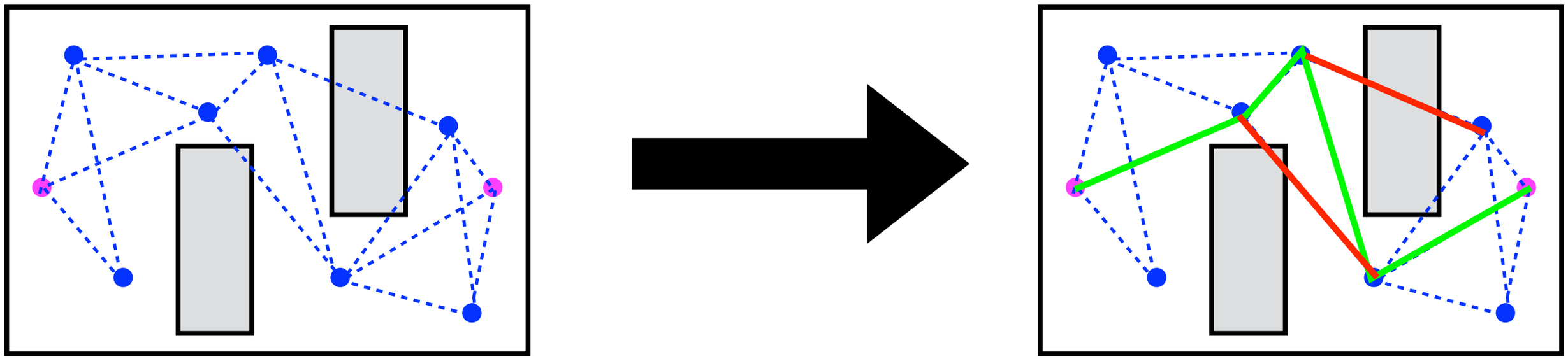
Heuristic Search

Instructor: Chris Mavrogiannis

TAs: Kay Ke, Gilwoo Lee, Matt Schmittle

*Slides based on or adapted from Sanjiban Choudhury, Steve Lavalle, Max Likhachev

General framework for motion planning



Create a graph

Search the graph

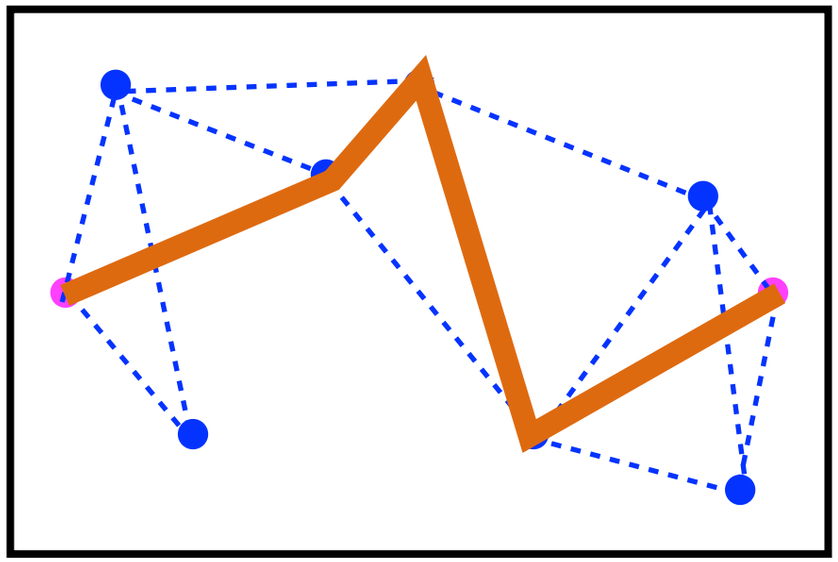
Interleave

General framework for motion planning

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

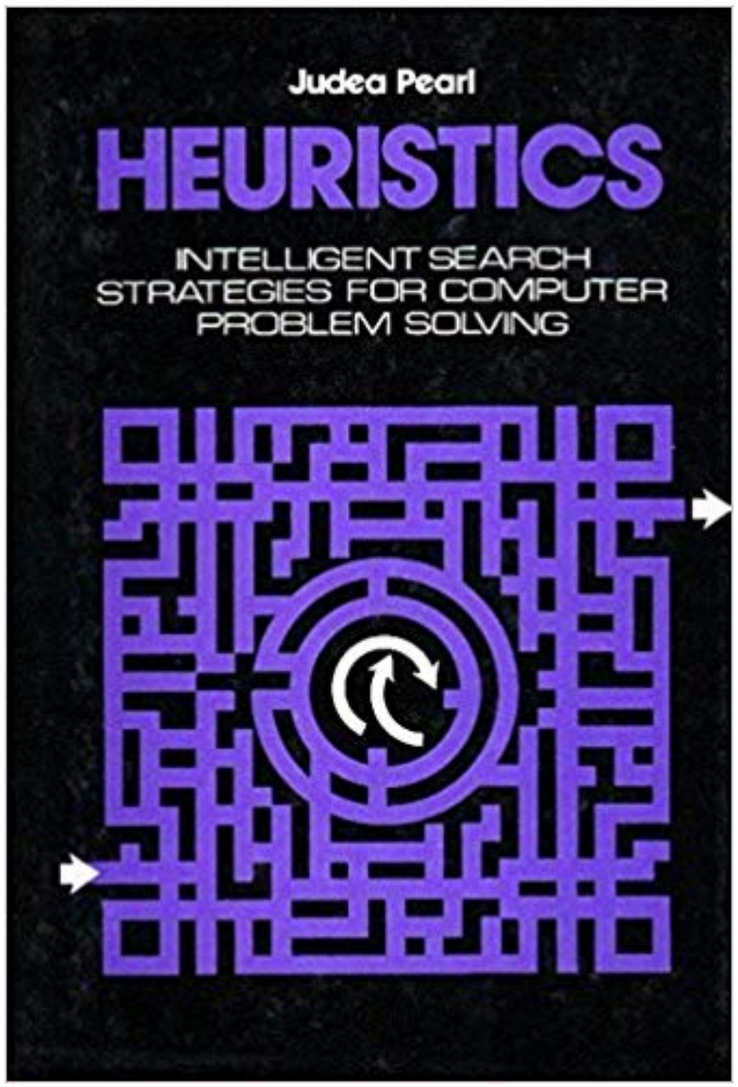
For this lecture....

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

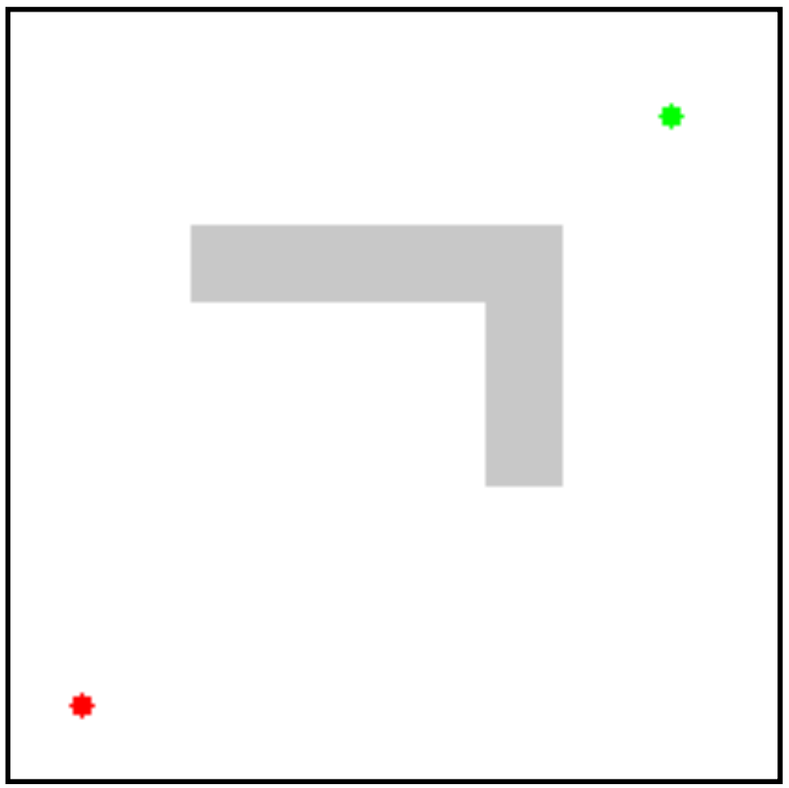
2. Dijkstra's Algorithm as a Best-First Search

3. A* as another instance of Best-First Search

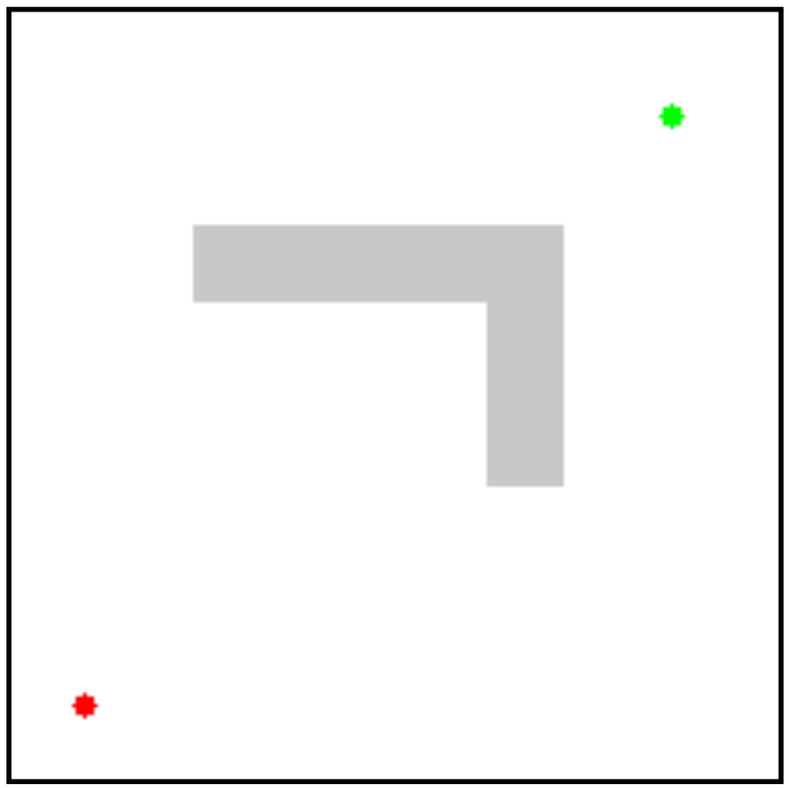
4. Heuristics for guiding Search

High-order bit

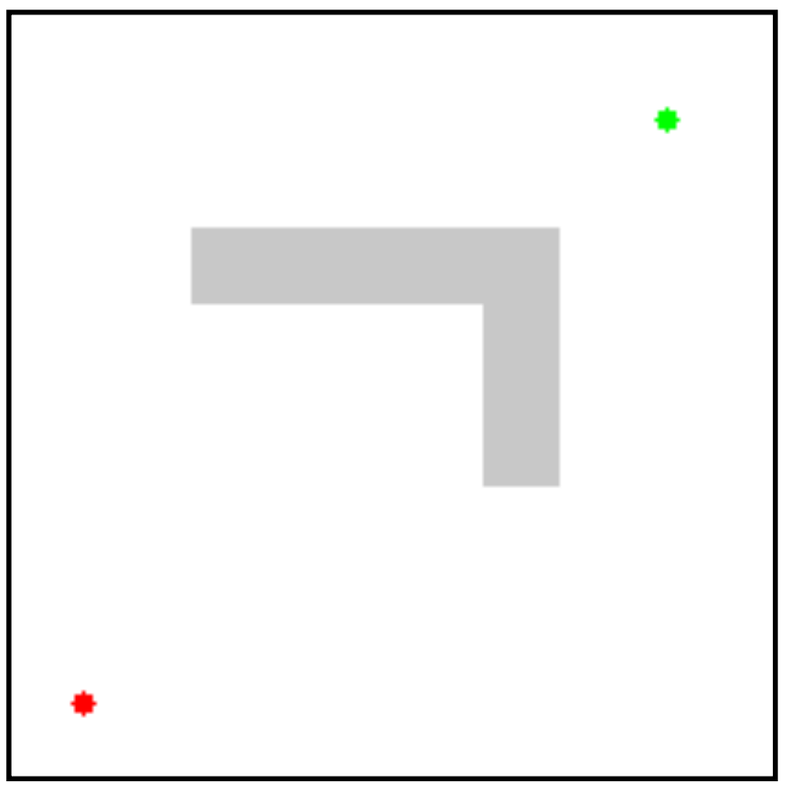
Expansion of a search wavefront from start to goal



Dijkstra



A*



Weighted A*

Courtesy wikipedia

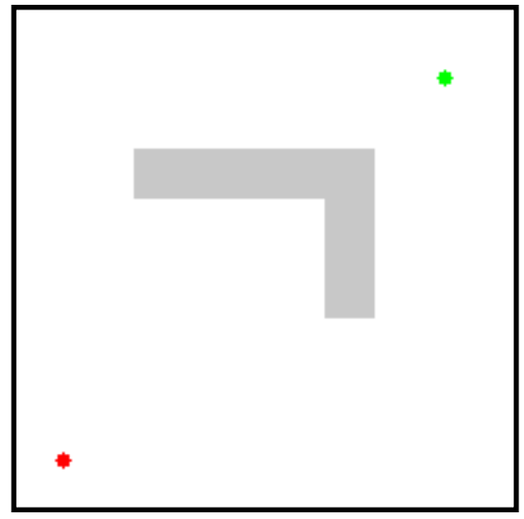
What do we want?

