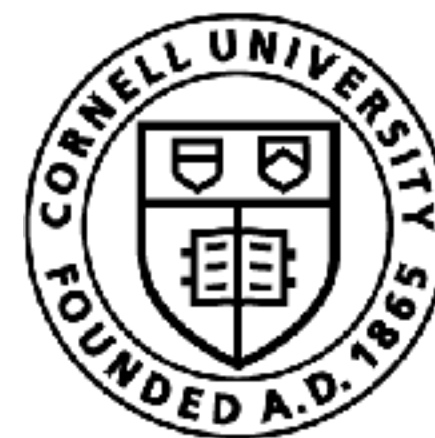
Conquering Motion Planning via Sampling and Search

Sanjiban Choudhury



Cornell Bowers CIS
Computer Science

Recap

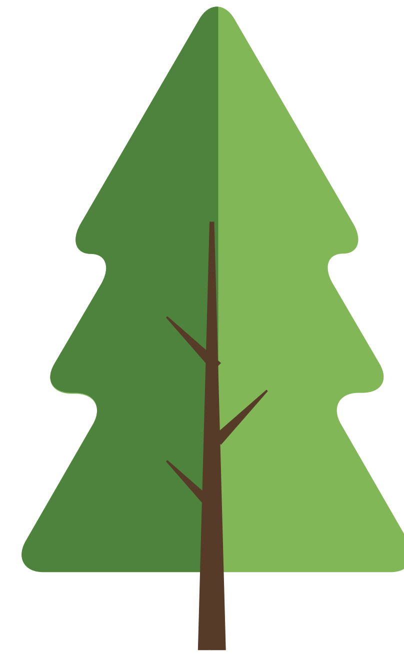
We saw how LQR gives us the optimal policy for linear, quadratic costs

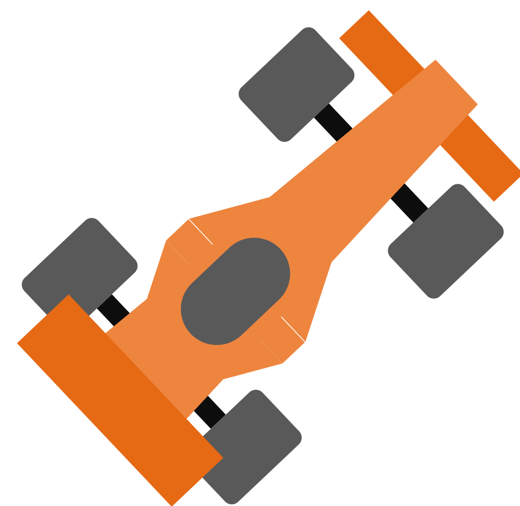
But how can we use LQR for general problems?

LQR for a *non-linear*, *non-quadratic* MDP

Cost

$\exp(-(x - x_{tree})^2 - (y - y_{tree})^2)$

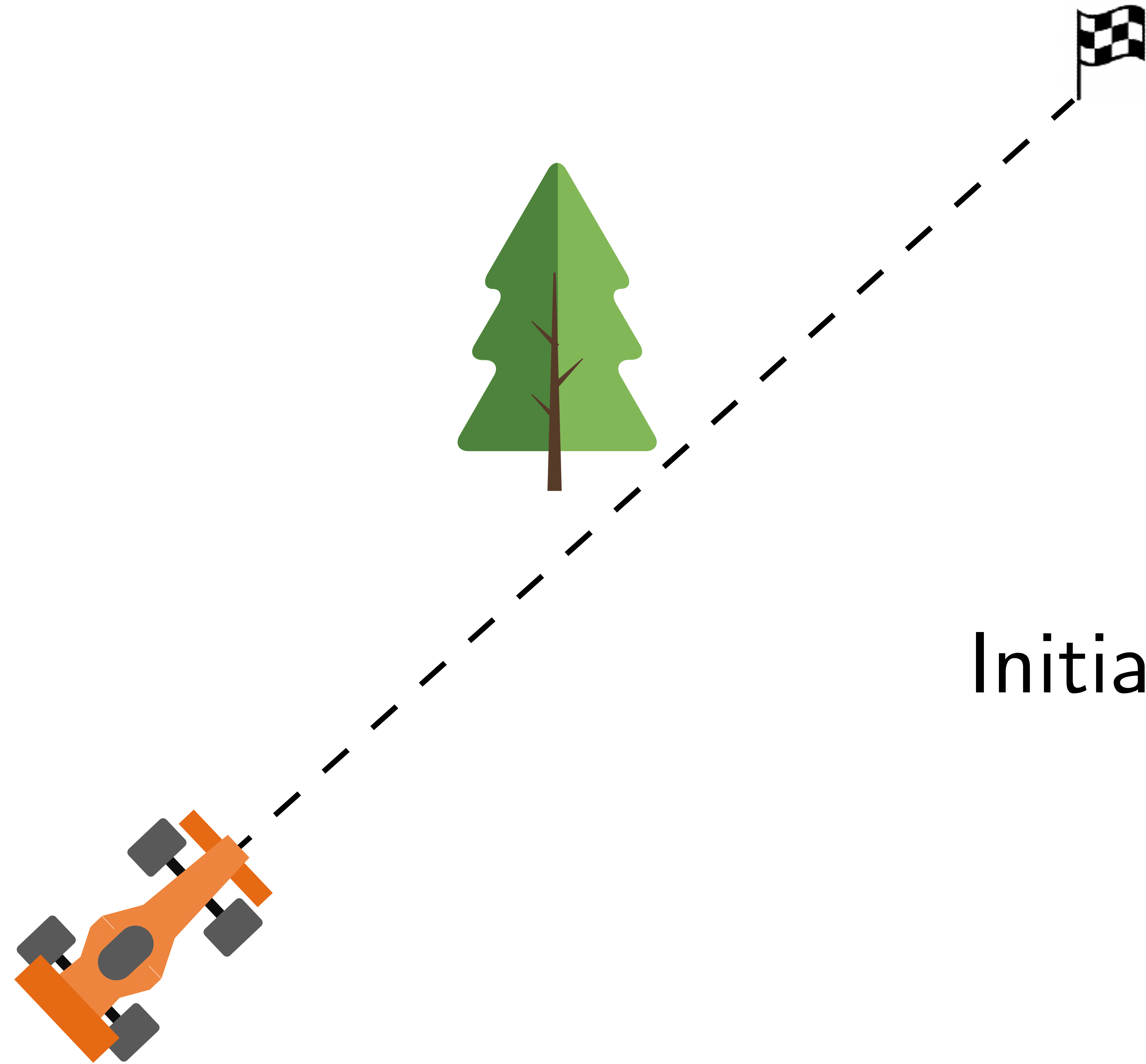




$$\dot{x} = u_s \cos\theta$$
$$\dot{y} = u_s \sin\theta$$
$$\dot{\theta} = u_\omega.$$

