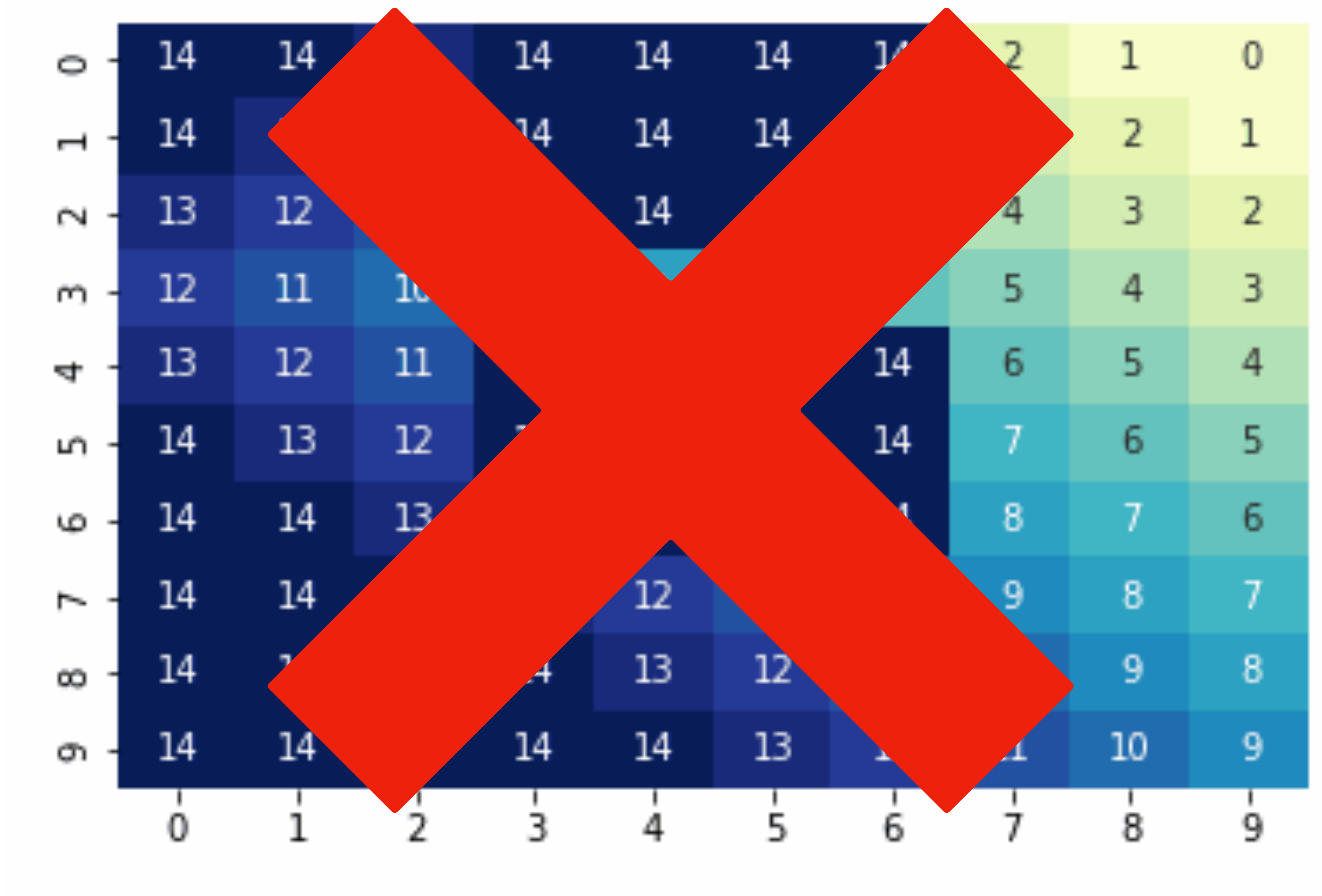
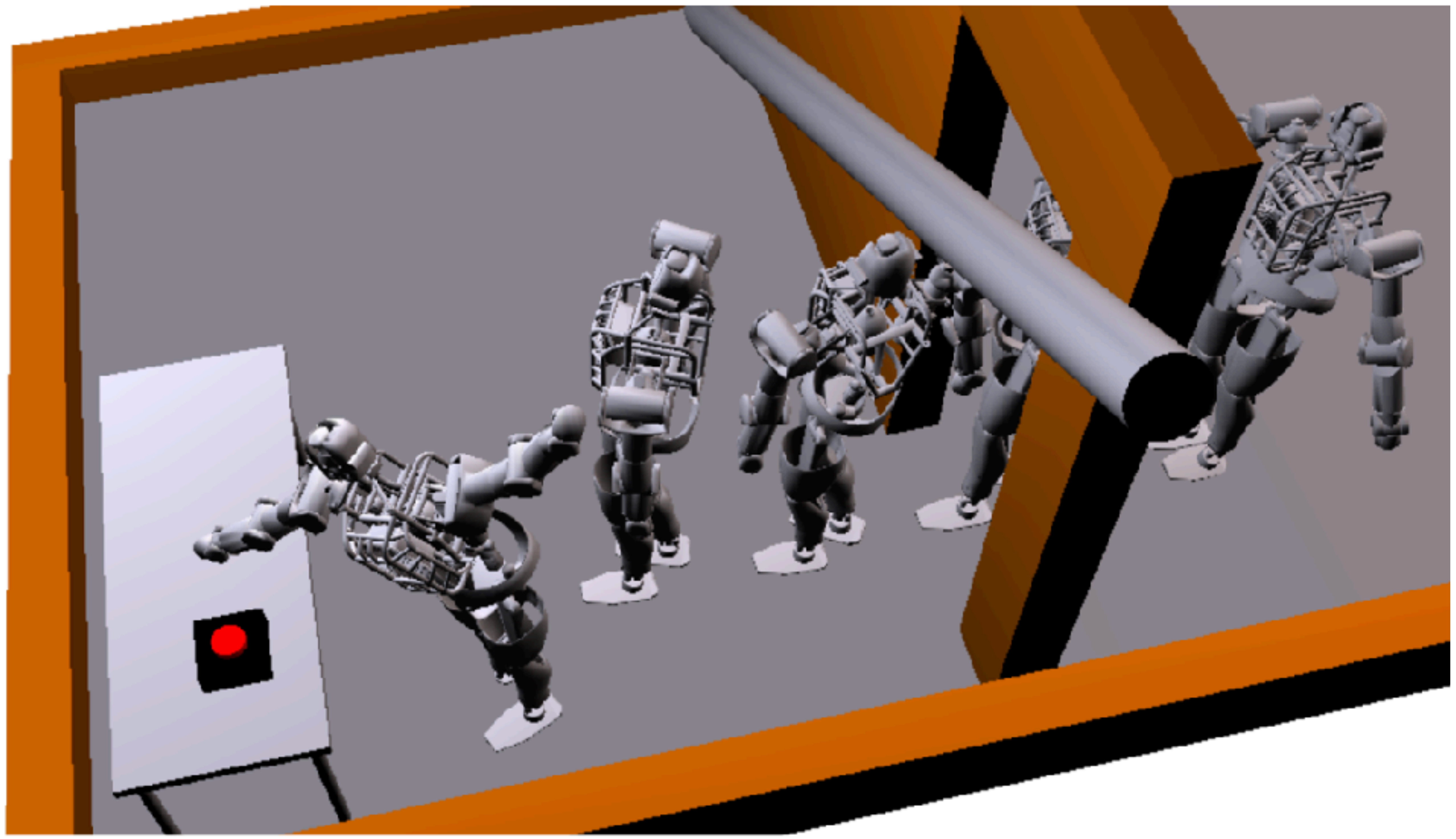
Conquering Motion Planning via Sampling and Search

Sanjiban Choudhury

Cornell Bowers C·IS Computer Science

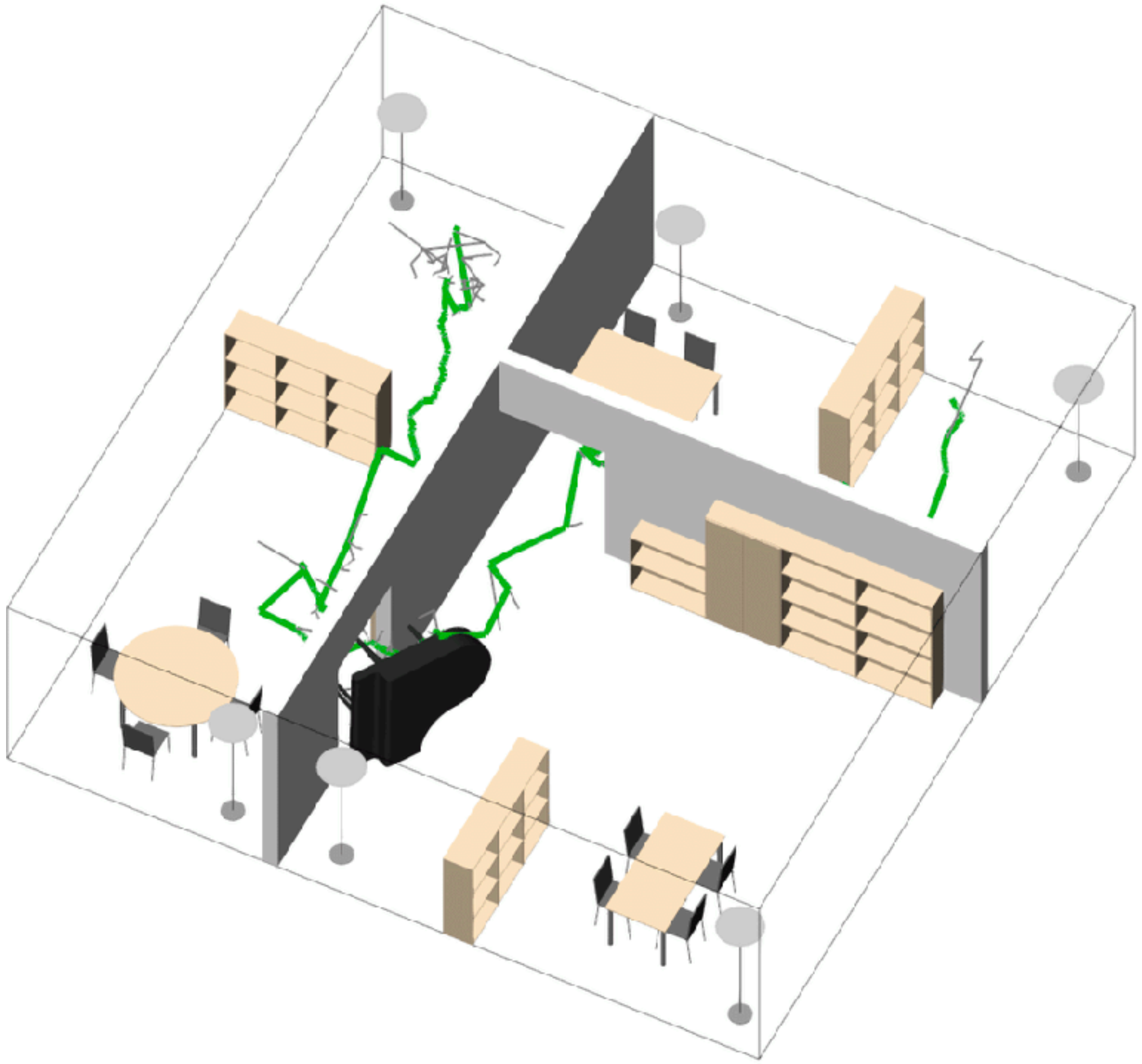
The Real World is not Tabular!



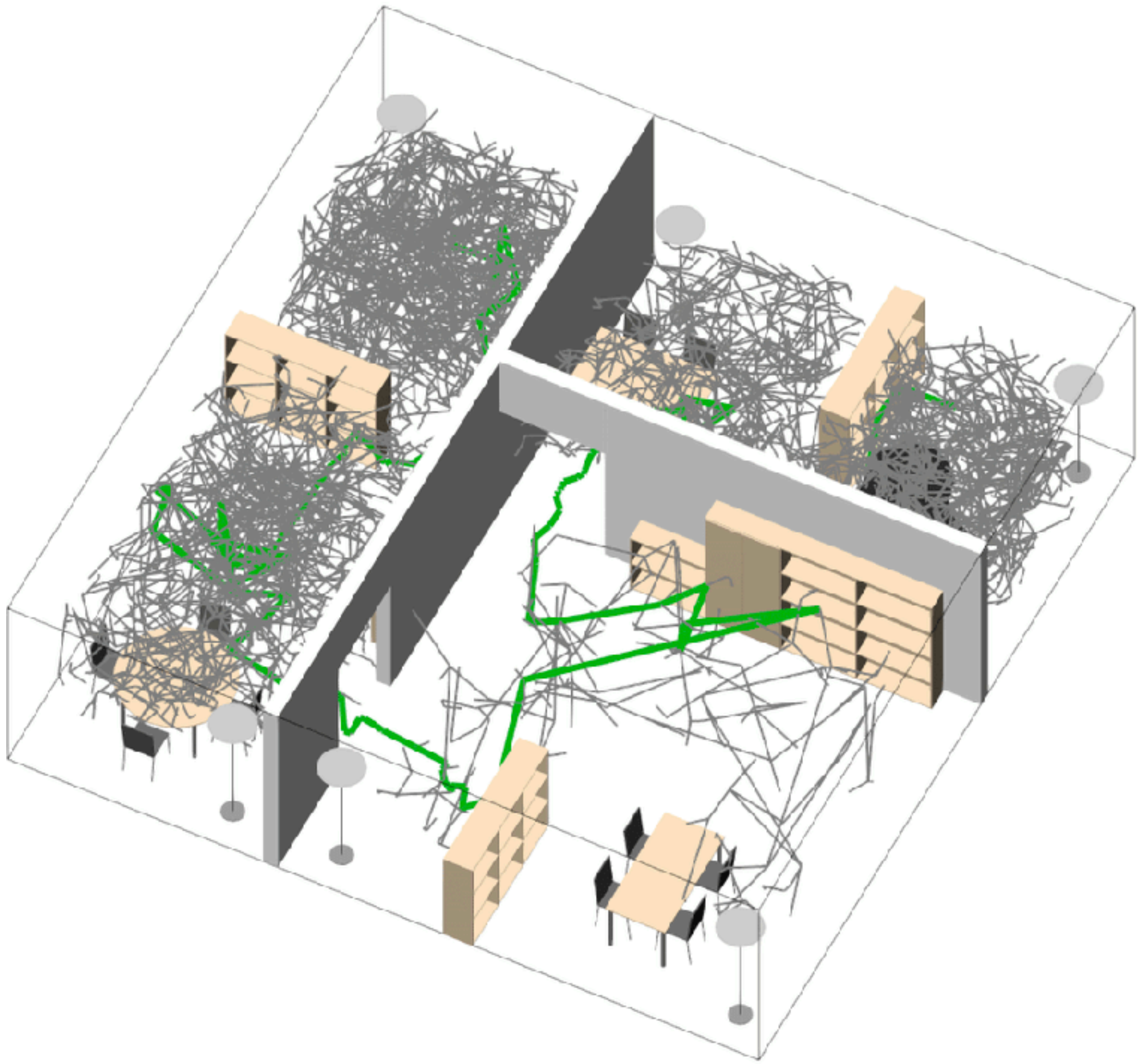


Why is robot motion planning hard?

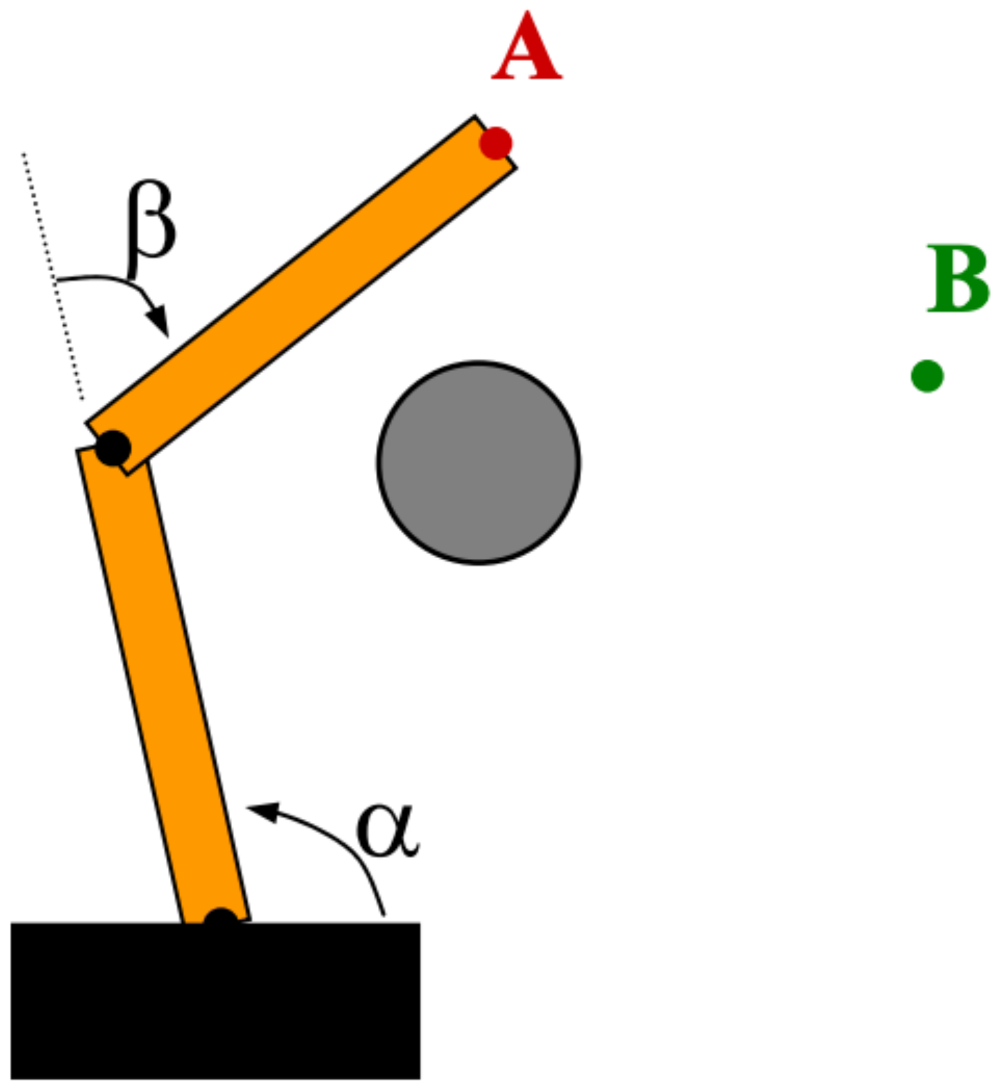
Challenge 1: Continuous



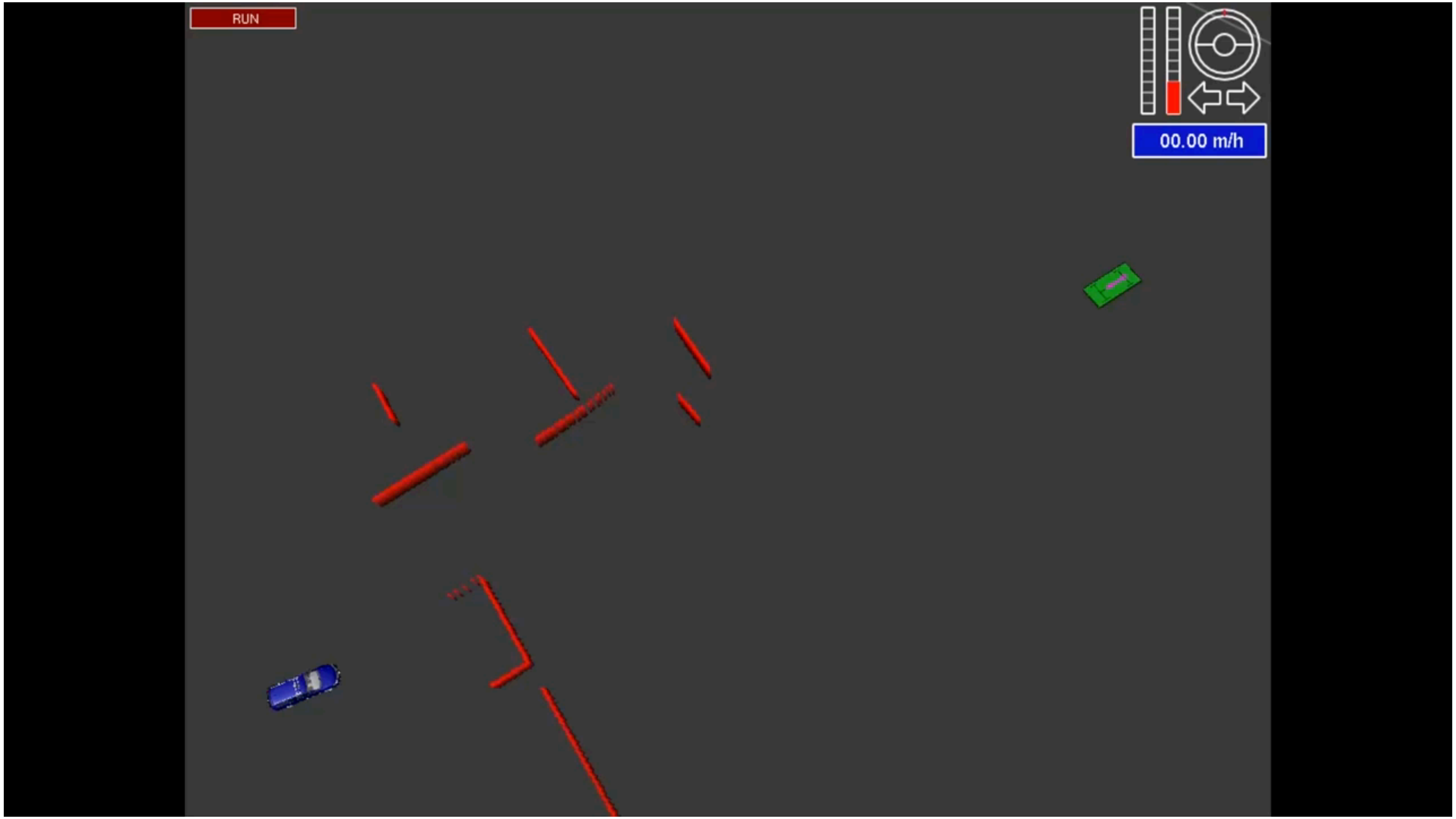
Challenge 1: Continuous



Challenge 2: Configuration Space Geometry



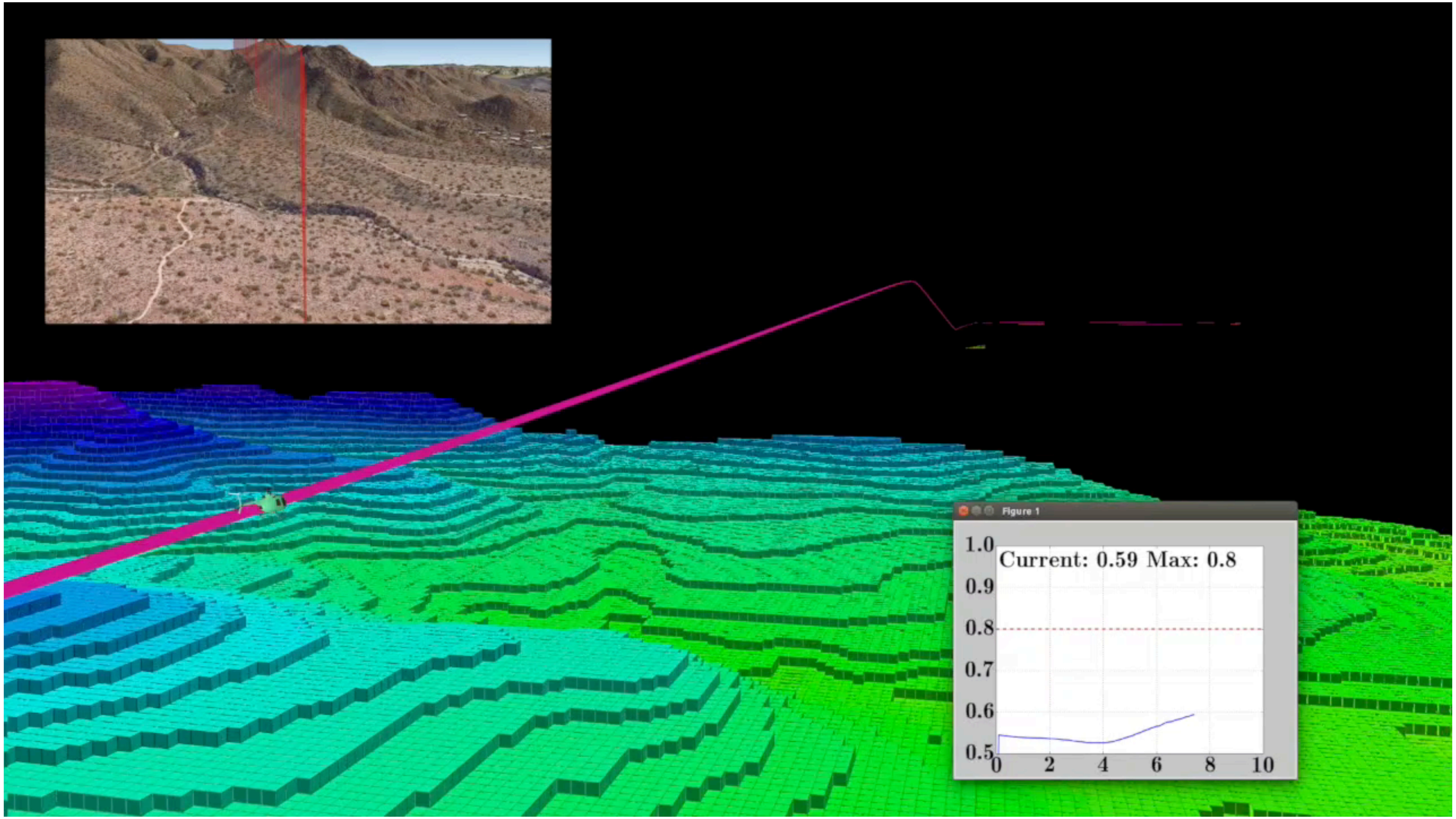
Challenge 3: Real-time Constraints



Stanford DARPA Challenge, 2007

https://www.youtube.com/watch?v=qXZt-B7iUyw&feature=youtu.be

Challenge 3: Real-time Constraints



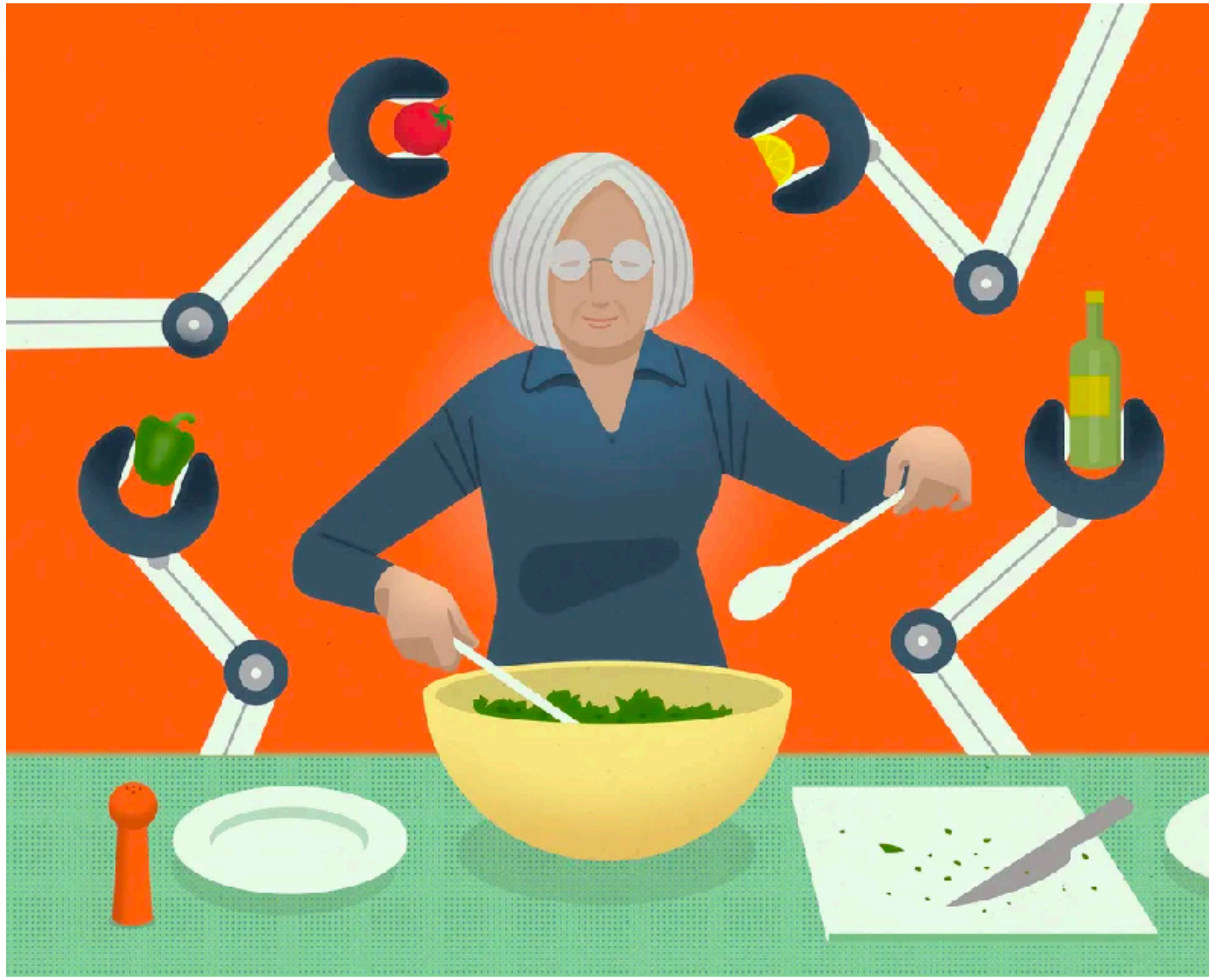
Activity!

