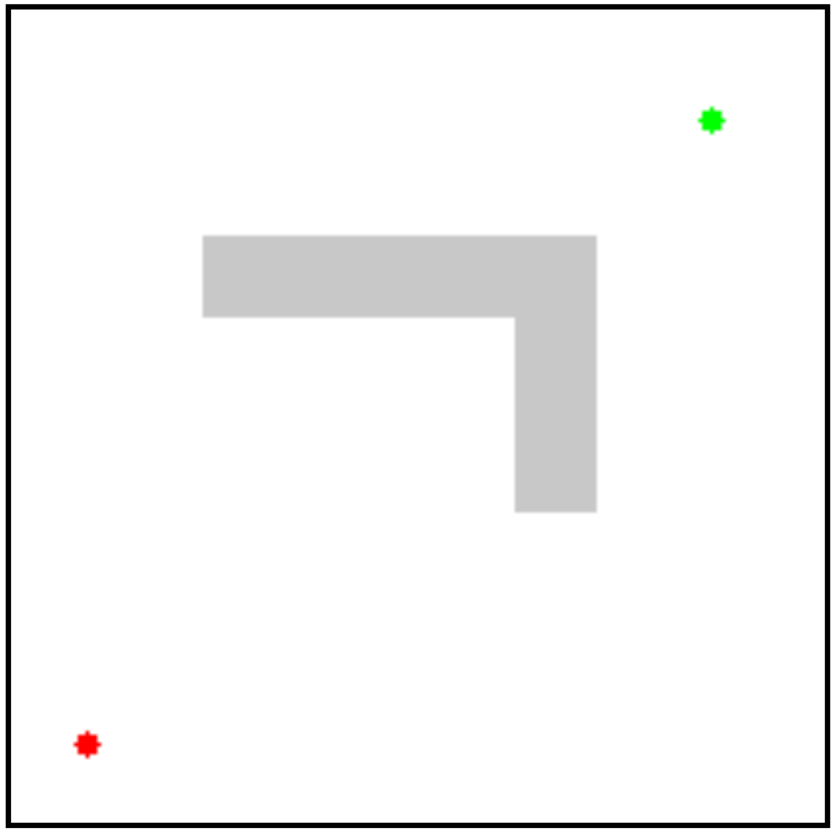
Lazy Search

Sanjiban Choudhury

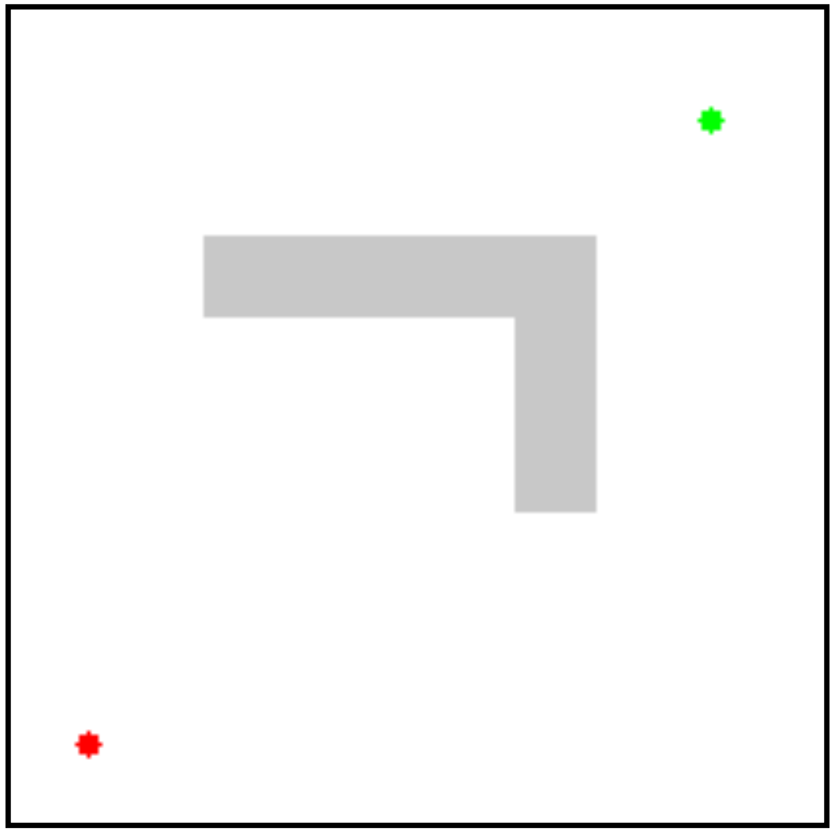
TAs: Matthew Rockett, Gilwoo Lee, Matt Schmittle

High-order bit

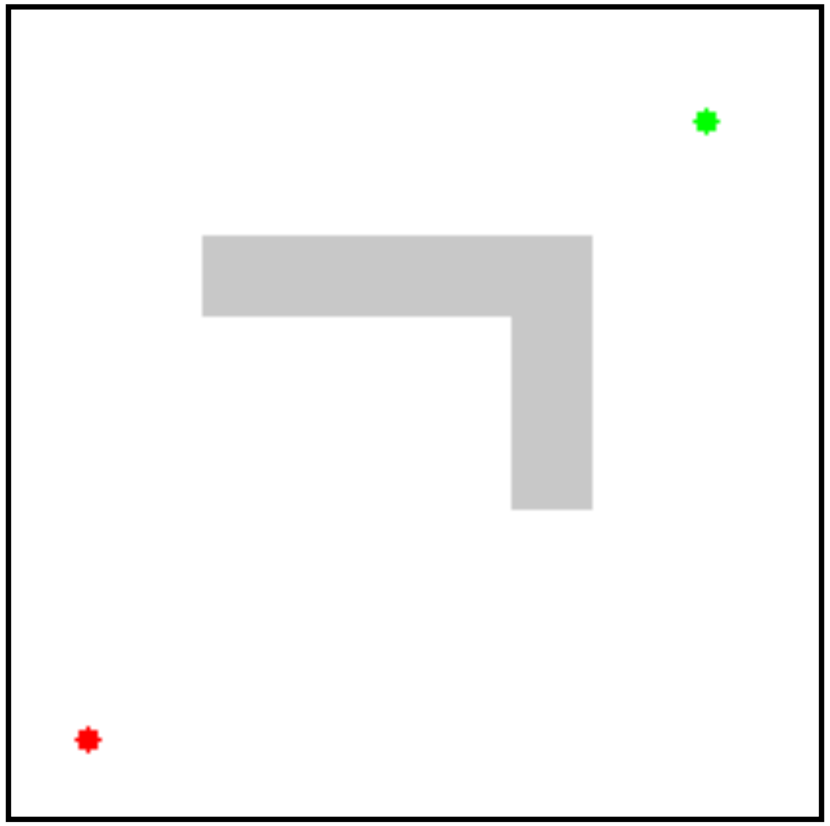
Expansion of a search wavefront from start to goal



Djikstra



A*

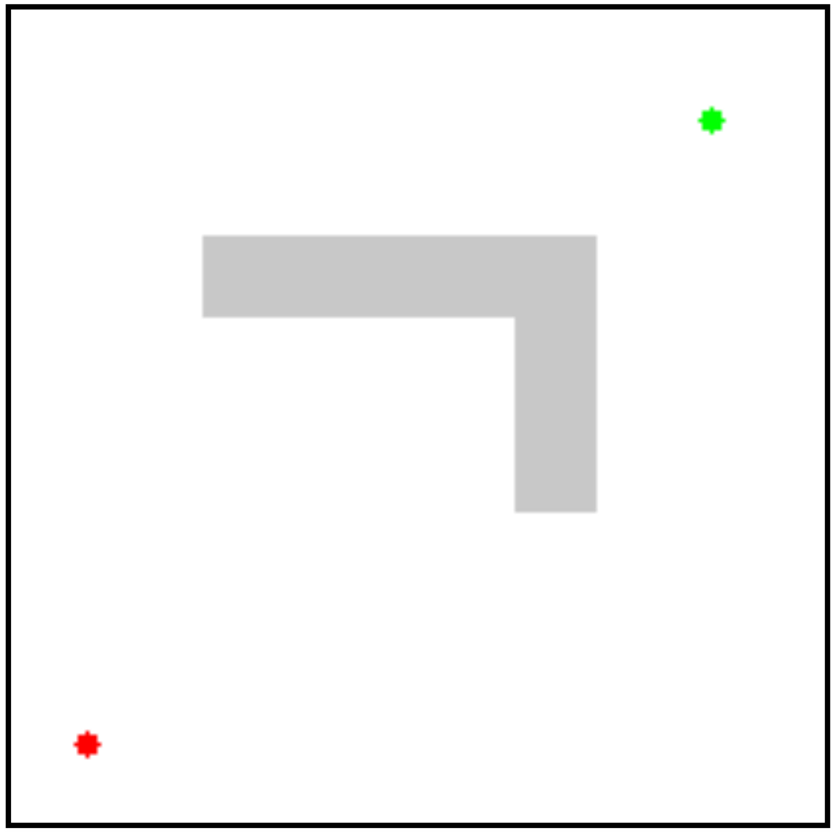


Weighted A*

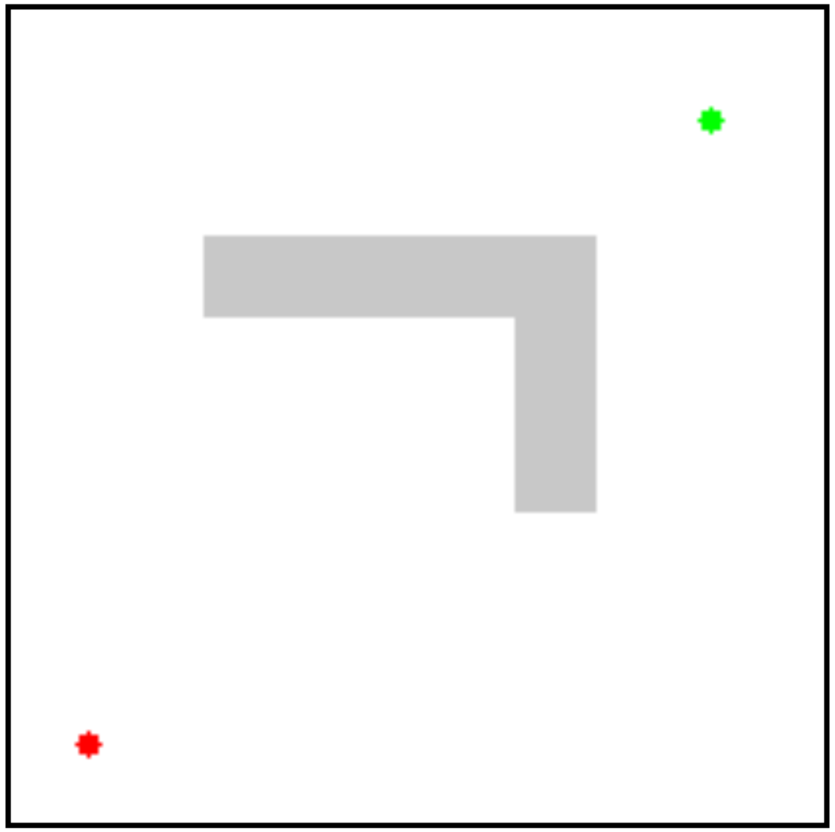
Courtesy wikipedia

High-order bit

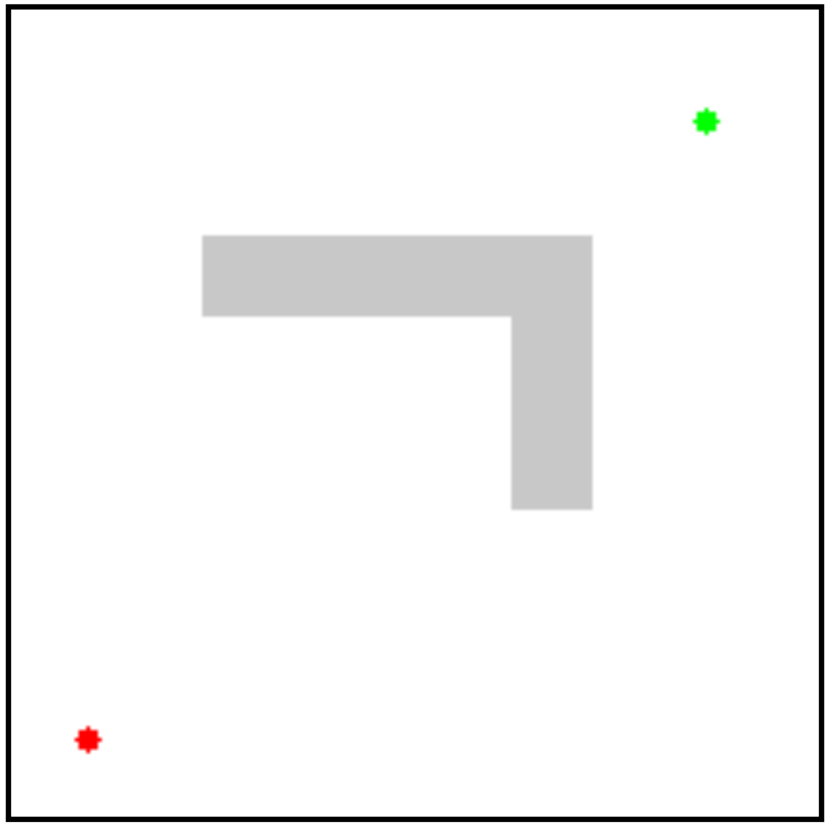
Expansion of a search wavefront from start to goal



Djikstra



A*



Weighted A*

Courtesy wikipedia

What do we want?

1. Search to systematically reason over the space of paths

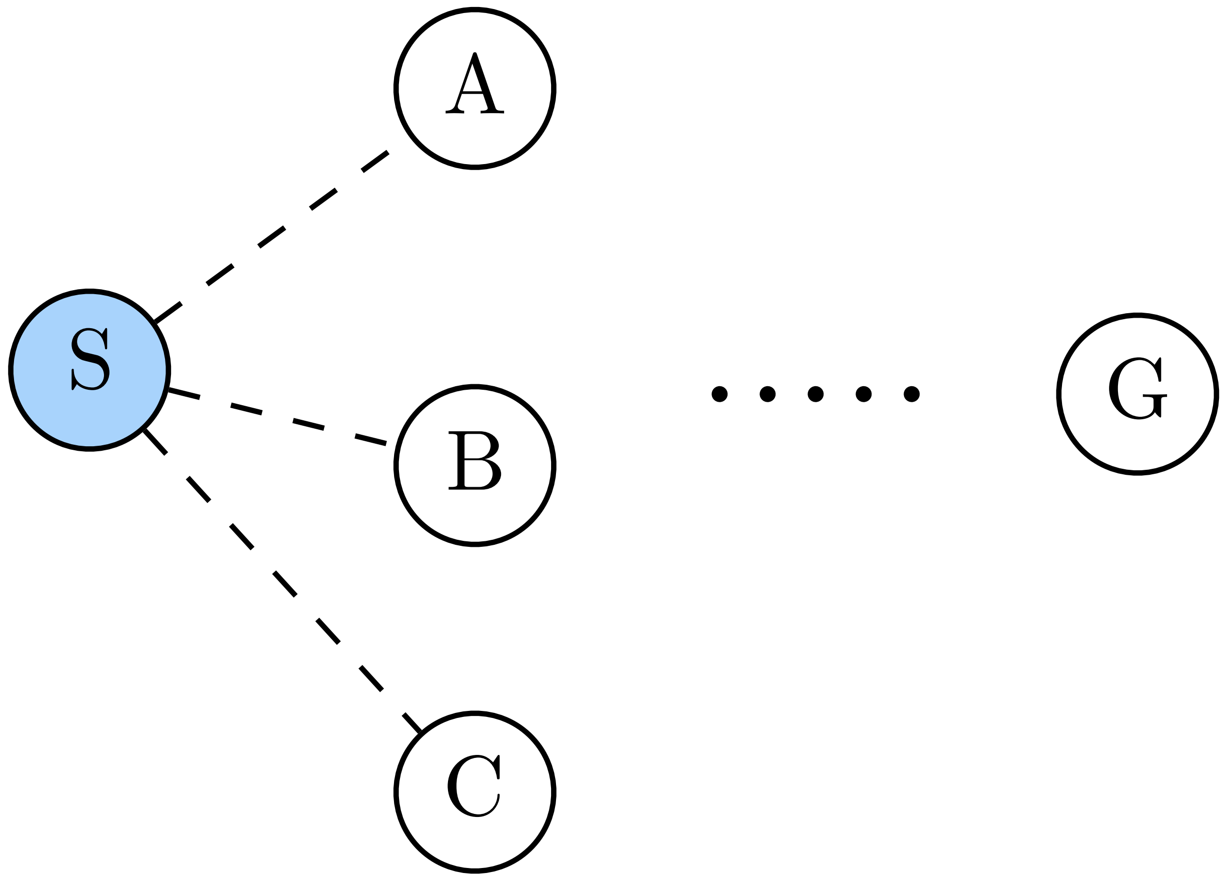
2. Find a (near)-optimal path quickly

(minimize planning effort)

Best First Search

(Explore the graph by expanding/processing promising nodes)

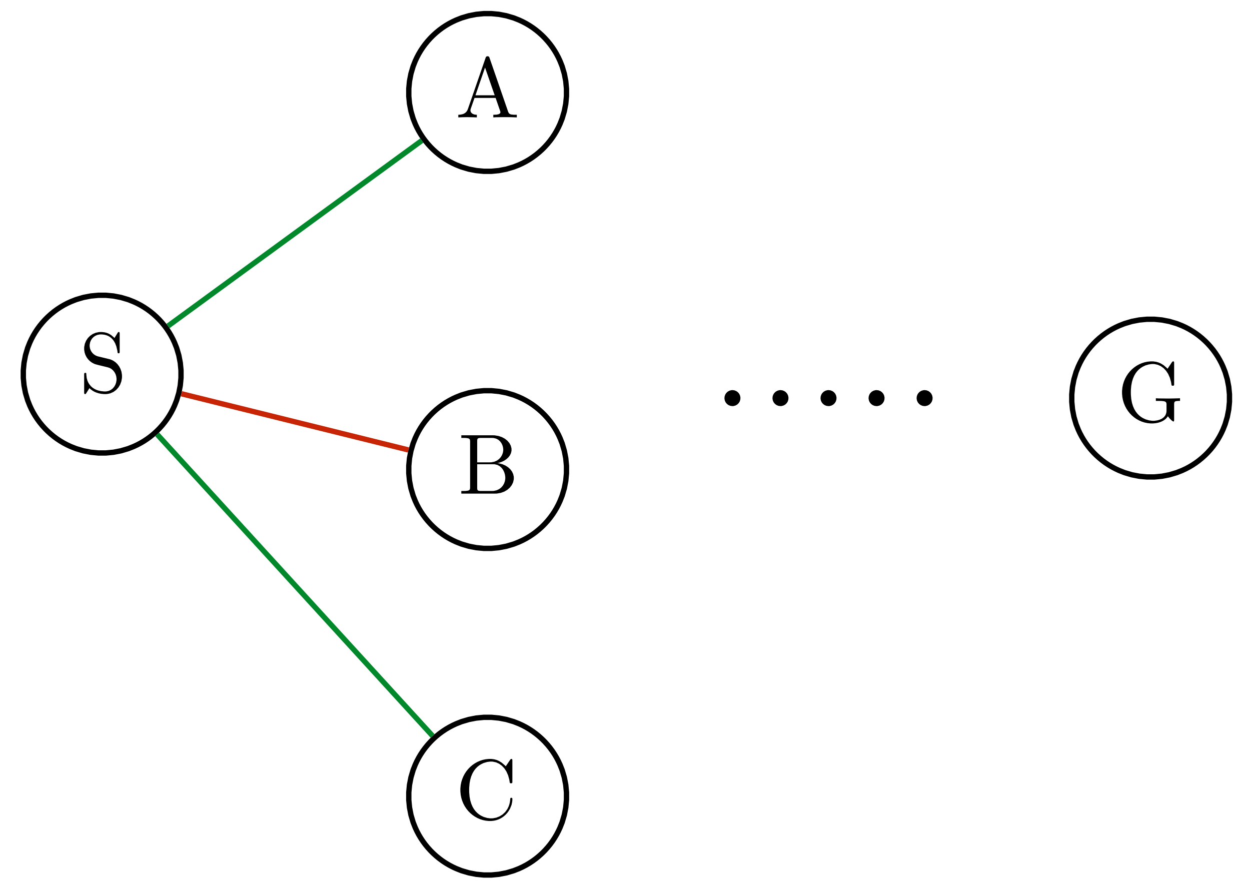
Element (Node)	**Priority Value (f-value)**
Node S	f(S)



Best First Search

(Explore the graph by expanding/processing promising nodes)

Element (Node)	**Priority Value (f-value)**
~~Node S~~	~~f(S)~~
Node A	f(A)
Node C	f(C)



Different f(s) leads to different algorithms

Algorithm	**f(s)**	**Optimality**	**Efficiency**
Djikstra	$f(s) = g(s)$	Yes	Poor
A*	$f(s) = g(s) + h(s)$	Yes* (if h(s) is admissible)	Good* (better if h(s) is consistent)

What is a heuristic h(s)?

So far, we have been thinking of heuristics as an estimate of cost-to-go to goal

What is a heuristic h(s)?

So far, we have been thinking of heuristics as an estimate of cost-to-go to goal

Goal is to come up with cheap-to-compute estimates

What is a heuristic h(s)?

So far, we have been thinking of heuristics as an estimate of cost-to-go to goal

Goal is to come up with cheap-to-compute estimates

- External knowledge about metrics
    - Euclidean distance admissible because triangle inequality

What is a heuristic h(s)?

So far, we have been thinking of heuristics as an
estimate of cost-to-go to goal

Goal is to come up with cheap-to-compute estimates

- External knowledge about metrics
  - Euclidean distance admissible because triangle inequality

- Solving a relaxation of the problem (which is easier)
  - Ignore obstacles, ignore dynamics

What is a heuristic h(s)?

So far, we have been thinking of heuristics as an estimate of cost-to-go to goal

Goal is to come up with cheap-to-compute estimates

- External knowledge about metrics
  - Euclidean distance admissible because triangle inequality

- Solving a relaxation of the problem (which is easier)
  - Ignore obstacles, ignore dynamics

- Statistical knowledge of the problem
  - Keep arms tucked in, don't drive headfirst into a wall

What is a heuristic h(s)?

So far, we have been thinking of heuristics as an estimate of cost-to-go to goal

What is a heuristic h(s)?

So far, we have been thinking of heuristics as an estimate of cost-to-go to goal

But another way to think of heuristics is as a ranking function

What is a heuristic h(s)?

So far, we have been thinking of heuristics as an
estimate of cost-to-go to goal

But another way to think of heuristics is as a ranking function

Even if estimates are off, as long as a heuristic is ranking
good states better than bad states,
it's useful

The case for inadmissible search

The case for inadmissible search

Let's say you have a h(s) that is very very admissible

The case for inadmissible search

Let's say you have a h(s) that is very very admissible

But it is actually good at ranking good vs bad states

The case for inadmissible search

Let's say you have a h(s) that is very very admissible

But it is actually good at ranking good vs bad states

$f(s) = g(s) + h(s)$ might not be best at ranking states. Why?

The case for inadmissible search

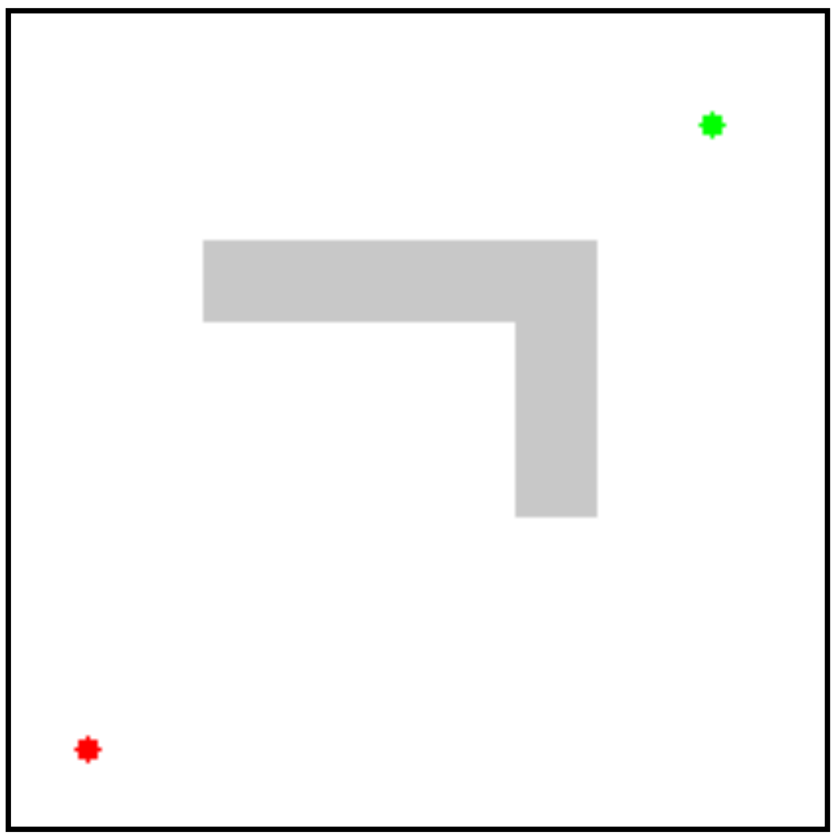
Let's say you have a h(s) that is very very admissible

But it is actually good at ranking good vs bad states

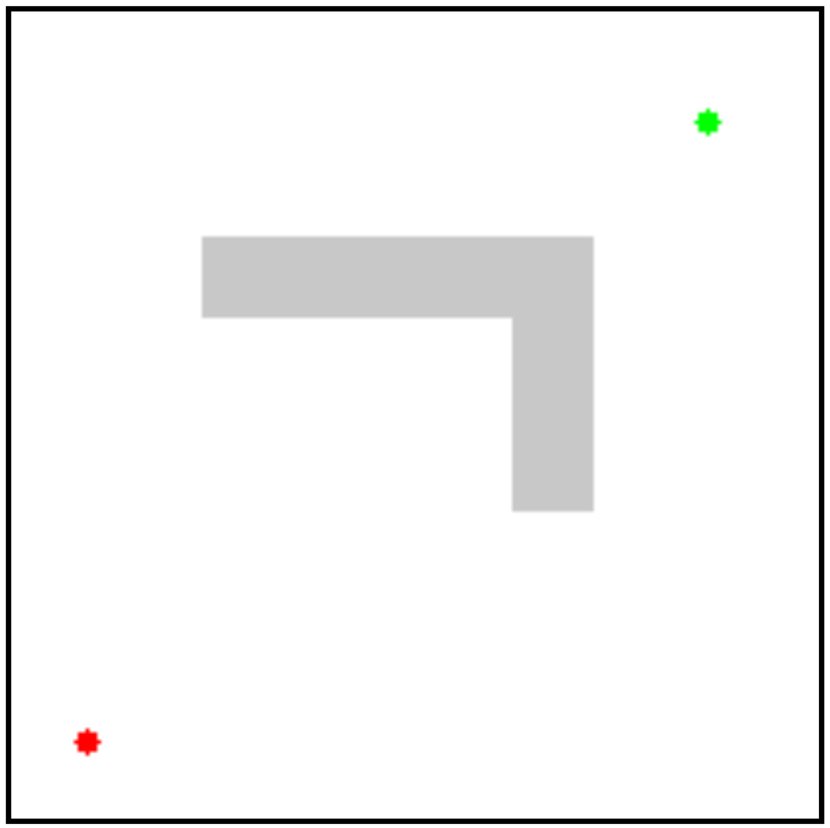
$f(s) = g(s) + h(s)$ might not be best at ranking states. Why?

$f(s) = g(s) + 10h(s)$ might be much better. Why?