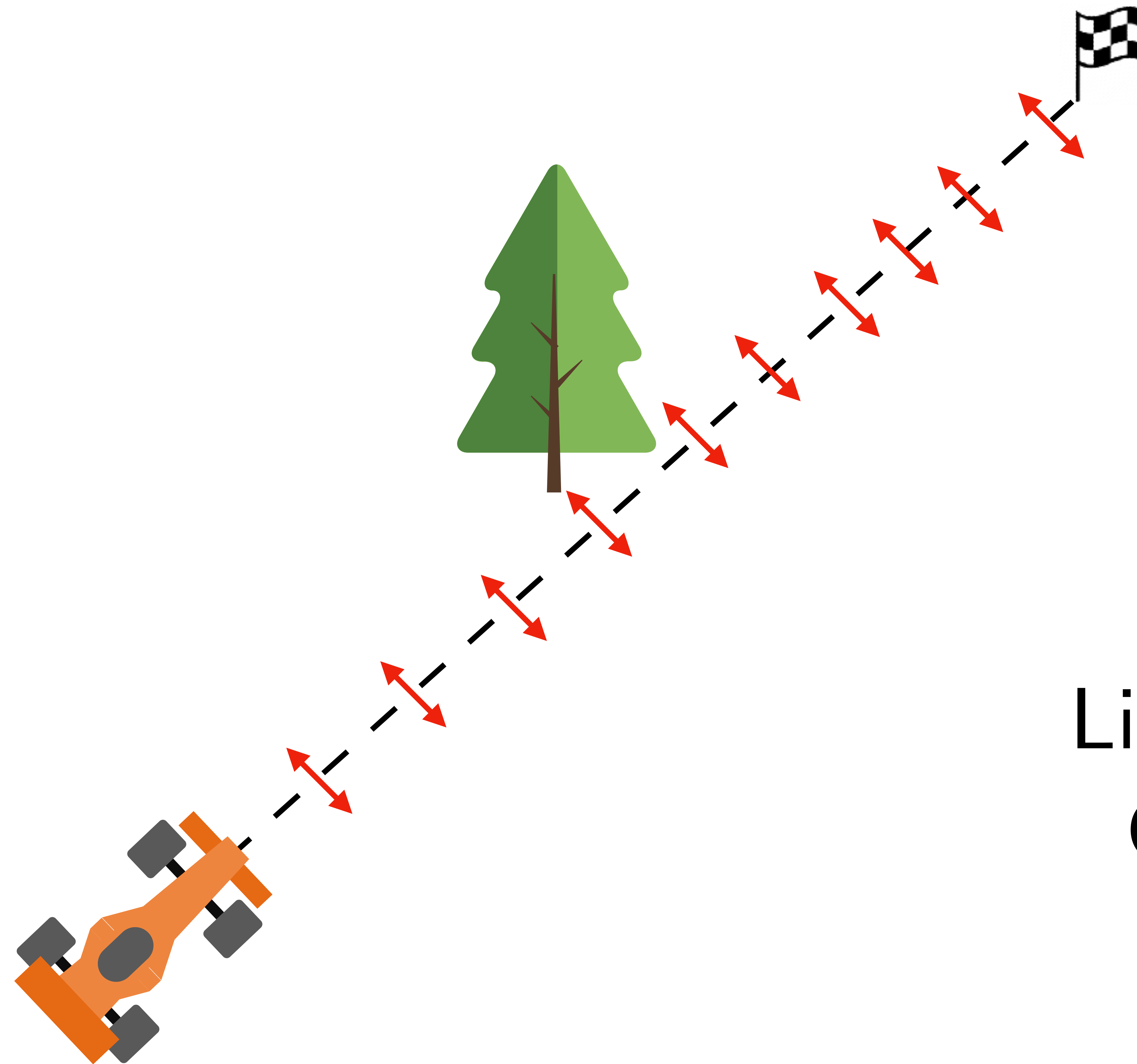
Dynamics

LQR for a *non-linear*, *non-quadratic* MDP



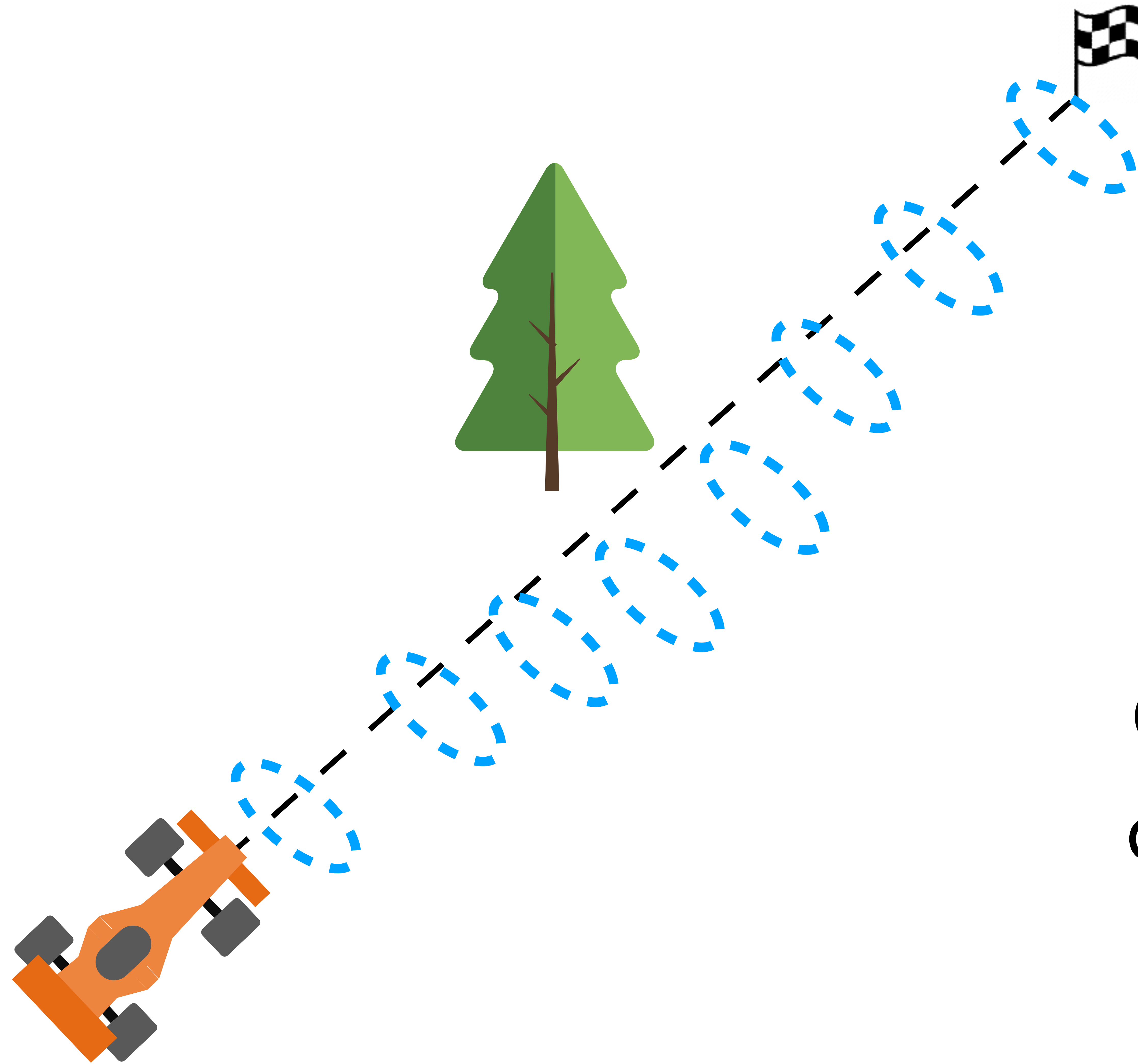
Initialize with a sequence of actions

LQR for a *non-linear*, *non-quadratic* MDP



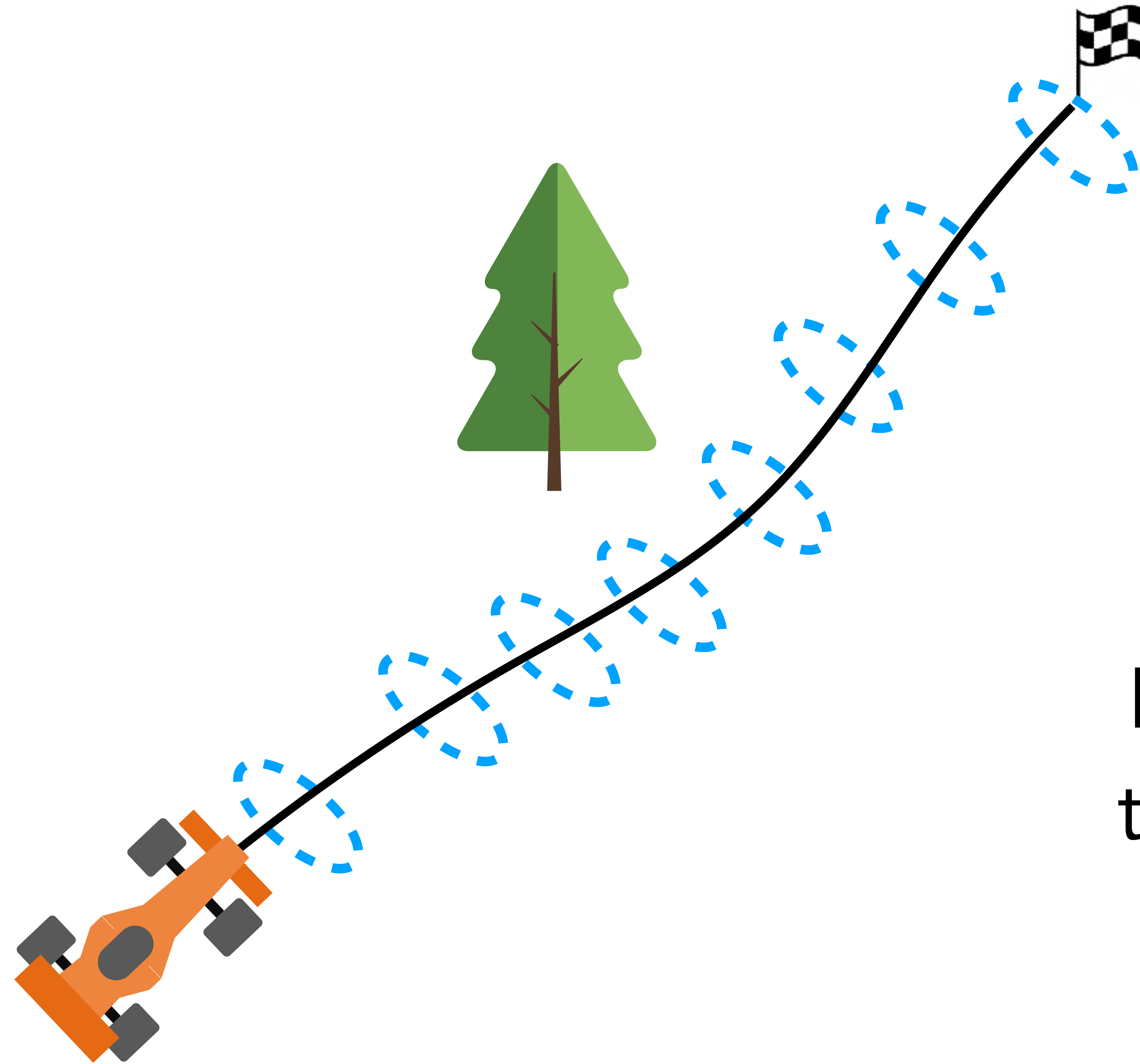
Linearize dynamics,
Quadricize costs

LQR for a *non-linear*, *non-quadratic* MDP



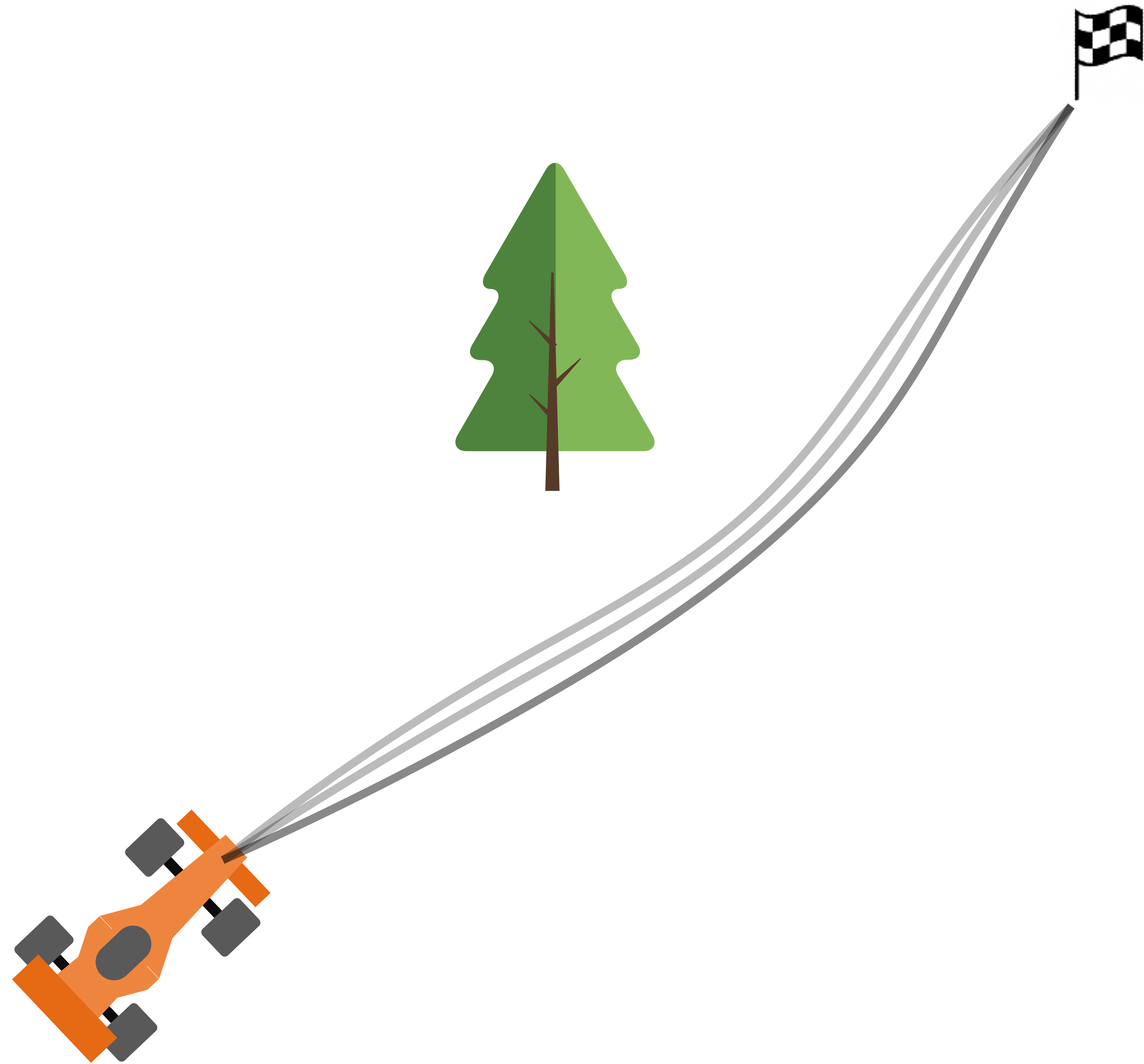
Call LQR to get quadratic values

LQR for a *non-linear, non-quadratic* MDP



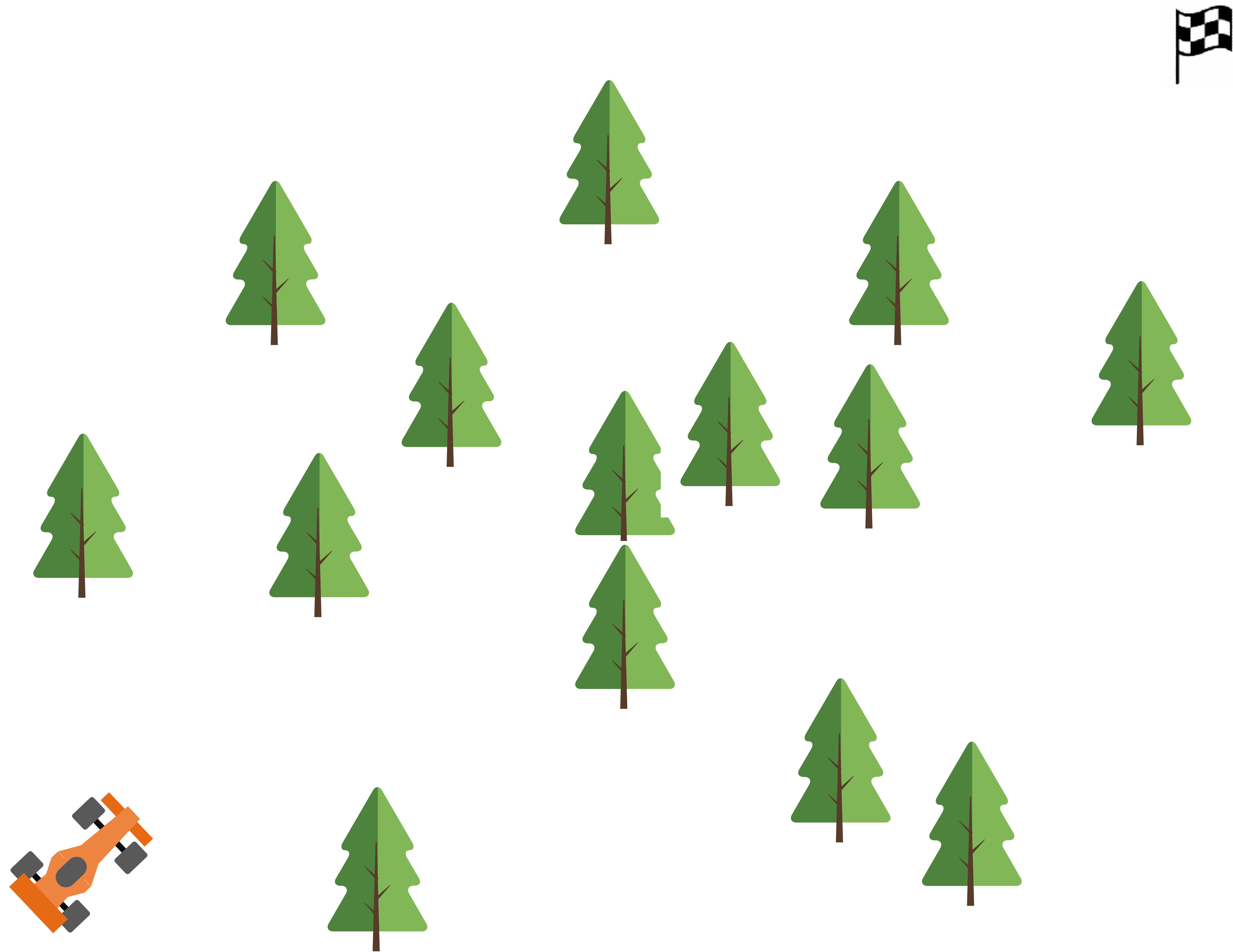
Execute LQR policy to get new sequence of actions

LQR for a *non-linear, non-quadratic* MDP



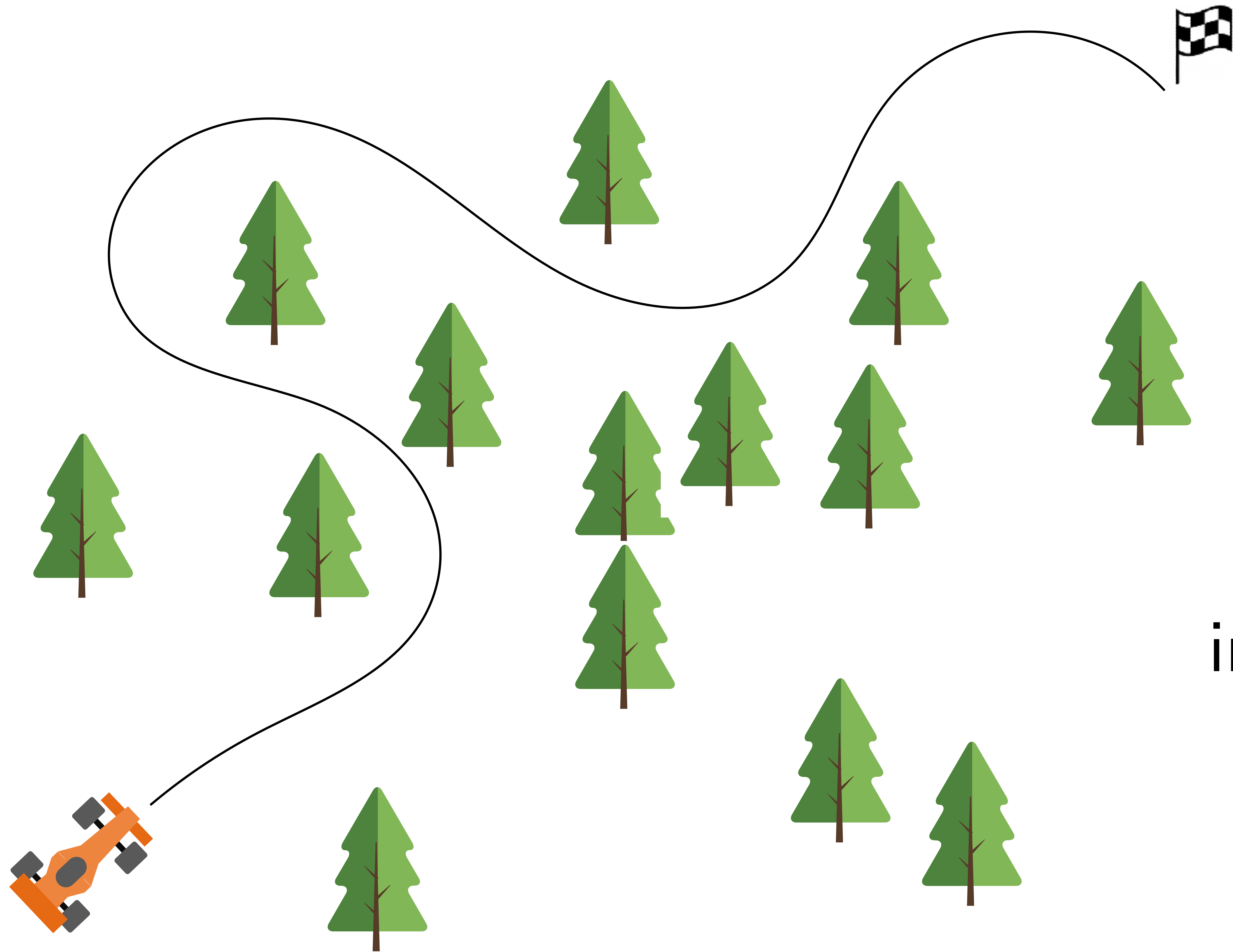
Repeat the process
till convergence!

But what happens when we have lots of trees?



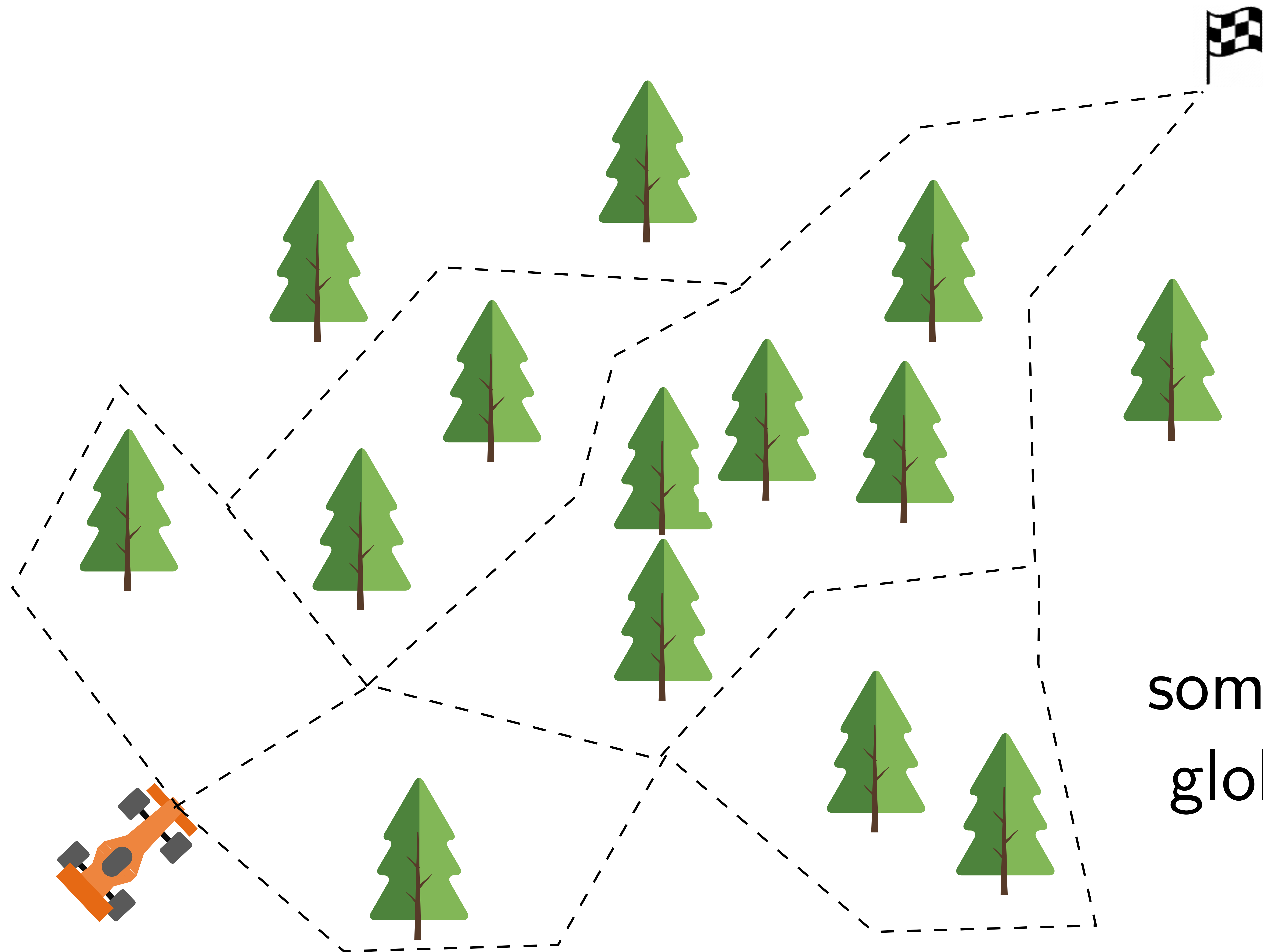
Many local optima!

