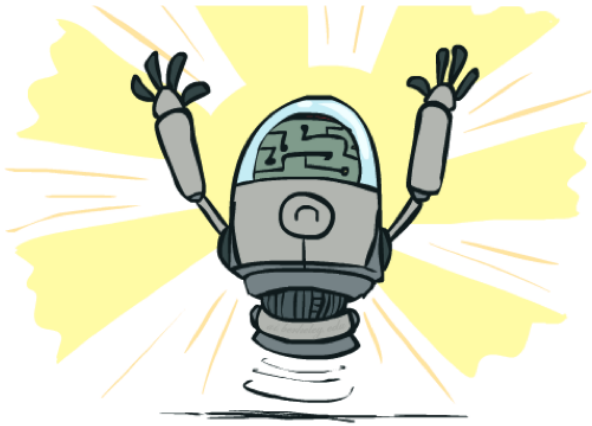
Proof:

$$f(y) = g(x) + cost(x, y) + h(y)$$
$$\geq g(x) + h(x) - h(y) + h(y) = g(x) + h(x) = f(x)$$

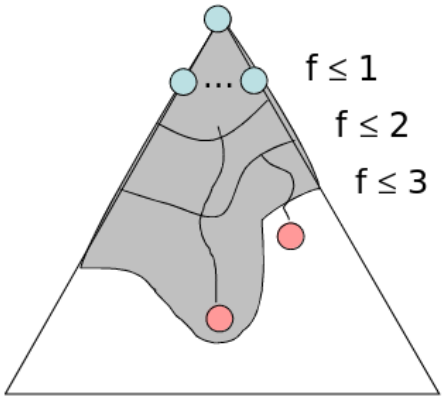
□

The "estimate" of plan cost keeps rising as you progress.

Optimality of A* Graph Search

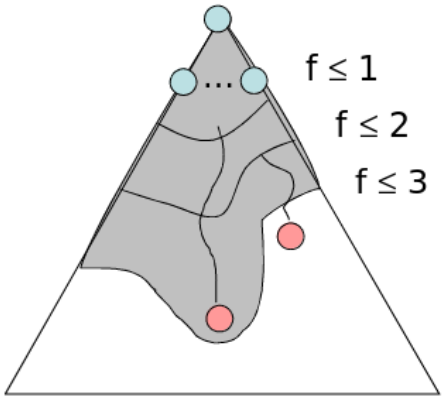


Optimality of A* Graph Search



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

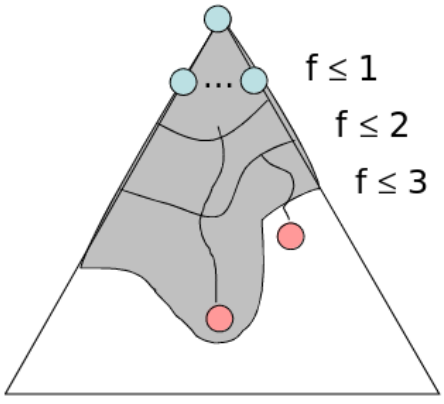
Optimality of A* Graph Search



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

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

Optimality of A* Graph Search

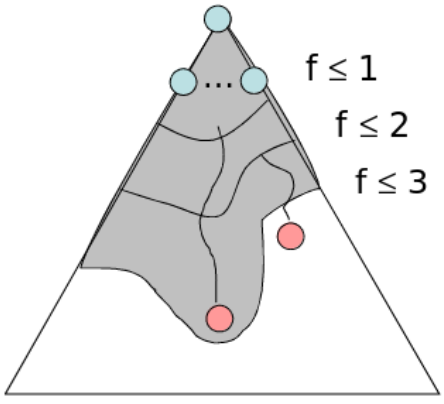


Sketch: consider what A* does with a consistent heuristic:

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

Fact 2: For every state *s*, the optimal path is discovered.