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

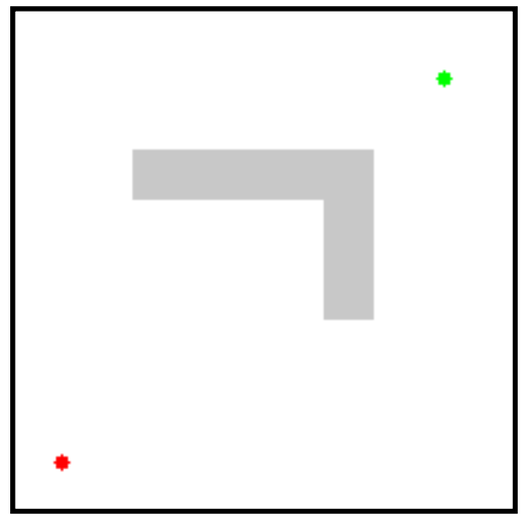
BFS maintains a priority queue of promising nodes

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)
.....	.....

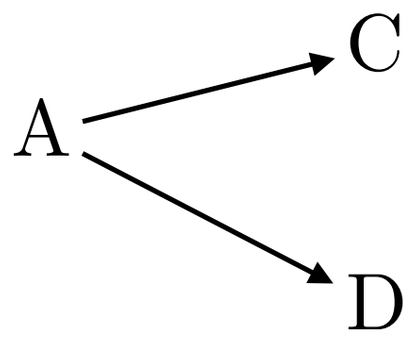
Best-First search



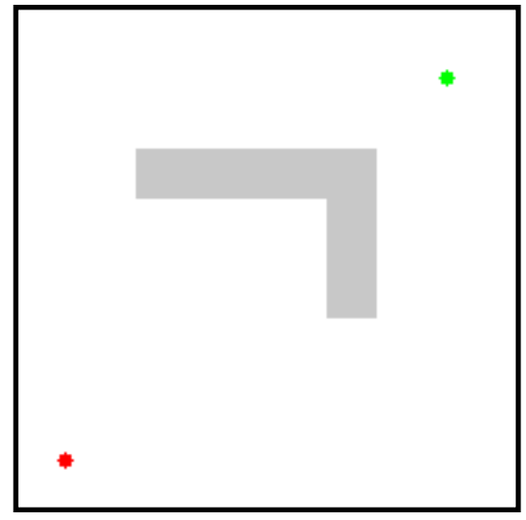
Search explores graph by expanding to most promising node min *f(s)*

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

Notation

Given:

Start s_{start} Goal s_{goal}

Cost $c(s, s')$

Graph $G(V,E)$

Objects created:

OPEN: priority queue of nodes that have not been visited

CLOSED: list of visited nodes

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

Best-First Search

Set $g(s_{start}) = 0$; all other g-values to inf; Set $OPEN/CLOSED$ to $empty$

Insert s_{start} into $OPEN$

While s_{goal} *not expanded* **do**

Remove s with the smallest $f(s)$ from $OPEN$

Insert s to $CLOSED$

For every successor s' **do**

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

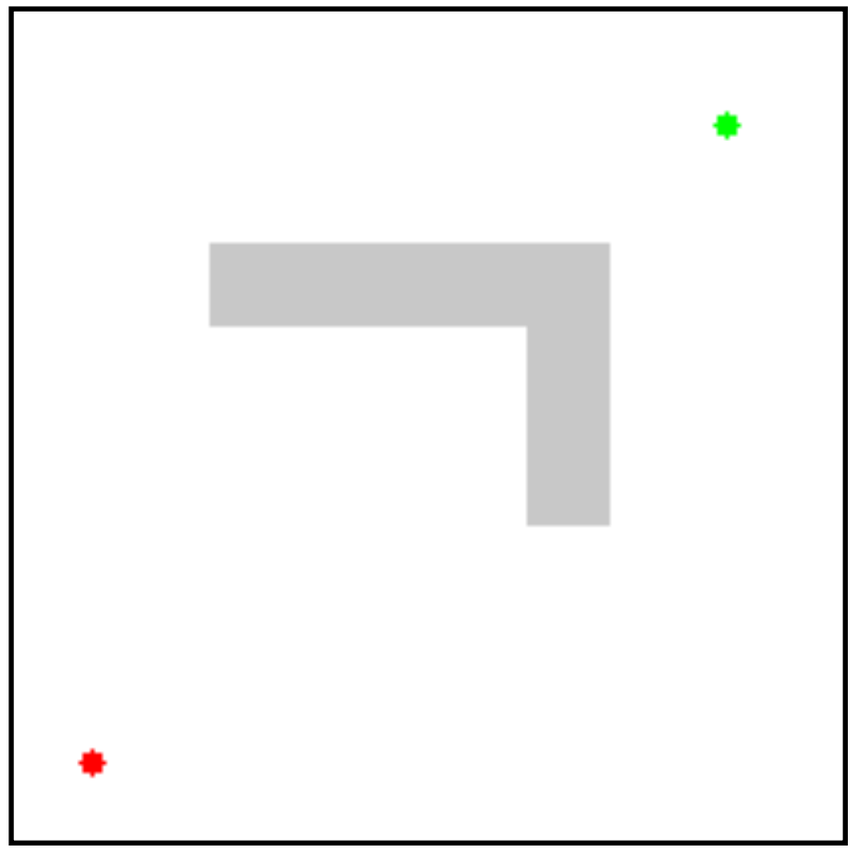
$g(s') = g(s) + c(s,s')$

Insert s' to $OPEN$

Dijkstra's Algorithm

Best-First with

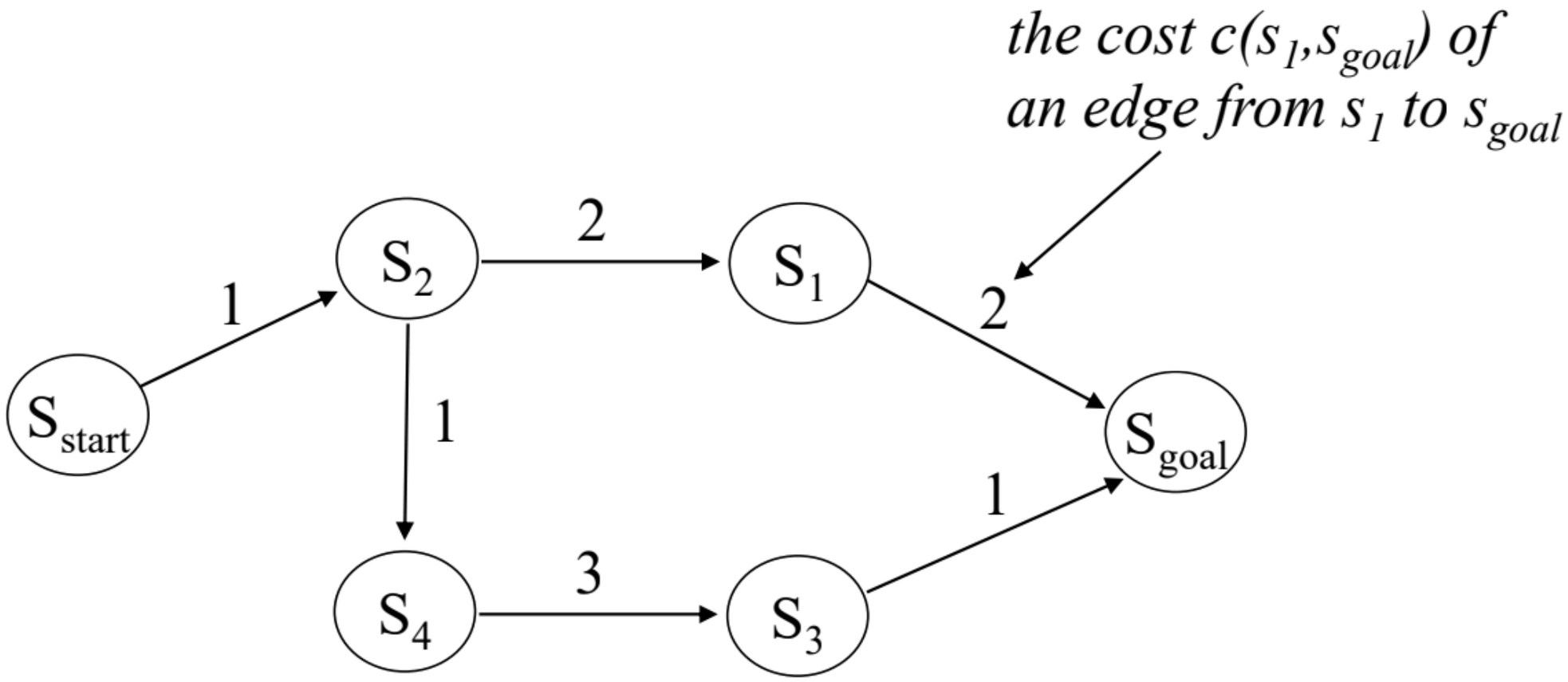
$f(s) = g(s)$



estimate of the least cost
from start to a given node

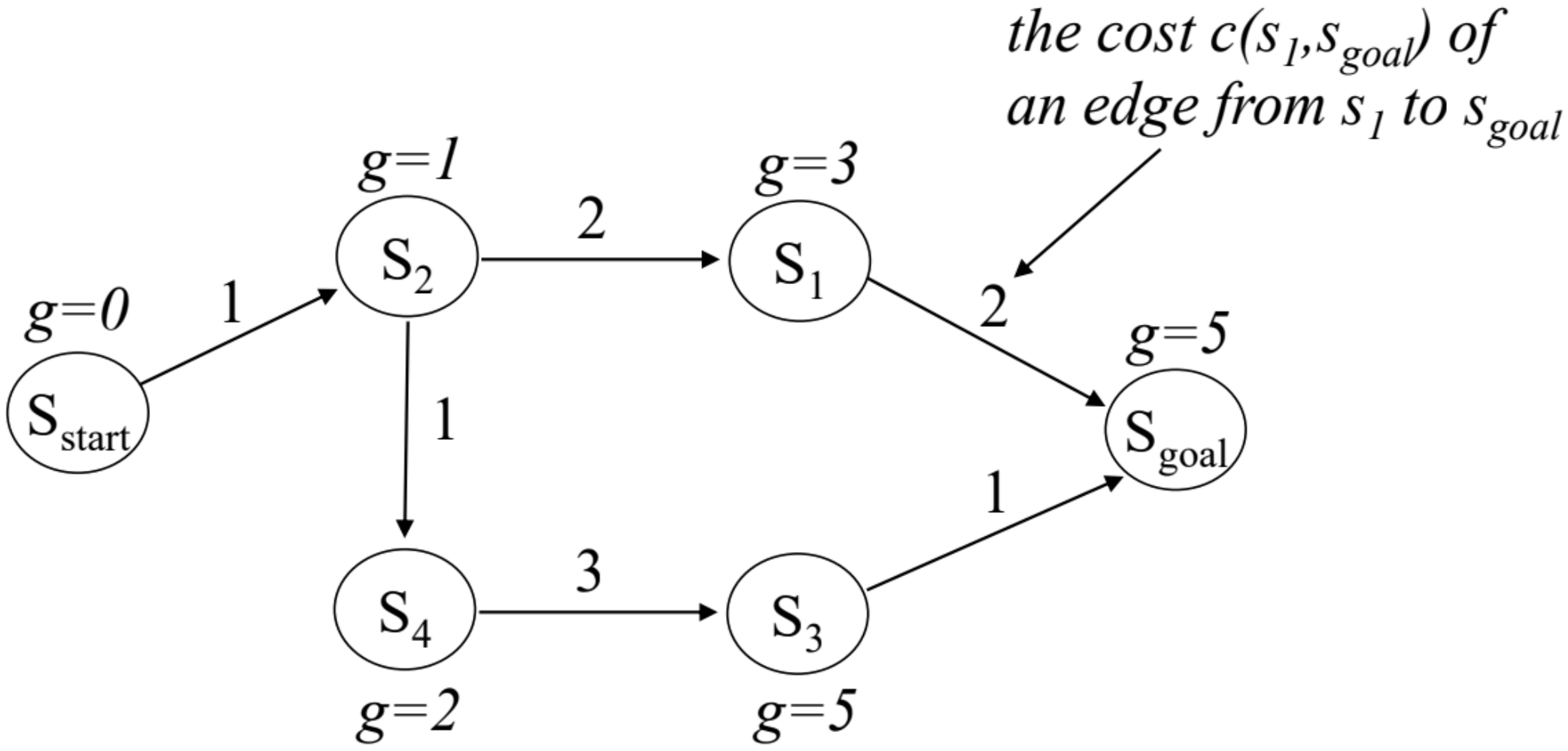
Dijkstra's Algorithm

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



Dijkstra'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.

