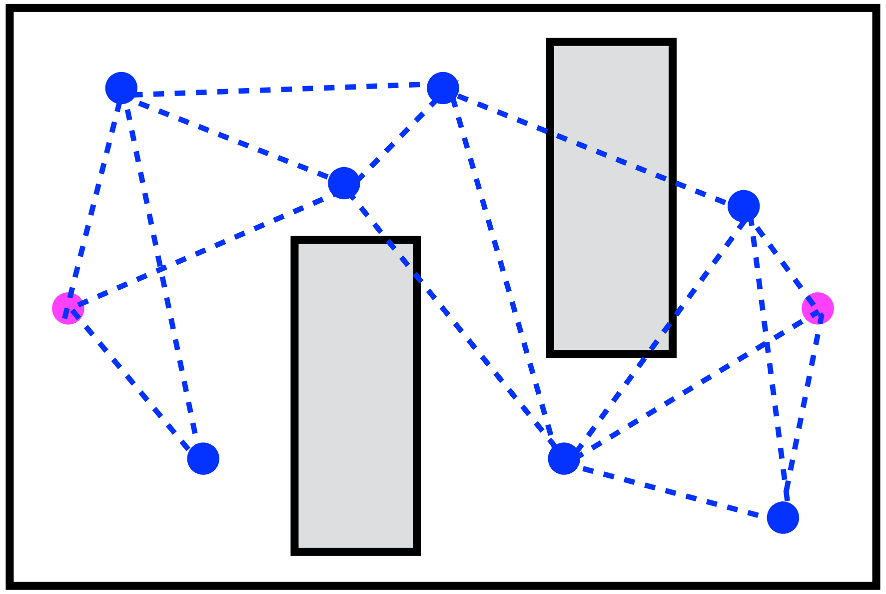
Heuristic Search

Sanjiban Choudhury

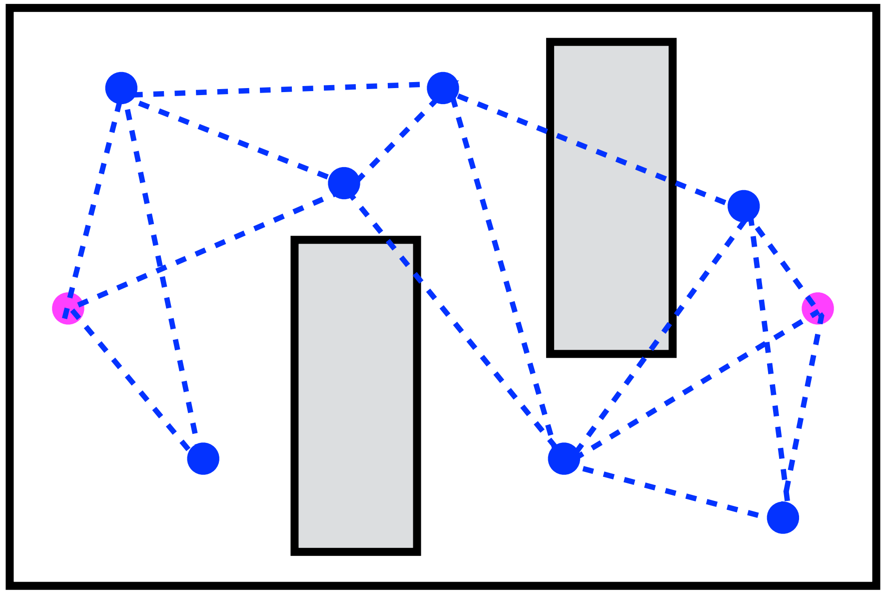
TAs: Matthew Rockett, Gilwoo Lee, Matt Schmittle

General framework for motion planning

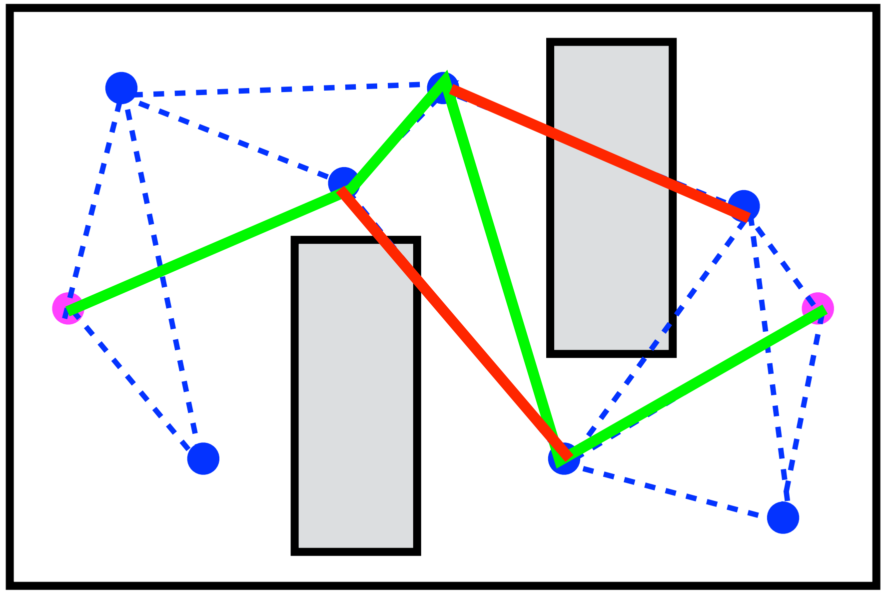


Create a graph

General framework for motion planning

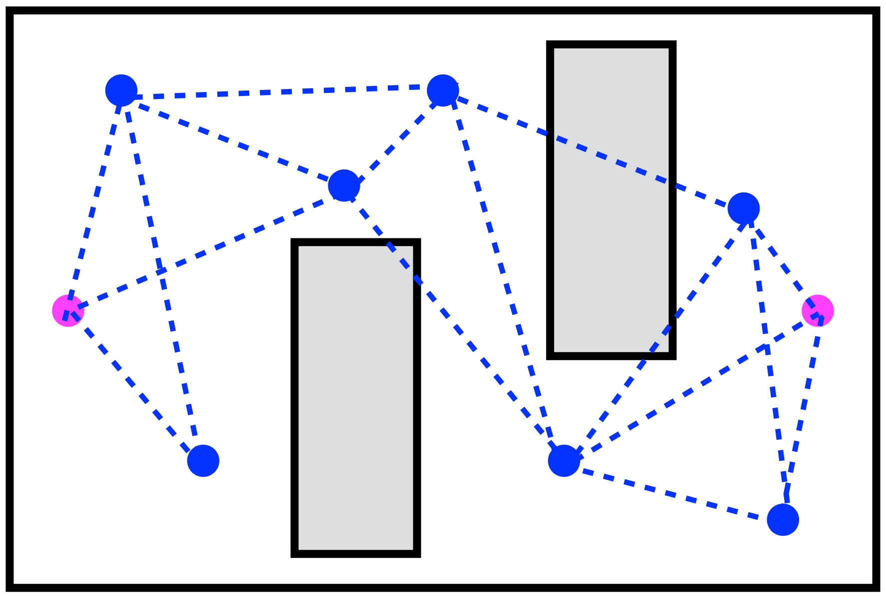


Create a graph

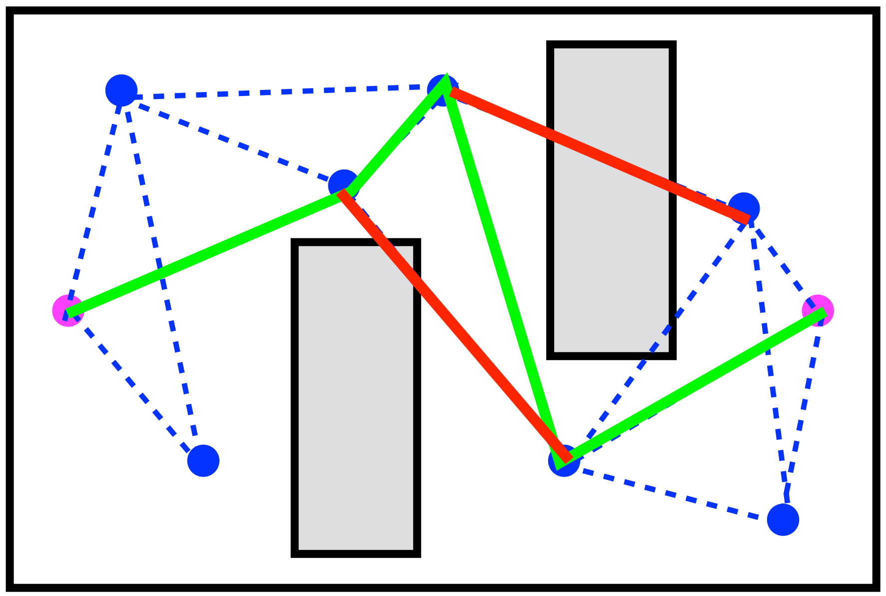


Search the graph

General framework for motion planning



Create a graph



Search the graph

Interleave

General framework for motion planning

Any planning algorithm	Create graph	Search graph	Interleave

General framework for motion planning

Any planning algorithm

Create graph

Search graph

Interleave

General framework for motion planning

Any planning algorithm

Create graph

Search graph

Interleave

=

General framework for motion planning

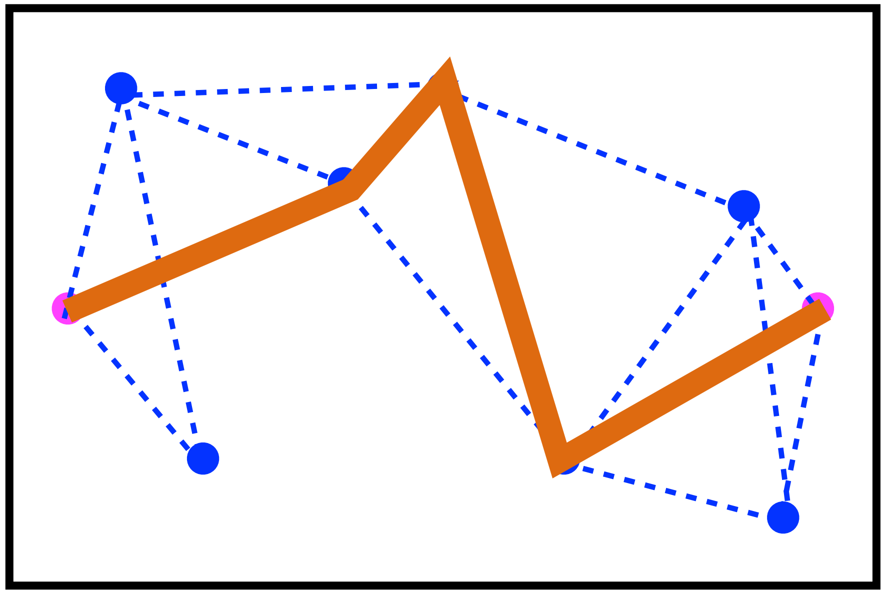
Any planning algorithm		Create graph		Search graph		Interleave
	=	e.g. fancy random sampler	×	e.g. fancy heuristic	×	e.g. fancy way of densifying

General framework for motion planning

Any planning algorithm		Create graph		Search graph		Interleave
	=	e.g. fancy random sampler	×	e.g. fancy heuristic	×	e.g. fancy way of densifying
		Whats the best we can do?		Whats the best we can do?		Whats the best we can do?

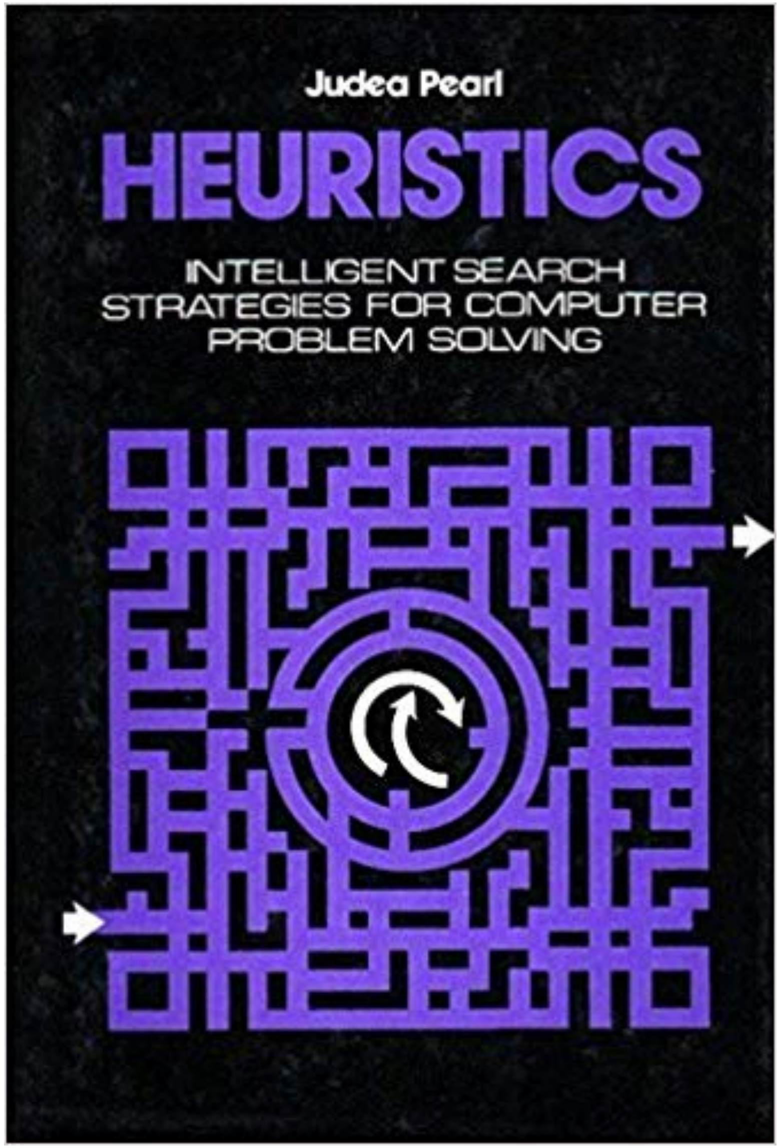
For this lecture....

We will focus on the search assuming everything we need is given



Optimal Path = SHORTESTPATH(V,E, start, goal)

If you are serious about heuristic search



This lecture:

Skewed view of search

that will be helpful for robot motion planning

Today's objective

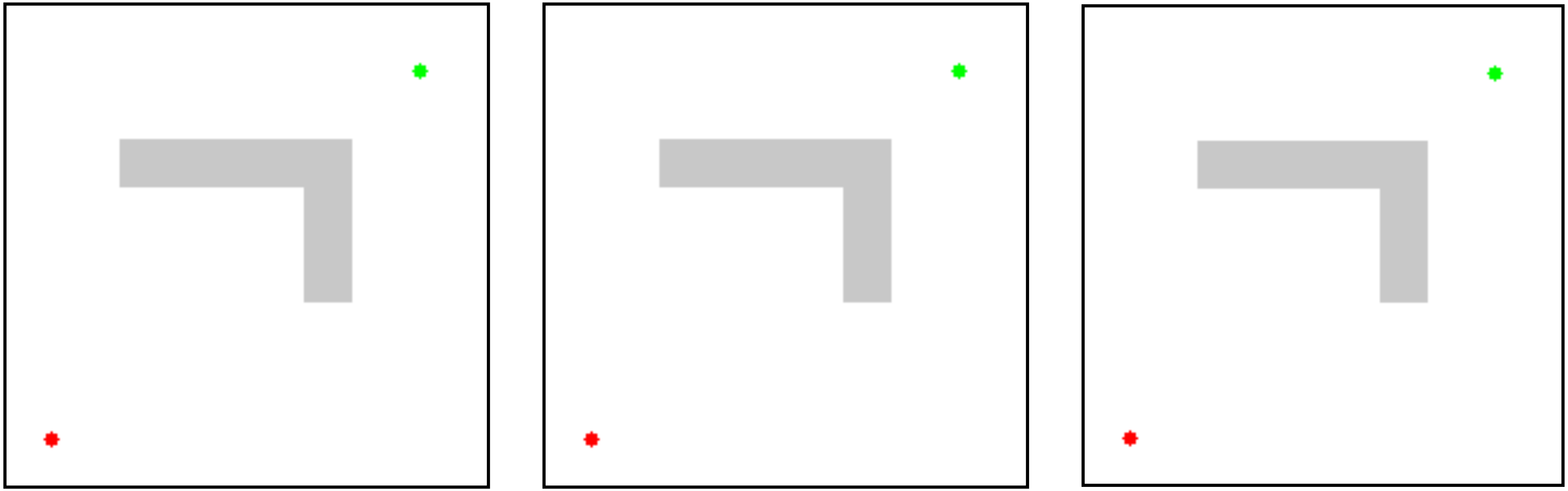
1. Best first search as a meta-algorithm

2. Heuristic search and what we want from it

3. Laziness in search

High-order bit

Expansion of a search wavefront from start to goal

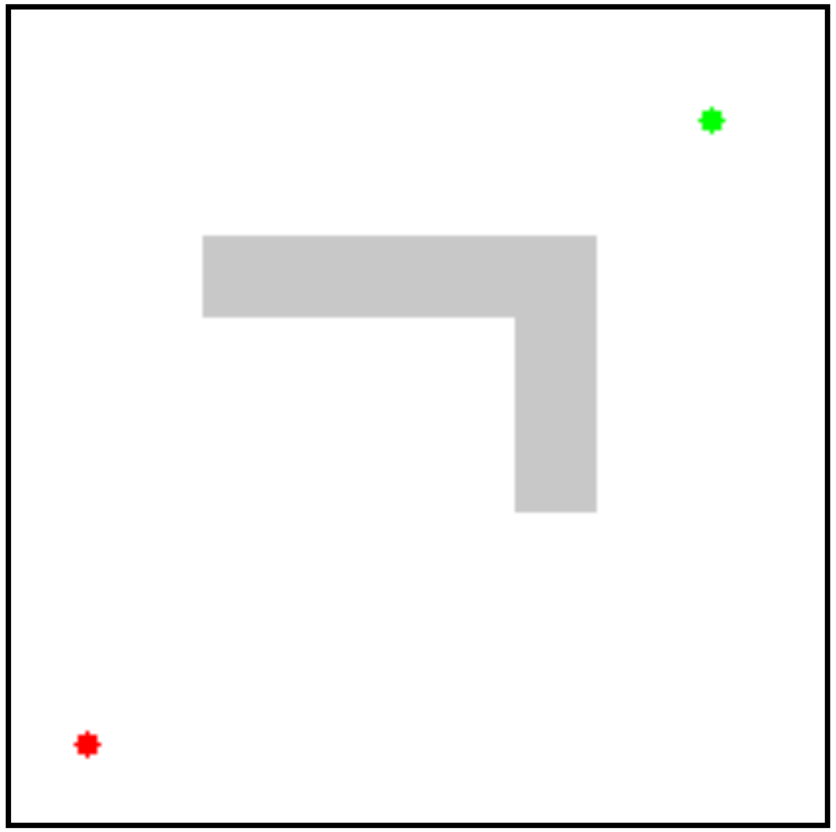


Djikstra A* Weighted A*

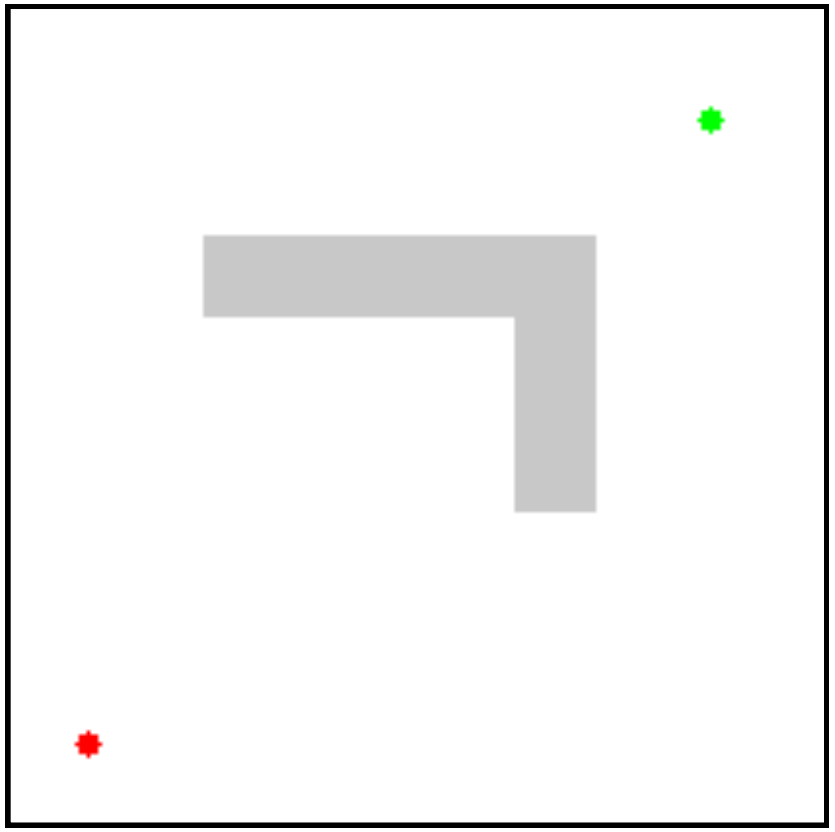
Courtesy wikipedia

High-order bit

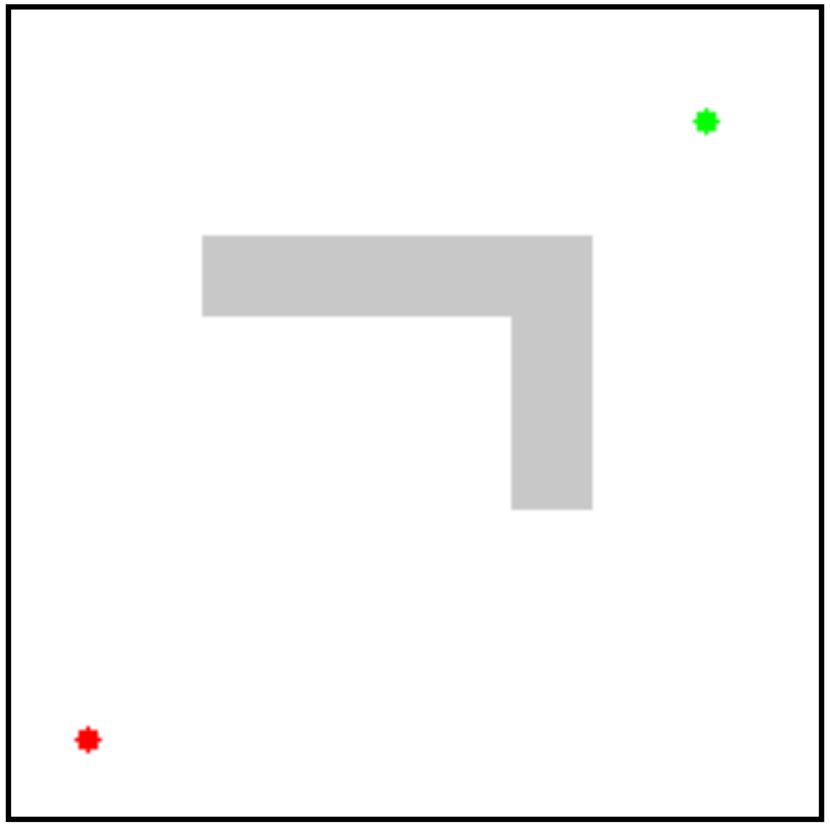
Expansion of a search wavefront from start to goal