Think-Pair-Share!

Think (30 sec): Let's say you have a robot arm cooking with grandma in the kitchen. How should it quickly plan safe paths?

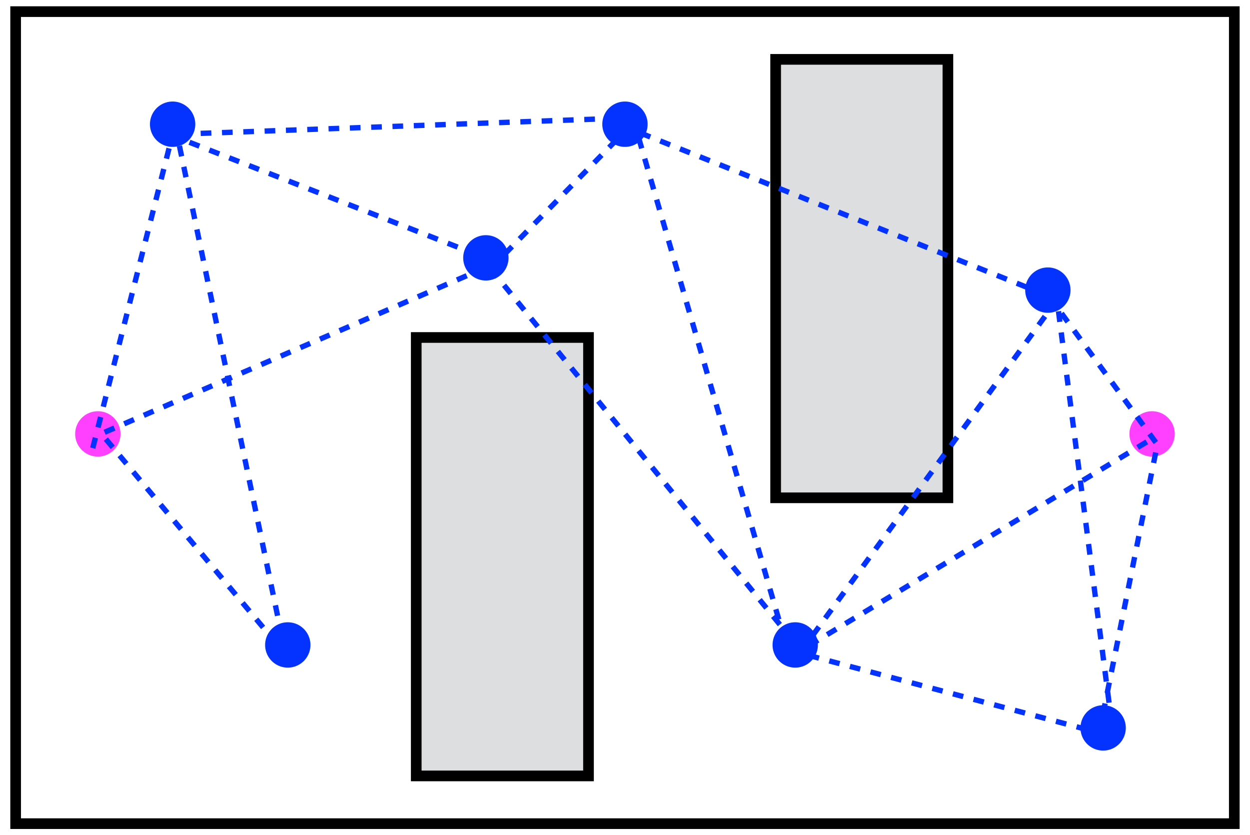
Pair: Find a partner

Share (45 sec): Partners exchange ideas

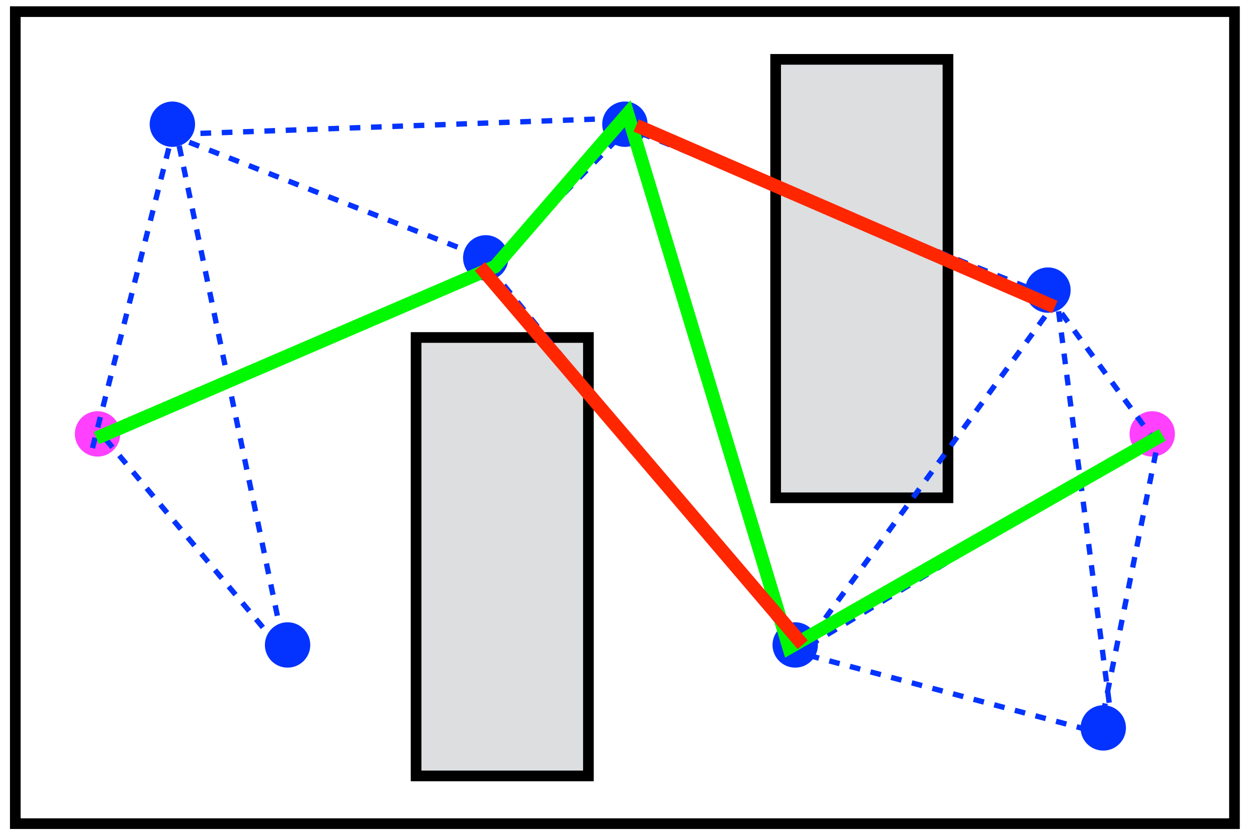


Unified Framework

General framework for motion planning



Create a graph

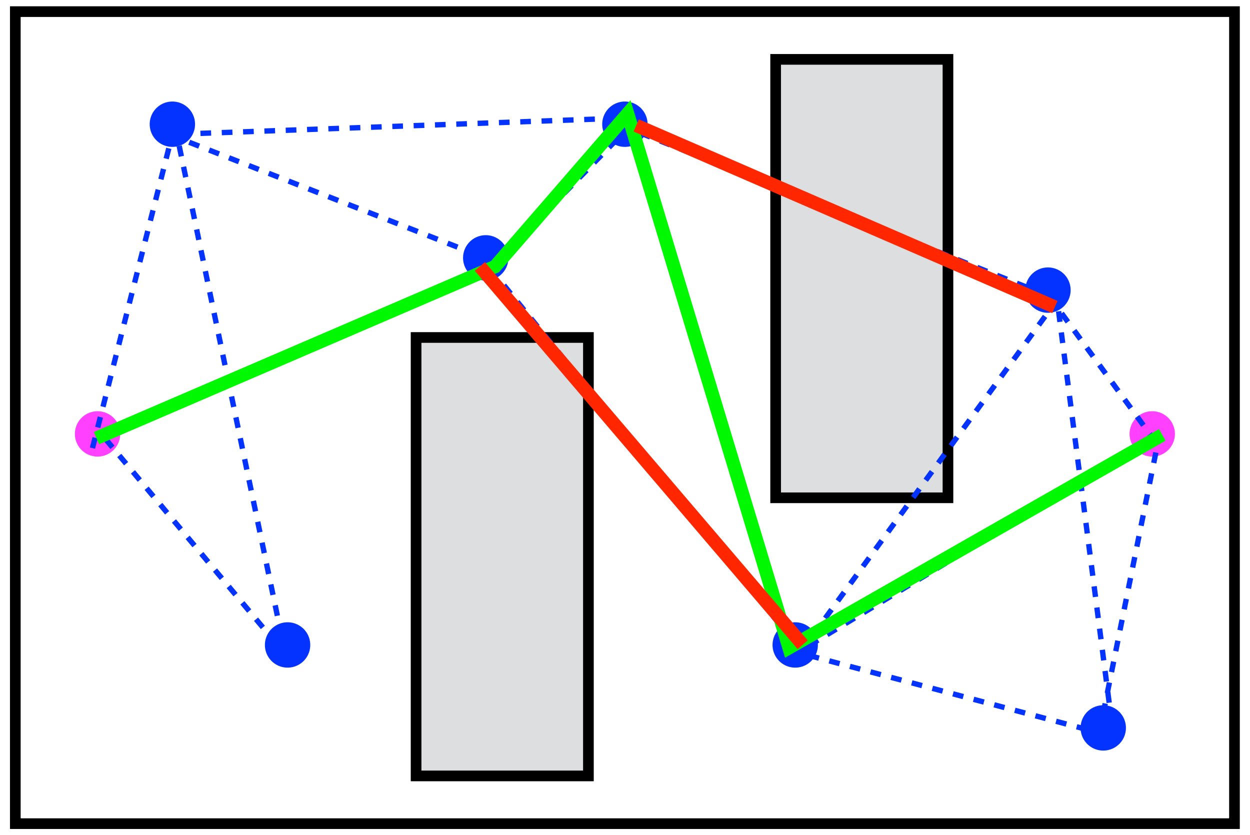


Search the graph

Interleave

General framework for motion planning

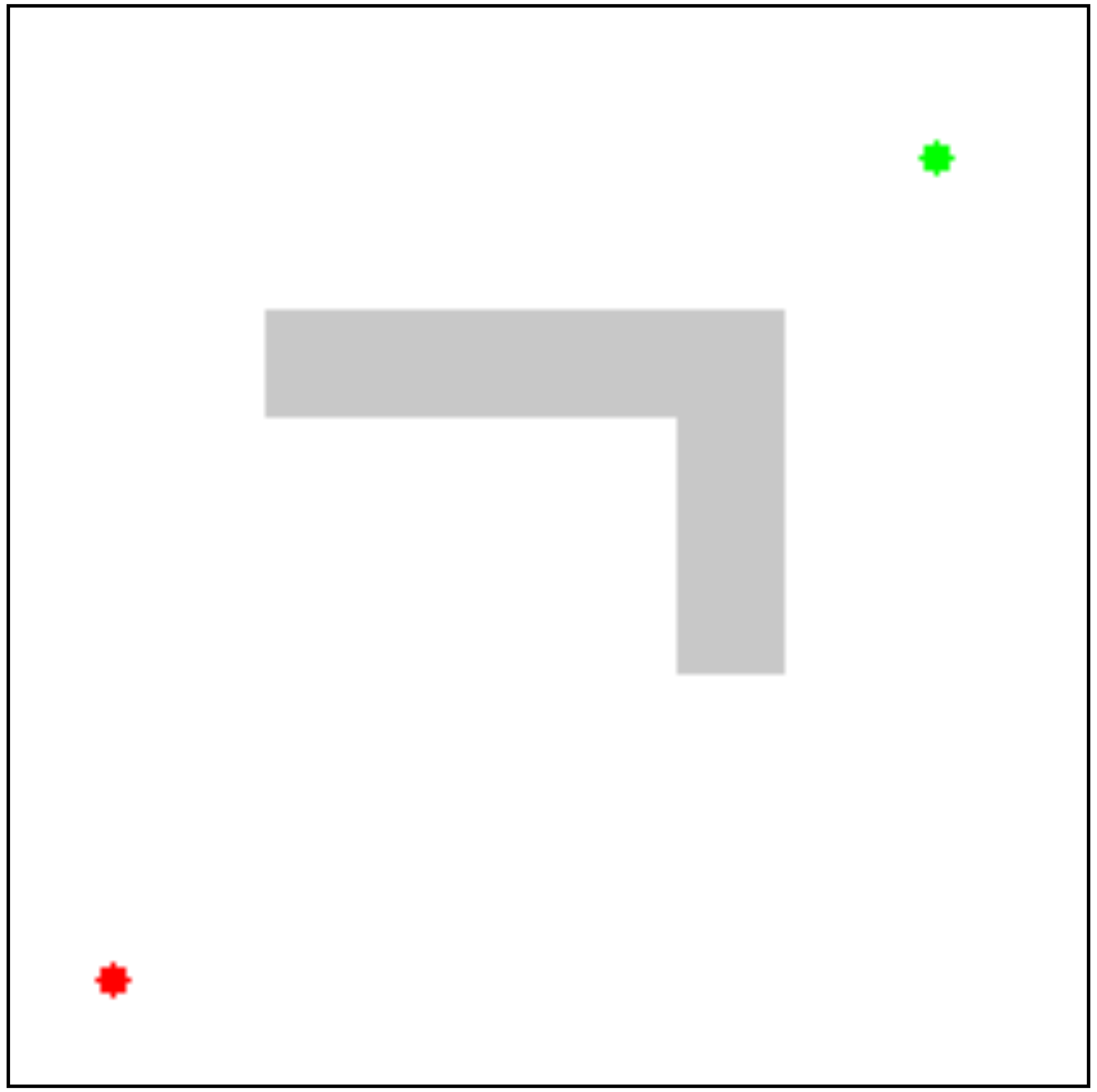
Create a graph



Search the graph

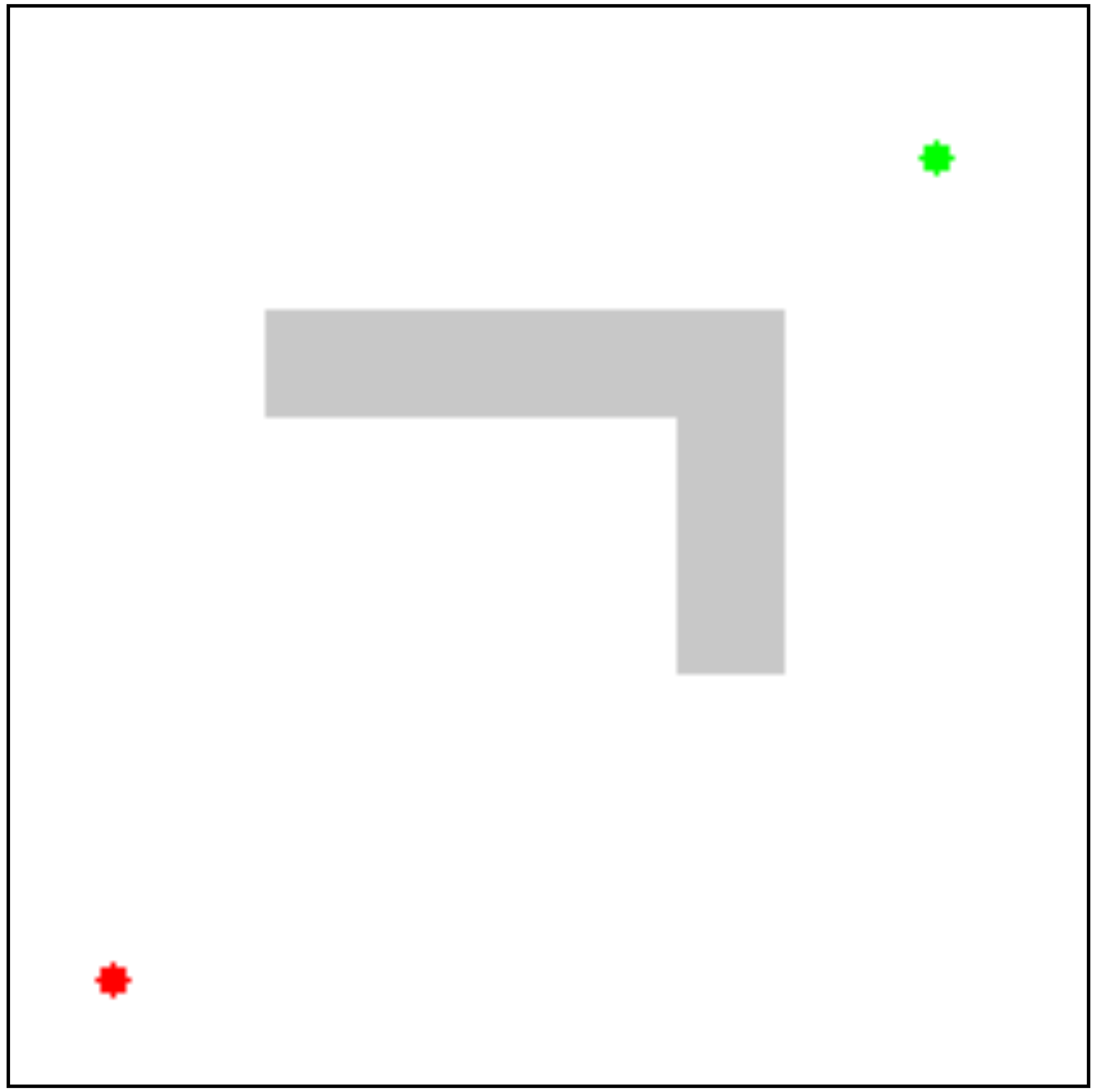
Interleave

How can we make this search faster?

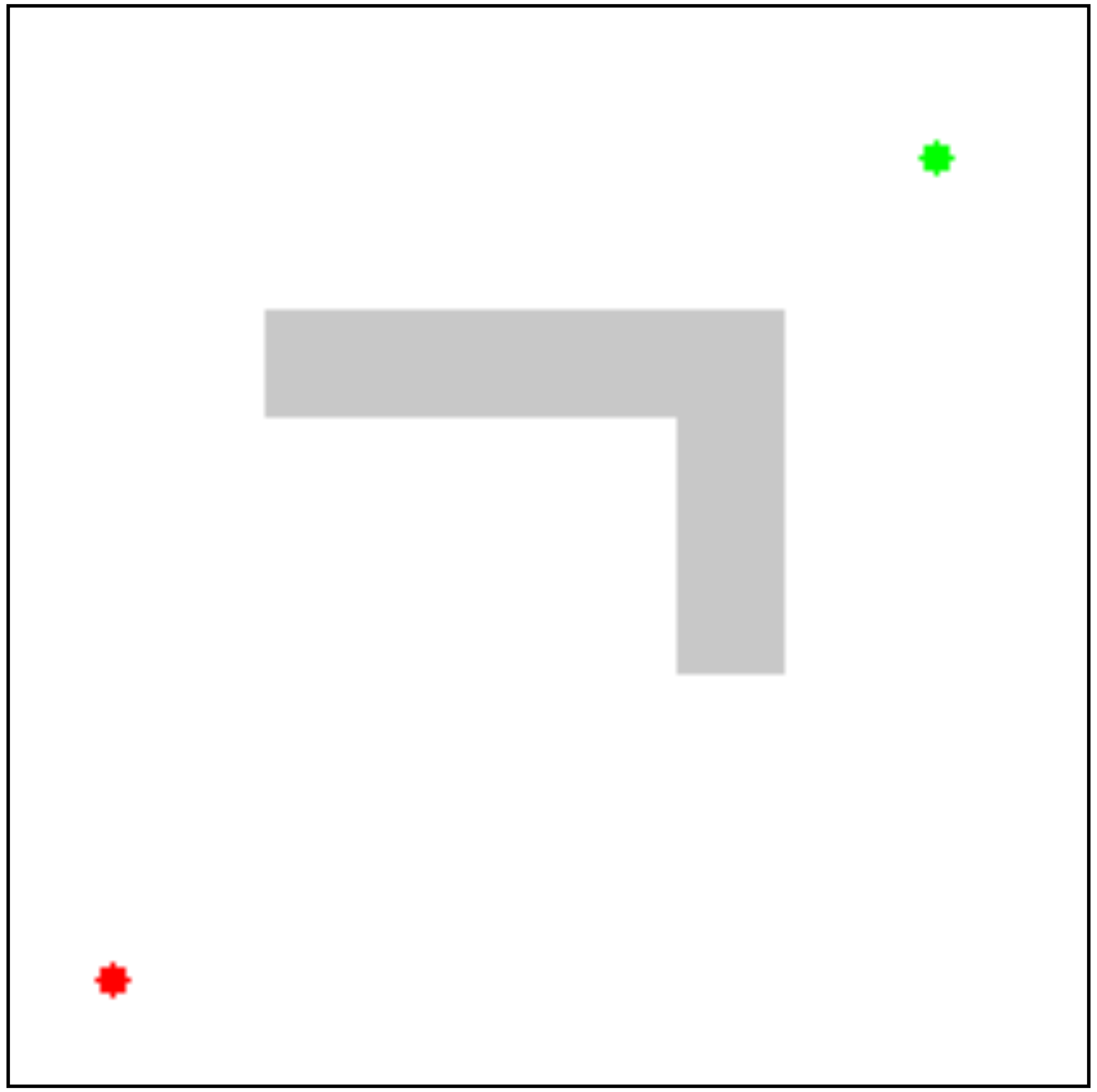


Dijkstra

How can we make this search faster?

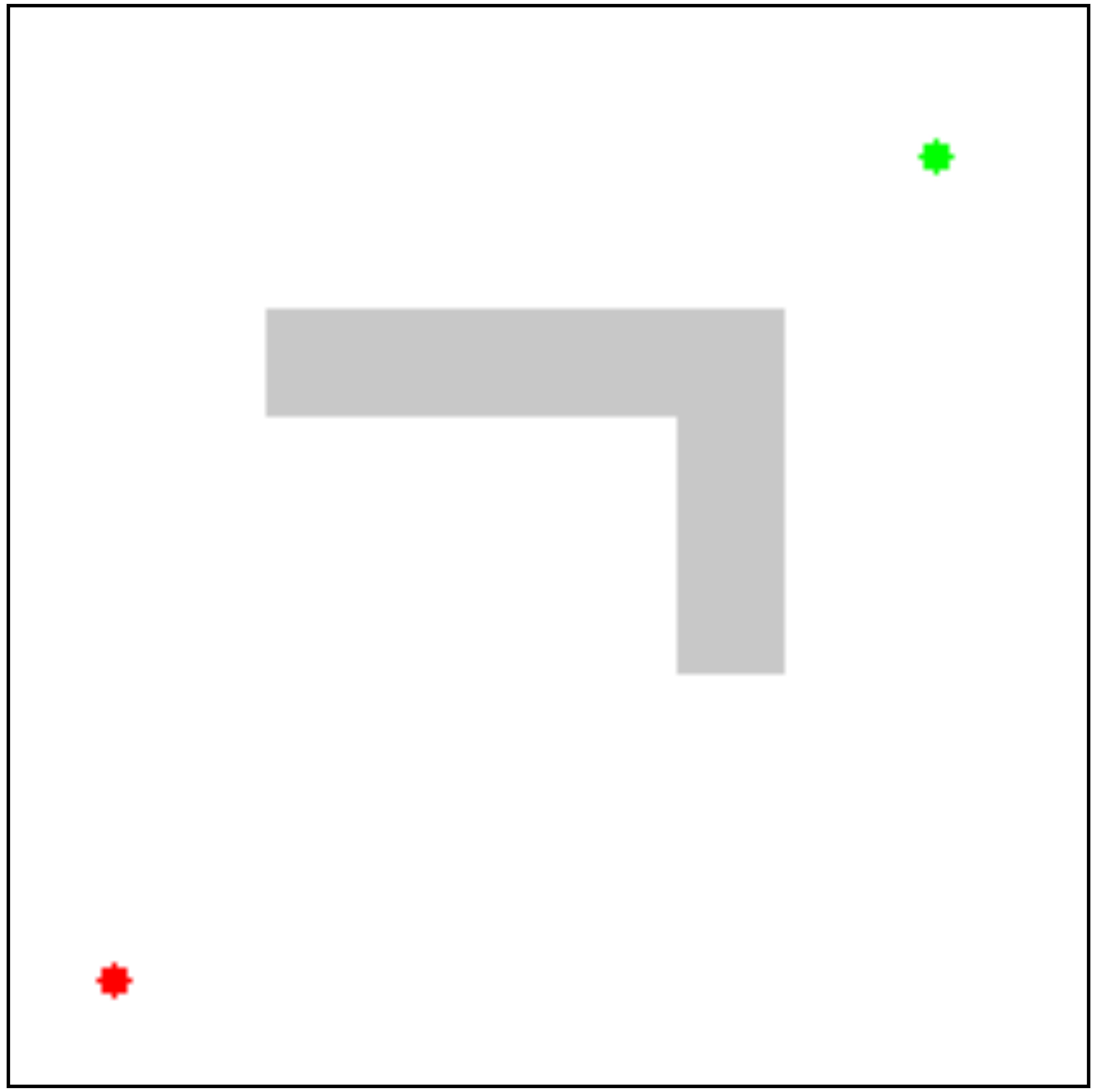


Dijkstra



A* with heuristic!

What makes a heuristic good?

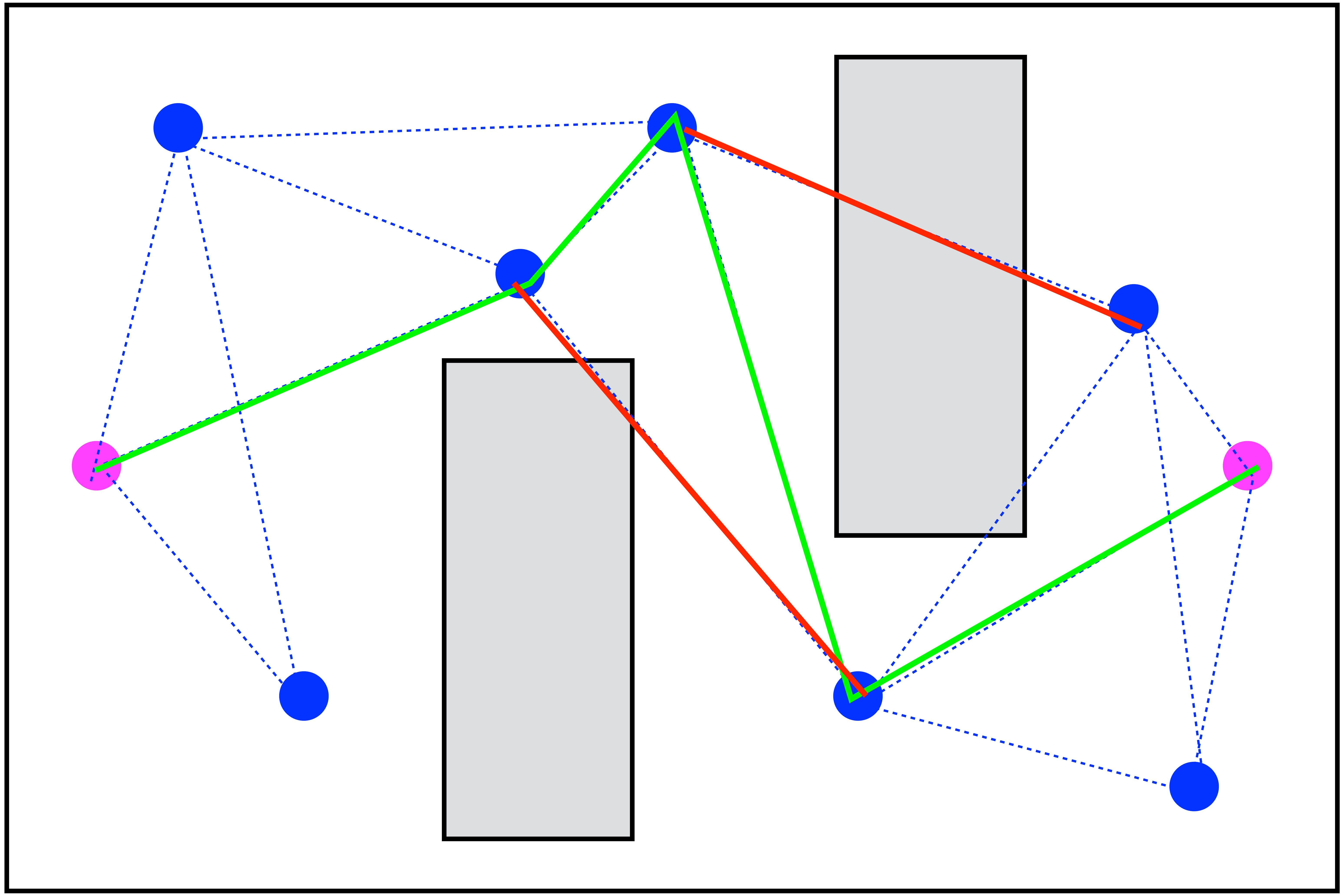


A* with heuristic!