But what happens when we have lots of trees?



If we initialize LQR
in a bad local basin,
it finds
a bad local optima

But what happens when we have lots of trees?



Instead we need something that can search globally to initialize LQR

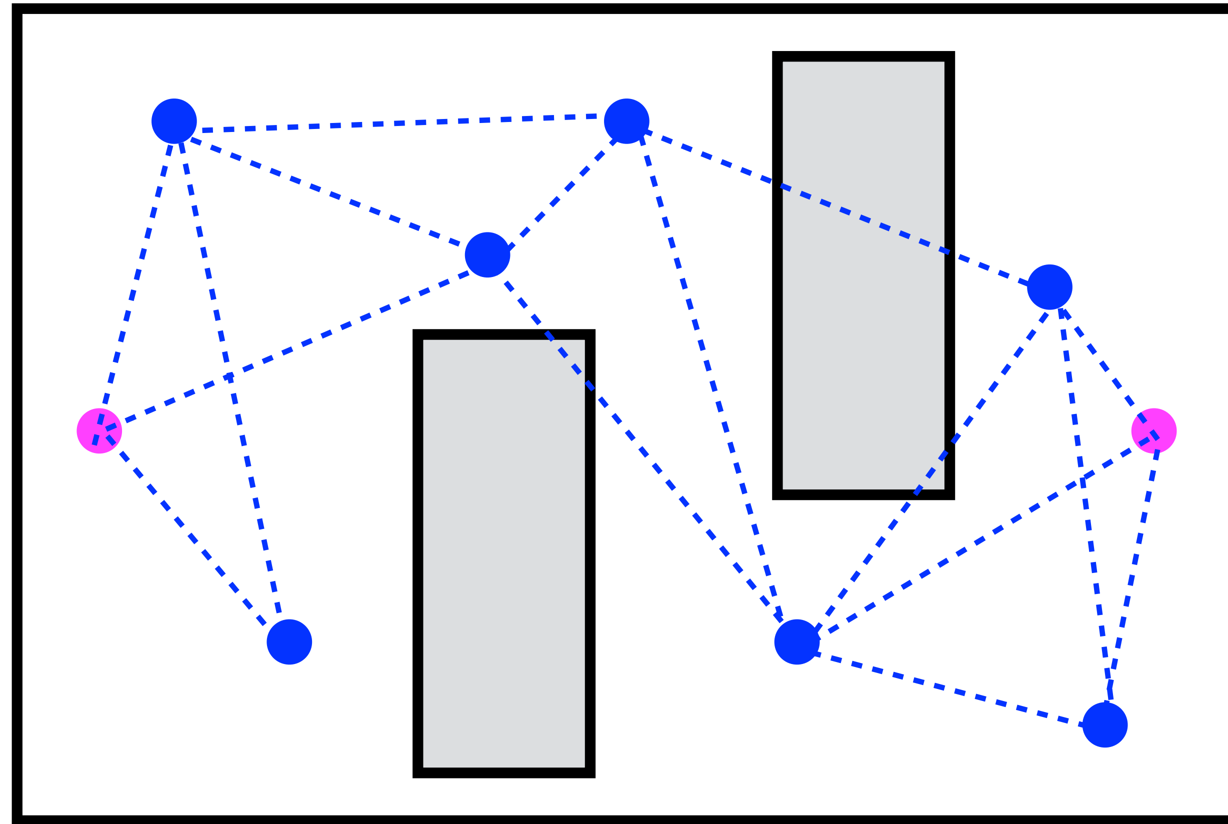
The Problem with General MDPs

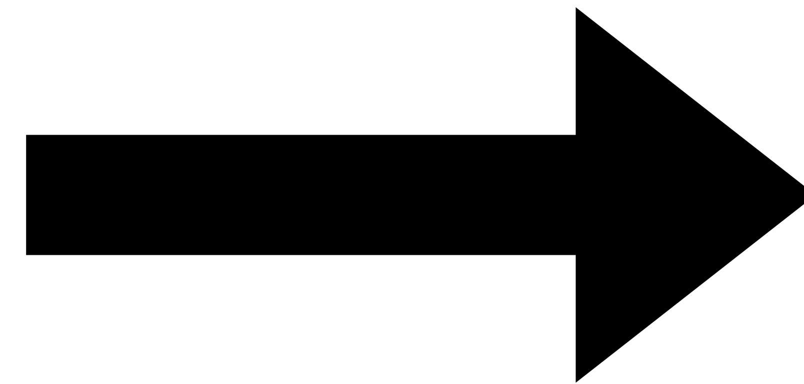
LQR reasons locally

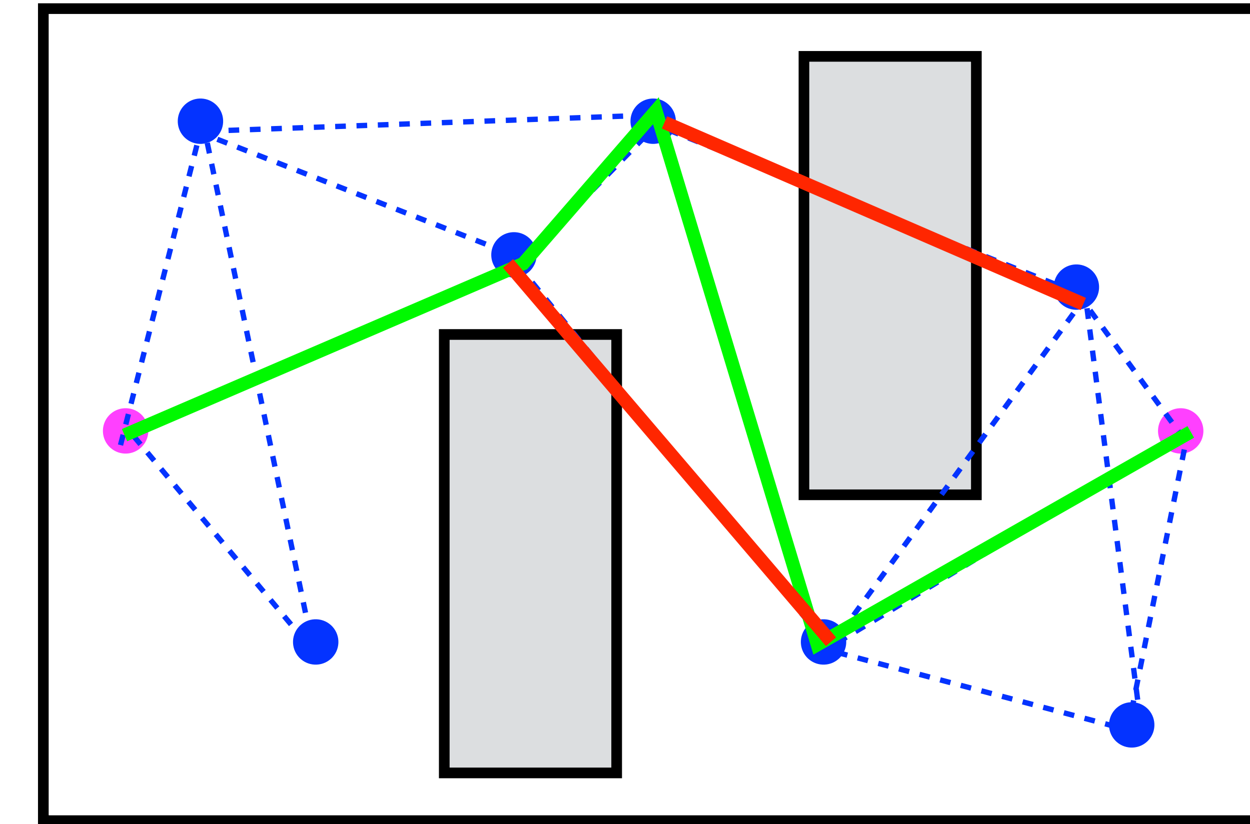
We need to combine it with something that reasons globally