Side-effects of inflating heuristics

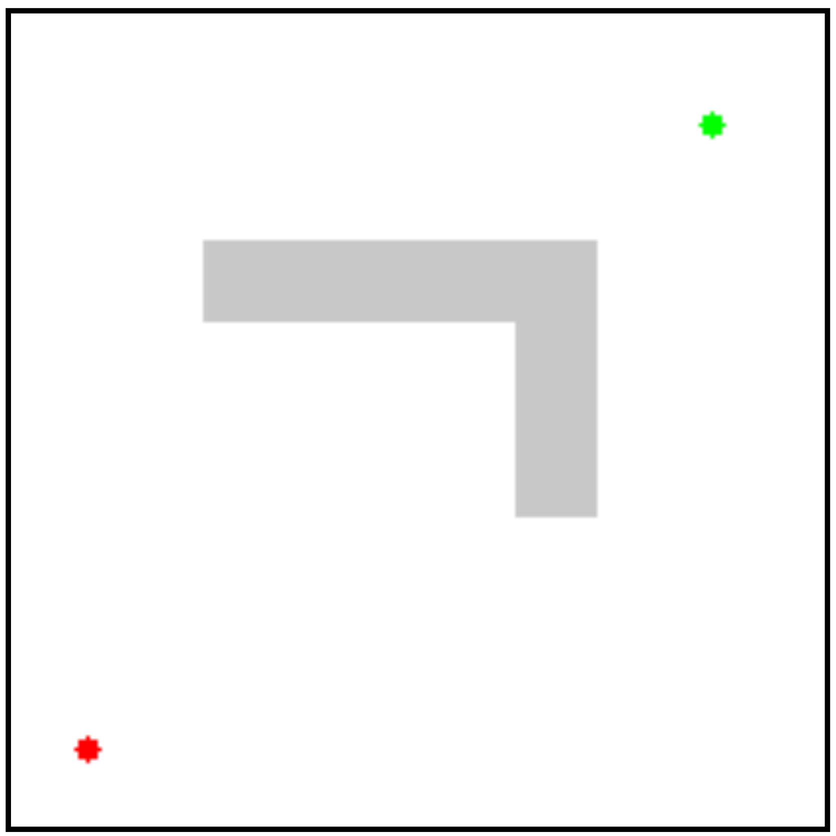


$$f(s) = g(s) + h(s)$$

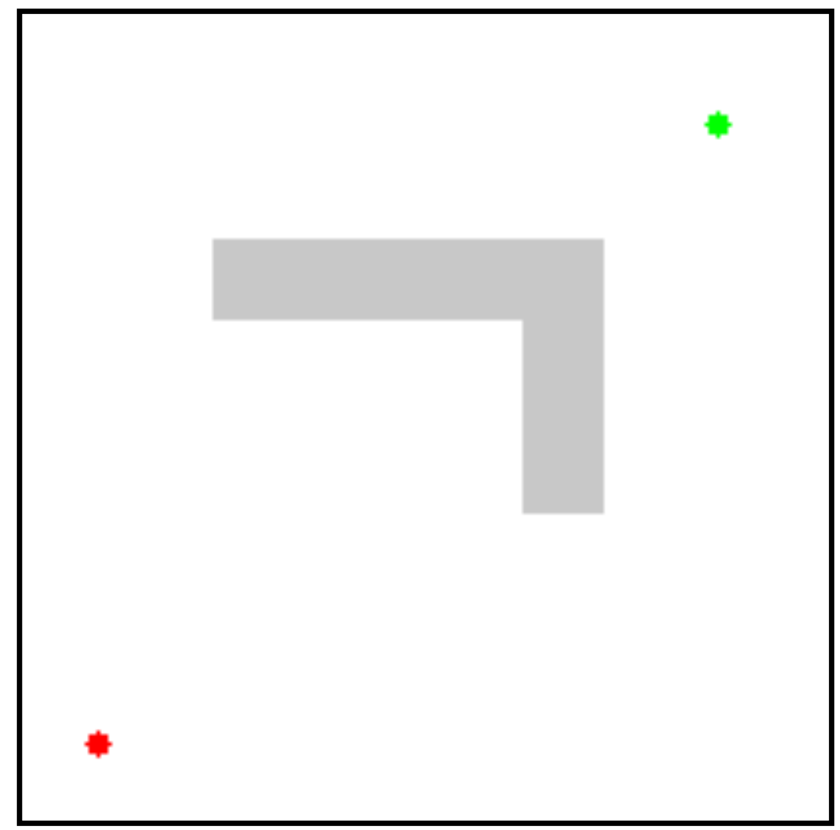


$$f(s) = g(s) + 5h(s)$$

Side-effects of inflating heuristics



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



$f(s) = g(s) + 5h(s)$

Can we bound the solution quality?

$$\epsilon \geq 1$$

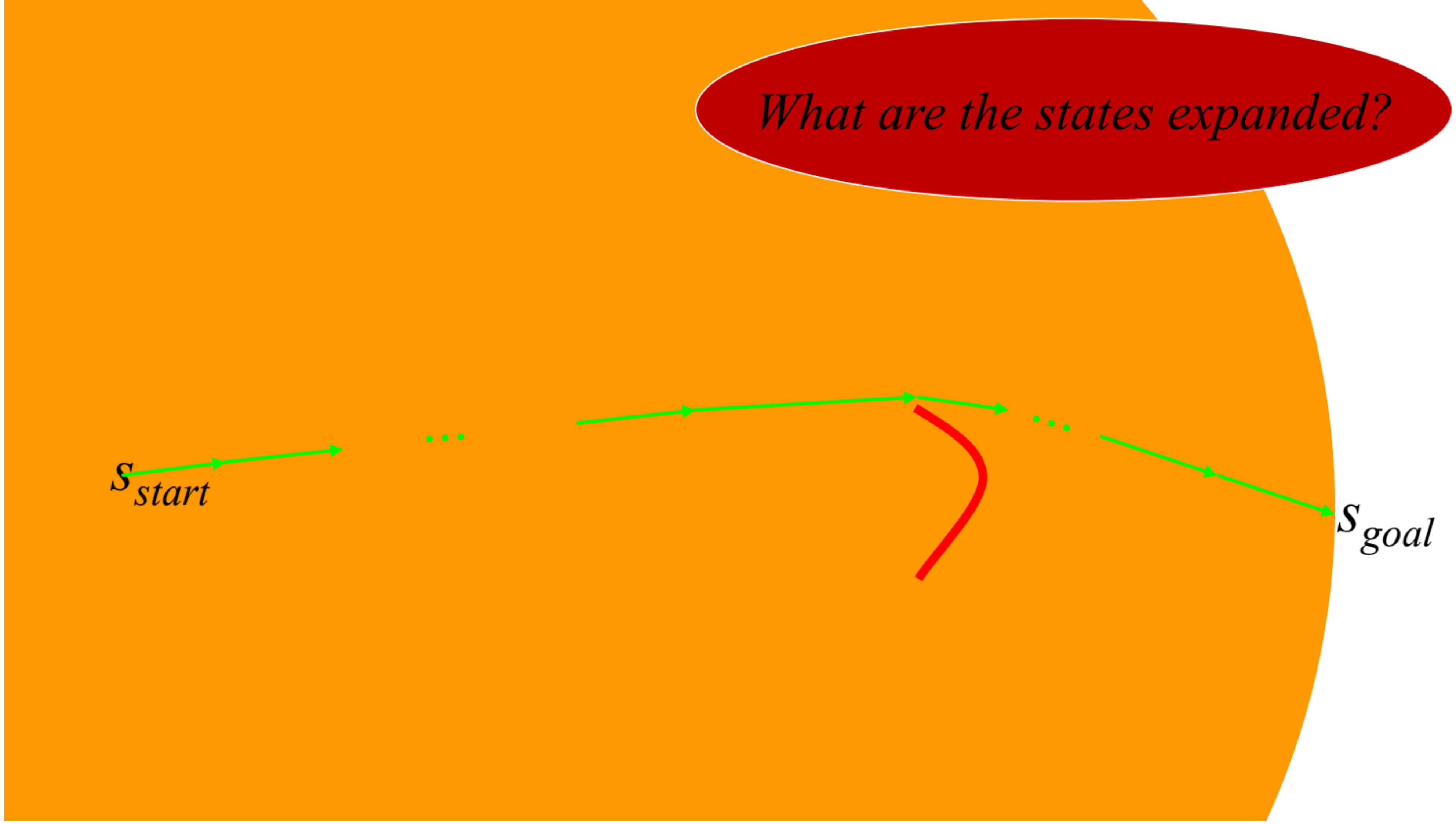
Can we bound the solution quality?

$$g(s) + \epsilon h(s) \qquad\qquad \epsilon \geq 1$$

results in

$$\text{cost}(\text{solution}) \leq \epsilon \, \text{cost}(\text{solution})$$

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



Courtesy Max Likhachev

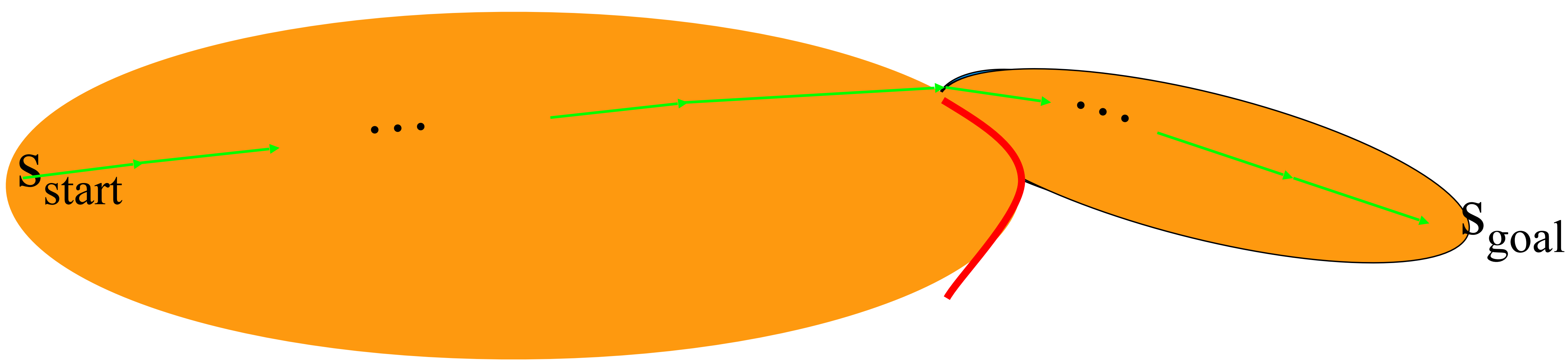
Effect of the Heuristic Function

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

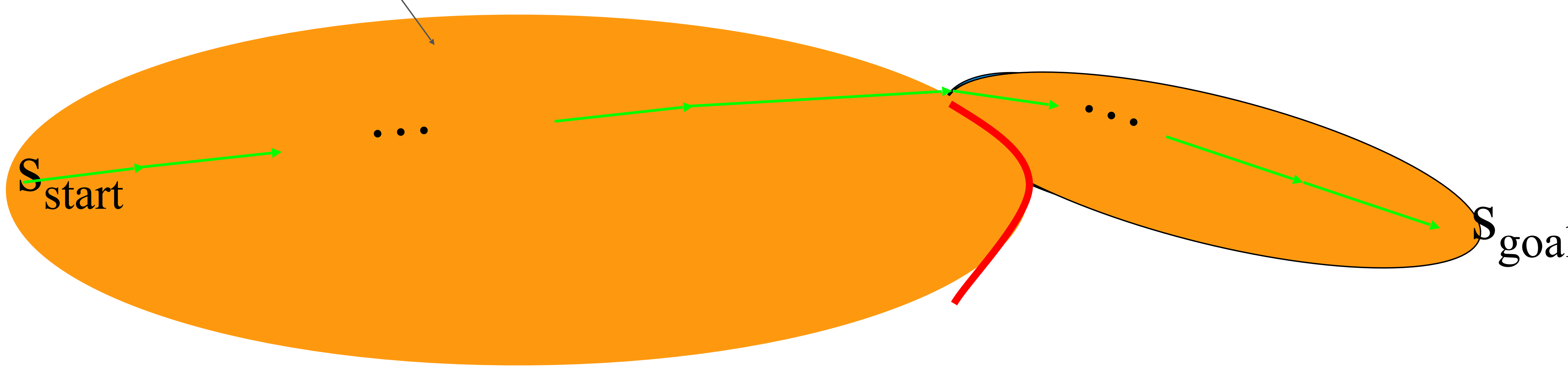


Courtesy Max Likhachev

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

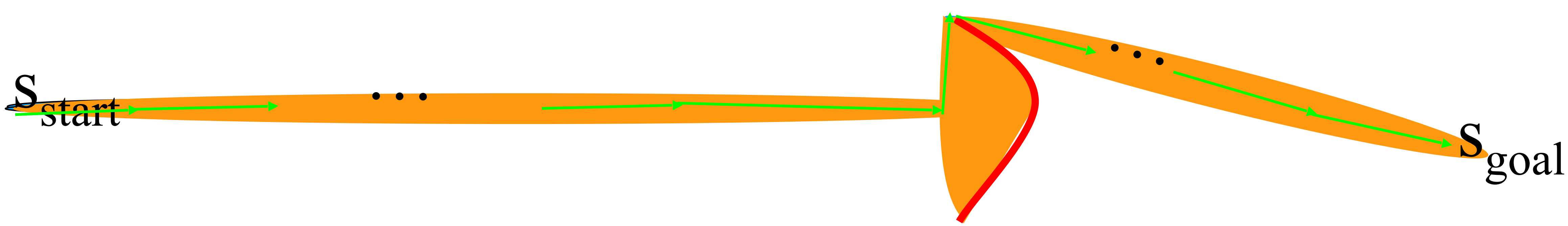
Effect of the Heuristic Function

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

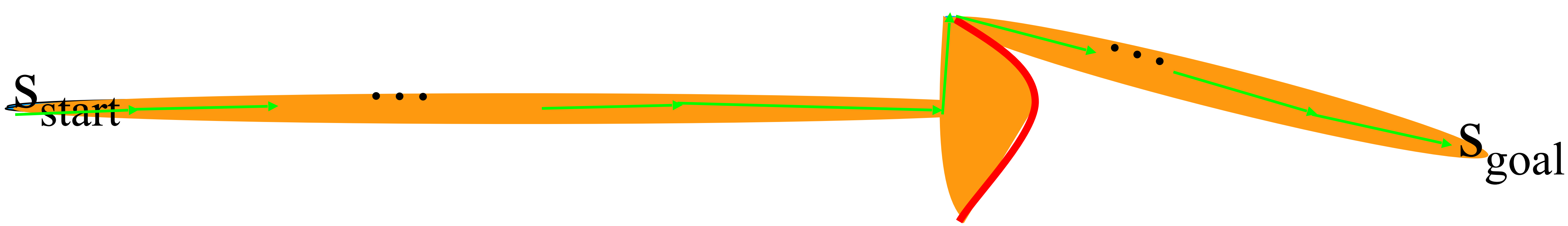


Courtesy Max Likhachev

Effect of the Heuristic Function

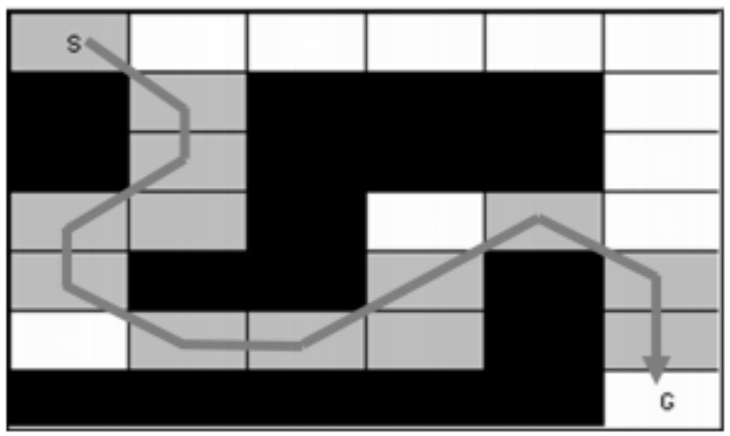
Weighted A* Search: expands states in the order of $f = g + \varepsilon h$ values, $\varepsilon > 1$ = bias towards states that are closer to goal

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

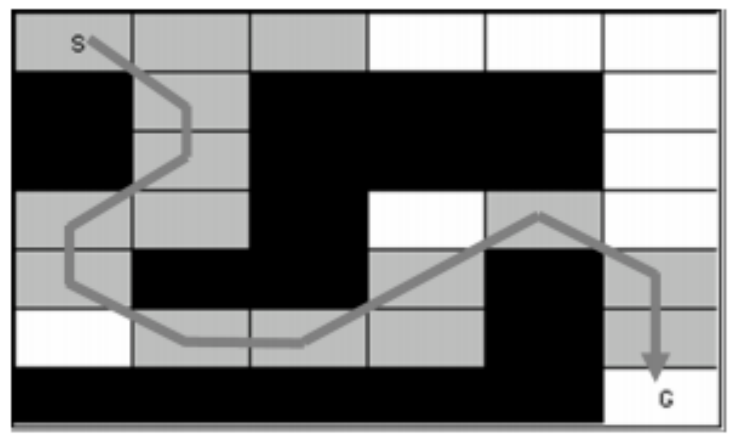


Courtesy Max Likhachev

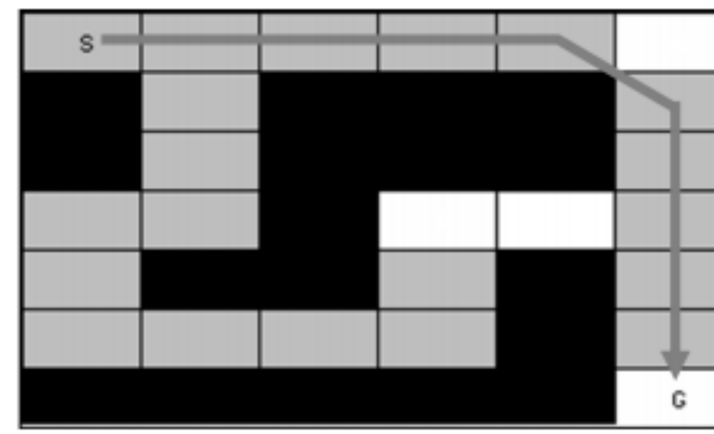
Effect of the Heuristic Function



$\epsilon = 2.5$



$\epsilon = 1.5$



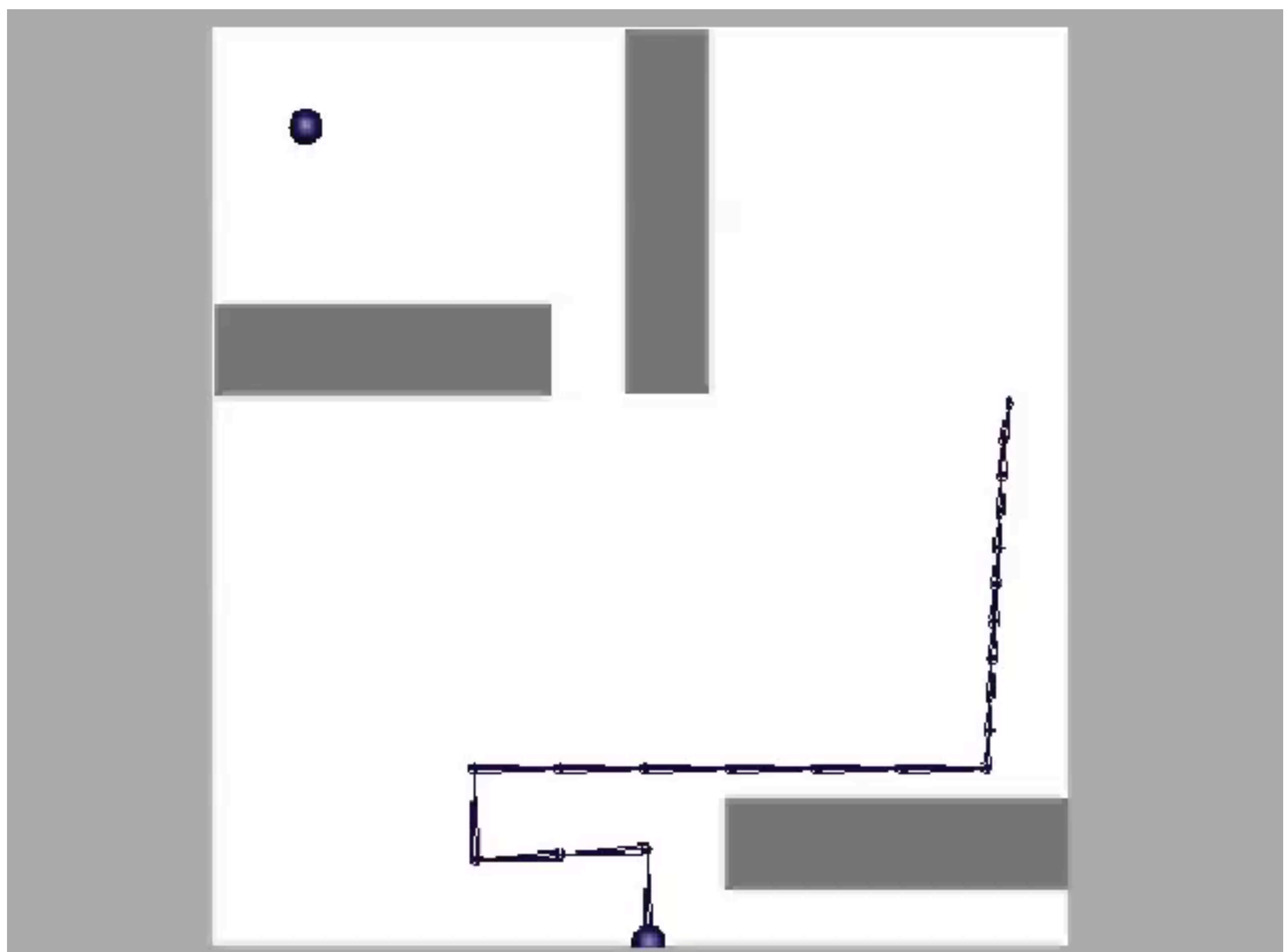
$\epsilon = 1.0$ (optimal search)

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



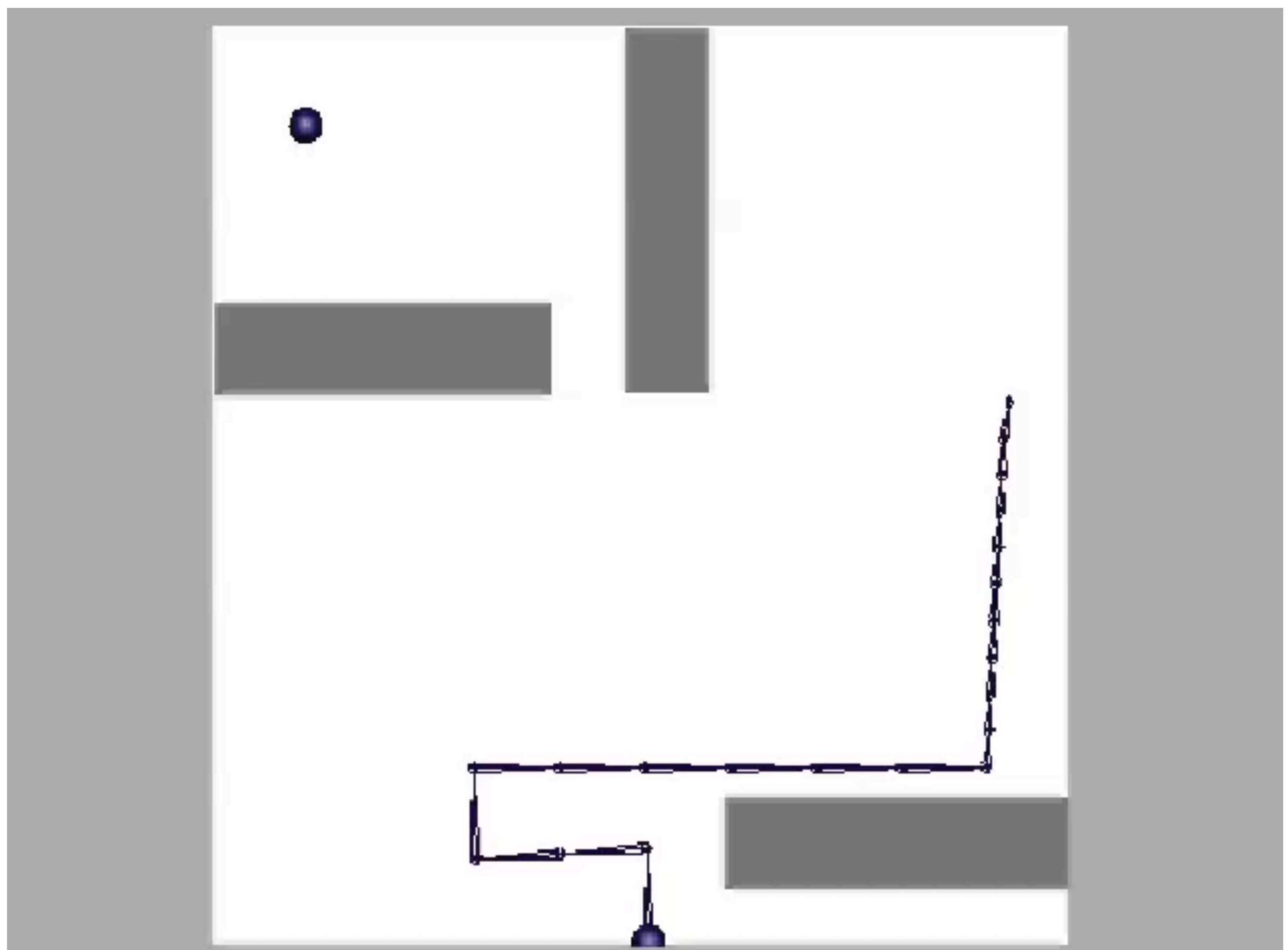
planning with ARA* (anytime version of weighted A*)

Courtesy Max Likhachev

Effect of the Heuristic Function

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



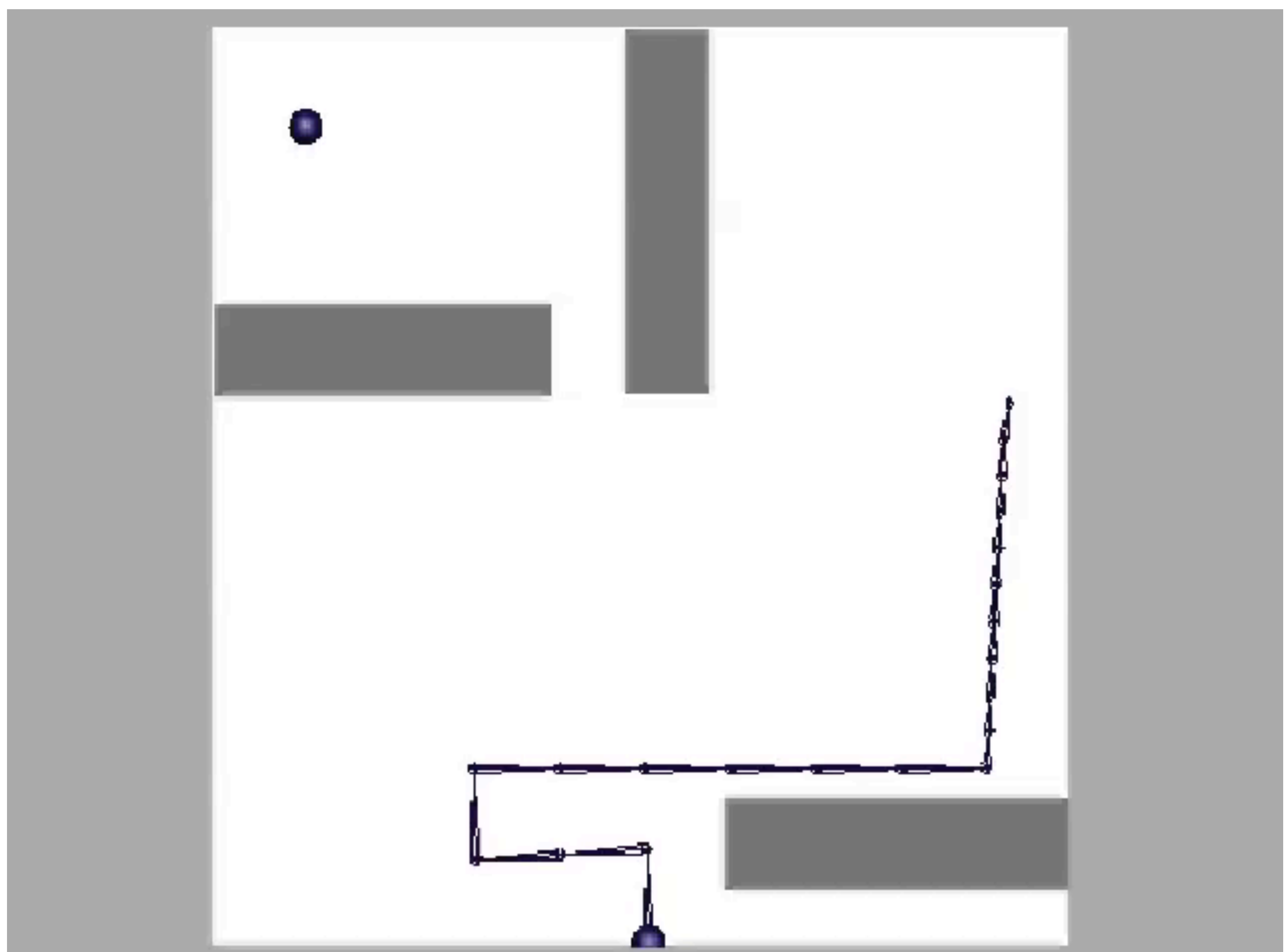
planning with ARA* (anytime version of weighted A*)

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



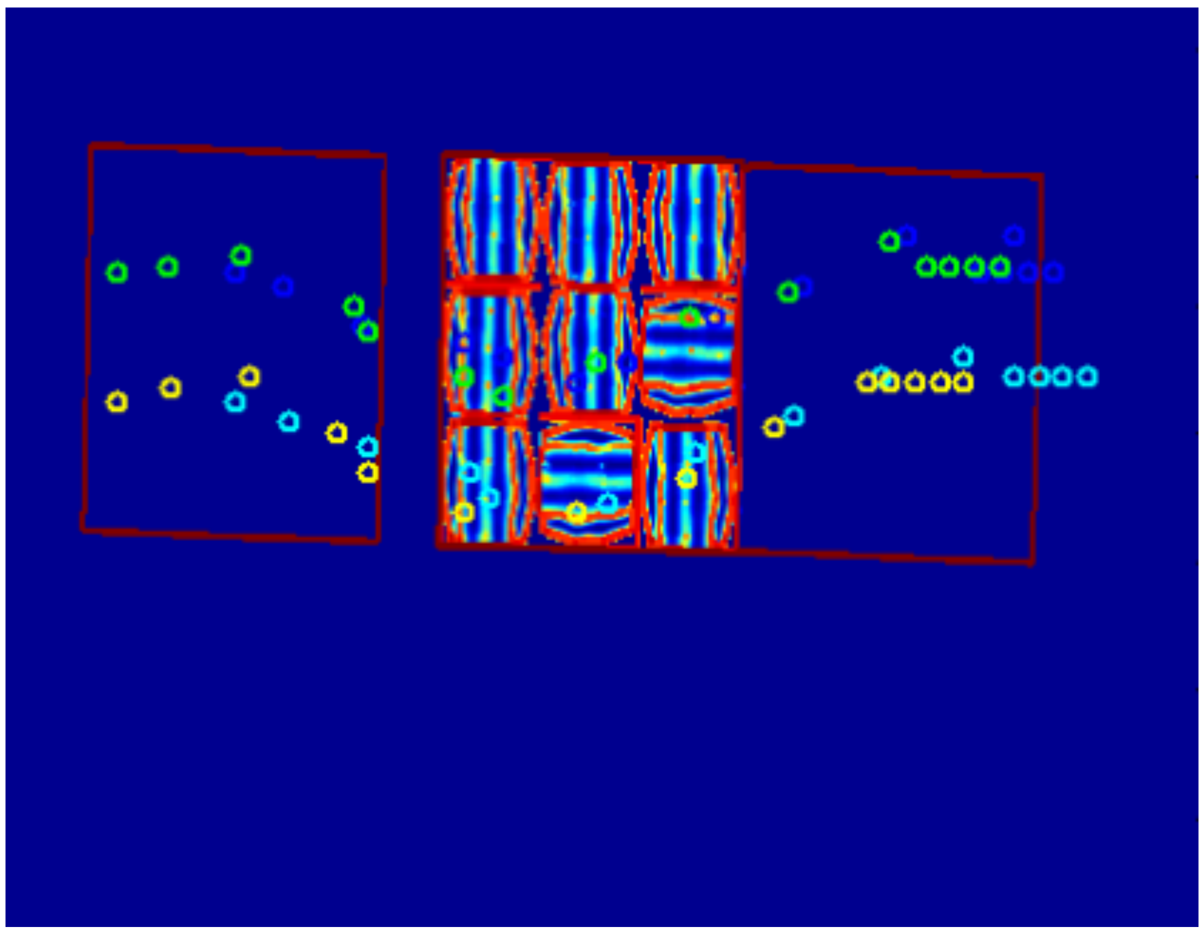
planning with ARA* (anytime version of weighted A*)

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

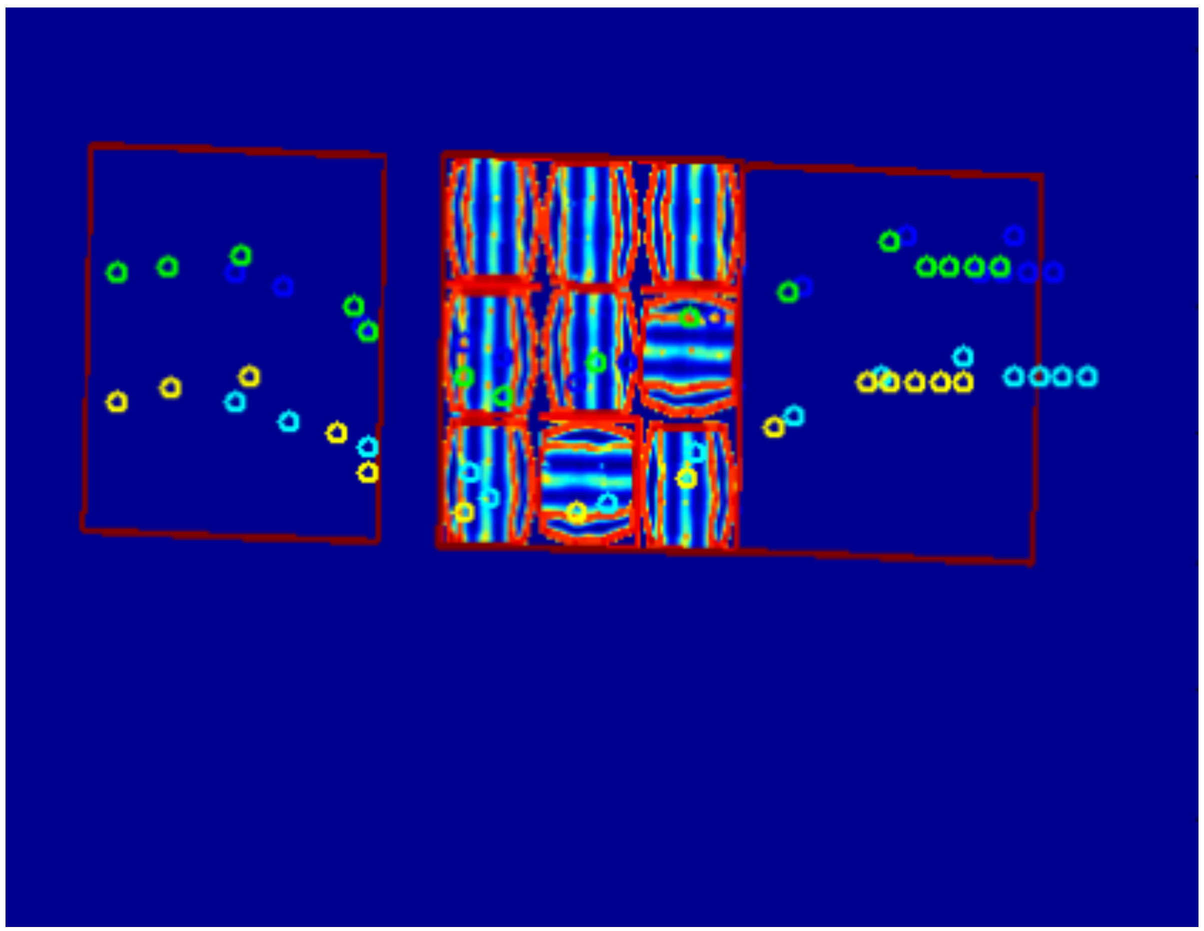


Uses R* - A randomized version of weighted A*
Joint work between Max Likhachev, Subhrajit Bhattacharya, Joh Bohren, Sachin Chitta, Daniel D. Lee, Aleksandr Kushleyev, and Paul Vernaza