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

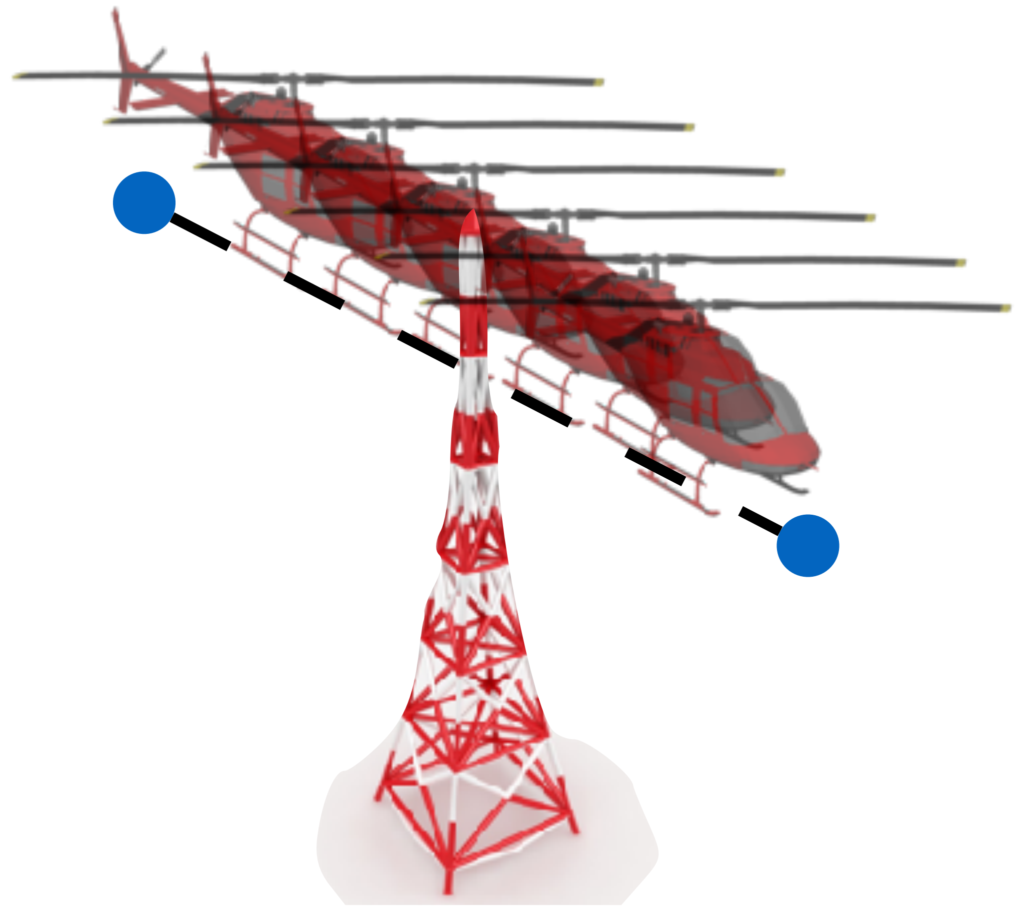
What is the most expensive step?

Edge evaluation is the most expensive step



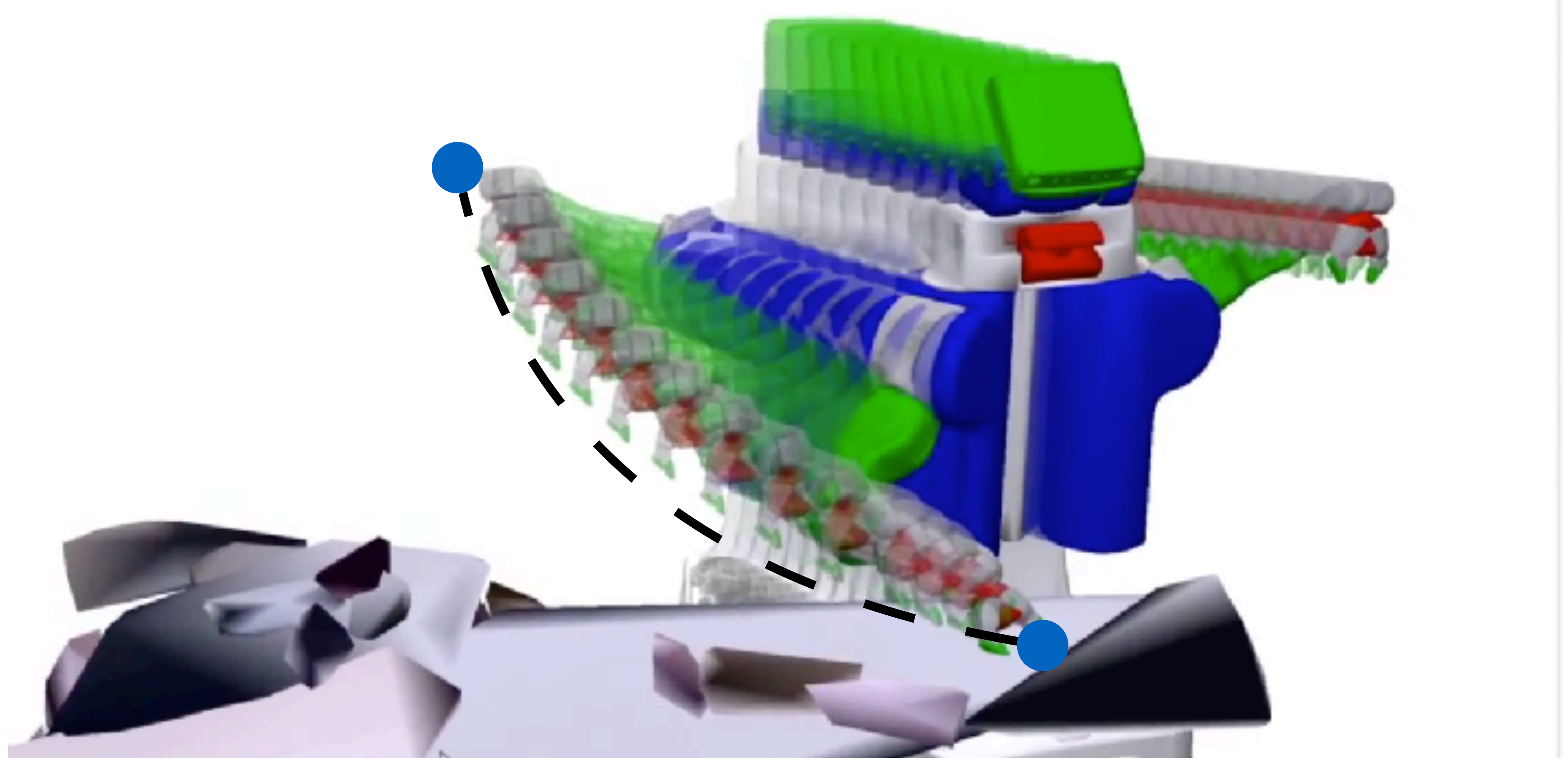
Why?

Edge evaluation requires expensive collision checking



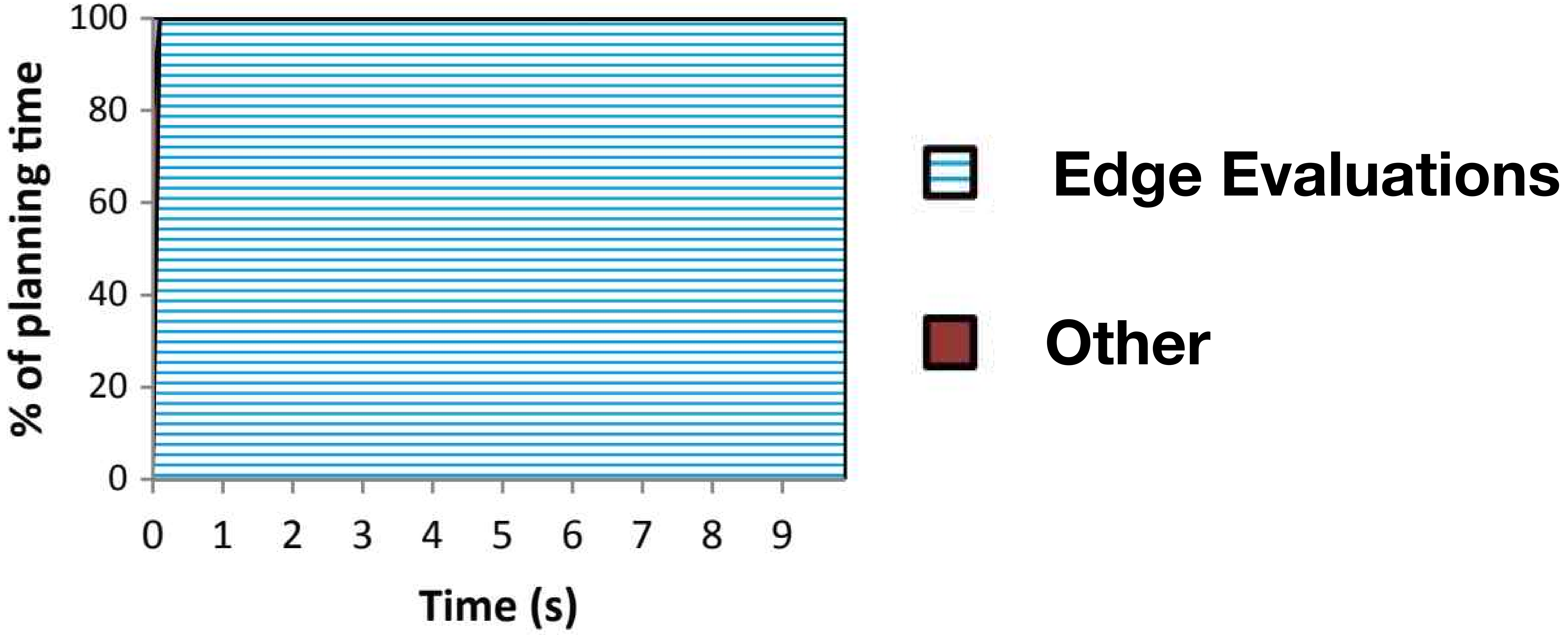
Check if helicopter intersects with tower

(Schluman et al. ’14)



Check if manipulator intersects with table

Edge evaluation dominates planning time

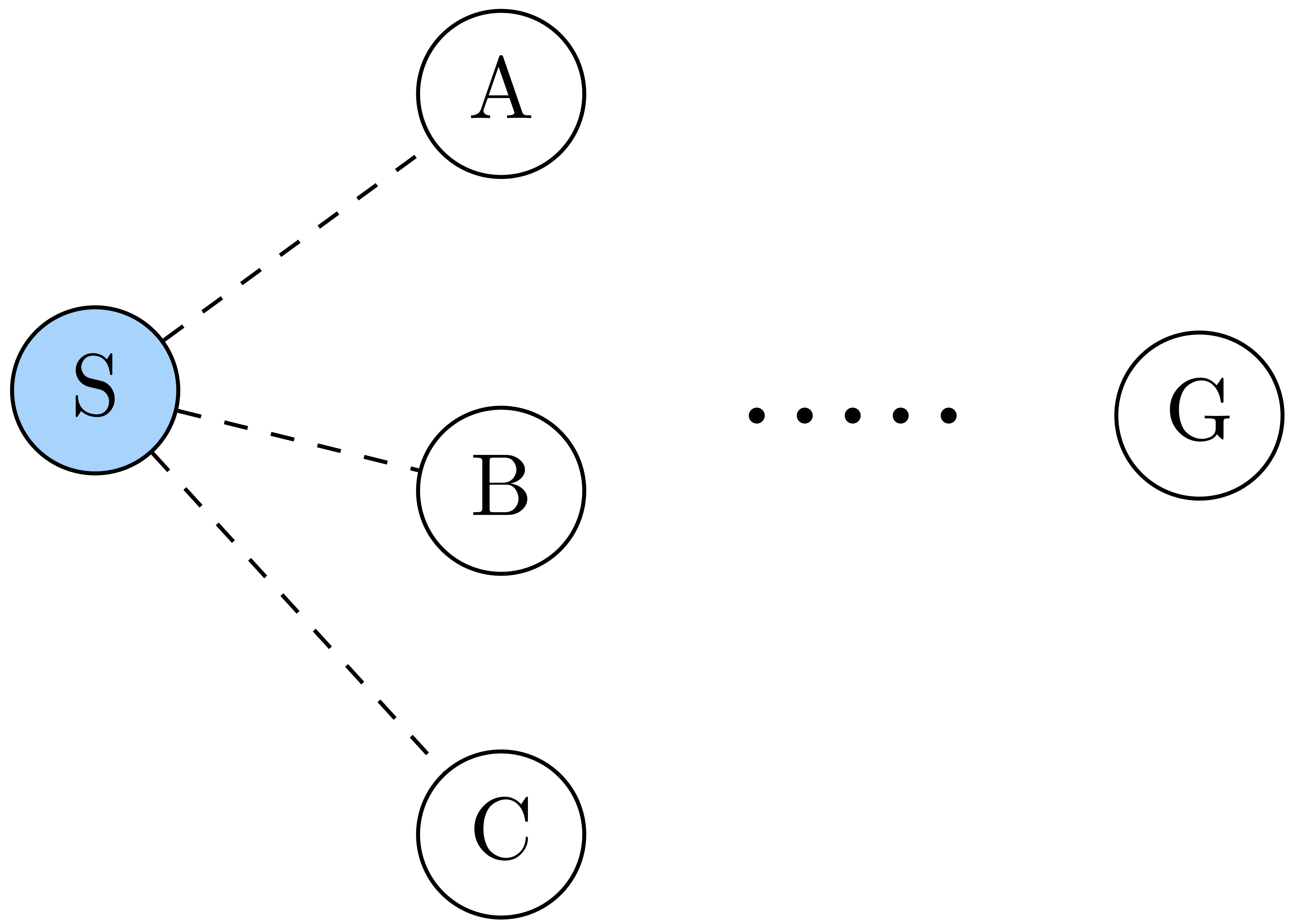


Hauser, Kris., Lazy collision checking in asymptotically-optimal motion planning. *ICRA* 2015

How do we modify A* search to minimize edge evaluation?

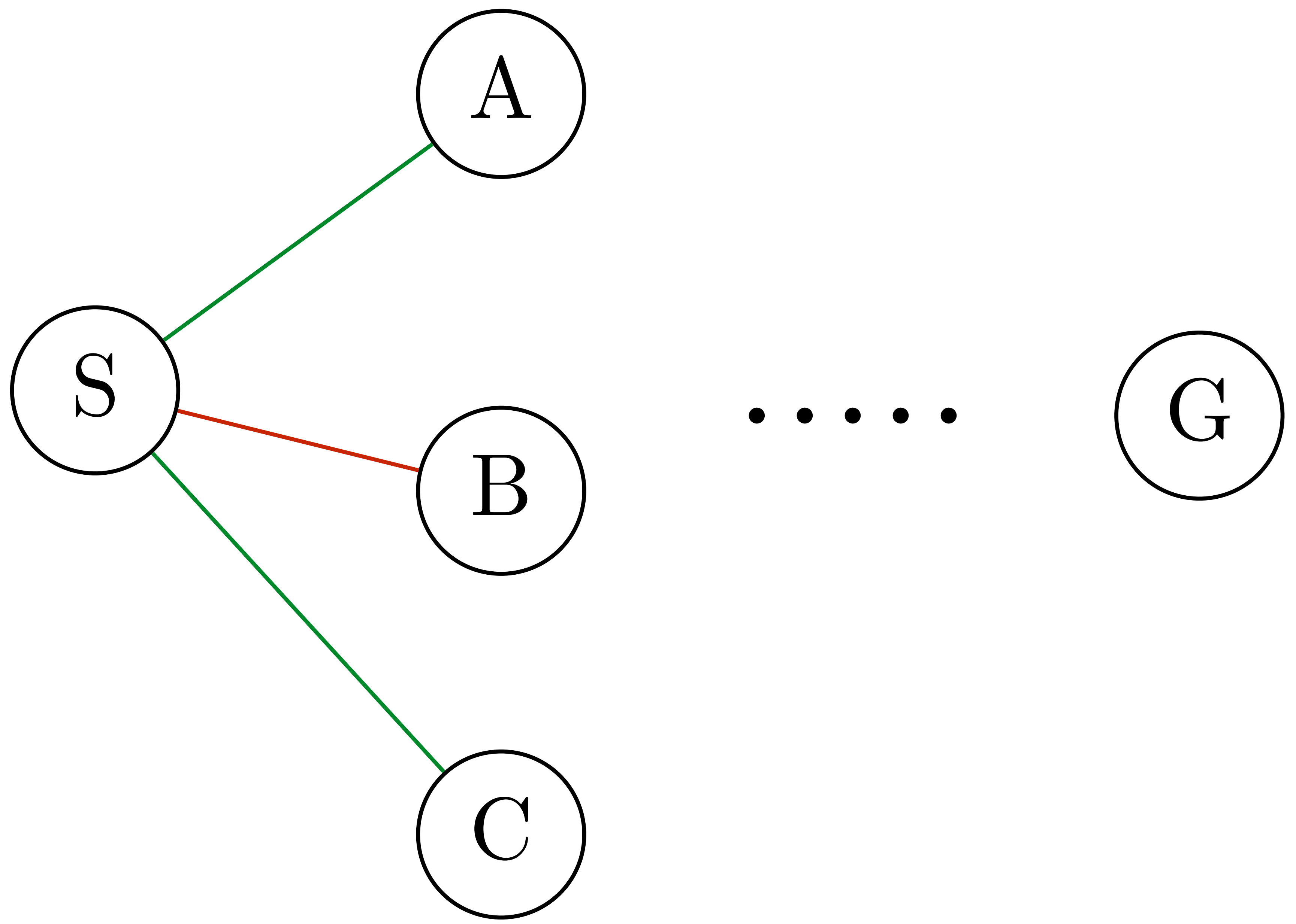
Let's revisit Best First Search

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



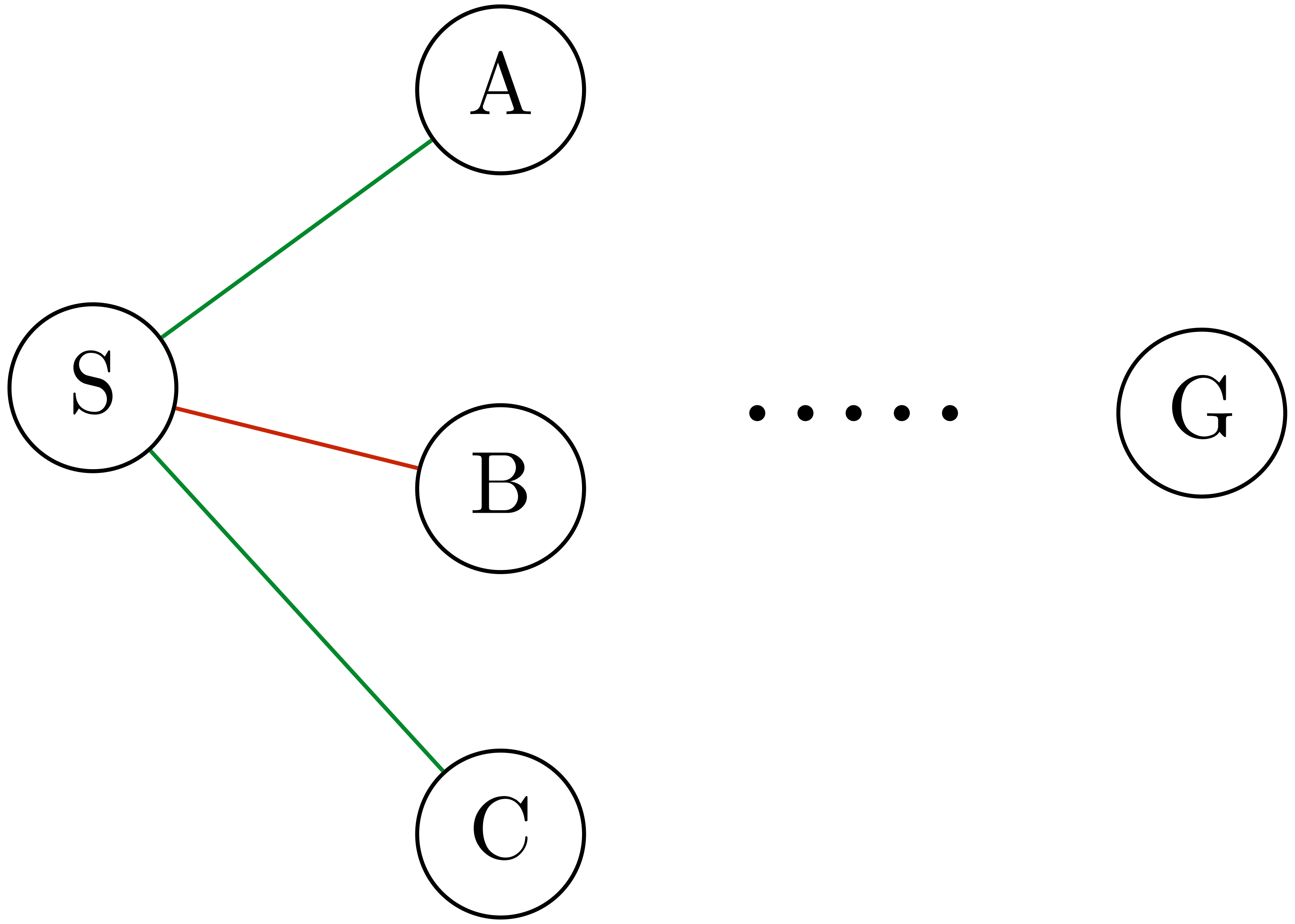
Let's revisit Best First Search

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 Virtue of Laziness

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

What is the laziest that we can be?

LazySP

(Lazy Shortest Path)

Dellin and Srinivasa, 2016

First Provably Edge-Optimal A*-like Search Algorithm

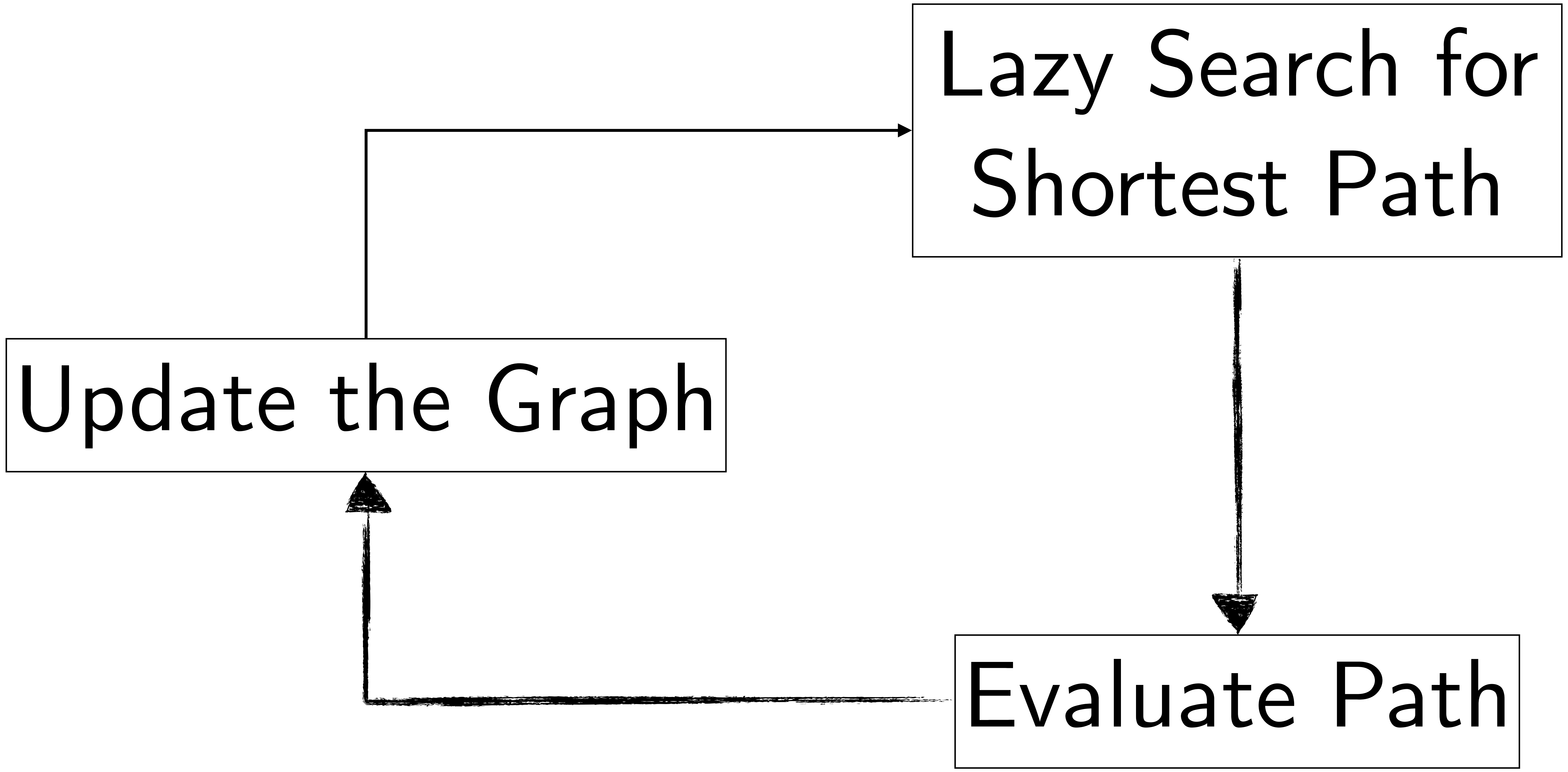
LazySP

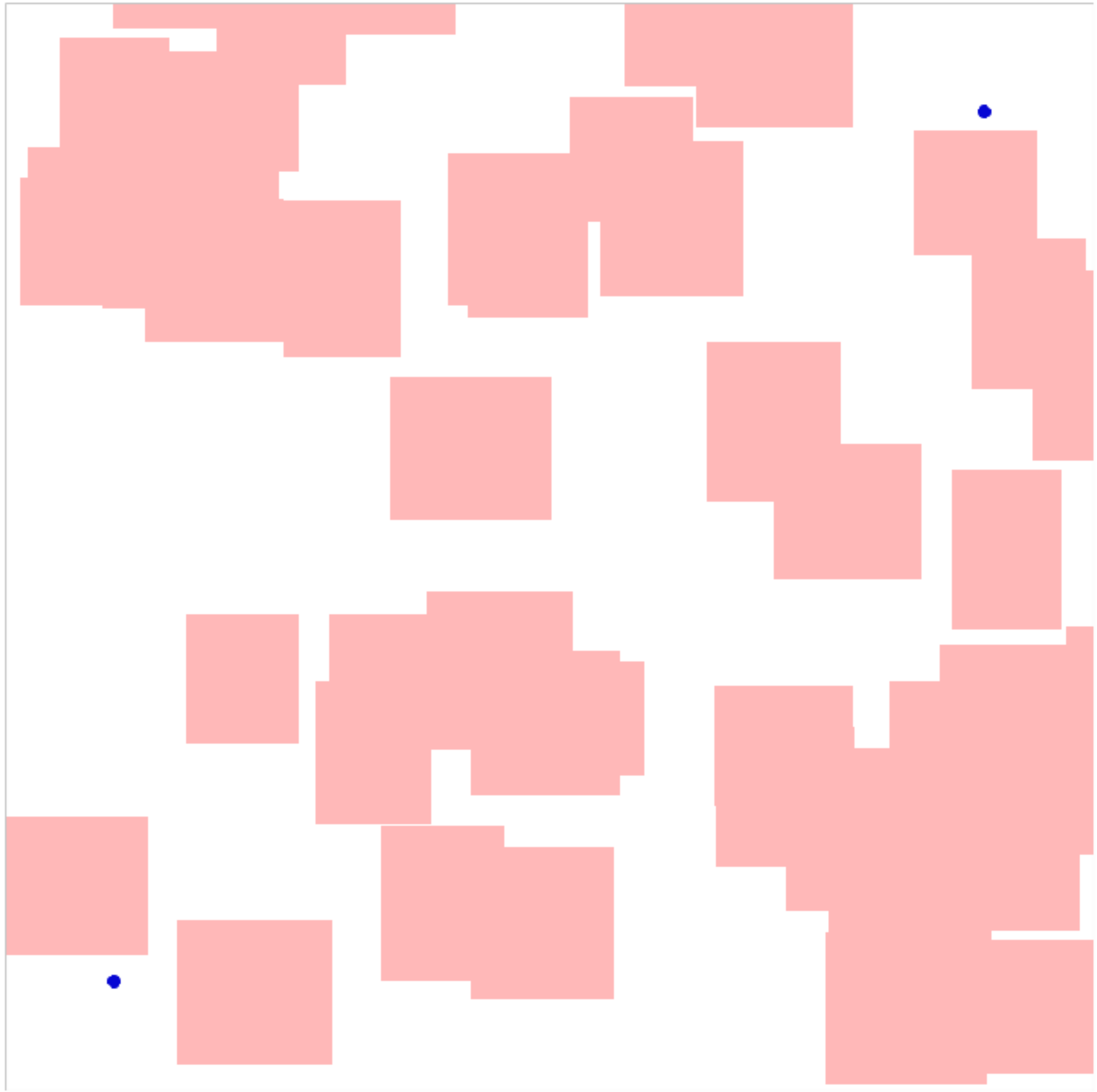
Greedy Best-first Search over Paths

To find the shortest path,
eliminate all shorter paths!