This global reasoning is typically done by motion planning

General framework for motion planning





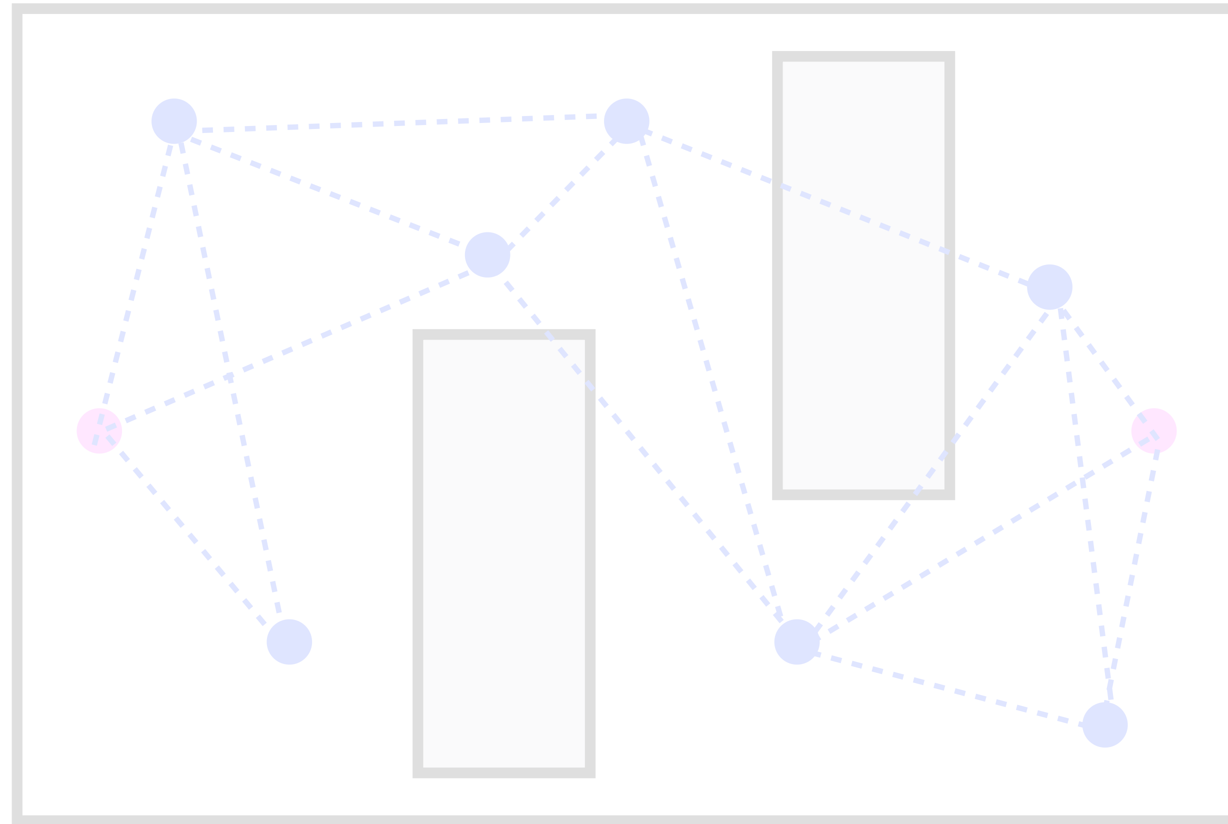


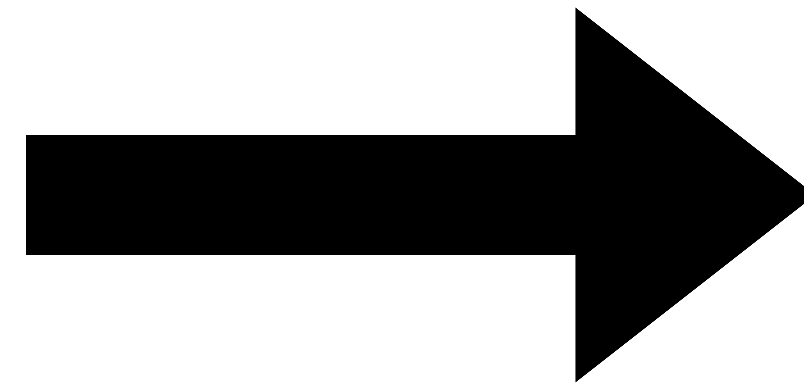
Create a graph

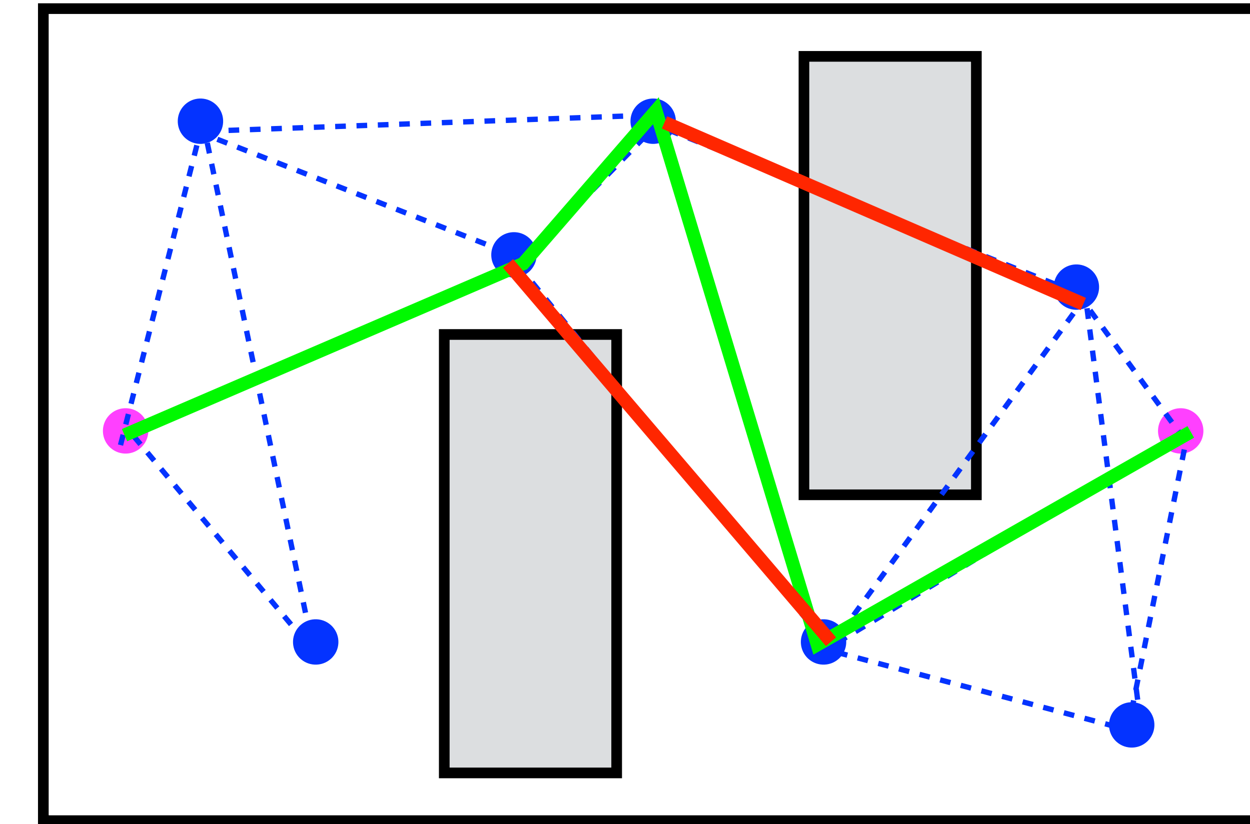
Search the graph

Interleave

General framework for motion planning







Create a graph

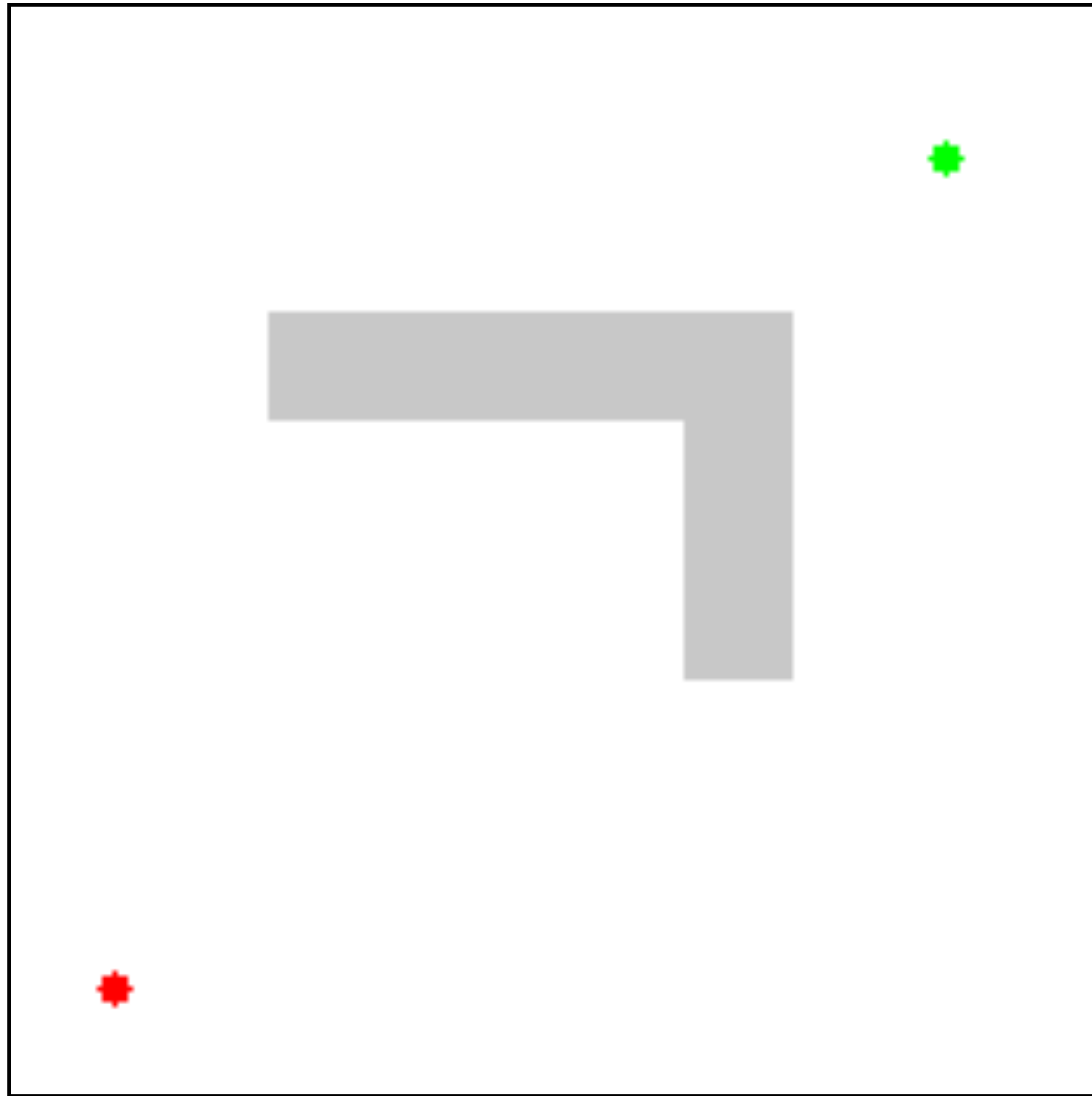
Search the graph

Interleave

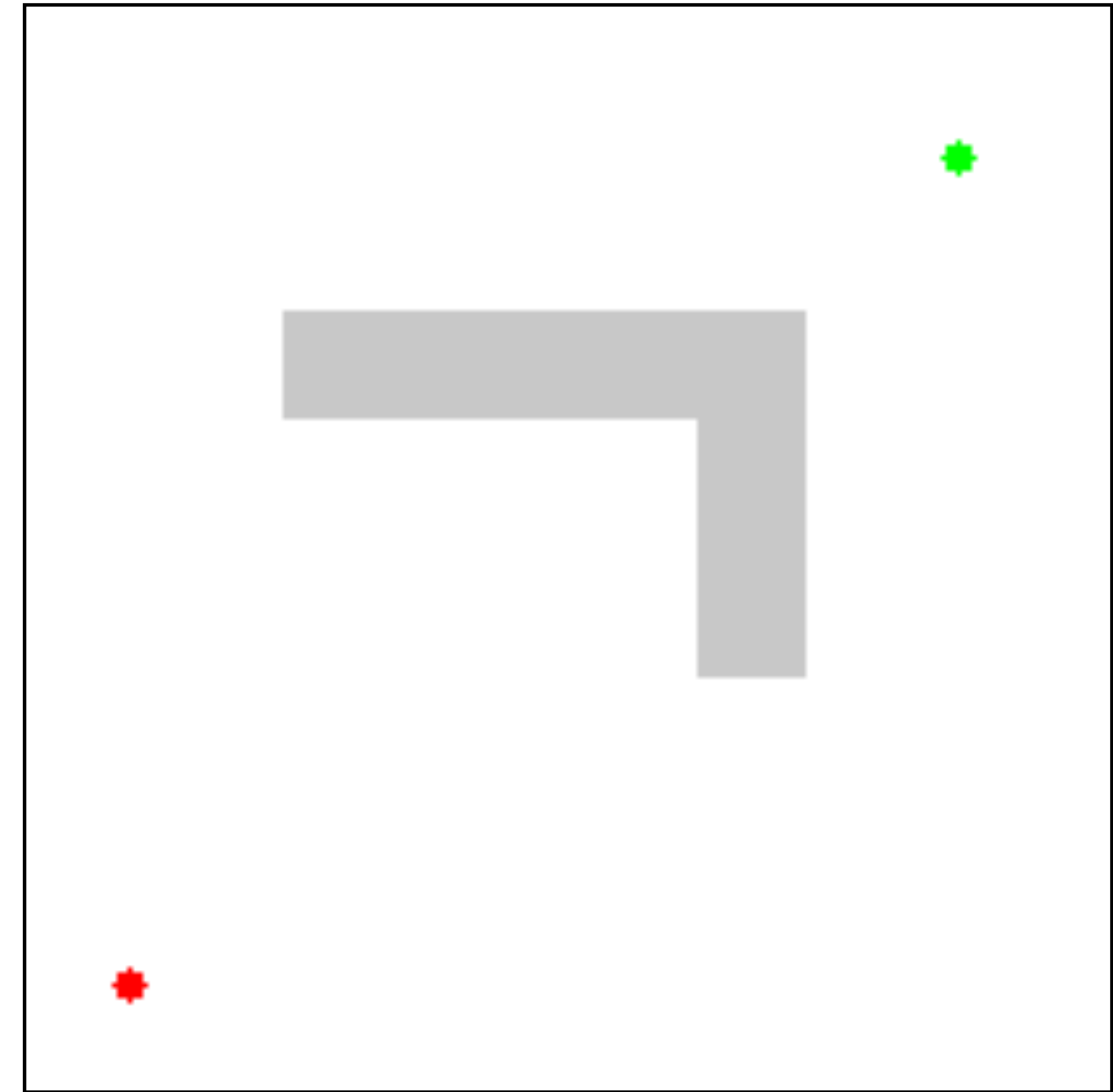
How can we make this search faster?

Dijkstra

How can we make this search faster?

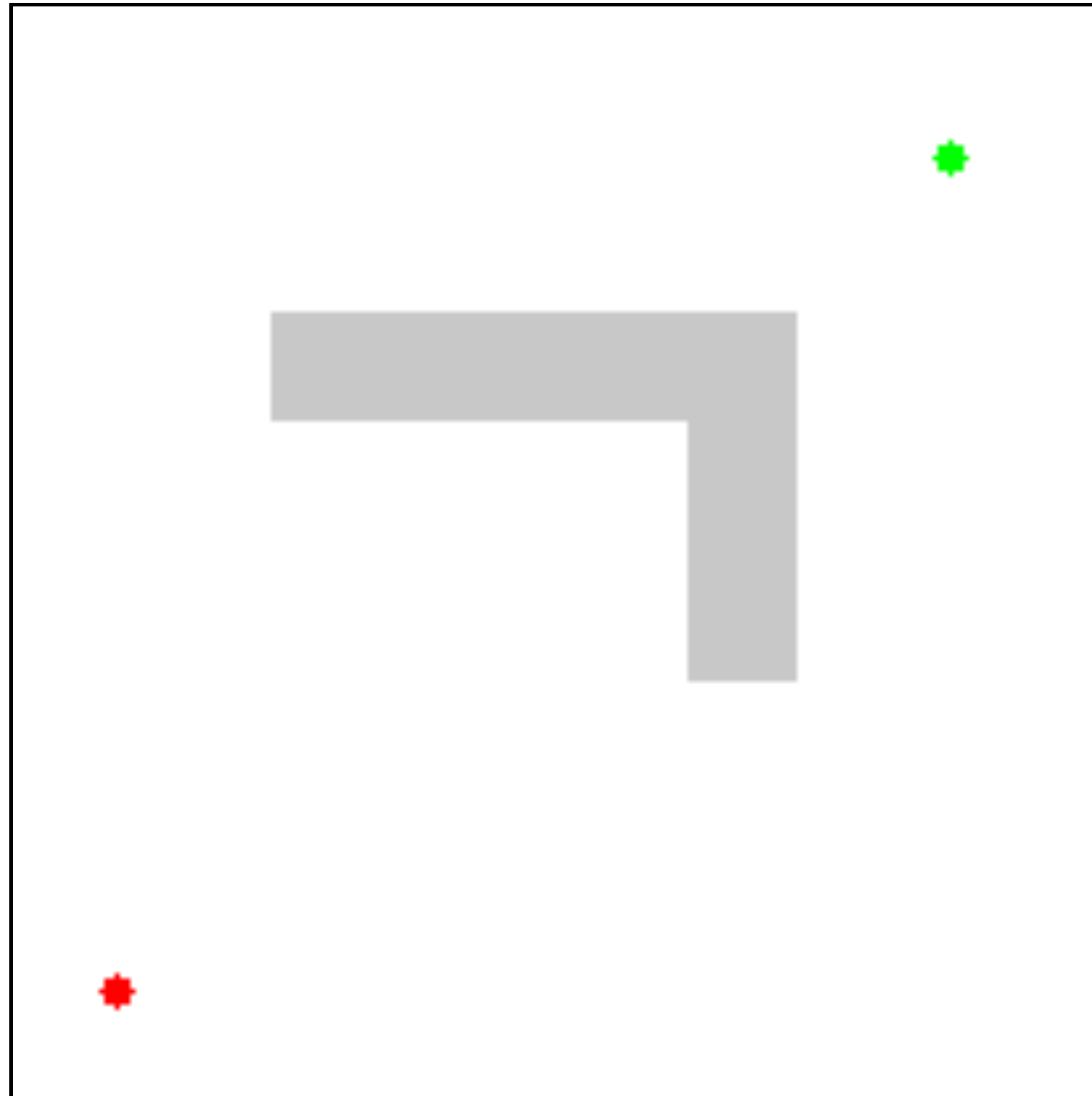


Dijkstra



A* with heuristic!

What can we prove about A*



A* with heuristic!

1. A* gives us the optimal path

(If heuristic is admissible)

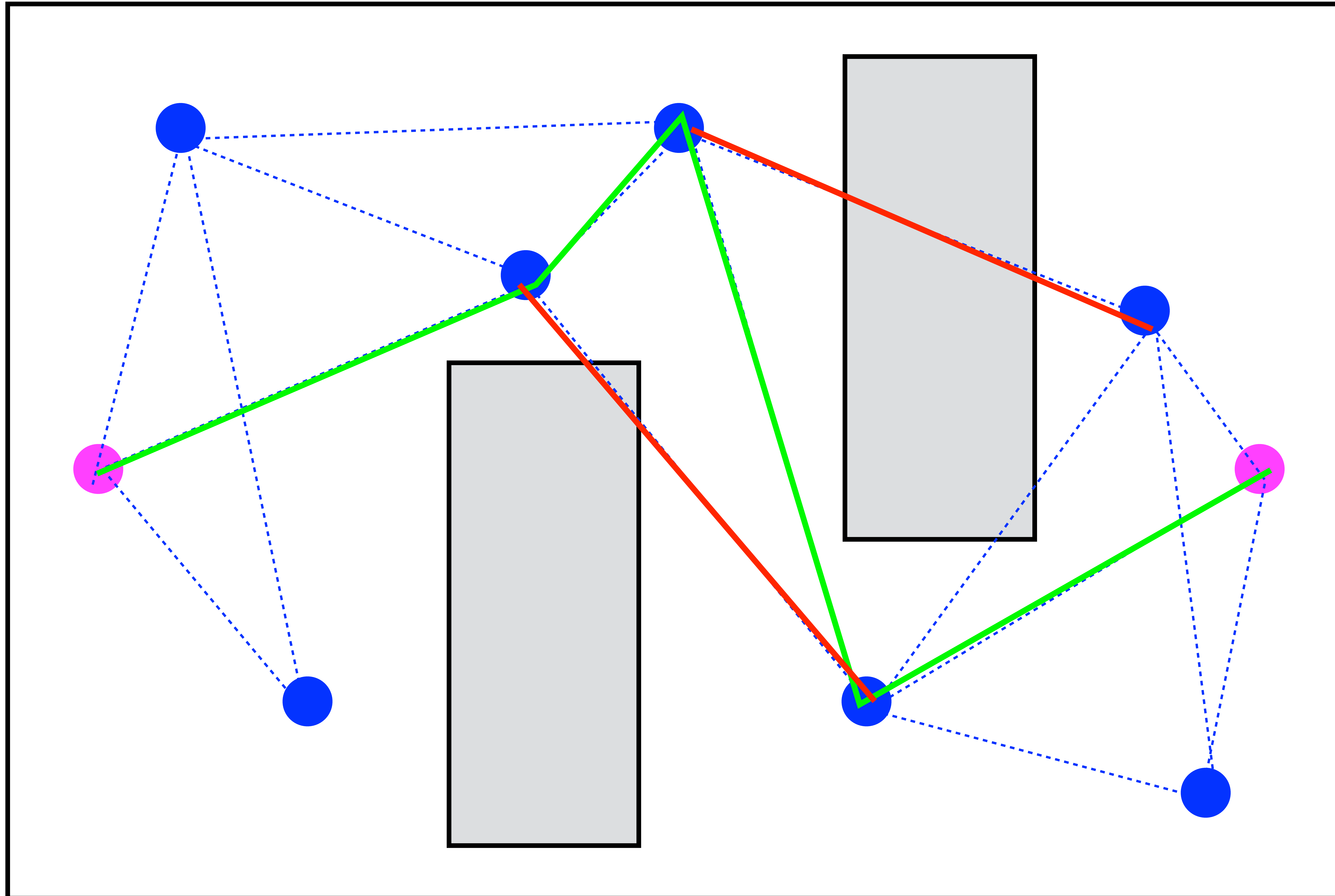
2. A* expands the optimal number of vertices

(If heuristic is consistent)

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

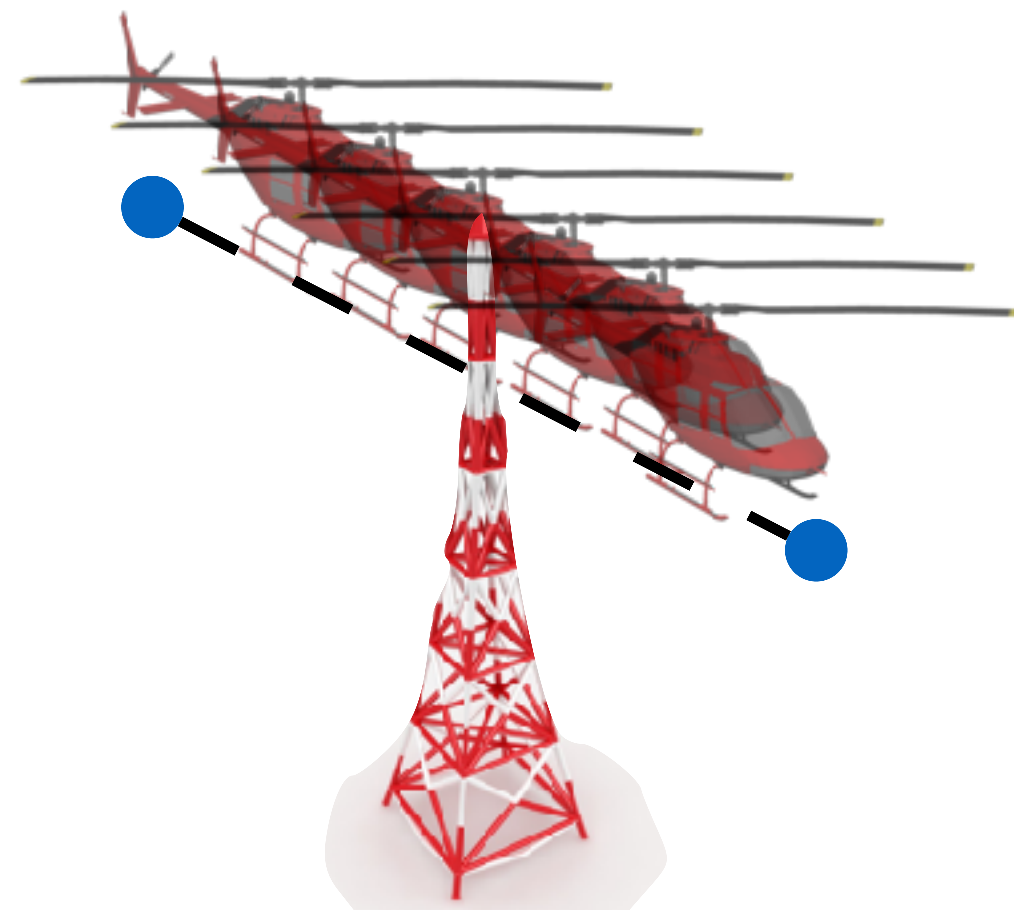
What is the most expensive step?