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



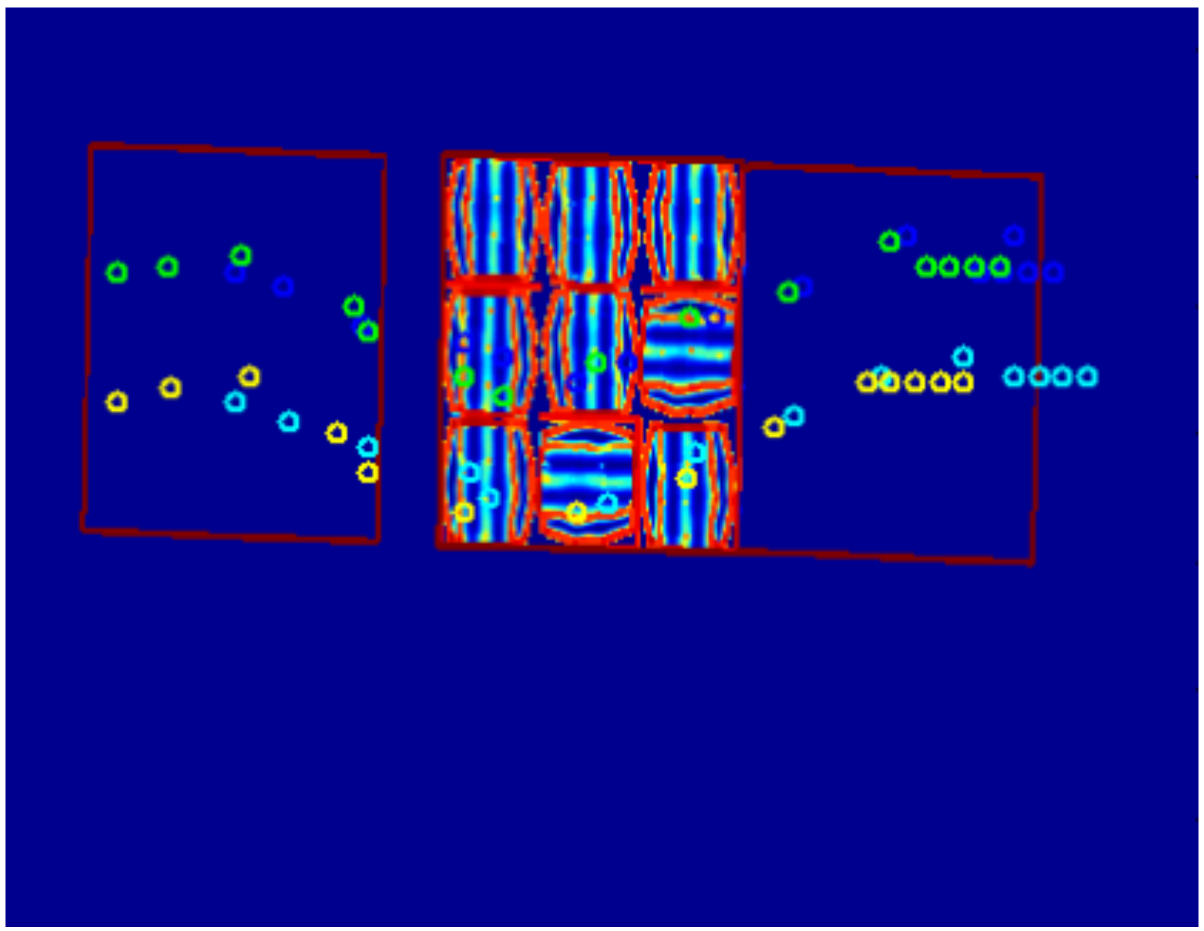
Uses R* - A randomized version of weighted A*

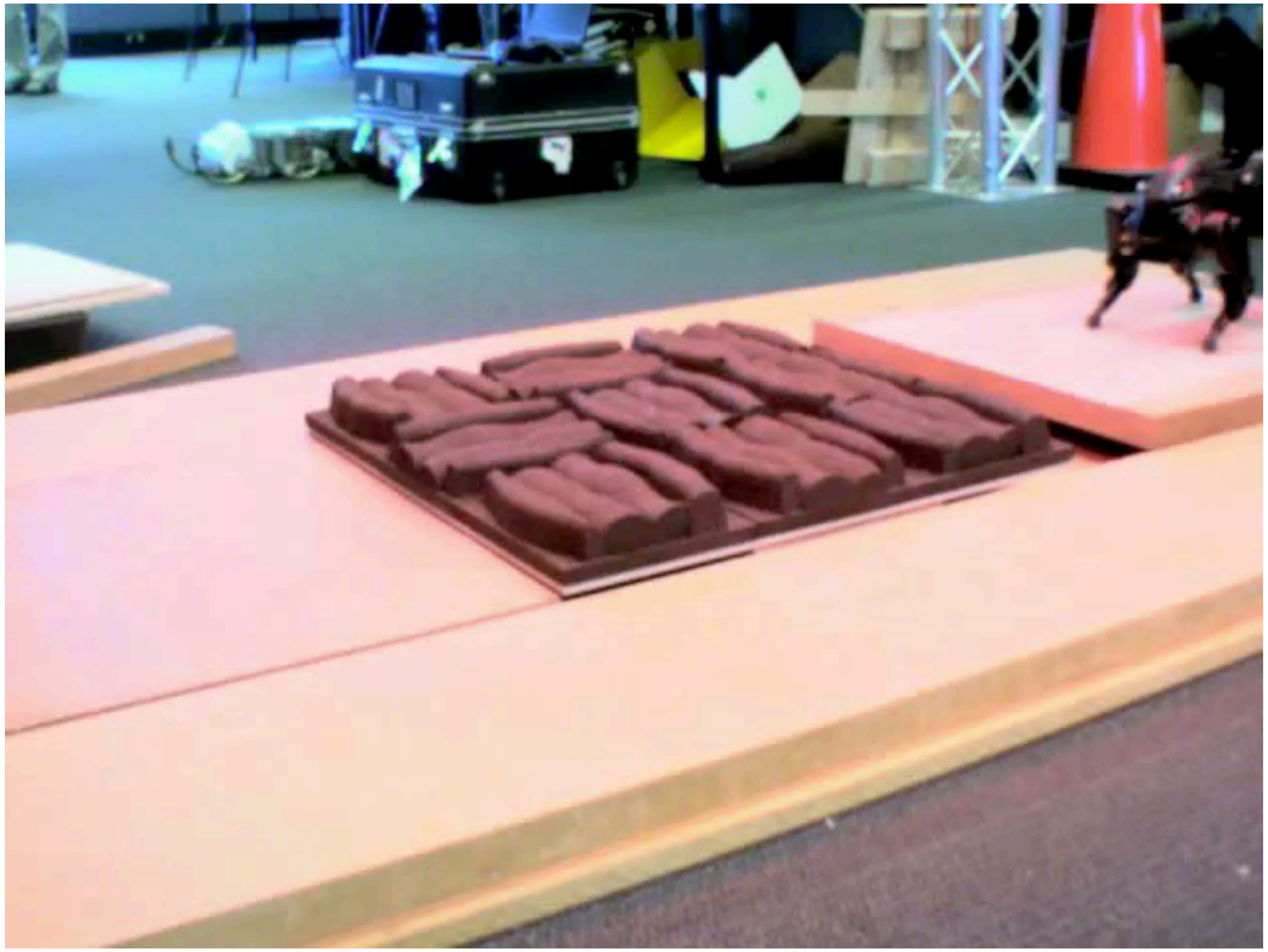
Joint work between Max Likhachev, Subhrajit Bhattacharya, Joh Bohren, Sachin Chitta, Daniel D. Lee, Aleksandr Kushleyev, and Paul Vernaza

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





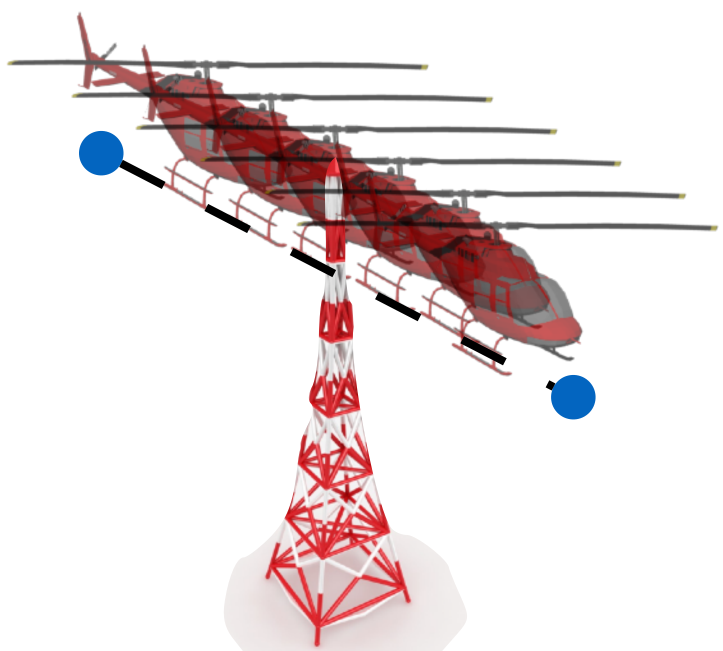
Uses R* - A randomized version of weighted A*

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

But is the number of expansions really what we want to minimize in motion planning?

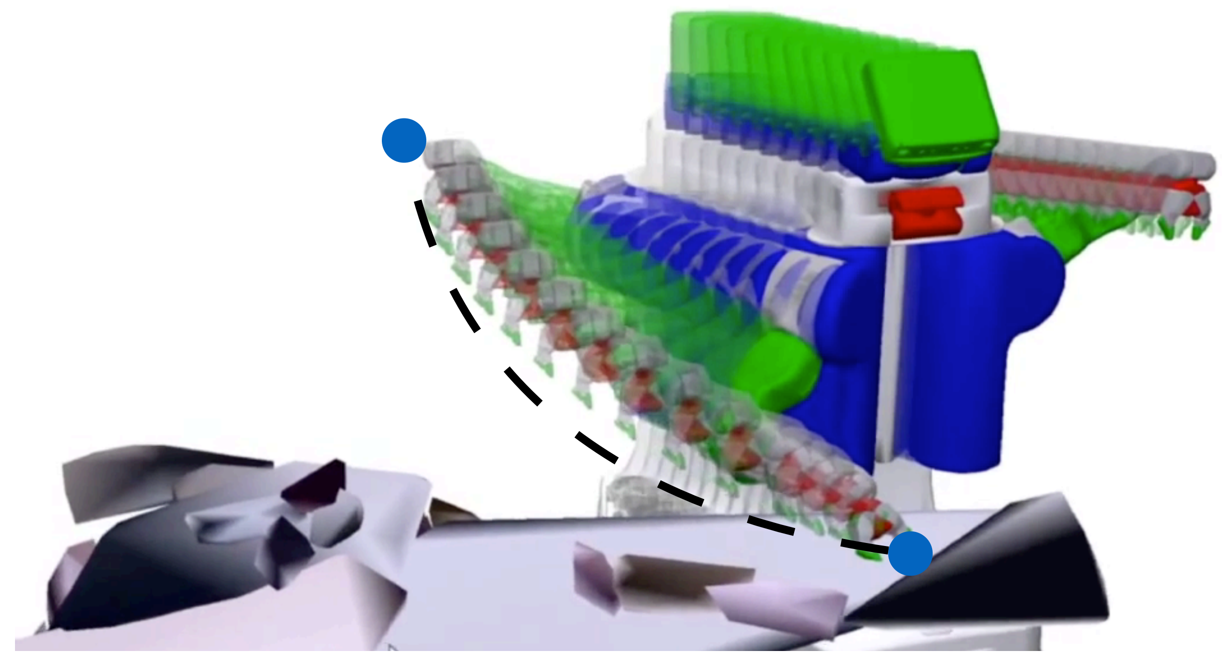
What is the most expensive step?

Edge evaluation is expensive



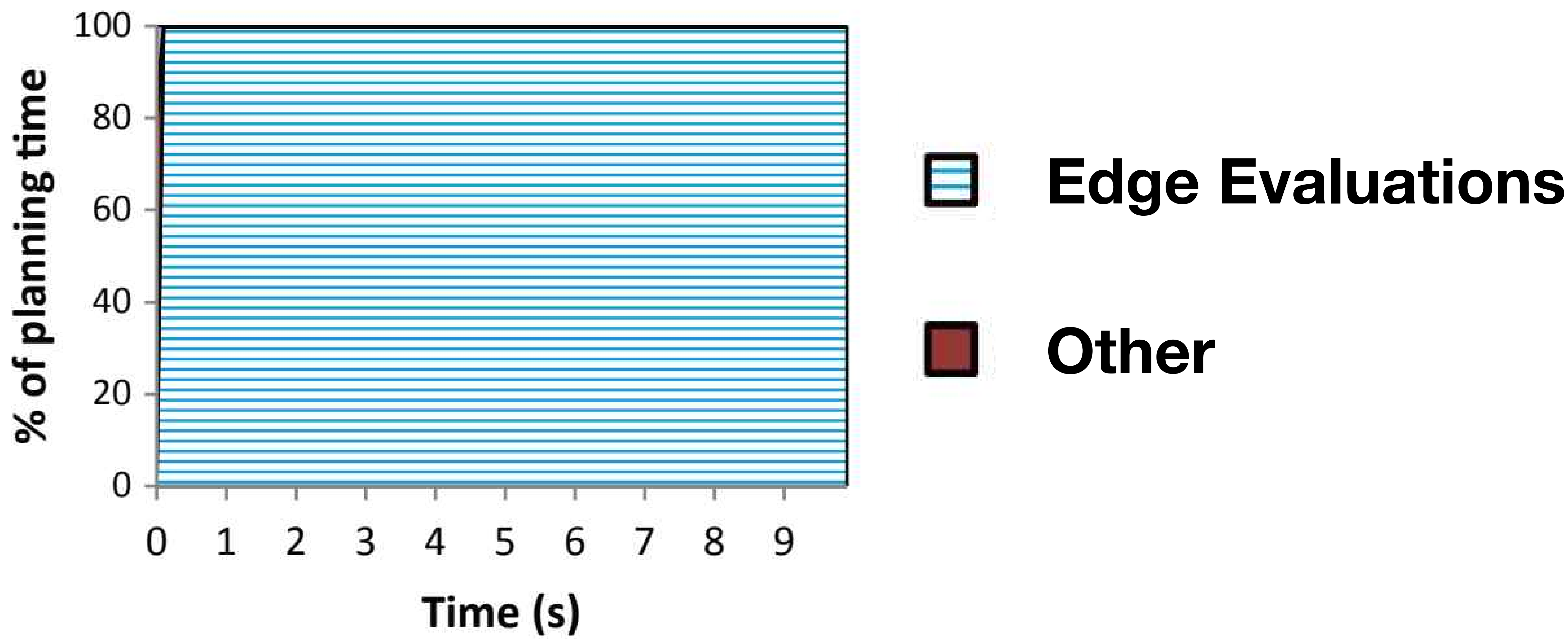
Check if helicopter intersects with tower

(Schluman et al. ’14)



Check if manipulator intersects with table

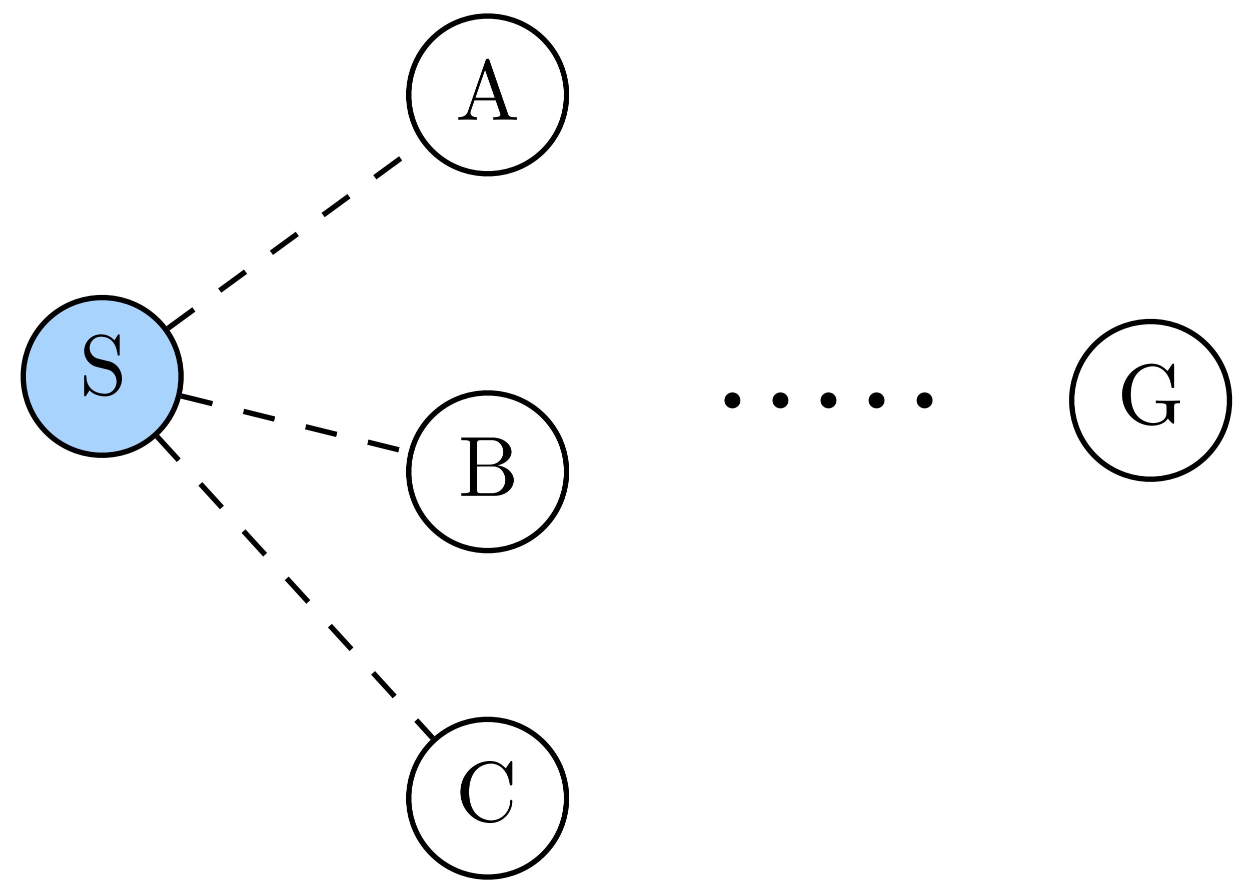
Edge evaluation dominates planning time



Hauser, Kris., Lazy collision checking in asymptotically-optimal motion planning. *ICRA* 2015

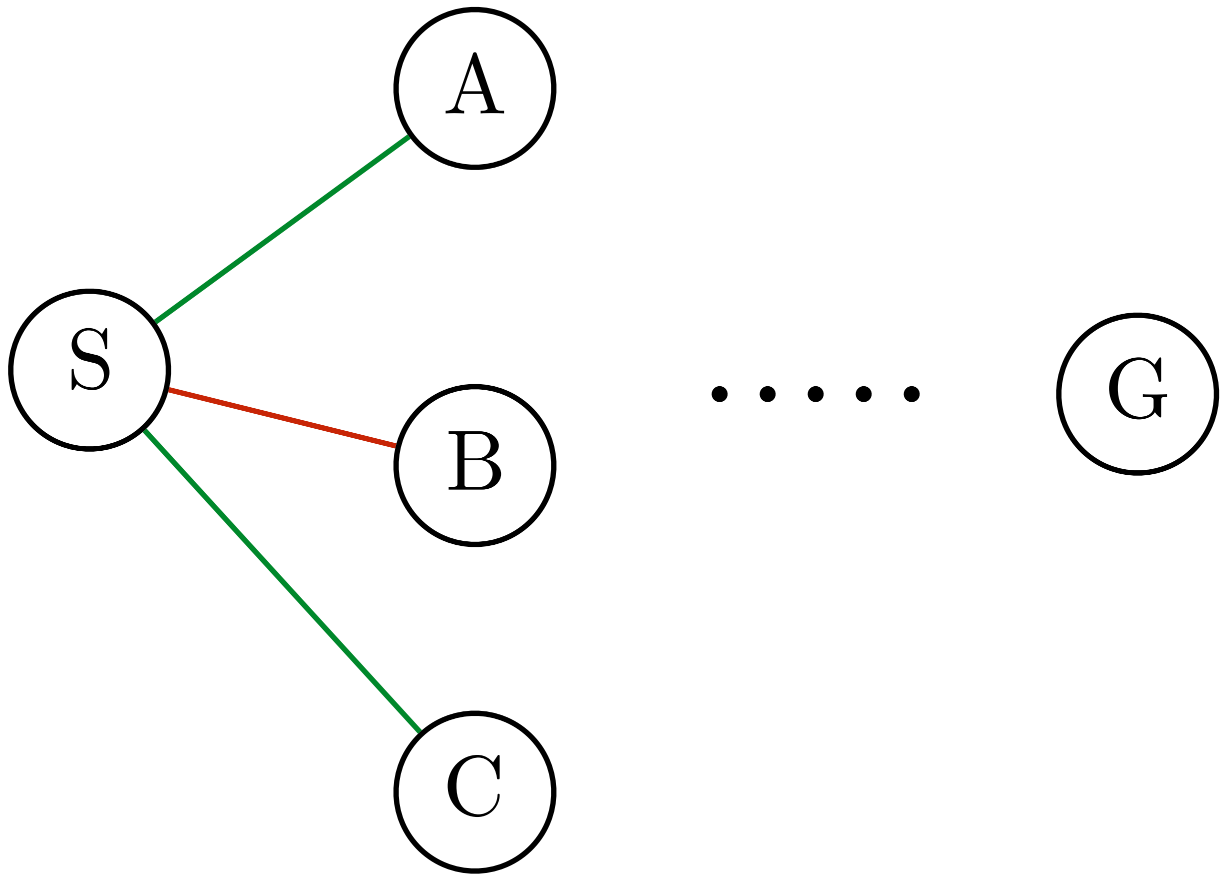
Let's revisit Best First Search

Element (Node)	Priority Value (f-value)
Node S	f(S)



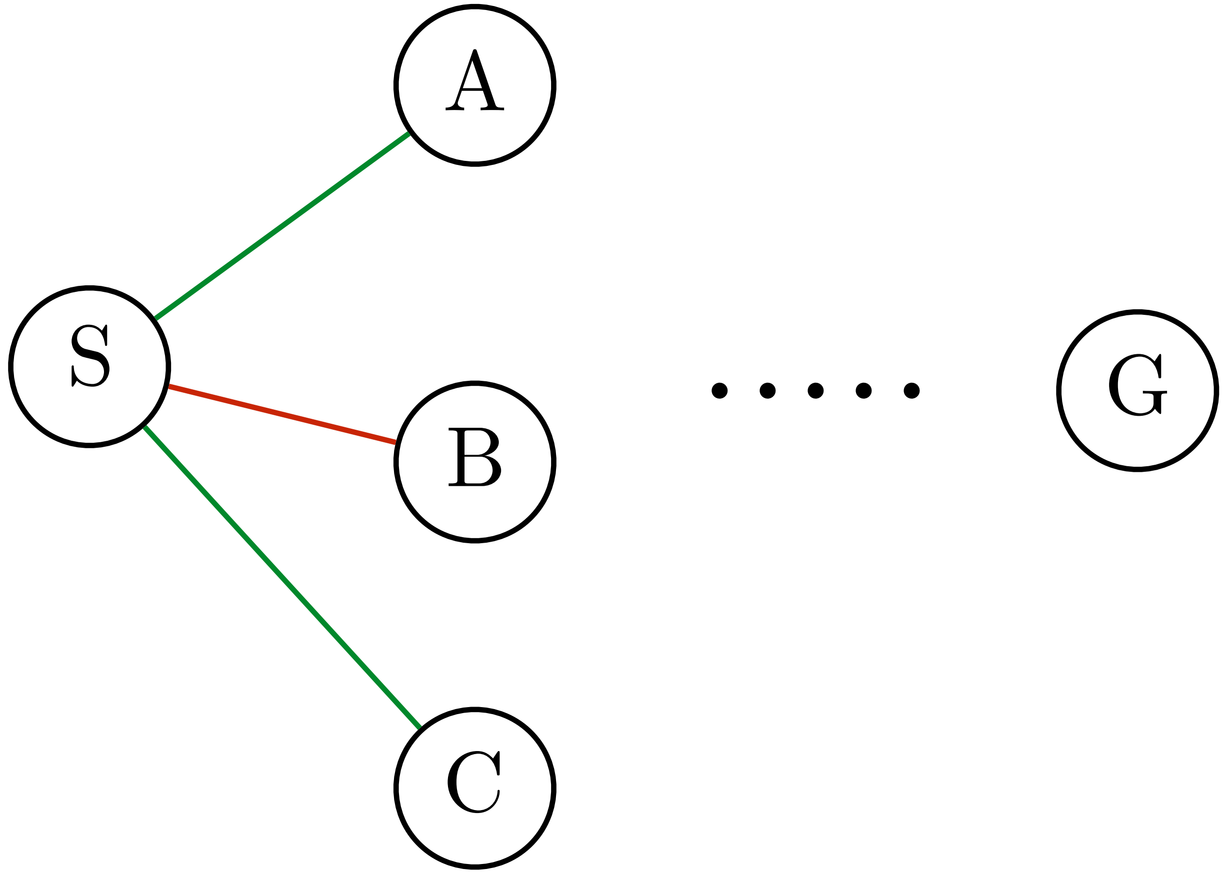
Let's revisit Best First Search

Element (Node)	**Priority Value (f-value)**
~~Node S~~	~~f(S)~~
Node A	f(A)
Node C	f(C)



What if we never use C? Wasted collision check!

Element (Node)	**Priority Value (f-value)**
~~Node S~~	~~f(S)~~
Node A	f(A)
Node C	f(C)



The provable virtue of laziness:

Take the thing that's expensive
(collision checking)
and
procrastinate as long as possible
till you have to evaluate it!