Optimality of A* Graph Search



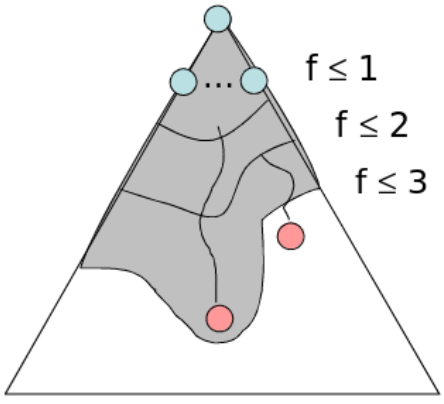
Sketch: consider what A* does with a consistent heuristic:

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

Fact 2: For every state *s*, the optimal path is discovered.

Result: A* graph search is optimal

Optimality of A* Graph Search



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

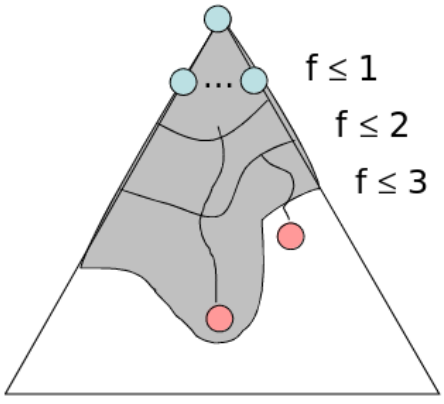
Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)

Fact 2: For every state *s*, the optimal path is discovered.

Result: A* graph search is optimal

Fact 1 Proof. Previous slide.

Optimality of A* Graph Search



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

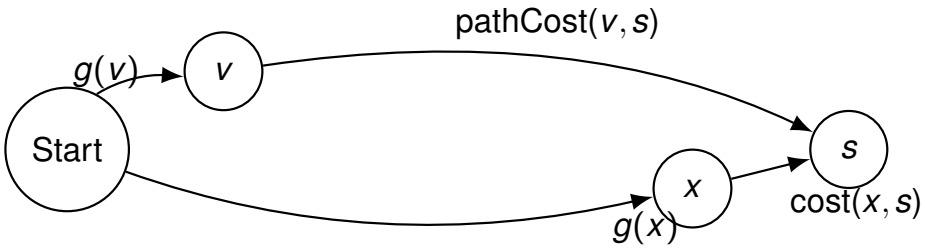
Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)

Fact 2: For every state *s*, the optimal path is discovered.

Result: A* graph search is optimal

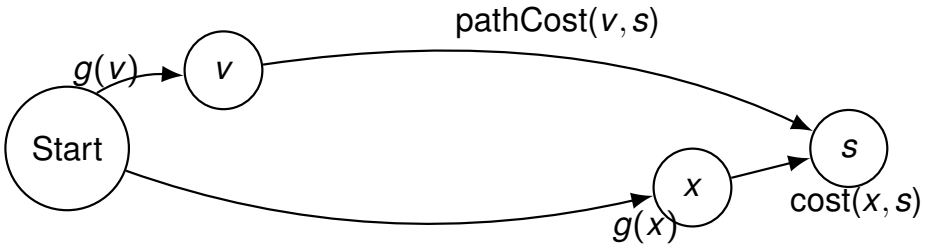
Fact 1 Proof. Previous slide.

Proof of A* optimality.



Fact 2: The optimal path is discovered to every state s.

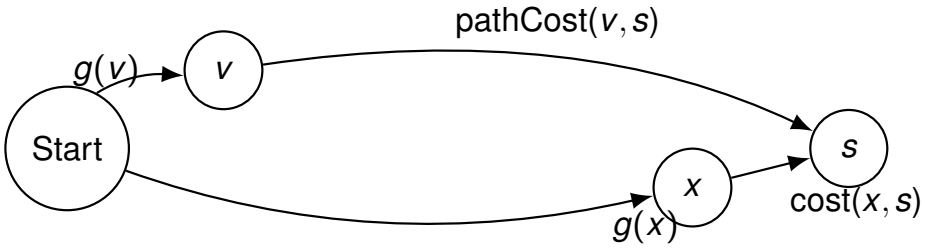
Proof of A* optimality.



Fact 2: The optimal path is discovered to every state s.

Proof: Consider first error.