Edge evaluation is the most expensive step

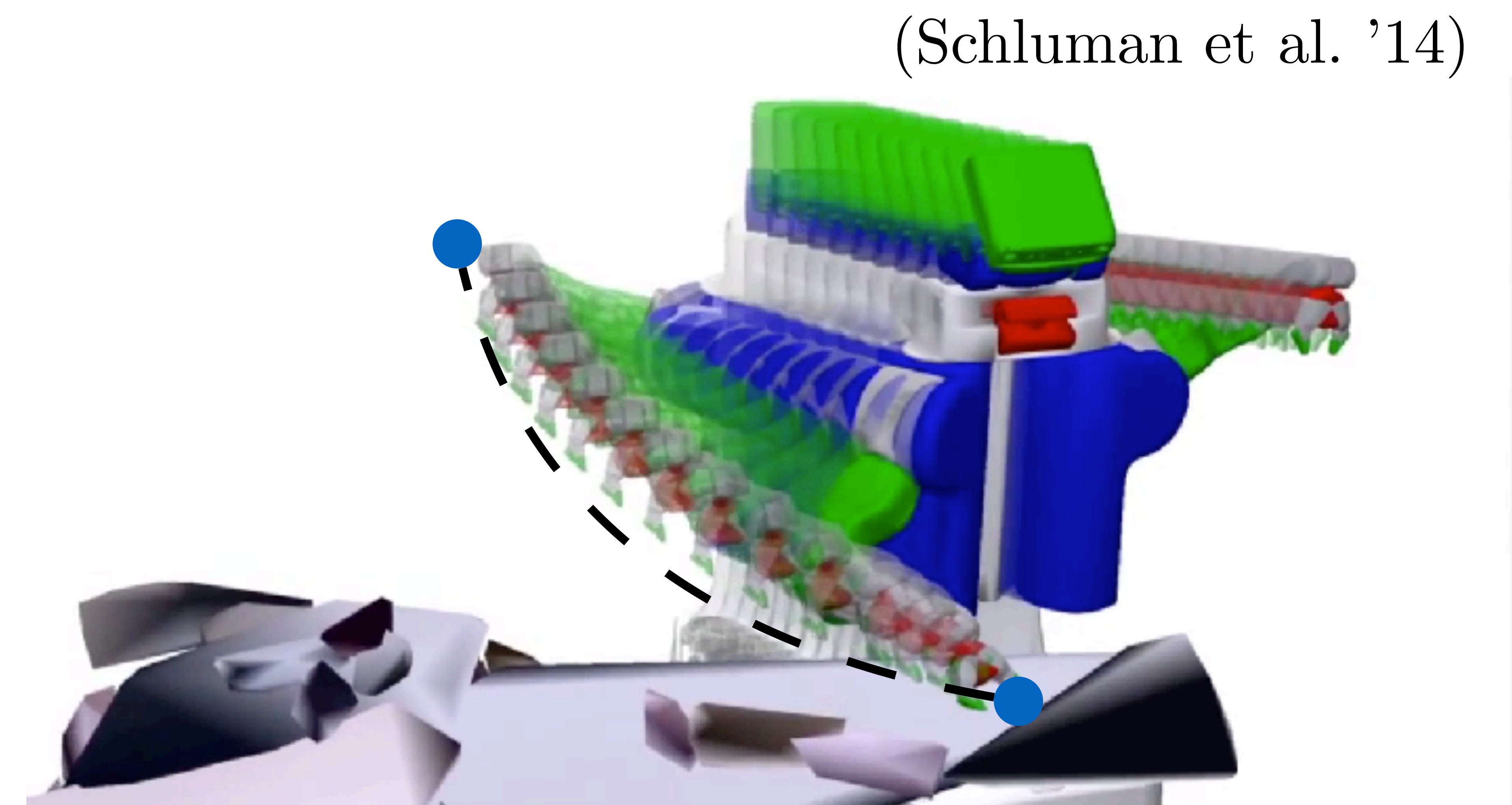


Why?

Edge evaluation requires expensive collision checking

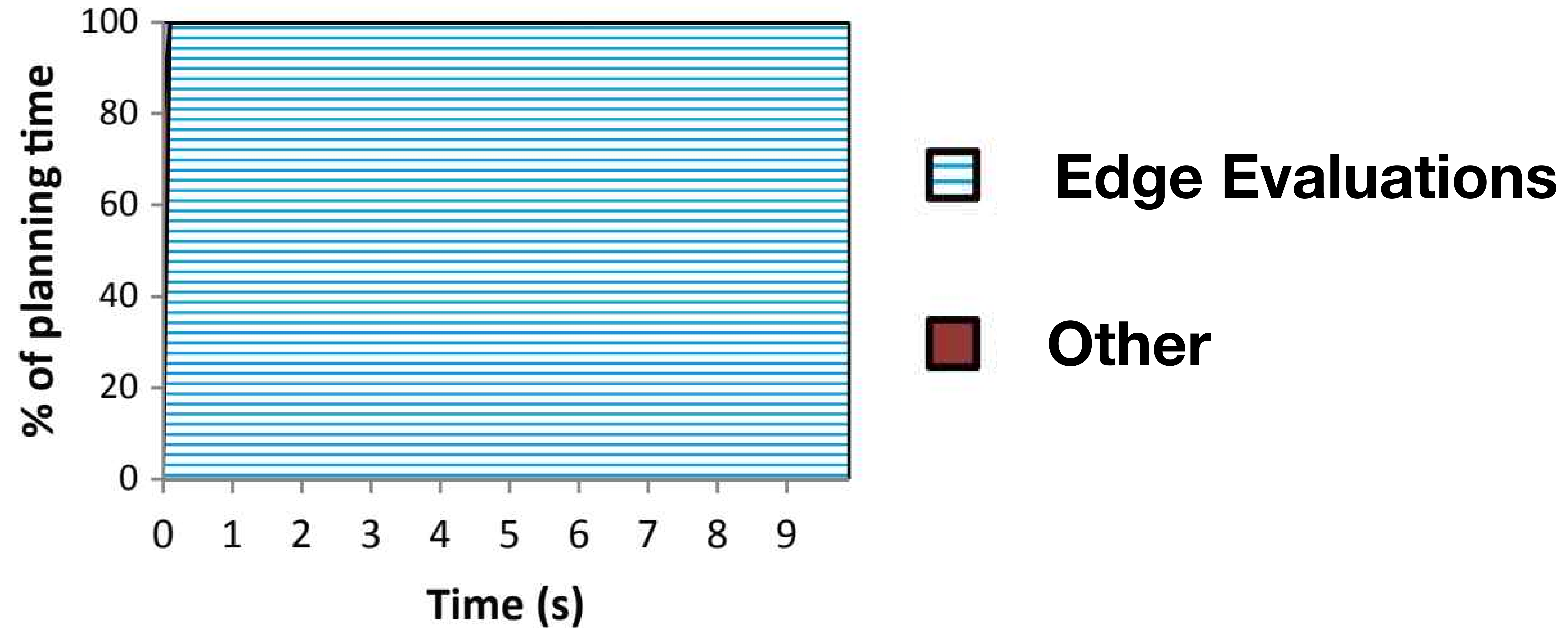


Check if helicopter
intersects with tower



Check if manipulator
intersects with table

Edge evaluation dominates planning time

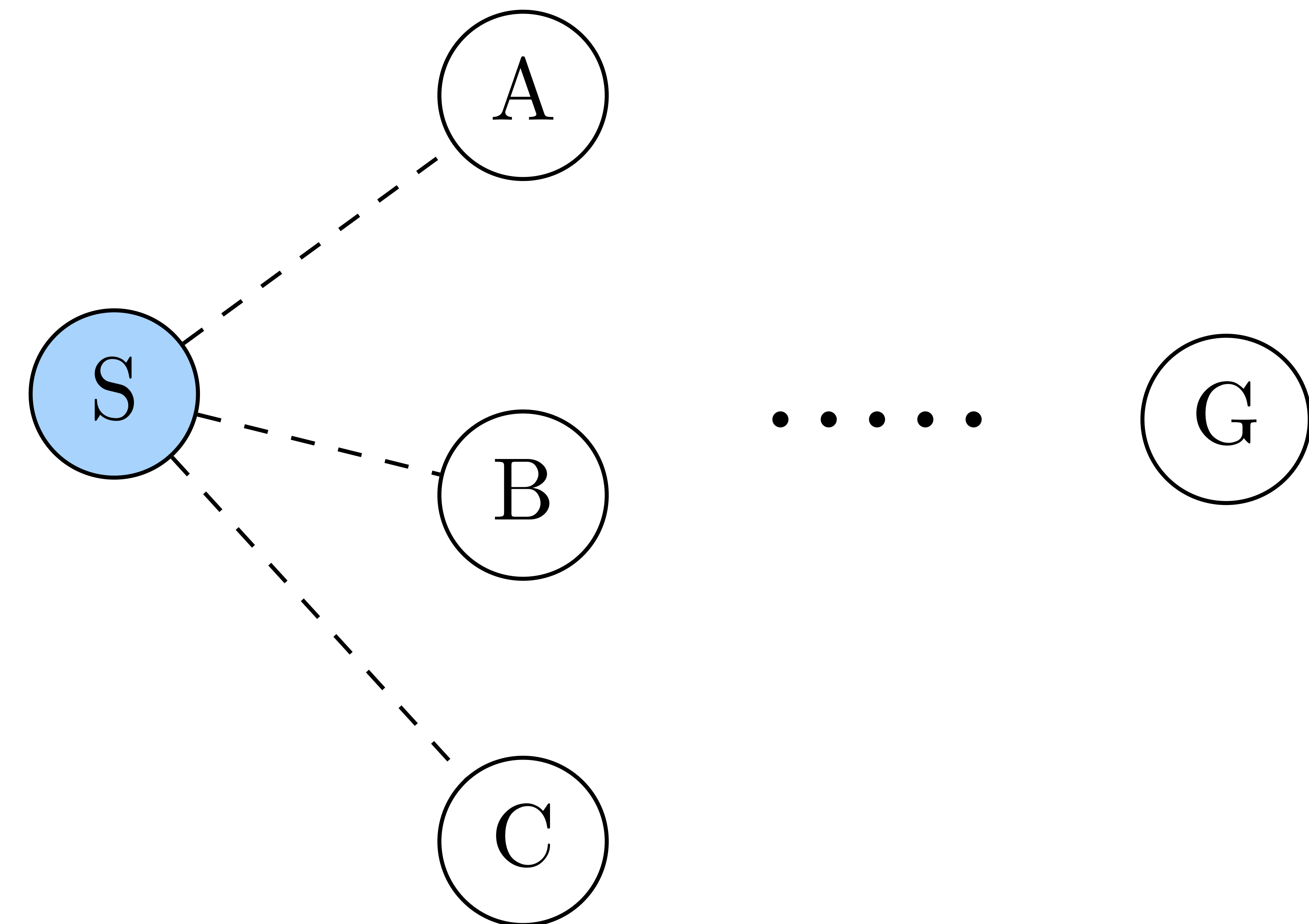


Hauser, Kris., Lazy collision checking in asymptotically-optimal motion planning. *ICRA* 2015

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

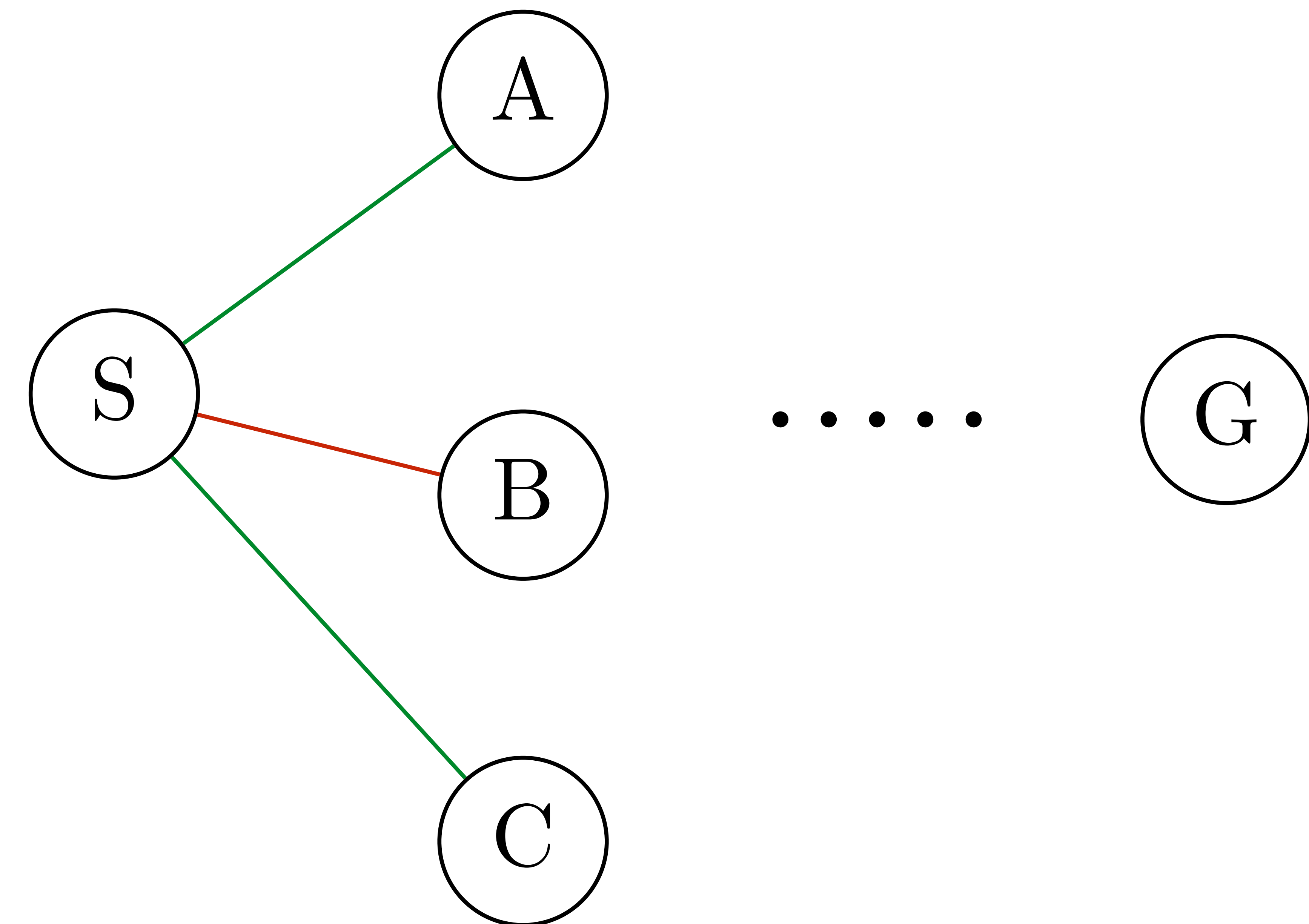
Let's revisit Best First Search

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)