LazySP

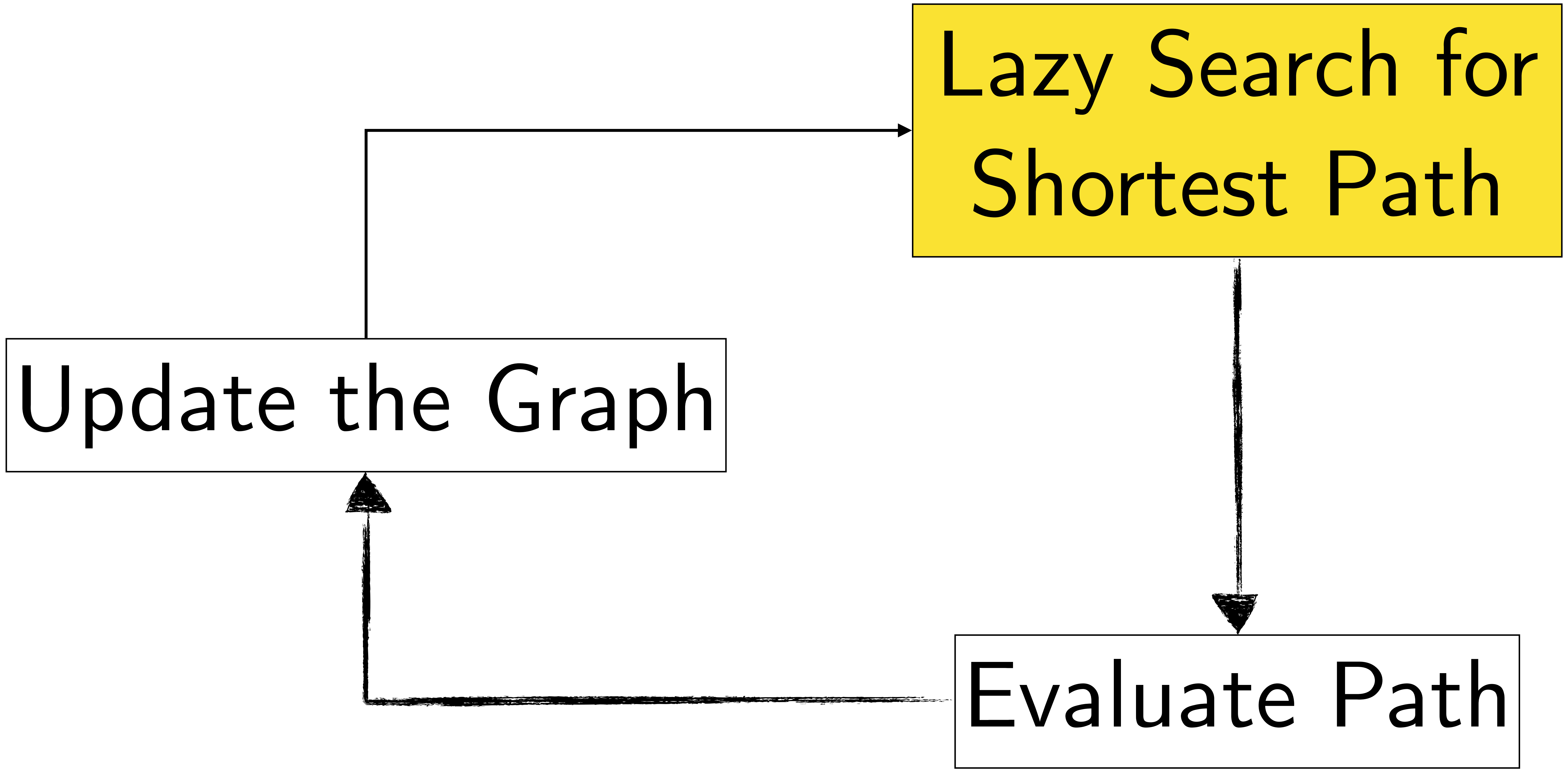
Optimism Under Uncertainty

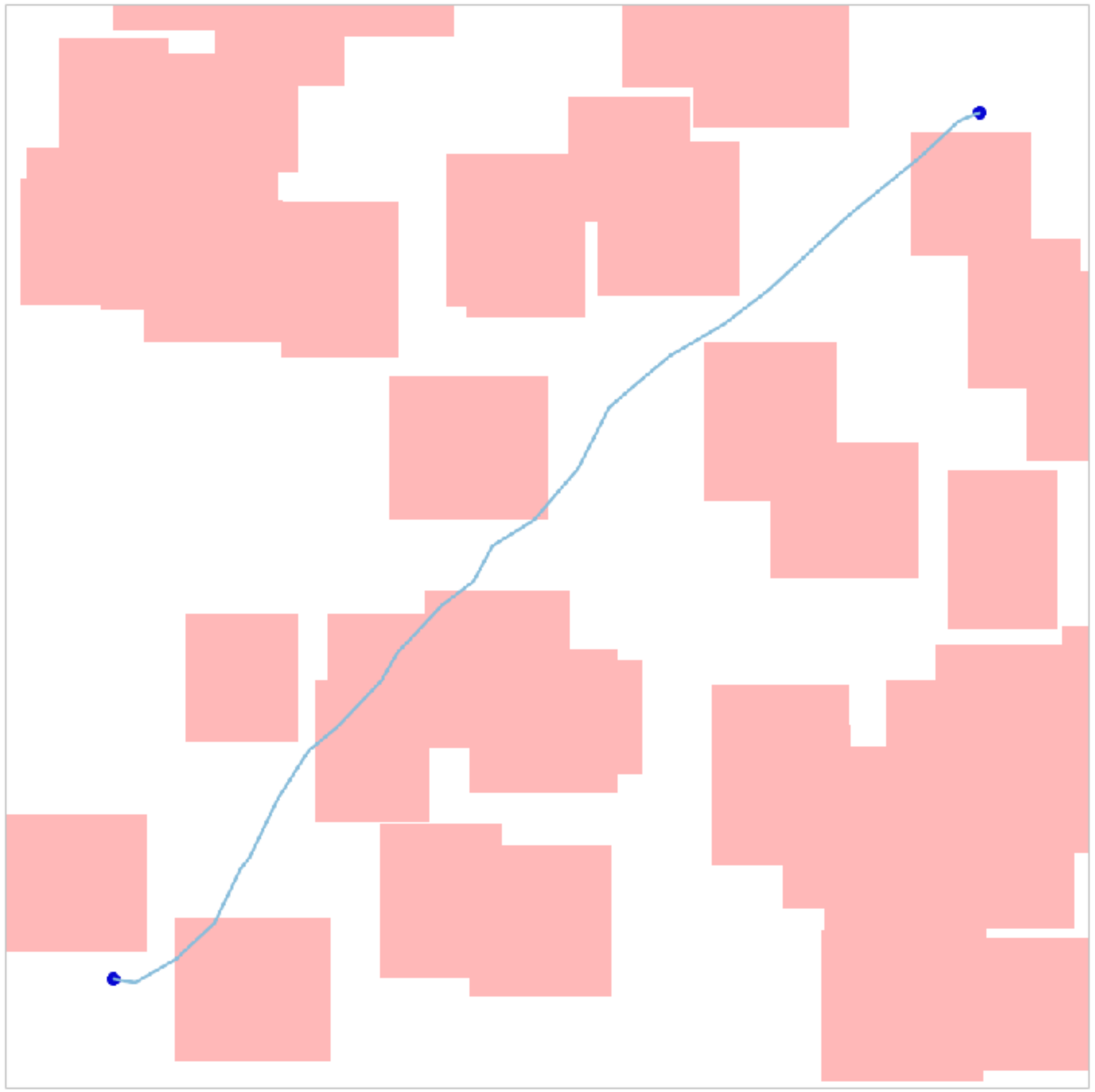




LazySP

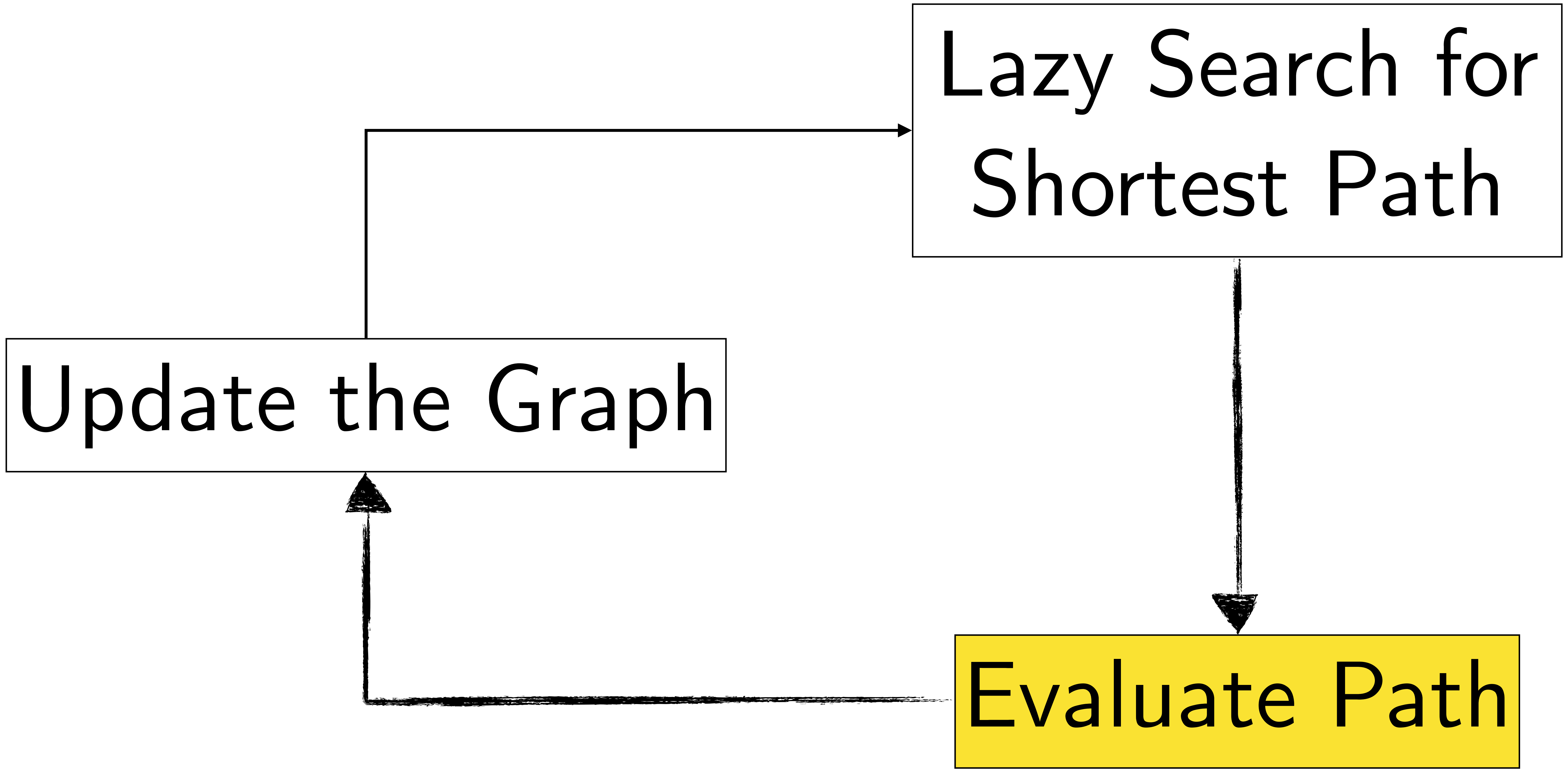
Optimism Under Uncertainty

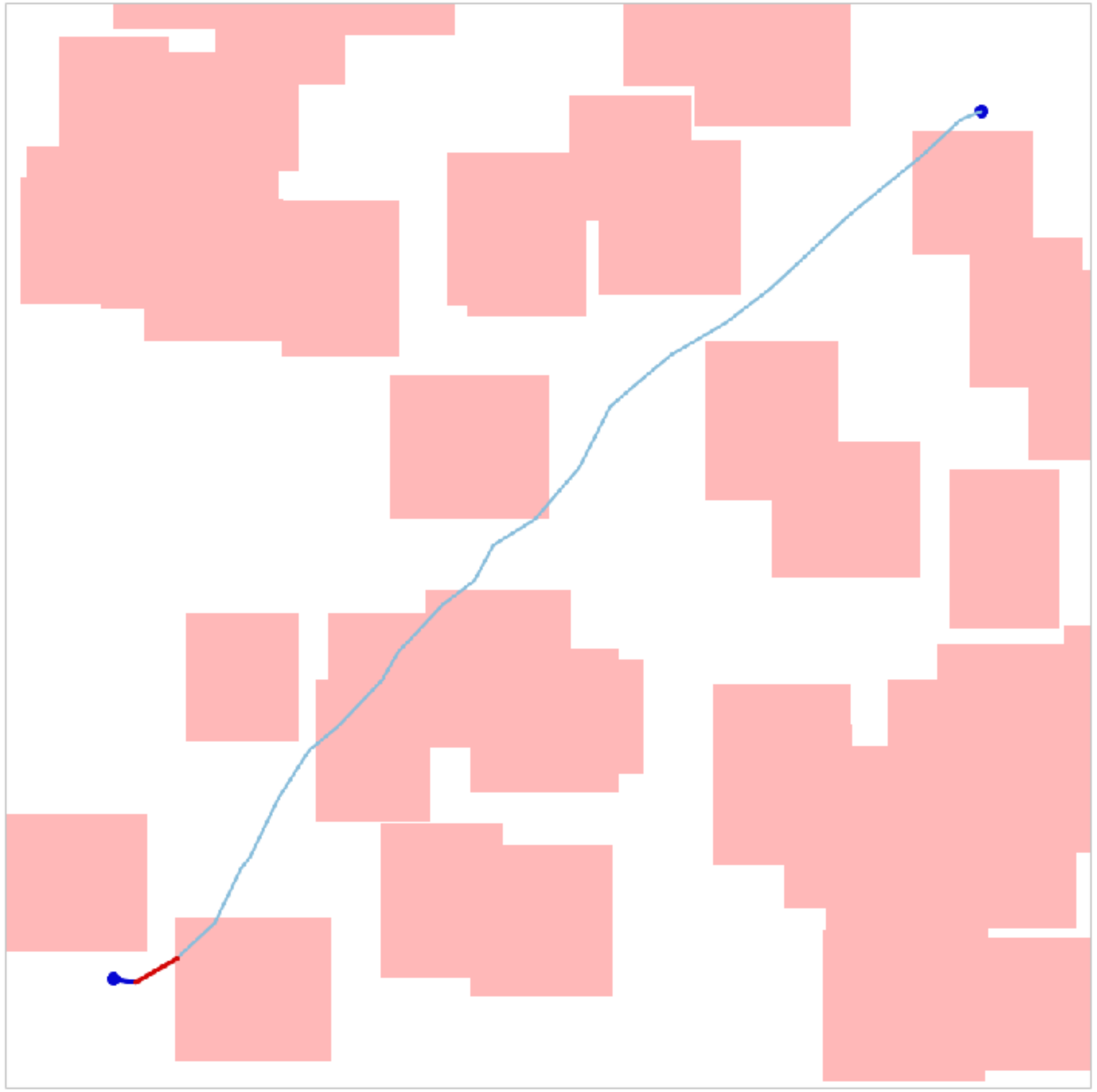




LazySP

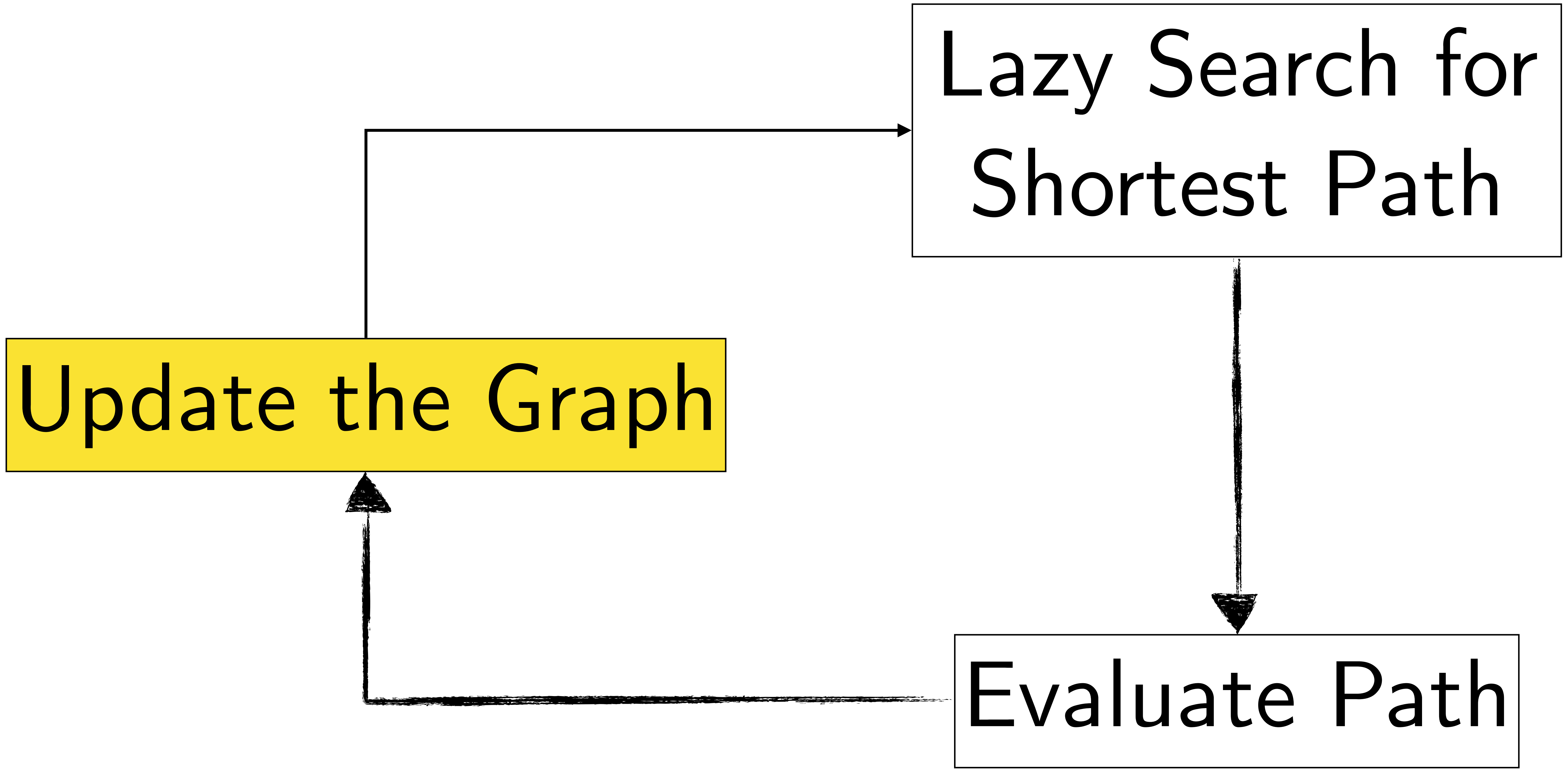
Optimism Under Uncertainty

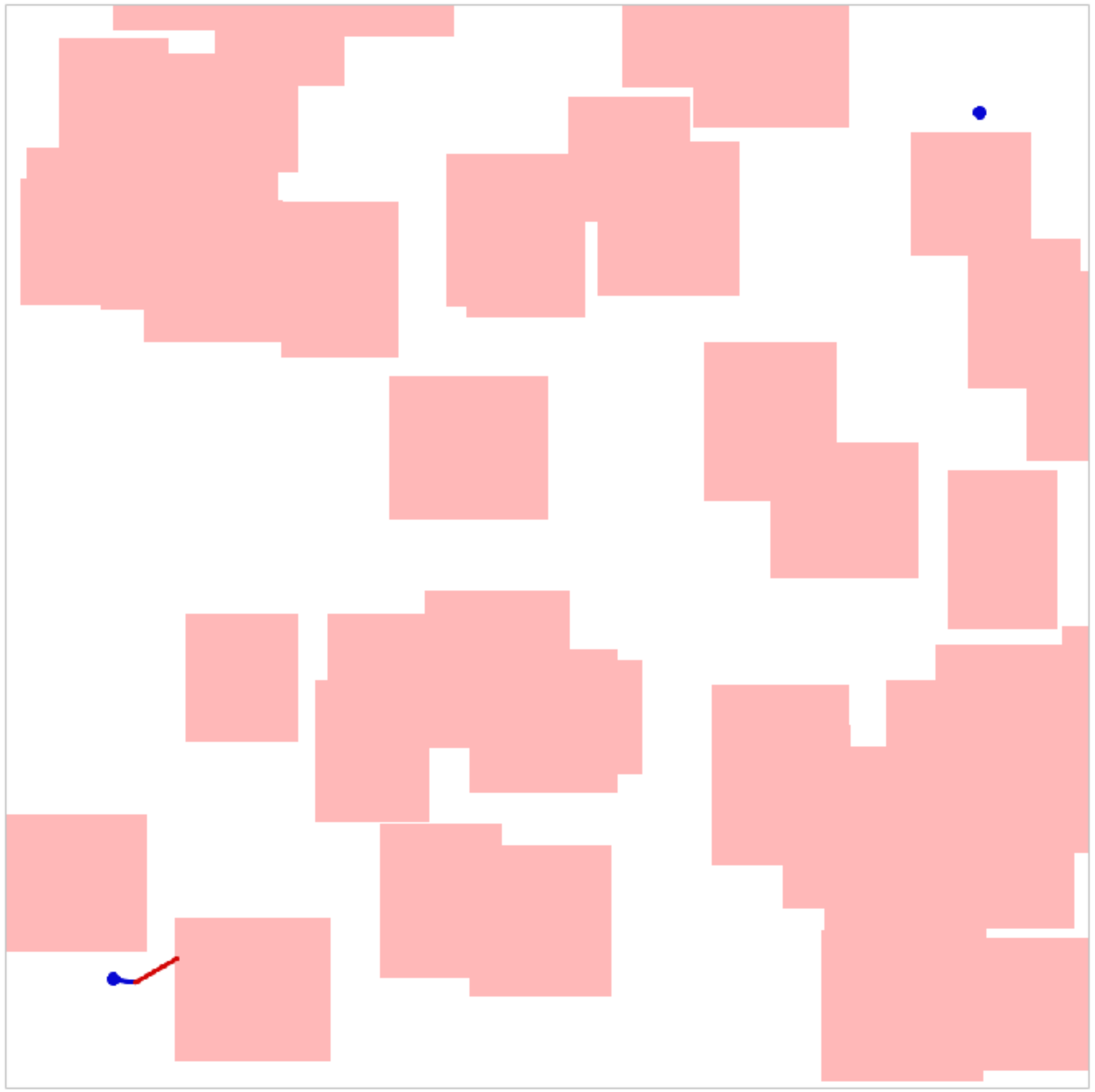




LazySP

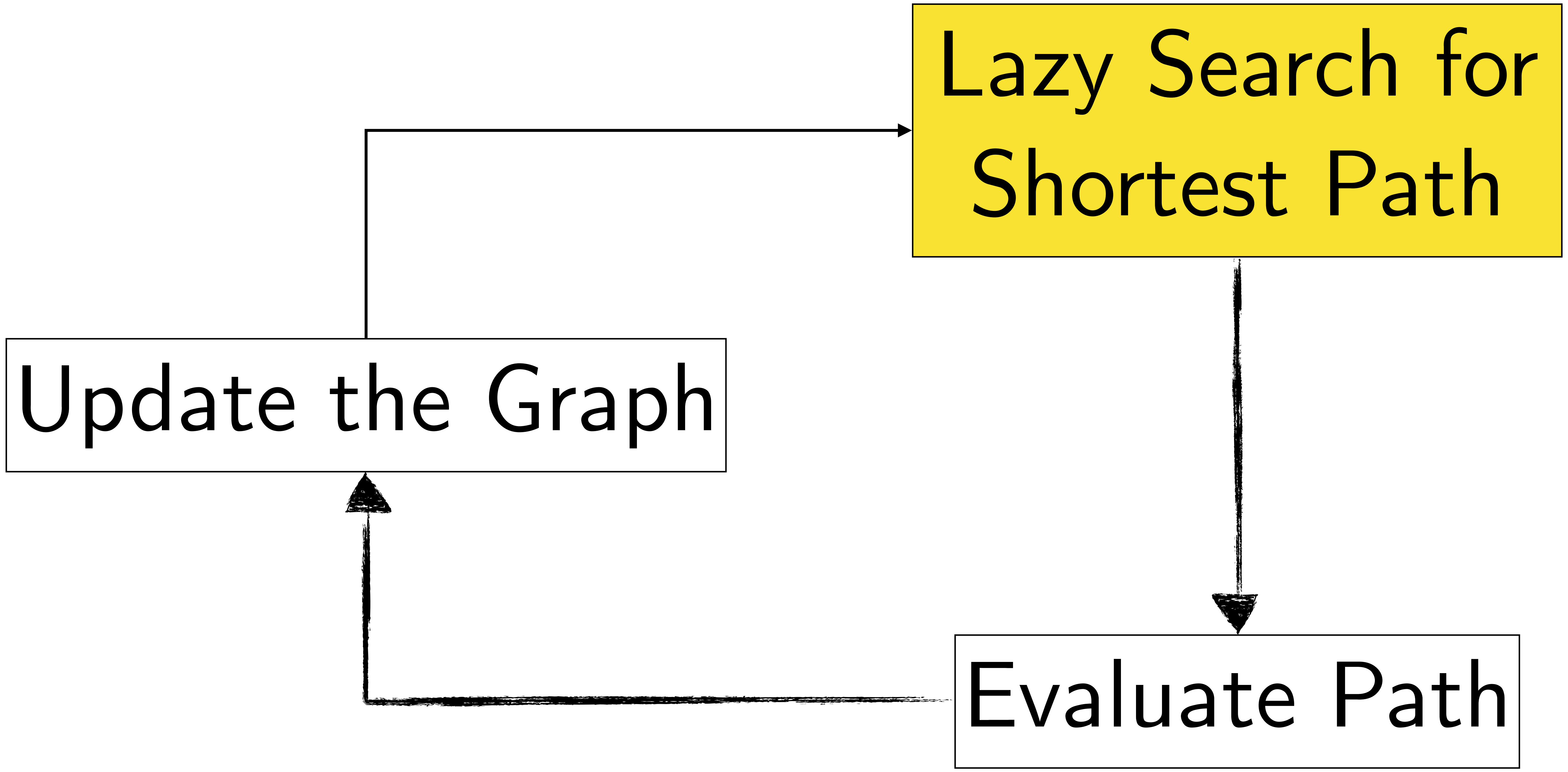
Optimism Under Uncertainty

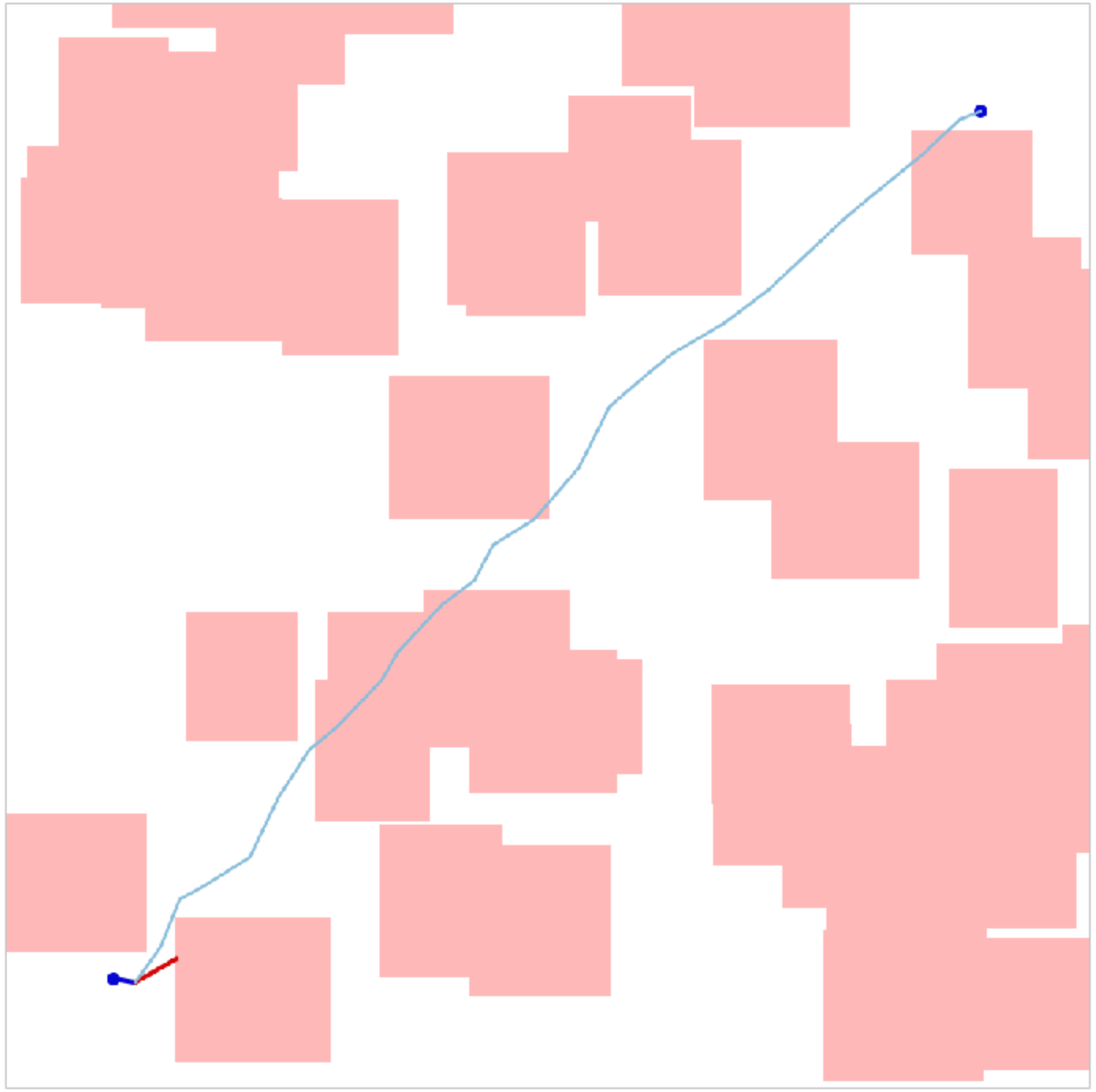




LazySP