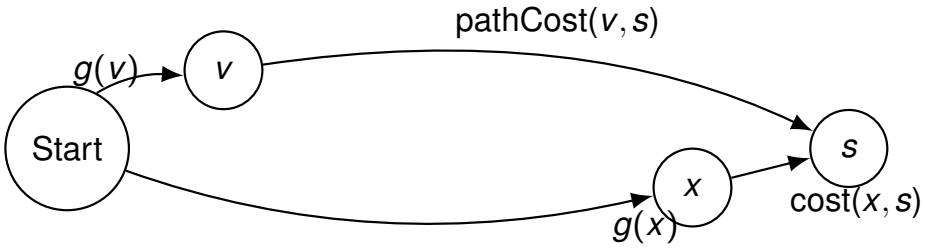
Proof of A* optimality.



Fact 2: The optimal path is discovered to every state s.

Proof: Consider first error.
State s discovered from x.

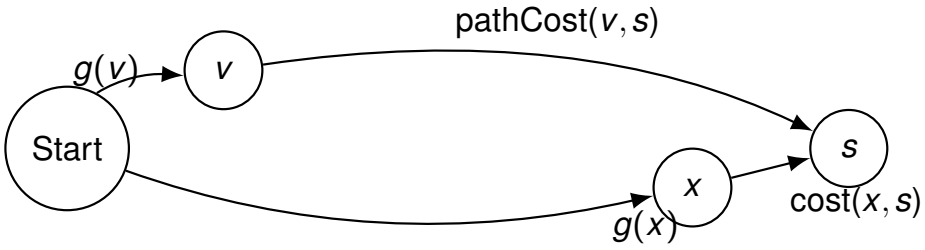
Proof of A* optimality.



Fact 2: The optimal path is discovered to every state s.

Proof: Consider first error.
State s discovered from x.
Optimal path is from $y \neq x$.

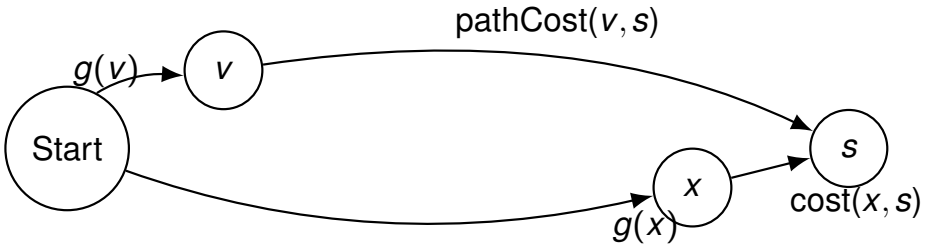
Proof of A* optimality.



Fact 2: The optimal path is discovered to every state s.

Proof: Consider first error.
State s discovered from x.
Optimal path is from $y \neq x$.
There is a vertex v in the optimal path to y in fringe.

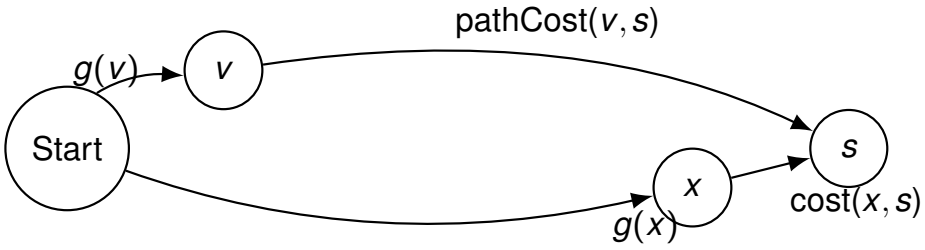
Proof of A* optimality.



Fact 2: The optimal path is discovered to every state s.

Proof: Consider first error.
State s discovered from x.
Optimal path is from $y \neq x$.
There is a vertex v in the optimal path to y in fringe.
s in fringe with key $f(s) = g(x) + cost(x, s) + h(s)$.