Lazy (weighted) A*

Cohen, Phillips, and Likhachev 2014

Lazy (weighted) A*

Cohen, Phillips, and Likhachev 2014

Key Idea:

Lazy (weighted) A*

Cohen, Phillips, and Likhachev 2014

Key Idea:

1. When expanding a node, don't collision check edge to successors

(be optimistic and assume the edge will be valid)

Lazy (weighted) A*

Cohen, Phillips, and Likhachev 2014

Key Idea:

1. When expanding a node, don't collision check edge to successors

   (be optimistic and assume the edge will be valid)

2. When expanding a node, collision check the edge to parent

   (expansion means this node is good and worth the effort)

Lazy (weighted) A*

Cohen, Phillips, and Likhachev 2014

Key Idea:

1. When expanding a node, don't collision check edge to successors

   (be optimistic and assume the edge will be valid)

2. When expanding a node, collision check the edge to parent

   (expansion means this node is good and worth the effort)

3. Important: OPEN list will have multiple copies of a node

   (multiple candidate parents since we haven't collision check)

Lazy A*

Cohen, Phillips, and Likhachev 2014

Non lazy A*

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 *edge* (s,s') in collision

$c(s,s') = \infty$

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

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

insert s' into *OPEN*;

Lazy A*

Cohen, Phillips, and Likhachev 2014

Non lazy A*

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 *edge* (s,s') in collision
$c(s,s') = \infty$
if $g(s') > g(s) + c(s,s')$
$g(s') = g(s) + c(s,s')$;
insert s' into *OPEN*;

Lazy A*

while(s_{goal} is not expanded)

remove s with the smallest
$[f(s) = g(s)+h(s)]$ from *OPEN*;
if s is in CLOSED
continue;

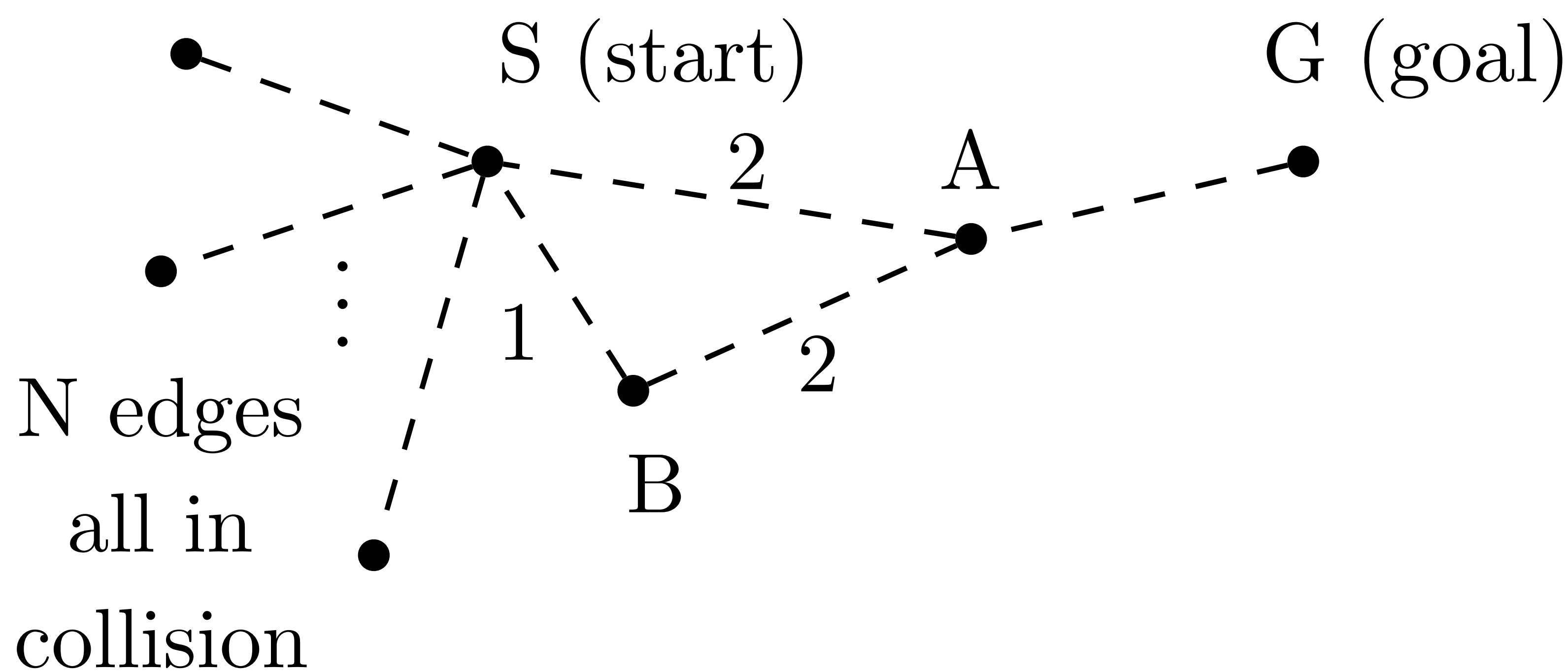
if *edge(parent(s), s)* in collision
continue;

insert s into *CLOSED*;

for every successor s' of s such
that s' not in *CLOSED*
no collision checking of edge
if $g(s') > g(s) + c(s,s')$
$g(s') = g(s) + c(s,s')$;
insert s' into *OPEN*; // multiple
copies

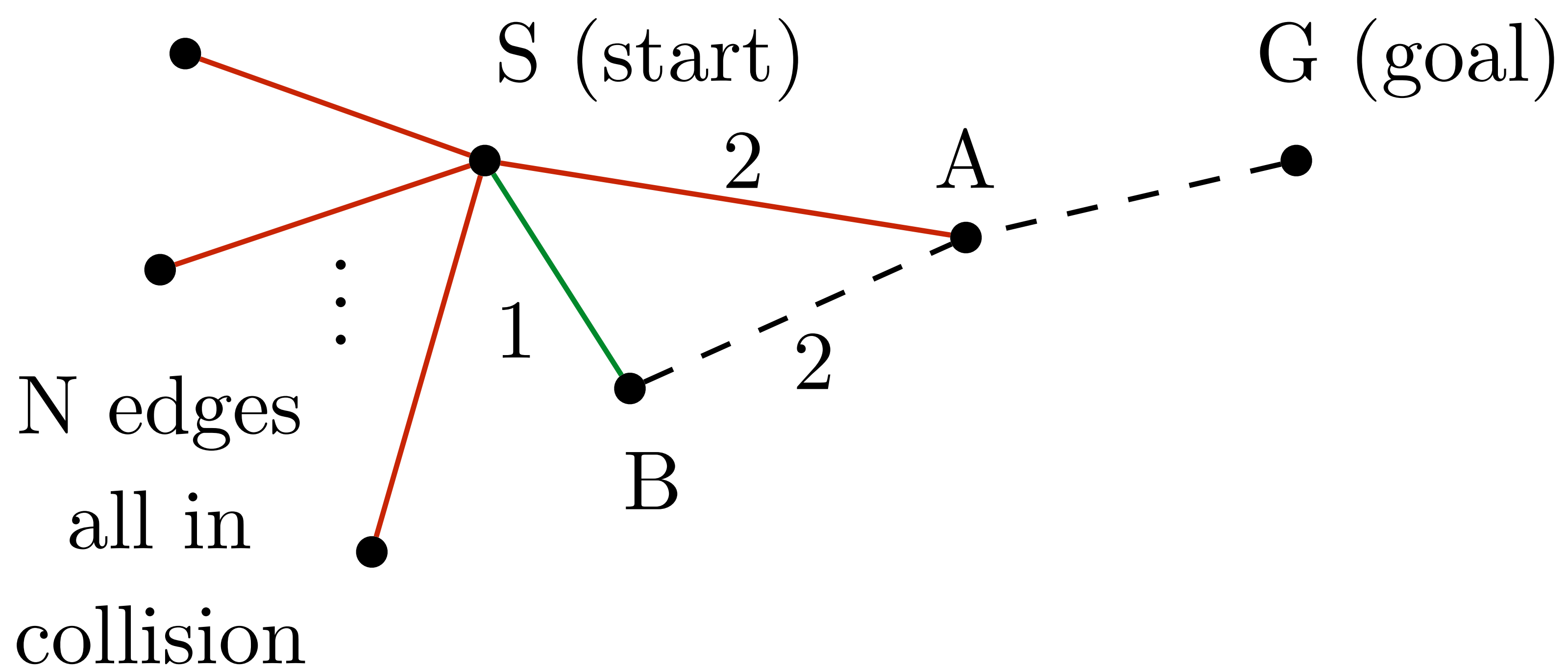
A*

Let's say S-A is in collision and true shortest path is S-B-A-G



A*

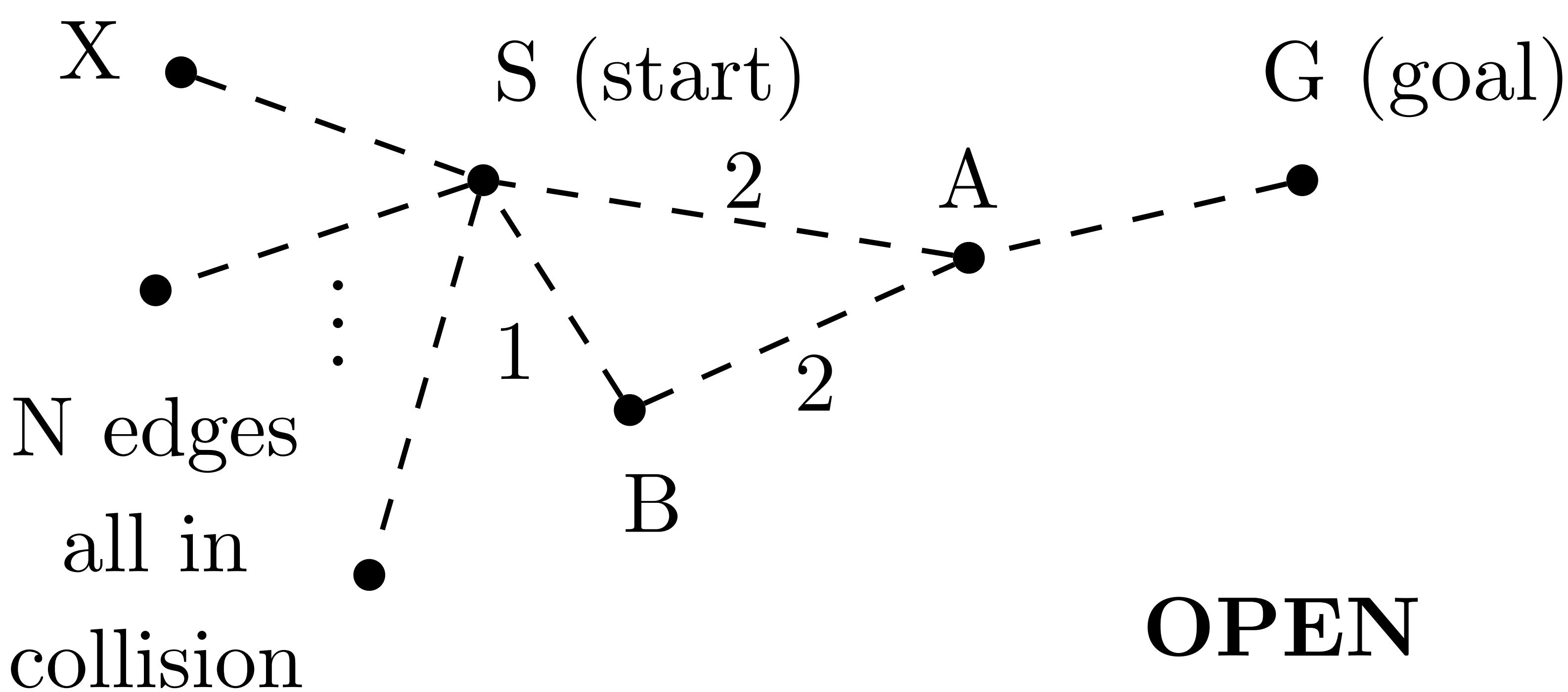
Let's say S-A is in collision and true shortest path is S-B-A-G



A* will
collision check
all N+2 edges!

Lazy A*

Let's say S-A is in collision and true shortest path is S-B-A-G

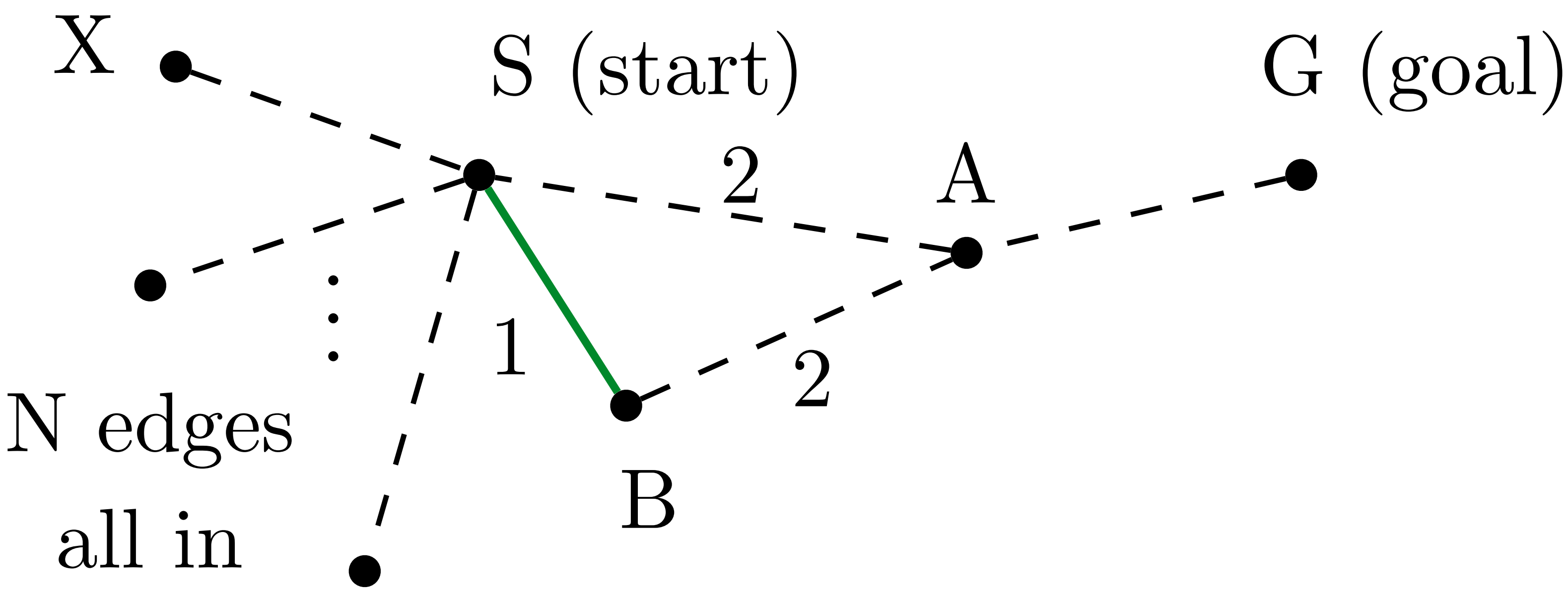


Lets set f(s) = g(s)

	OPEN	**CLOSED**	**CollChecked**
f = 1	B (from S)	S	
f = 2	A (from S)		
f = 1000	X (from S)		
	.....		

Lazy A*

Let's say S-A is in collision and true shortest path is S-B-A-G

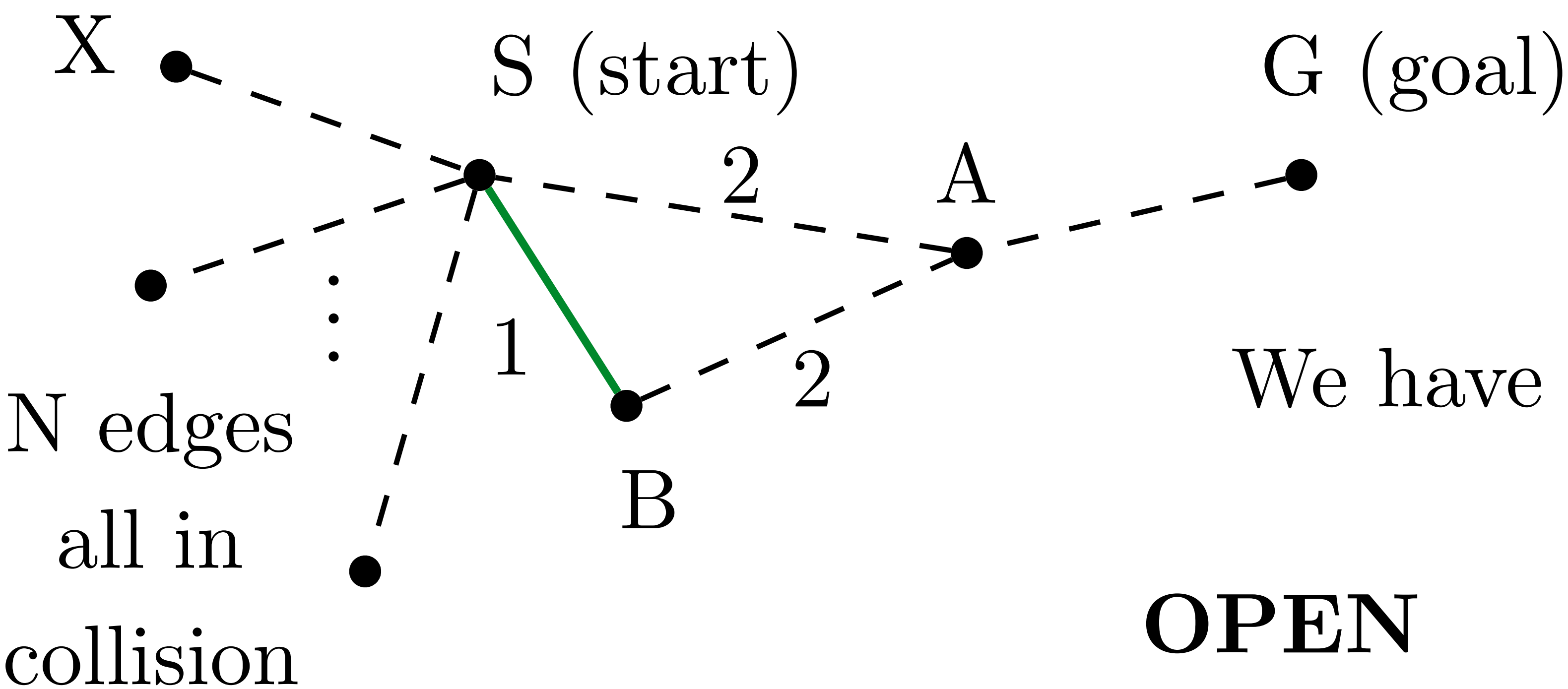


	OPEN	CLOSED	CollChecked
f = 2	A (from S)	S	S-B
f = 3	A (from B)	B	

.....

Lazy A*

Let's say S-A is in collision and true shortest path is S-B-A-G



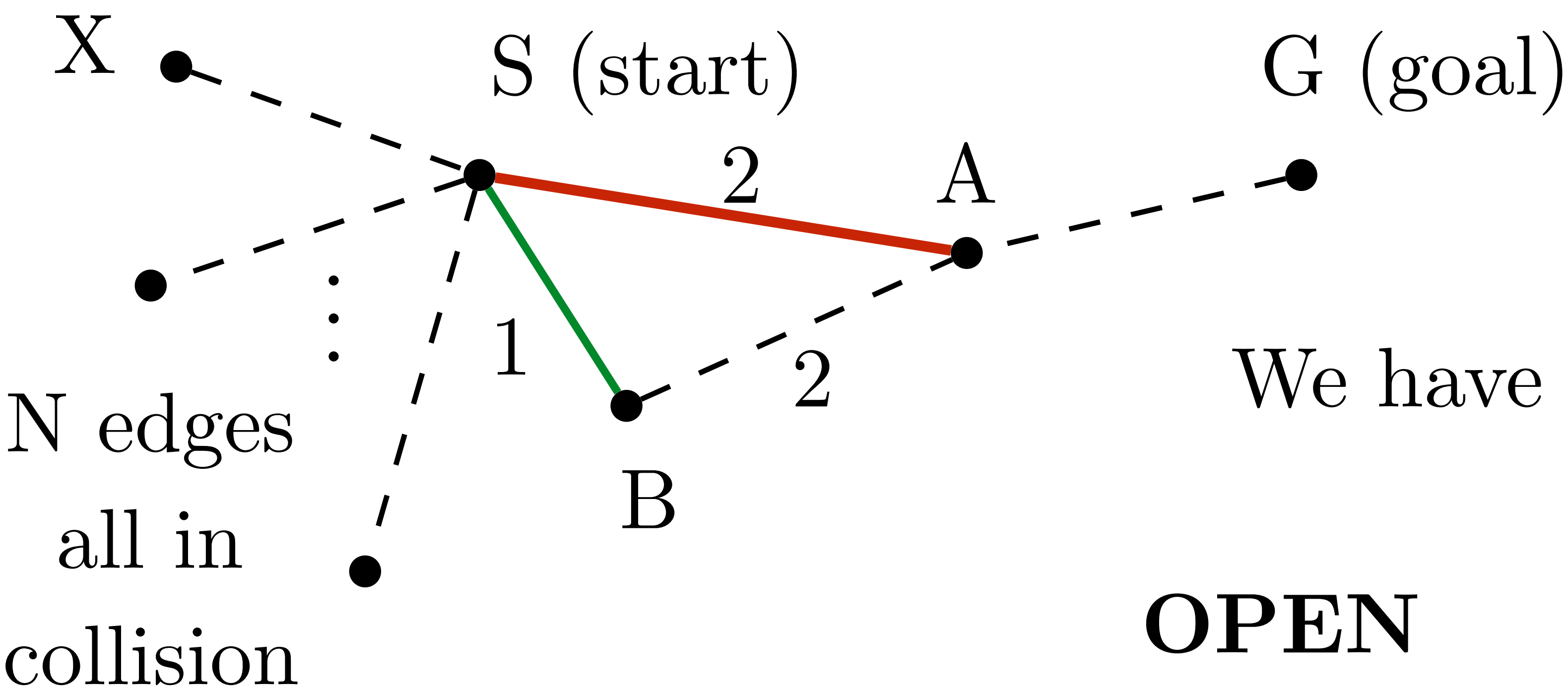
We have TWO copies of A in OPEN!

	OPEN	**CLOSED**	**CollChecked**
f = 2	A (from S)	S	S-B
f = 3	A (from B)	B	

.....

Lazy A*

Let's say S-A is in collision and true shortest path is S-B-A-G

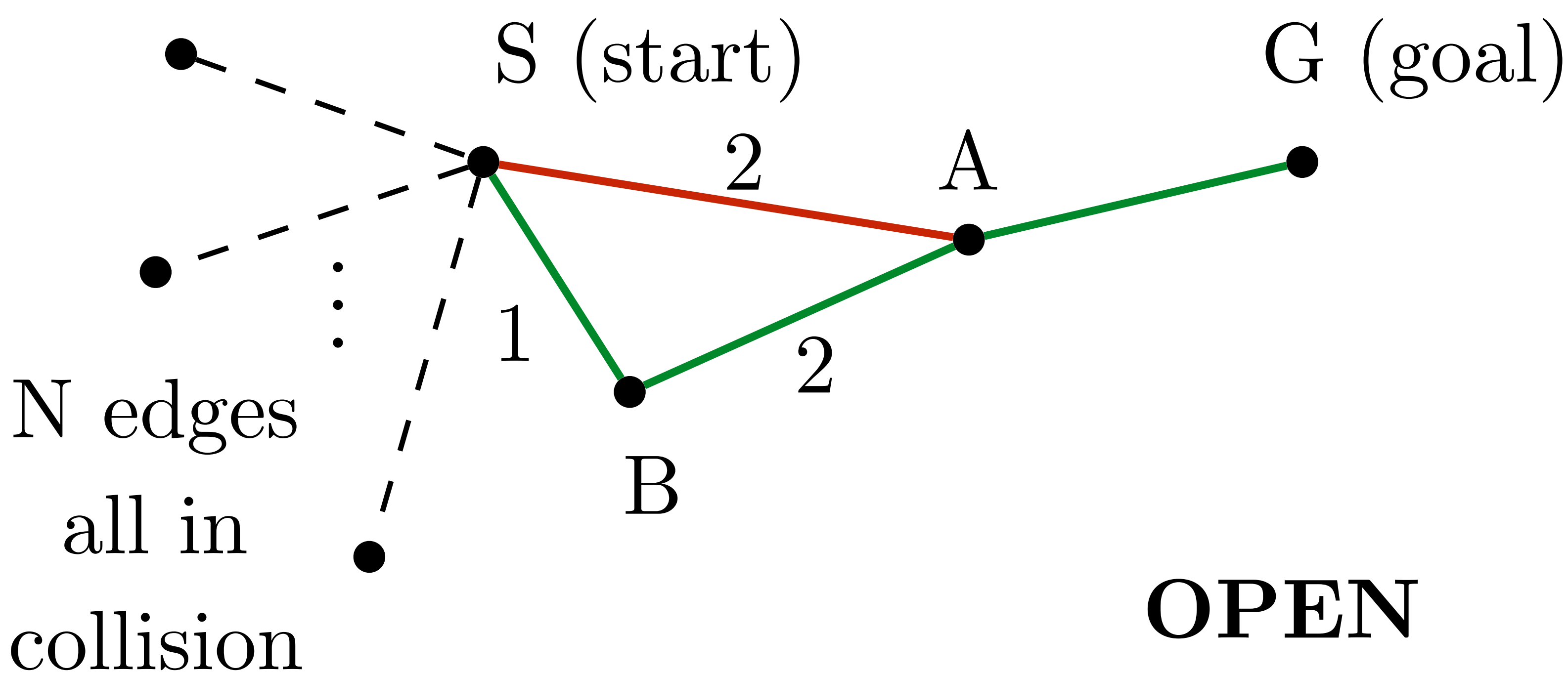


We have TWO copies of A in OPEN!

	OPEN	CLOSED	CollChecked
f = 3	A (from B)	S	S-B
		B	
	.....	A	

Lazy A*

Let's say S-A is in collision and true shortest path is S-B-A-G



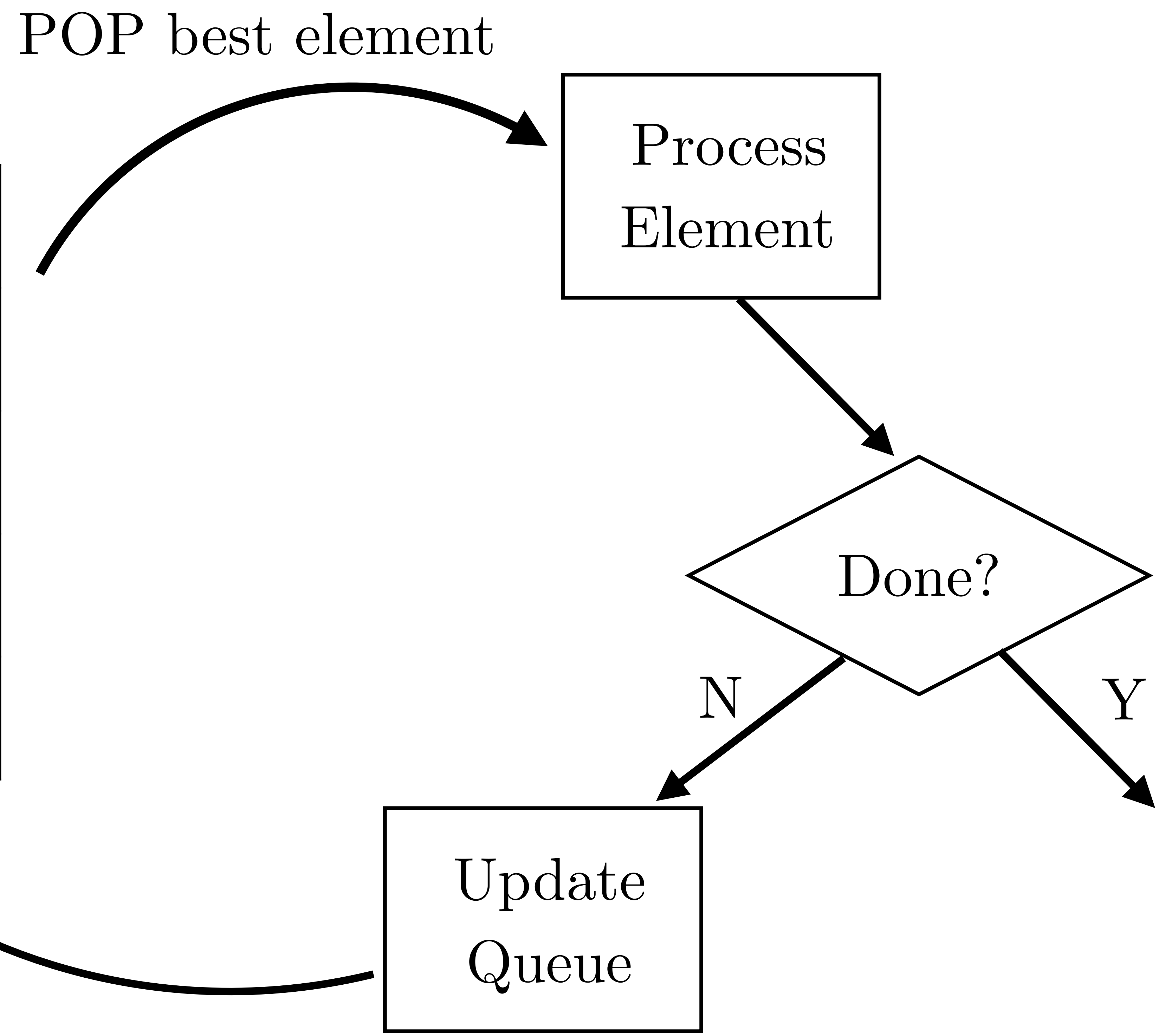
OPEN	**CLOSED**	**CollChecked**
	S	S-B
	B	S-A
	A	B-A
	G	A-G

Yet another algorithm to remember....!?!?!

Best First Search explains it all

Best First Search as a Meta++ algorithm

Element	**Ranking value**
Element A	f(A)
Element B	f(B)



If we want to minimize the number of vertices expanded, then elements of the queue should be candidate vertices to expand!

If we want to minimize the

number of edges evaluated,

then

……

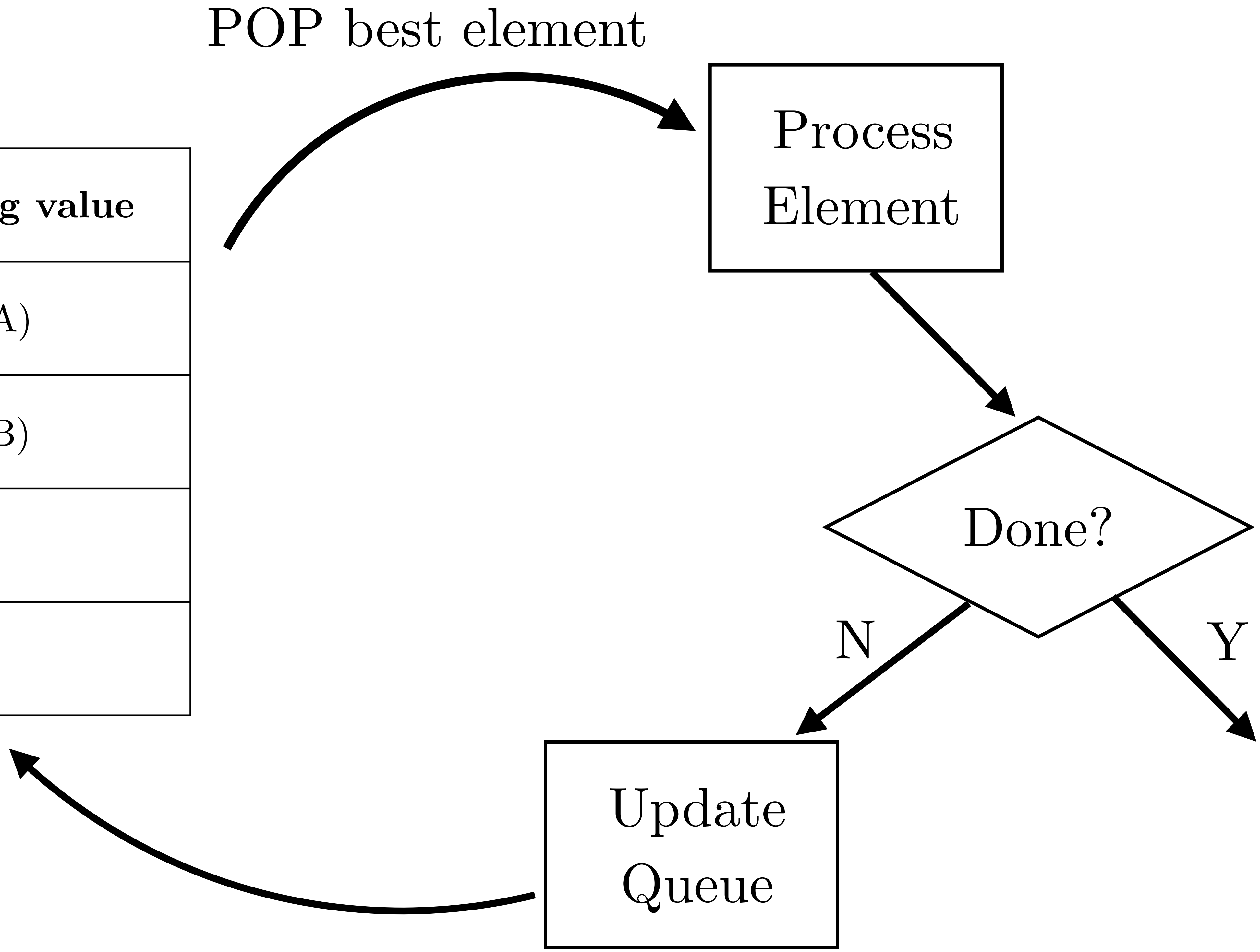
If we want to minimize the number of edges evaluated, then

......

elements of the queue should be candidate edges to evaluate!

Best First Search as a Meta++ algorithm

Element	**Ranking value**
Element A	f(A)
Element B	f(B)



How do we define a queue over edges???

OPEN list of A*

Element (Vertex)	**Value (f-value of path through vertex)**
Vertex A	$f(A) = g(A) + h(A)$
Vertex B	$f(B) = g(B) + h(B)$

How do we define a queue over edges???