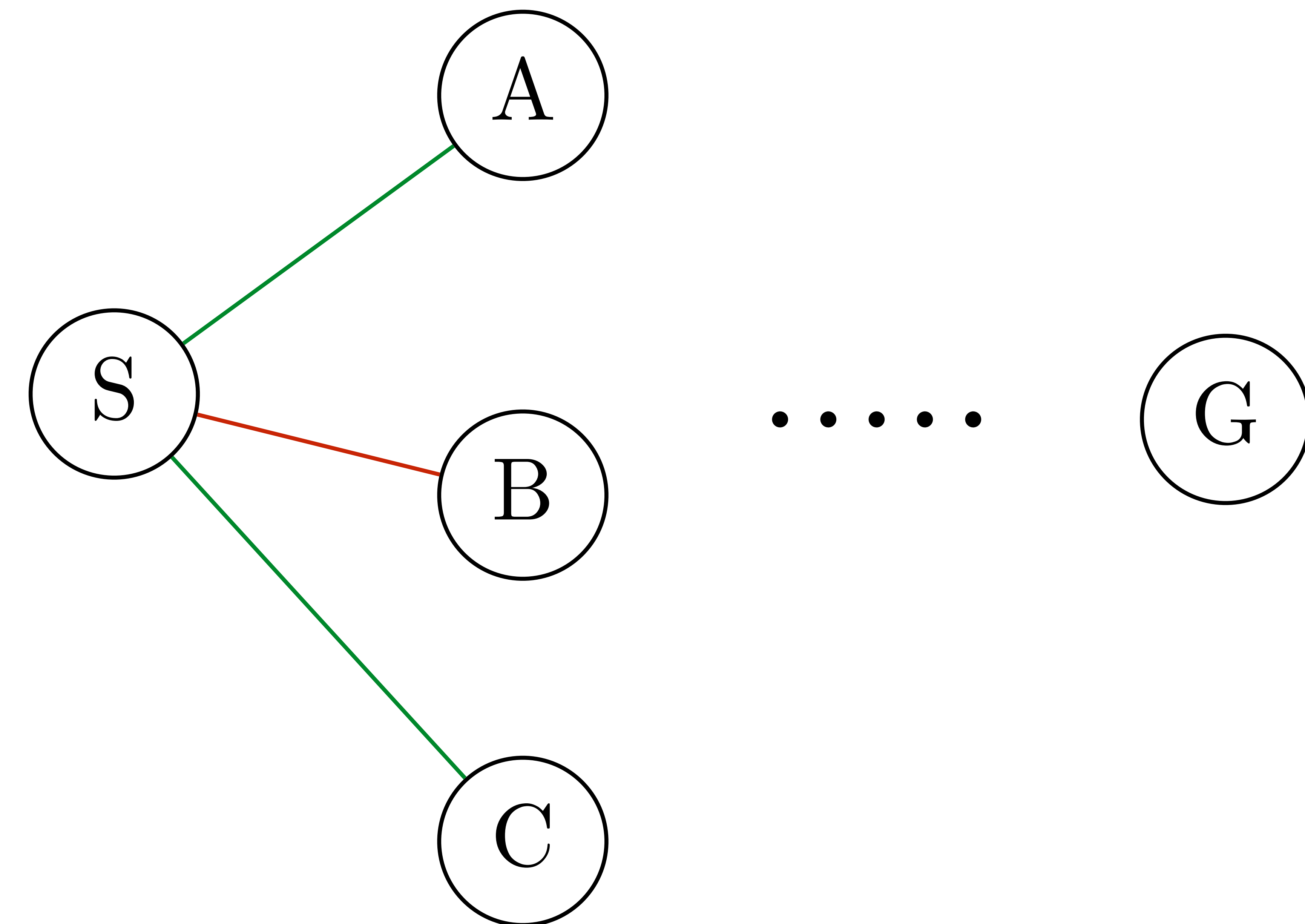


Evaluate eges (S,A), (S,B), (S,C)

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

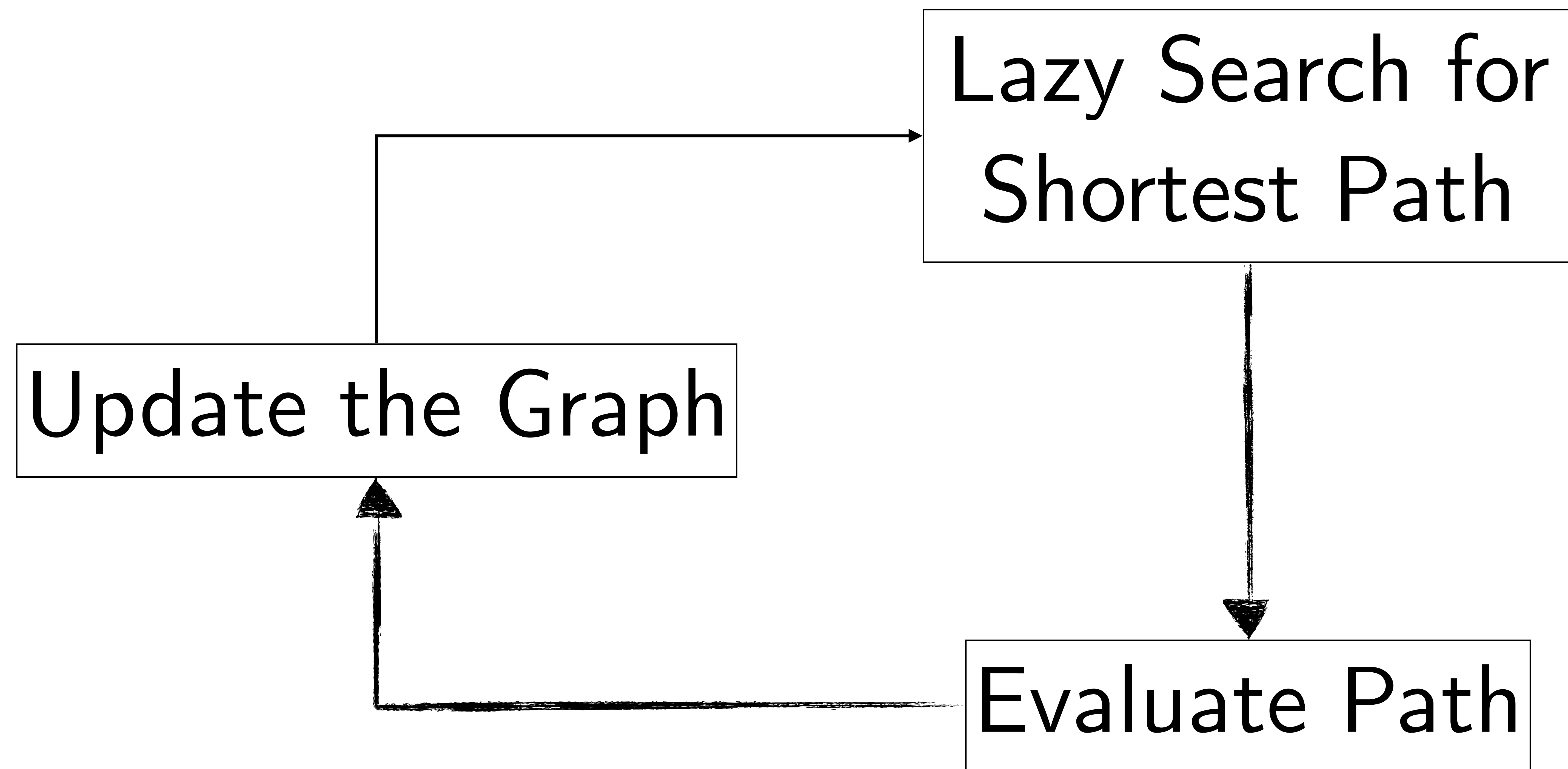
LazySP

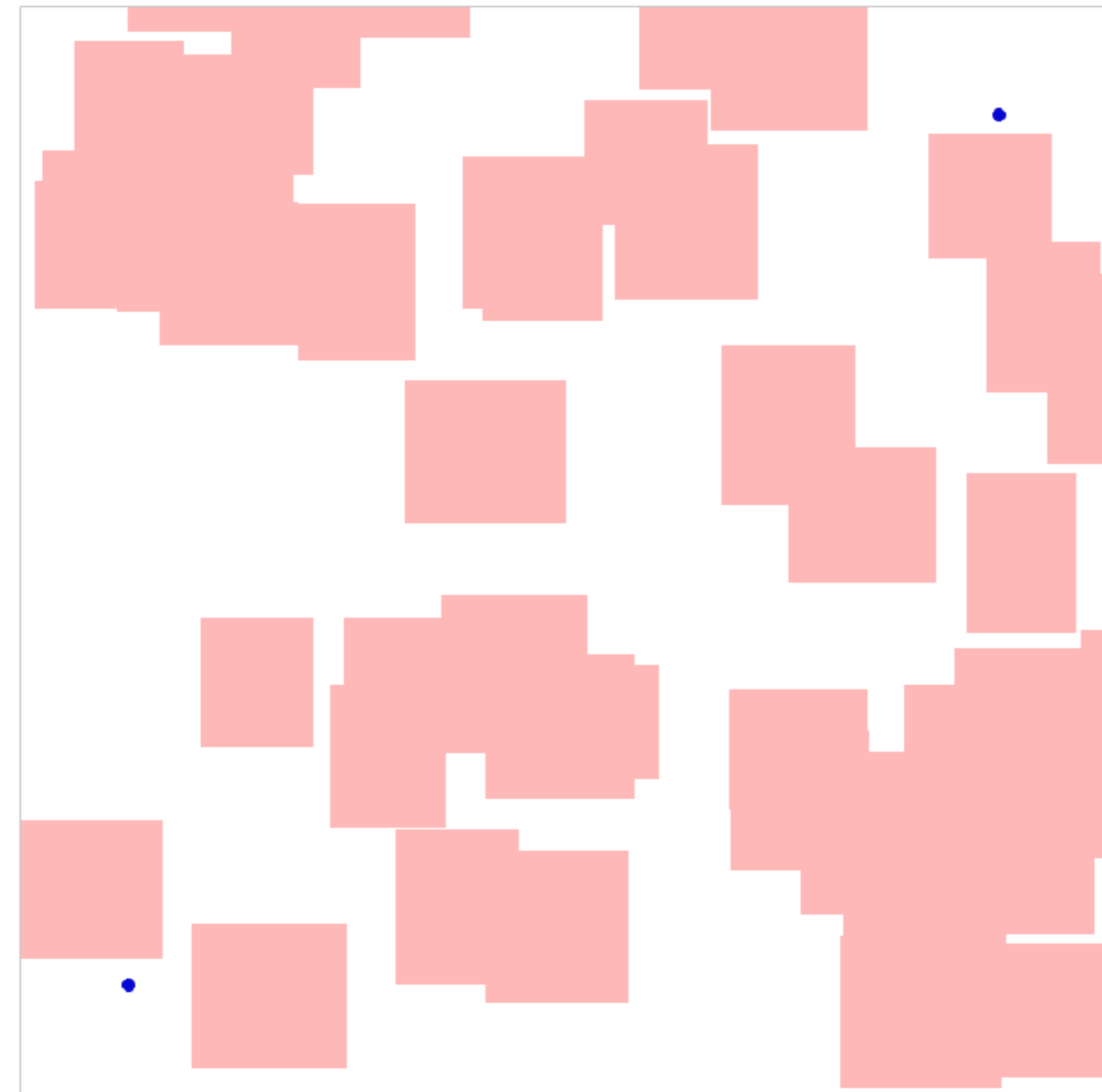
Greedy Best-first Search over Paths

To find the shortest path,
eliminate all shorter paths!