Djikstra



A*



Weighted A*

Courtesy wikipedia

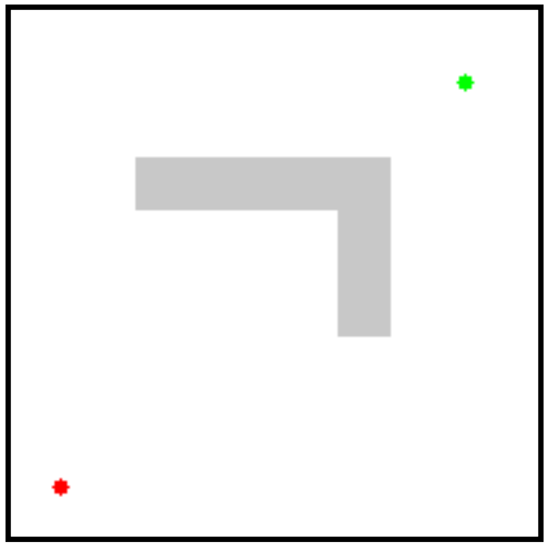
What do we want?

1. Search to systematically reason over the space of paths

2. Find a (near)-optimal path quickly

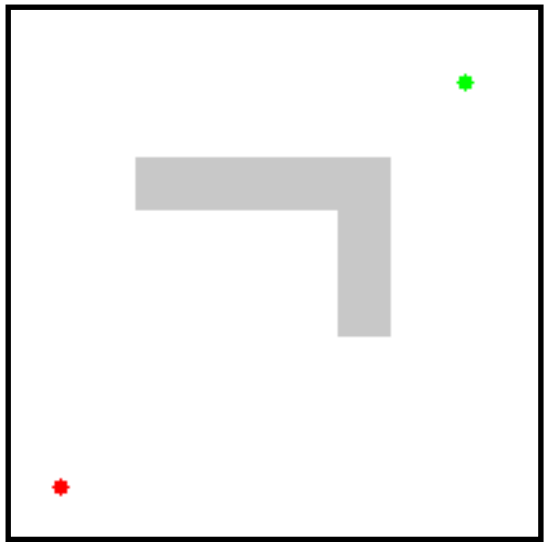
(minimize planning effort)

Best first search



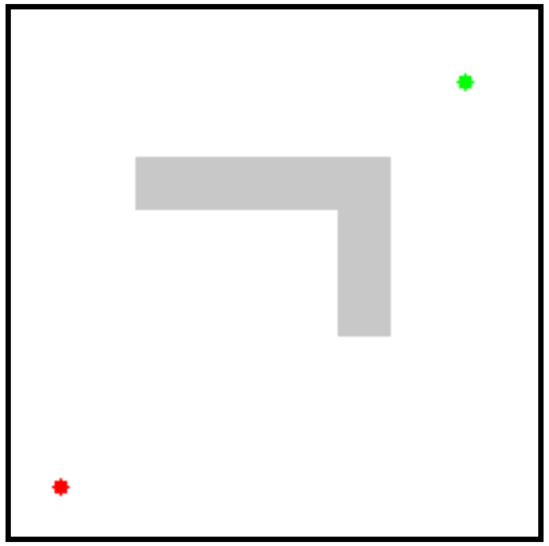
This is a meta-algorithm

Best first search



This is a meta-algorithm

Best first search



This is a meta-algorithm

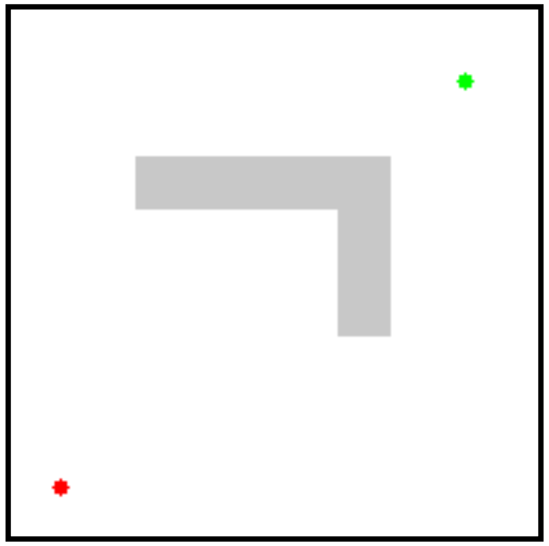
BFS maintains a priority queue of promising nodes

Each node s ranked by a function $f(s)$

Populate queue initially with start node

Element (Node)	**Priority Value (f-value)**
Node A	f(A)
Node B	f(B)
.....	......

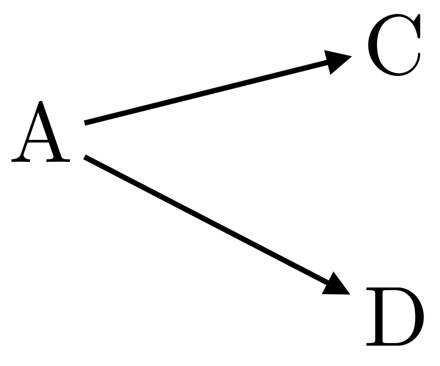
Best first search



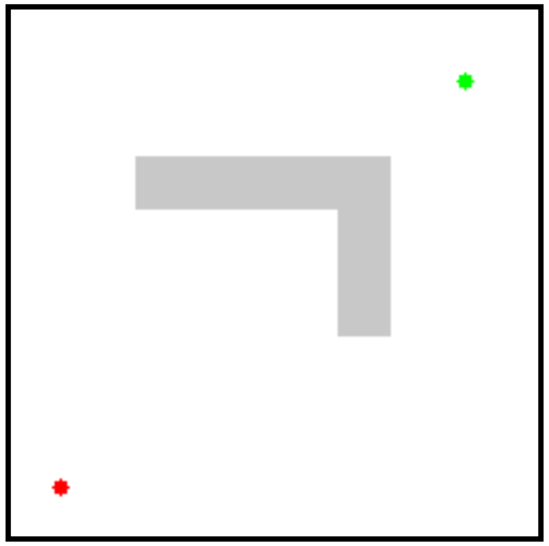
Search explores graph by expanding most promising node min *f(s)*

Terminate when you find the goal

Element (Node)	**Priority Value (f-value)**
Node A	f(A)
Node D	f(D)
Node B	f(B)
Node C	f(C)



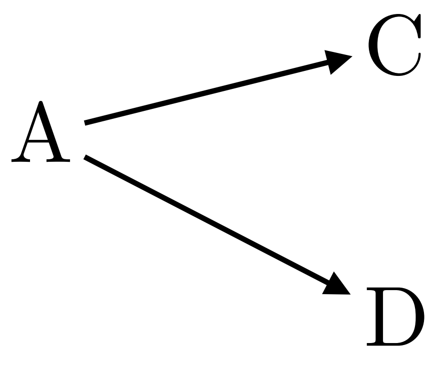
Best first search



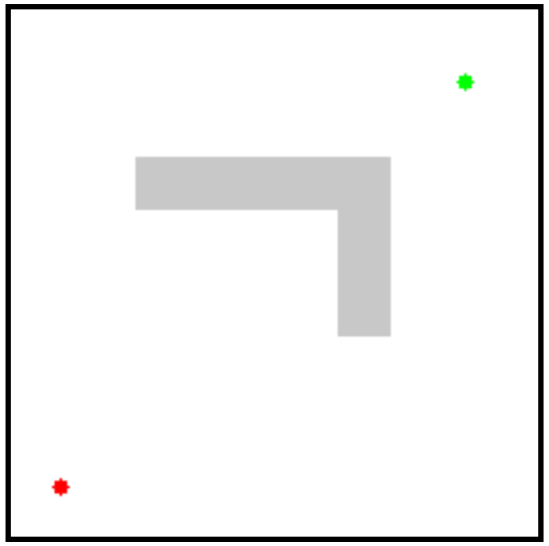
Search explores graph by expanding most promising node min $f(s)$

Terminate when you find the goal

Element (Node)	Priority Value (f-value)
~~Node A~~	~~f(A)~~
Node D	f(D)
Node B	f(B)
Node C	f(C)



Best first search

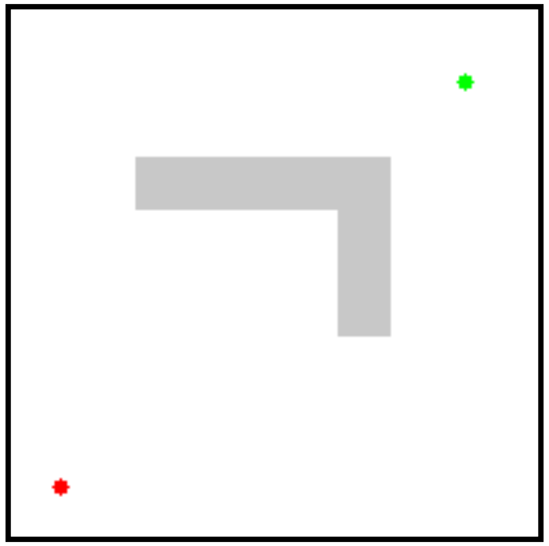


Key Idea: Choose *f(s)* wisely!

- when goal found, it has (near) optimal path

    - minimize the number of expansions

Best first search



Key Idea: Choose $f(s)$ wisely!

- when goal found, it has (near) optimal path
  - minimize the number of expansions

Notations

Given:

Start s_{start} Goal s_{goal}

Cost $c(s, s')$

Objects created:

OPEN: priority queue of nodes to be processed

CLOSED: list of nodes already processed

$g(s)$: estimate of the least cost from start to a given node

Pseudocode

Push *start* into OPEN

While *goal* not expanded

Pop *best* from OPEN

Add *best* to CLOSED

For every successor s'