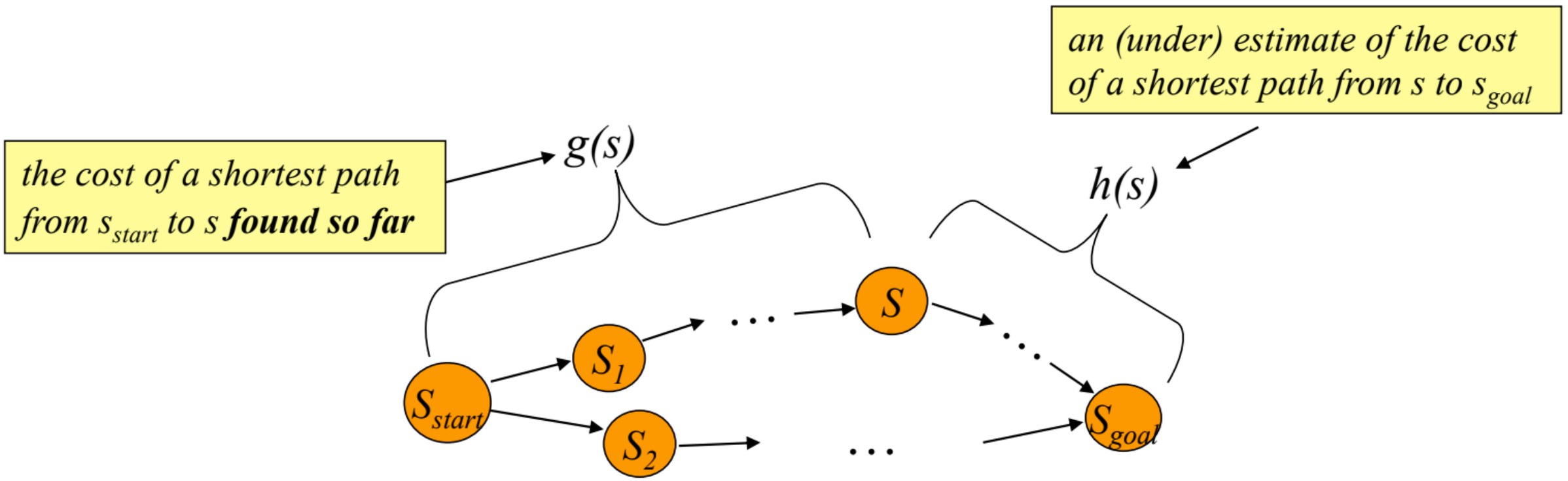
Can we have a better f(s)?

Yes!

Wouldn't it be better if f(s) "knew" the goal?

Heuristics

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

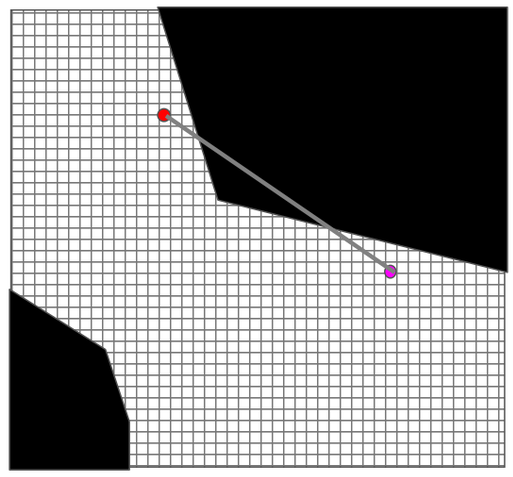


Set the evaluation function $f(s) = g(s) + h(s)$

Example of heuristics?

1. Minimum number of nodes to go to goal

2. Euclidean distance to goal



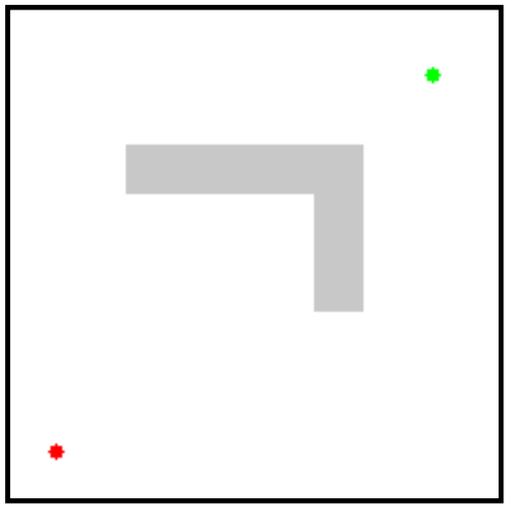
3. Solution to a relaxed problem

4. Domain knowledge / Learning ....

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

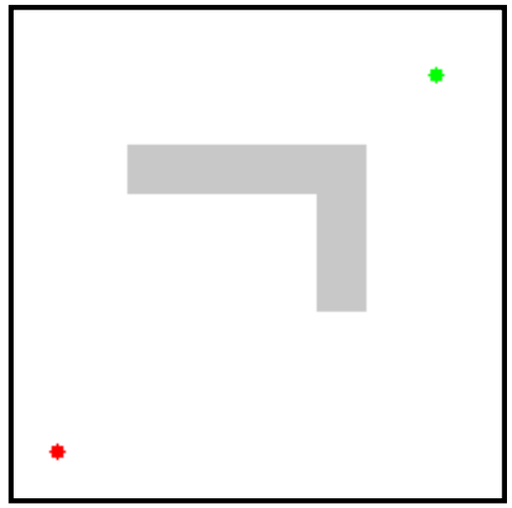
Let L be the length of the shortest path

Dijkstra



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

Set *f*-values, *g*-values to *inf*; Set *OPEN/CLOSED* to *empty*; *h* given

Insert s_{start} into *OPEN*

While s_{goal} *not expanded* **do**

Remove *s* with the smallest *f(s) = g(s)+h(s)* from *OPEN*

Insert *s* to *CLOSED*

For every successor s', *not* in *CLOSED* **do**

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

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

Insert s' to *OPEN*

Computes **optimal** g-values for relevant states!