LazySP

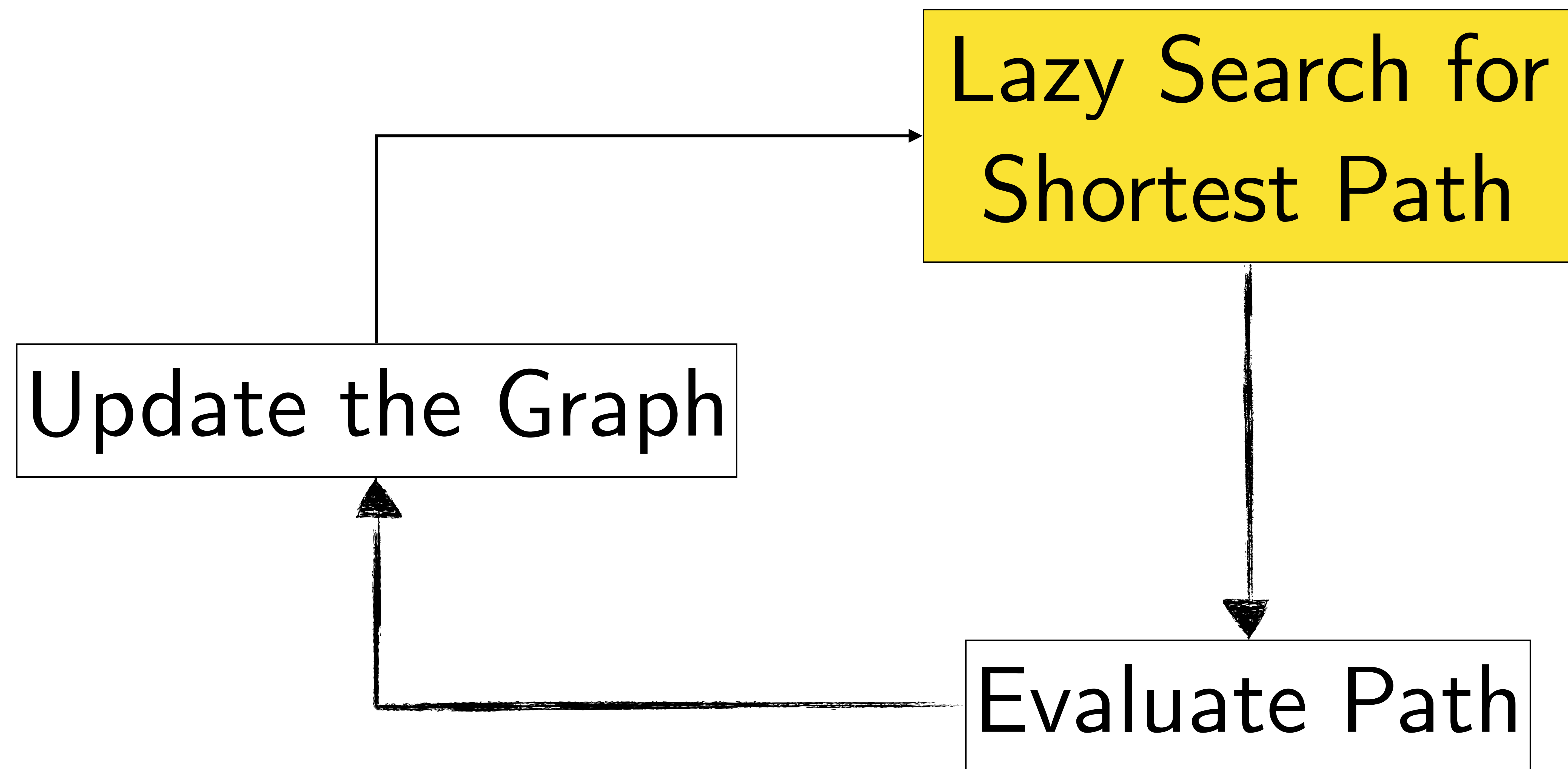
Optimism Under Uncertainty

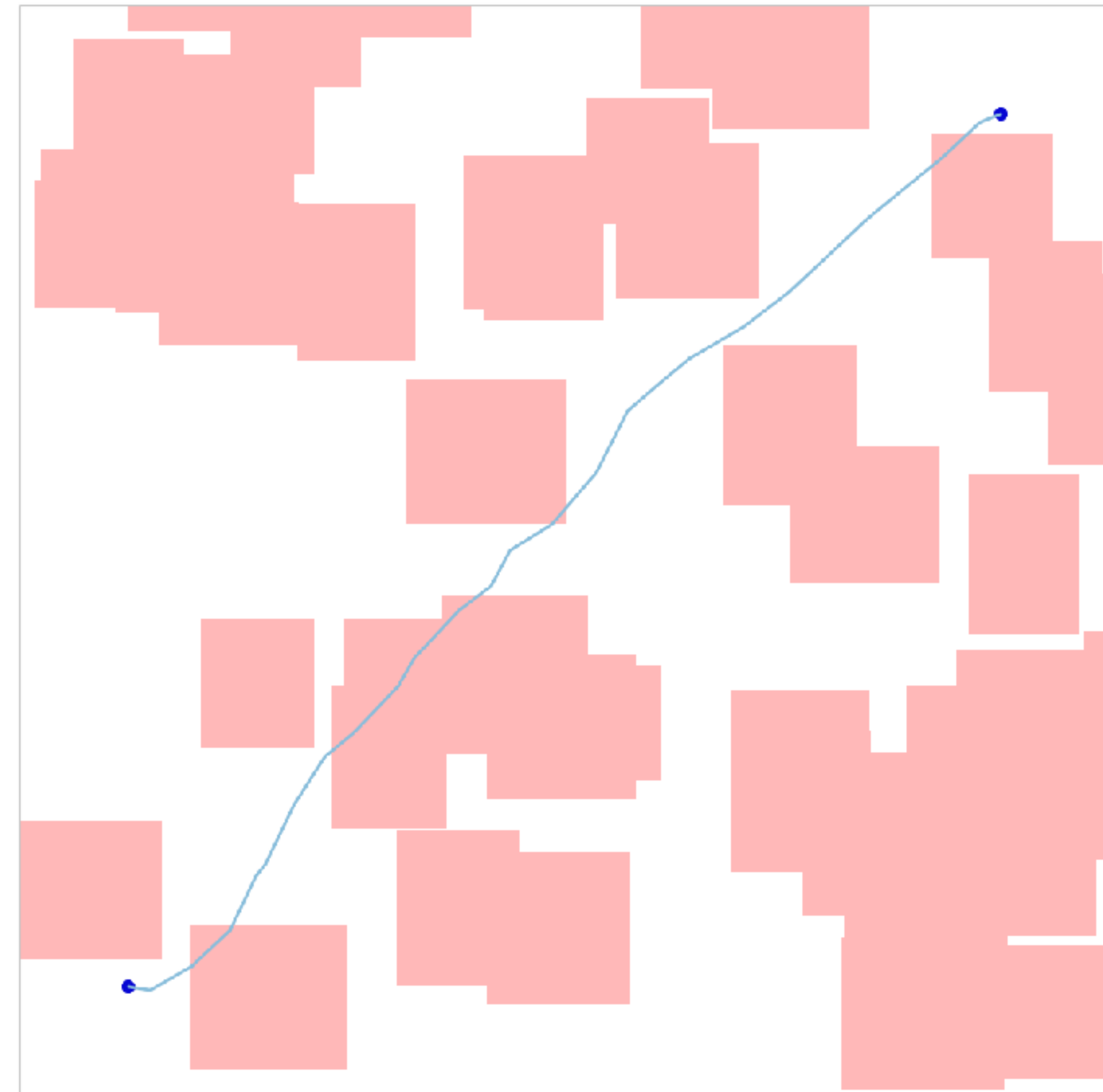




LazySP

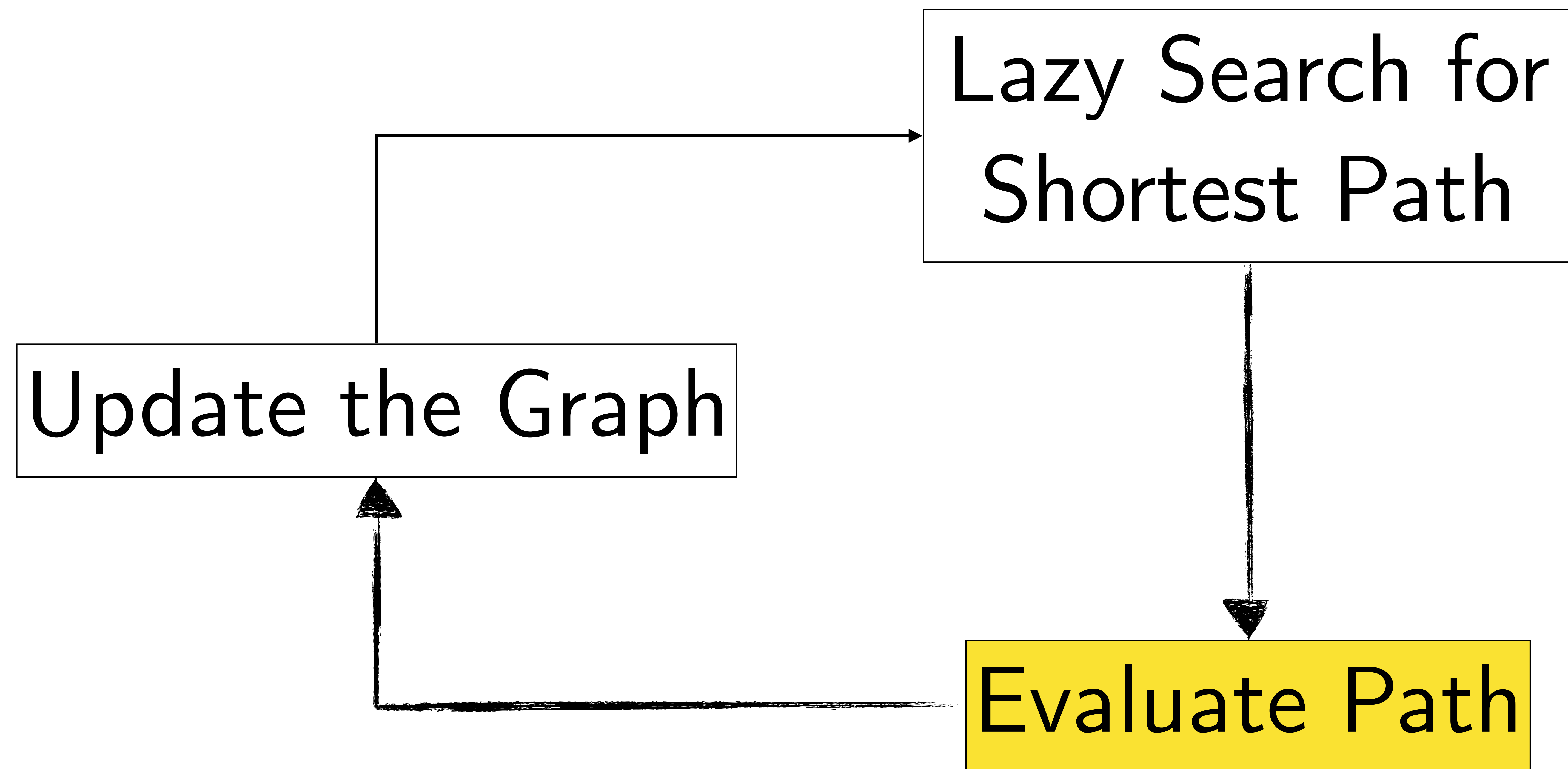
Optimism Under Uncertainty

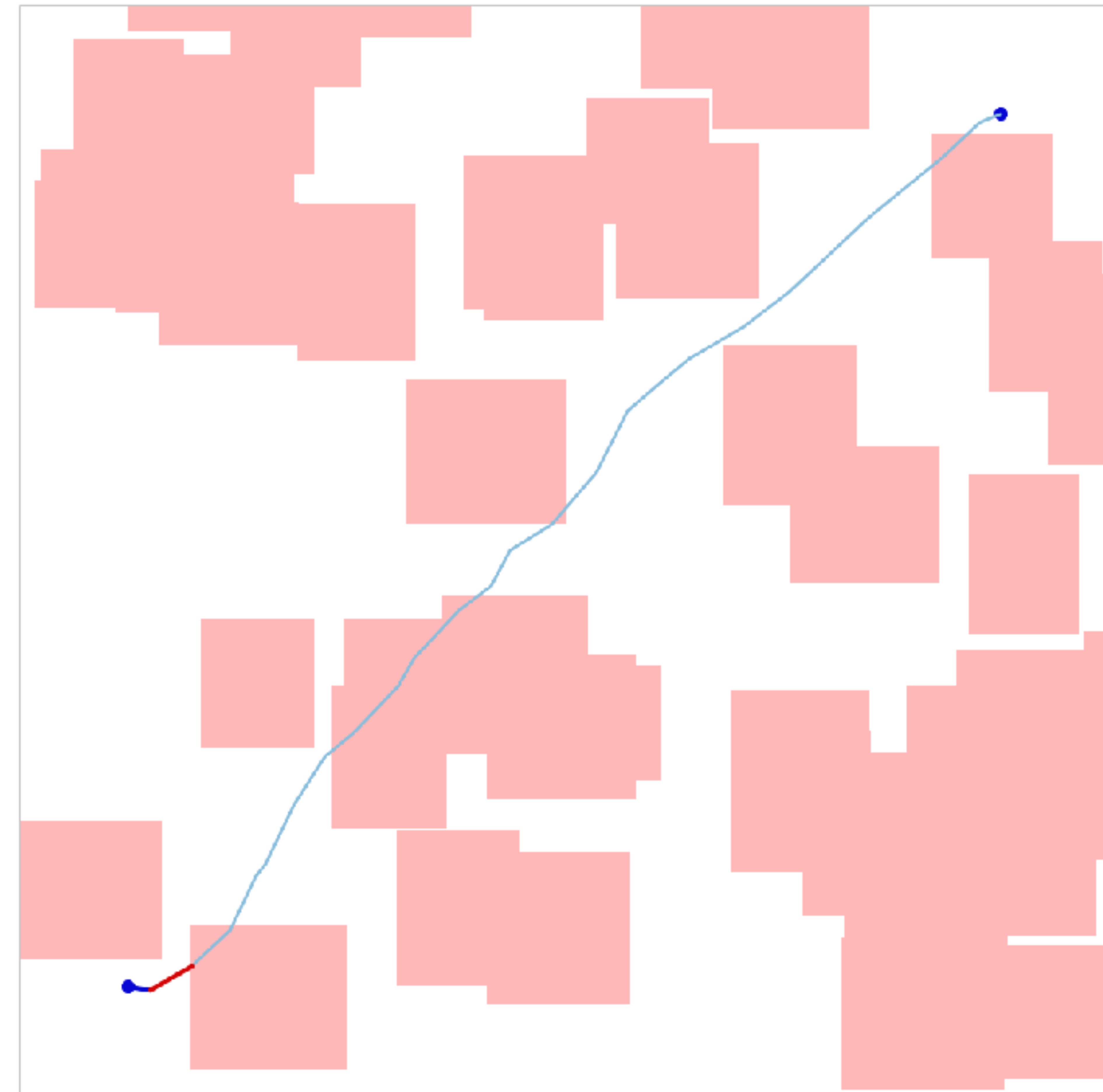




LazySP

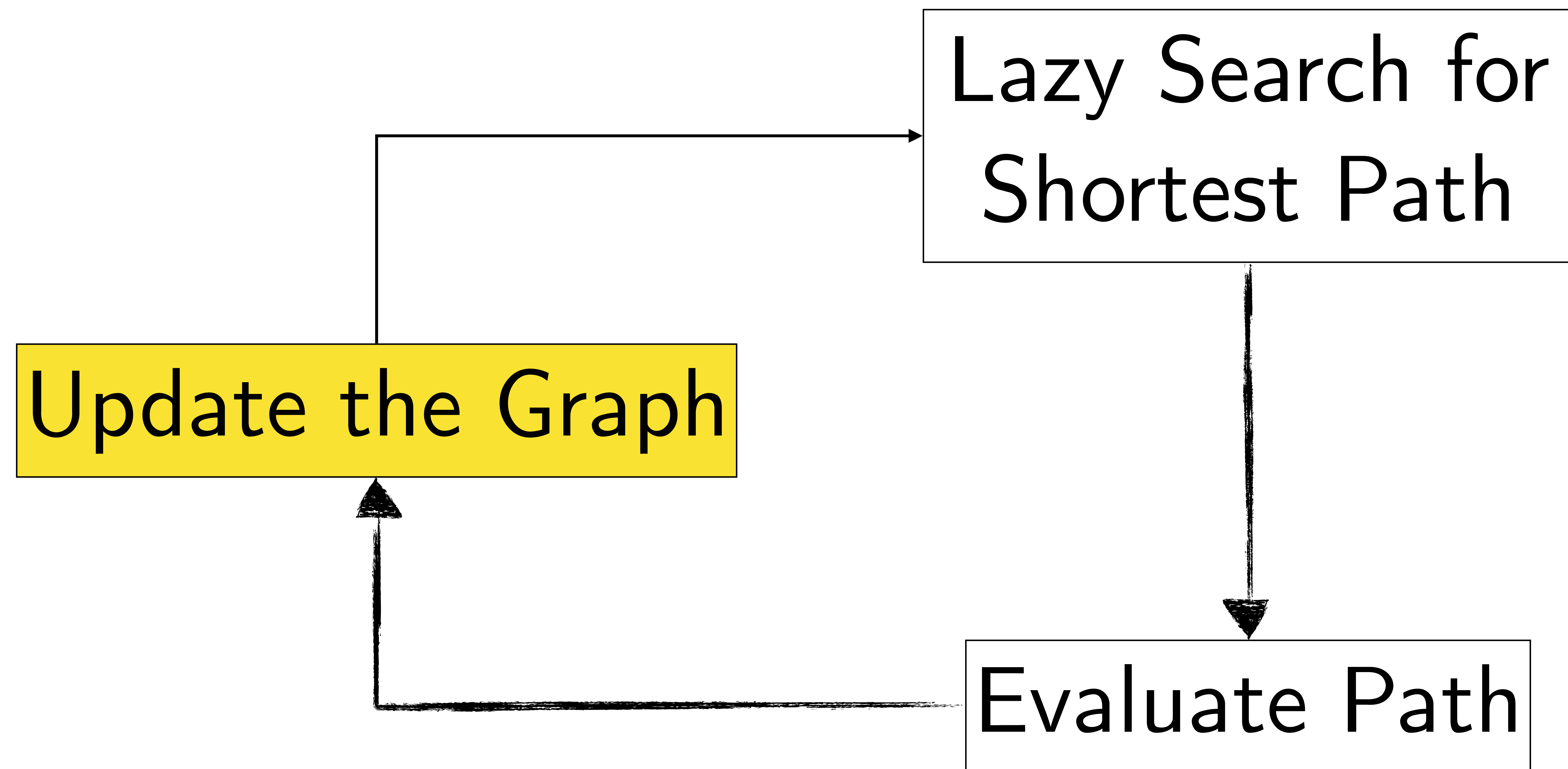
Optimism Under Uncertainty