OPEN list of A*

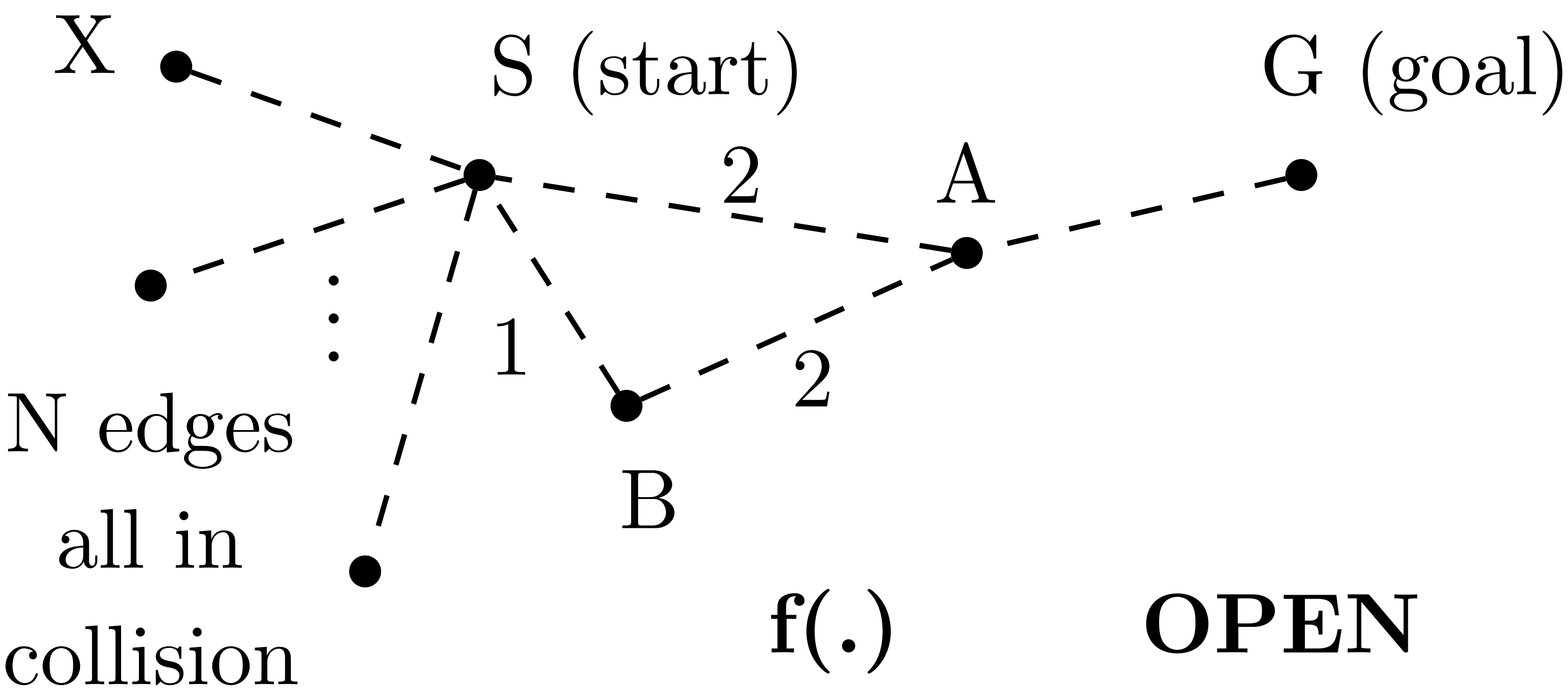
Element (Vertex)	**Value (f-value of path through vertex)**
Vertex A	f(A) = g(A) + h(A)
Vertex B	f(B) = g(B) + h(B)

OPEN list of Lazy A*

Element (Edge)	**Value (f-value of path through edge)**
Edge (X,Y)	f(X,Y) = g(X) + c(X,Y) + h(Y)
Edge (P,Q)	f(P,Q) = g(P) + c(P,Q) + h(Q)
Edge (W,Y)	f(W,Y) = g(W) + c(W,Y) + h(Y)

Lazy A*

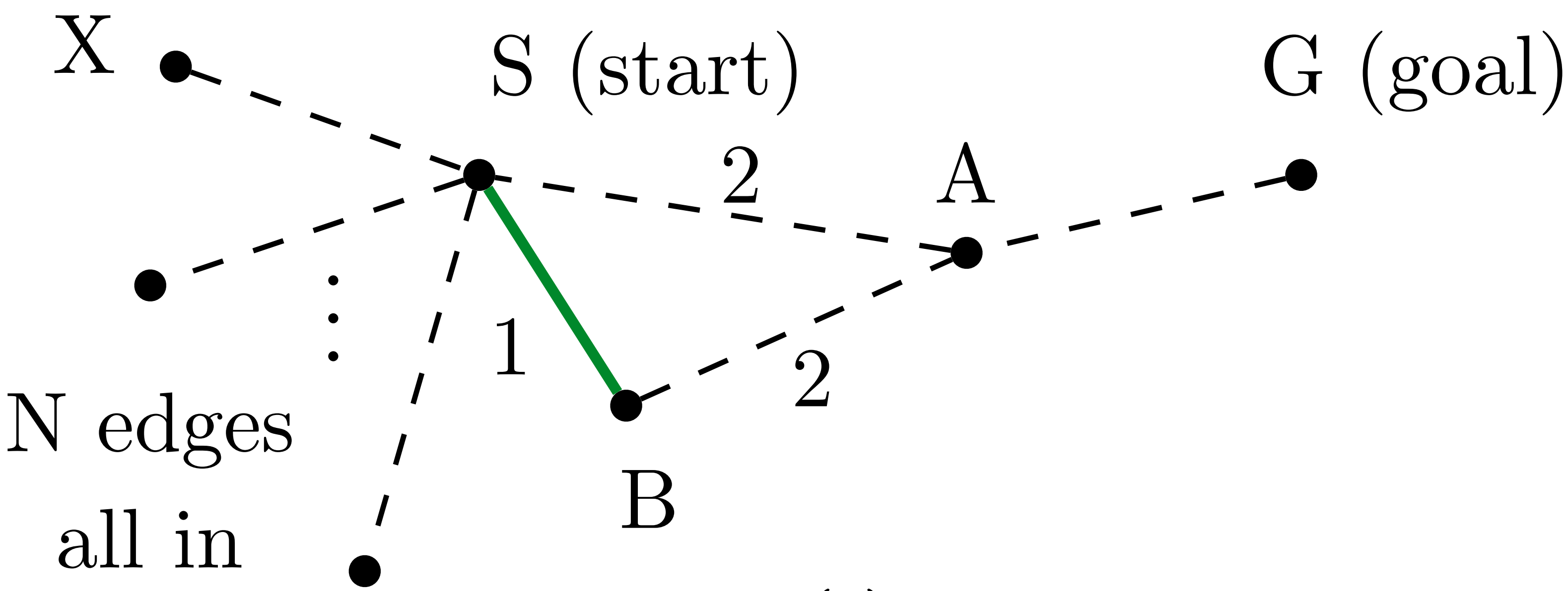
Let's say S-A is in collision and true shortest path is S-B-A-G



f(.)	OPEN	CLOSED	CollChecked
g(S) + c(S,B) + h(B)	S——B	S	
g(S) + c(S,A) + h(A)	S——A		
	.....		

Lazy A*

Let's say S-A is in collision and true shortest path is S-B-A-G



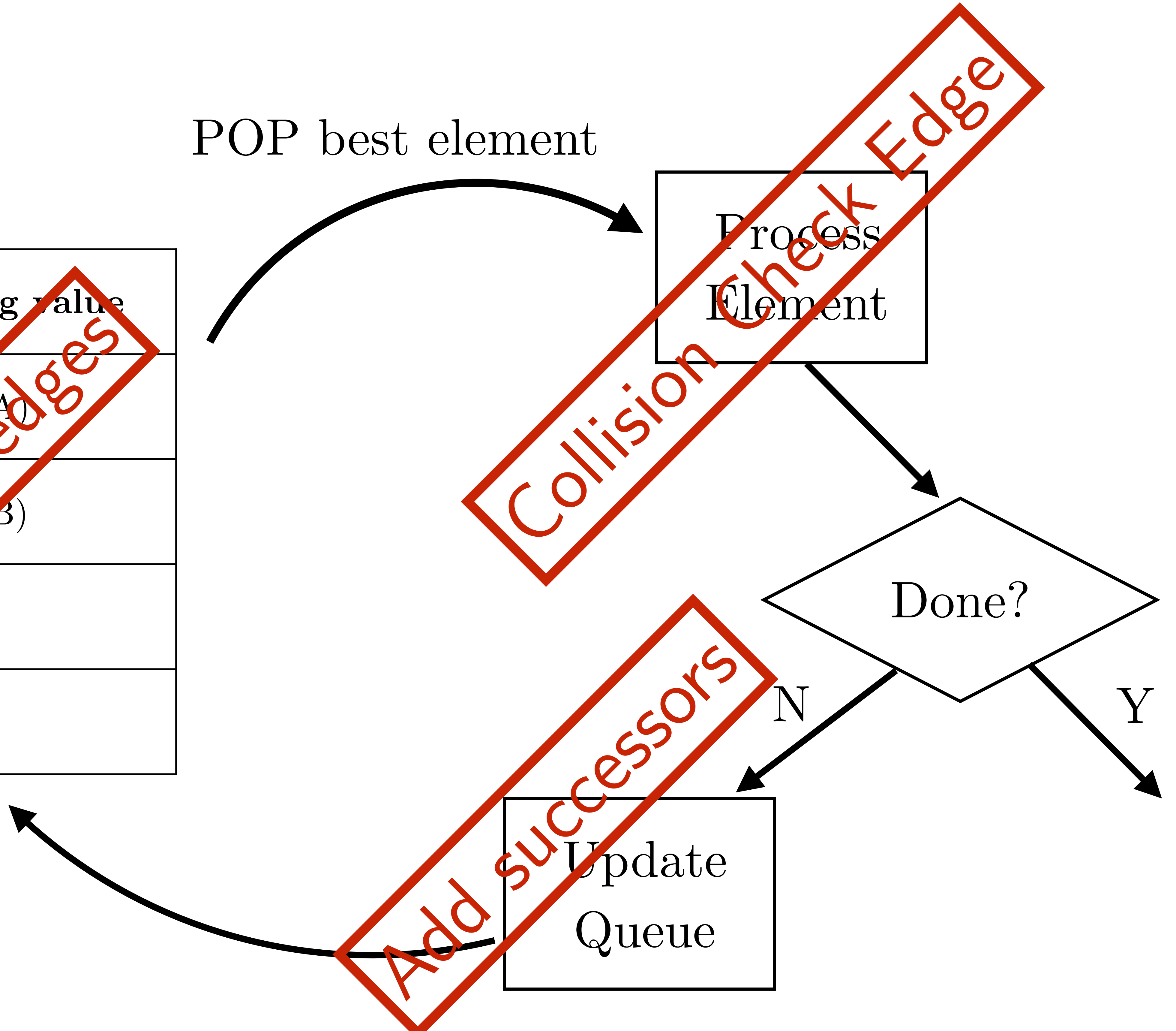
f(.)	**OPEN**	**CLOSED**	**CollChecked**
g(S) + c(S,A) + h(A)	S——A	S	S——B
g(B) + c(B,A) + h(A)	B——A	B	

.....

Lazy A* as BFS

Element	**Ranking value**
Element A	f(A)
Element B	f(B)

Queue over edges



What is the laziest that we can be?

What is the laziest that we can be?

LazySP

(Lazy Shortest Path)

Dellin and Srinivasa, 2016

First Provably Edge-Optimal A*-like Search Algorithm

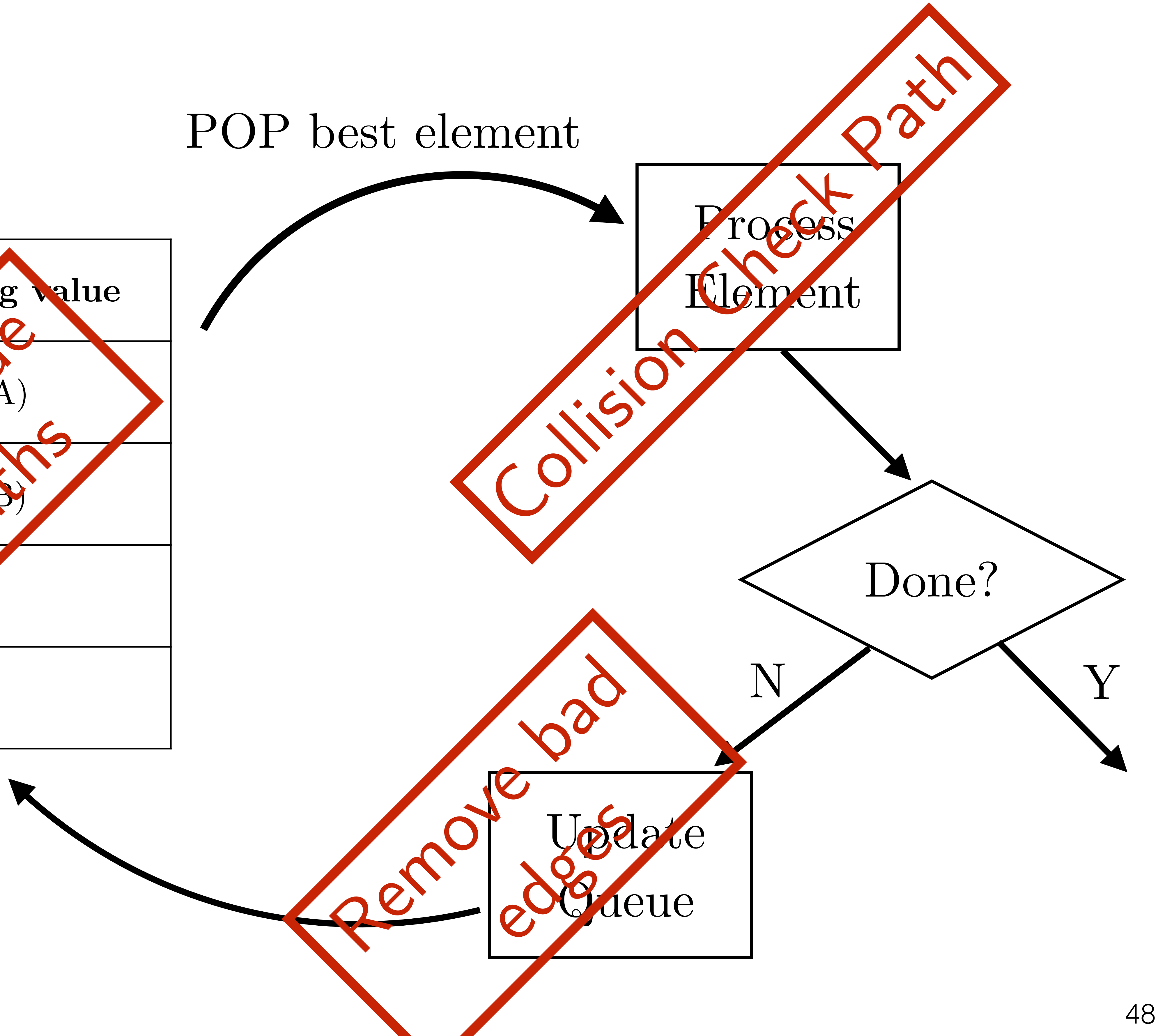
LazySP

Greedy Best-first Search over Paths

To find the shortest path,
eliminate all shorter paths!

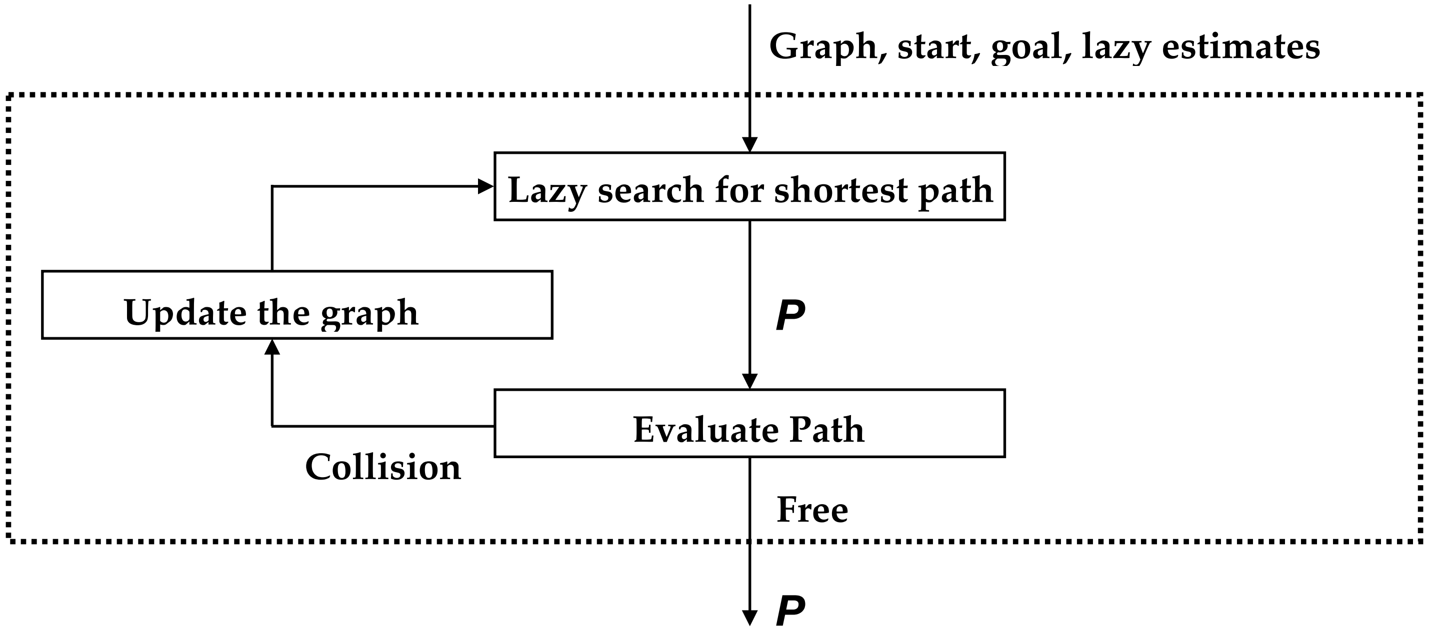
Lazy A* as BFS

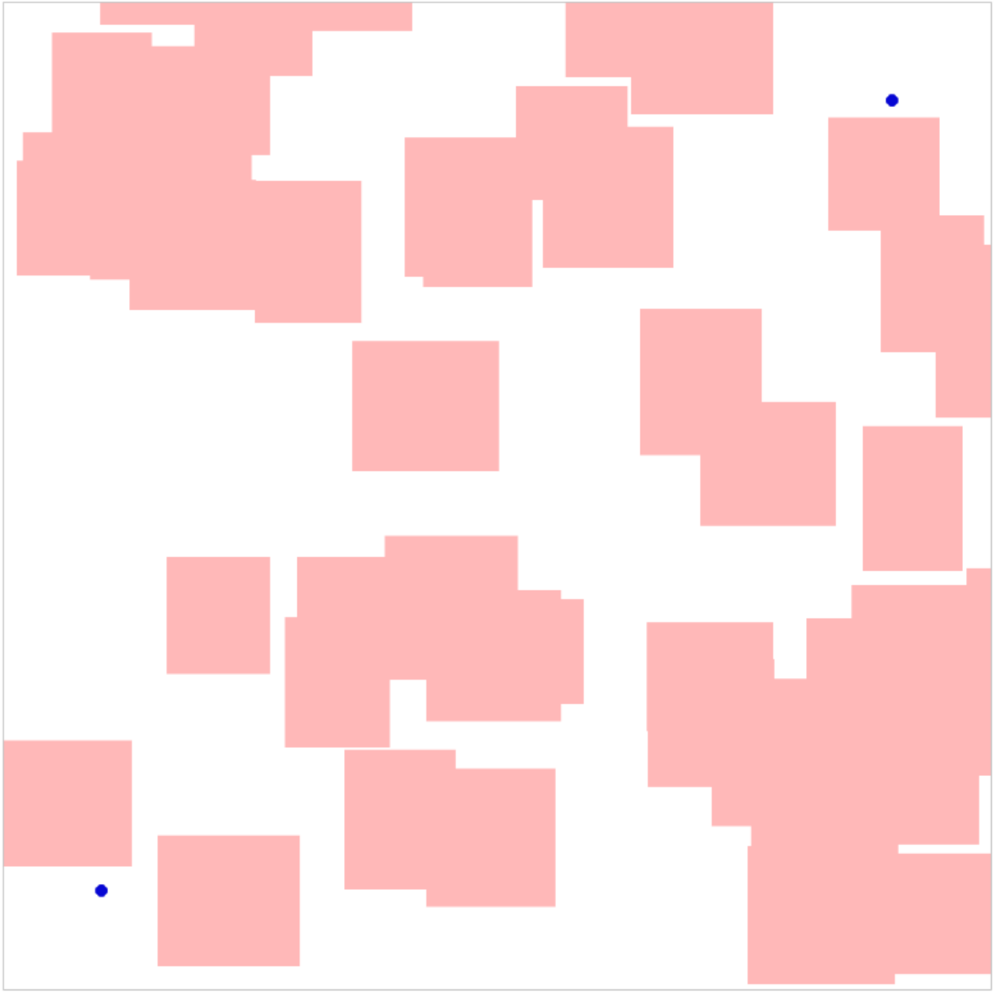
Element	Ranking value
Element A	f(A)
Element B	f(B)