If $g(s') > g(s) + c(s,s')$

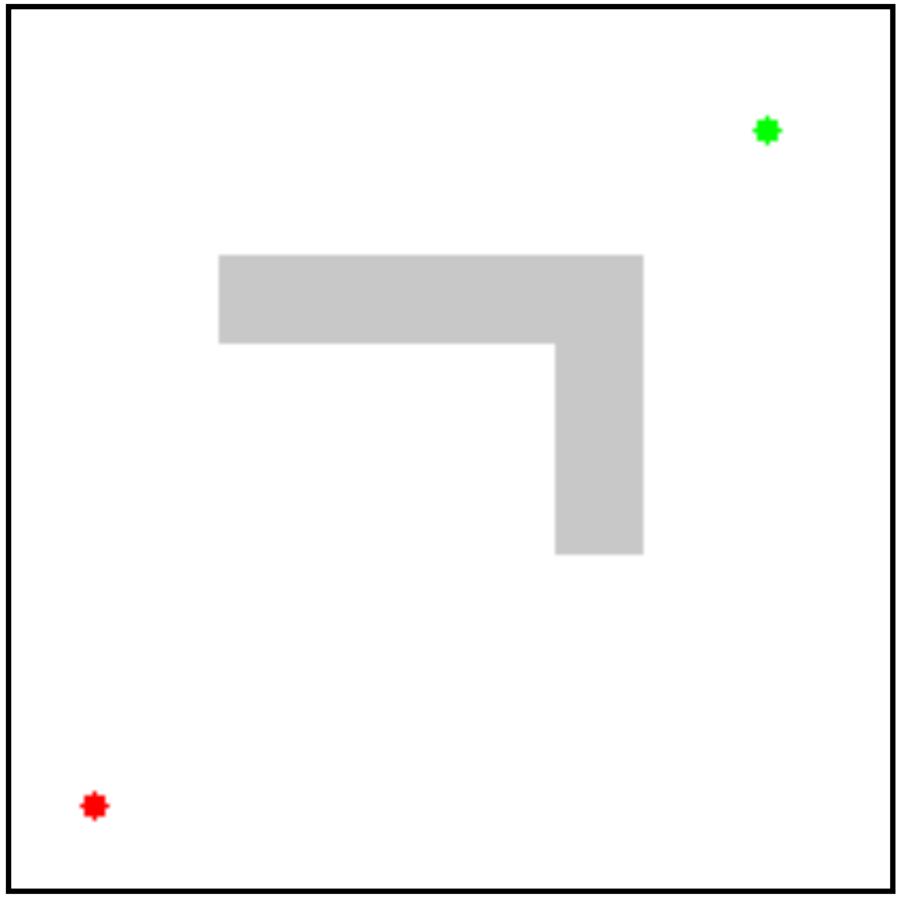
$g(s') = g(s) + c(s,s')$

Add (or update) s' to OPEN

Djikstra's Algorithm

Set

$$f(s) = g(s)$$

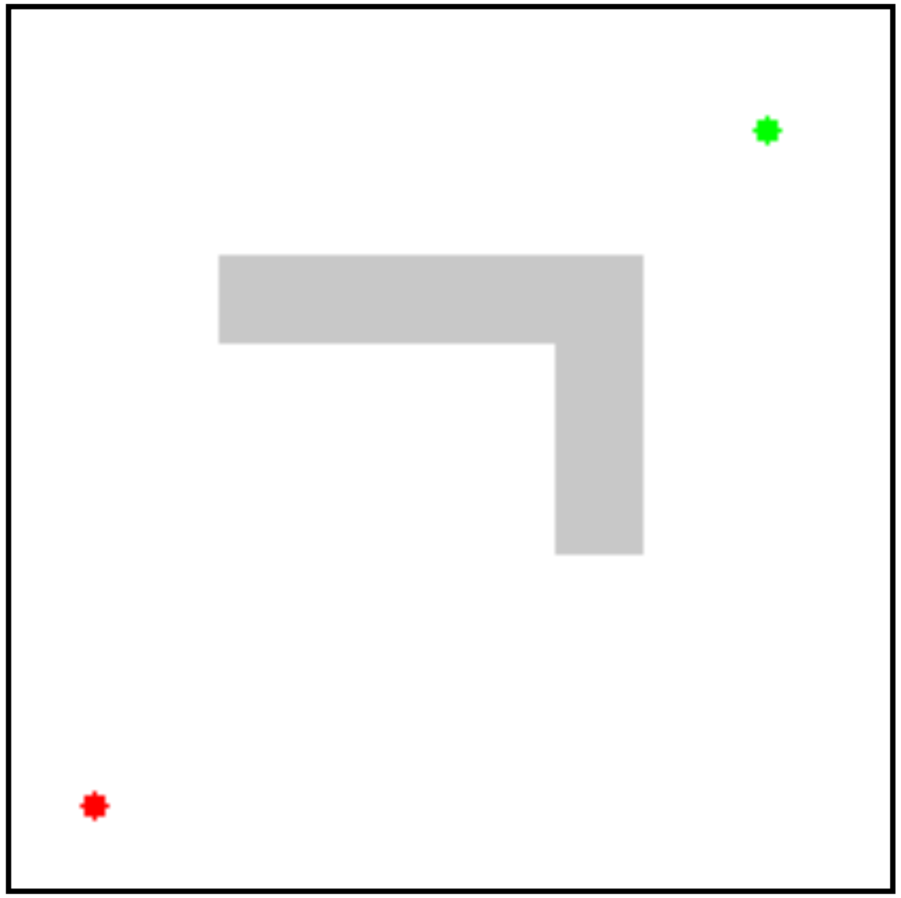


Sort nodes by their cost to come

Djikstra's Algorithm

Set

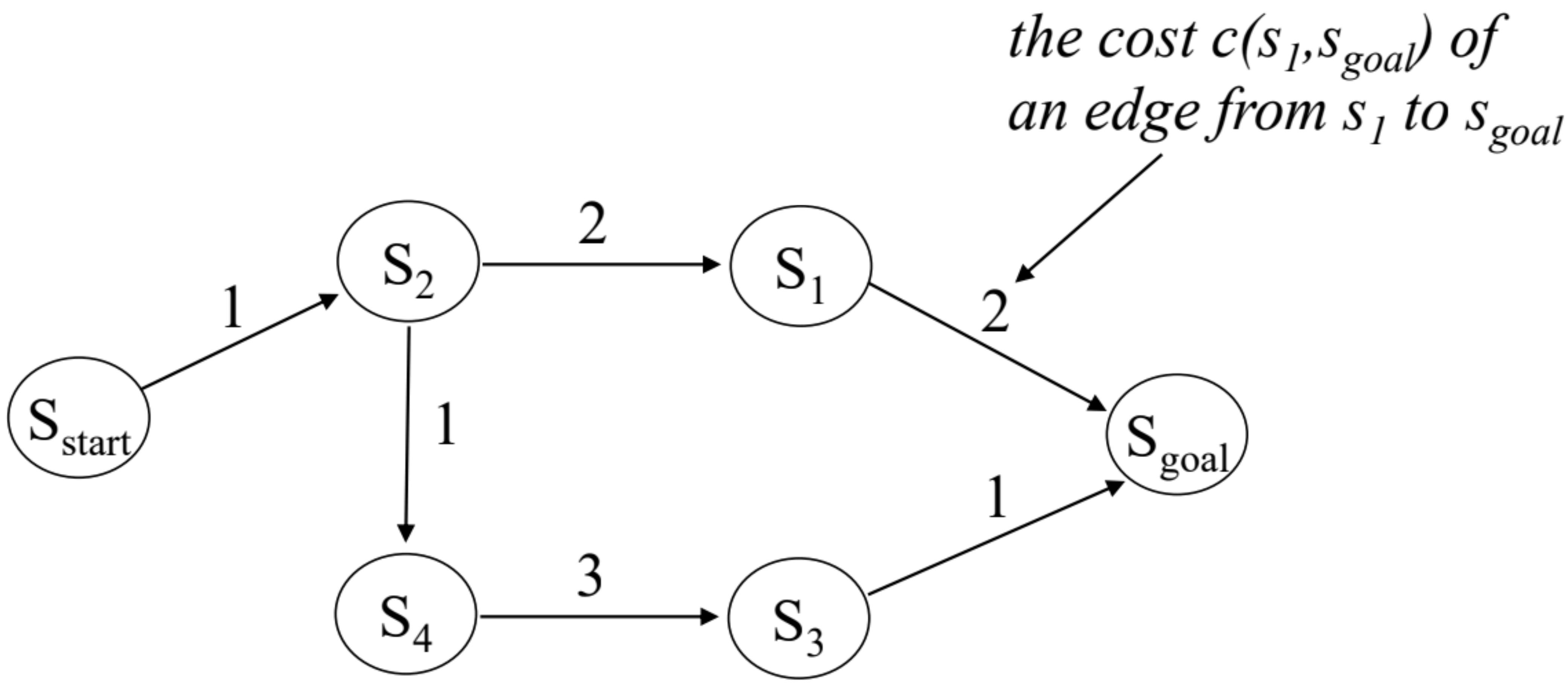
$$f(s) = g(s)$$



Sort nodes by their cost to come

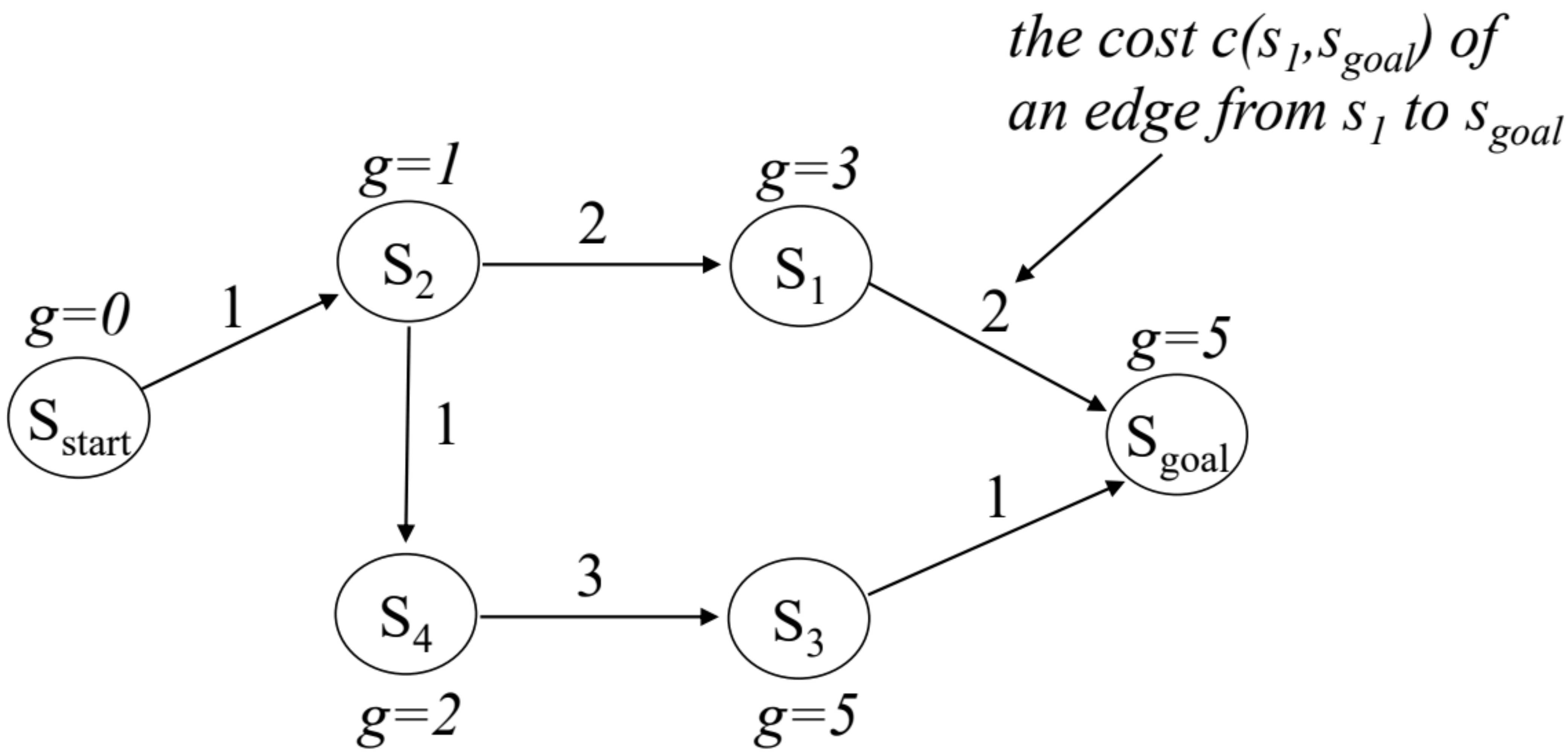
Djikstra's Algorithm

– optimal values satisfy: $g(s) = \min_{s'' \in pred(s)} g(s'') + c(s'',s)$



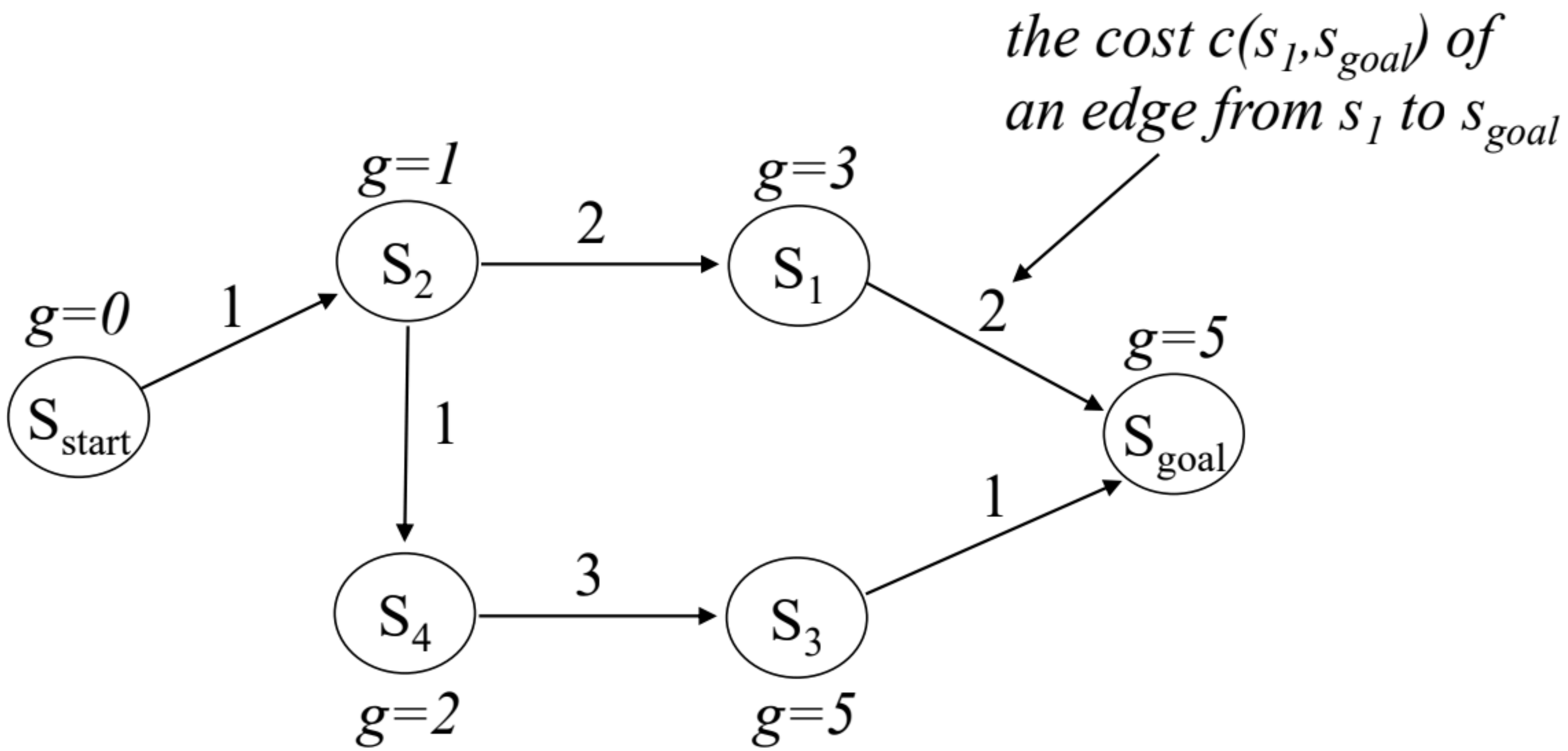
Djikstra's Algorithm

– optimal values satisfy: $g(s) = \min_{s'' \in pred(s)} g(s'') + c(s'',s)$



Djikstra's Algorithm

– optimal values satisfy: $g(s) = \min_{s'' \in pred(s)} g(s'') + c(s'',s)$



Nice property:

Only process nodes ONCE. Only process cheaper nodes than goal.

Can we have a better f(s)?

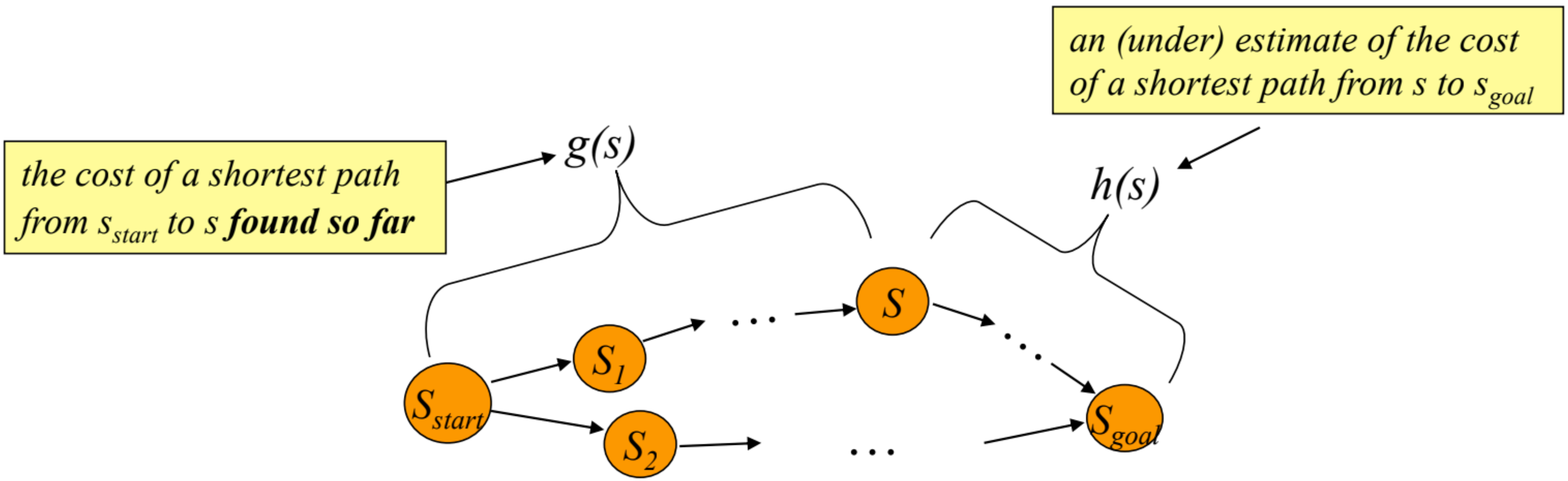
Can we have a better f(s)?

Yes!

f(s) should estimate the
cost of the path to goal

Heuristics

What if we had a heuristic $h(s)$ that estimated the cost to goal?



Set the evaluation function $f(s) = g(s) + h(s)$

Example of heuristics?

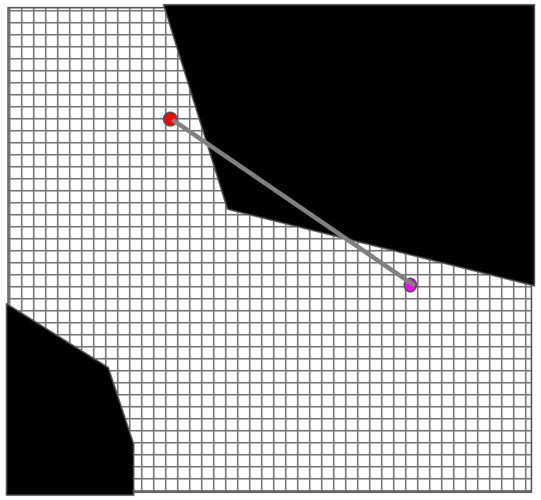
Example of heuristics?

1. Minimum number of nodes to go to goal

Example of heuristics?

1. Minimum number of nodes to go to goal

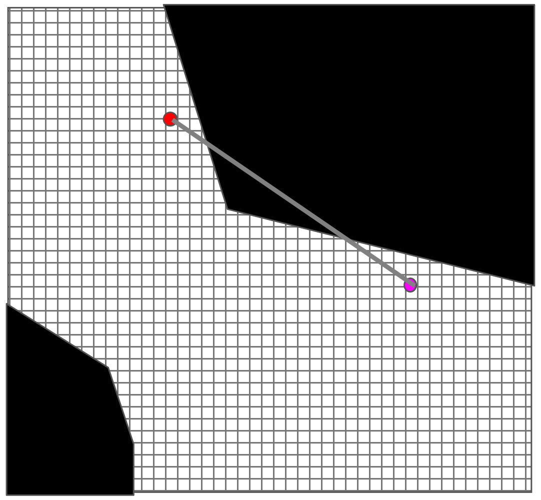
2. Euclidean distance to goal (if you know your cost is measuring length)



Example of heuristics?

1. Minimum number of nodes to go to goal

2. Euclidean distance to goal (if you know your cost is measuring length)

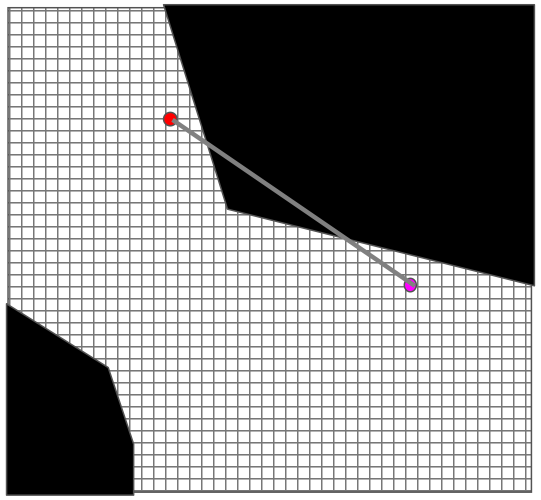


3. Solution to a relaxed problem

Example of heuristics?

1. Minimum number of nodes to go to goal

2. Euclidean distance to goal (if you know your cost is measuring length)



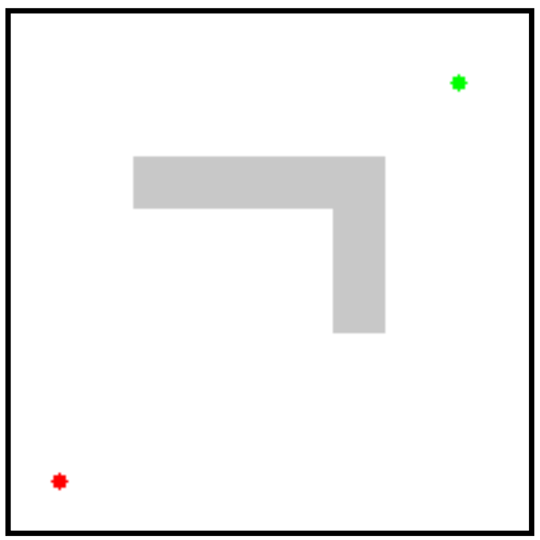
3. Solution to a relaxed problem

4. Domain knowledge / Learning ....

A* [Hart, Nillson, Raphael, '68]

Let L be the length of the shortest path

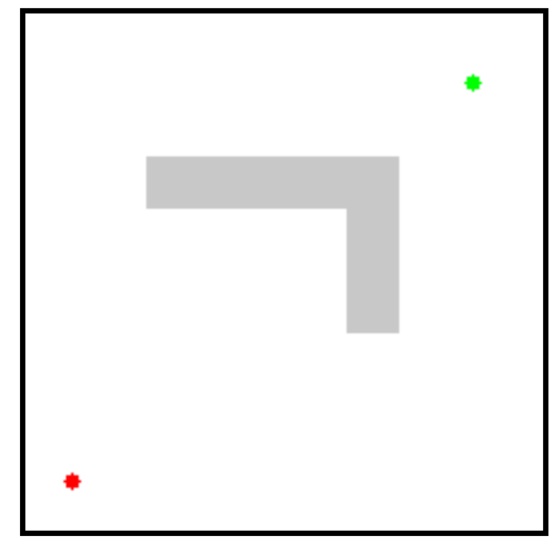
Djikstra



Expand every state

$g(s) < L$

A*



Expand every state

$f(s) = g(s) + h(s) < L$

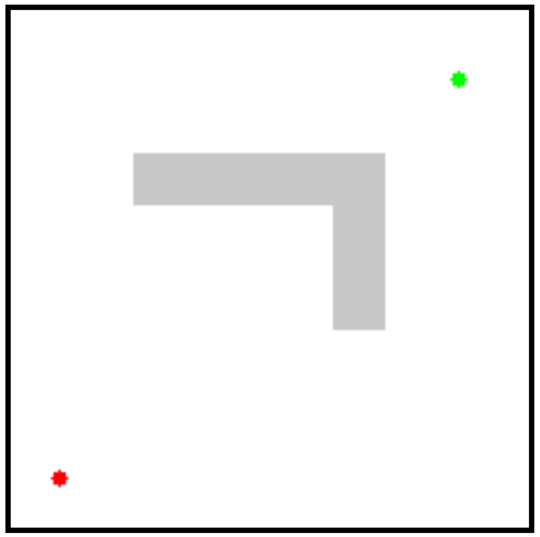
Both find the optimal path ...

but A* only expands relevant states, i.e., does much less work!

A* [Hart, Nillson, Raphael, '68]

Let L be the length of the shortest path

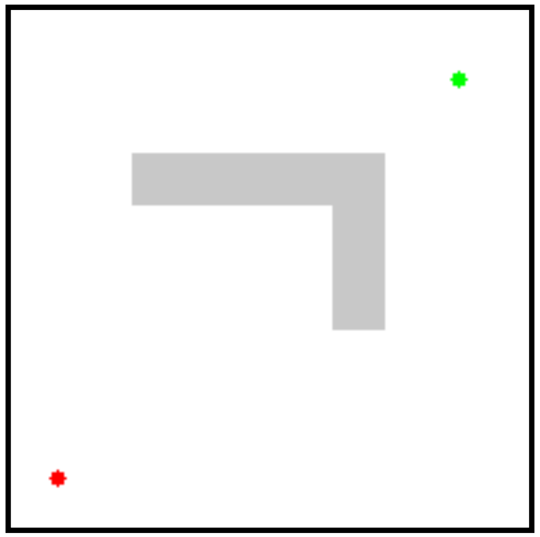
Djikstra



Expand every state

$g(s) < L$

A*



Expand every state

$f(s) = g(s) + h(s) < L$

Both find the optimal path ...

but A* only expands relevant states, i.e., does much less work!