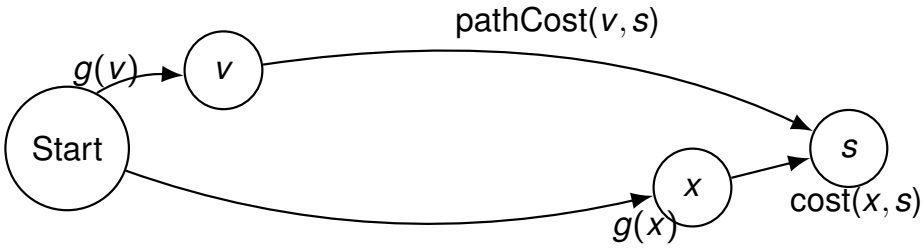
Proof of A* optimality.



Fact 2: The optimal path is discovered to every state s.

Proof: Consider first error.
State s discovered from x.
Optimal path is from $y \neq x$.
There is a vertex v in the optimal path to y in fringe.
s in fringe with key $f(s) = g(x) + cost(x, s) + h(s)$.
v in fringe with key $f(v) = g(v) + h(v)$.

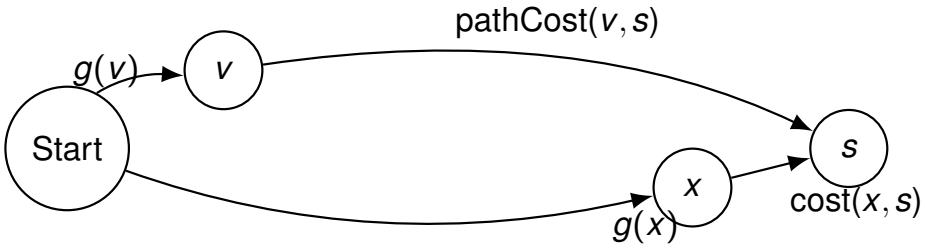
Proof of A* optimality.



Fact 2: The optimal path is discovered to every state s.

Proof: Consider first error.
State s discovered from x.
Optimal path is from $y \neq x$.
There is a vertex v in the optimal path to y in fringe.
s in fringe with key $f(s) = g(x) + cost(x, s) + h(s)$.
v in fringe with key $f(v) = g(v) + h(v)$.
$h(v) - h(s) \leq pathCost(v, s)$ by induction.

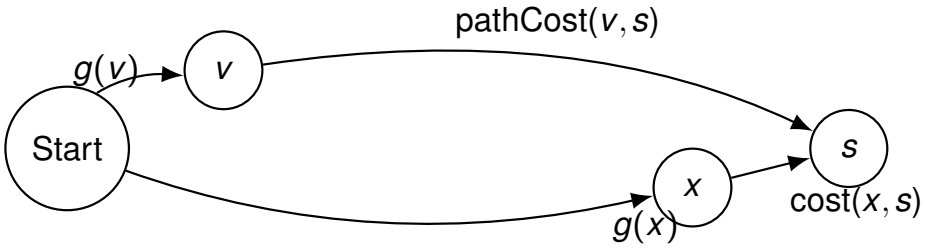
Proof of A* optimality.



Fact 2: The optimal path is discovered to every state s.

Proof: Consider first error.
State s discovered from x.
Optimal path is from $y \neq x$.
There is a vertex v in the optimal path to y in fringe.
s in fringe with key $f(s) = g(x) + cost(x, s) + h(s)$.
v in fringe with key $f(v) = g(v) + h(v)$.
$h(v) - h(s) \leq pathCost(v, s)$ by induction.
$g(v) + pathCost(v, s) < g(x) + cost(x, s)$

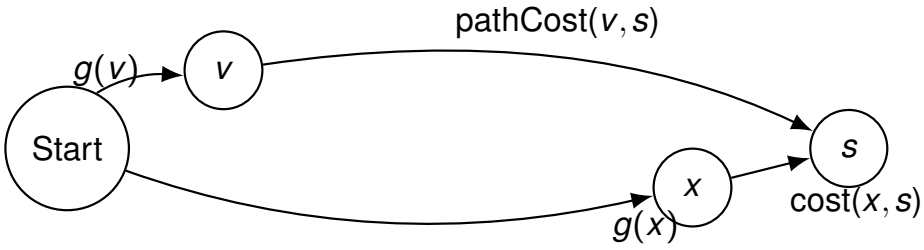
Proof of A* optimality.



Fact 2: The optimal path is discovered to every state s.