LazySP

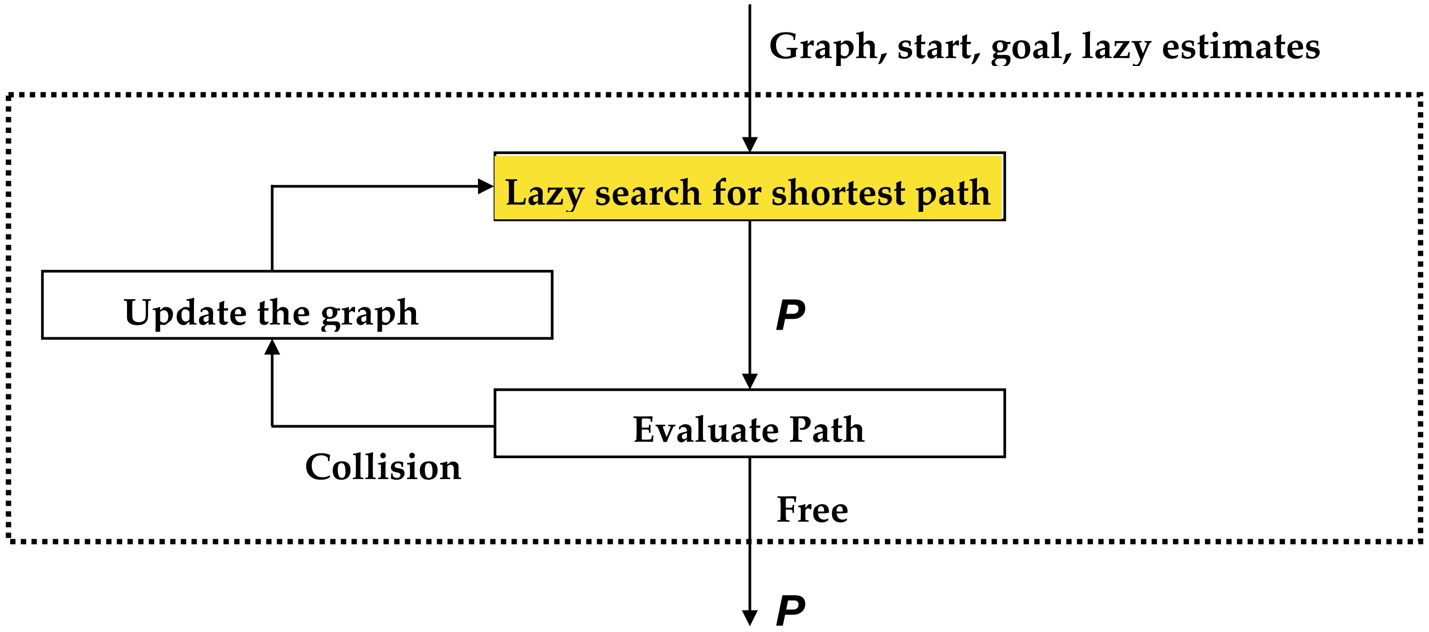
Optimism Under Uncertainty

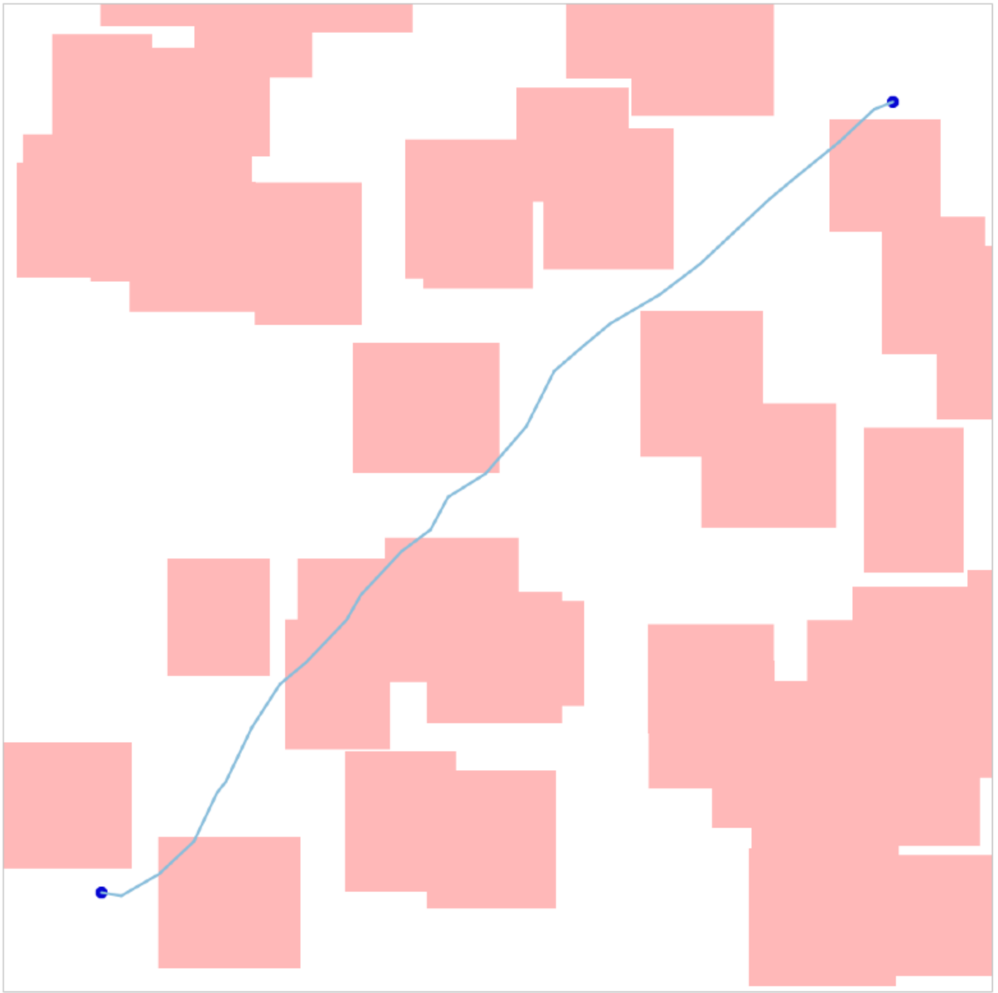




LazySP

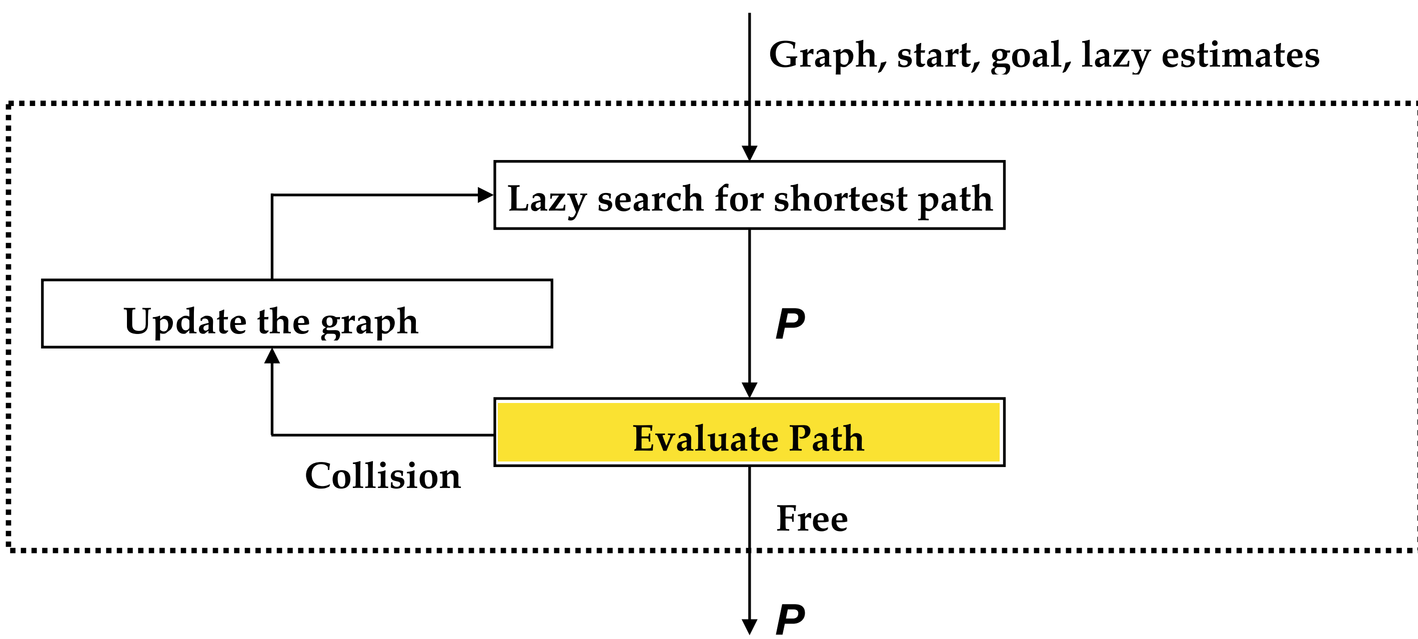
Optimism Under Uncertainty

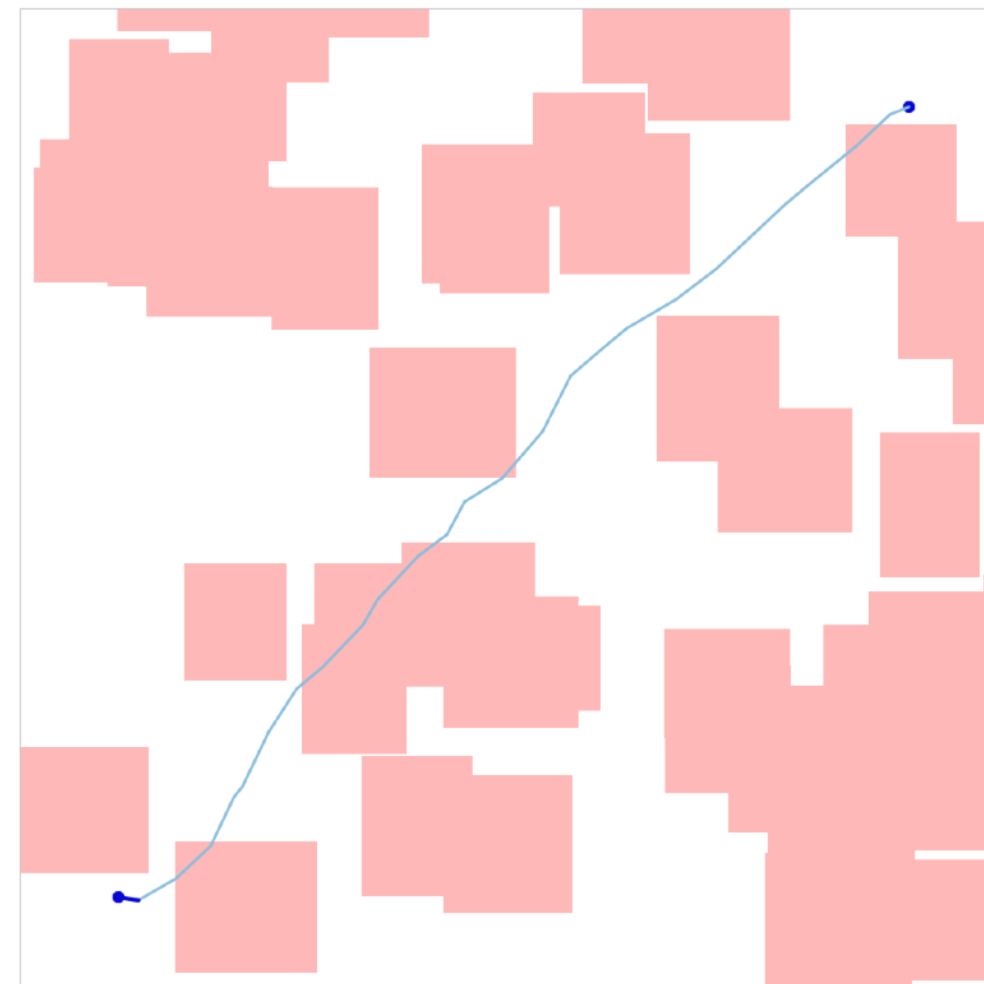




LazySP

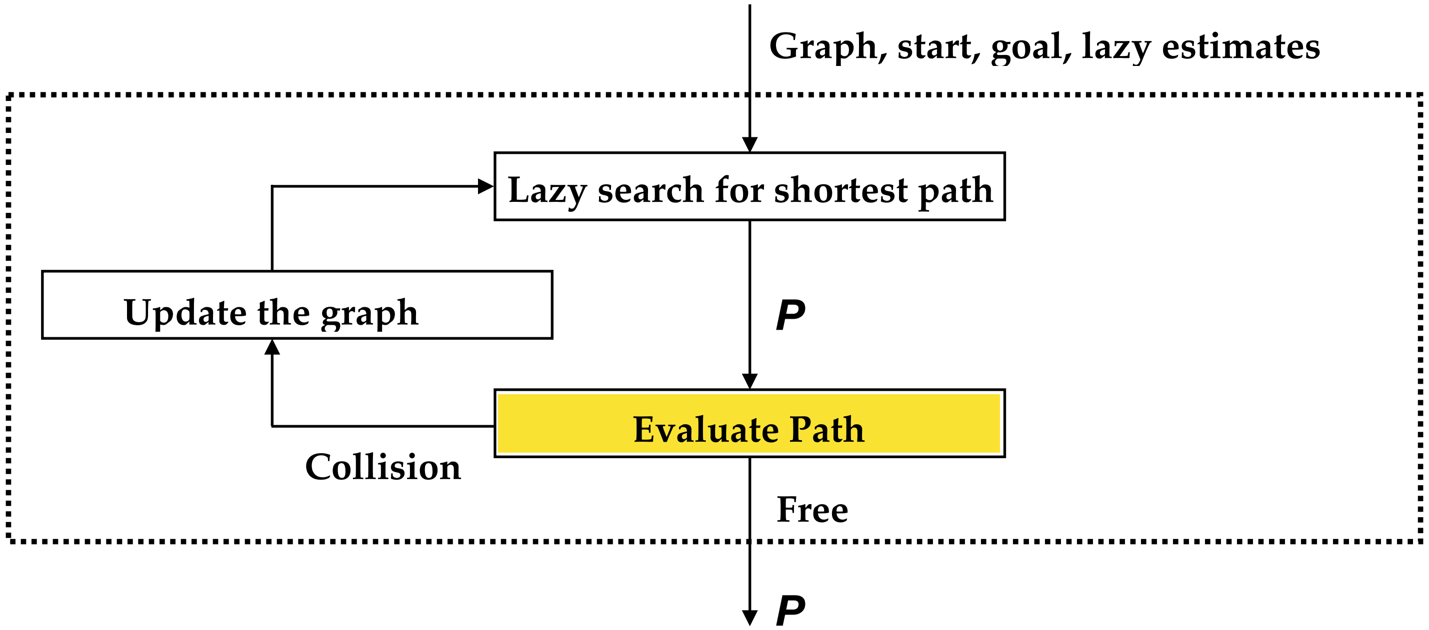
Optimism Under Uncertainty

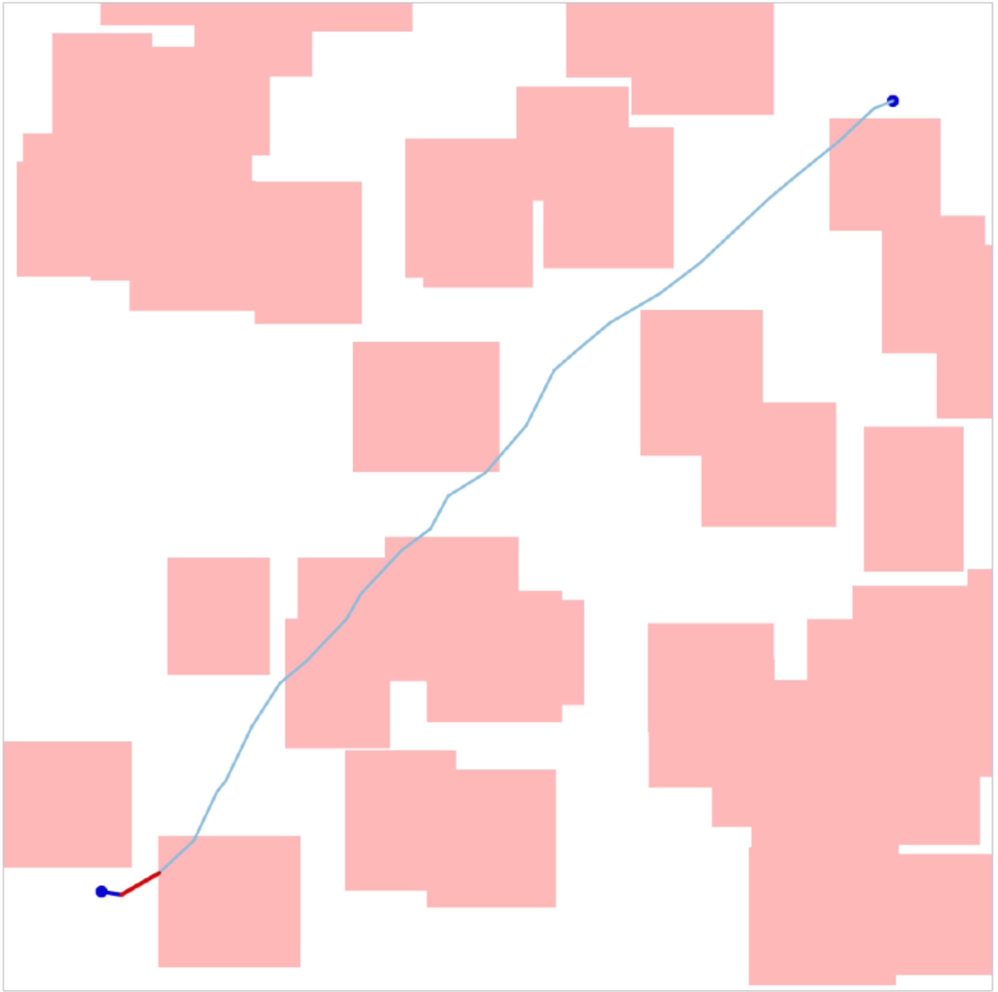




LazySP

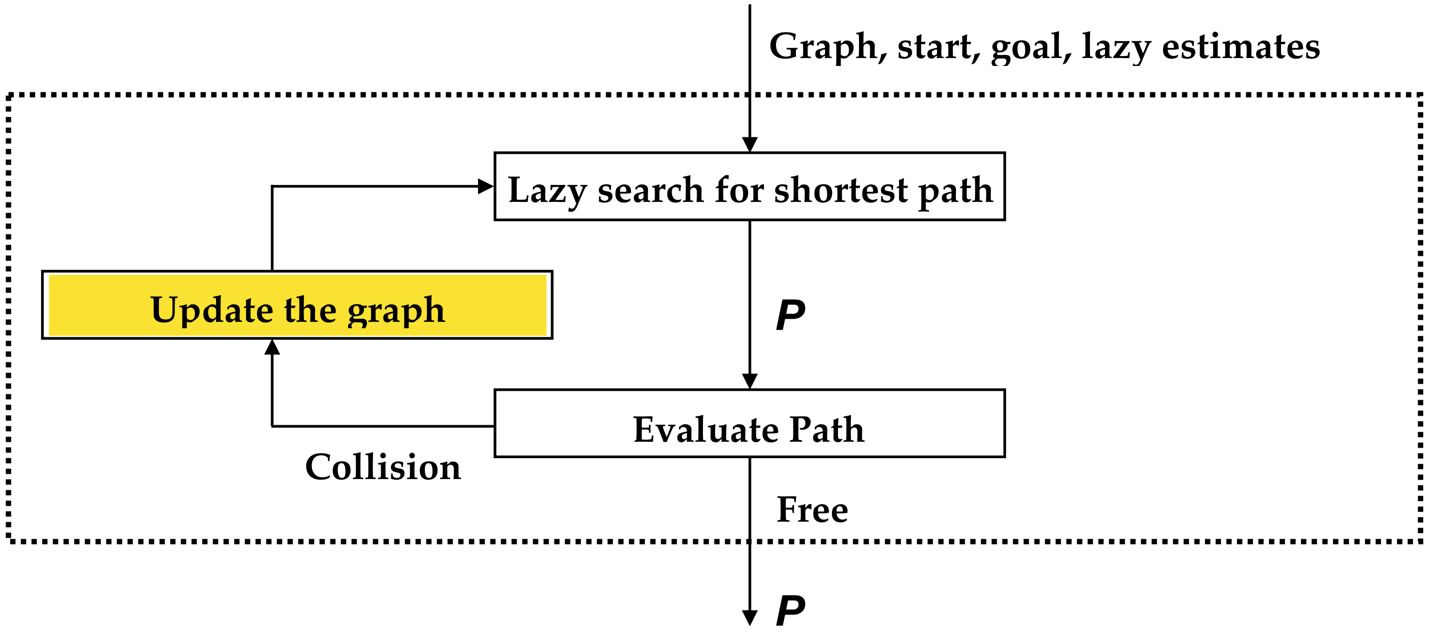
Optimism Under Uncertainty

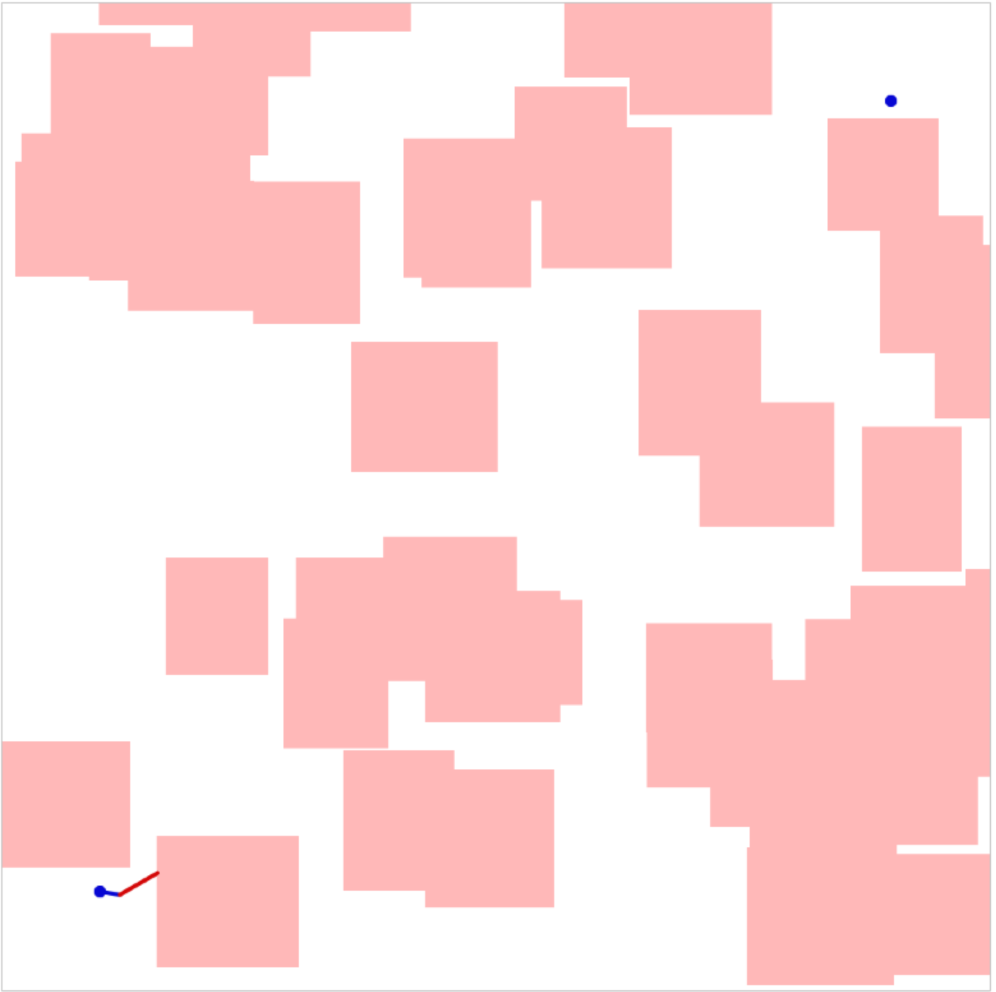




LazySP

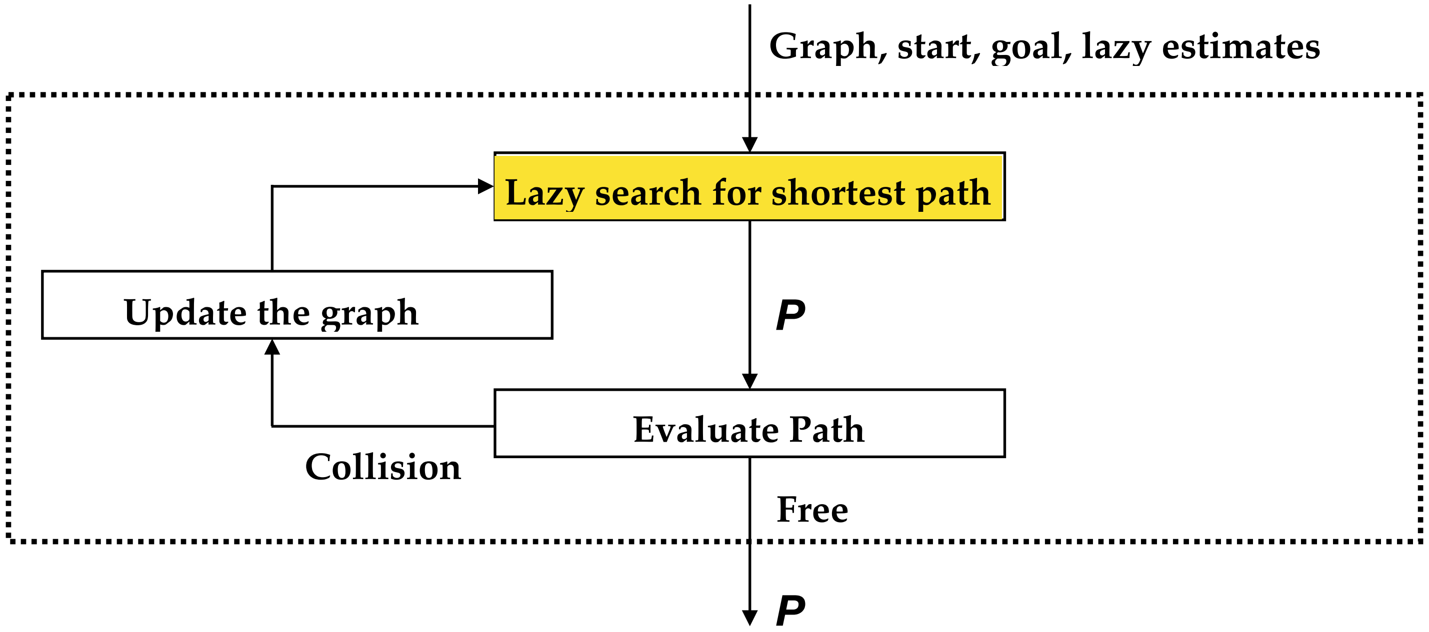
Optimism Under Uncertainty

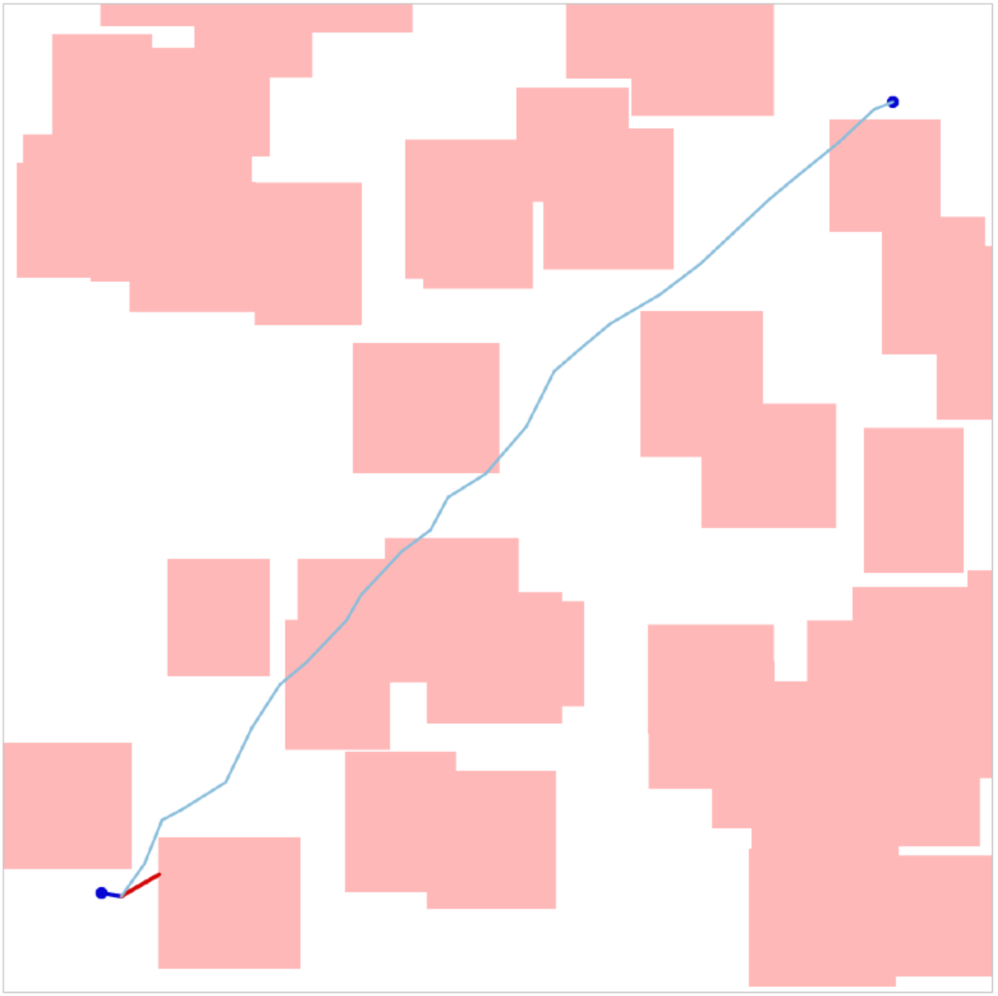




LazySP

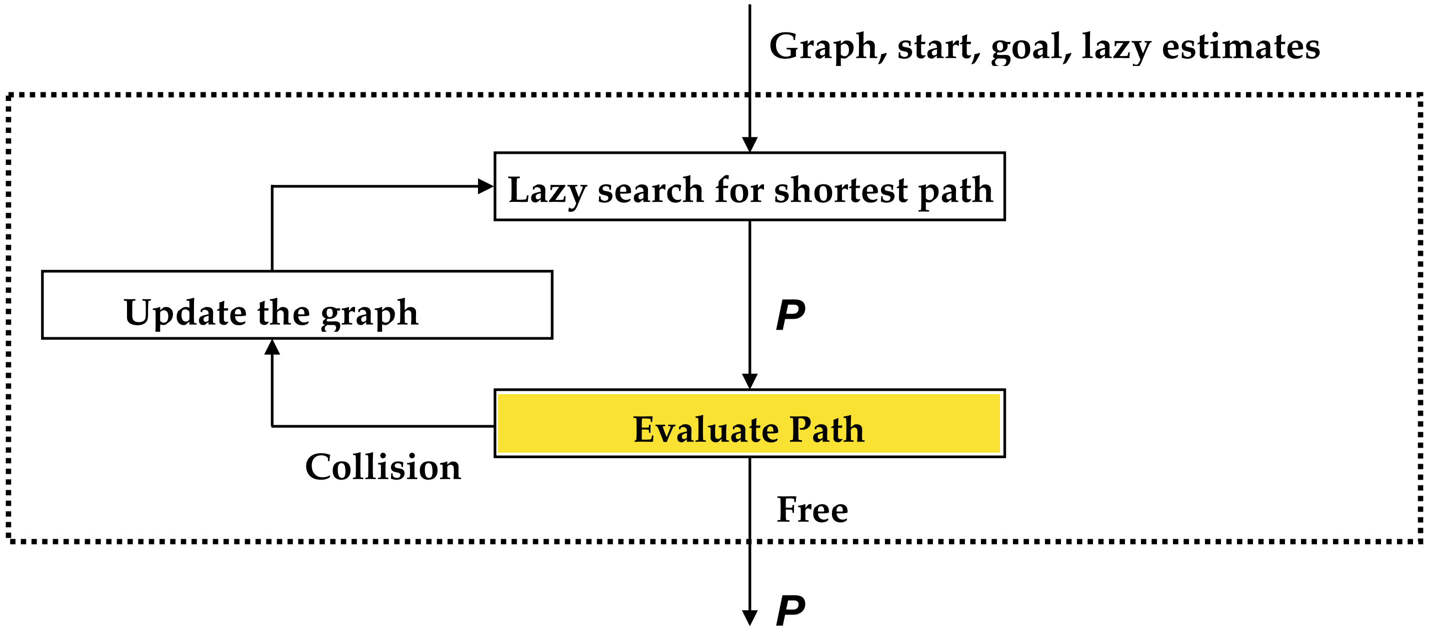
Optimism Under Uncertainty

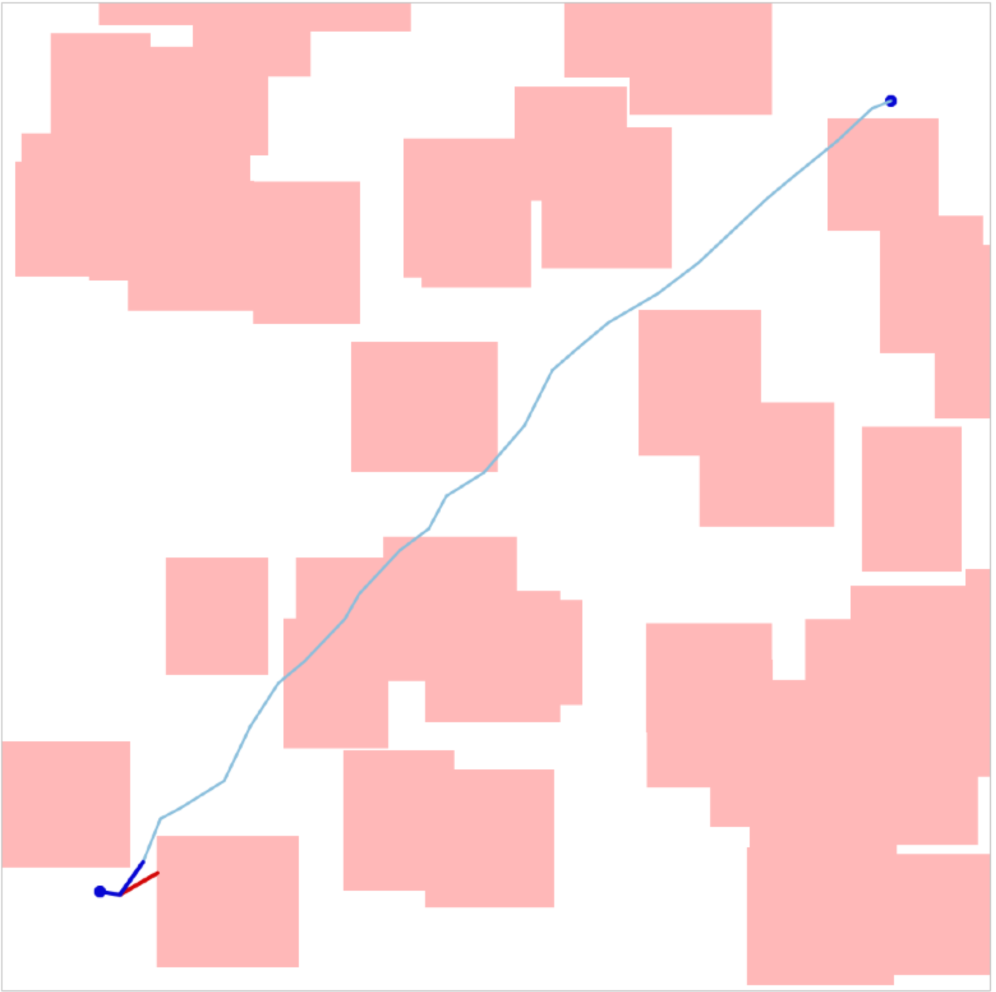




LazySP