A* Search

Computes optimal g-values for relevant states

while(s_{goal} is not expanded)

  remove s with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;

  insert s into *CLOSED*;

  for every successor s' of s such that s' not in *CLOSED*

    if $g(s') > g(s) + c(s,s')$

      $g(s') = g(s) + c(s,s')$;

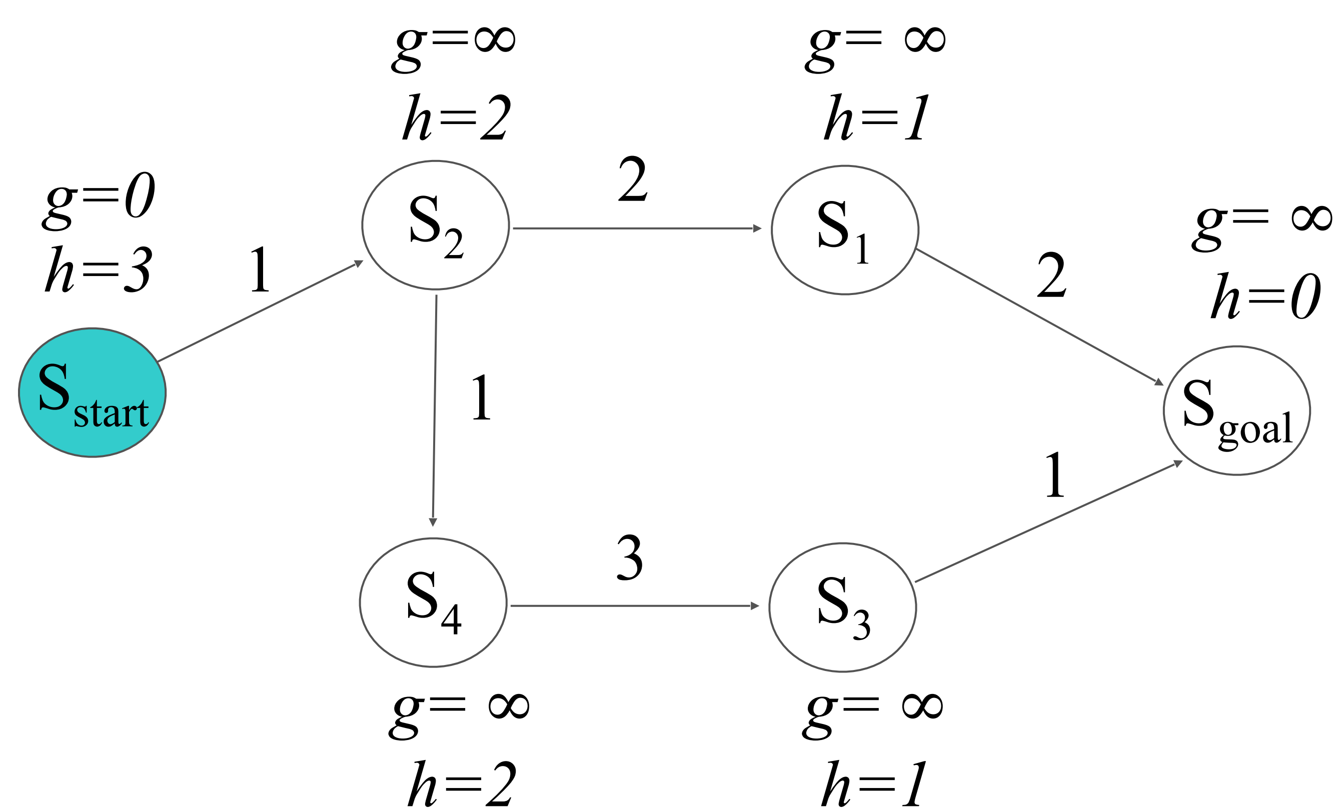
      insert s' into *OPEN*;

A* Search

Computes optimal g-values for relevant states

while(s_{goal} is not expanded)
  remove s with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;
  insert s into *CLOSED*;
  for every successor s' of s such that s' not in *CLOSED*
    if $g(s') > g(s) + c(s,s')$
      $g(s') = g(s) + c(s,s')$;
      insert s' into *OPEN*;

CLOSED = {}
OPEN = $\{s_{start}\}$
next state to expand: s_{start}

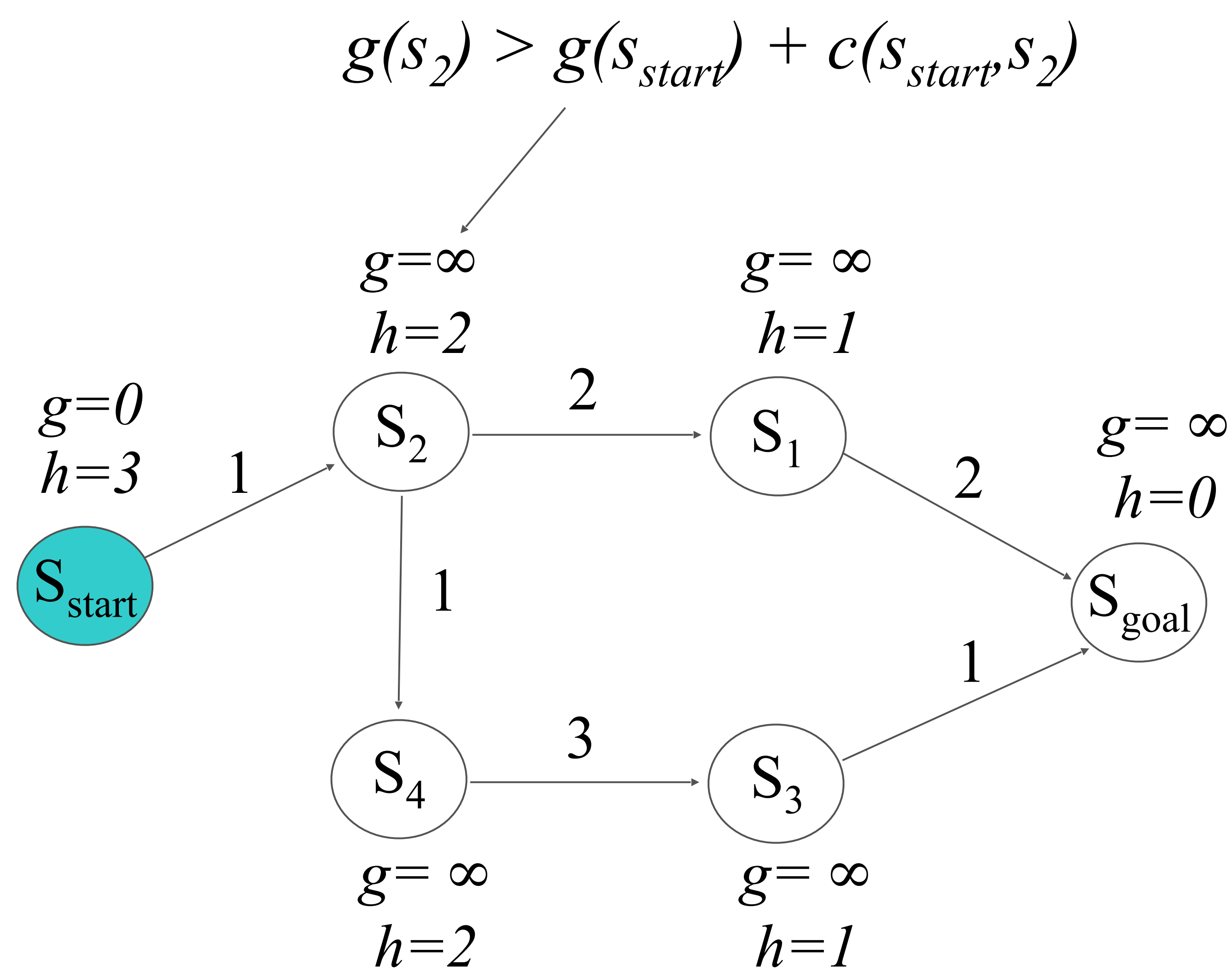


A* Search

Computes optimal g-values for relevant states

while(s_{goal} is not expanded)
remove s with the smallest $[f(s) = g(s)+h(s)]$ from $OPEN$;
insert s into $CLOSED$;
for every successor s' of s such that s' not in $CLOSED$
if $g(s') > g(s) + c(s,s')$
$g(s') = g(s) + c(s,s')$;
insert s' into $OPEN$;

$CLOSED = \{\}$
$OPEN = \{s_{start}\}$
next state to expand: s_{start}



A* Search

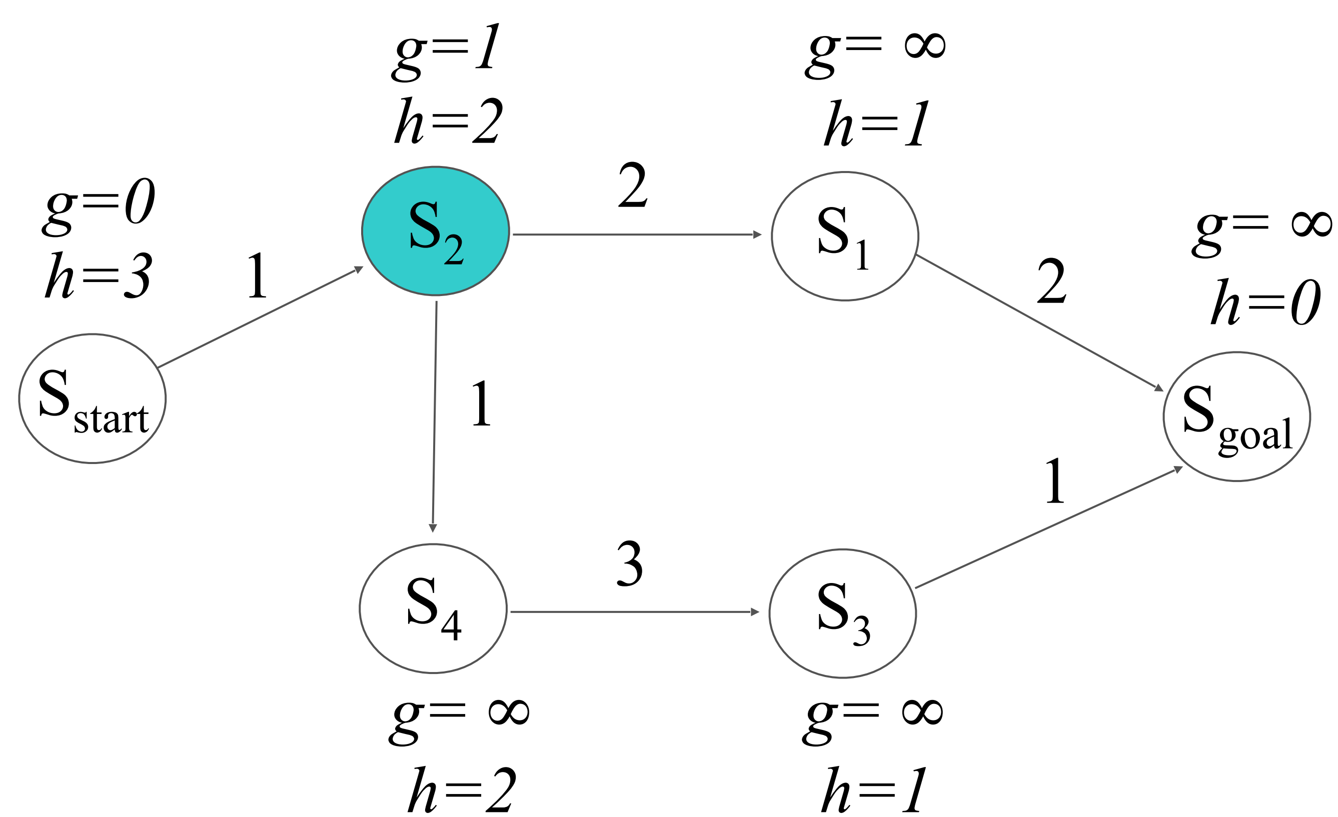
Computes optimal g-values for relevant states

while(s_{goal} is not expanded)
  remove s with the smallest $[f(s) = g(s)+h(s)]$ from $OPEN$;
  insert s into $CLOSED$;
  for every successor s' of s such that s' not in $CLOSED$
    if $g(s') > g(s) + c(s,s')$
      $g(s') = g(s) + c(s,s')$;
      insert s' into $OPEN$;

$CLOSED = \{s_{start}\}$

$OPEN = \{s_2\}$

next state to expand: s_2



A* Search

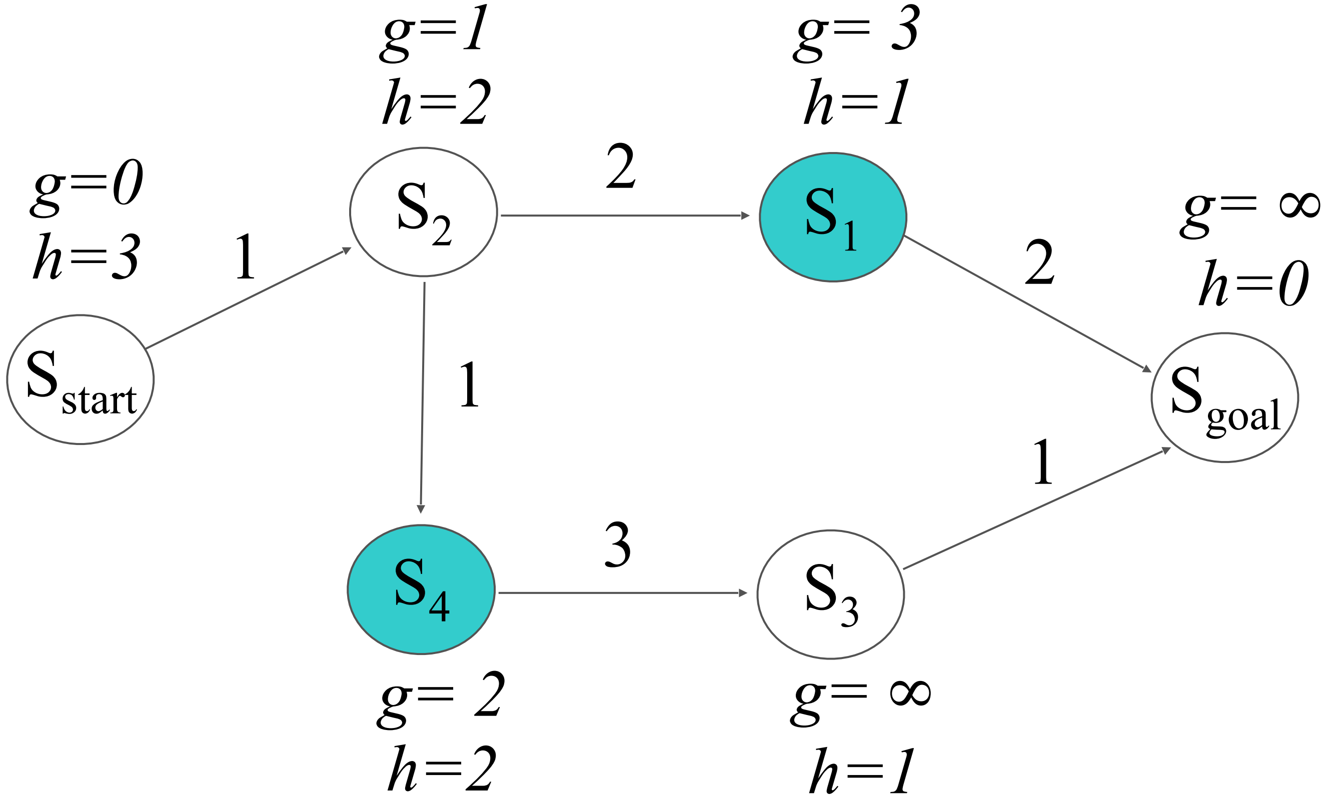
Computes optimal g-values for relevant states

while(s_{goal} is not expanded)
  remove s with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;
  insert s into *CLOSED*;
  for every successor s' of s such that s' not in *CLOSED*
    if $g(s') > g(s) + c(s,s')$
      $g(s') = g(s) + c(s,s')$;
      insert s' into *OPEN*;

$CLOSED = \{s_{start}, s_2\}$

$OPEN = \{s_1, s_4\}$

next state to expand: s_1



A* Search

Computes optimal g-values for relevant states

while(s_{goal} is not expanded)

remove s with the smallest $[f(s) = g(s)+h(s)]$ from $OPEN$;

insert s into $CLOSED$;

for every successor s' of s such that s' not in $CLOSED$

if $g(s') > g(s) + c(s,s')$

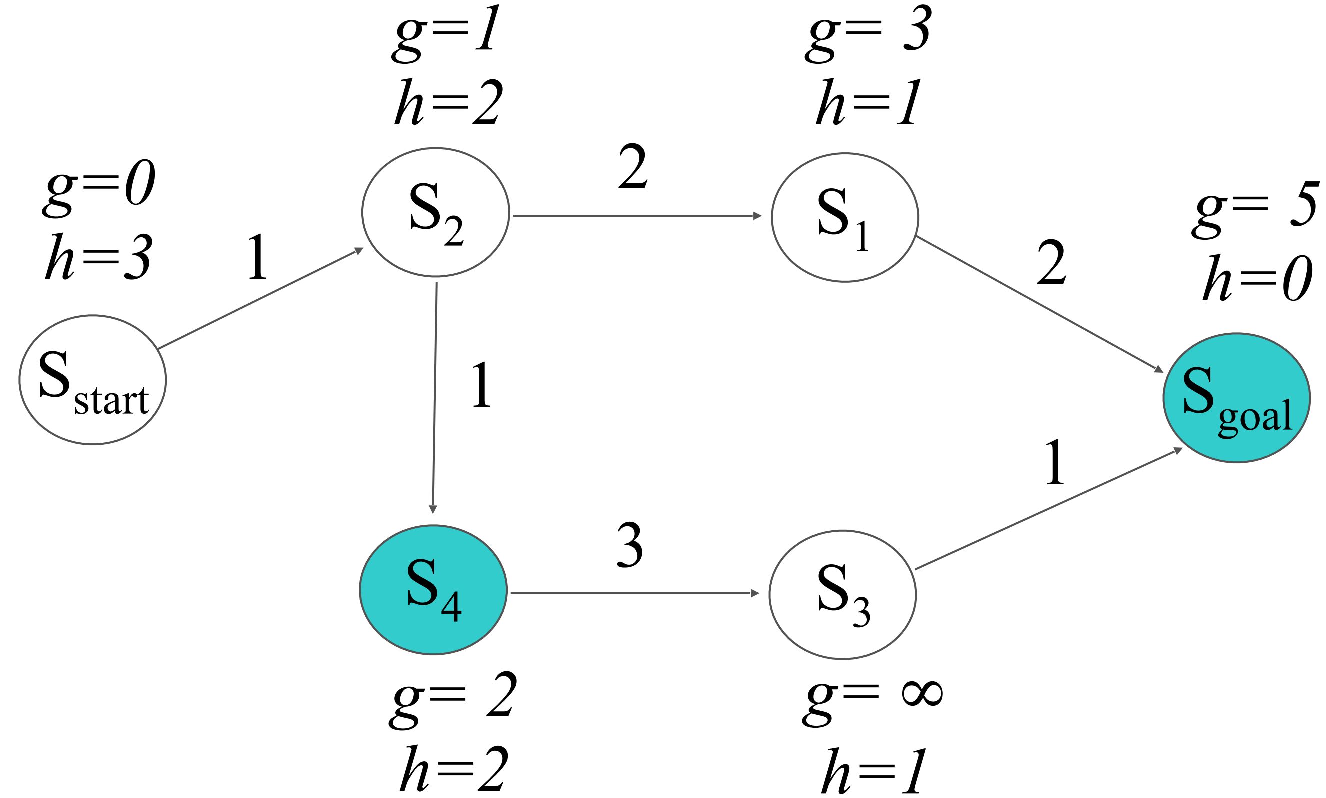
$g(s') = g(s) + c(s,s')$;

insert s' into $OPEN$;

$CLOSED = \{s_{start}, s_2, s_1\}$

$OPEN = \{s_4, s_{goal}\}$

next state to expand: s_4



A* Search

Computes optimal g-values for relevant states

while(s_{goal} is not expanded)

  remove s with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;

  insert s into *CLOSED*;

  for every successor s' of s such that s' not in *CLOSED*

    if $g(s') > g(s) + c(s,s')$

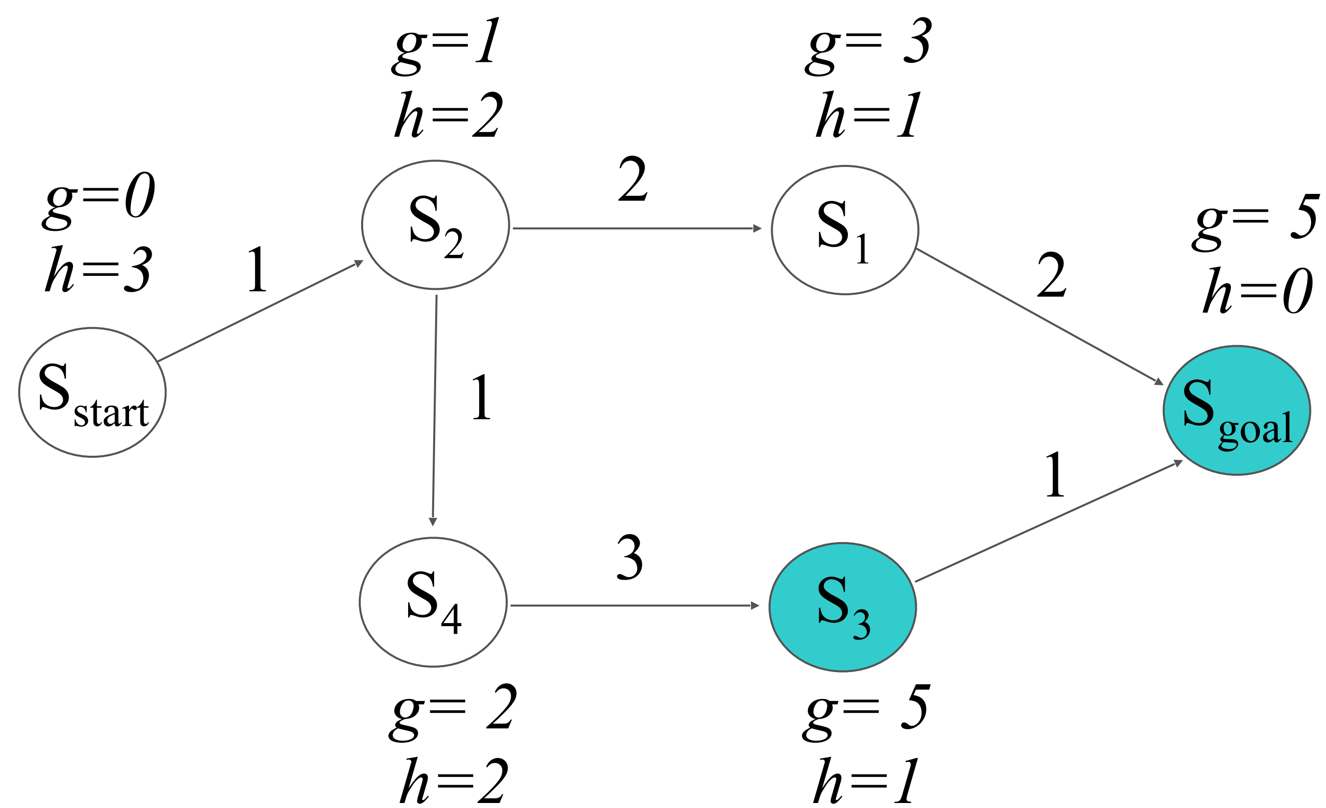
      $g(s') = g(s) + c(s,s');$

      insert s' into *OPEN*;