A* Search

Set *f*-values, *g*-values to *inf*; Set *OPEN/CLOSED* to *empty*; *h* given

Insert s_{start} into *OPEN*

While s_{goal} *not expanded* **do**

Remove *s* with the smallest $f(s) = g(s)+h(s)$ from *OPEN*

Insert *s* to *CLOSED*

For every successor s', *not* in *CLOSED* **do**

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

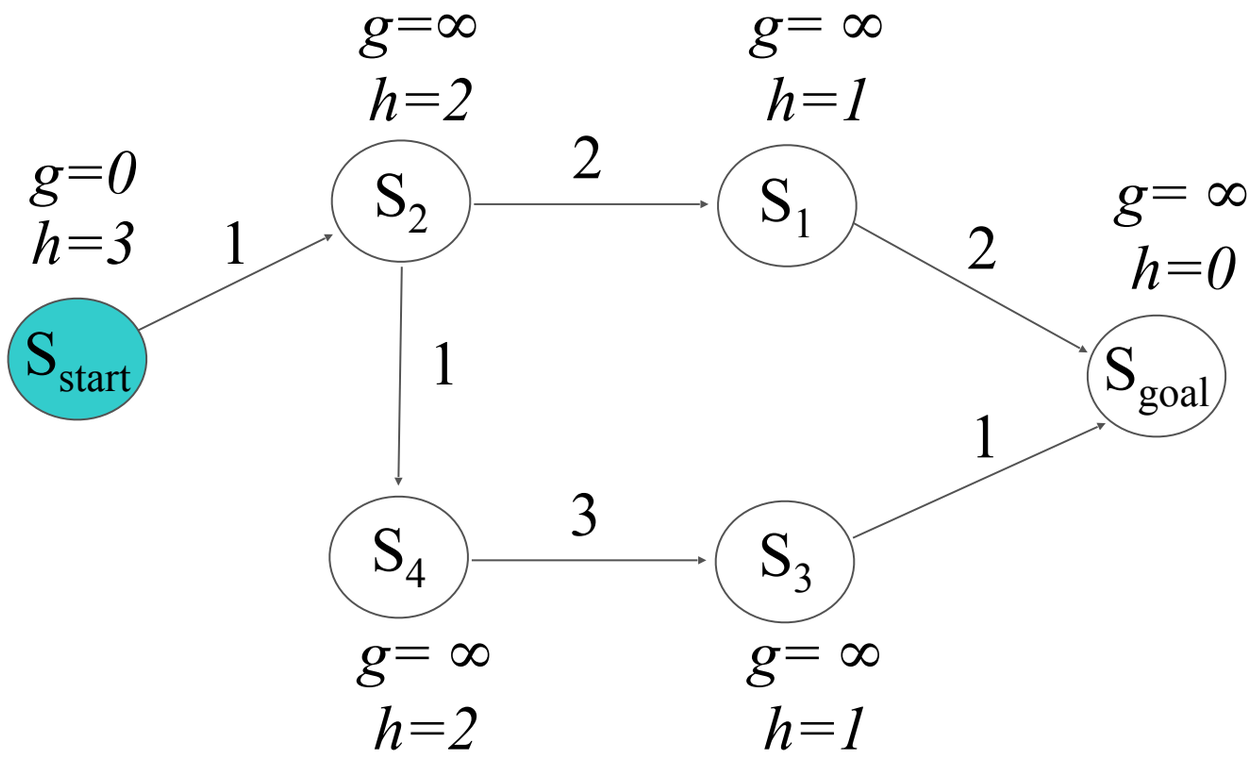
$g(s') = g(s) + c(s,s')$

Insert s' to *OPEN*

CLOSED = {}

OPEN = $\{s_{start}\}$

next state to expand: s_{start}



A* Search

Set *f*-values, *g*-values to *inf*; Set *OPEN/CLOSED* to *empty*; *h* given

Insert s_{start} into *OPEN*

While s_{goal} *not expanded* **do**

Remove *s* with the smallest $f(s) = g(s)+h(s)$ from *OPEN*

Insert *s* to *CLOSED*

For every successor s', *not* in *CLOSED* **do**

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

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

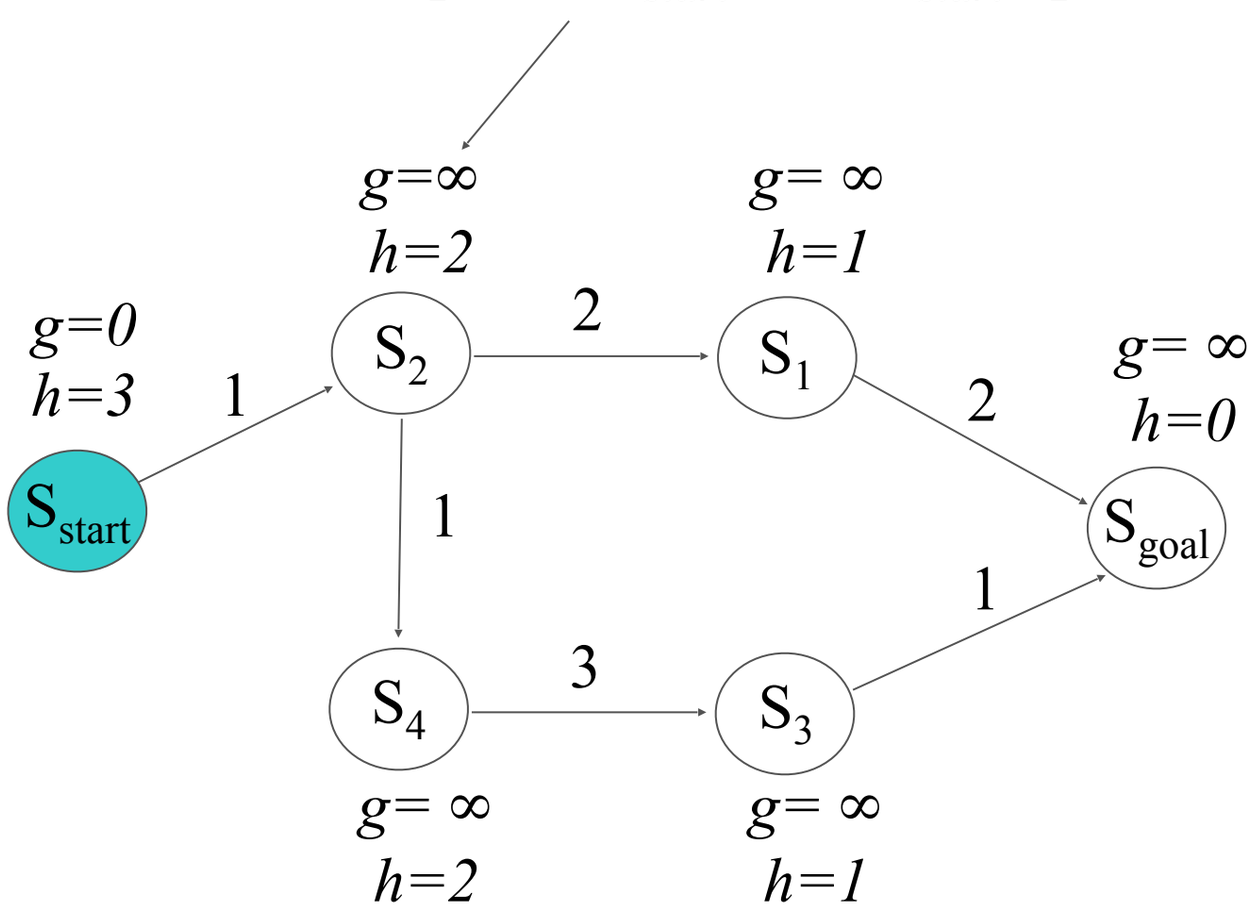
Insert s' to *OPEN*

$g(s_2) > g(s_{start}) + c(s_{start},s_2)$

CLOSED = {}

OPEN = $\{s_{start}\}$

next state to expand: s_{start}



A* Search

Set *f*-values, *g*-values to *inf*; Set *OPEN/CLOSED* to *empty*; *h* given

Insert s_{start} into *OPEN*

While s_{goal} *not expanded* **do**

Remove *s* with the smallest $f(s) = g(s)+h(s)$ from *OPEN*

Insert *s* to *CLOSED*

For every successor s', *not* in *CLOSED* **do**

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

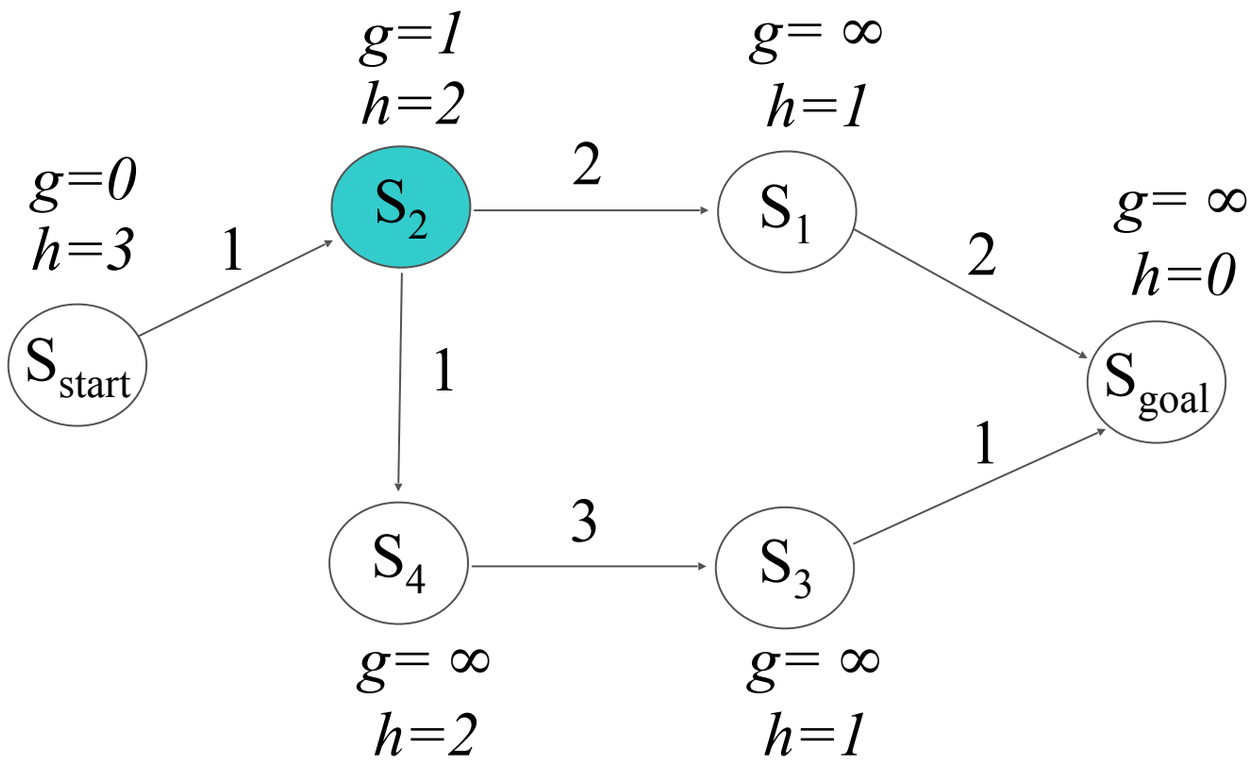
$g(s') = g(s) + c(s,s')$

Insert s' to *OPEN*

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

$OPEN = \{s_2\}$

next state to expand: s_2



A* Search

Set *f*-values, *g*-values to *inf*; Set *OPEN/CLOSED* to *empty*; *h* given

Insert s_{start} into *OPEN*

While s_{goal} *not expanded* **do**

Remove *s* with the smallest *f(s) = g(s)+h(s)* from *OPEN*

Insert *s* to *CLOSED*

For every successor s', *not* in *CLOSED* **do**

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

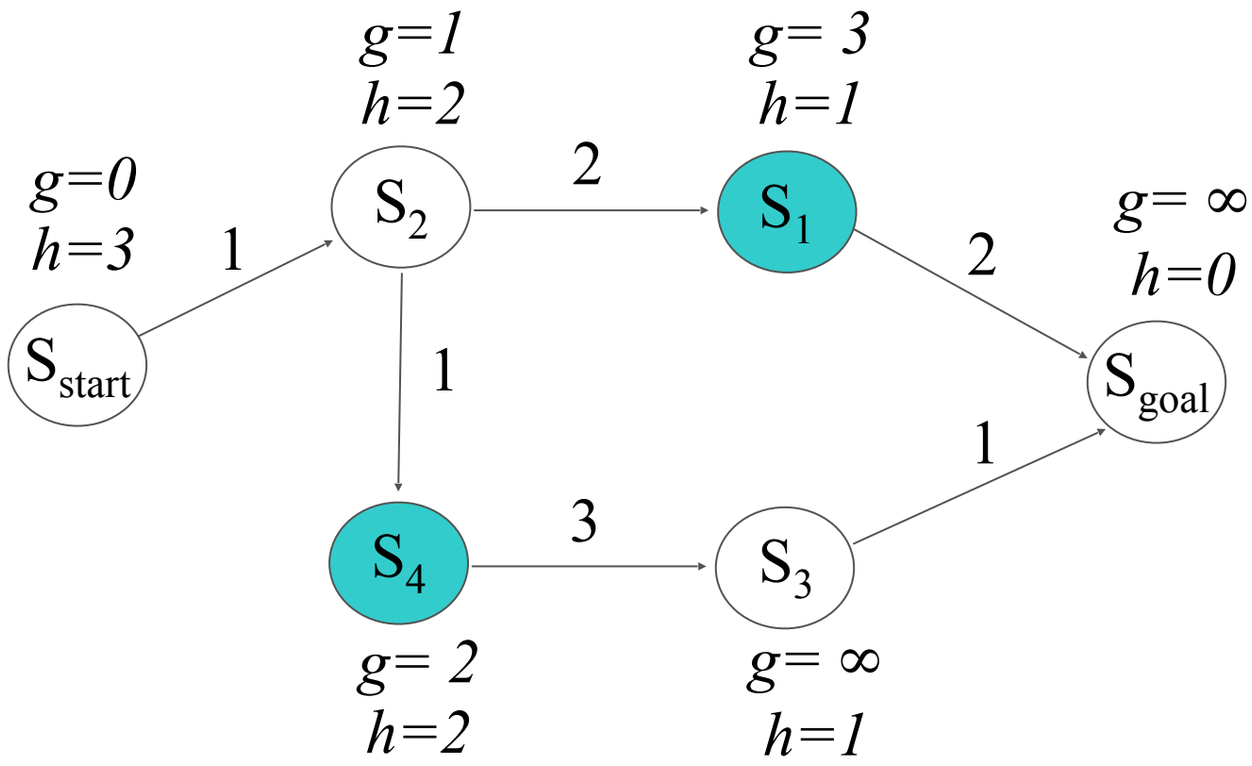
g(s') = g(s) + c(s,s')