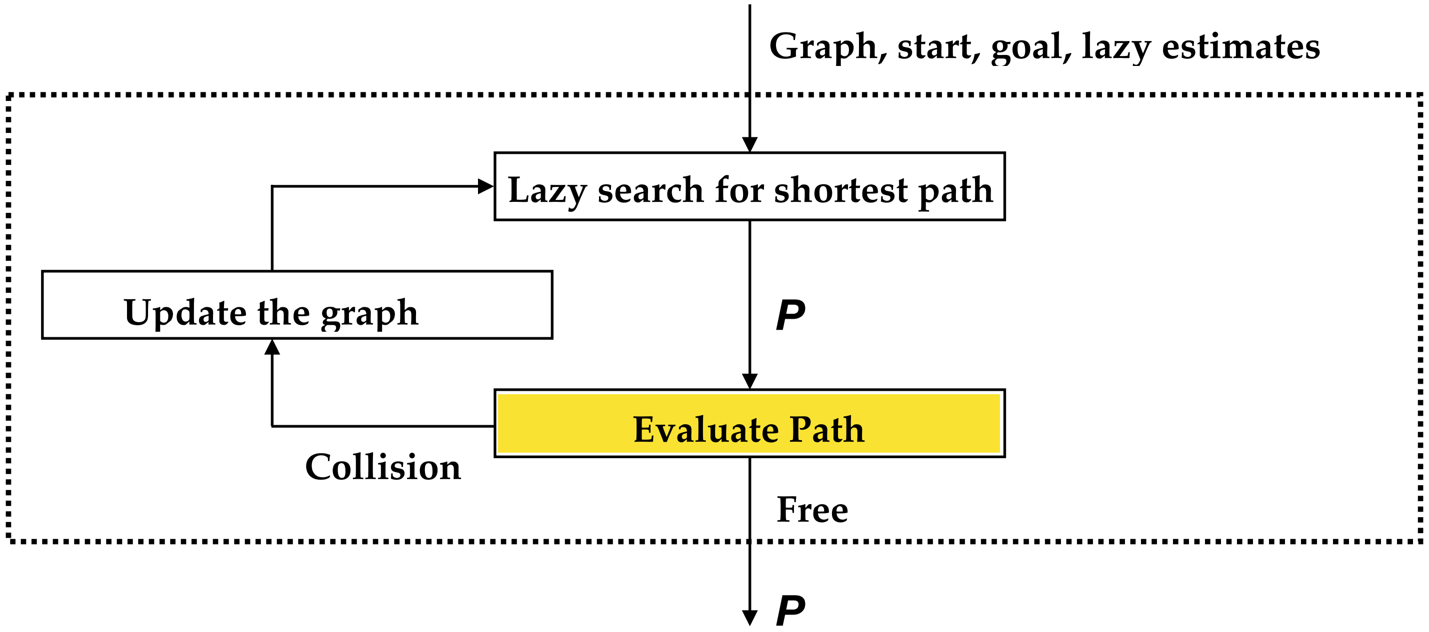
Optimism Under Uncertainty

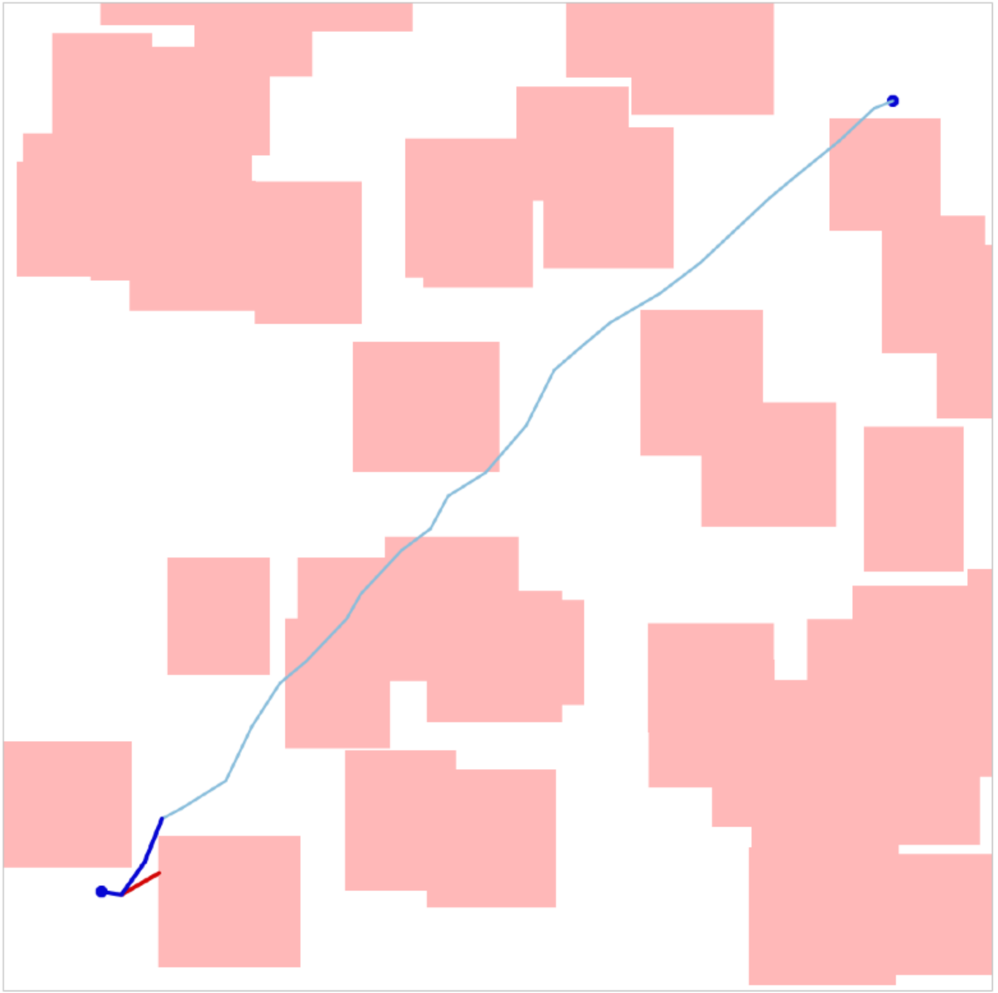




LazySP

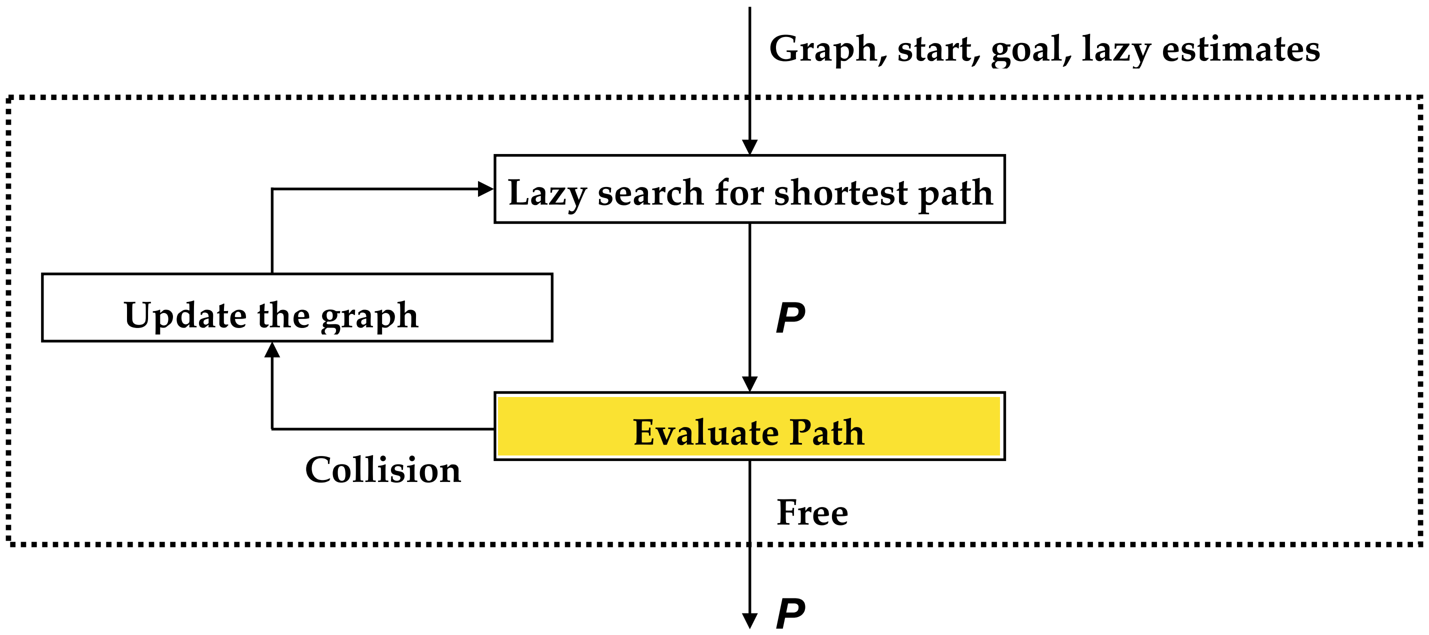
Optimism Under Uncertainty

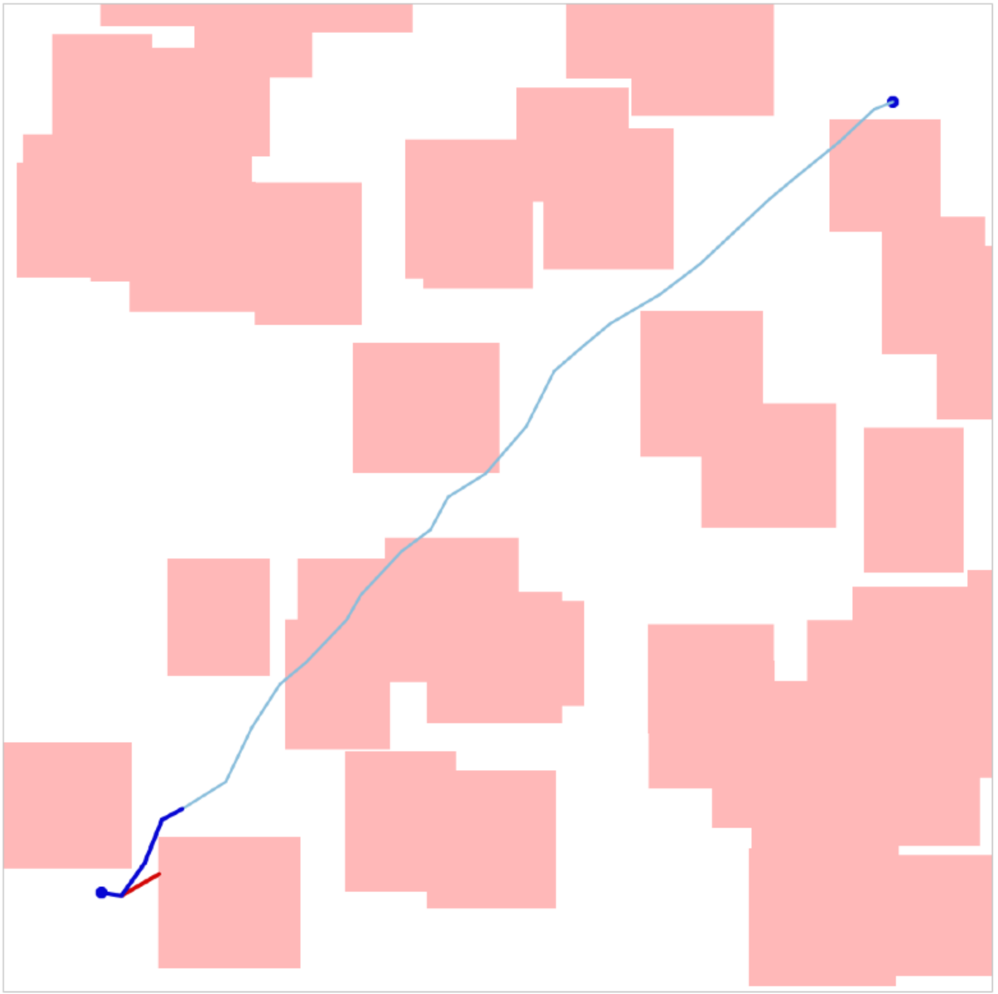




LazySP

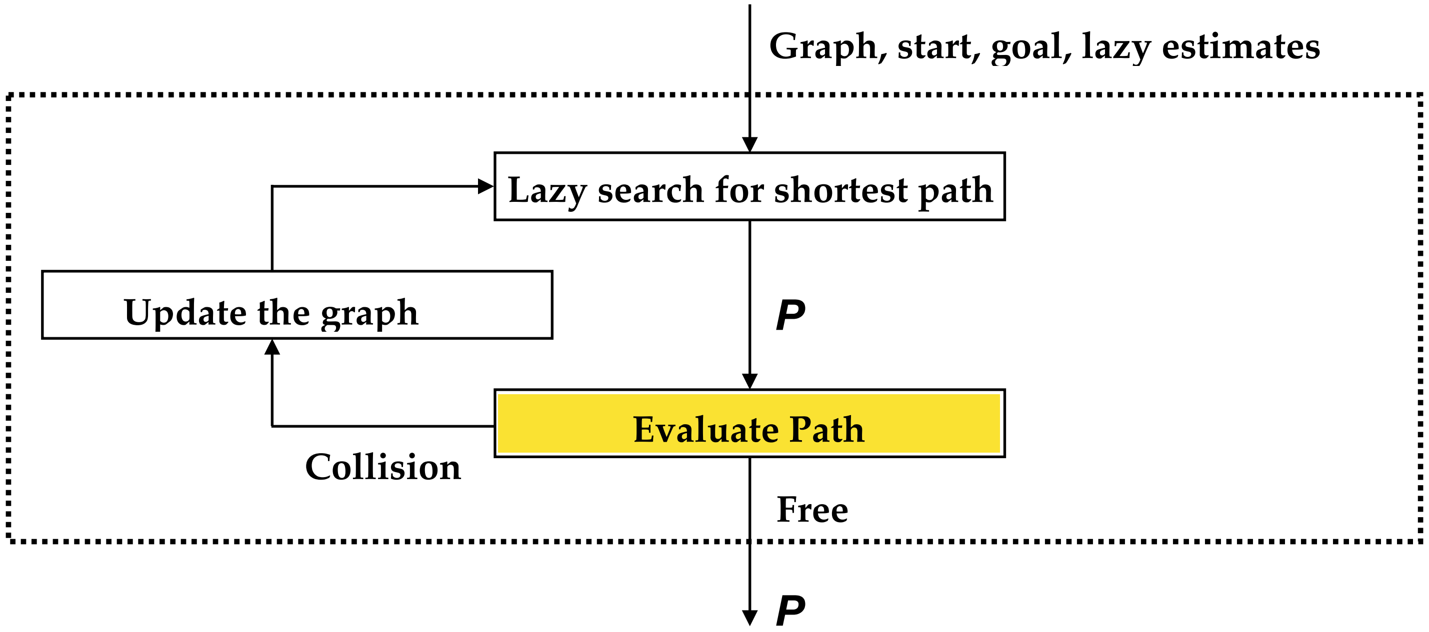
Optimism Under Uncertainty

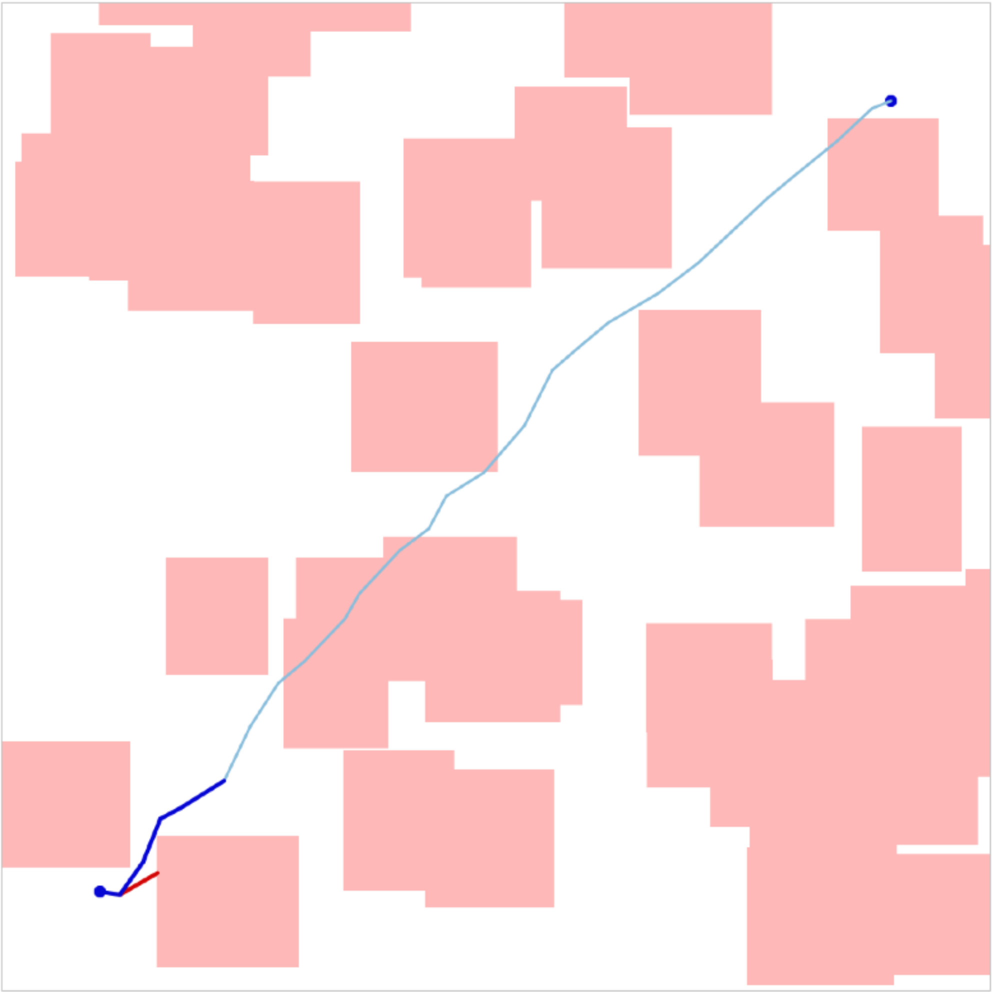




LazySP

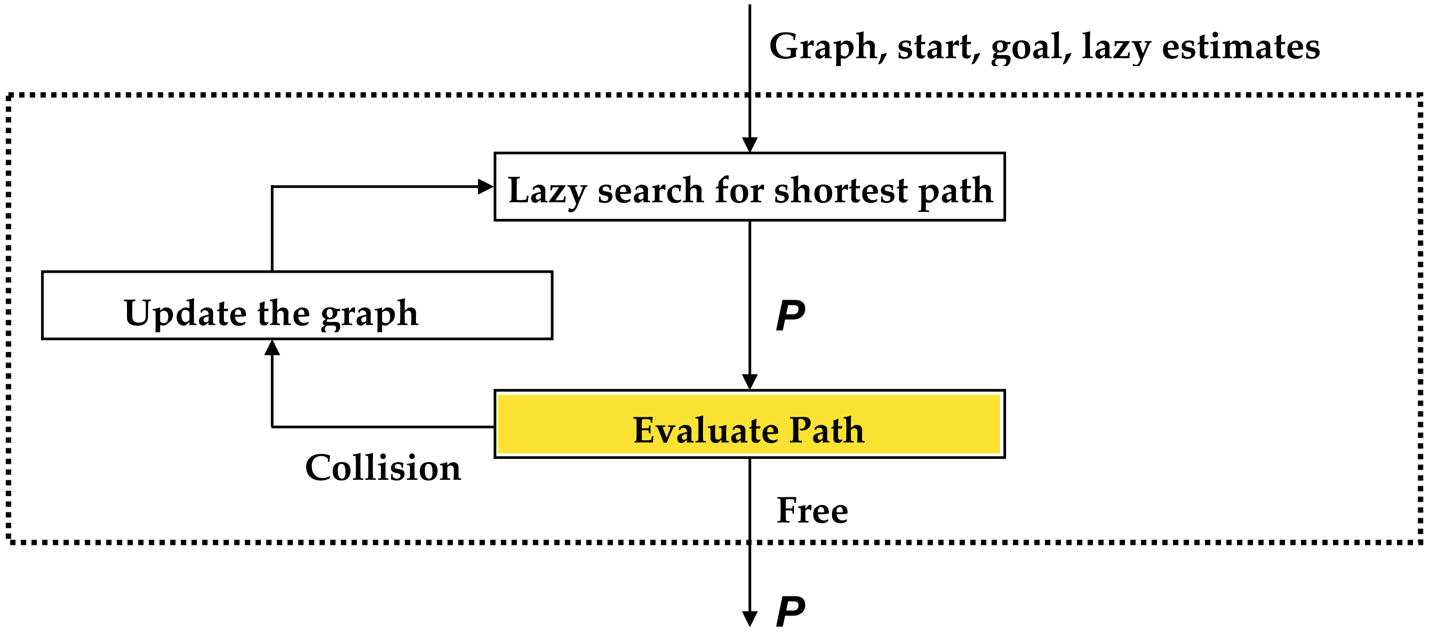
Optimism Under Uncertainty

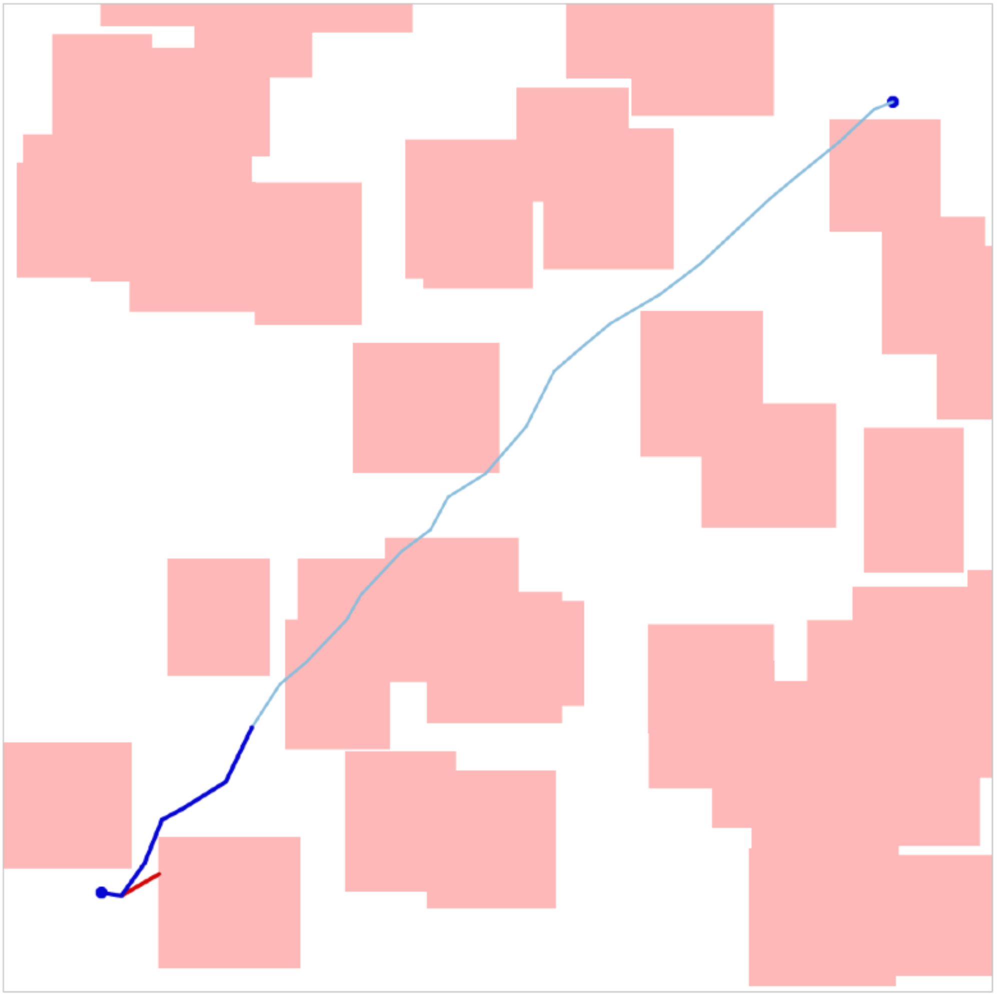




LazySP

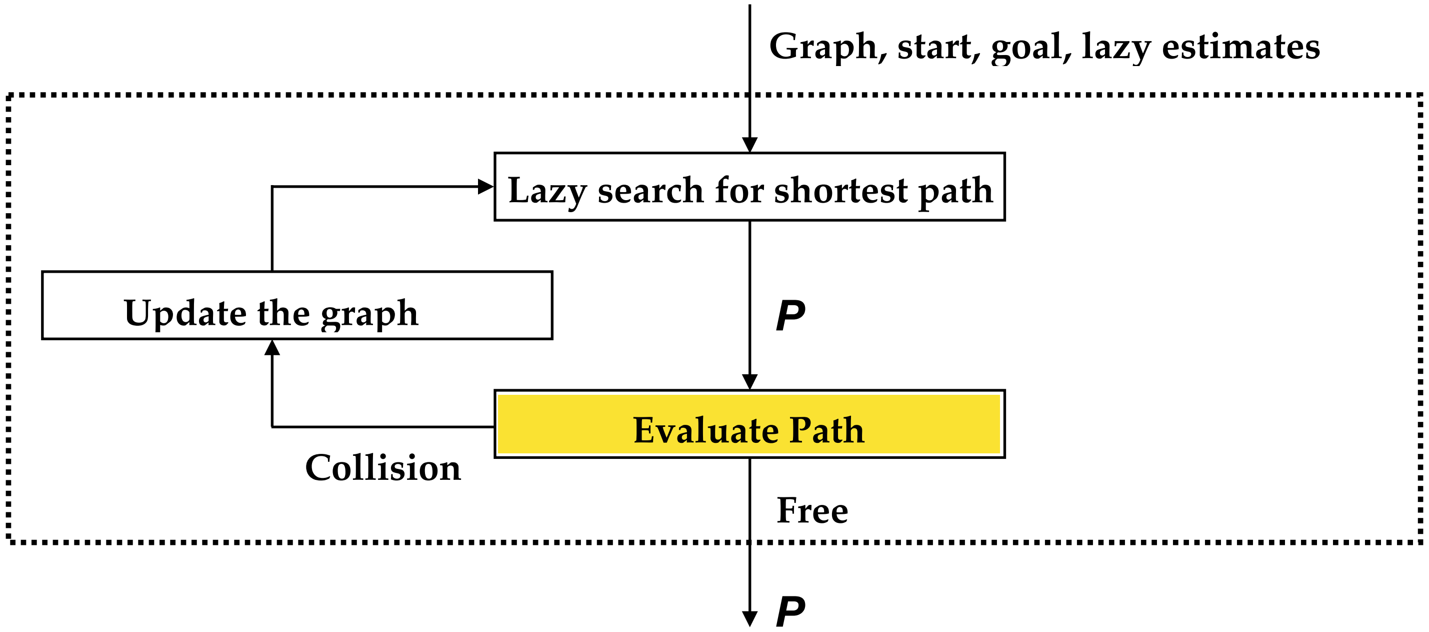
Optimism Under Uncertainty

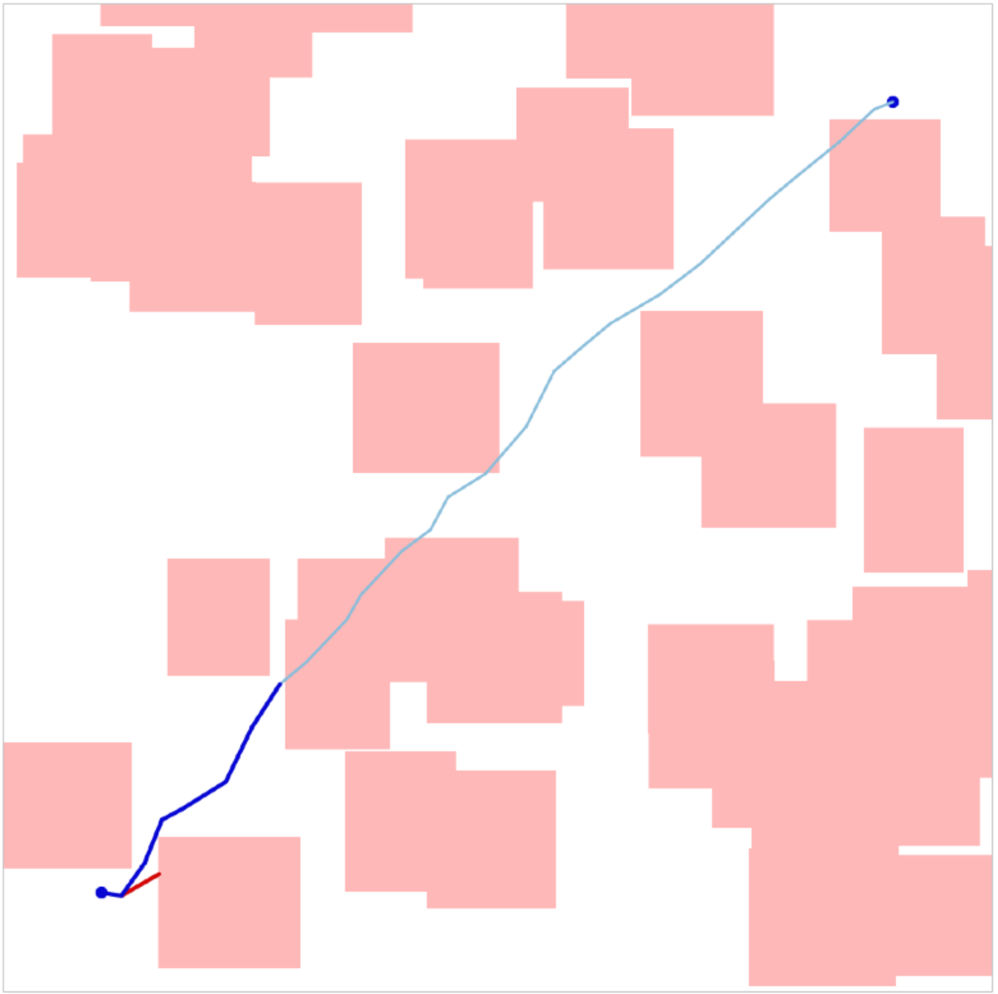




LazySP

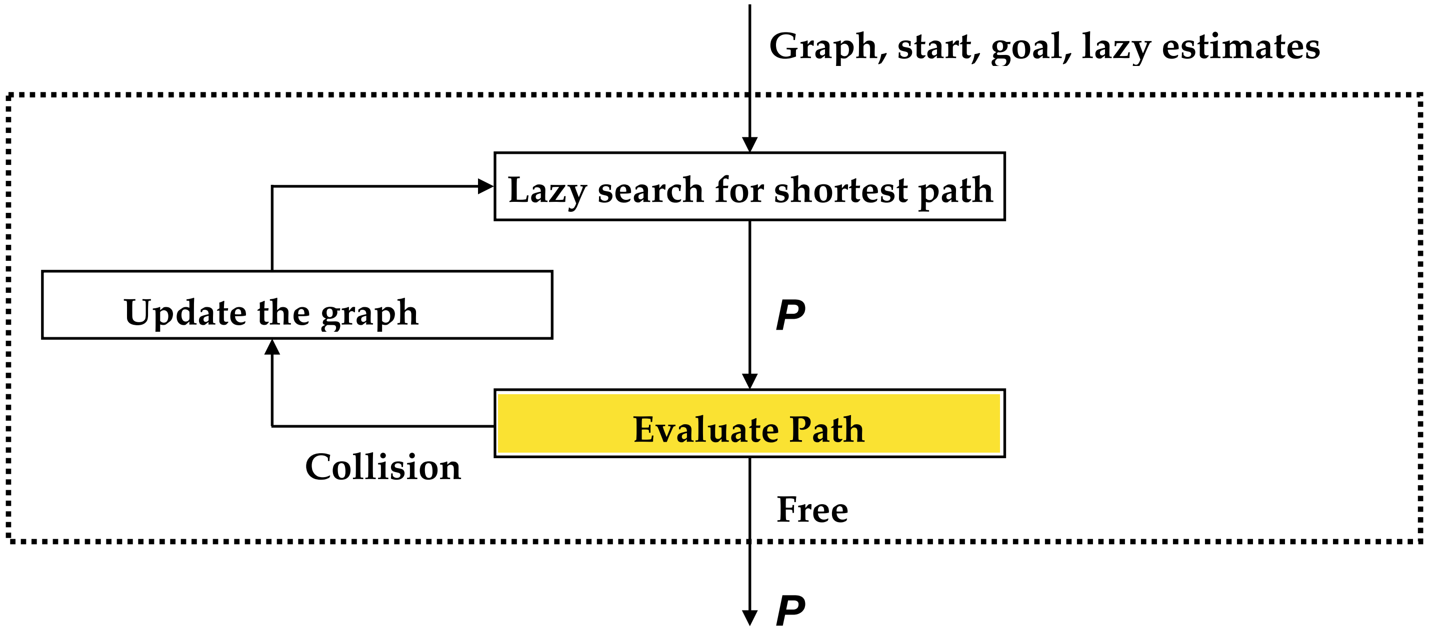
Optimism Under Uncertainty

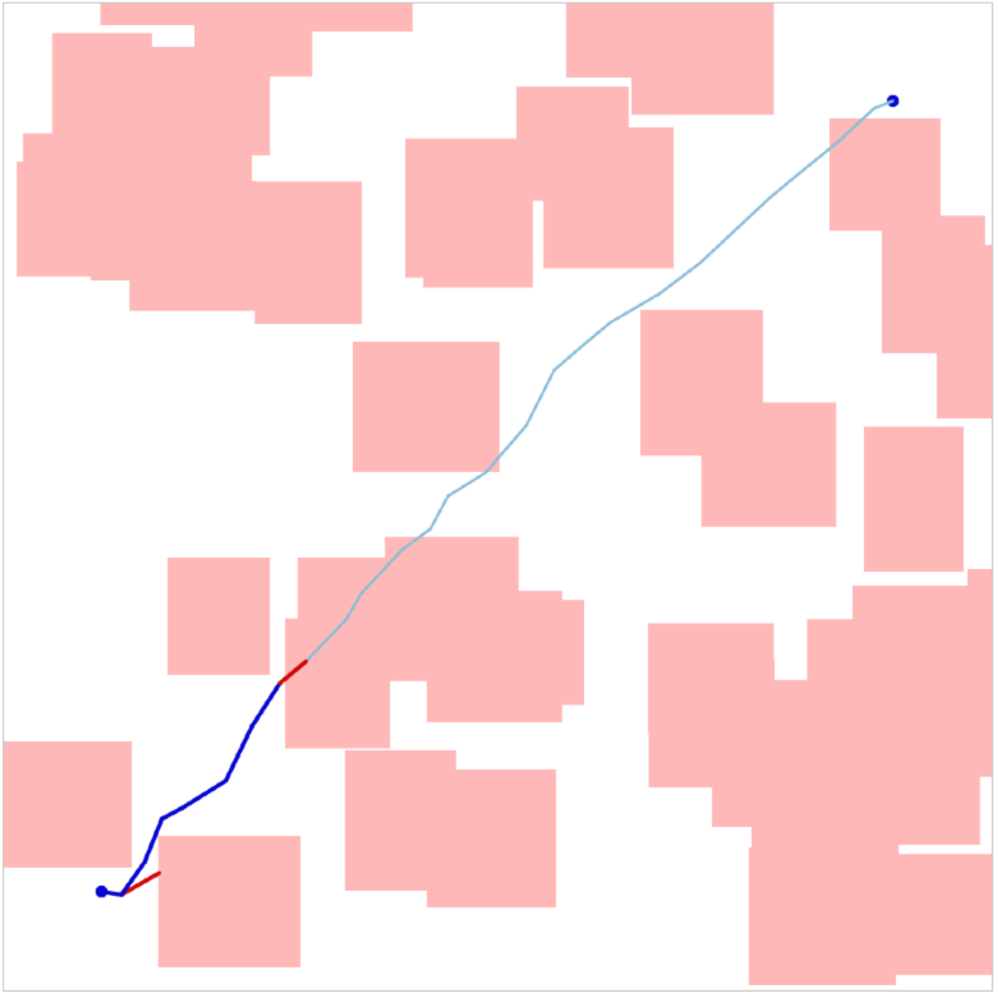




LazySP

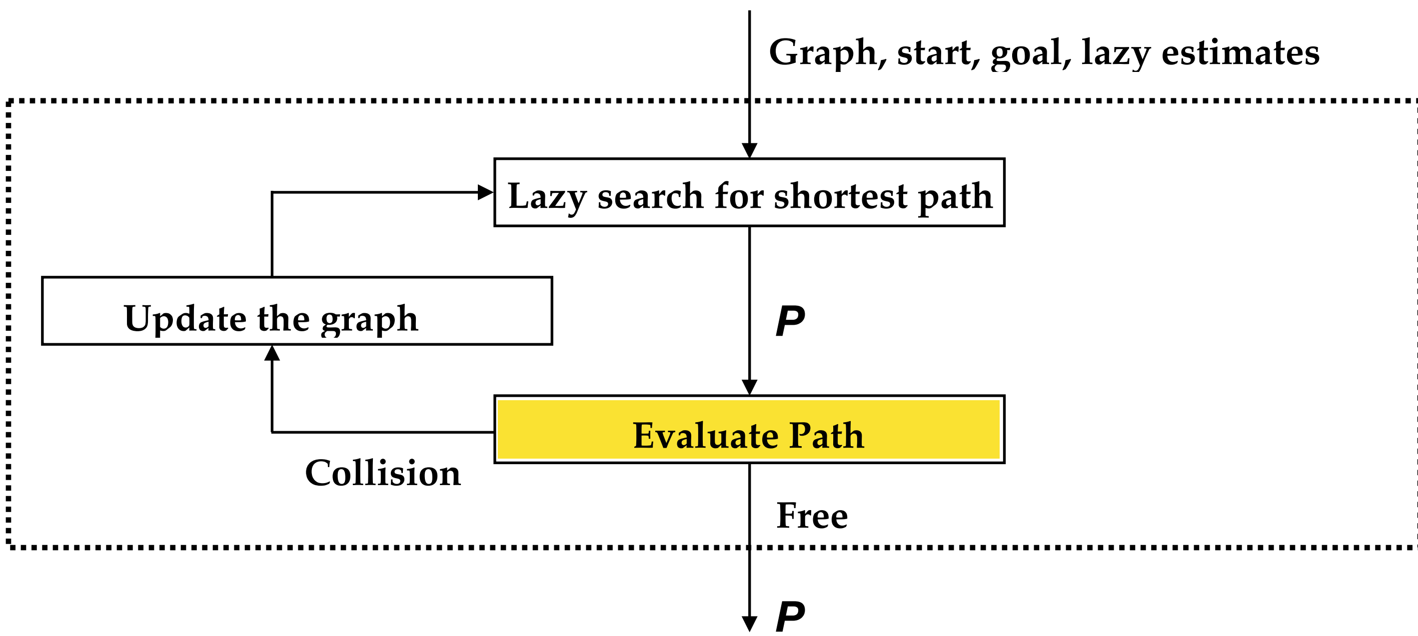