$CLOSED = \{s_{start}, s_2, s_1, s_4\}$

$OPEN = \{s_3, s_{goal}\}$

next state to expand: s_{goal}



A* Search

Computes optimal g-values for relevant states

while(s_{goal} is not expanded)

remove s with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;

insert s into *CLOSED*;

for every successor s' of s such that s' not in *CLOSED*

if $g(s') > g(s) + c(s,s')$

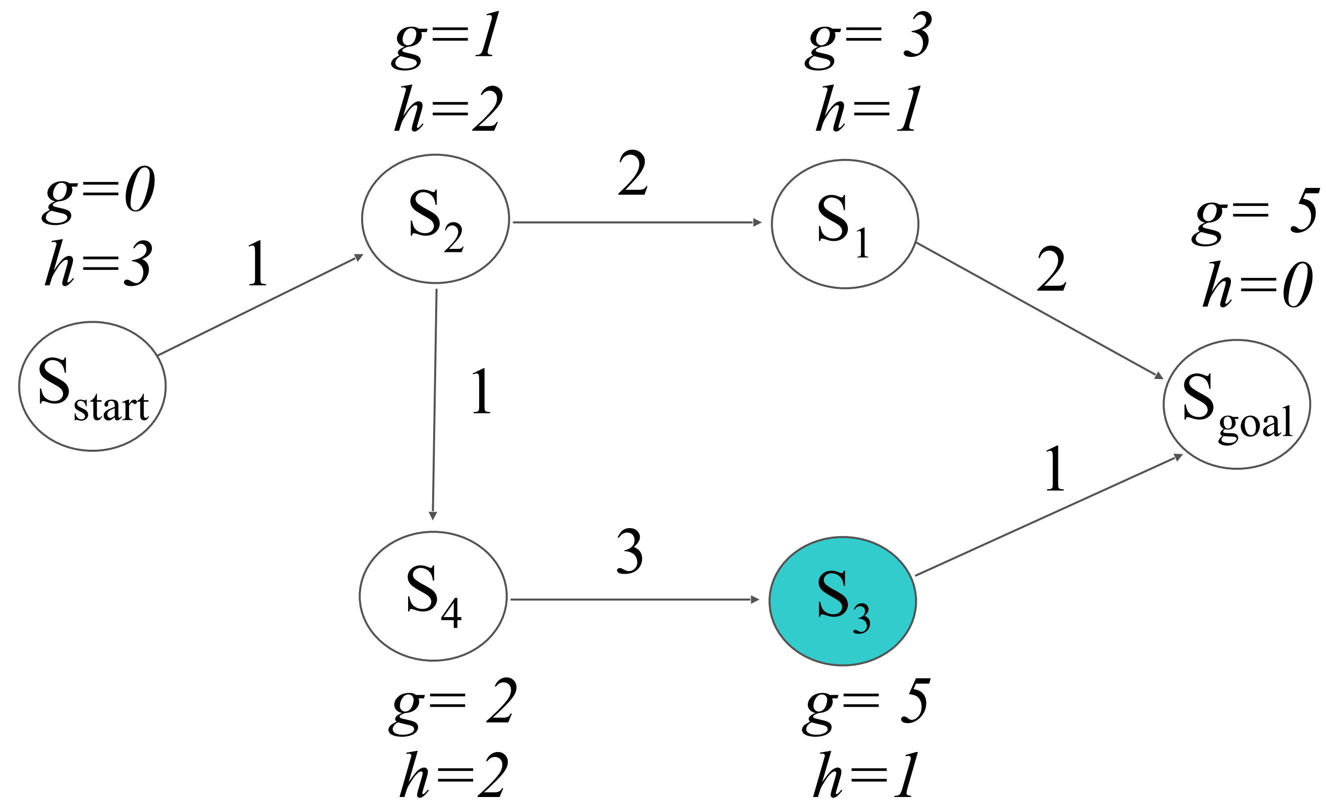
$g(s') = g(s) + c(s,s');$

insert s' into *OPEN*;

$$CLOSED = \{s_{start}, s_2, s_1, s_4, s_{goal}\}$$

$$OPEN = \{s_3\}$$

done



A* Search

Computes optimal g-values for relevant states

while(s_{goal} is not expanded)

remove s with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;

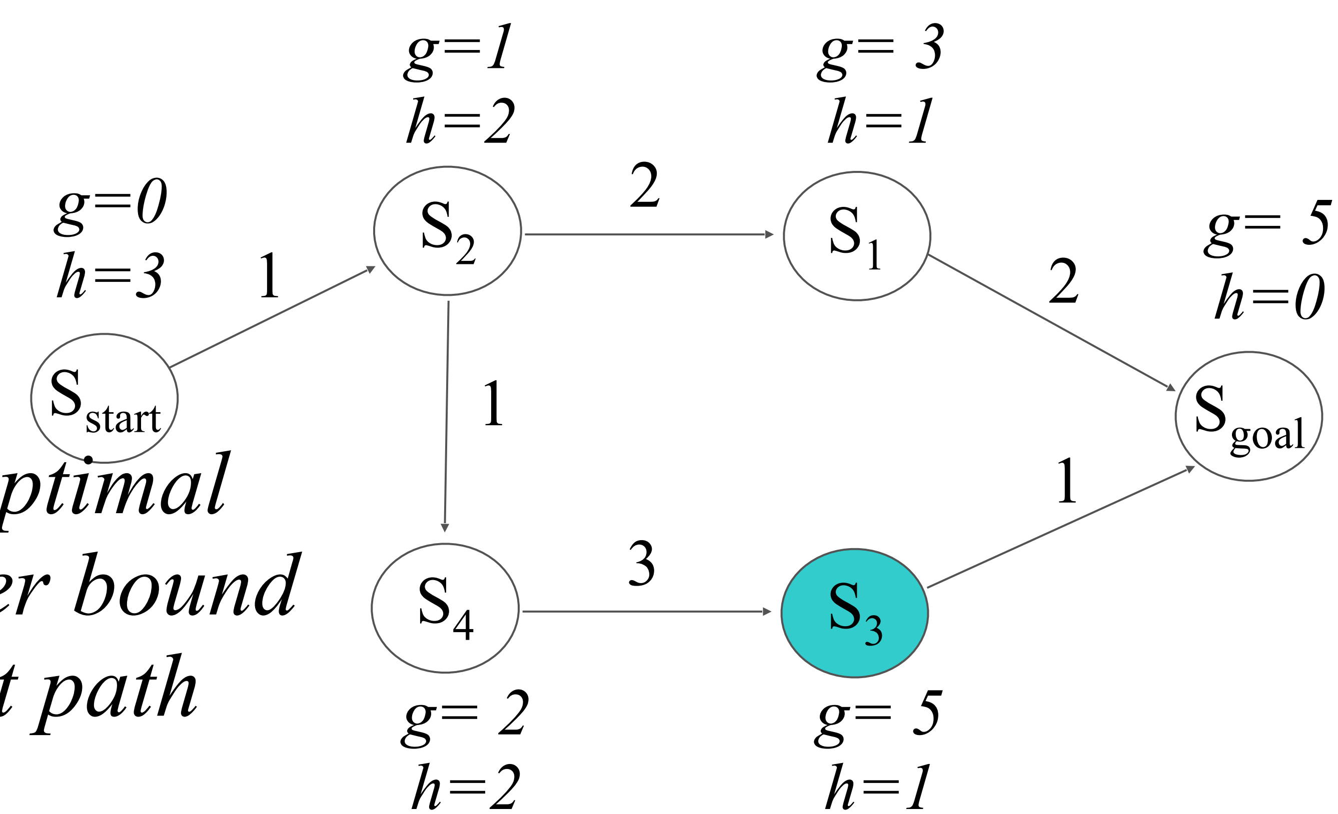
insert s into *CLOSED*;

for every successor s' of s such that s' not in *CLOSED*

if $g(s') > g(s) + c(s,s')$

$g(s') = g(s) + c(s,s');$

insert s' into *OPEN*;

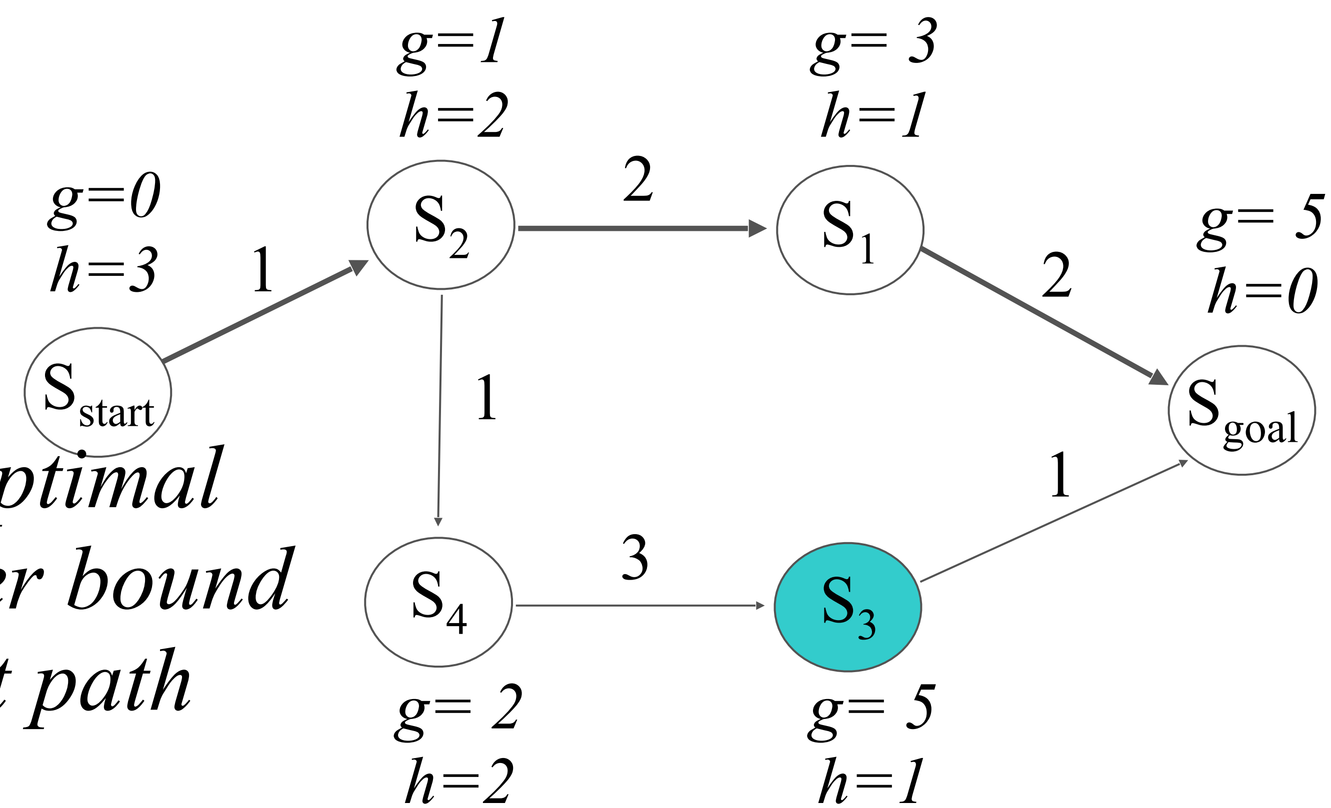


for every expanded state g(s) is optimal
for every other state g(s) is an upper bound
we can now compute a least-cost path

A* Search

Computes optimal g-values for relevant states

while(s_{goal} is not expanded)
  remove s with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;
  insert s into *CLOSED*;
  for every successor s' of s such that s' not in *CLOSED*
    if $g(s') > g(s) + c(s,s')$
      $g(s') = g(s) + c(s,s')$;
      insert s' into *OPEN*;



for every expanded state g(s) is optimal
for every other state g(s) is an upper bound
we can now compute a least-cost path

Properties of heuristics

What properties should $h(s)$ satisfy? How does it affect search?

Properties of heuristics

What properties should *h(s)* satisfy? How does it affect search?

Admissible: $h(s) <= h^*(s)$ $h(\text{goal}) = 0$

If this true, the path returned by A* is optimal

Properties of heuristics

What properties should *h(s)* satisfy? How does it affect search?

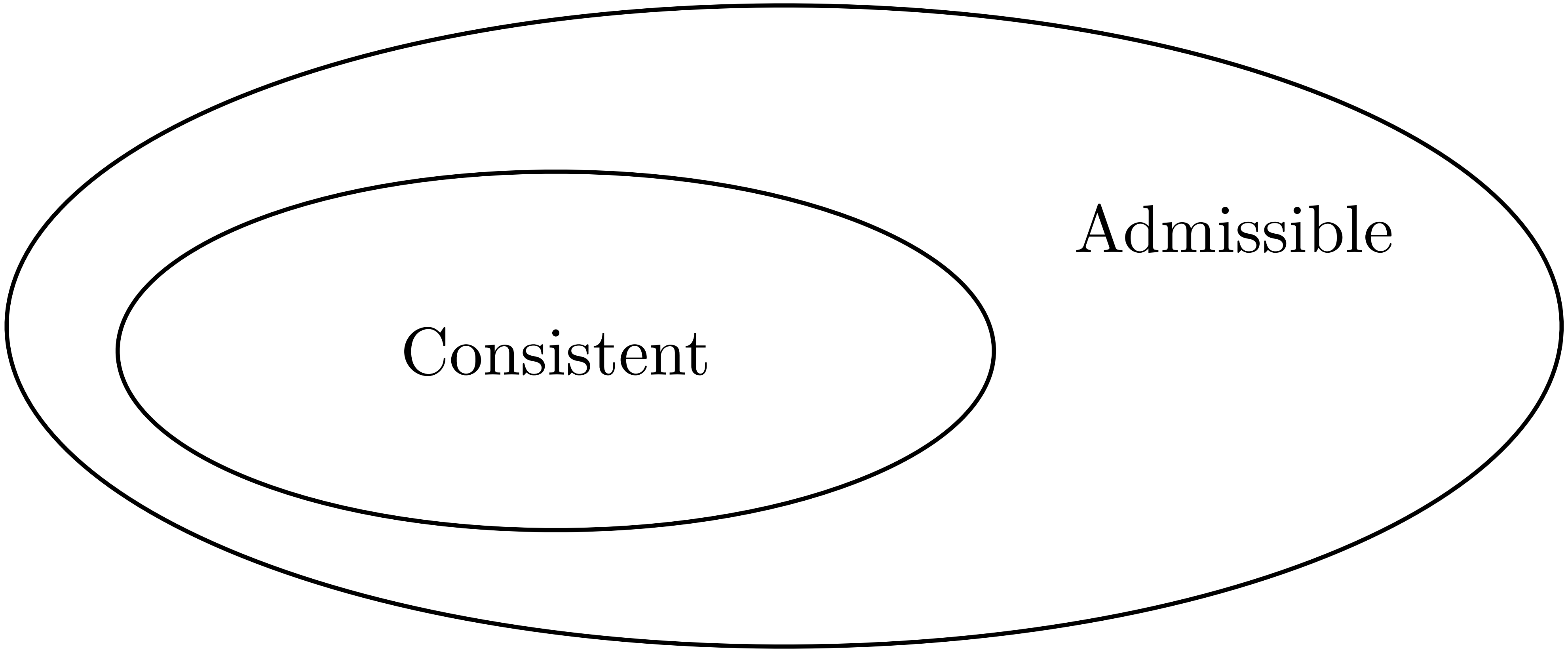
Admissible: $h(s) <= h^*(s)$ $\quad$ $h(goal) = 0$

If this true, the path returned by A* is optimal

Consistency: $h(s) <= c(s,s') + h(s')$ $\quad$ $h(goal) = 0$

If this true, A* is optimal AND efficient (will not re-expand a node)

Admissible vs Consistent



Theorem: ALL consistent heuristics are admissible, not vice versa!

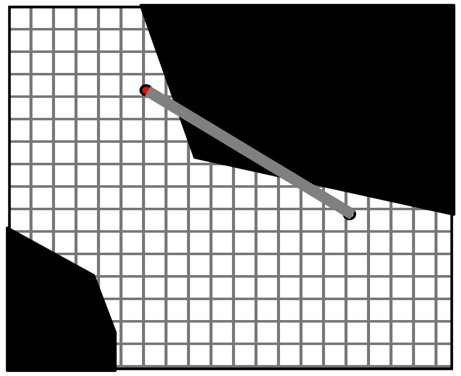
Takeaway:

Heuristics are great because they focus search on relevant states

AND

still give us optimal solution

Design of Informative Heuristics



- For grid-based navigation:
  - Euclidean distance
  - Manhattan distance: $h(x,y) = abs(x-x_{goal}) + abs(y-y_{goal})$
  - Diagonal distance: $h(x,y) = max(abs(x-x_{goal}), abs(y-y_{goal}))$
  - More informed distances???

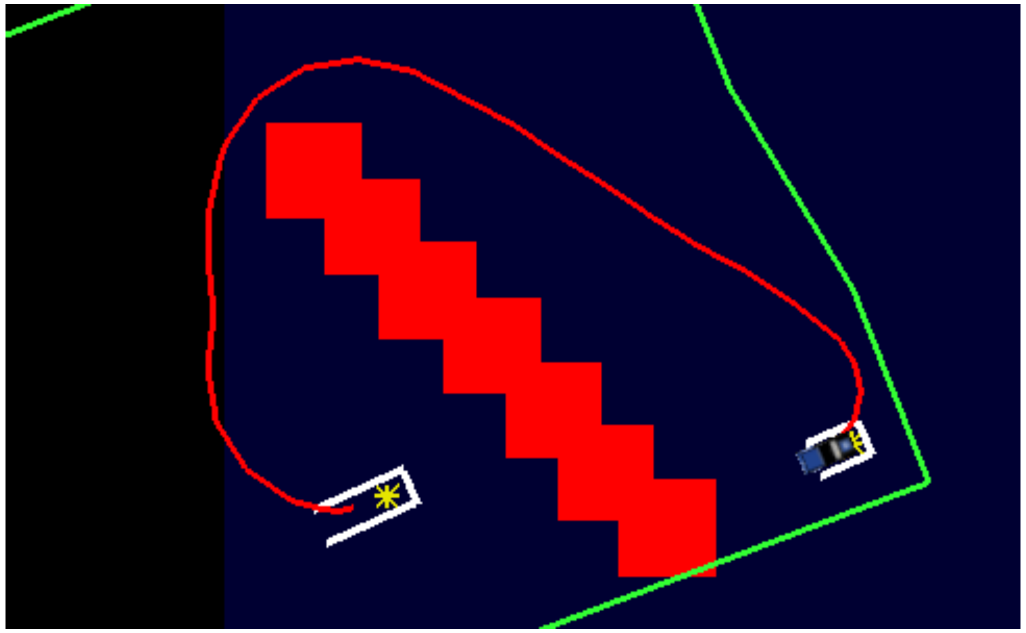
Which heuristics are admissible for
4-connected grid?
8-connected grid?

Courtesy Max Likhachev

Design of Informative Heuristics

- For lattice-based 3D (x,y,Θ) navigation:

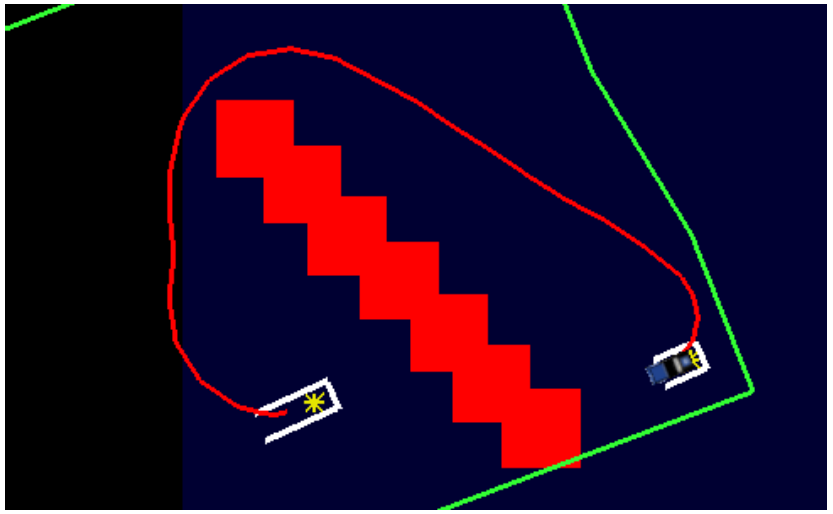
Any ideas?



Courtesy Max Likhachev

Design of Informative Heuristics

- For lattice-based 3D (x,y,Θ) navigation:



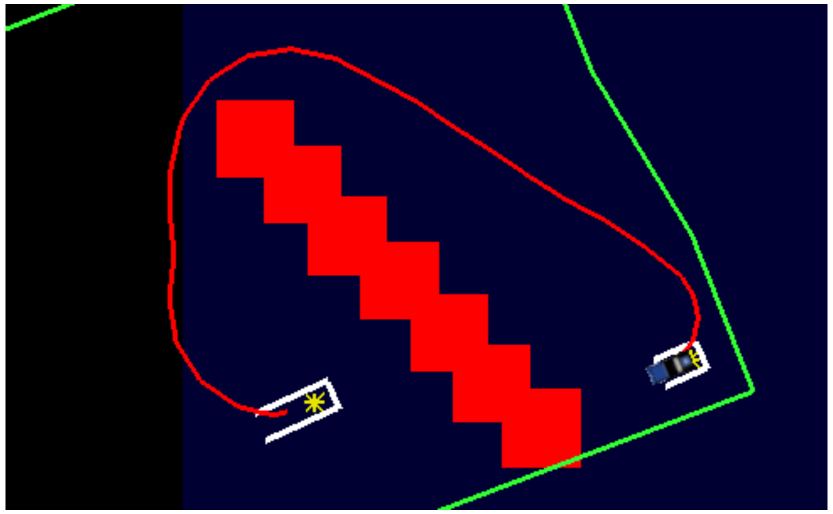
  - 2D (x,y) distance accounting for obstacles (single Dijkstra's on 2D grid cell starting at goalcell will give us these values)

Any problems where it will be highly uninformative?