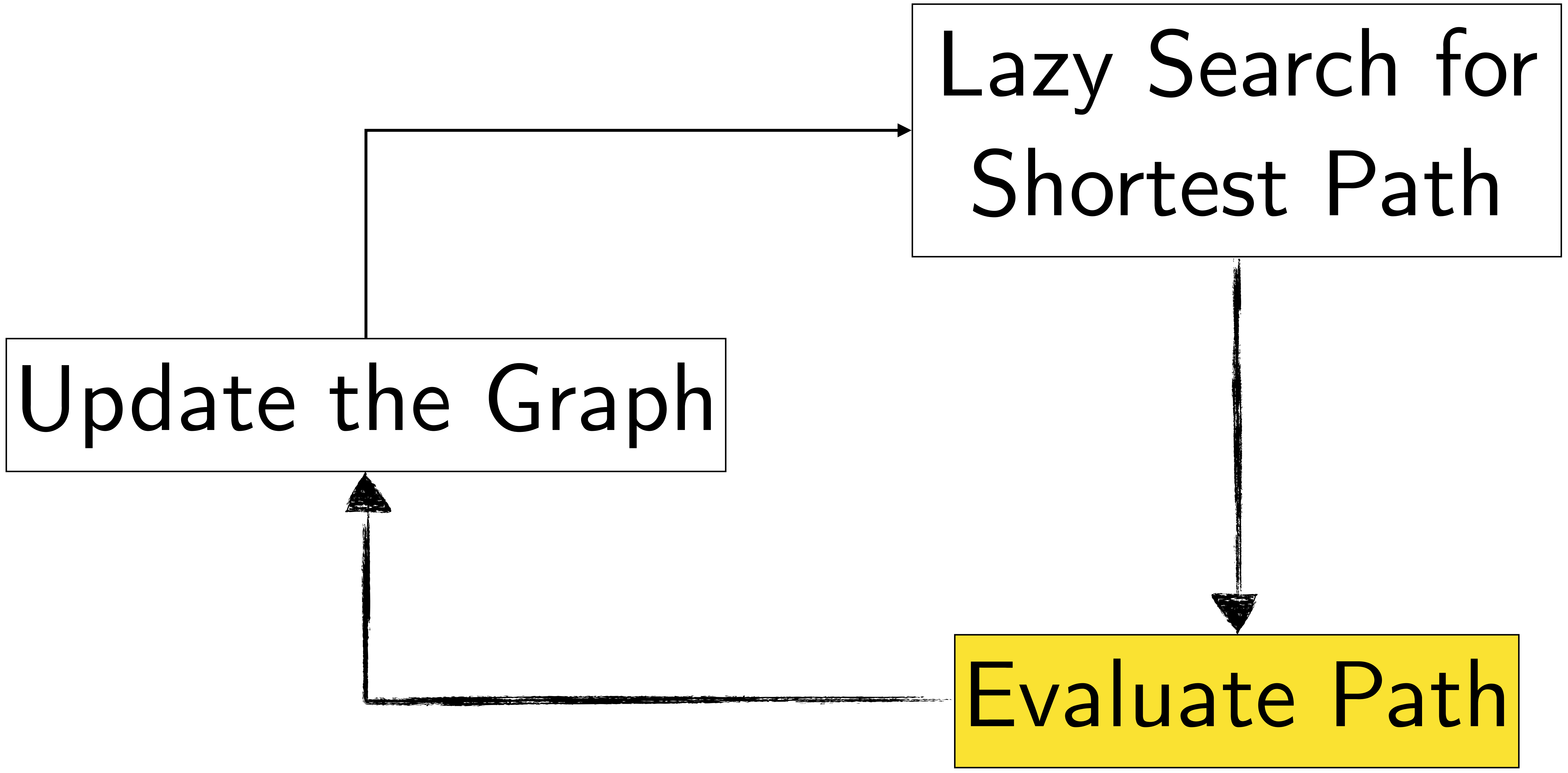
Optimism Under Uncertainty

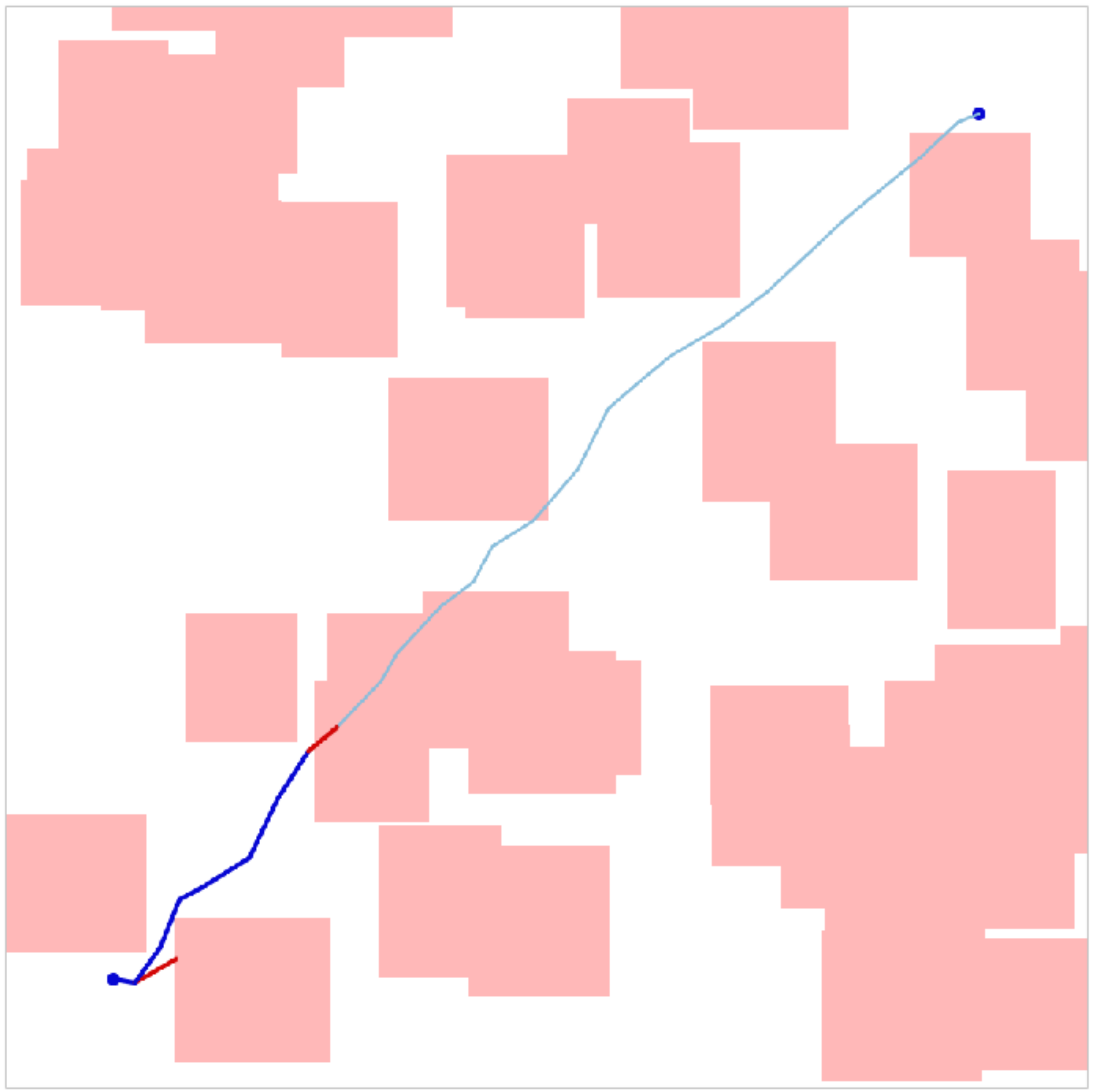




LazySP

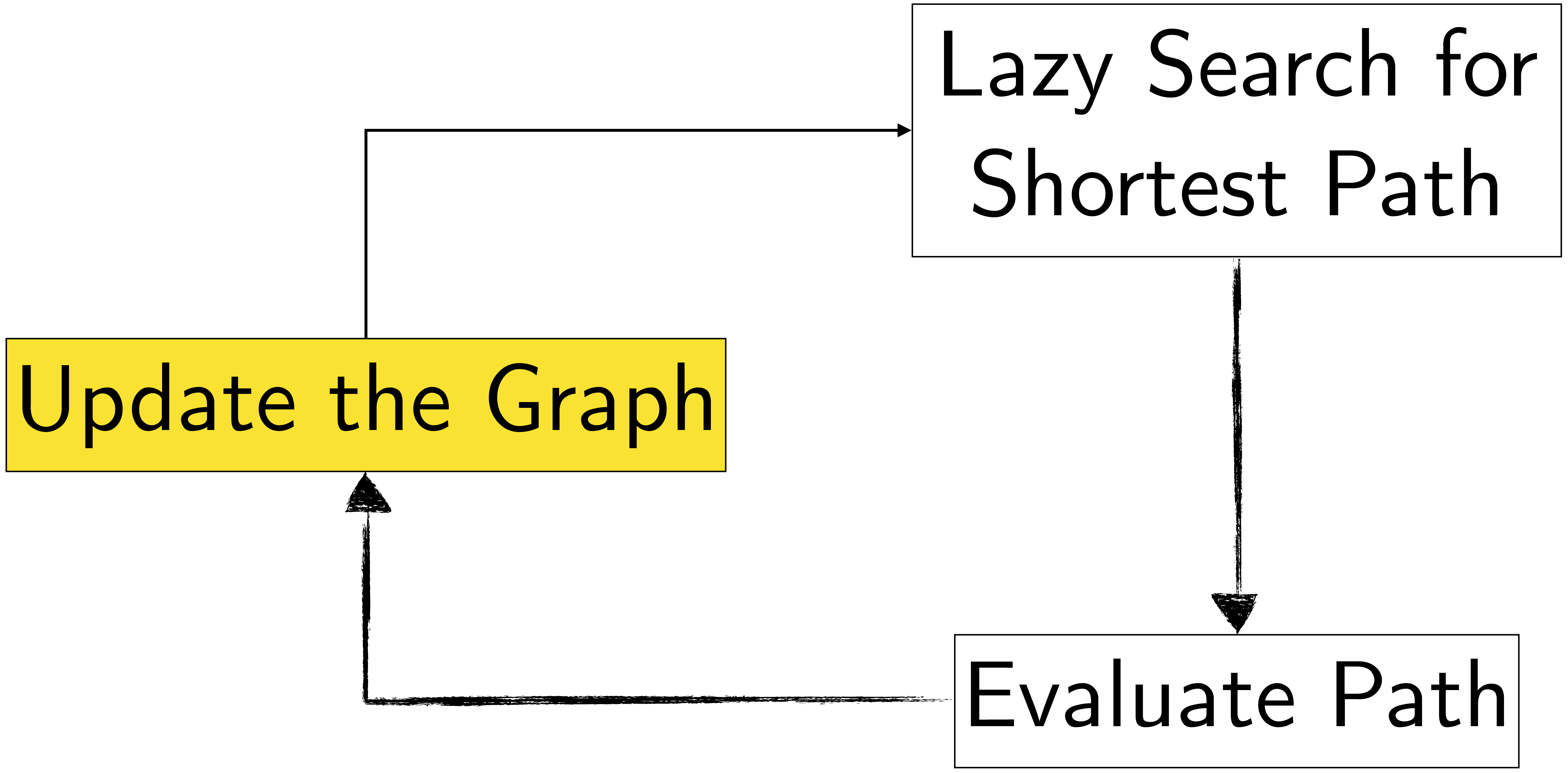
Optimism Under Uncertainty





LazySP

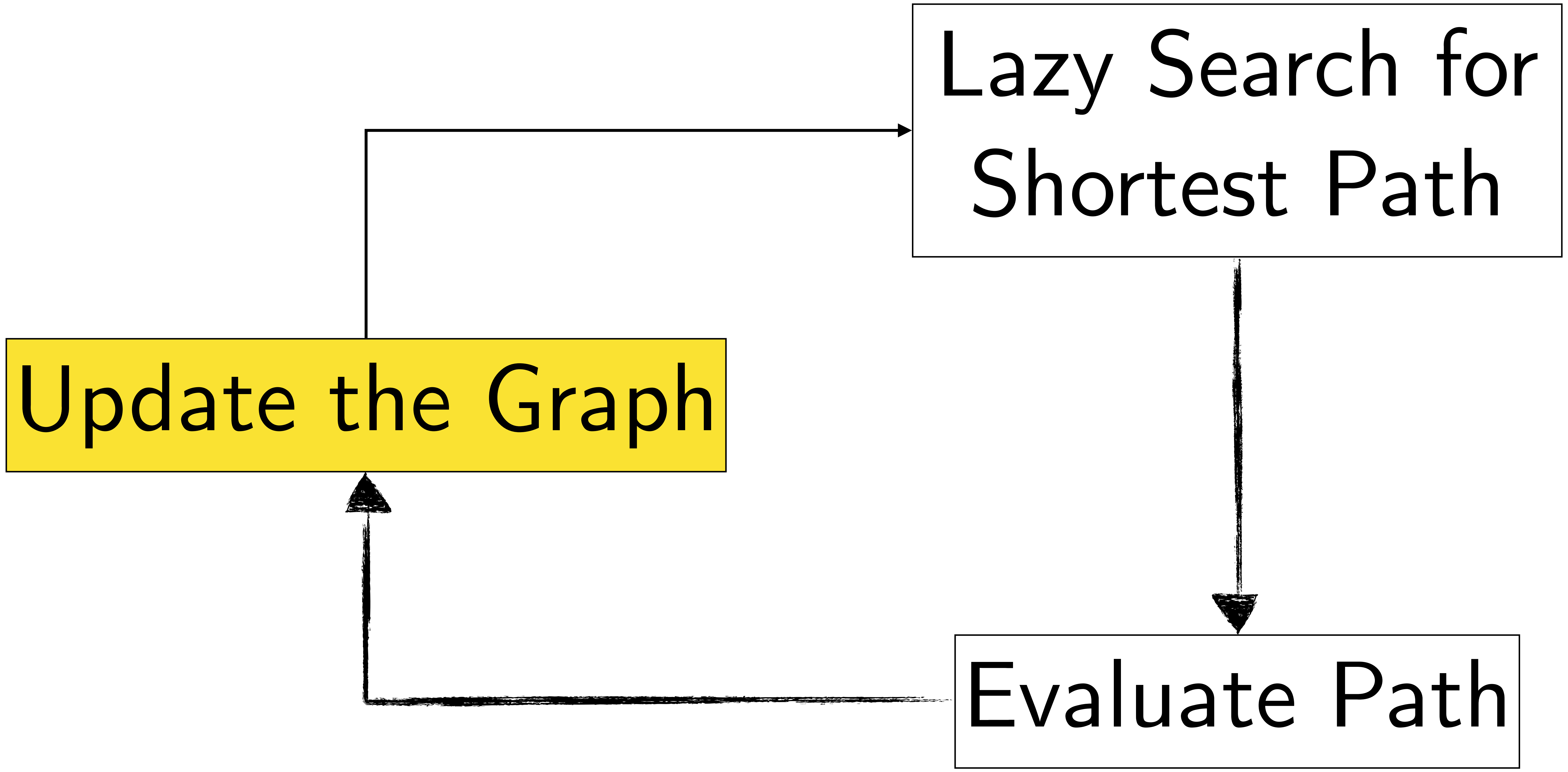
Optimism Under Uncertainty

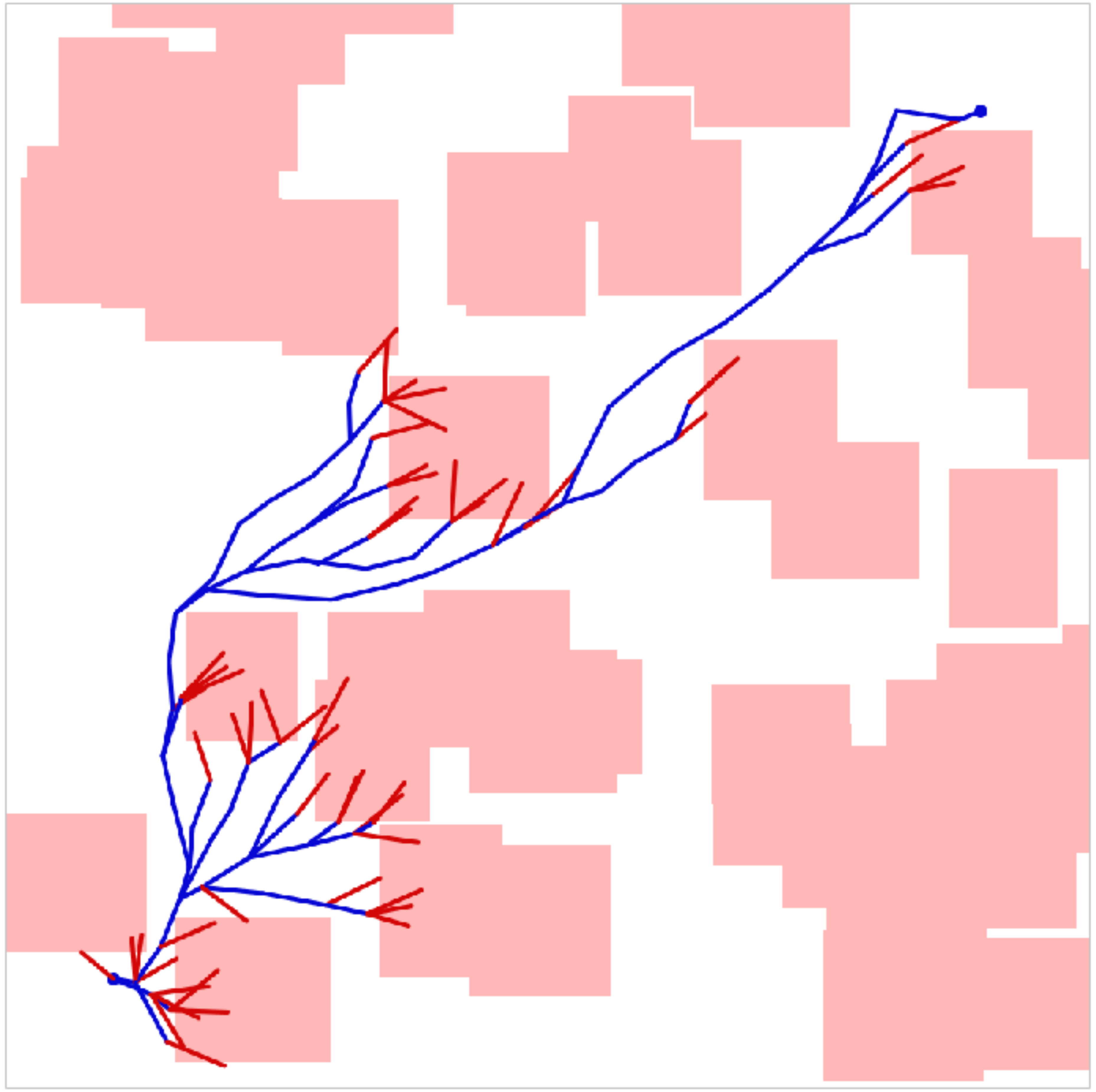


....

LazySP

Optimism Under Uncertainty

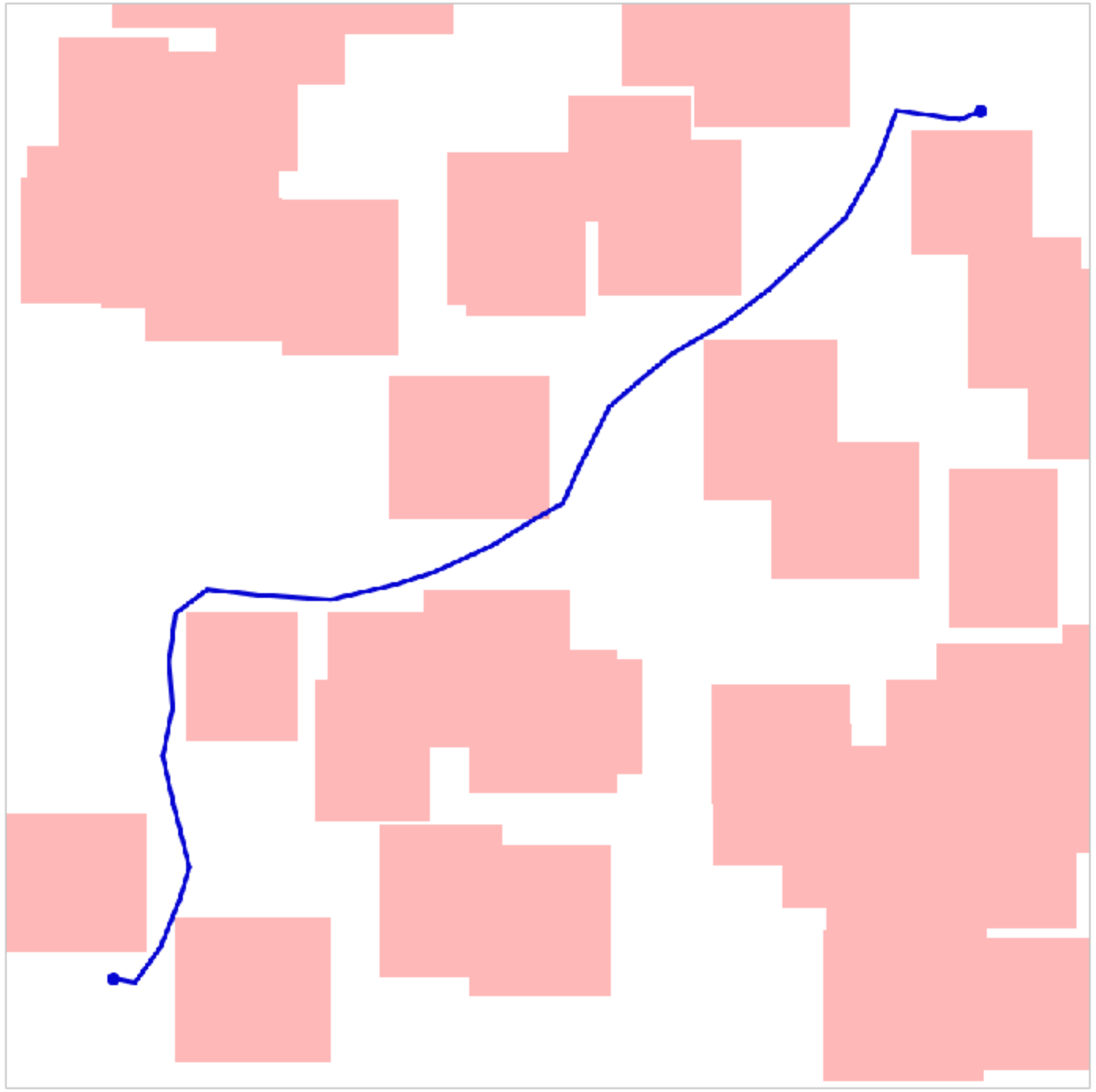




LazySP

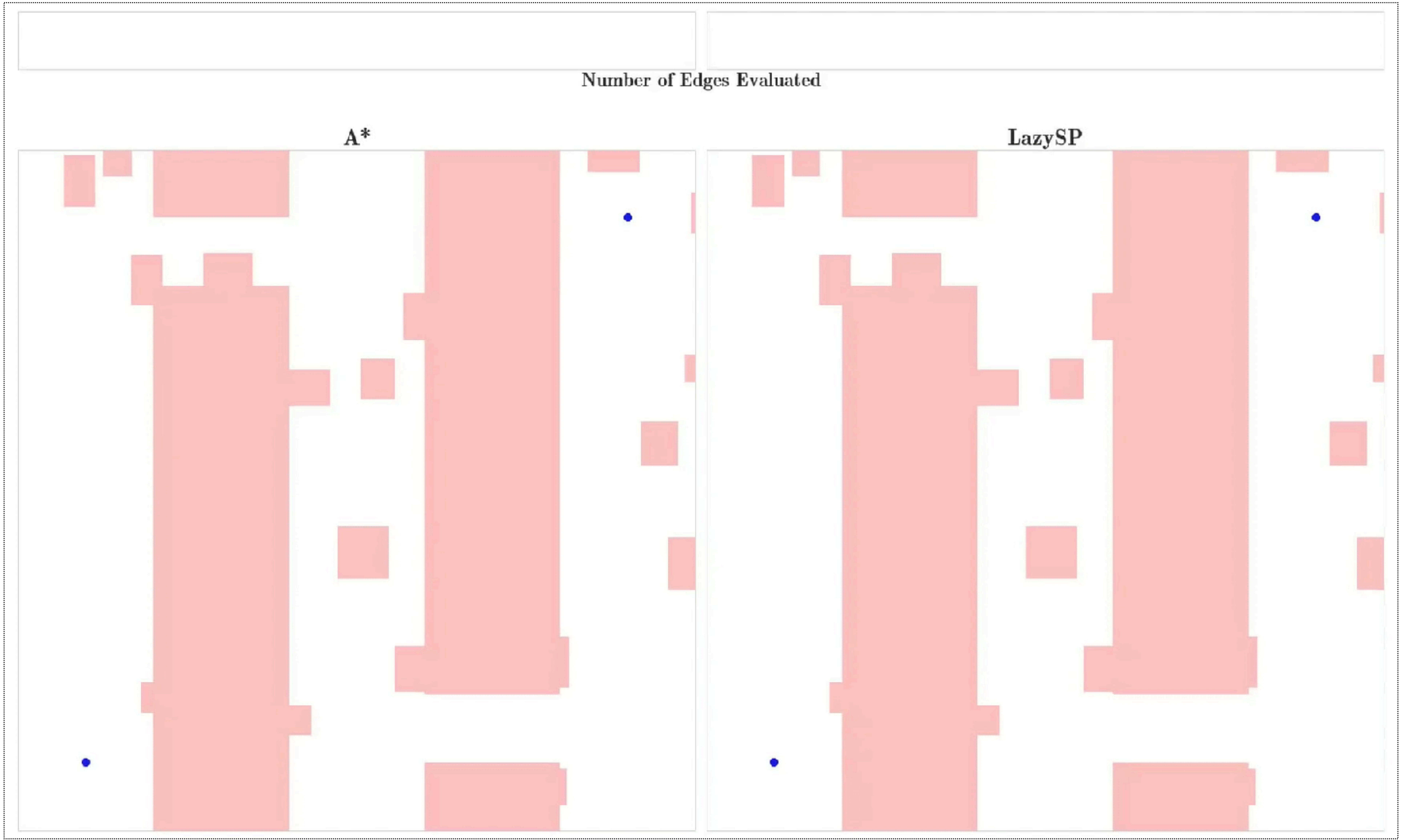
Optimism Under Uncertainty

Return shortest feasible path!