Proof: Consider first error.
State s discovered from x.
Optimal path is from $y \neq x$.
There is a vertex v in the optimal path to y in fringe.
s in fringe with key $f(s) = g(x) + cost(x, s) + h(s)$.
v in fringe with key $f(v) = g(v) + h(v)$.
$h(v) - h(s) \leq pathCost(v, s)$ by induction.
$g(v) + pathCost(v, s) < g(x) + cost(x, s)$
$\implies f(v) < f(s)$.

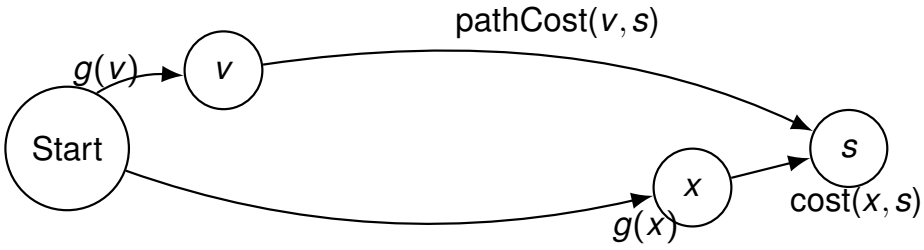
Proof of A* optimality.



Fact 2: The optimal path is discovered to every state s.

Proof: Consider first error.
State s discovered from x.
Optimal path is from $y \neq x$.
There is a vertex v in the optimal path to y in fringe.
s in fringe with key $f(s) = g(x) + cost(x, s) + h(s)$.
v in fringe with key $f(v) = g(v) + h(v)$.
$h(v) - h(s) \leq pathCost(v, s)$ by induction.
$g(v) + pathCost(v, s) < g(x) + cost(x, s)$
$\Longrightarrow$ $f(v) < f(s)$.
But then v would have been expanded before s!

Proof of A* optimality.



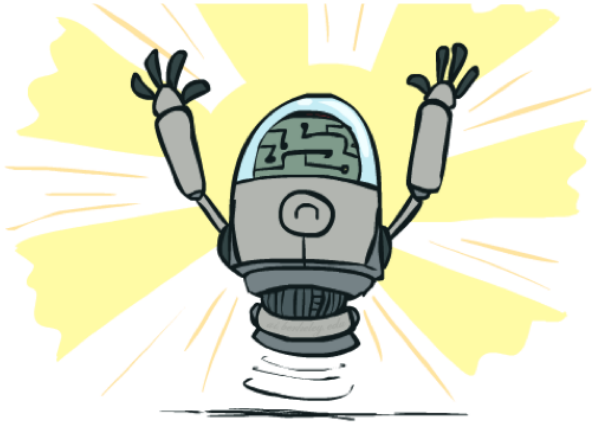
Fact 2: The optimal path is discovered to every state s.

Proof: Consider first error.
State s discovered from x.
Optimal path is from $y \neq x$.
There is a vertex v in the optimal path to y in fringe.
s in fringe with key $f(s) = g(x) + cost(x, s) + h(s)$.
v in fringe with key $f(v) = g(v) + h(v)$.
$h(v) - h(s) \leq pathCost(v, s)$ by induction.
$g(v) + pathCost(v, s) < g(x) + cost(x, s)$
$\Longrightarrow$ $f(v) < f(s)$.
But then v would have been expanded before s! □

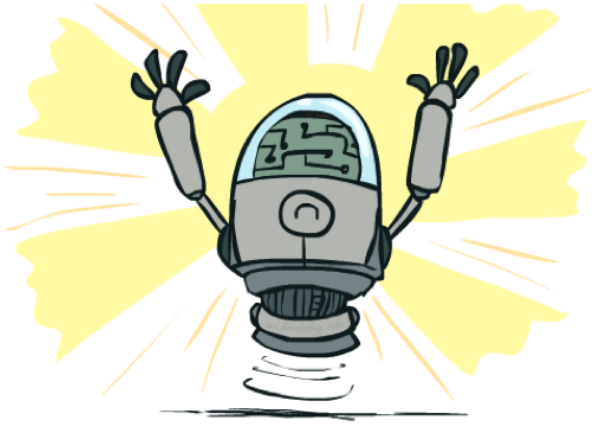
Optimality

Tree search:

A* is optimal if heuristic is admissible.