Courtesy Max Likhachev

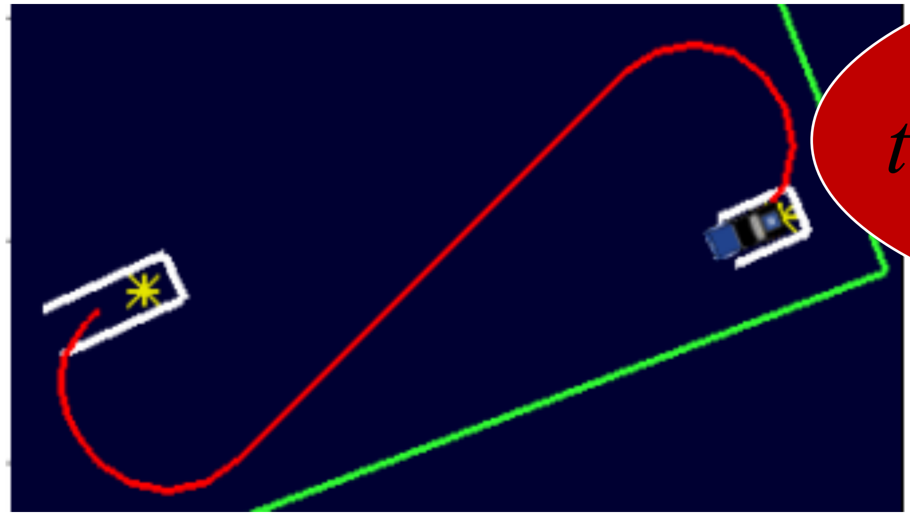
Design of Informative Heuristics

- For lattice-based 3D (x,y,Θ) navigation:



  – 2D (x,y) distance accounting for obstacles (single Dijkstra's on 2D grid cell starting at goalcell will give us these values)

Any problems where it will be highly uninformative?



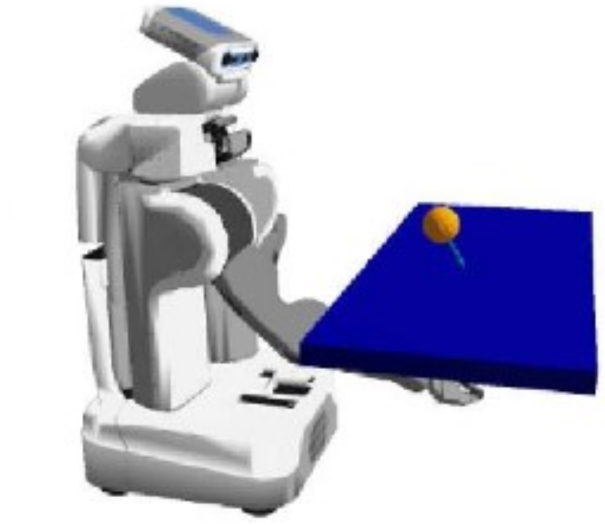
Any heuristic functions that will guide search well in this example?

Courtesy Max Likhachev

Design of Informative Heuristics

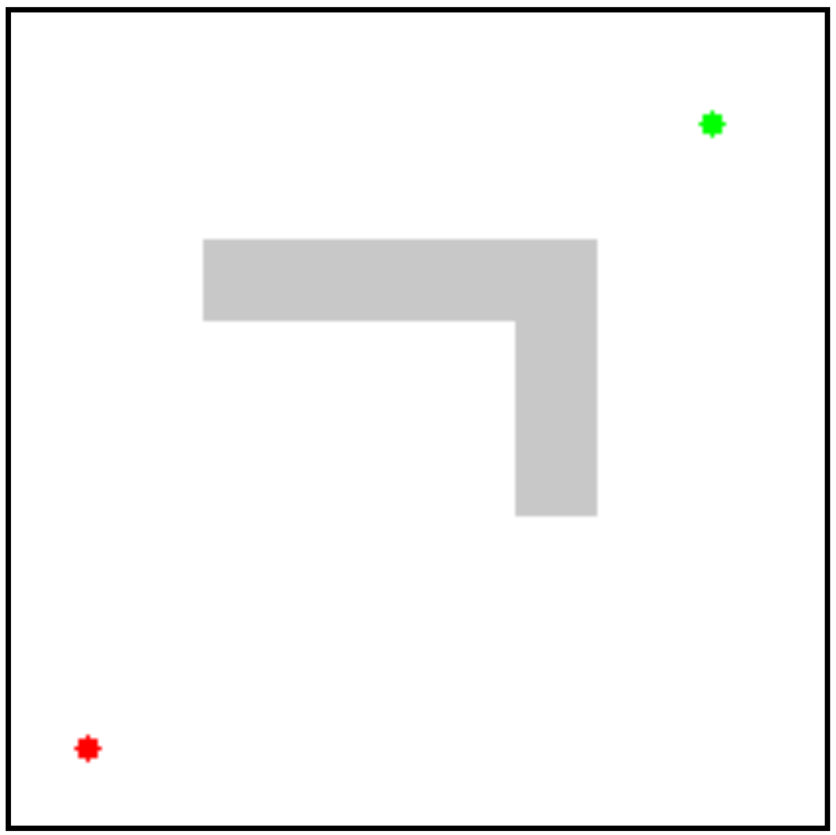
- Arm planning in 3D:

Any ideas?

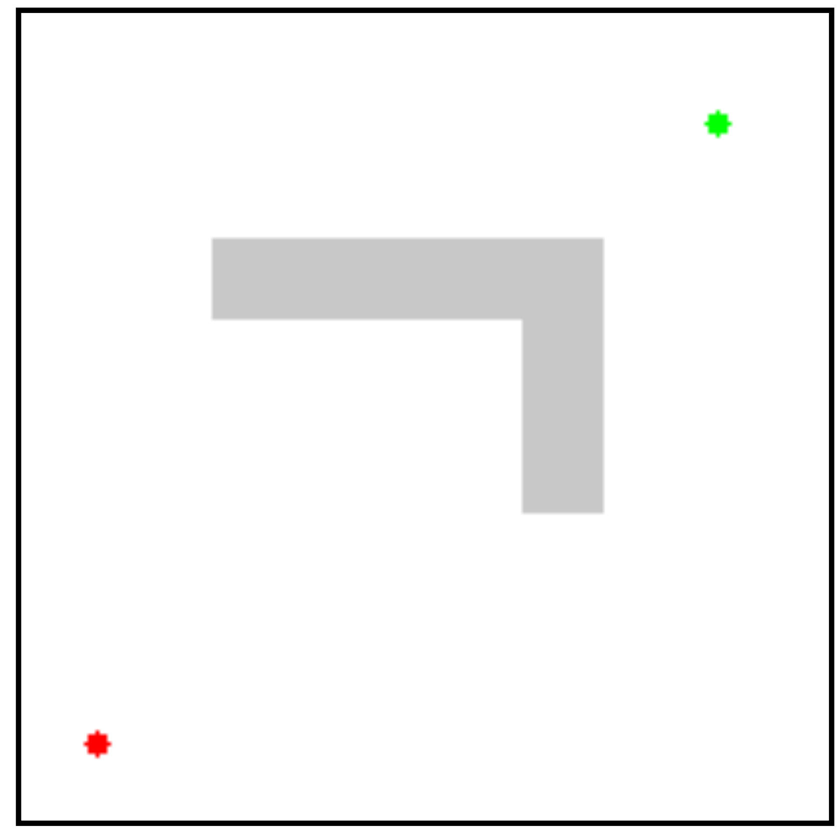


Courtesy Max Likhachev

Is admissibility always what we want?

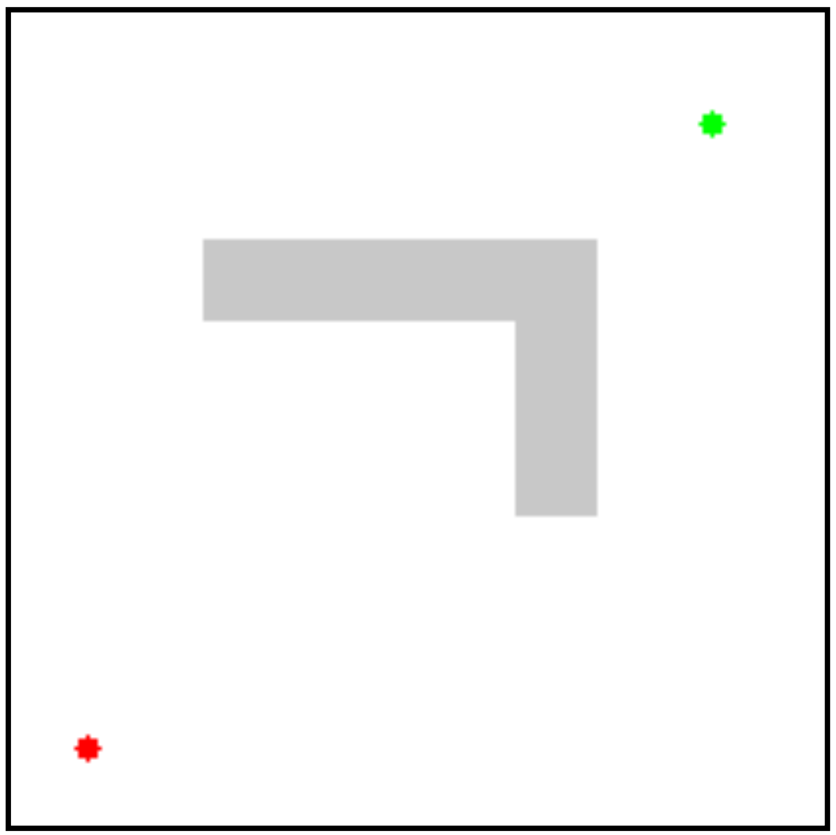


Admissible

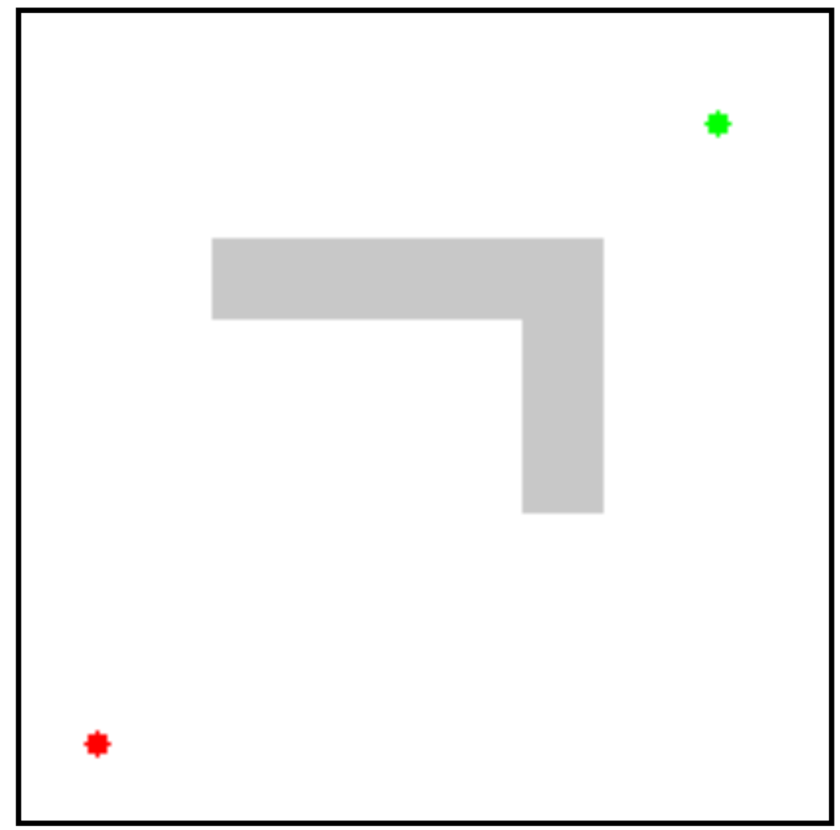


Inadmissible

Is admissibility always what we want?



Admissible



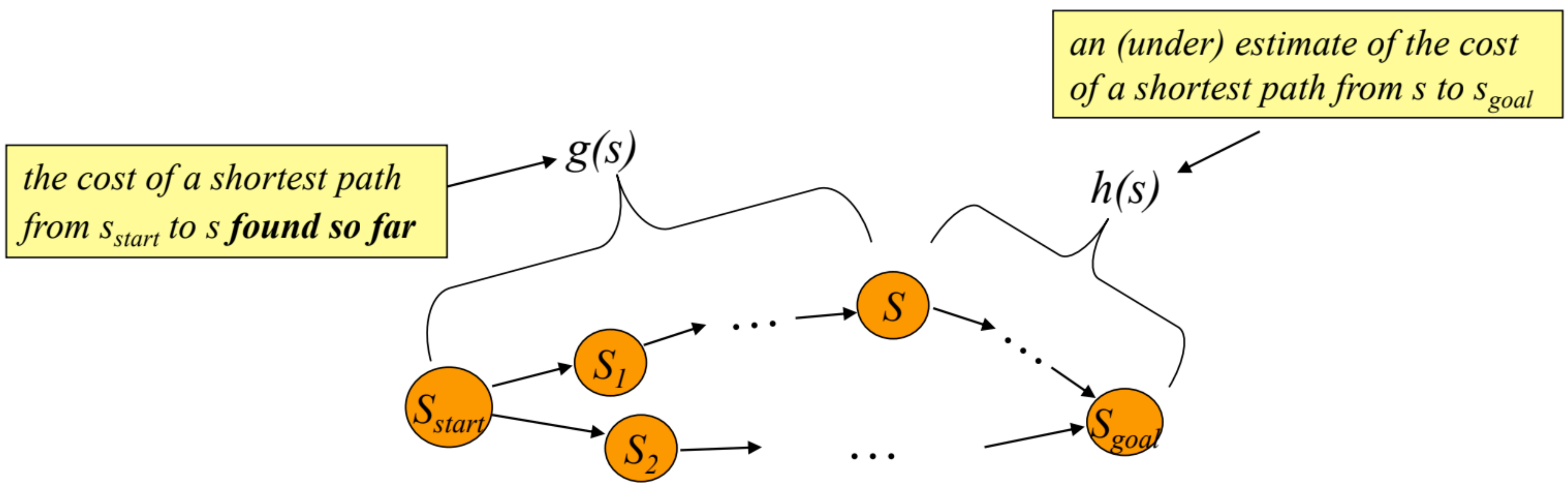
Inadmissible

Can inadmissible heuristics help us with this tradeoff?

Solution Quality ⟷ Number of states expanded

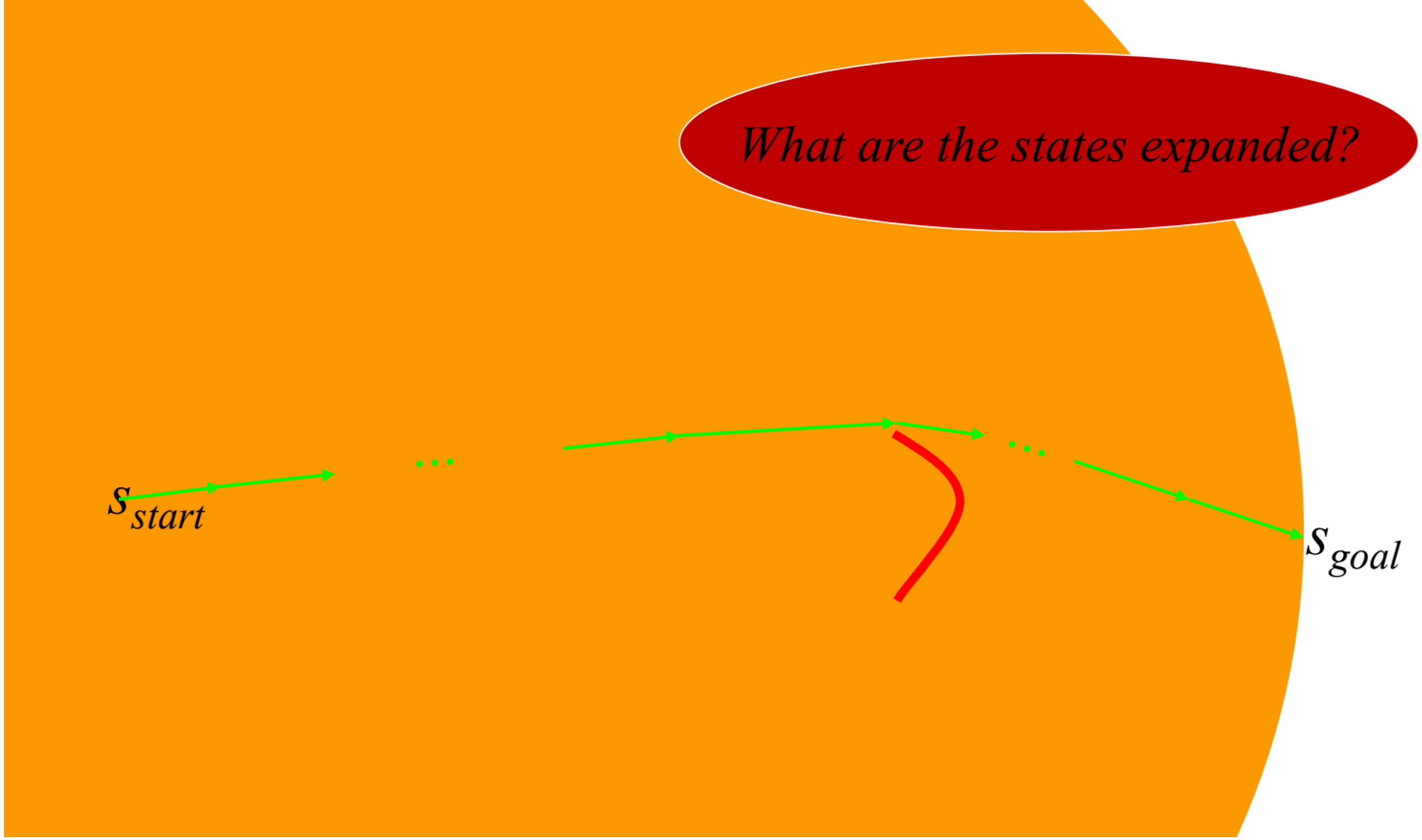
- A* Search: expands states in the order of $f = g+h$ values
- Dijkstra's: expands states in the order of $f = g$ values

- **Weighted A*:** expands states in the order of $f = g+\varepsilon h$ values, $\varepsilon > 1$ = bias towards states that are closer to goal



Courtesy Max Likhachev

- Dijkstra's: expands states in the order of $f = g$ values



Courtesy Max Likhachev

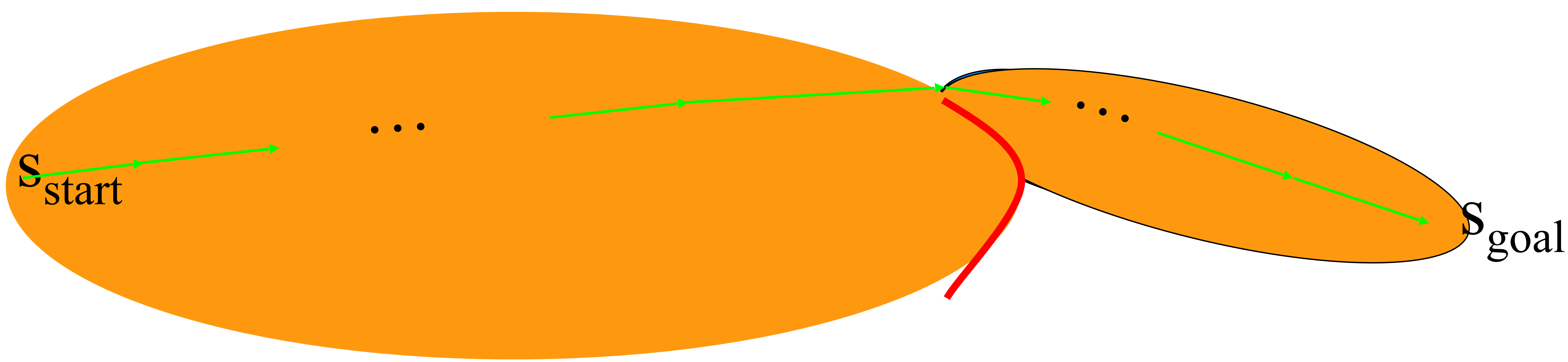
Effect of the Heuristic Function

A* Search: expands states in the order of $f = g+h$ values

Courtesy Max Likhachev

Effect of the Heuristic Function

A* Search: expands states in the order of $f = g+h$ values

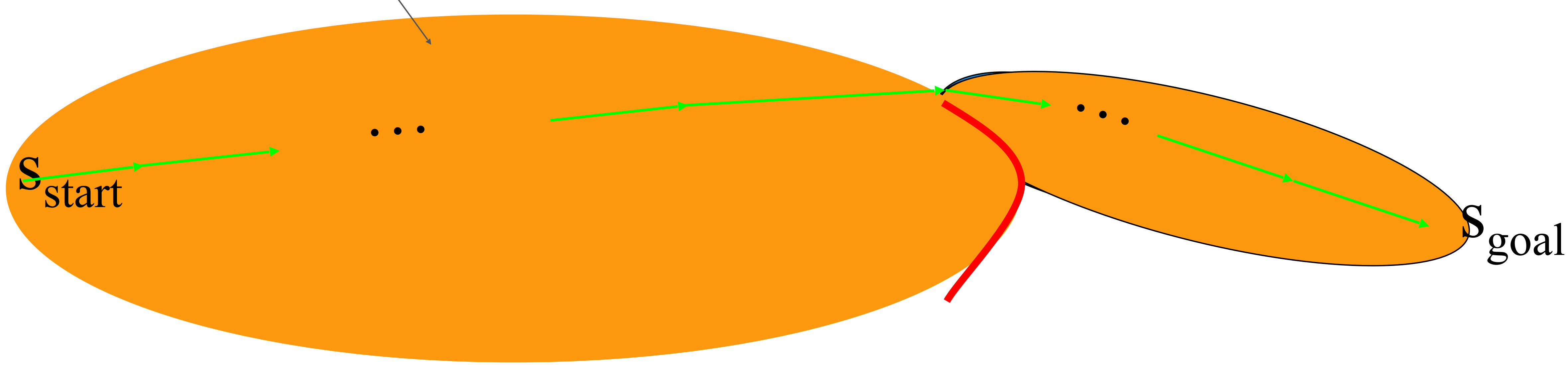


Courtesy Max Likhachev

Effect of the Heuristic Function

A* Search: expands states in the order of $f = g+h$ values

for large problems this results in A* quickly running out of memory (memory: O(n))