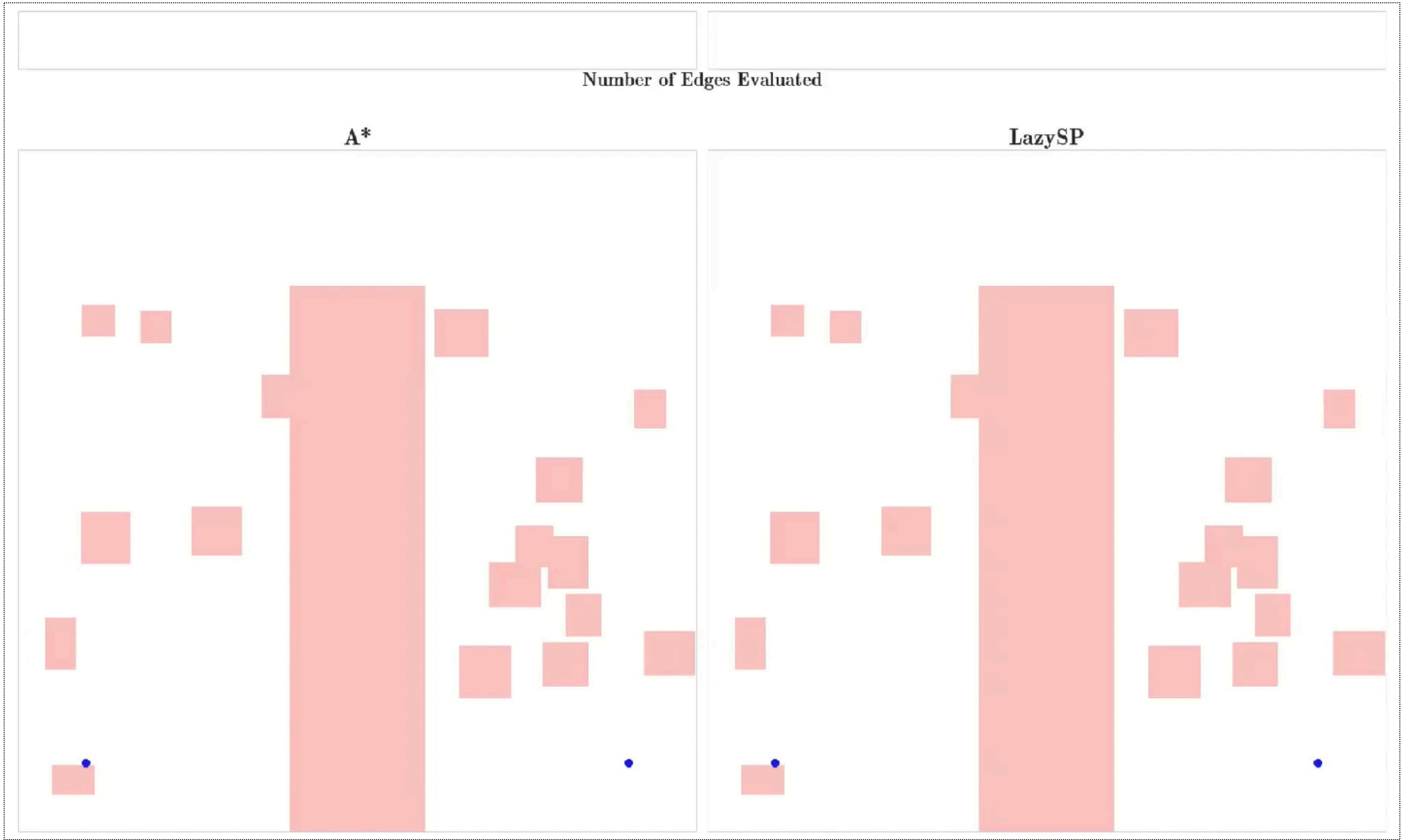


A* vs LazySP

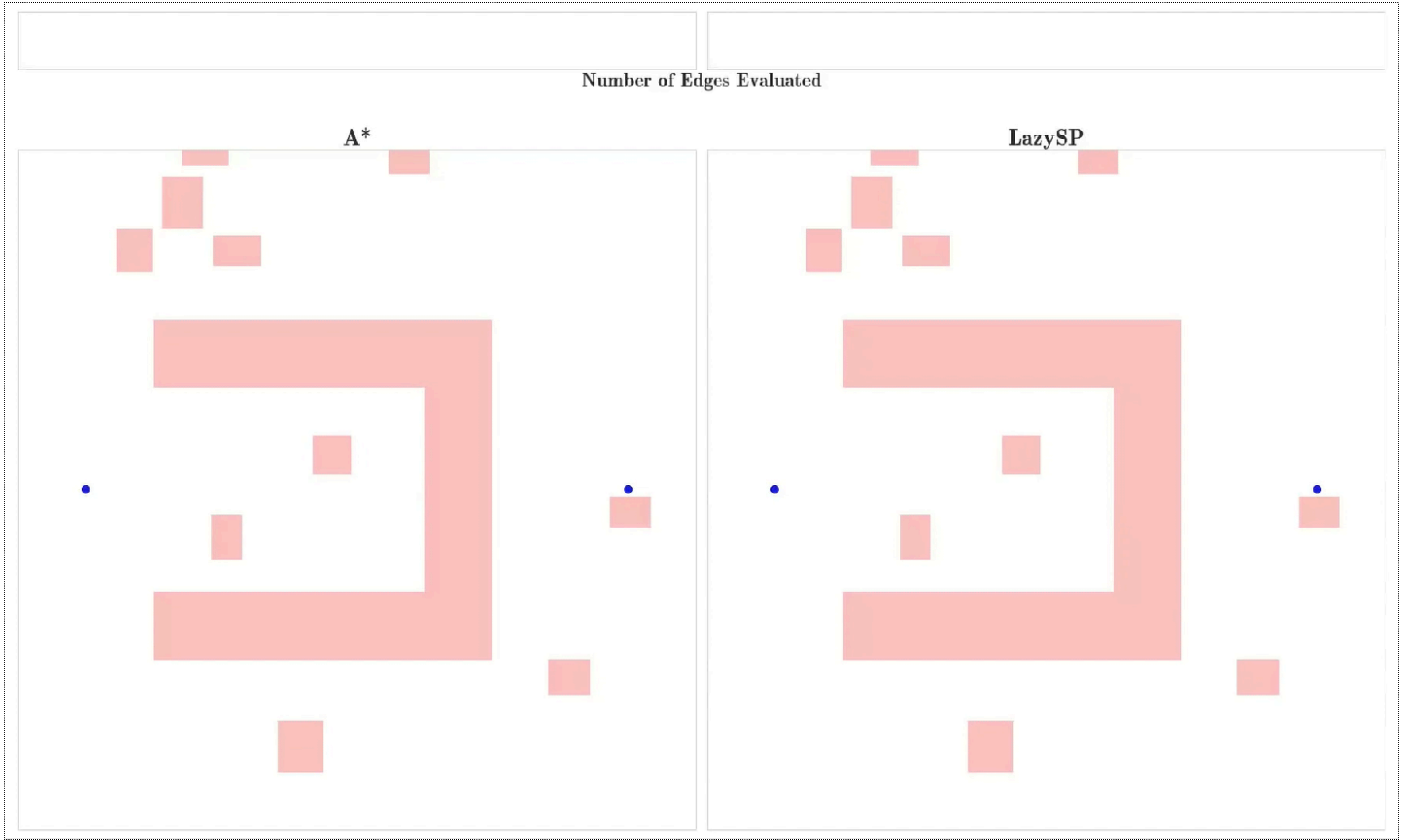
A* vs LazySP



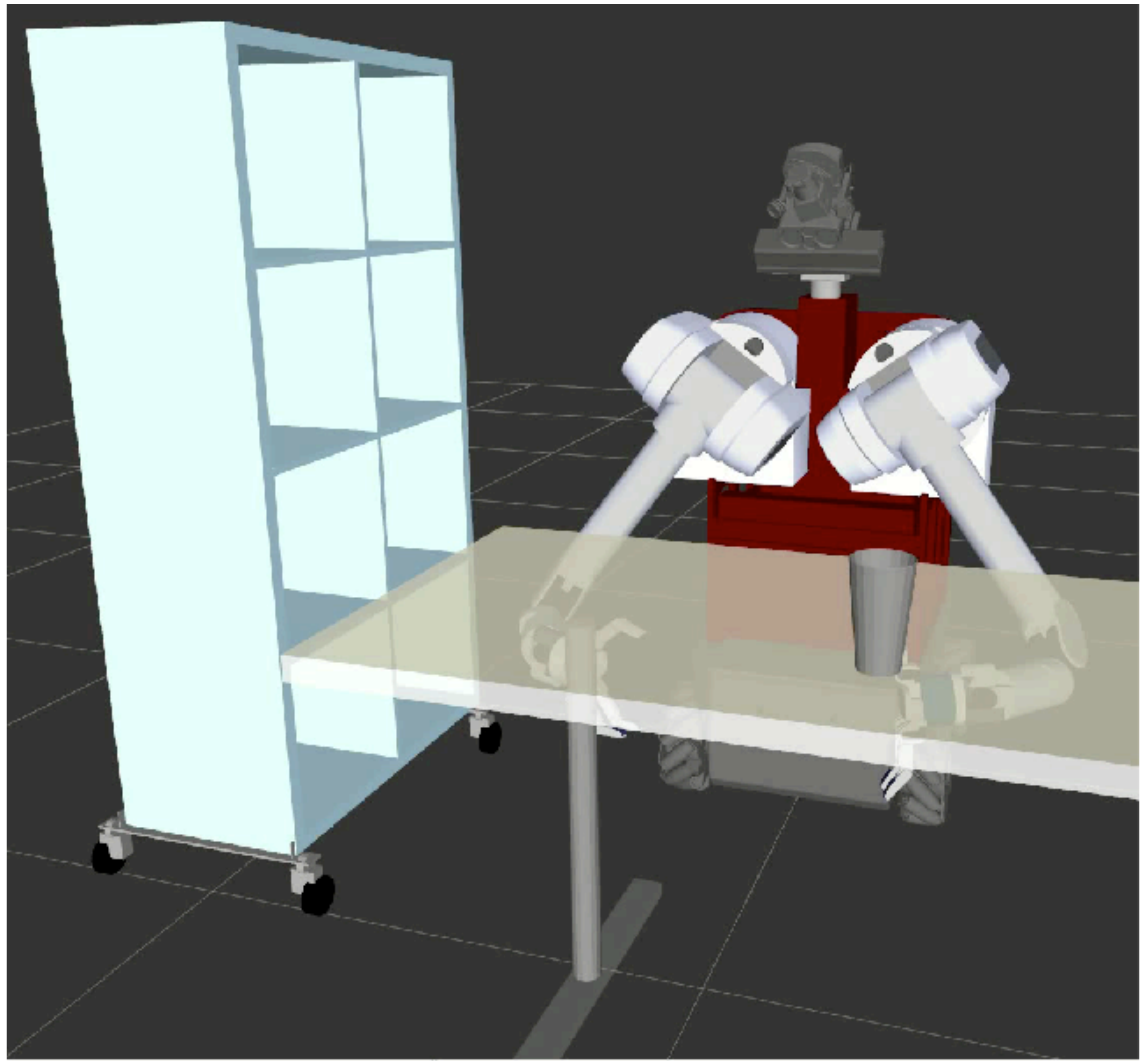
A* vs LazySP



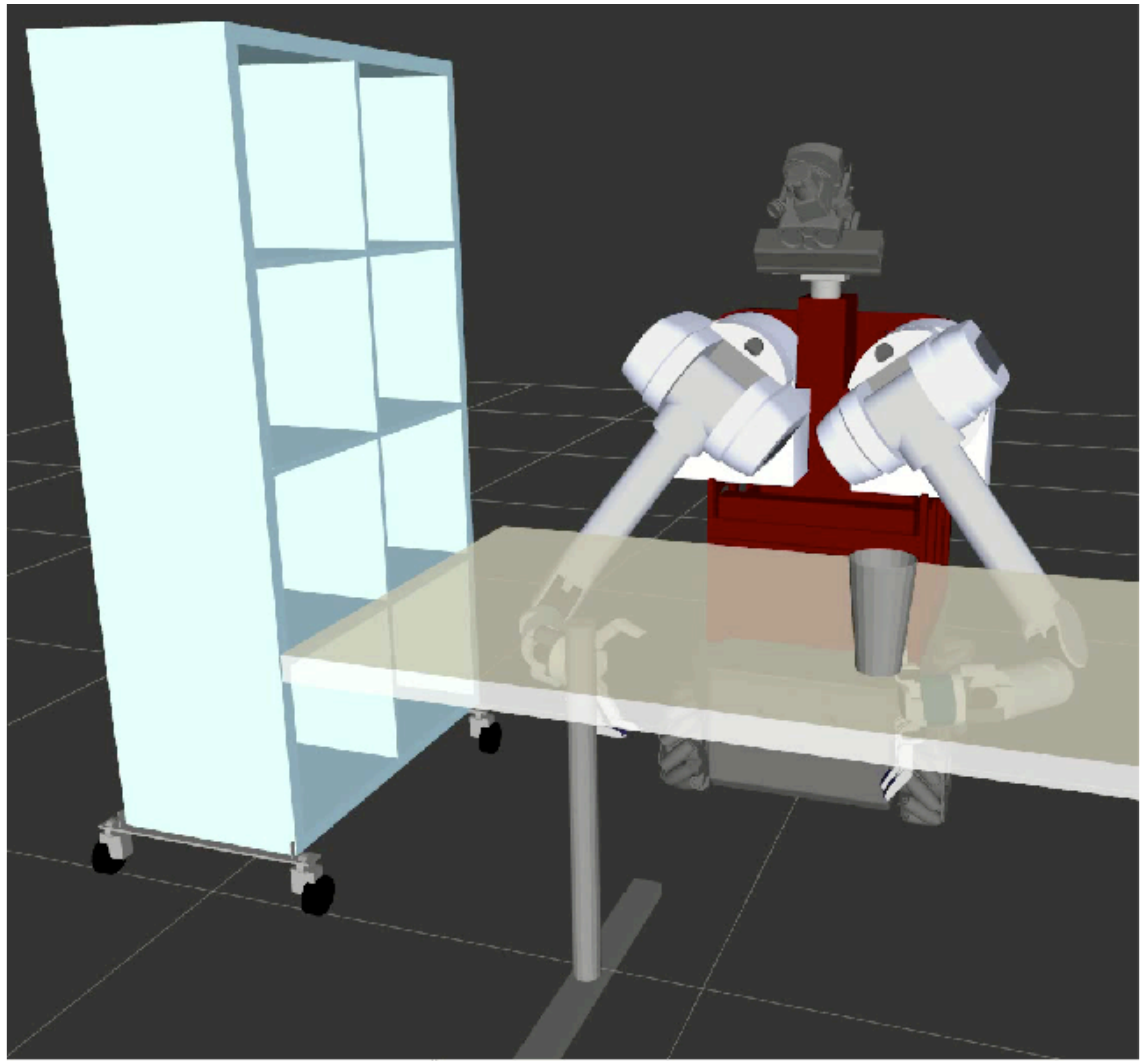
A* vs LazySP



A* vs LazySP

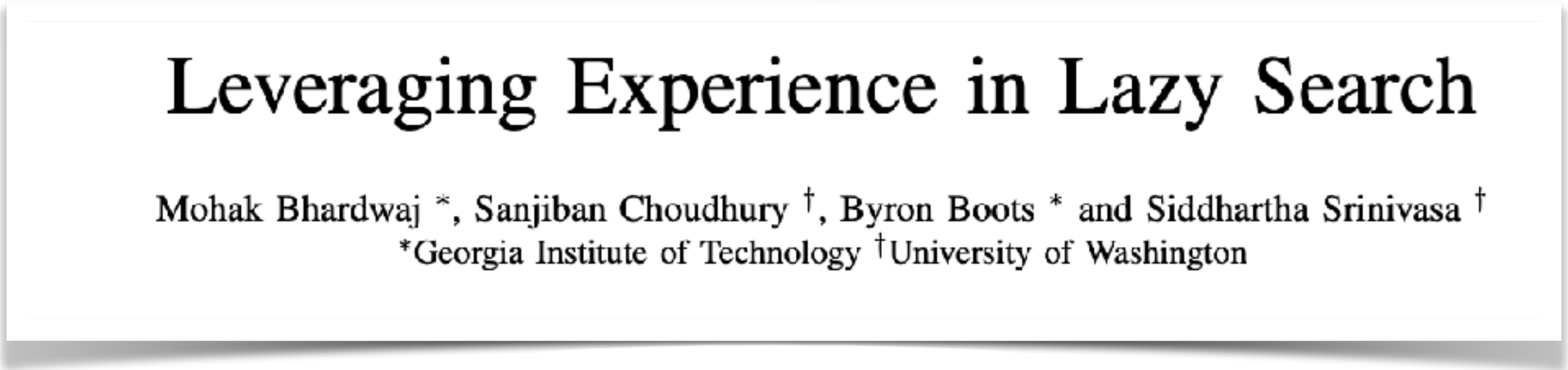


A* (191 edges)



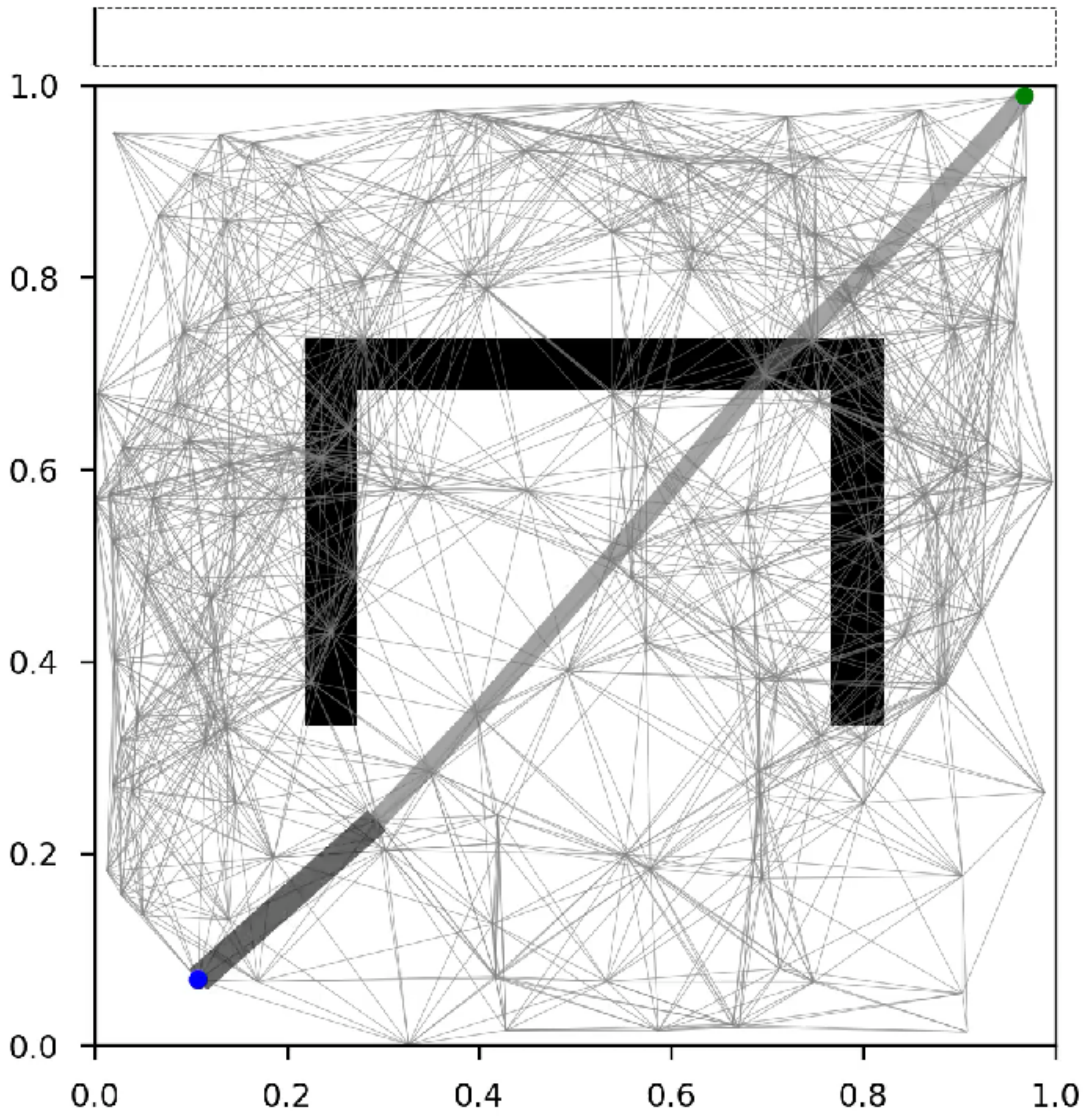
LazySP (38 edges)

How can learning help make LazySP even lazier? (i.e. faster)

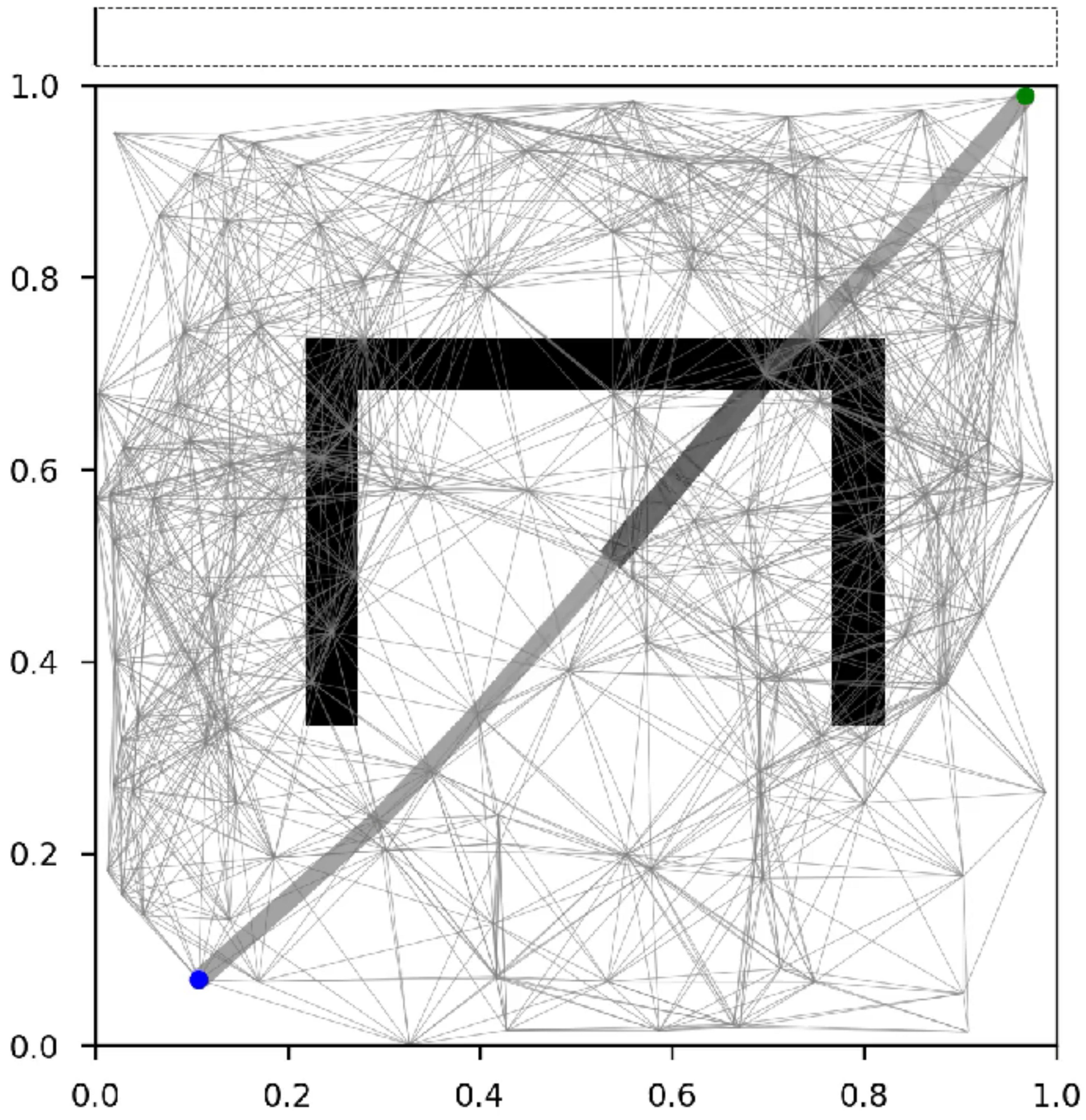

Leveraging Experience in Lazy Search

Mohak Bhardwaj [*], Sanjiban Choudhury [†], Byron Boots [*] and Siddhartha Srinivasa [†]
[*]Georgia Institute of Technology [†]University of Washington

Learn which edges to evaluate (STROLL)

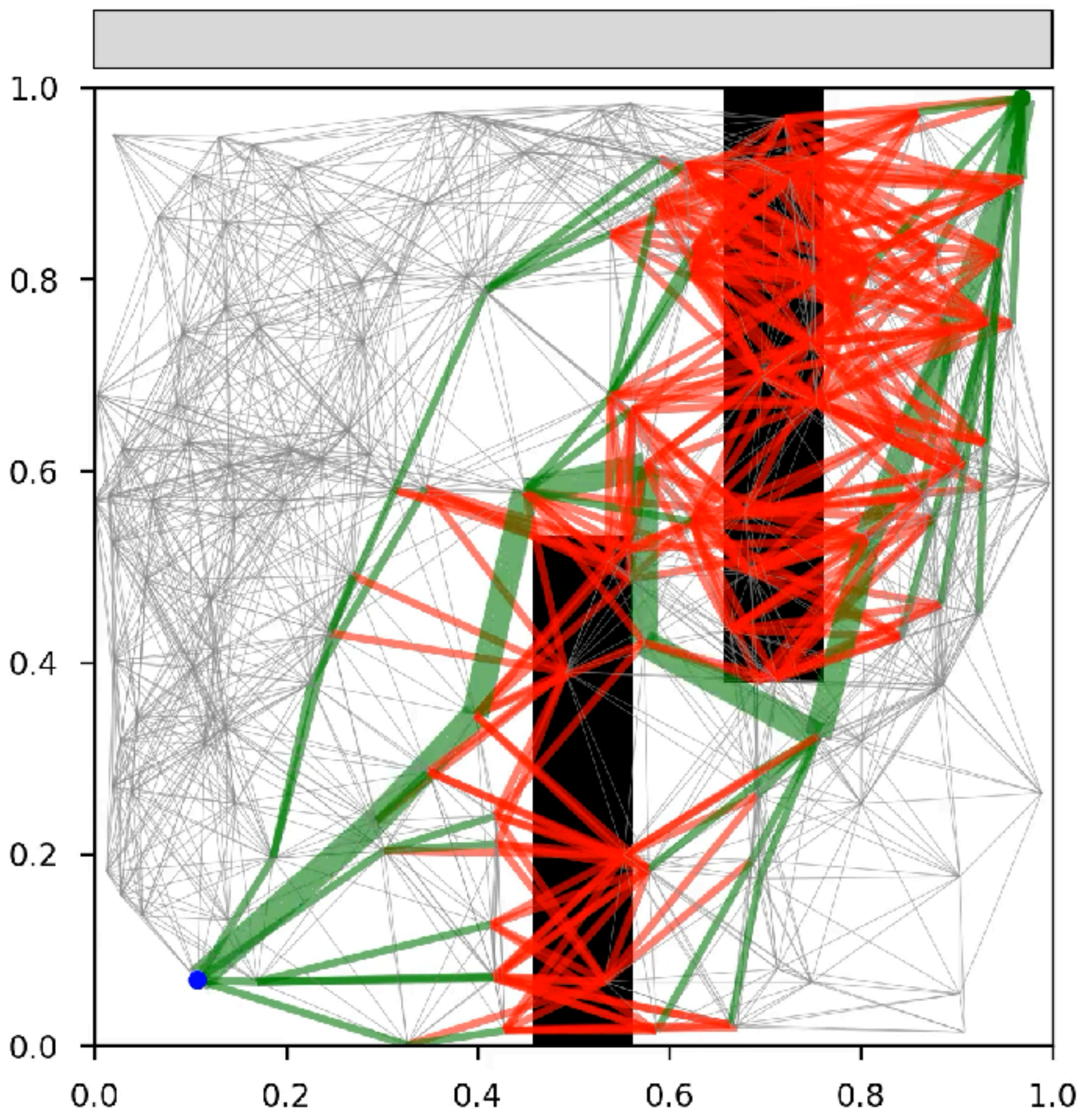


LazySP

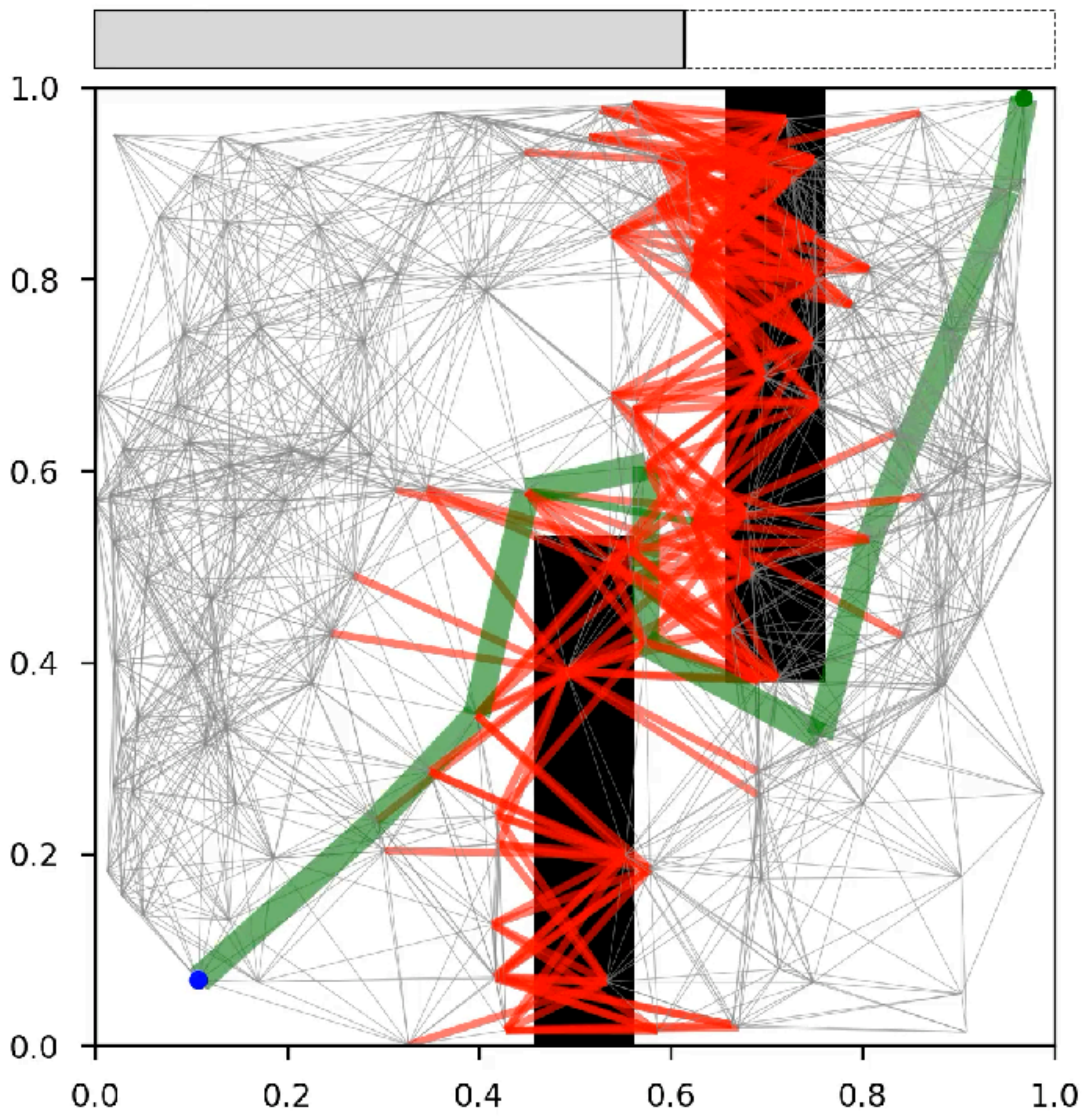


STROLL

Learn which edges to evaluate (STROLL)