Insert s' to *OPEN*

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

$OPEN = \{s_1, s_4\}$

next state to expand: s_1



A* Search

Set *f*-values, *g*-values to *inf*; Set *OPEN/CLOSED* to *empty*; *h* given

Insert s_{start} into *OPEN*

While s_{goal} *not expanded* **do**

Remove *s* with the smallest $f(s) = g(s)+h(s)$ from *OPEN*

Insert *s* to *CLOSED*

For every successor s', *not* in *CLOSED* **do**

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

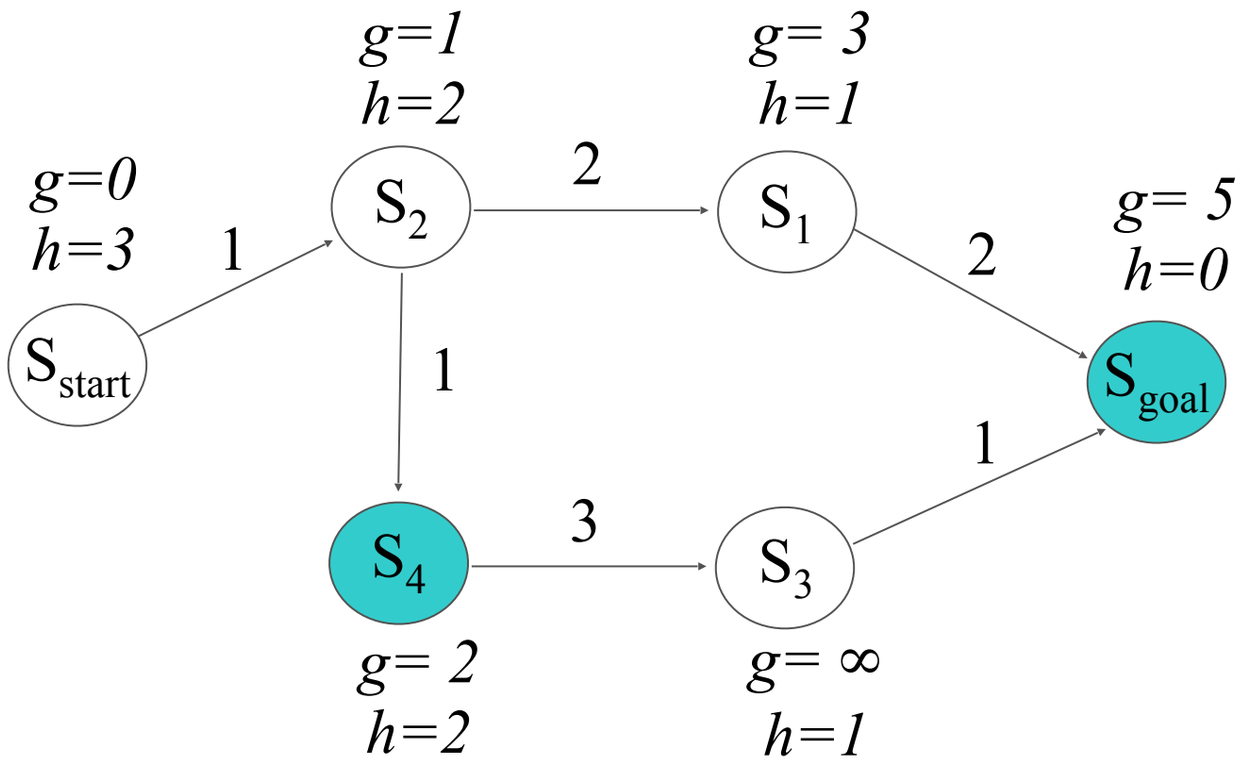
$g(s') = g(s) + c(s,s')$

Insert s' to *OPEN*

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

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

next state to expand: s_4



A* Search

Set *f*-values, *g*-values to *inf*; Set *OPEN/CLOSED* to *empty*; *h* given

Insert s_{start} into *OPEN*

While s_{goal} *not expanded* **do**

Remove *s* with the smallest $f(s) = g(s)+h(s)$ from *OPEN*

Insert *s* to *CLOSED*

For every successor s', *not* in *CLOSED* **do**

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

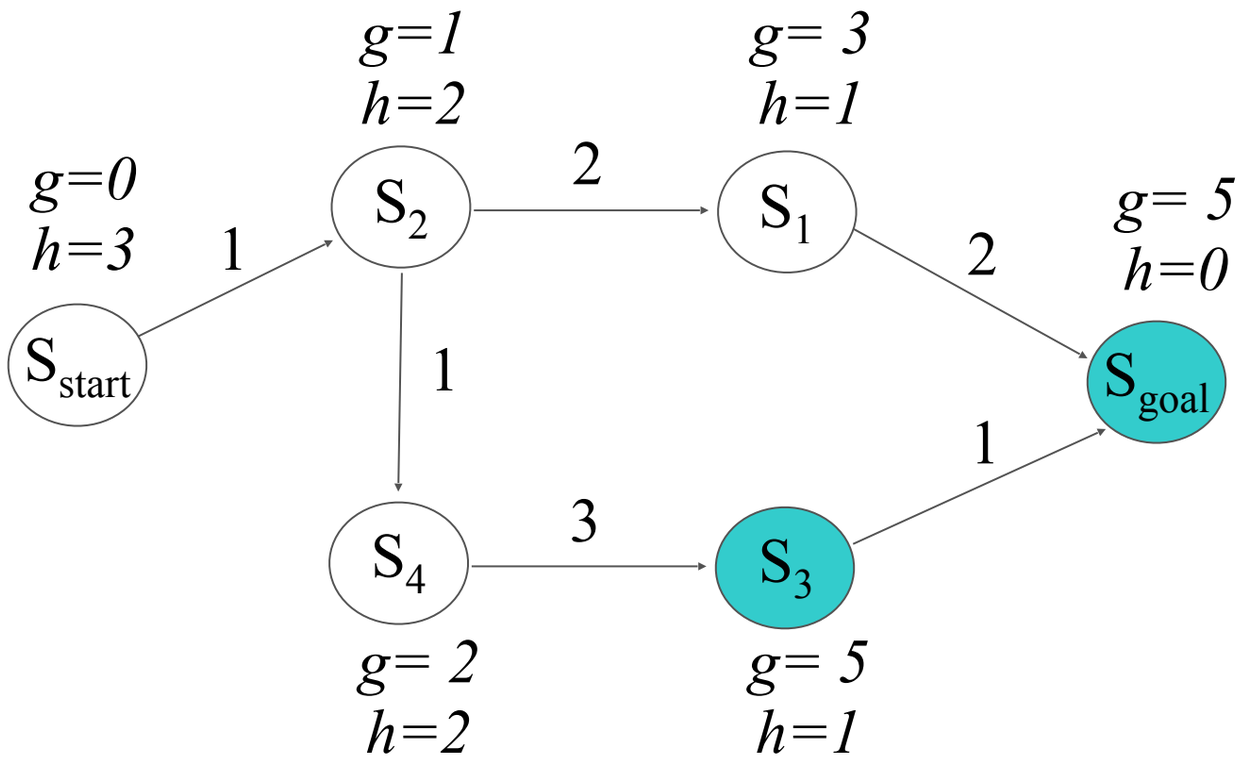
$g(s') = g(s) + c(s,s')$

Insert s' to *OPEN*

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

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

next state to expand: s_{goal}



A* Search

Set *f*-values, *g*-values to *inf*; Set *OPEN/CLOSED* to *empty*; *h* given

Insert s_{start} into *OPEN*

While s_{goal} *not expanded* **do**

Remove *s* with the smallest $f(s) = g(s)+h(s)$ from *OPEN*

Insert *s* to *CLOSED*

For every successor s', *not* in *CLOSED* **do**

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

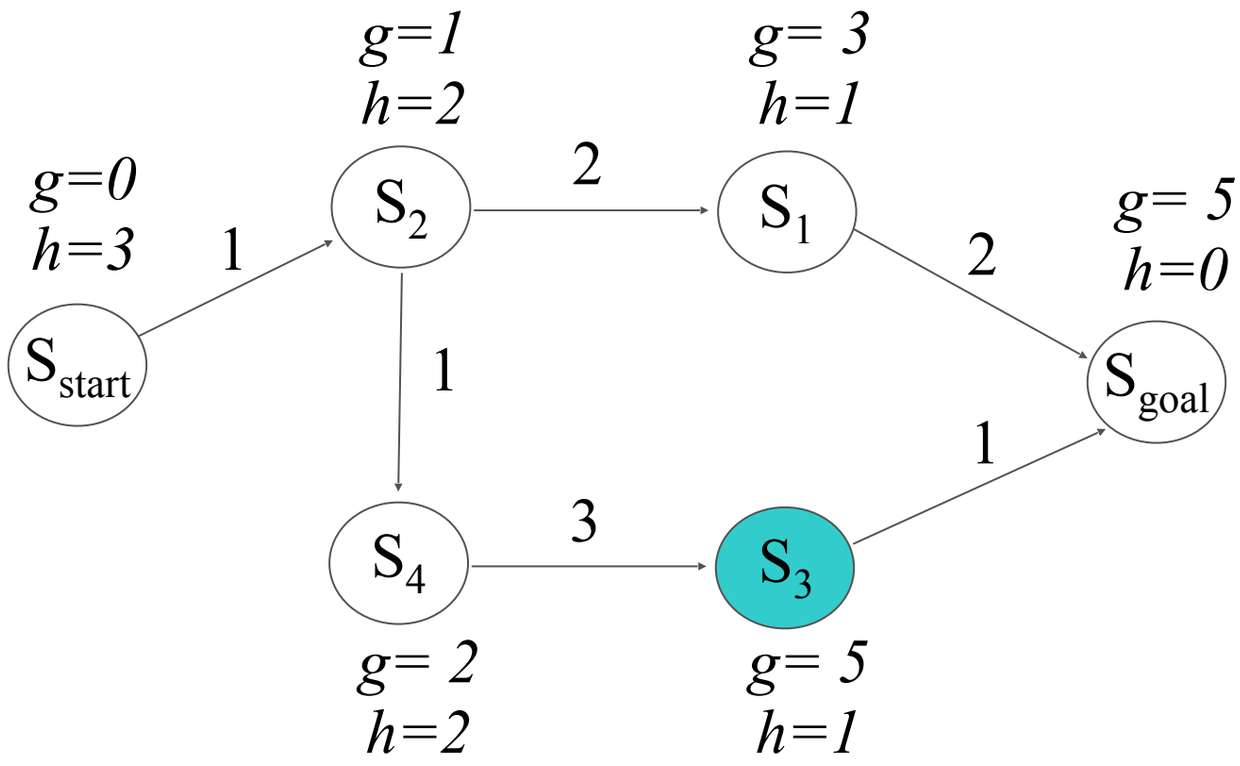
$g(s') = g(s) + c(s,s')$

Insert s' to *OPEN*

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

$OPEN = \{s_3\}$

done



A* Search

Set *f*-values, *g*-values to *inf*; Set *OPEN/CLOSED* to *empty*; *h* given

Insert s_{start} into *OPEN*

While s_{goal} *not expanded* **do**

Remove *s* with the smallest $f(s) = g(s)+h(s)$ from *OPEN*

Insert *s* to *CLOSED*

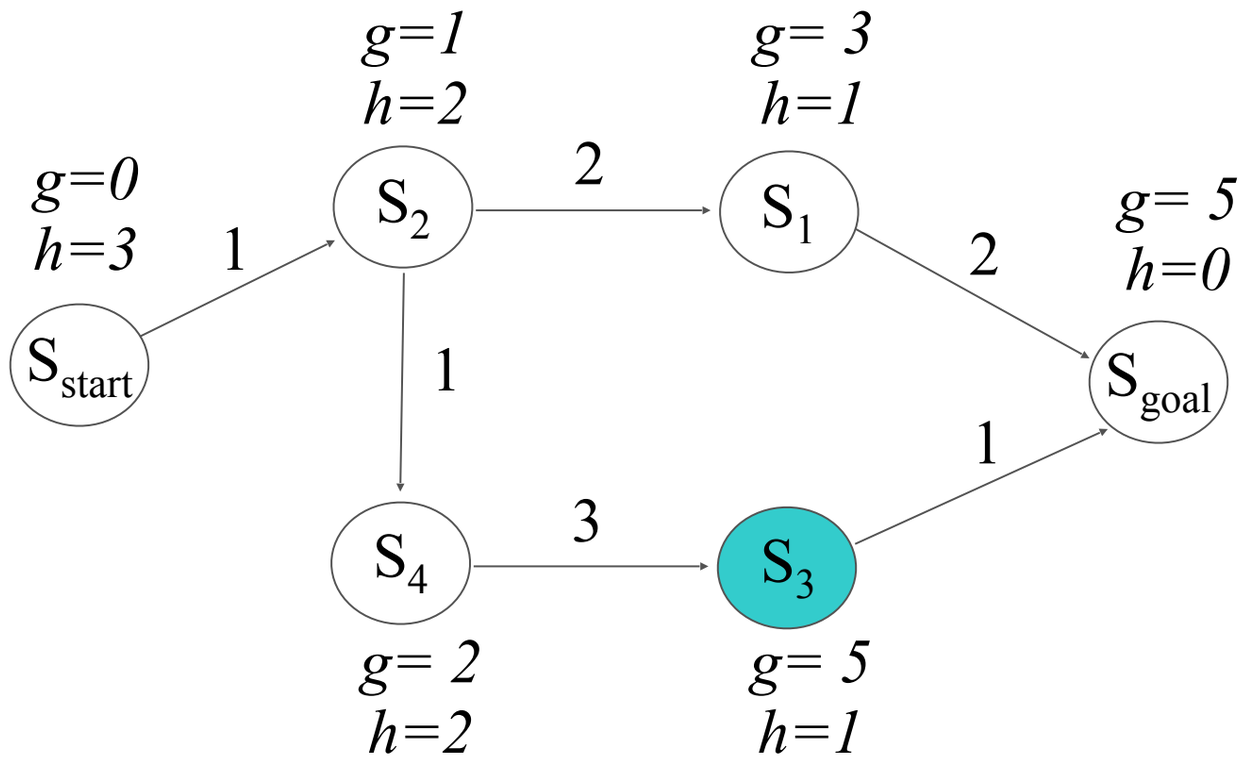
For every successor s', *not* in *CLOSED* **do**