Courtesy Max Likhachev

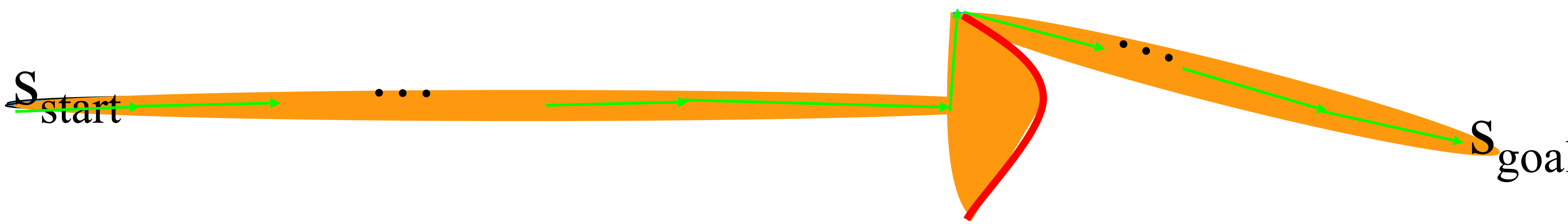
Effect of the Heuristic Function

Weighted A* Search: expands states in the order of $f = g+\varepsilon h$ values, $\varepsilon > 1$ = bias towards states that are closer to goal

Courtesy Max Likhachev

Effect of the Heuristic Function

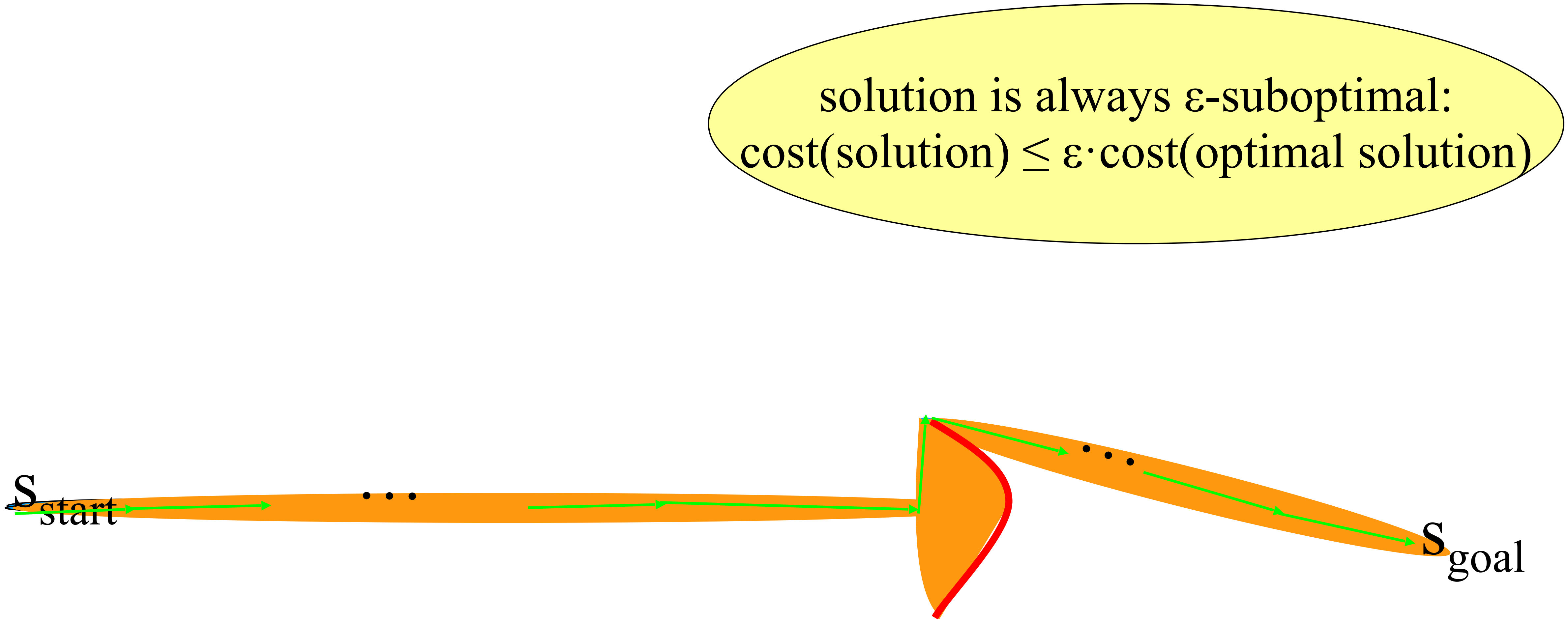
Weighted A* Search: expands states in the order of $f = g+\varepsilon h$ values, $\varepsilon > 1$ = bias towards states that are closer to goal



Courtesy Max Likhachev

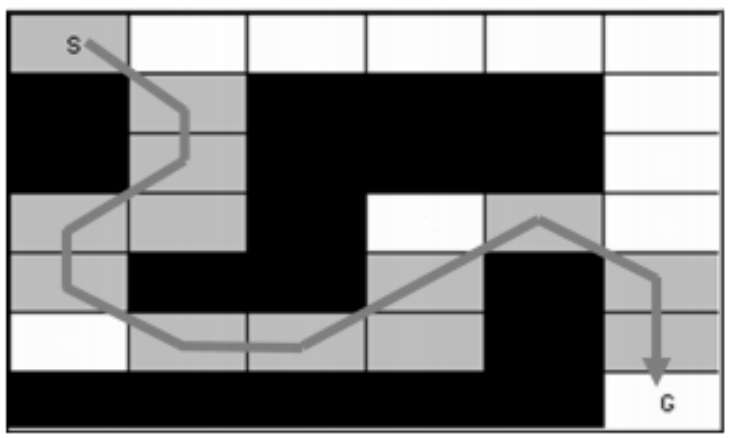
Effect of the Heuristic Function

Weighted A* Search: expands states in the order of $f = g + \varepsilon h$ values, $\varepsilon > 1$ = bias towards states that are closer to goal

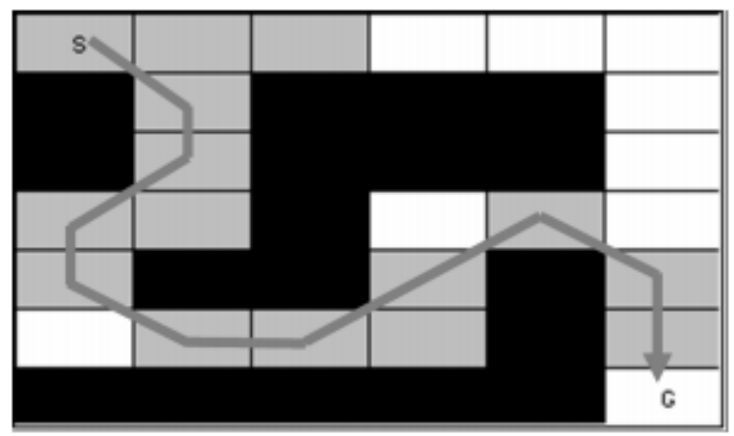


Courtesy Max Likhachev

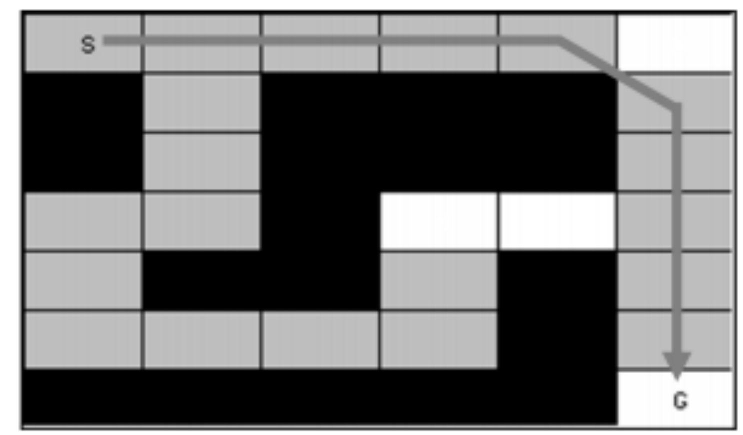
Effect of the Heuristic Function



$\epsilon = 2.5$



$\epsilon = 1.5$



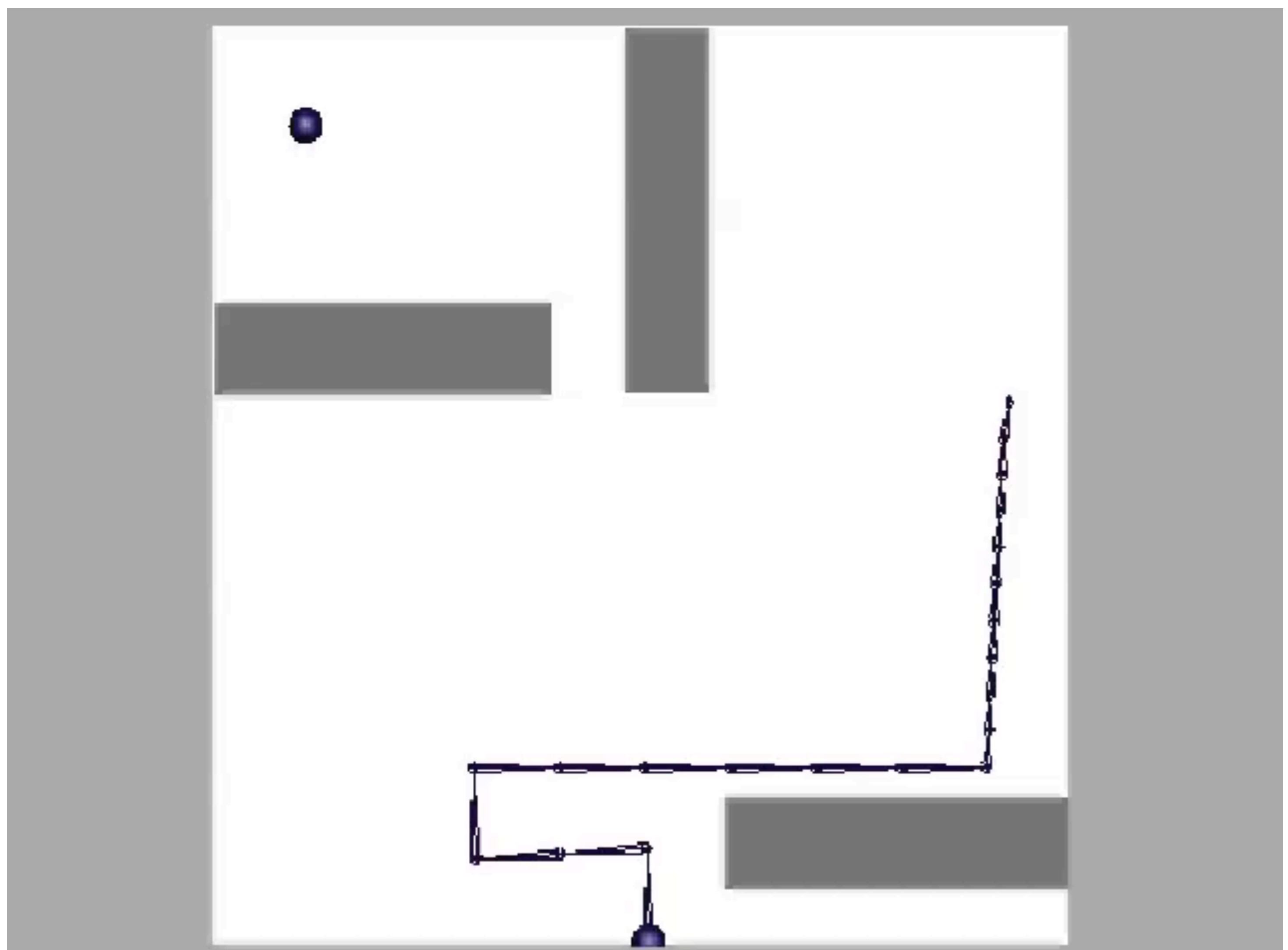
$\epsilon = 1.0$ (optimal search)

Courtesy Max Likhachev

Effect of the Heuristic Function

Weighted A* Search: expands states in the order of $f = g+\varepsilon h$ values, $\varepsilon > 1$ = bias towards states that are closer to goal

20DOF simulated robotic arm
state-space size: over 10^{26} states



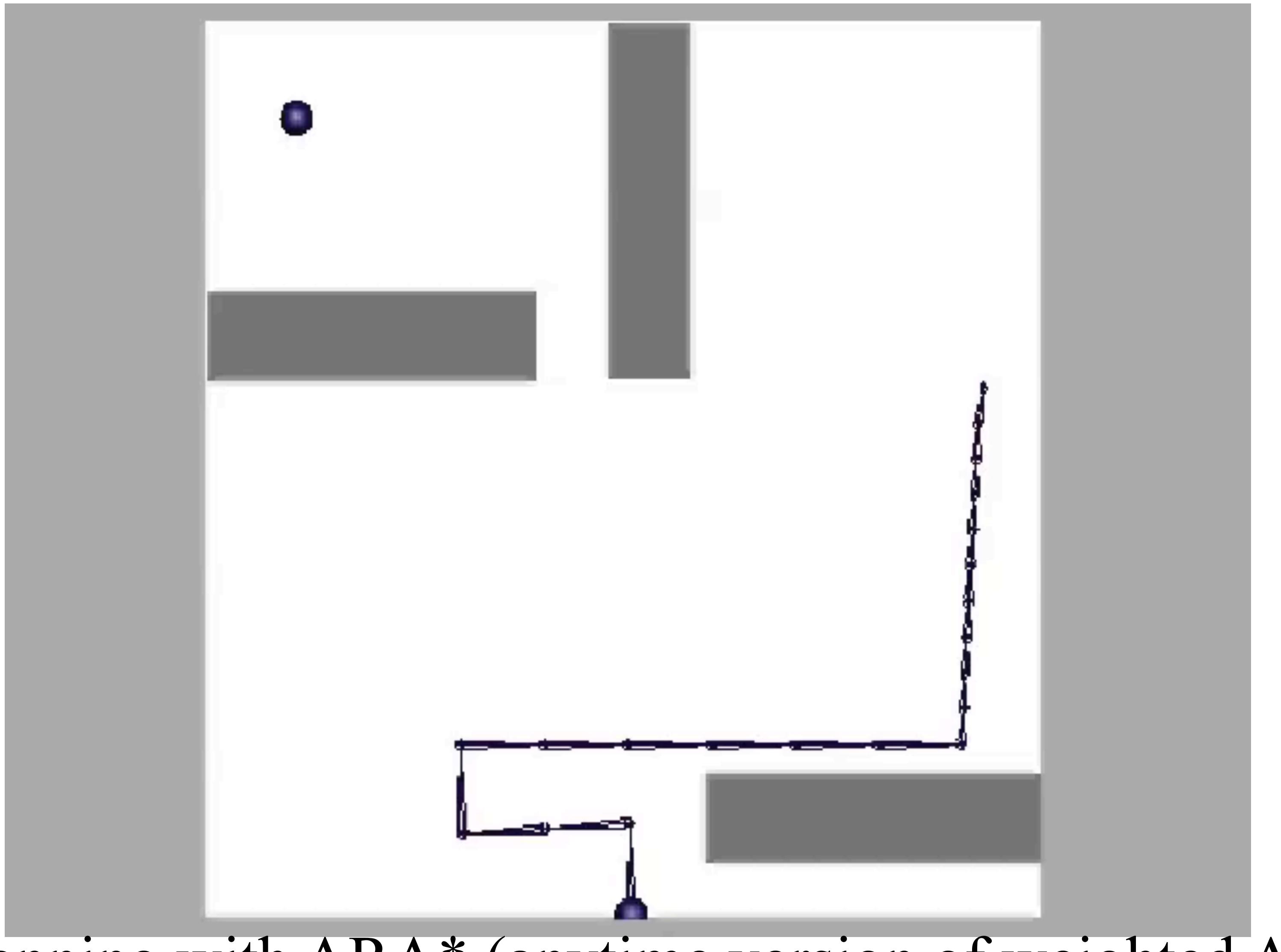
planning with ARA* (anytime version of weighted A*)

Courtesy Max Likhachev

Effect of the Heuristic Function

Weighted A* Search: expands states in the order of $f = g+\varepsilon h$ values, $\varepsilon > 1$ = bias towards states that are closer to goal

20DOF simulated robotic arm
state-space size: over 10^{26} states



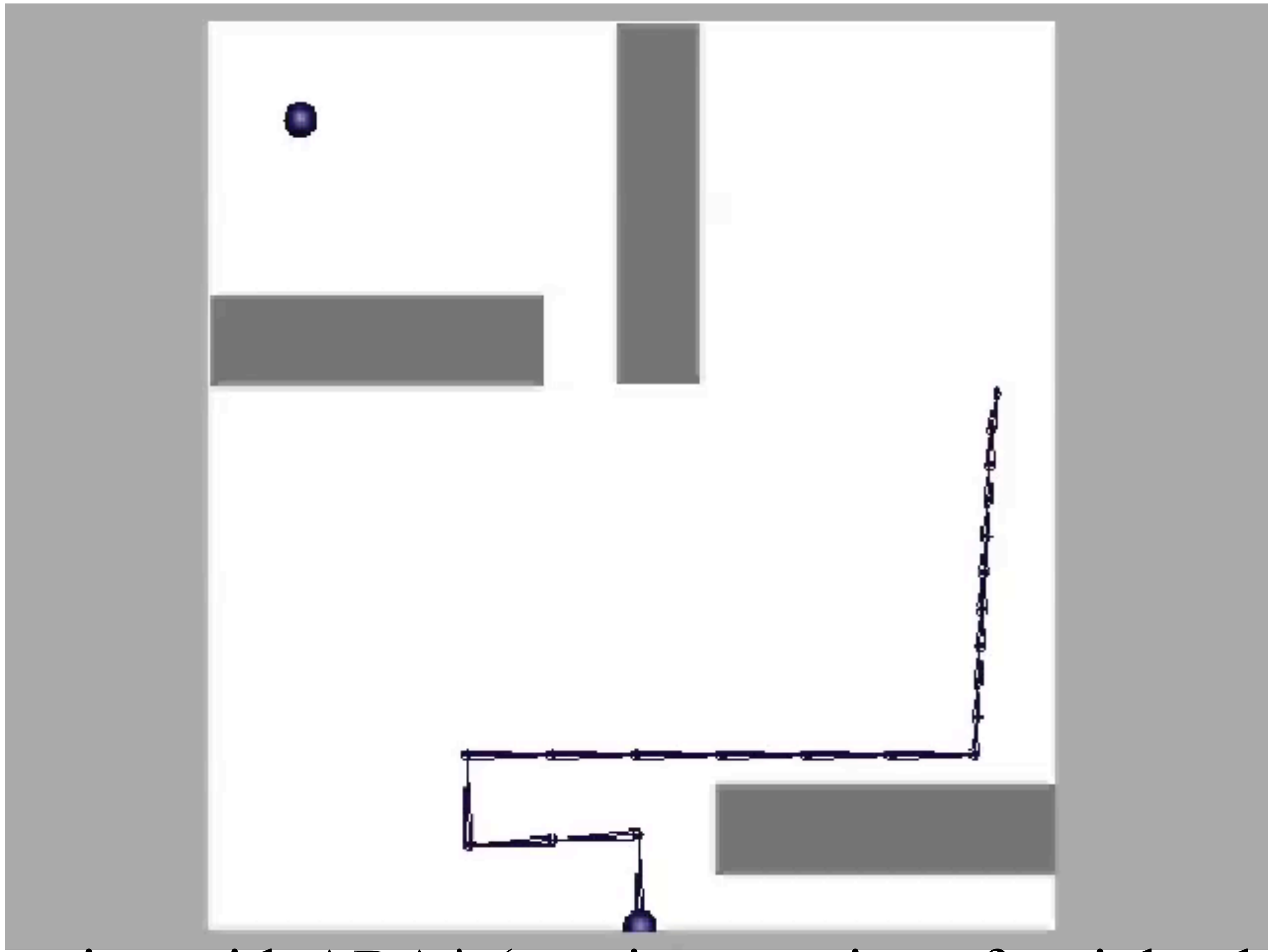
planning with ARA* (anytime version of weighted A*)