Optimism Under Uncertainty

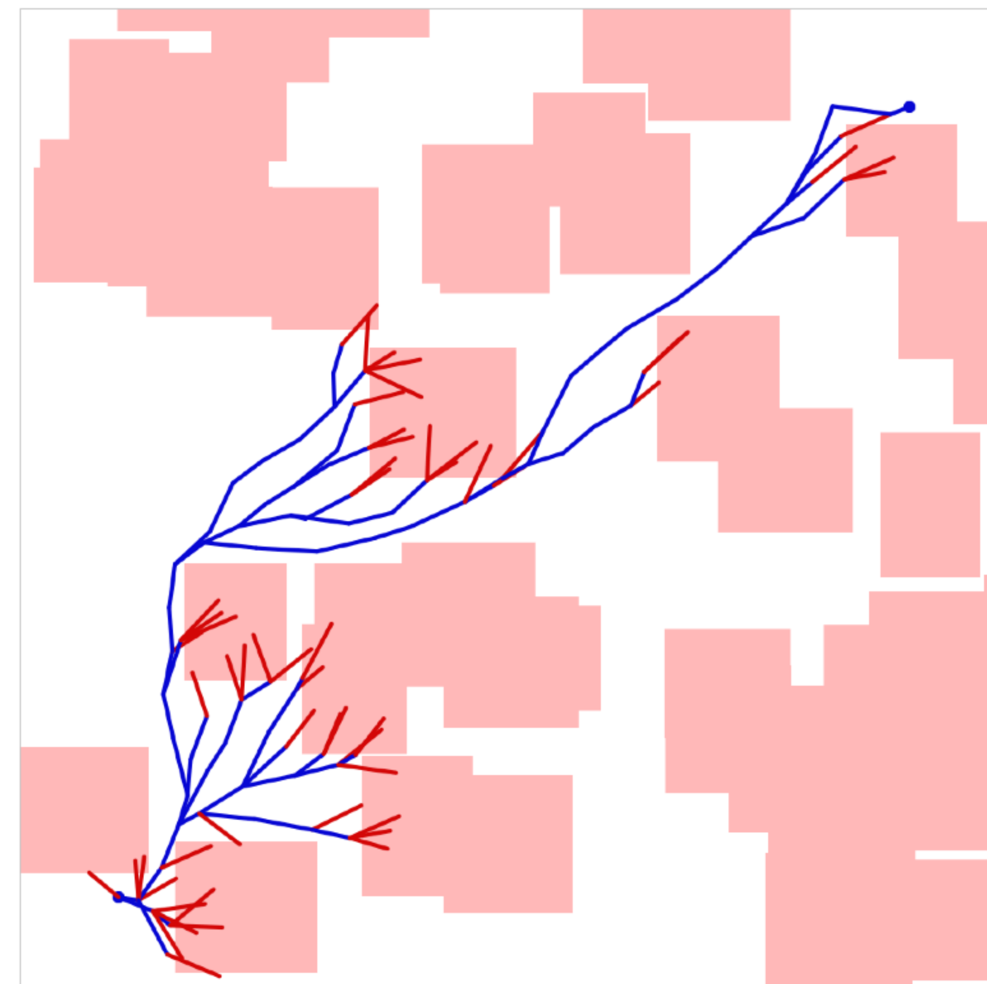




LazySP

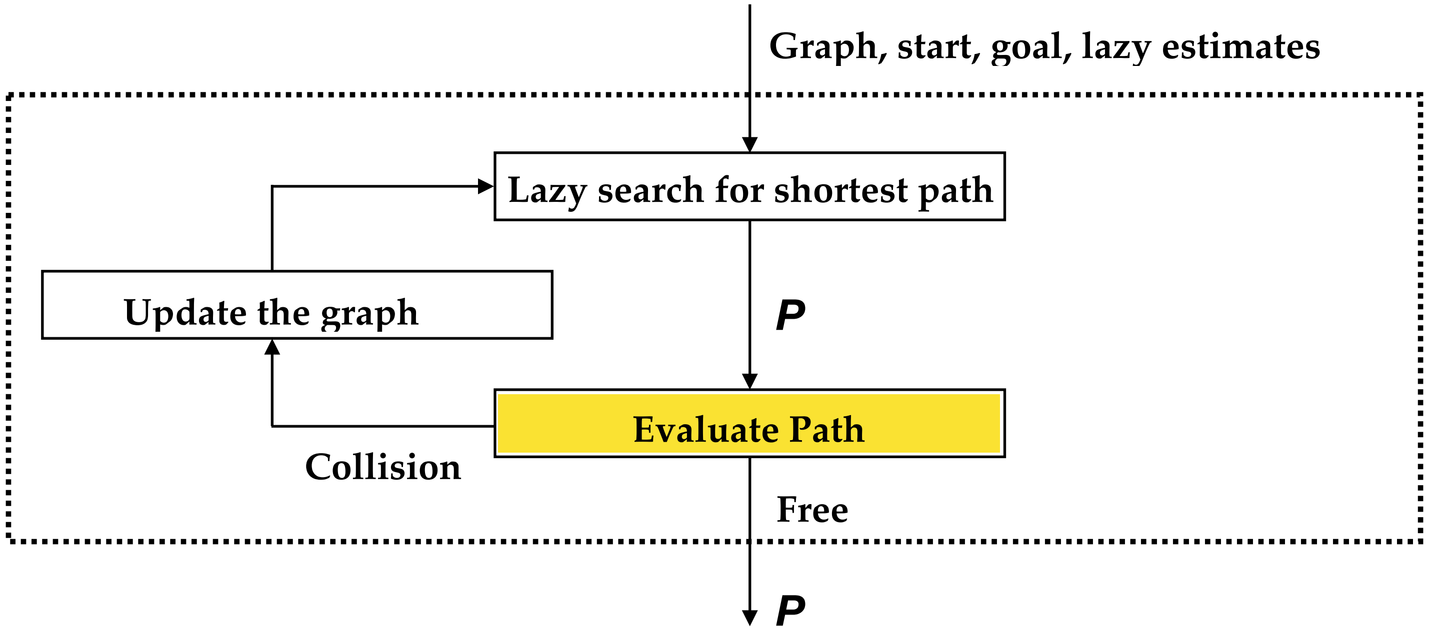
Optimism Under Uncertainty

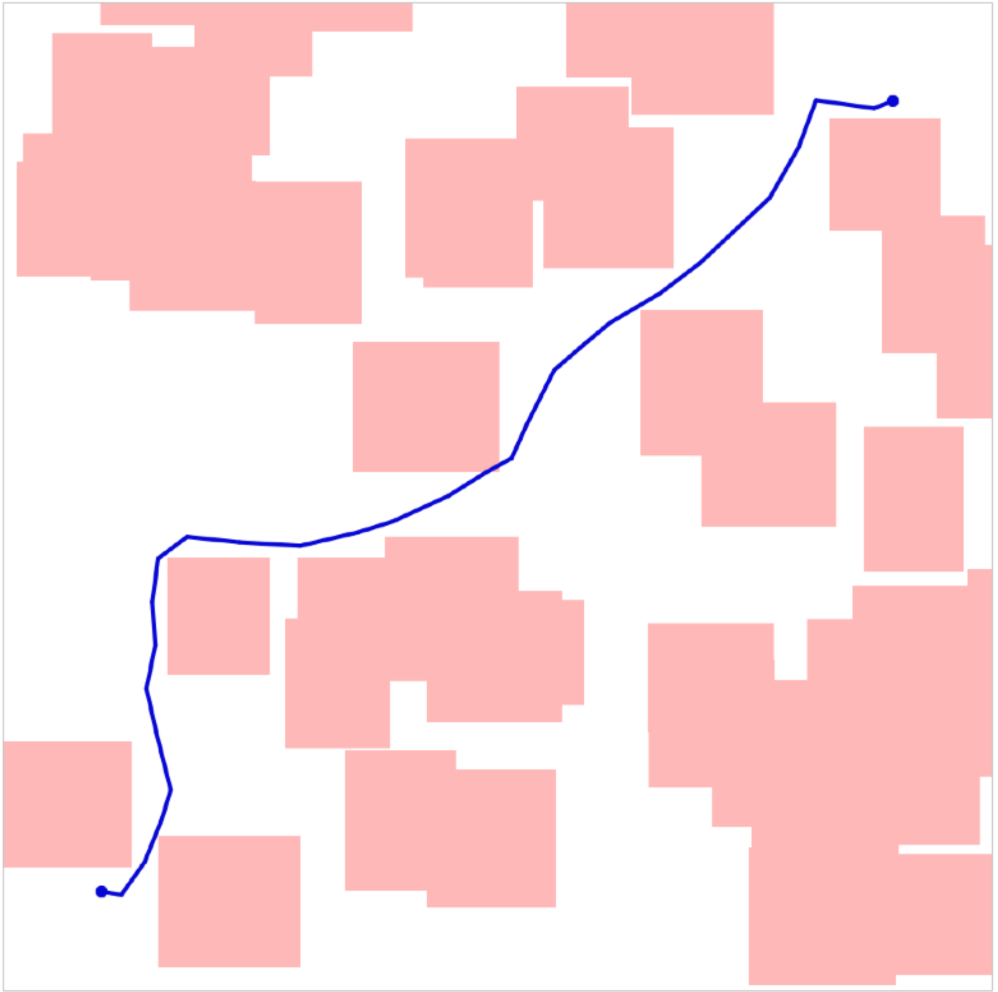




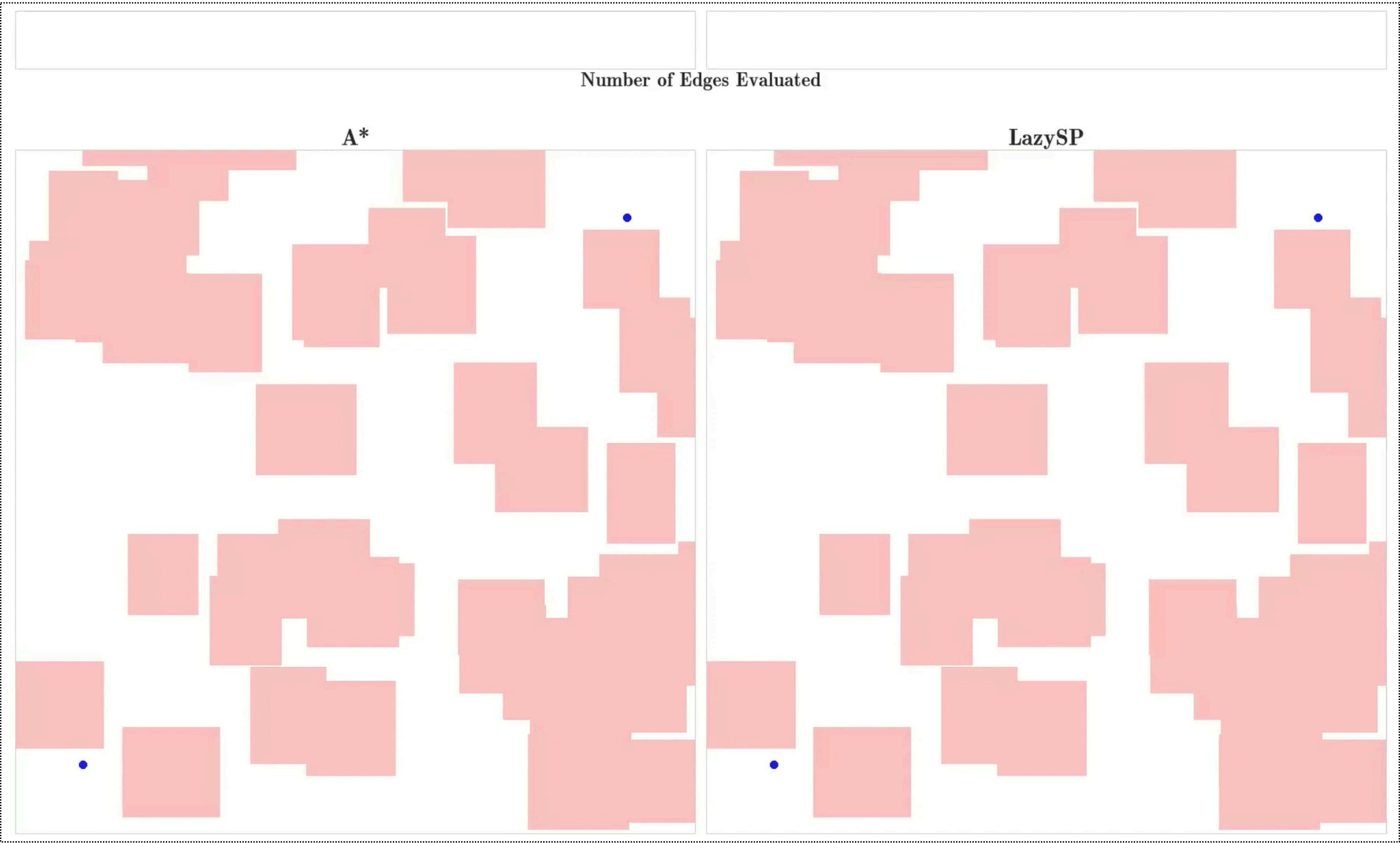
LazySP

Optimism Under Uncertainty



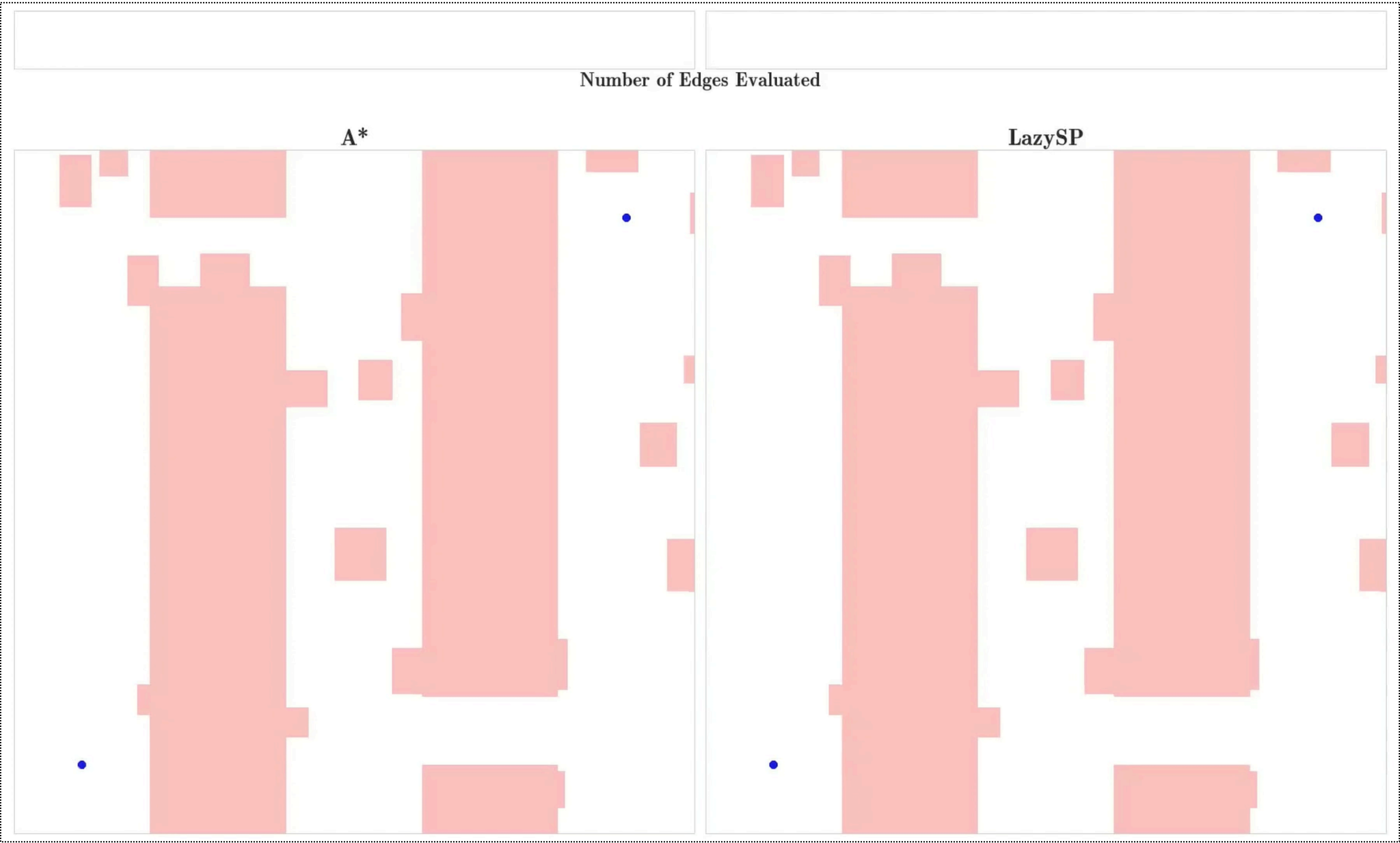


Comparison across environments

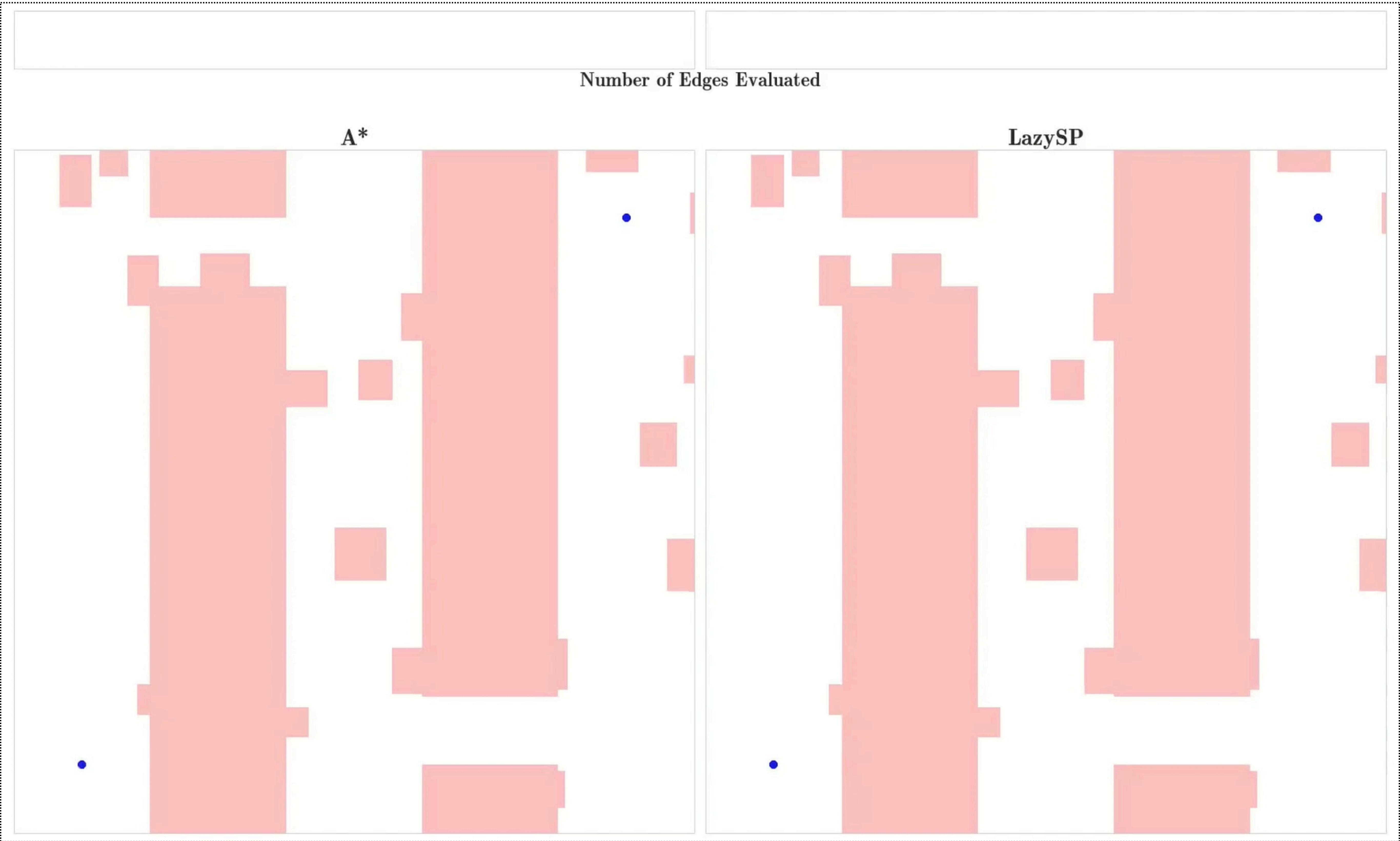


Comparison across environments

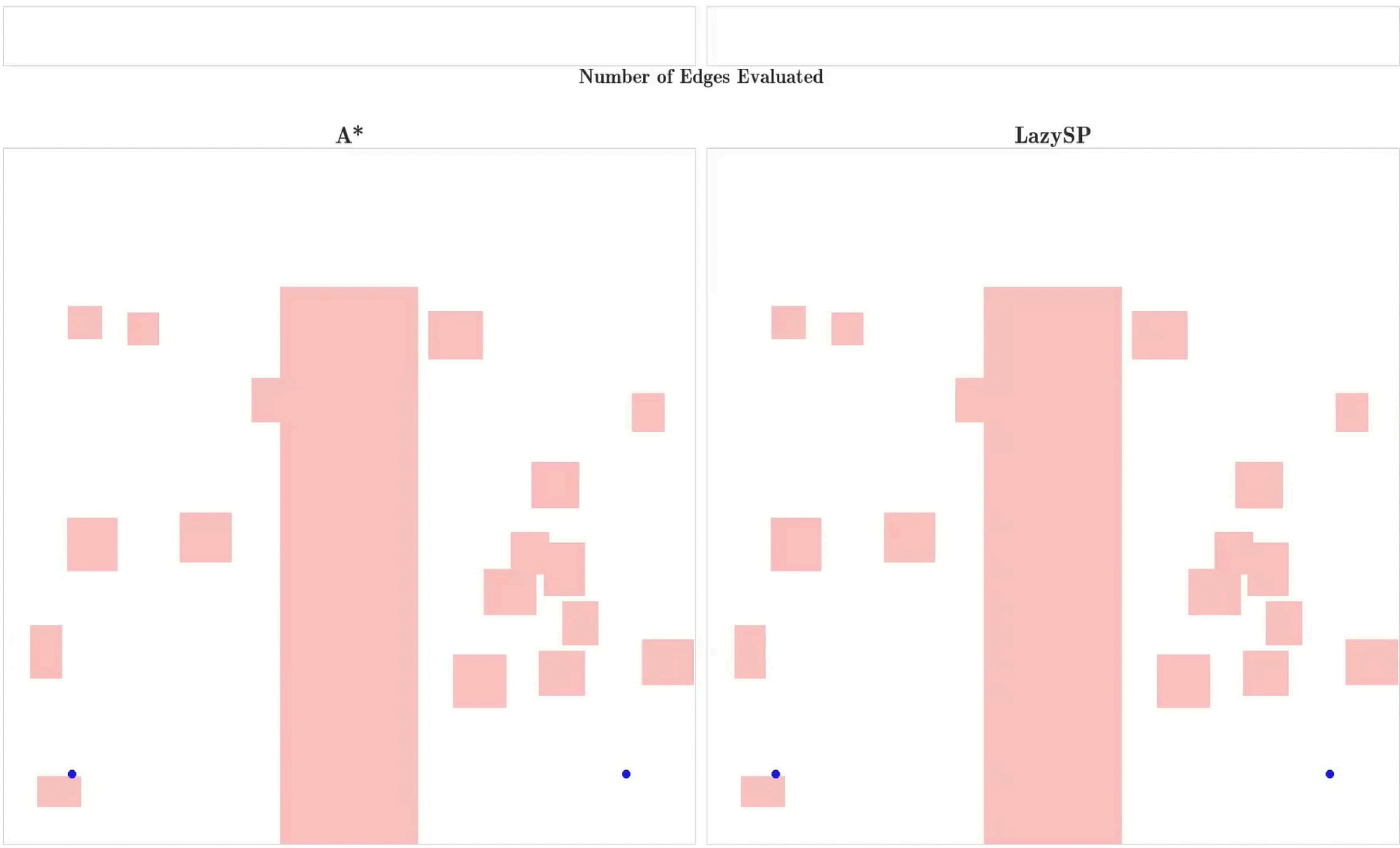
Comparison across environments



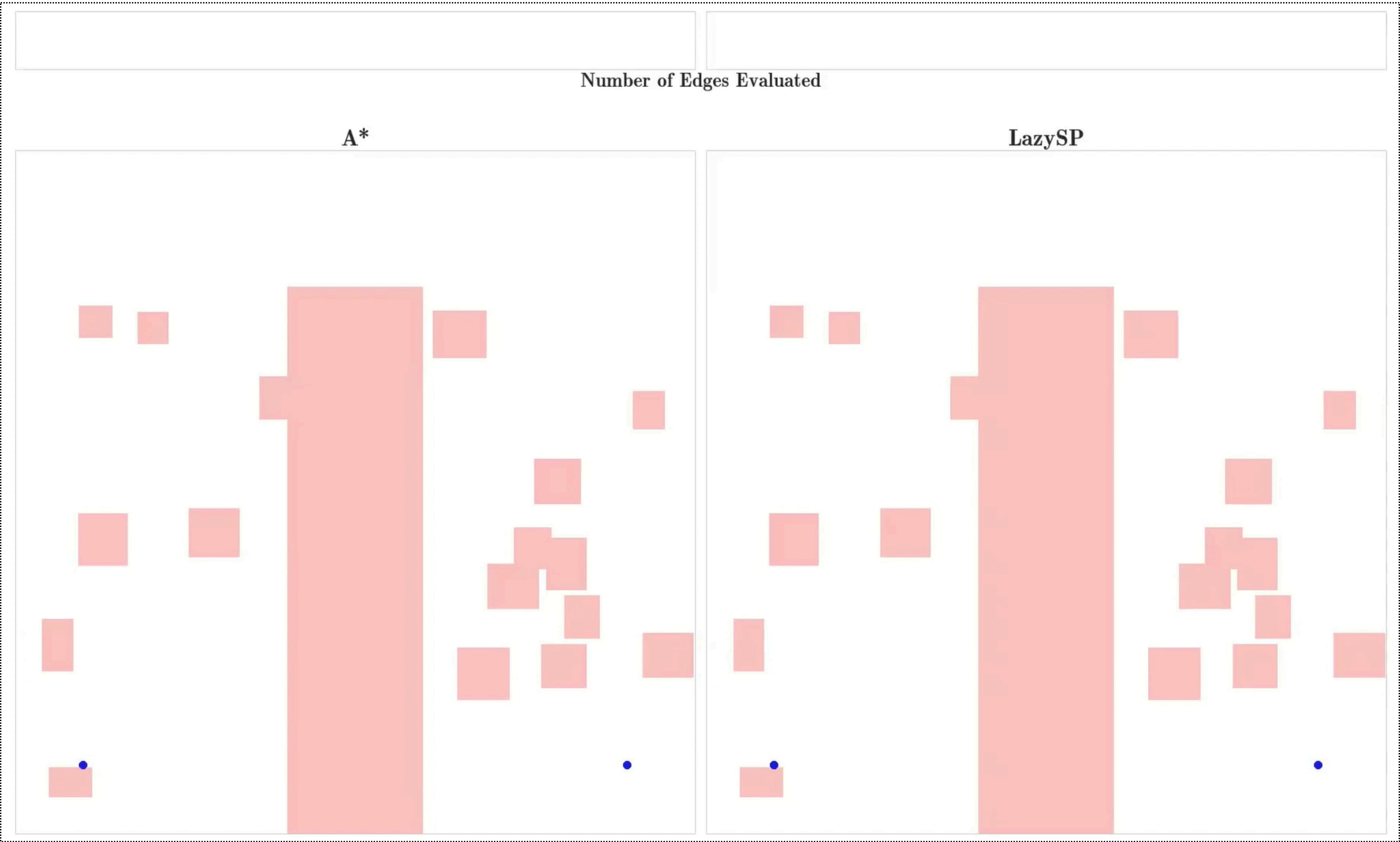
Comparison across environments



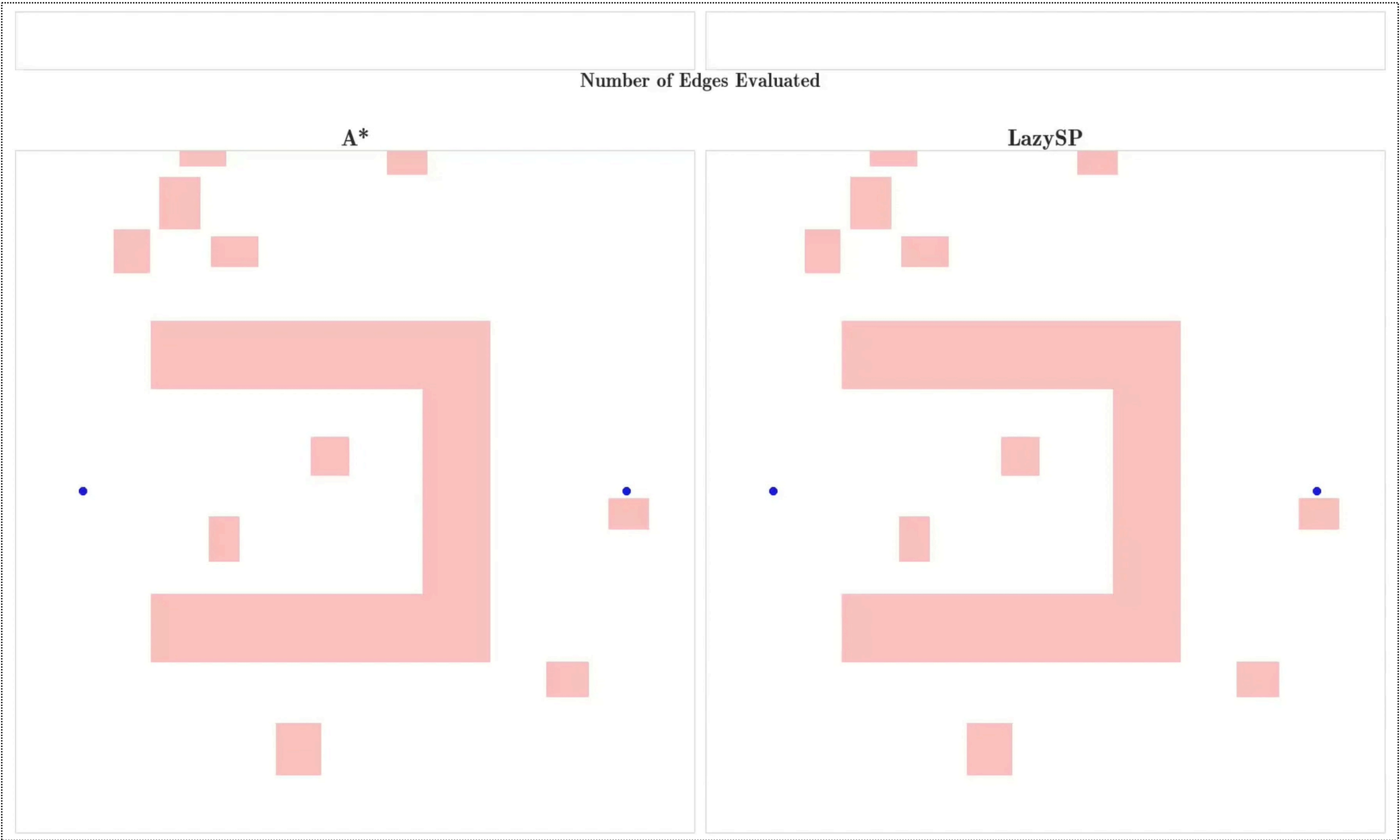
Comparison across environments



Comparison across environments



Comparison across environments



Comparison across environments