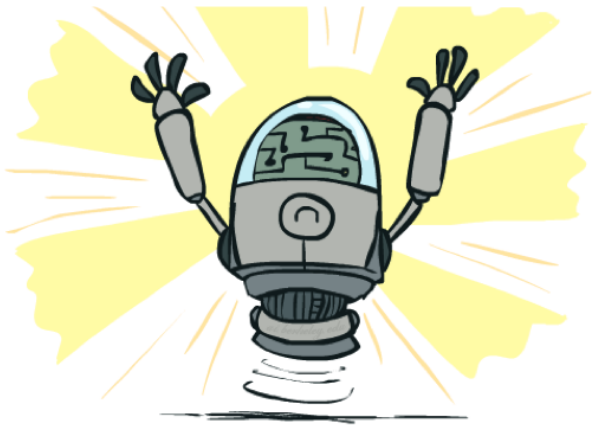
Optimality



Tree search:

- A* is optimal if heuristic is admissible.
- UCS is a special case (h = 0)

Optimality

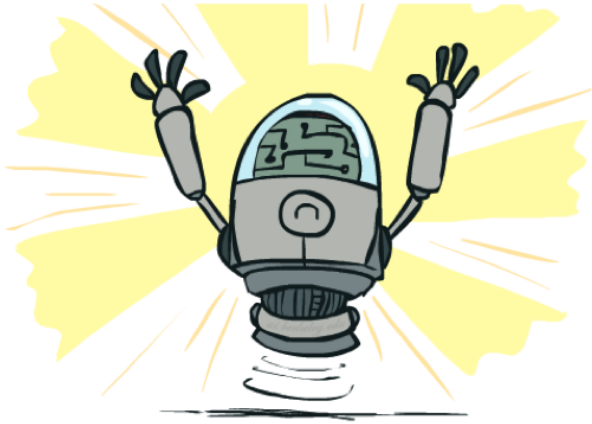


Tree search:

- A* is optimal if heuristic is admissible.
- UCS is a special case (h = 0)

Graph search:

Optimality



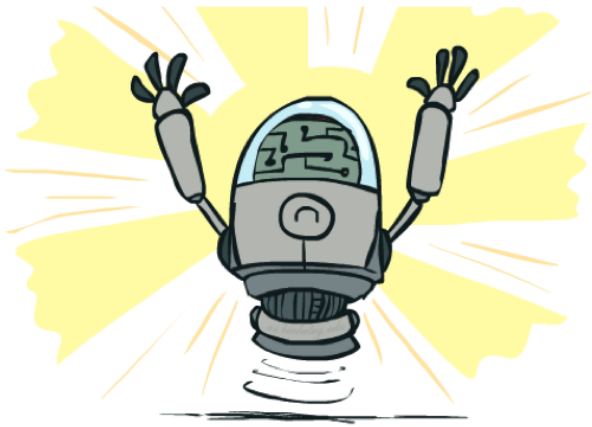
Tree search:

- A* is optimal if heuristic is admissible.
- UCS is a special case (h = 0)

Graph search:

- A* optimal if heuristic is consistent

Optimality



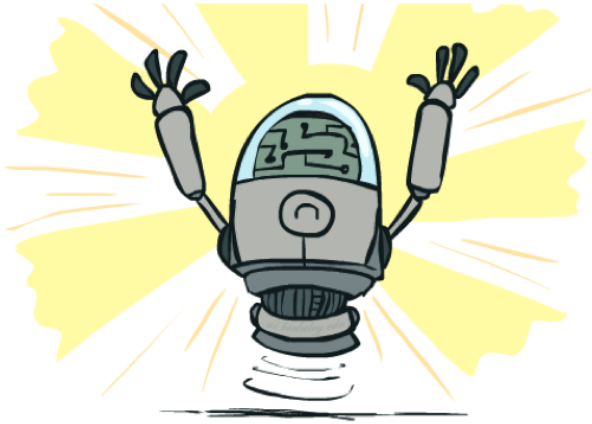
Tree search:

- 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)

Optimality



Tree search:

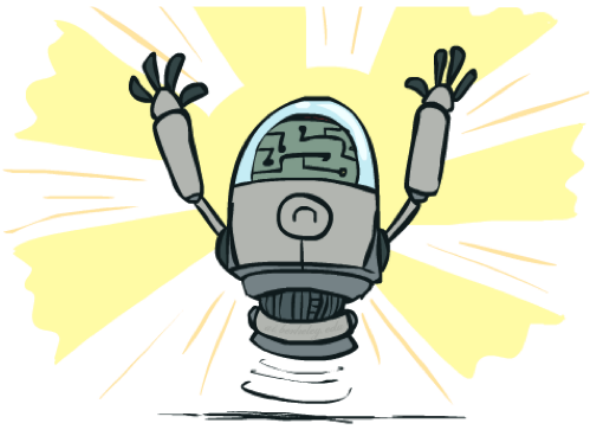
- A* is optimal if heuristic is admissible.
- UCS is a special case (h = 0)

Graph search:

- A* optimal if heuristic is consistent
- UCS optimal (h = 0 is consistent)

Consistency implies admissibility

Optimality



Tree search:

- A* is optimal if heuristic is admissible.
- UCS is a special case (h = 0)

Graph search:

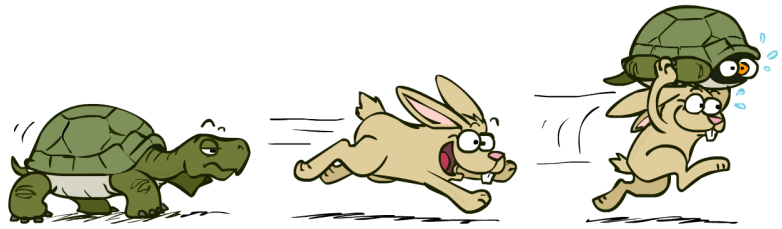
- A* optimal if heuristic is consistent
- UCS optimal (h = 0 is consistent)

Consistency implies admissibility

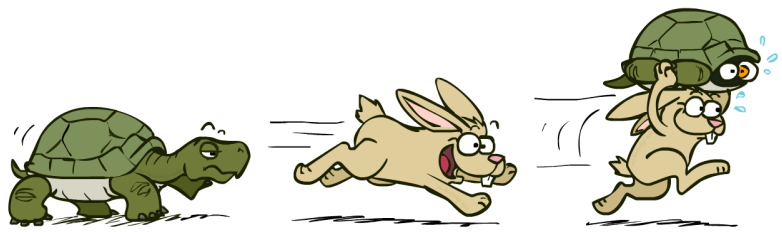
In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems

A*: Summary

A*: Summary

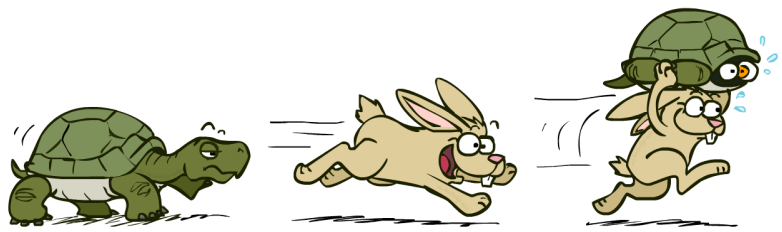


A*: Summary



A* uses both backward costs and (estimates of) forward costs

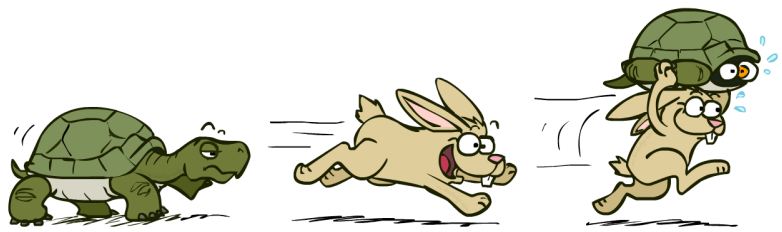
A*: Summary



A* uses both backward costs and (estimates of) forward costs

A* is optimal with admissible / consistent heuristics

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

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
```

Yaay!