Courtesy Max Likhachev

Effect of the Heuristic Function

Weighted A* Search: expands states in the order of $f = g+\varepsilon h$ values, $\varepsilon > 1$ = bias towards states that are closer to goal

20DOF simulated robotic arm
state-space size: over 10^{26} states



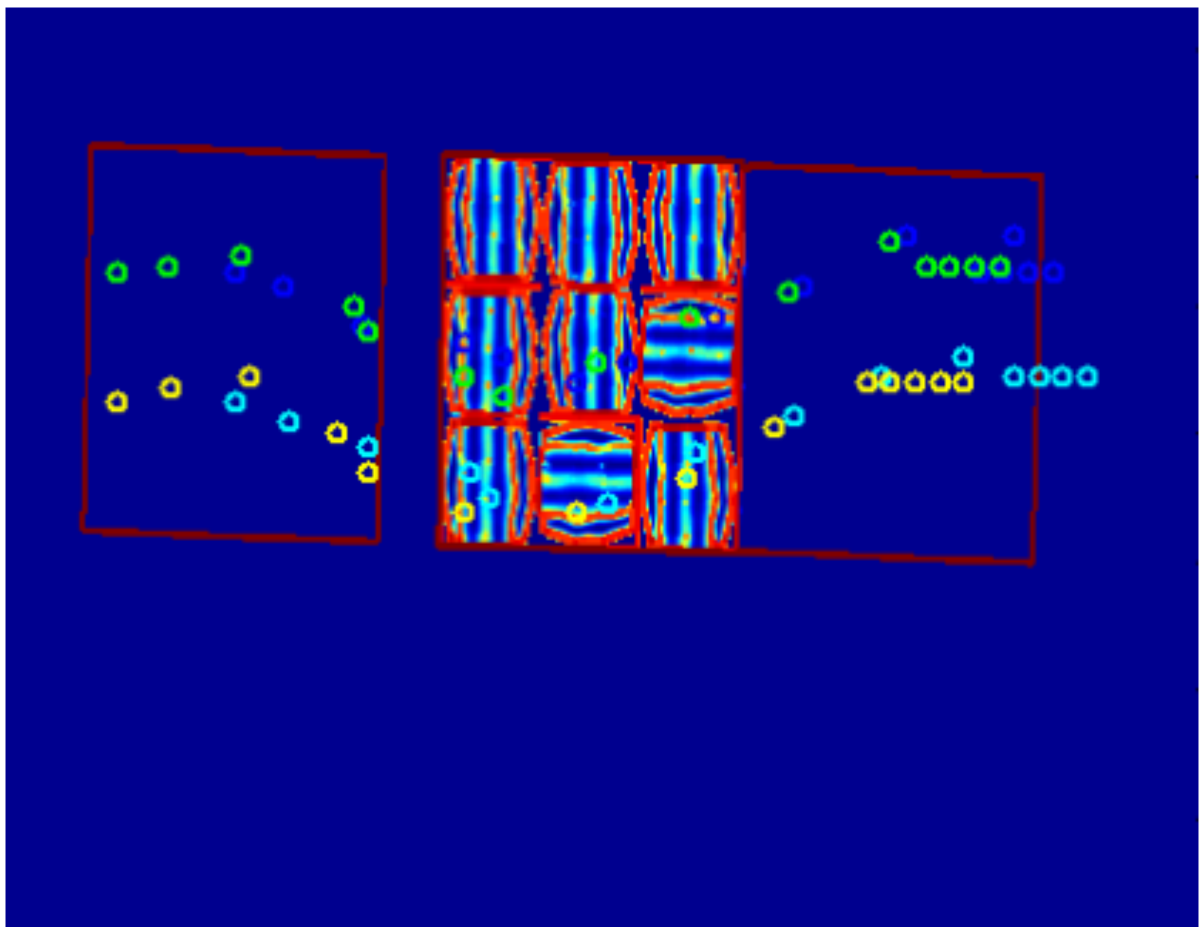
planning with ARA* (anytime version of weighted A*)

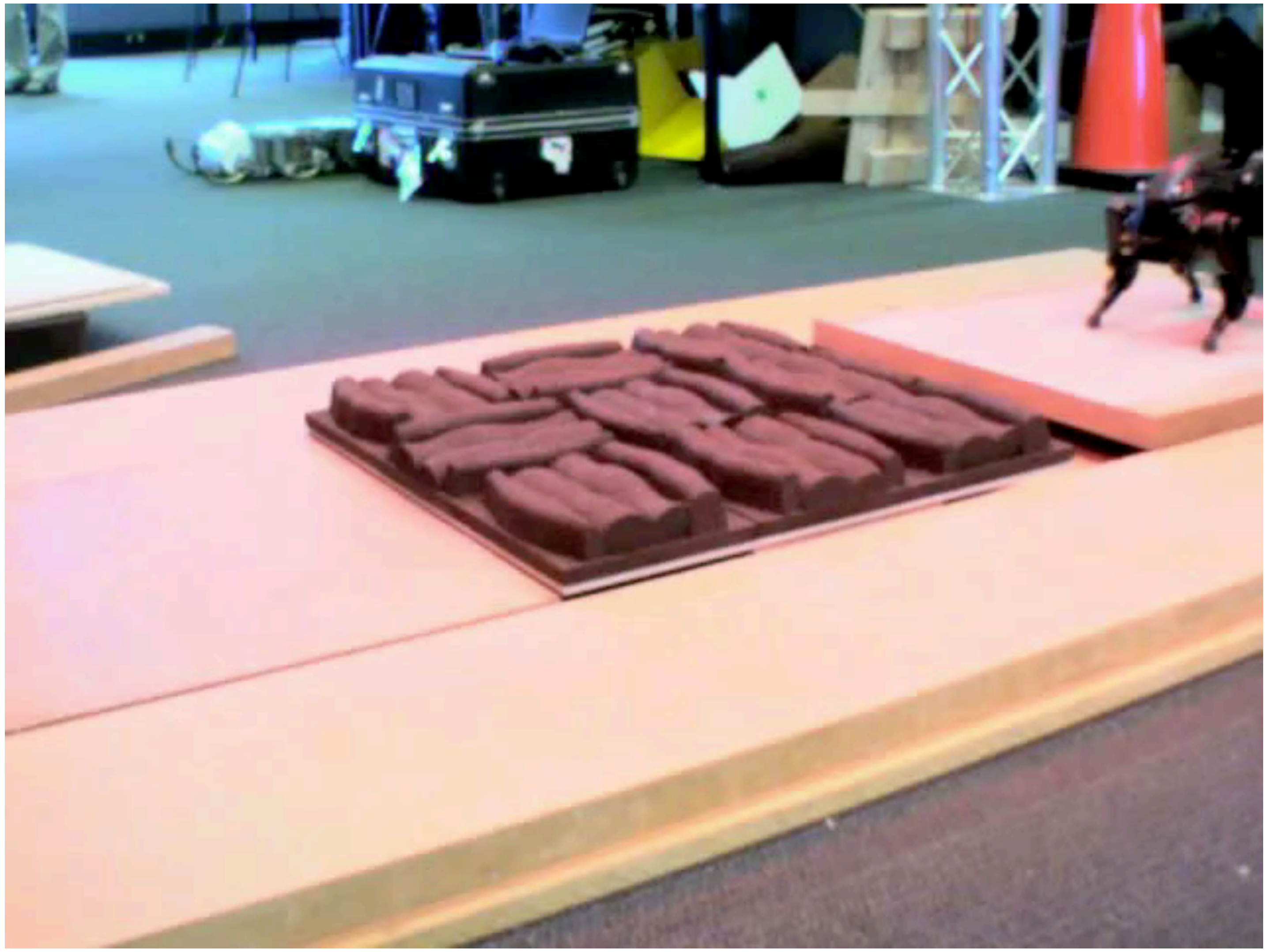
Courtesy Max Likhachev

Effect of the Heuristic Function

Courtesy Max Likhachev

- planning in 8D (<x,y> for each foothold)
- heuristic is Euclidean distance from the center of the body to the goal location
- cost of edges based on kinematic stability of the robot and quality of footholds





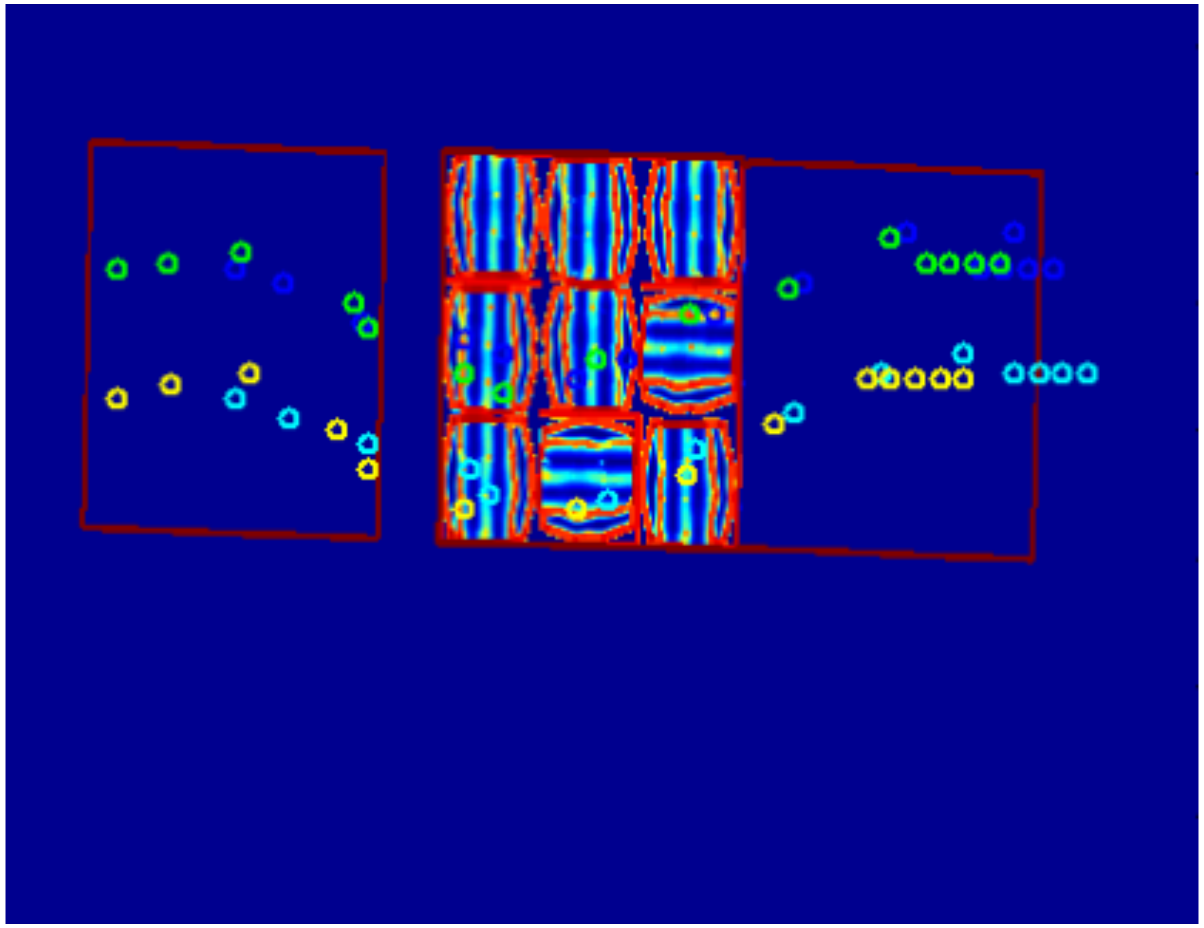
Uses R* - A randomized version of weighted A*

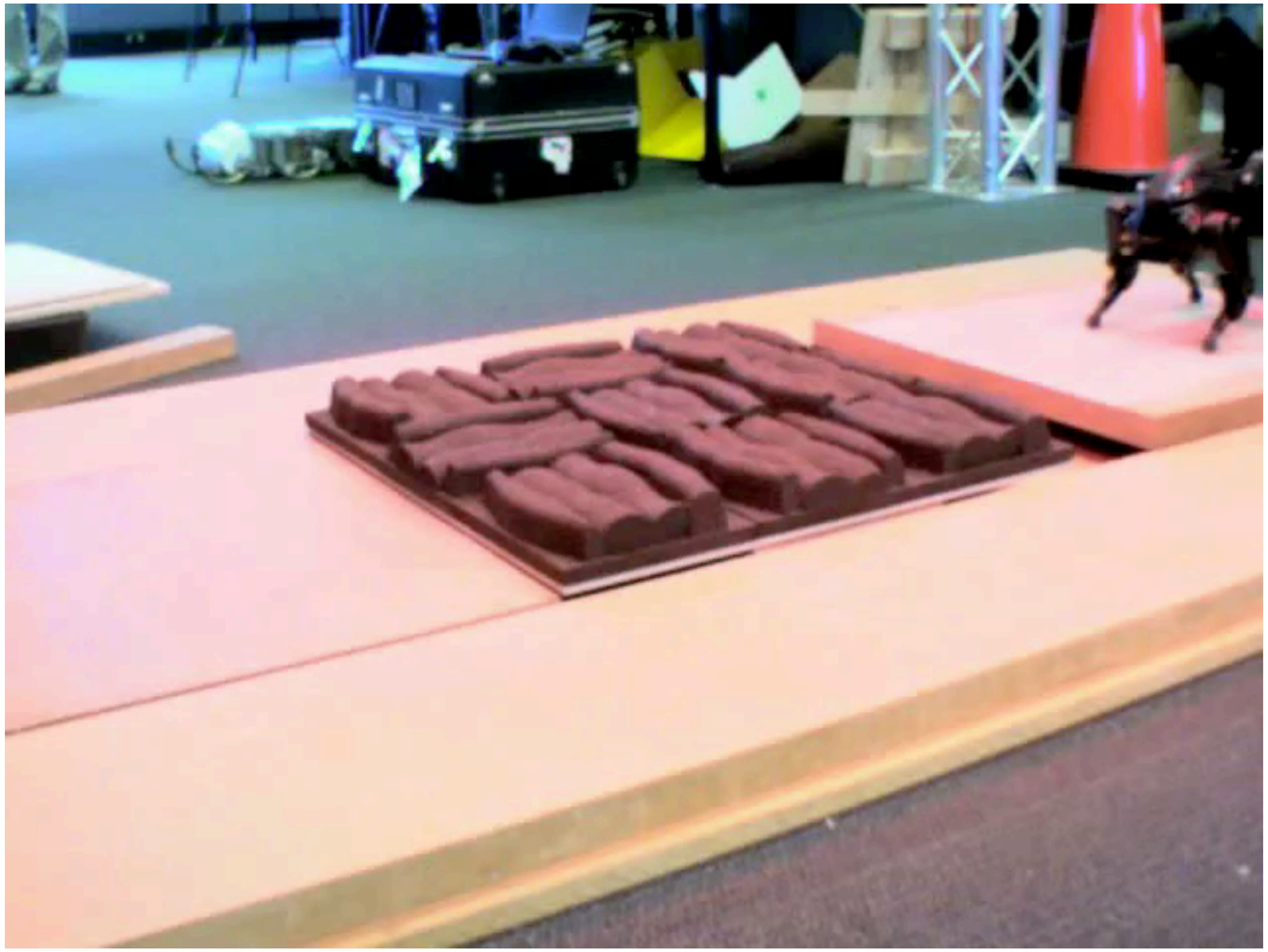
Joint work between Max Likhachev, Subhrajit Bhattacharya, Joh Bohren, Sachin Chitta, Daniel D. Lee, Aleksandr Kushleyev, and Paul Vernaza

Effect of the Heuristic Function

Courtesy Max Likhachev

- planning in 8D (<x,y> for each foothold)
- heuristic is Euclidean distance from the center of the body to the goal location
- cost of edges based on kinematic stability of the robot and quality of footholds



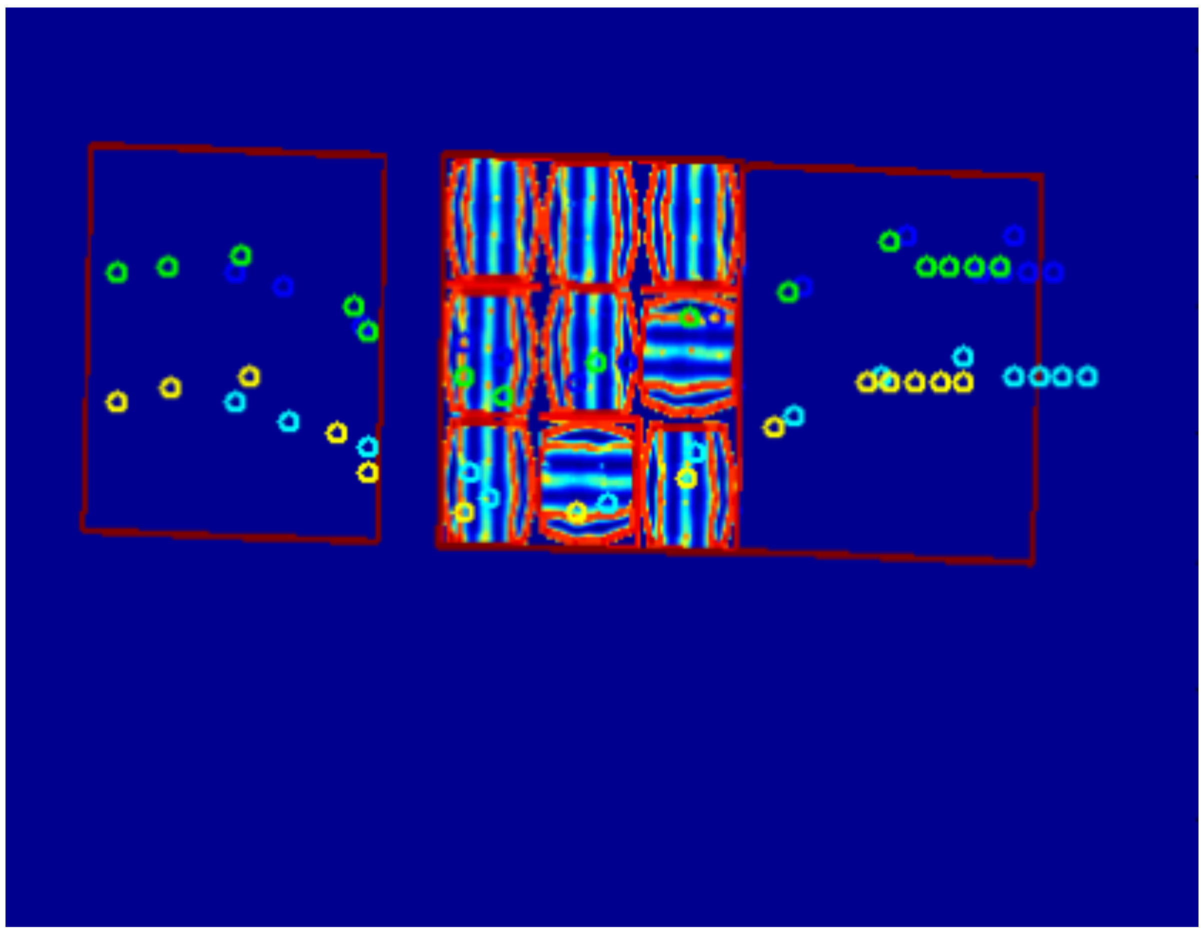


Uses R* - A randomized version of weighted A*
Joint work between Max Likhachev, Subhrajit Bhattacharya, Joh Bohren, Sachin Chitta, Daniel D. Lee, Aleksandr Kushleyev, and Paul Vernaza

Effect of the Heuristic Function

Courtesy Max Likhachev

- planning in 8D (<x,y> for each foothold)
- heuristic is Euclidean distance from the center of the body to the goal location
- cost of edges based on kinematic stability of the robot and quality of footholds



Uses R* - A randomized version of weighted A*
Joint work between Max Likhachev, Subhrajit Bhattacharya, Joh Bohren, Sachin Chitta, Daniel D. Lee, Aleksandr Kushleyev, and Paul Vernaza