LazySP



STROLL

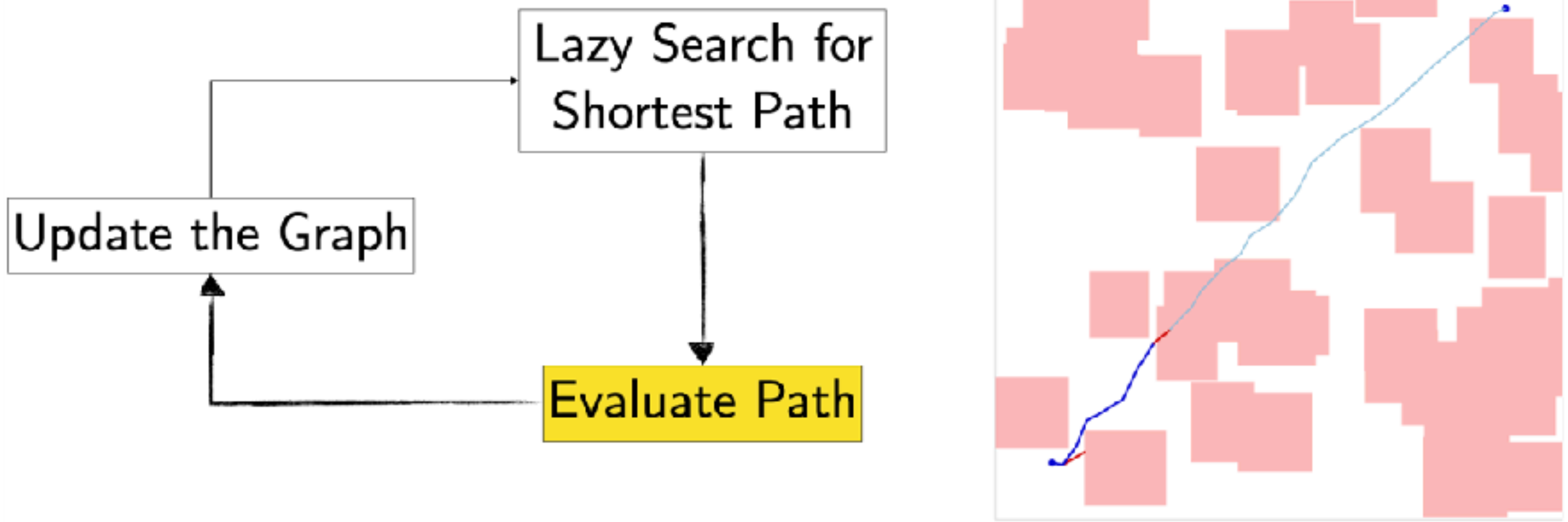
tl;dr

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

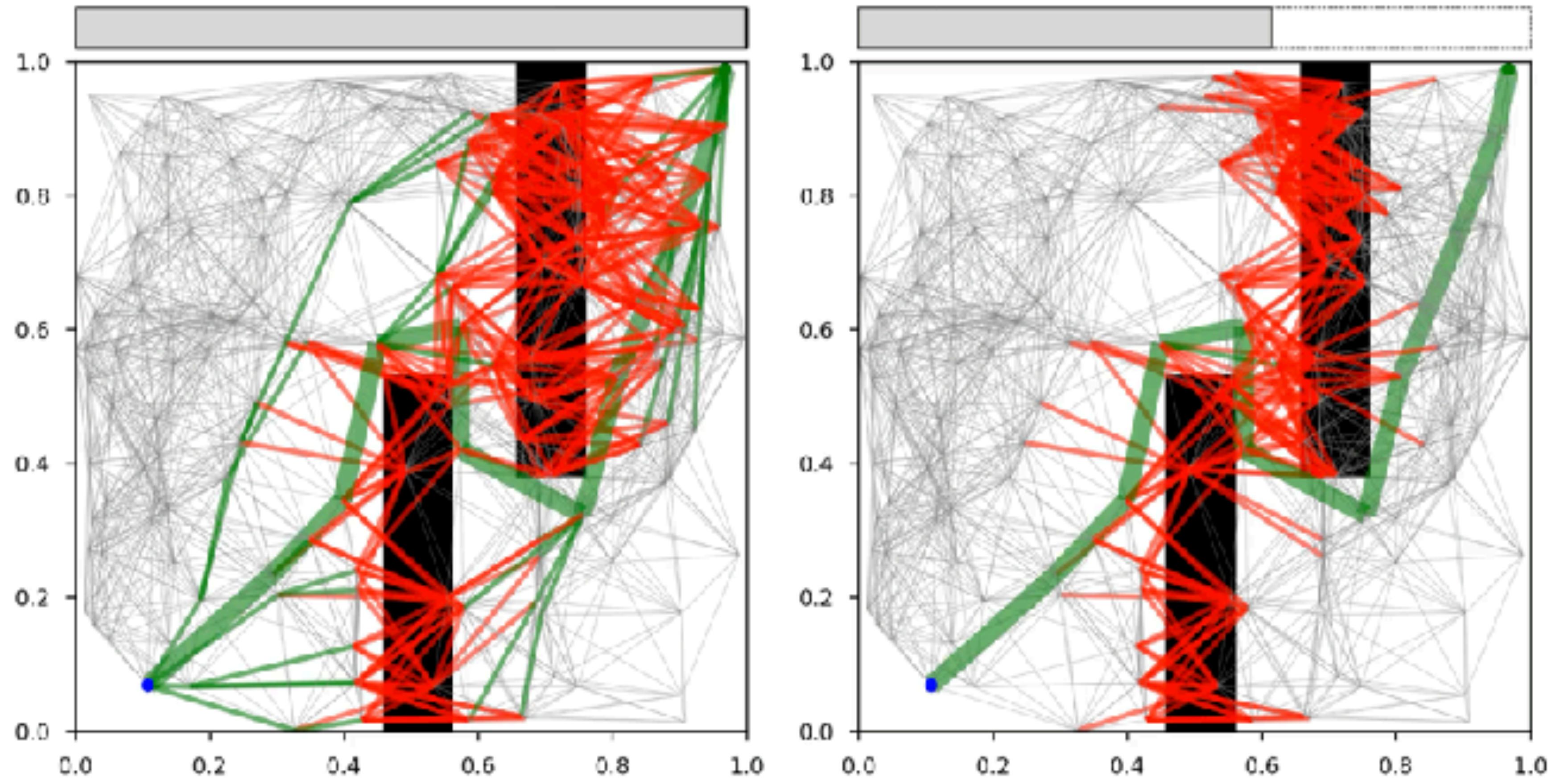
What is the most expensive step?

LazySP

Optimism Under Uncertainty

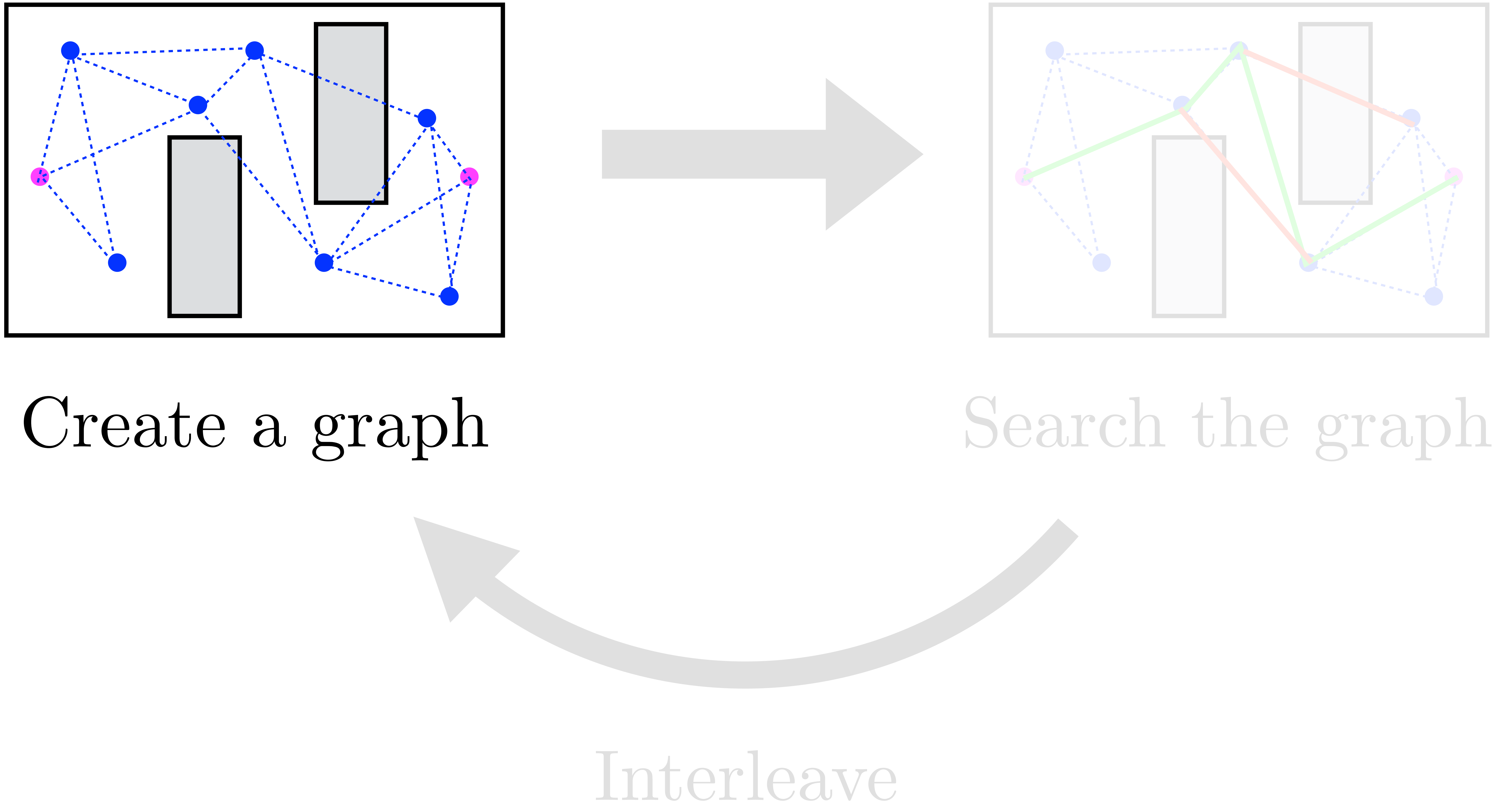


Learn which edges to evaluate (STROLL)



LazySP STROLL

General framework for motion planning



Creating a graph: Abstract algorithm

$$G = (V, E)$$

Vertices: set of configurations

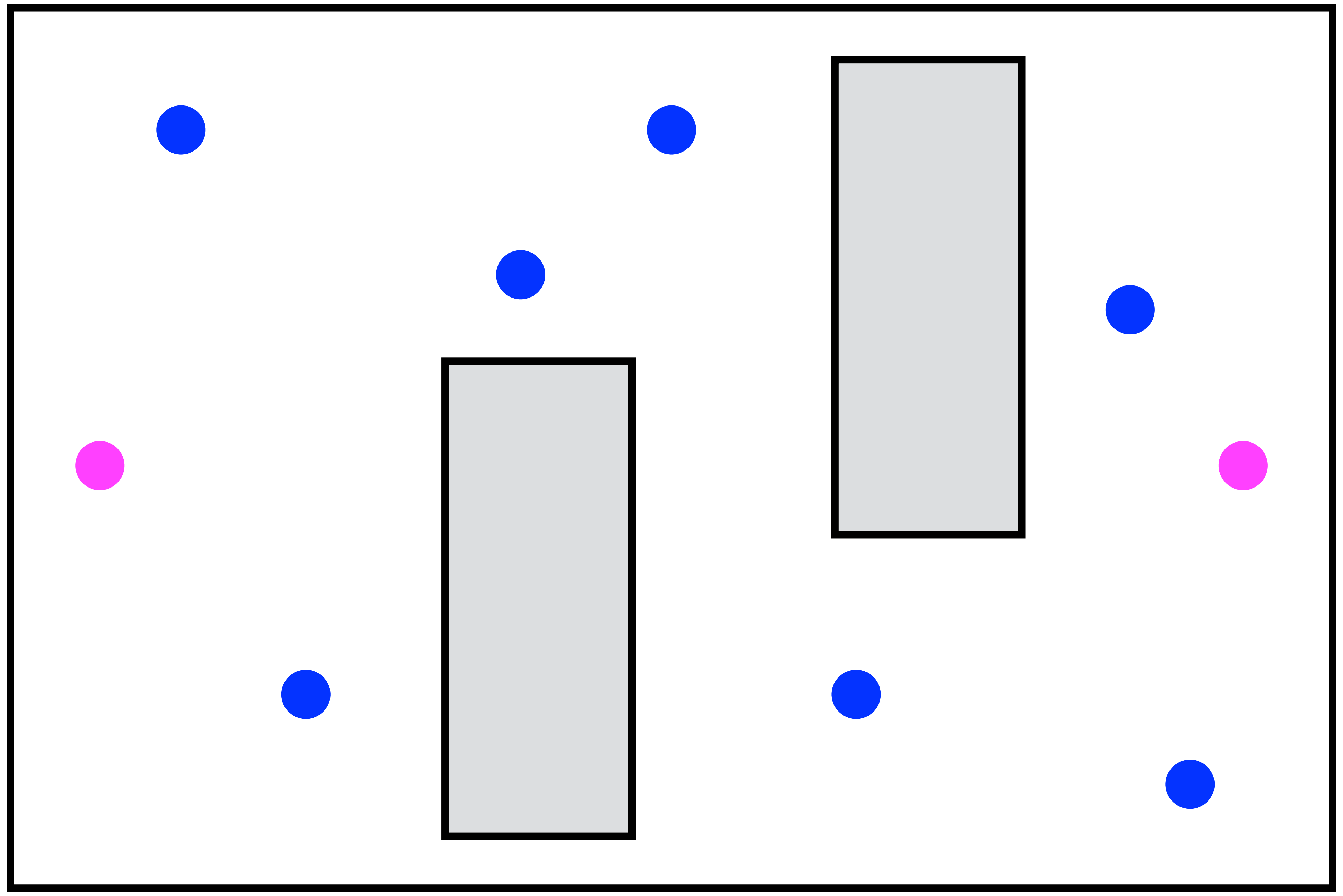
Edges: paths connecting configurations

Creating a graph: Abstract algorithm

$$G = (V, E)$$

Vertices: set of configurations

Edges: paths connecting configurations



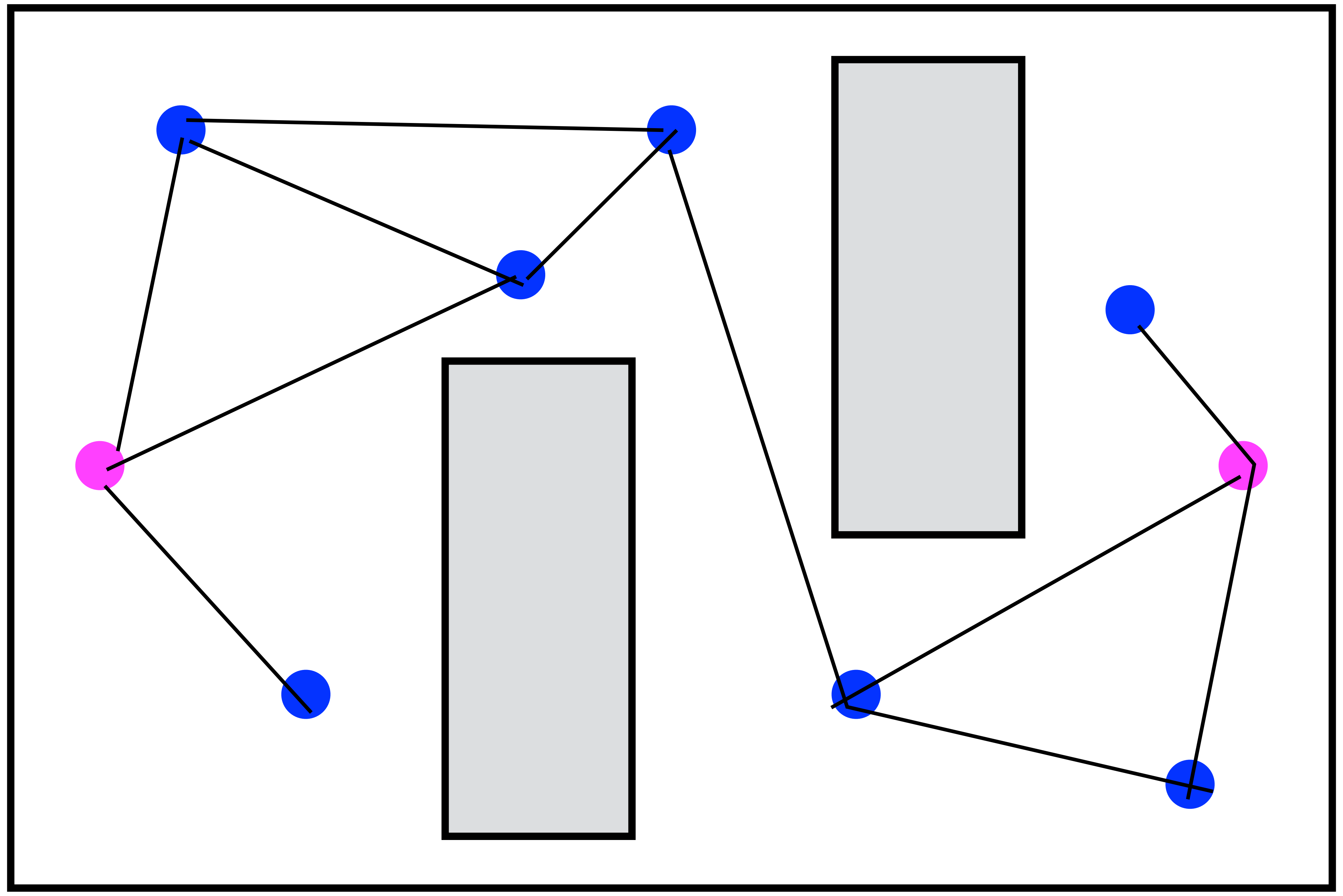
1. Sample a set of collision free vertices V (add start and goal)

Creating a graph: Abstract algorithm

$$G = (V, E)$$

Vertices: set of configurations

Edges: paths connecting configurations

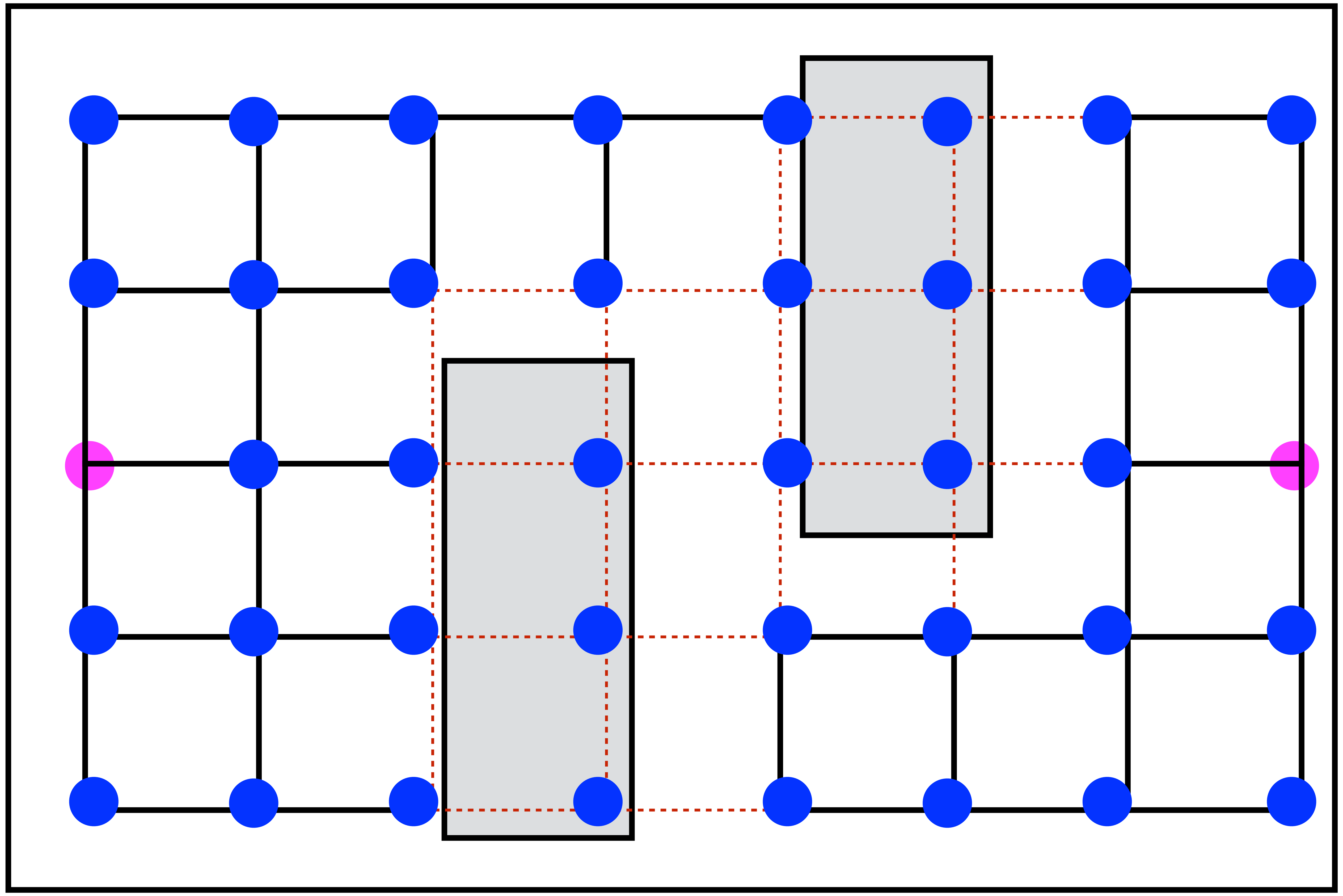


1. Sample a set of collision free vertices V (add start and goal)

2. Connect "neighboring" vertices to get edges E

Strategy 1: Discretize configuration space

Create a lattice. Connect neighboring points (4-conn, 8-conn, ...)

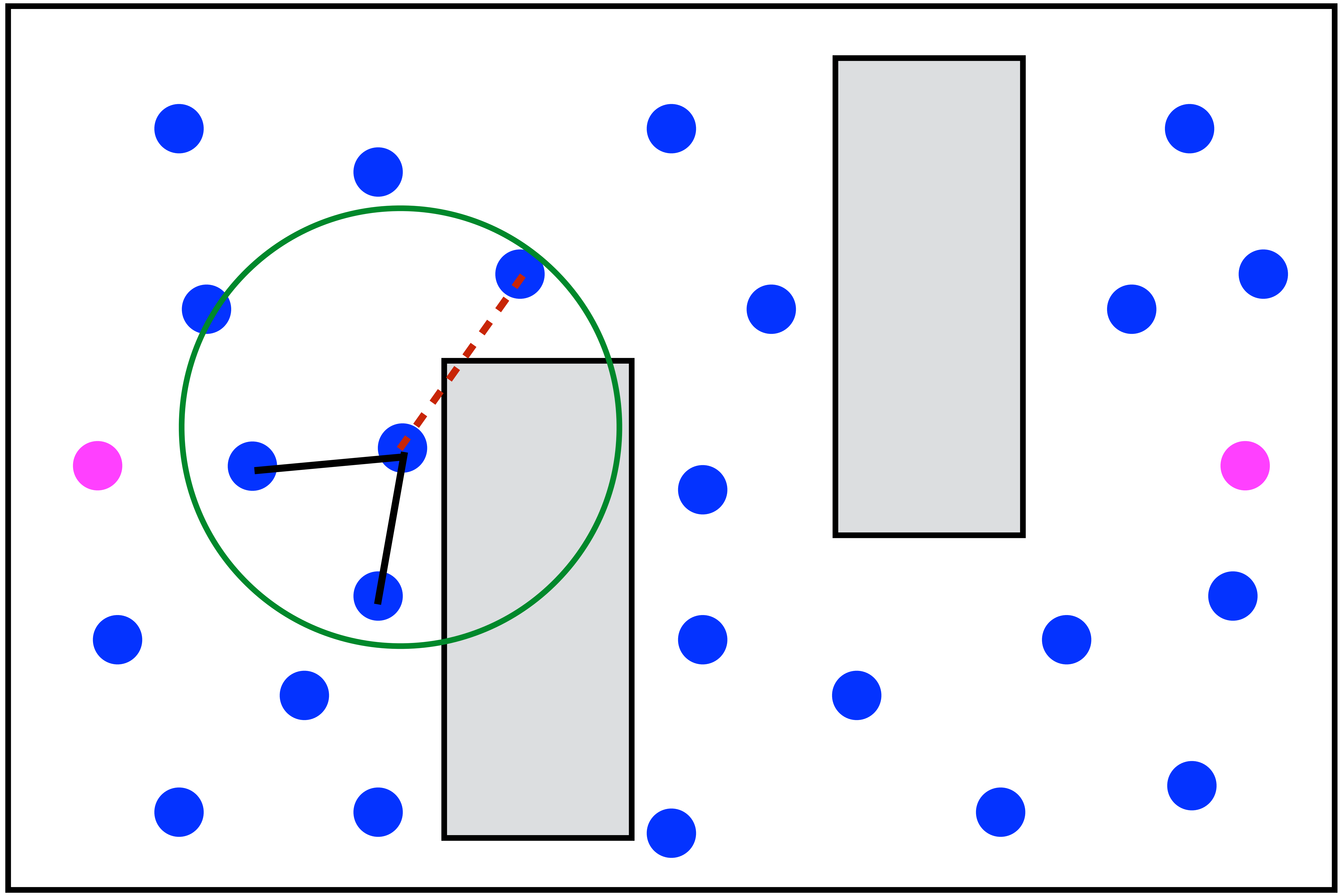


Theoretical guarantees: Resolution complete

What are the pros? What are the cons?