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

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

Insert s' to *OPEN*

Output path that minimizes Σg



A* Search

Set *f*-values, *g*-values to *inf*; Set *OPEN/CLOSED* to *empty*; *h* given

Insert s_{start} into *OPEN*

While s_{goal} *not expanded* **do**

Remove *s* with the smallest $f(s) = g(s)+h(s)$ from *OPEN*

Insert *s* to *CLOSED*

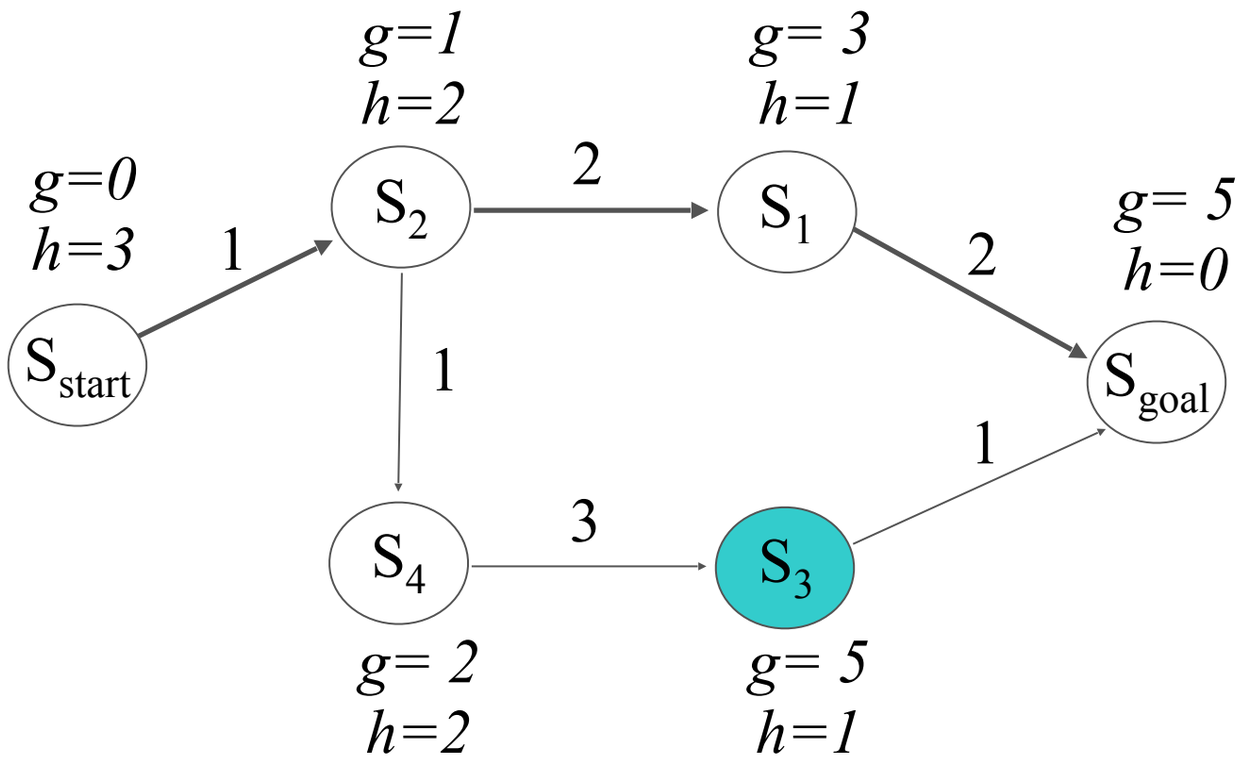
For every successor s', *not* in *CLOSED* **do**

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

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

Insert s' to *OPEN*

$$PATH = \{s_{start}, s_2, s_1, s_{goal}\}$$

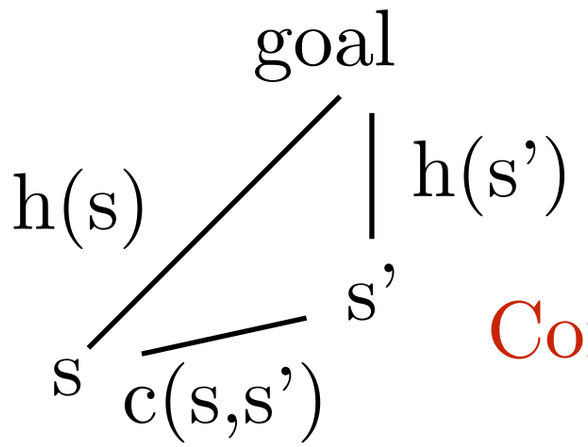


Properties of heuristics

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

Admissible: h(s) <= h*(s) h(goal) = 0

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

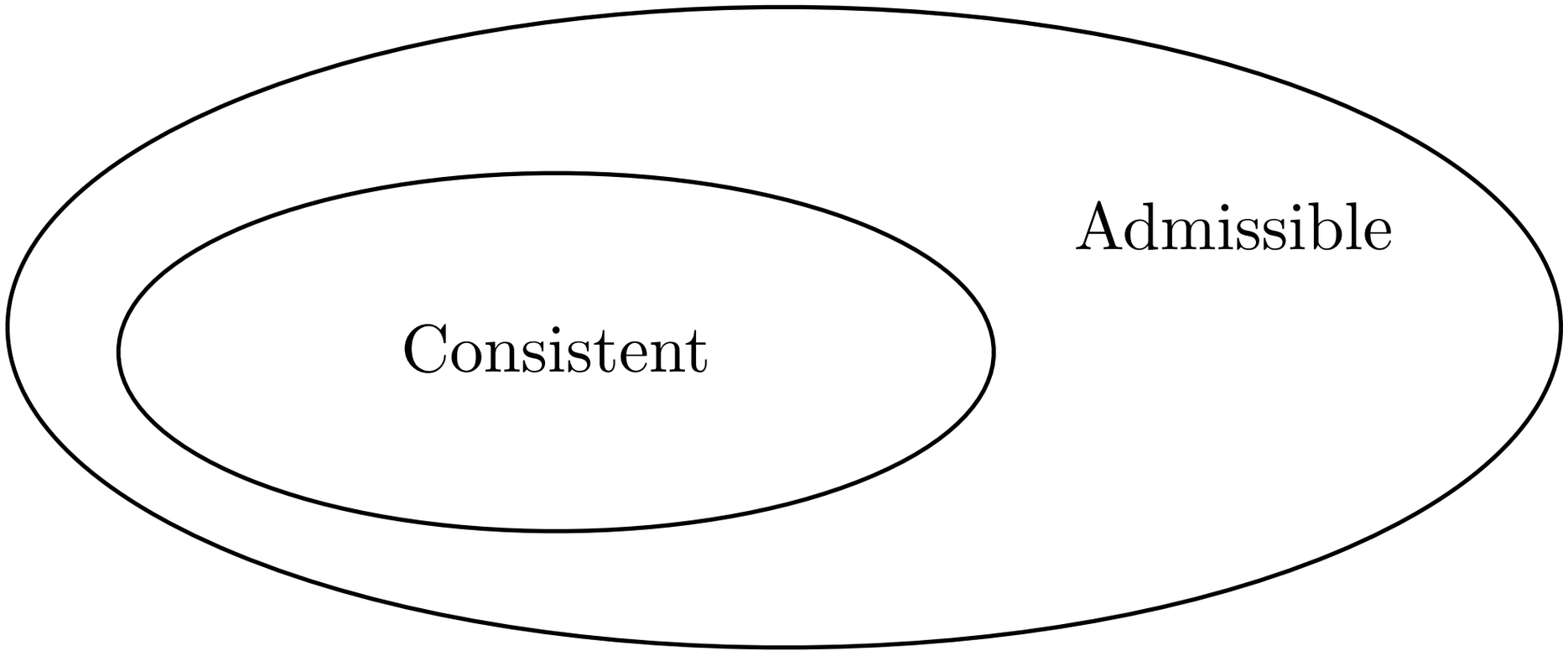


(triangle inequality)

Consistency: h(s) <= c(s,s') + h(s') 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!

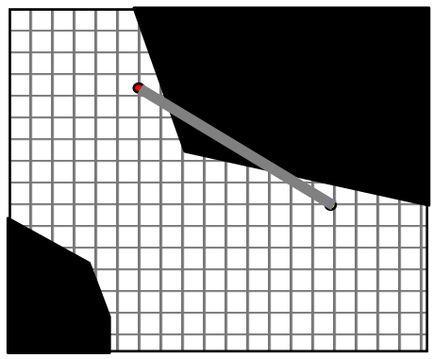
Takeaway:

Heuristics are great because they focus search on relevant states

AND

still give us optimal solution

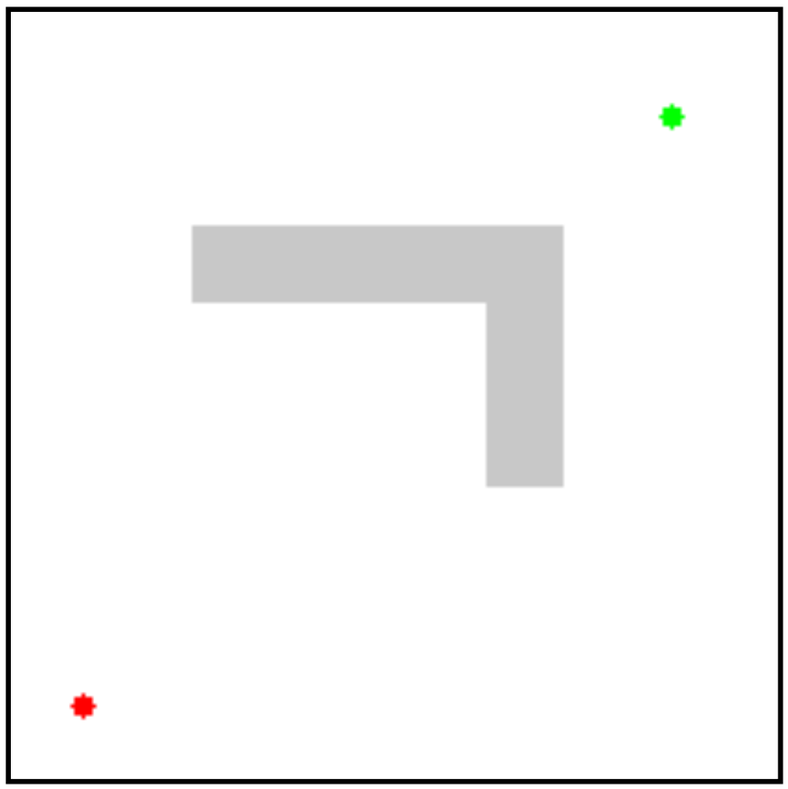
Design of Informative Heuristics



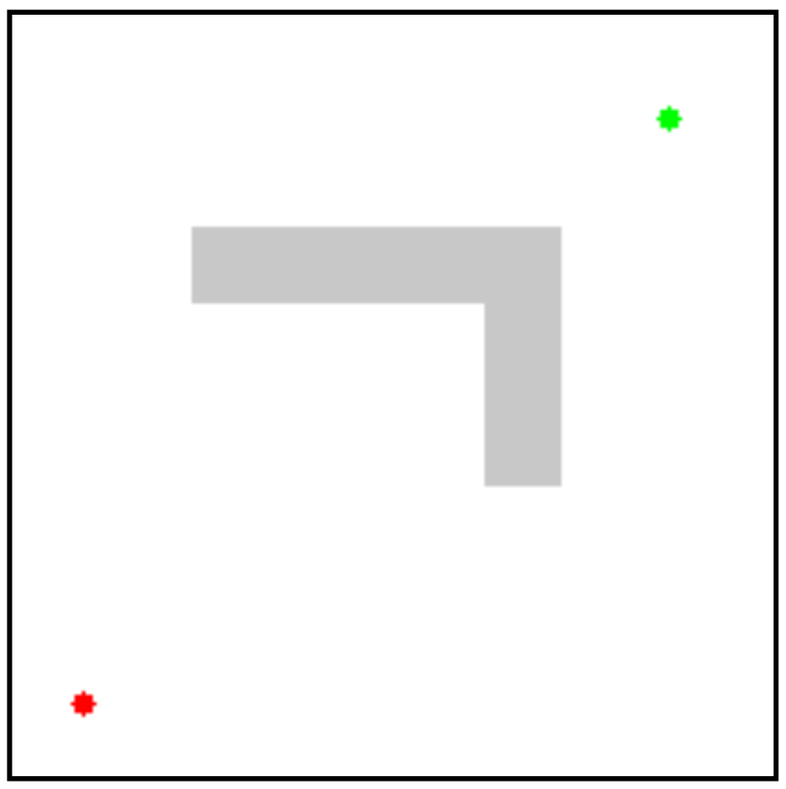
- For grid-based navigation:
  - Euclidean distance
  - Manhattan distance: $h(x,y) = abs(x-x_{goal}) + abs(y-y_{goal})$
  - Diagonal distance: $h(x,y) = max(abs(x-x_{goal}), abs(y-y_{goal}))$

Courtesy Max Likhachev

Is admissibility always what we want?