Strategy 2: Uniformly randomly sample

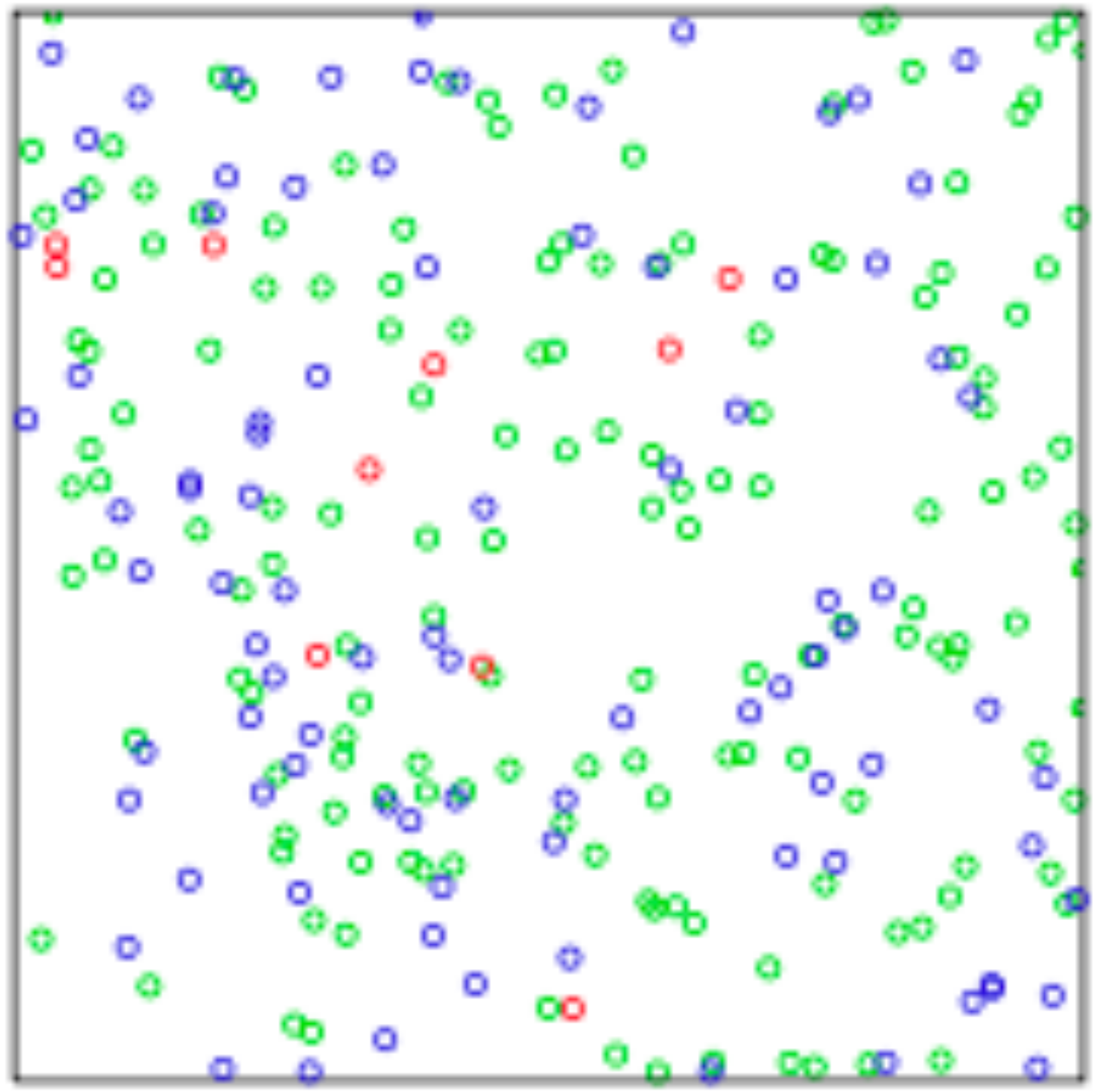
Randomly sample points. Connect all neighbors in a ball!



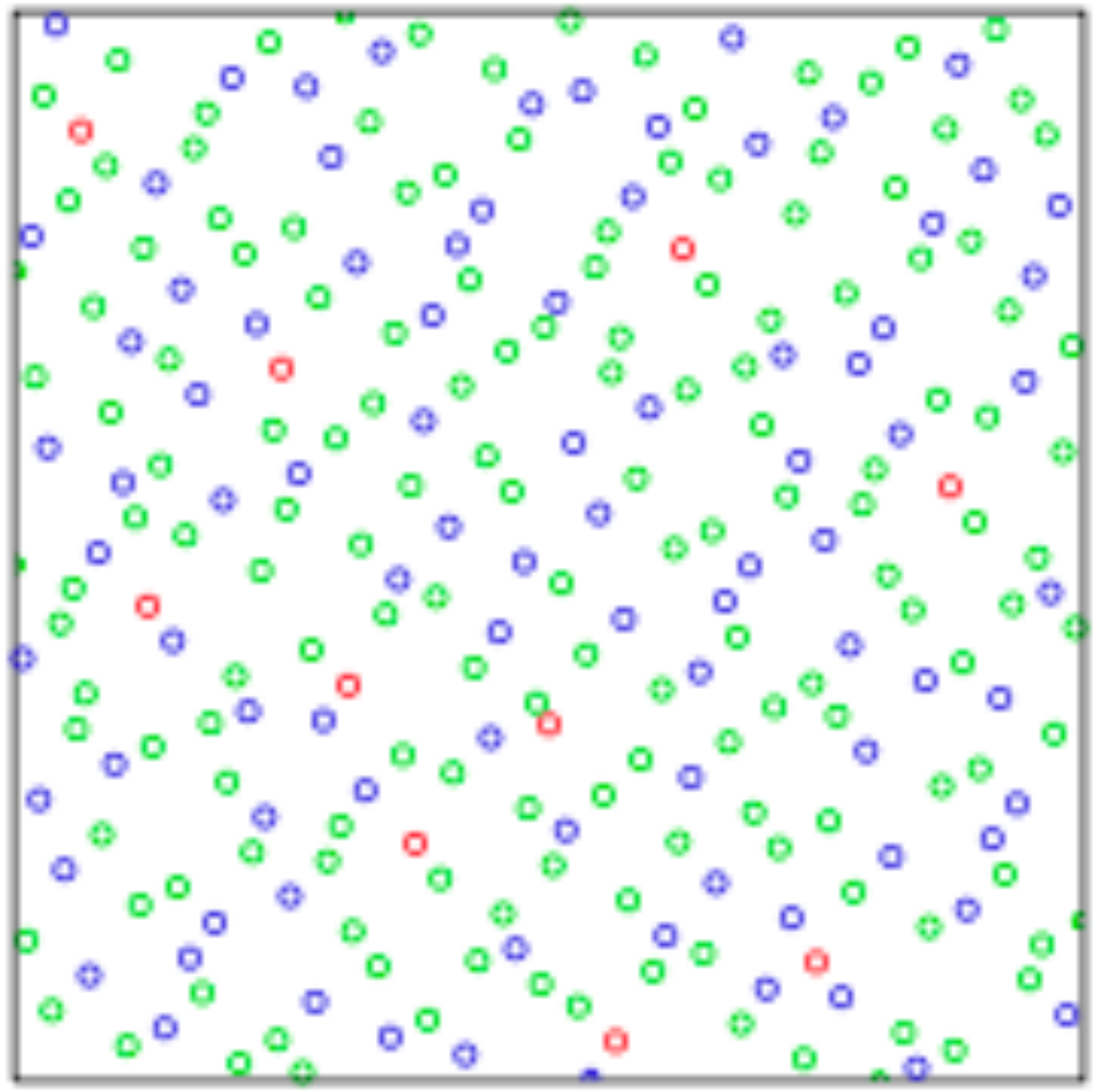
Theoretical guarantees: Probabilistically complete

What are the pros? What are the cons?

Can we do better than random?



Uniform random sampling tends to clump

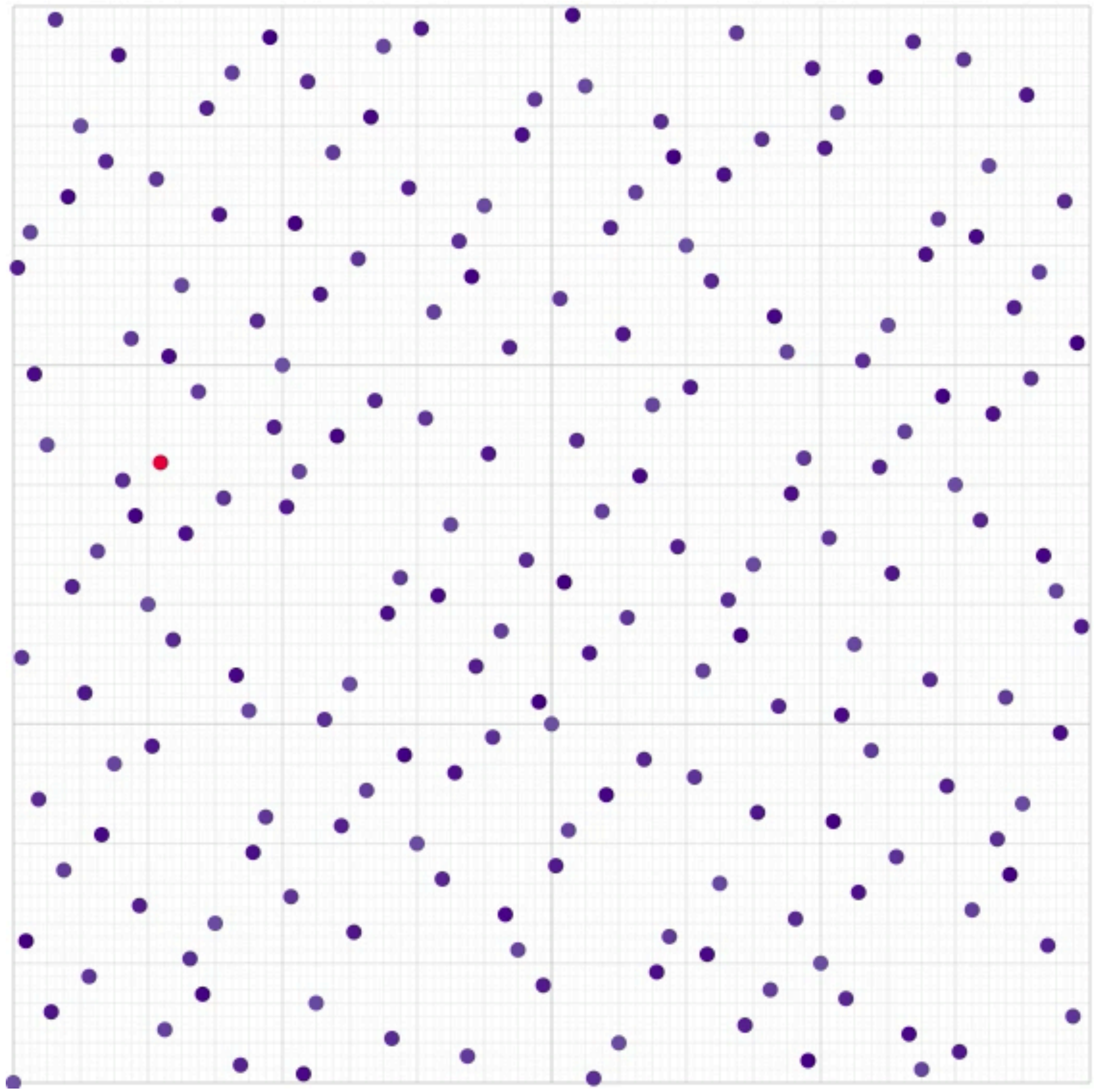


Ideally we would want points to be spread out evenly

Question: How do we do this without discretization?

Halton Sequence

Intuition: Create a sequence using prime numbers that uniformly densify space



Link for exact algorithm:

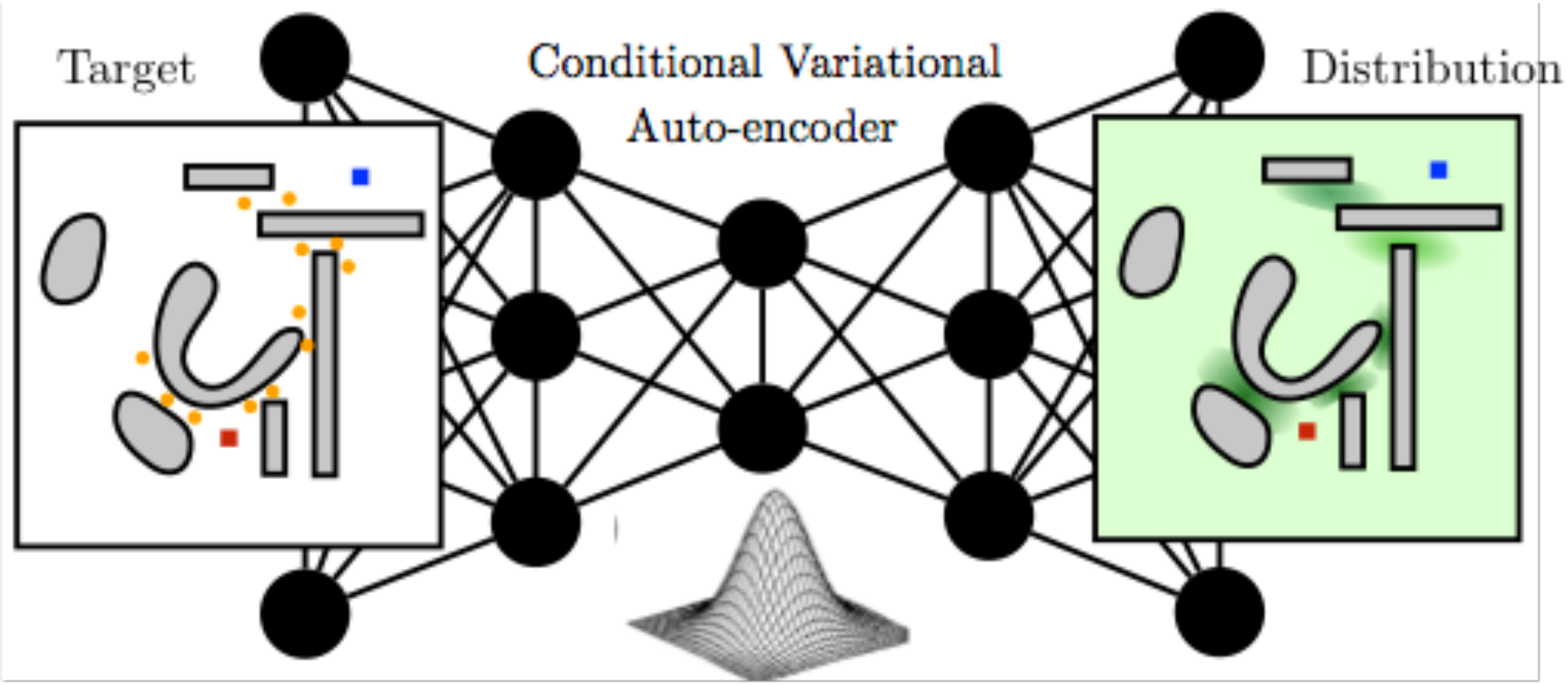
https://observablehq.com/@jrus/halton

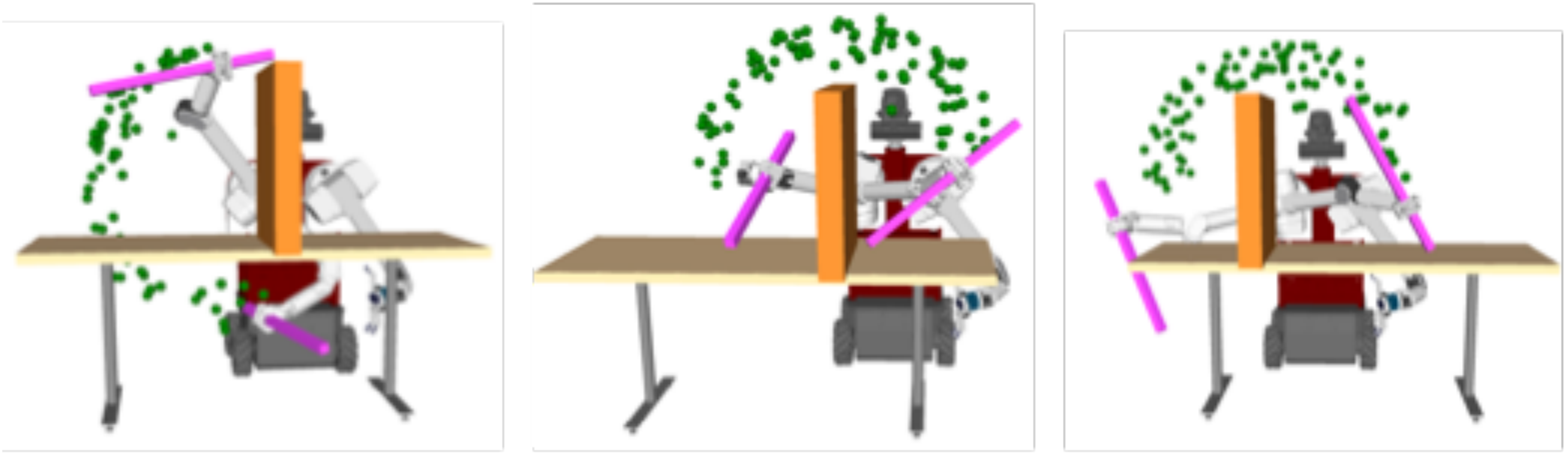
How can learning help make better graphs?

LEGO: Leveraging Experience in Roadmap Generation for Sampling-Based Planning

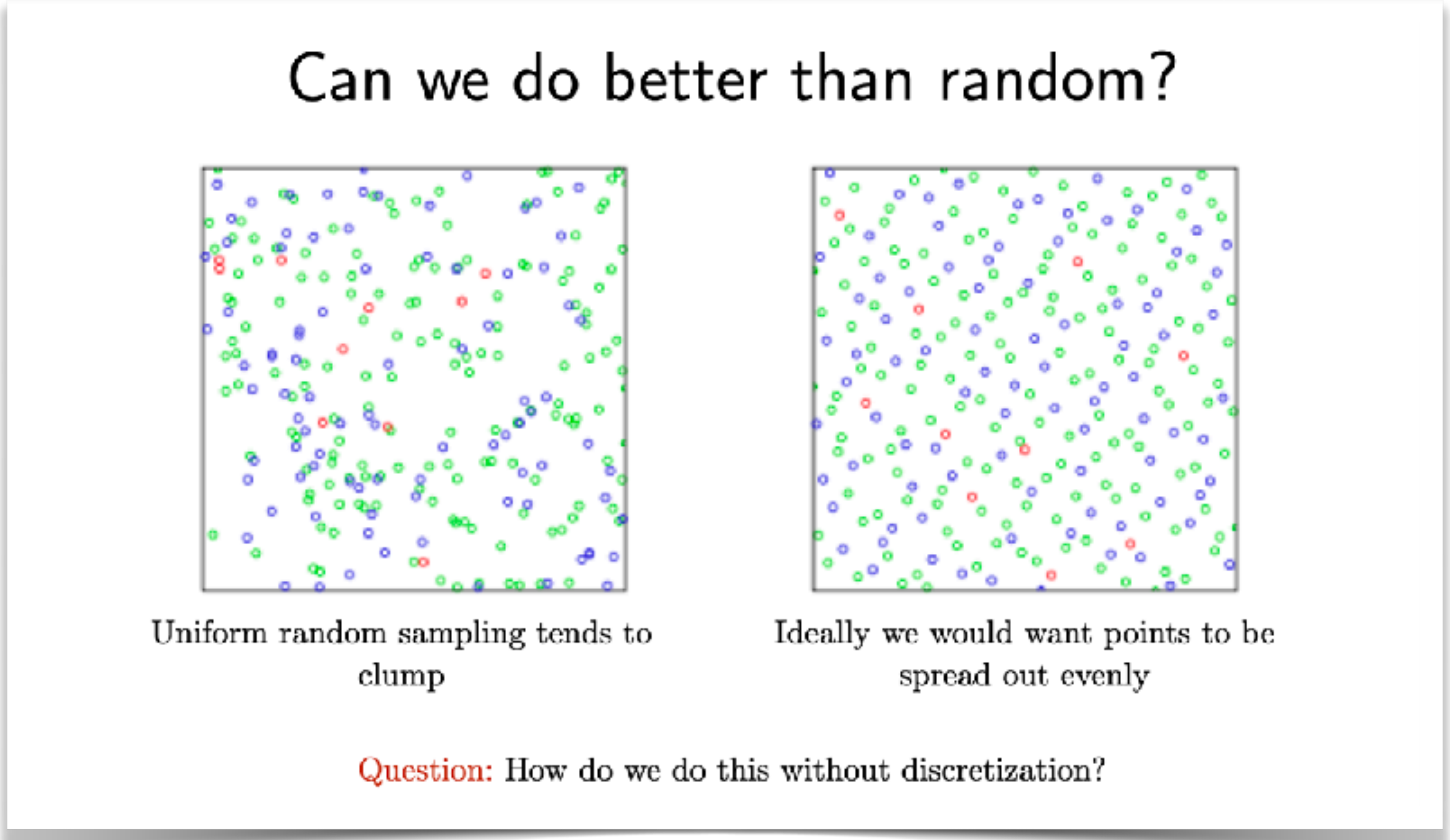
Rahul Kumar*[1], Aditya Mandalika*[2], Sanjiban Choudhury*[2] and Siddhartha S. Srinivasa*[2]

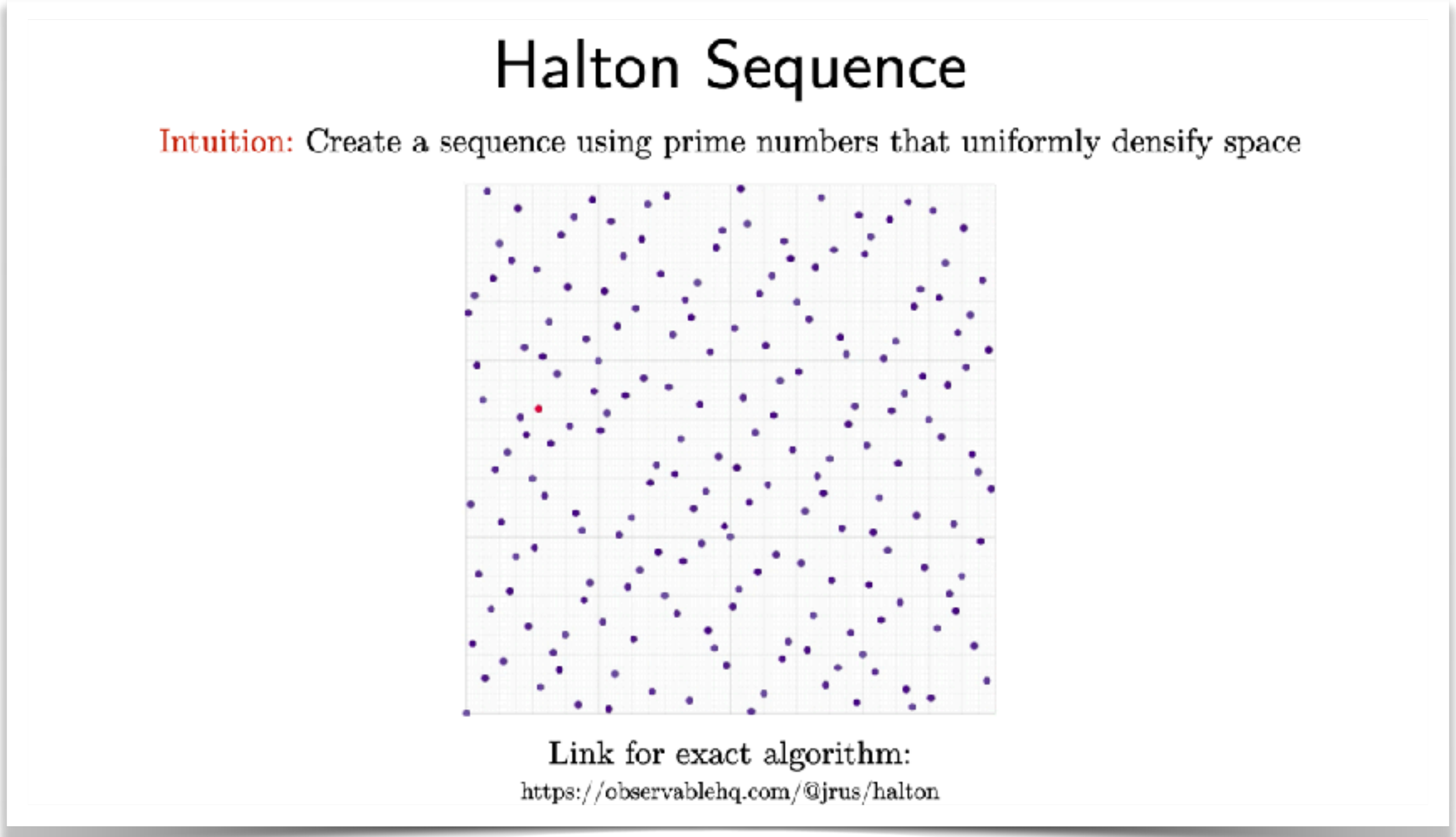
Learning a Sampler (LEGO)

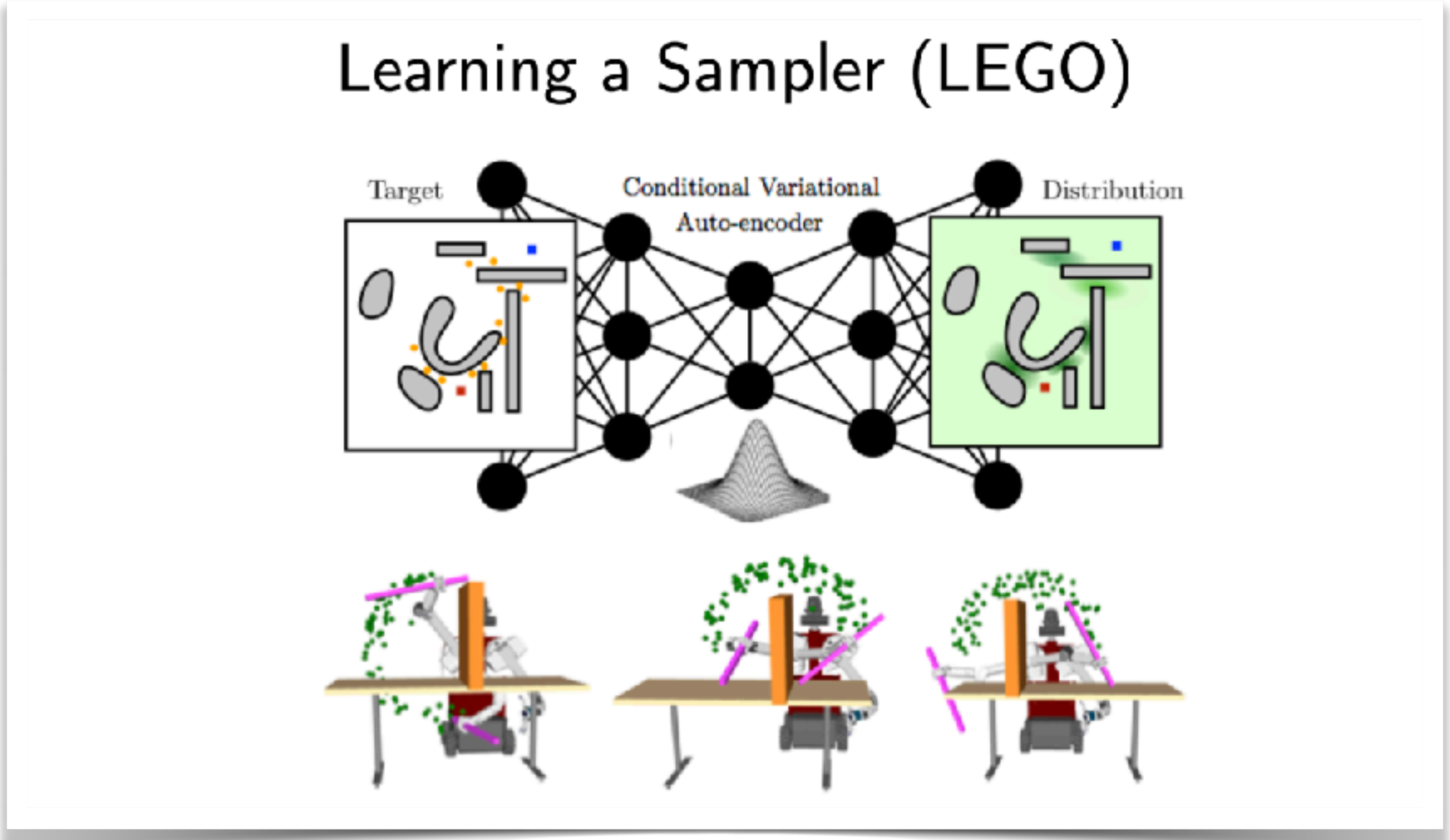




tl;dr


Can we do better than random?
Uniform random sampling tends to clump
Ideally we would want points to be spread out evenly
Question: How do we do this without discretization?


Halton Sequence
Intuition: Create a sequence using prime numbers that uniformly densify space
Link for exact algorithm:
https://observablehq.com/@jrus/halton


Learning a Sampler (LEGO)
Target
Conditional Variational Auto-encoder
Distribution