Admissible



Inadmissible

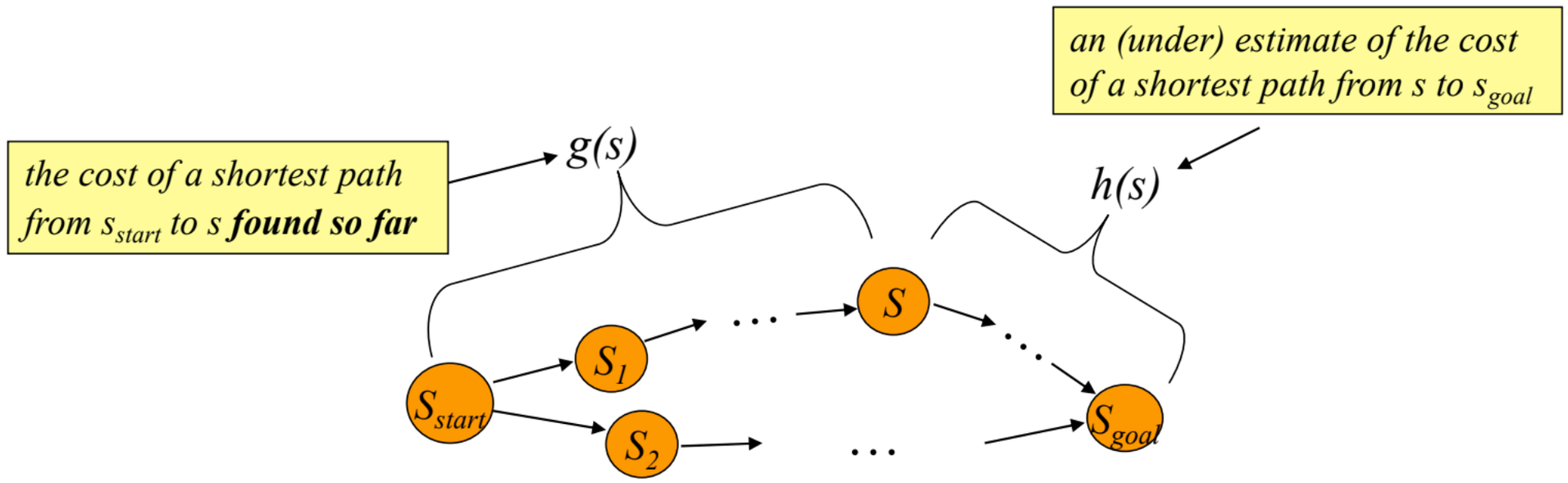
Can inadmissible heuristics help us with this tradeoff?

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

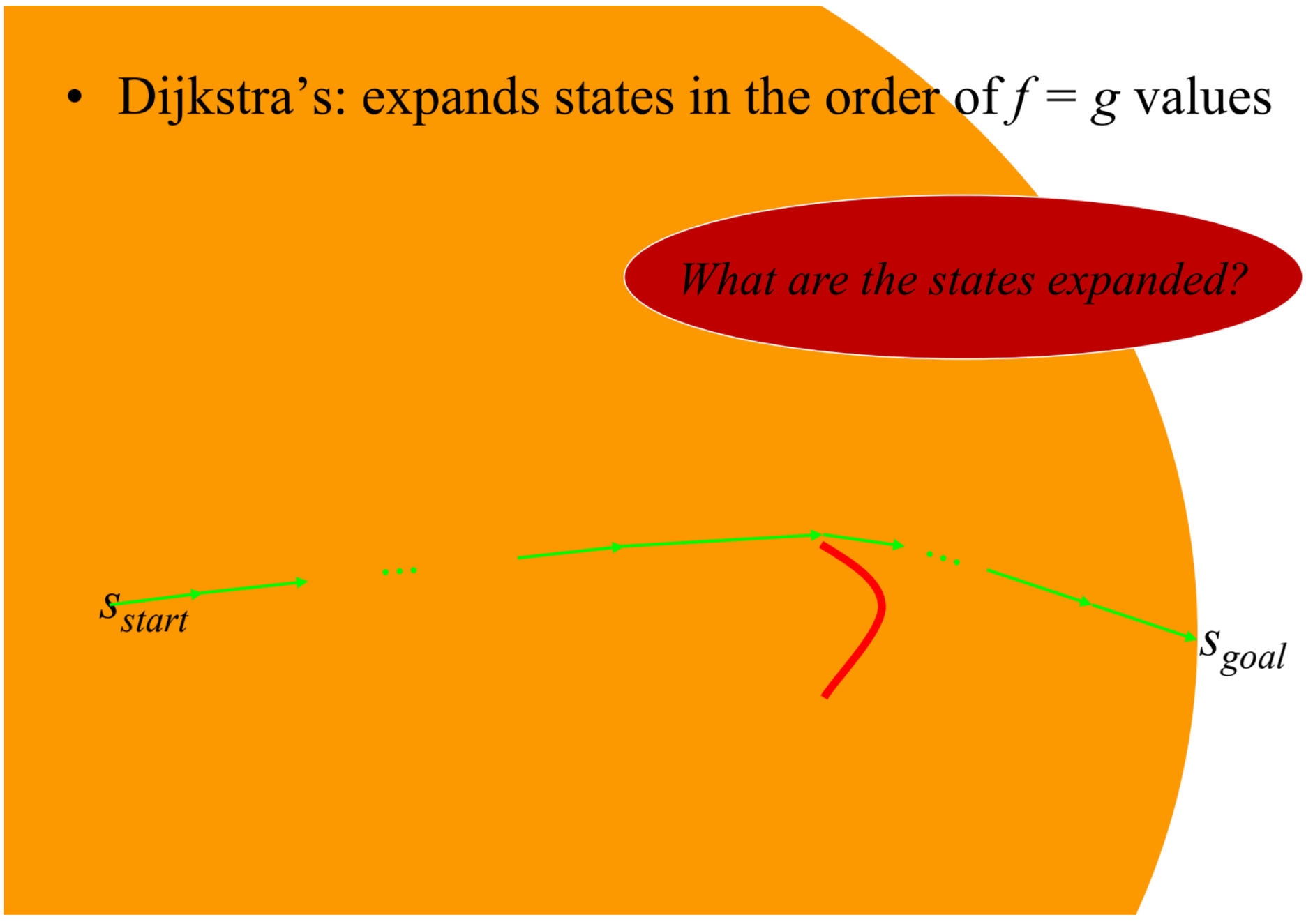
- **Weighted A*:** expands states in the order of $f = g+\varepsilon h$ values, $\varepsilon > 1$ = bias towards states that are closer to goal



Courtesy Max Likhachev

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

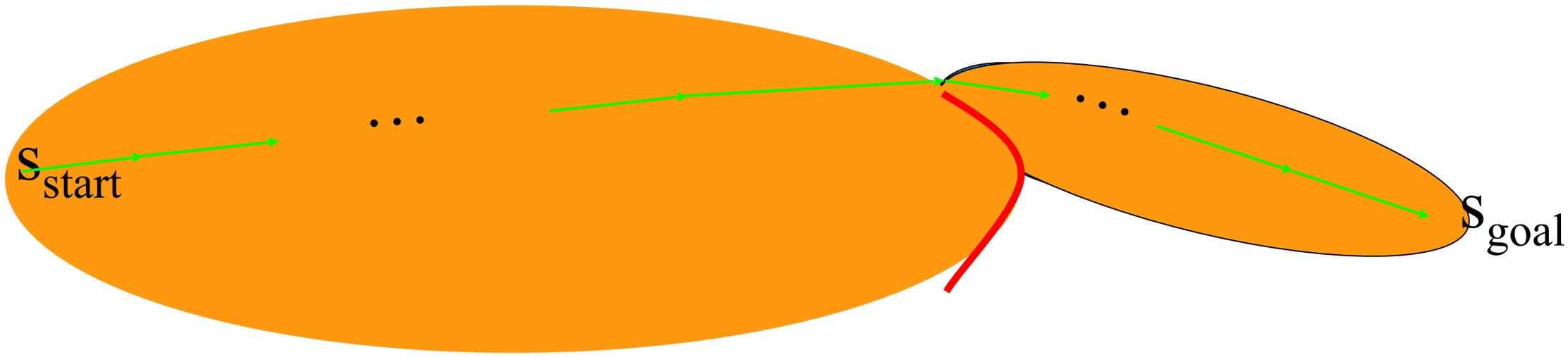
What are the states expanded?



Courtesy Max Likhachev

Effect of the Heuristic Function

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

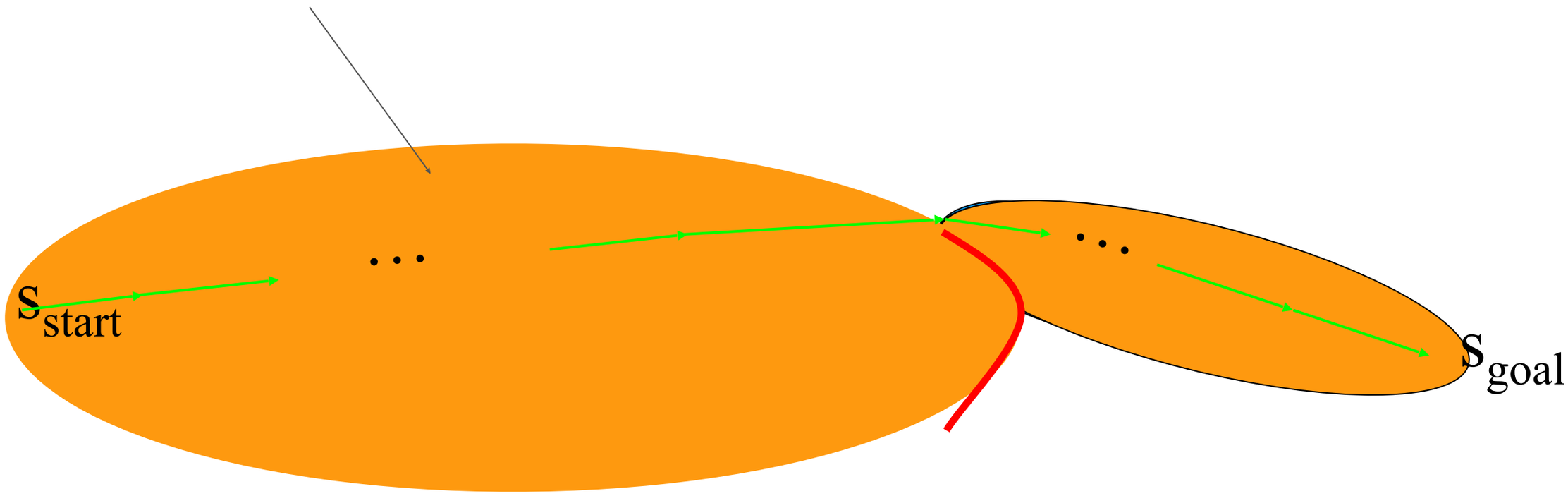


Courtesy Max Likhachev

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))

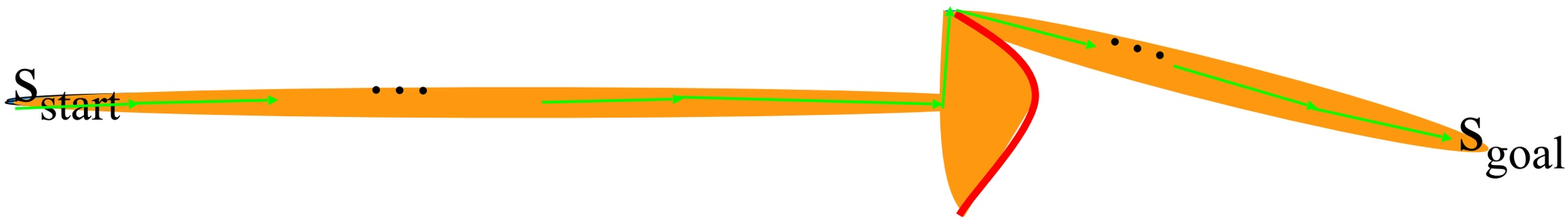


Courtesy Max Likhachev

Effect of the Heuristic Function

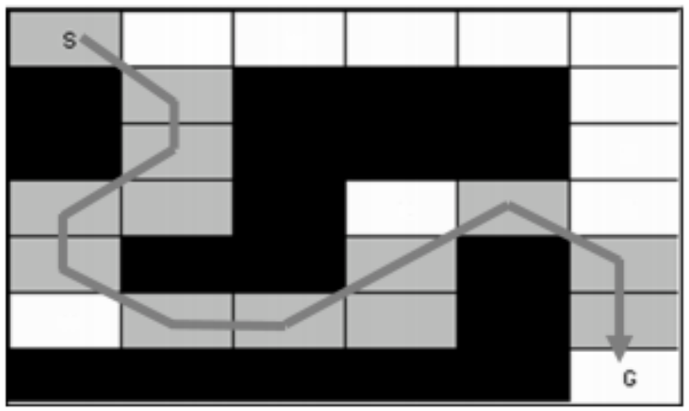
Weighted A* Search: expands states in the order of $f = g+\varepsilon h$ values, $\varepsilon > 1$ = bias towards states that are closer to goal

solution is always ε-suboptimal:
cost(solution) ≤ ε·cost(optimal solution)

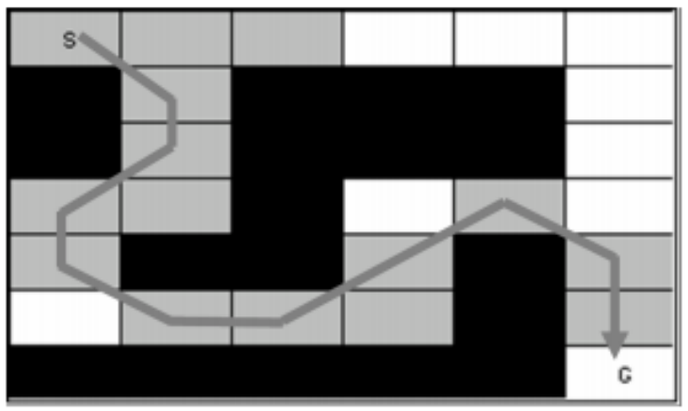


Courtesy Max Likhachev

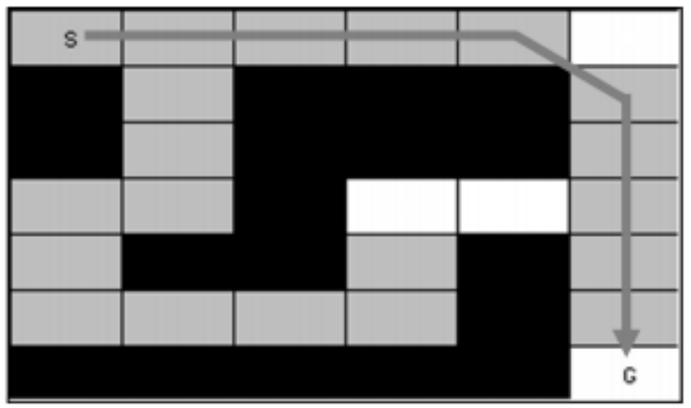
Effect of the Heuristic Function



$\epsilon = 2.5$



$\epsilon = 1.5$



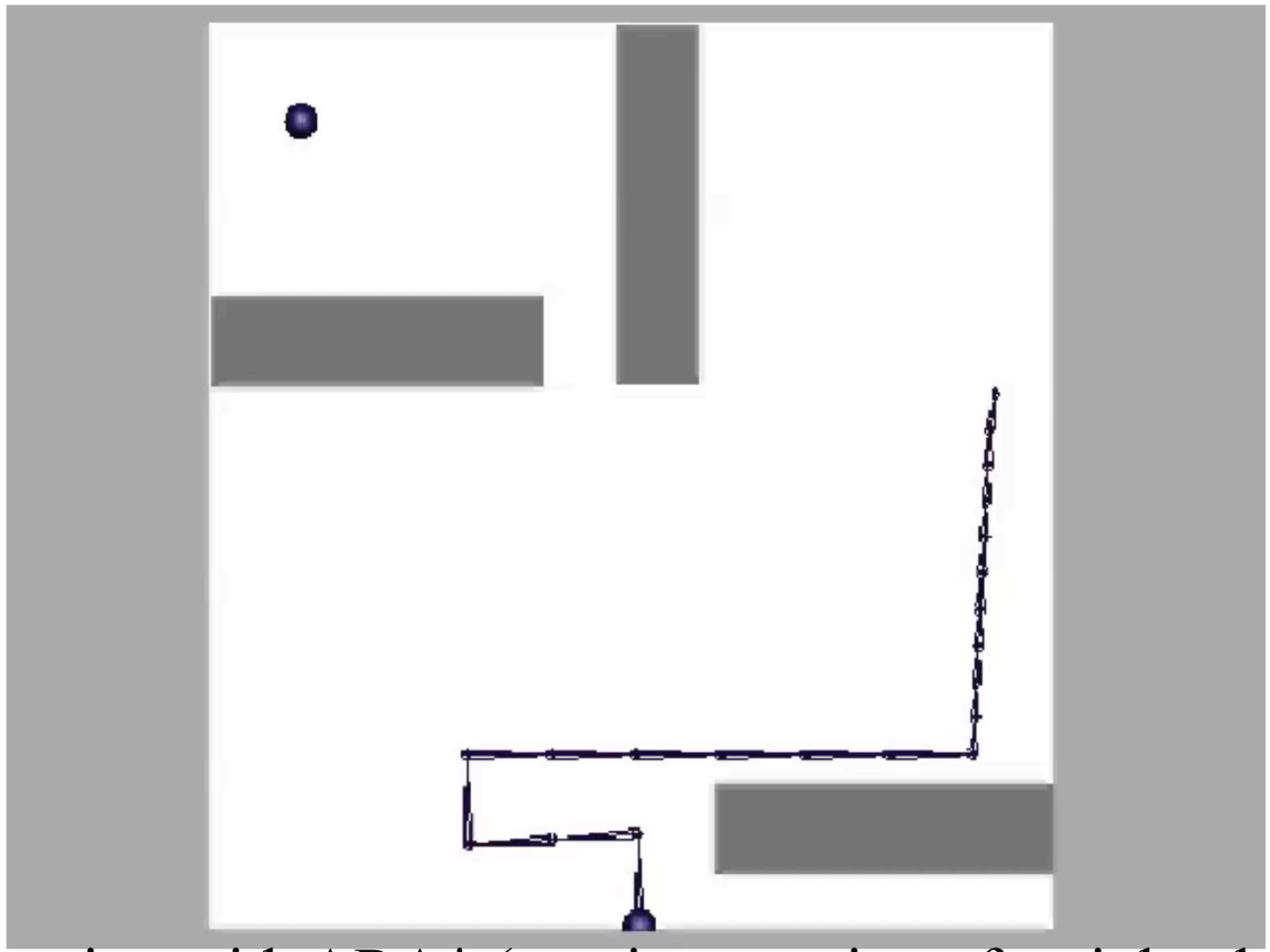
$\epsilon = 1.0$ (optimal search)

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



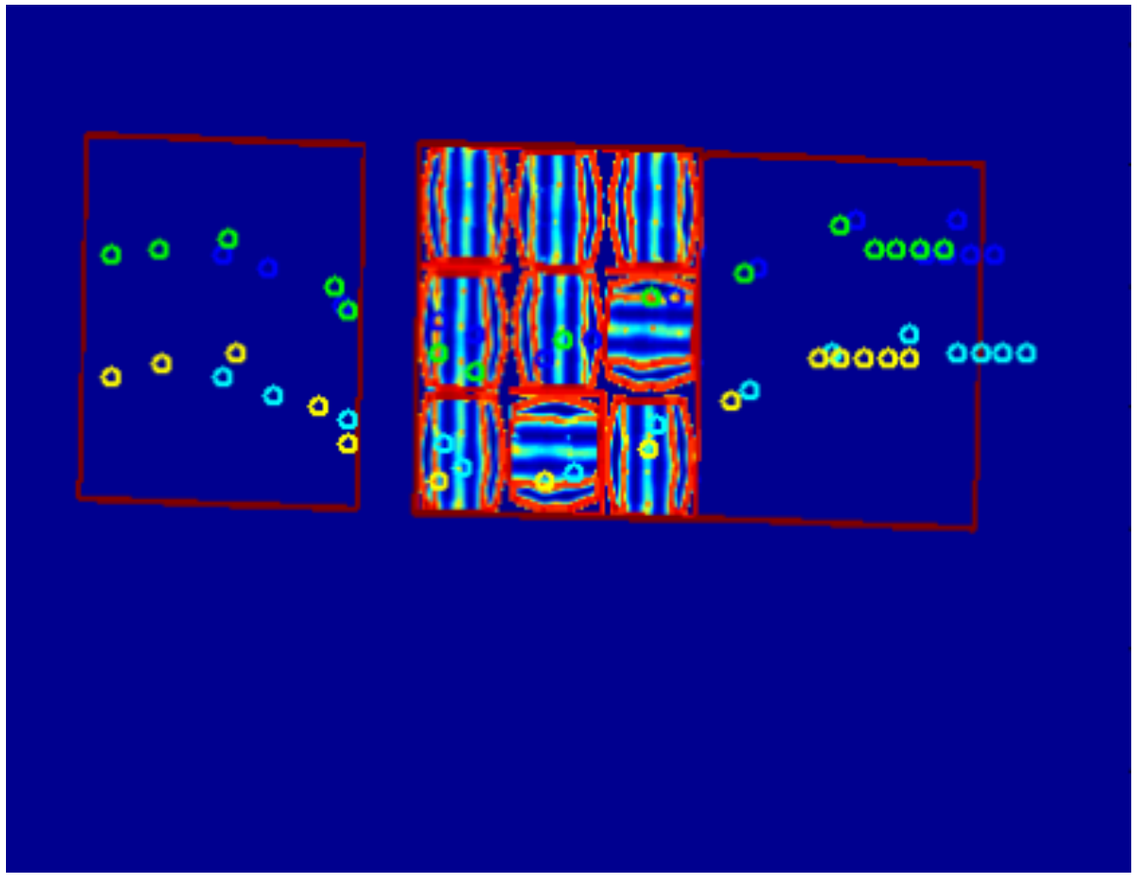
planning with ARA* (anytime version of weighted A*)

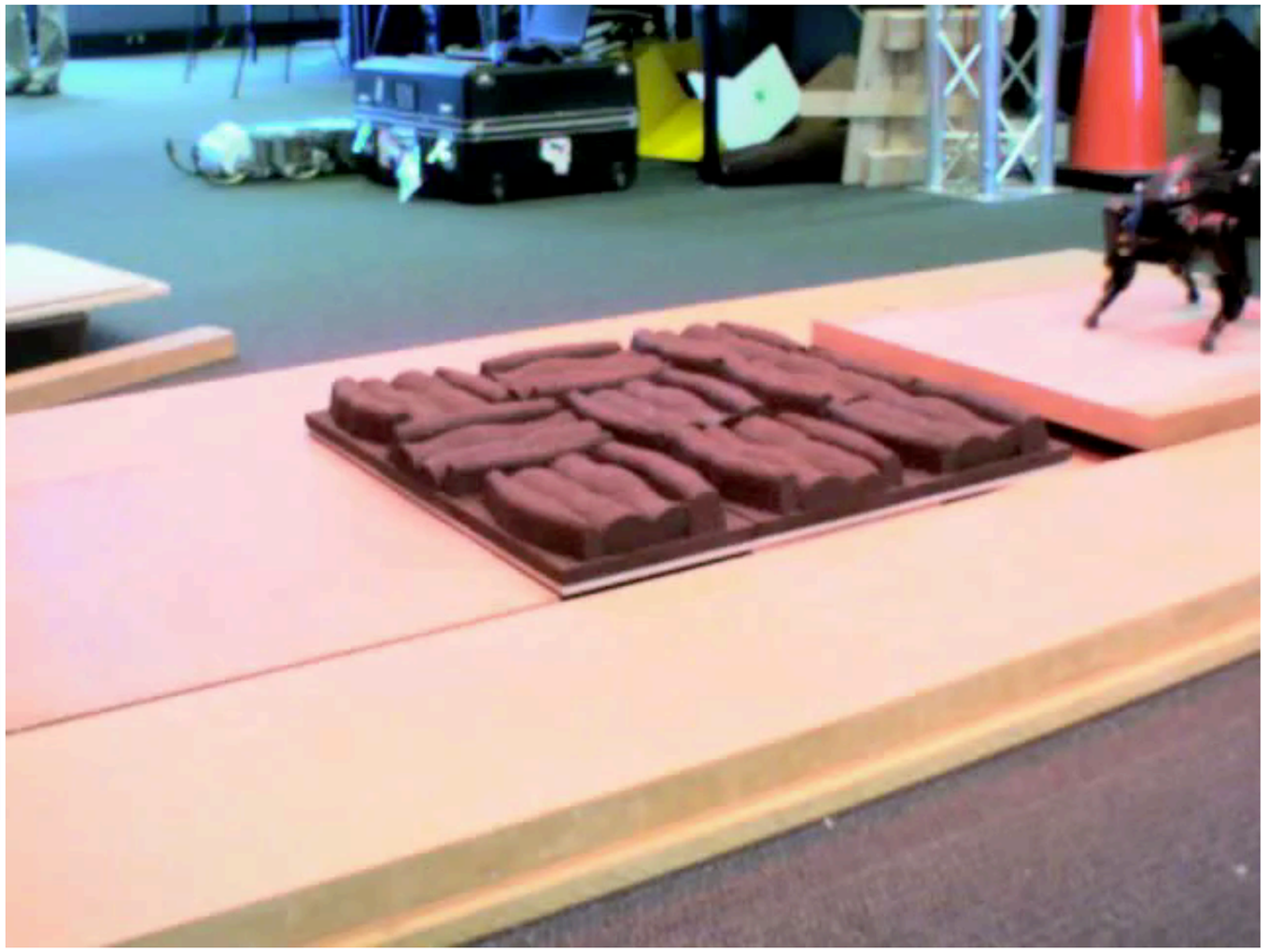
Courtesy Max Likhachev

Effect of the Heuristic Function

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