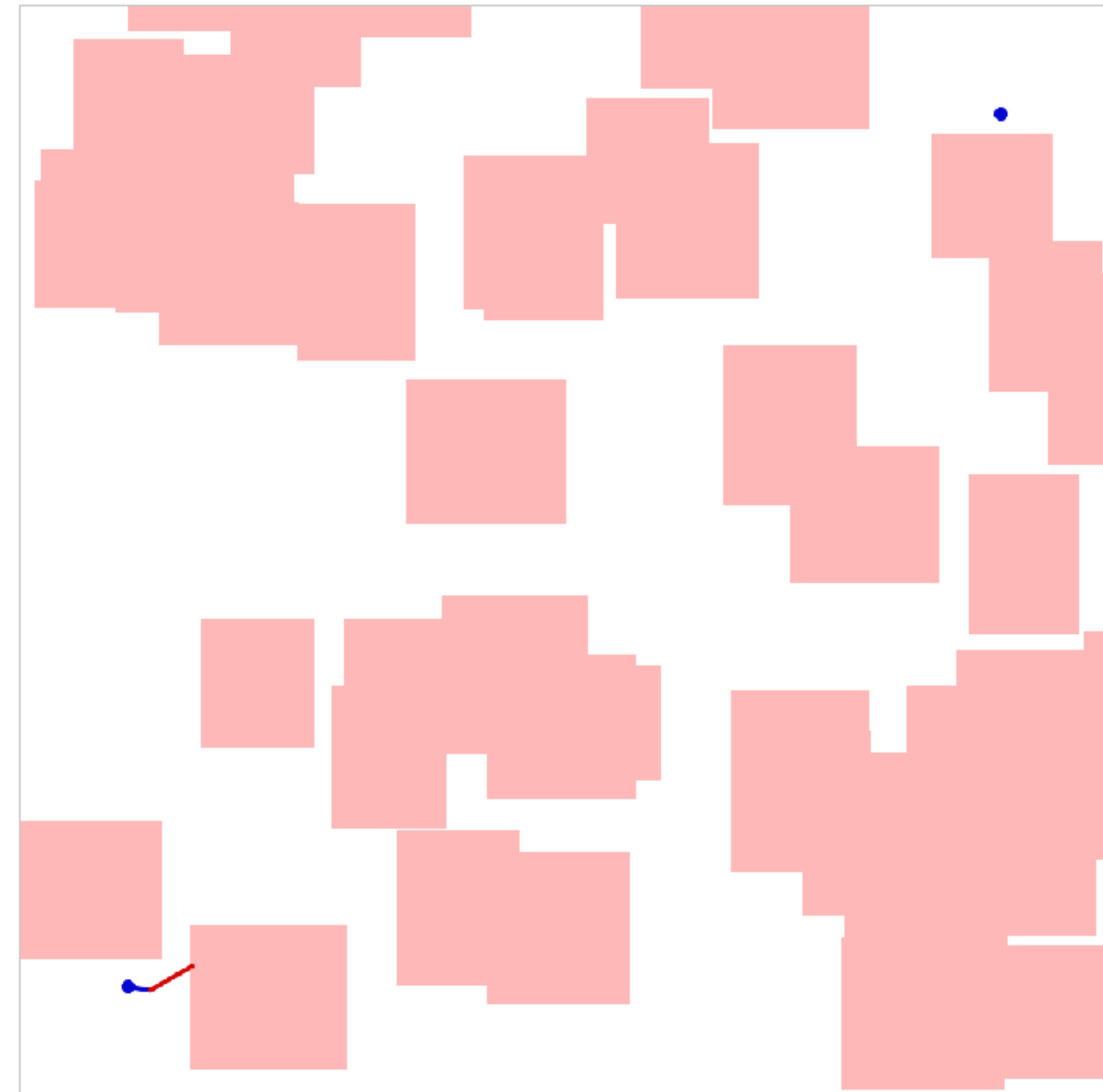




LazySP

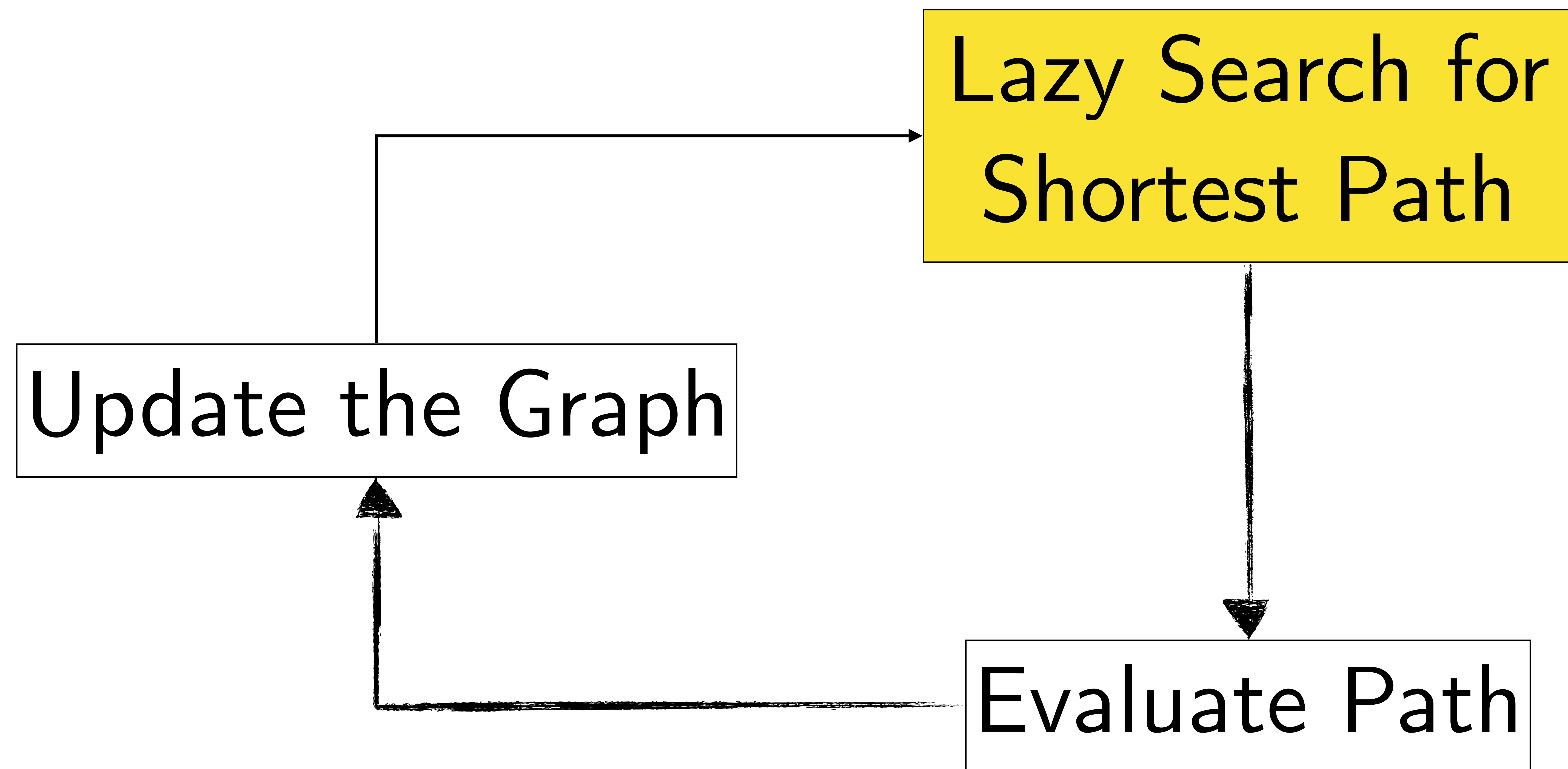
Optimism Under Uncertainty

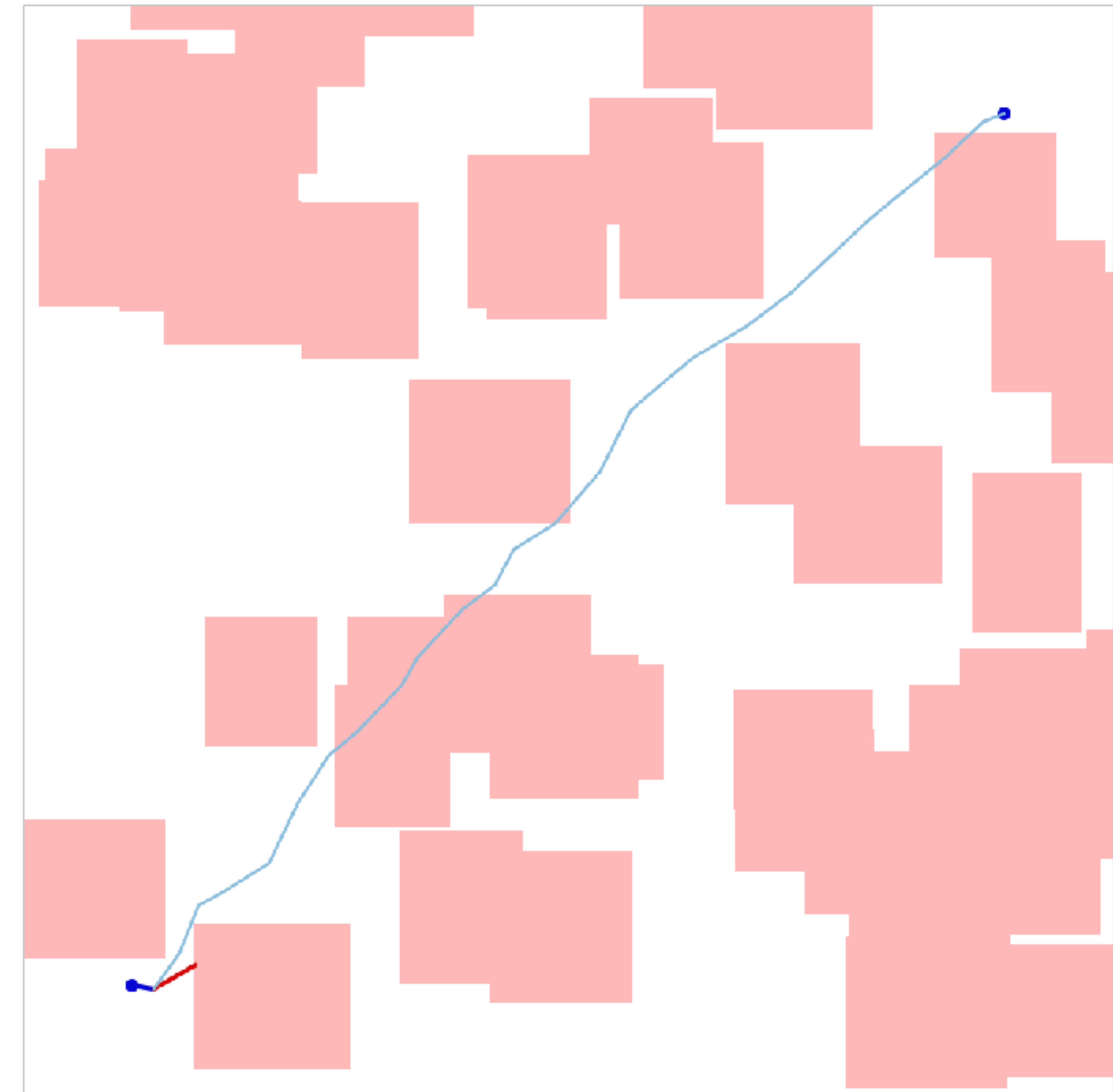




LazySP

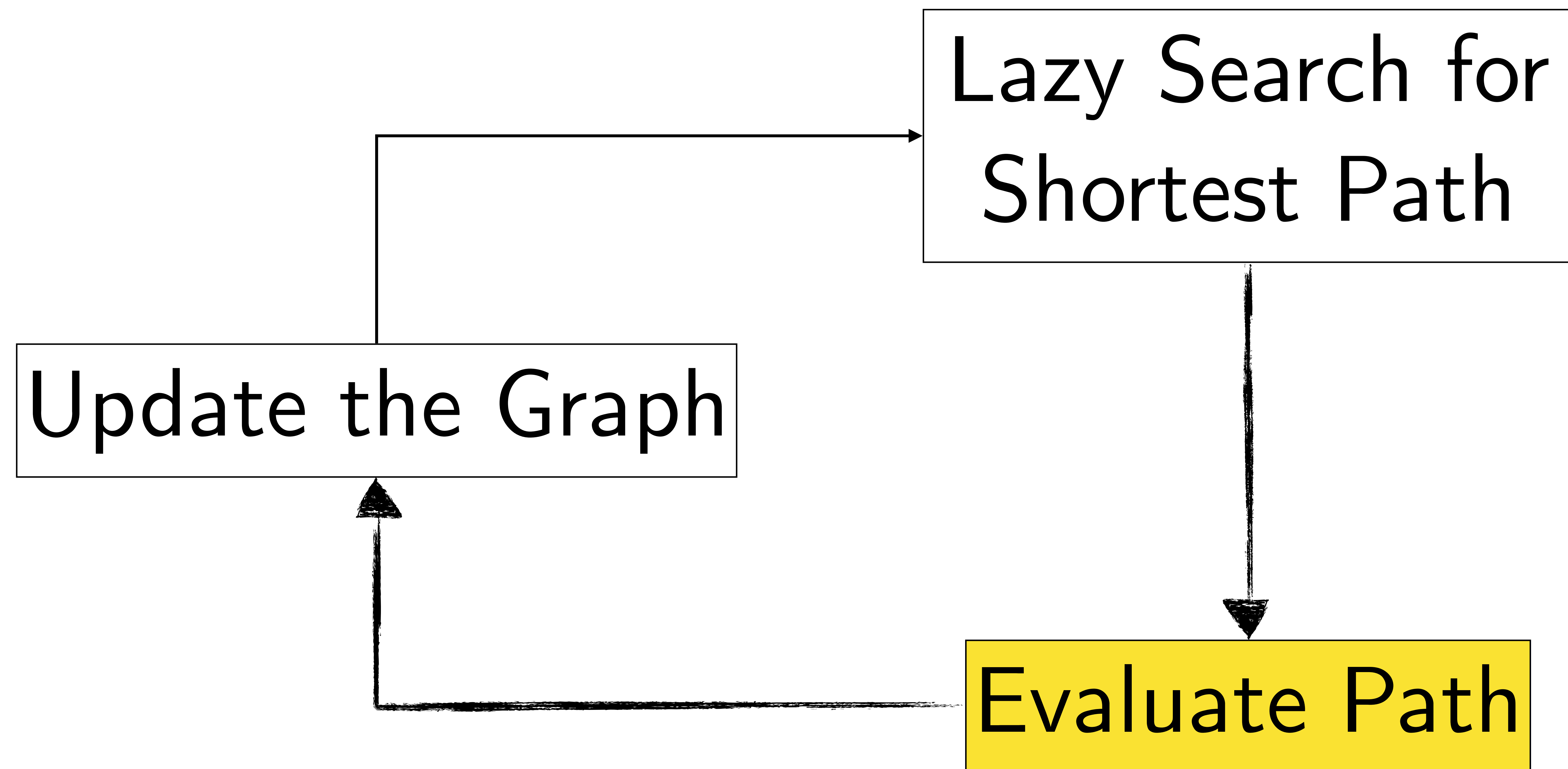
Optimism Under Uncertainty

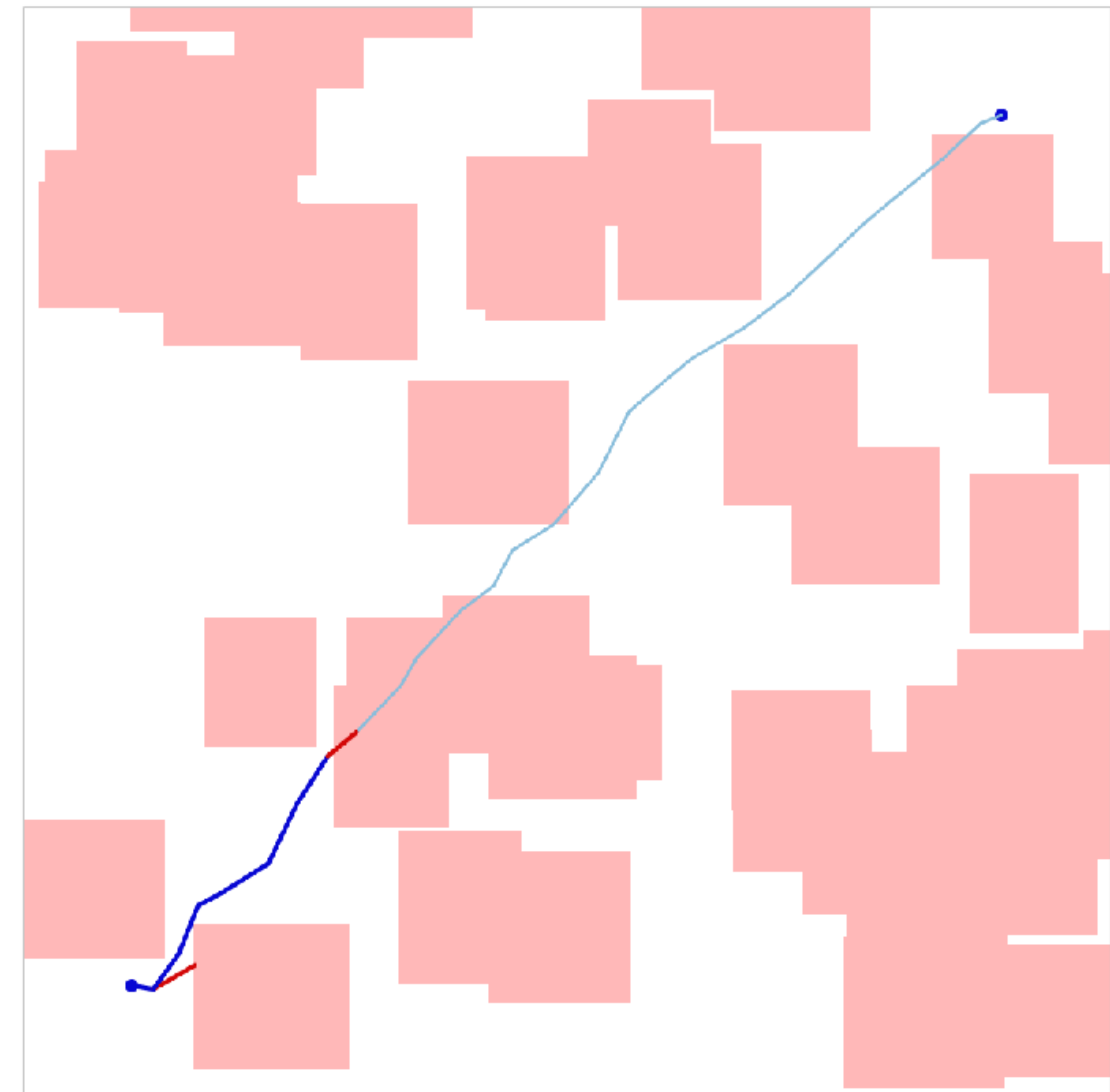




LazySP

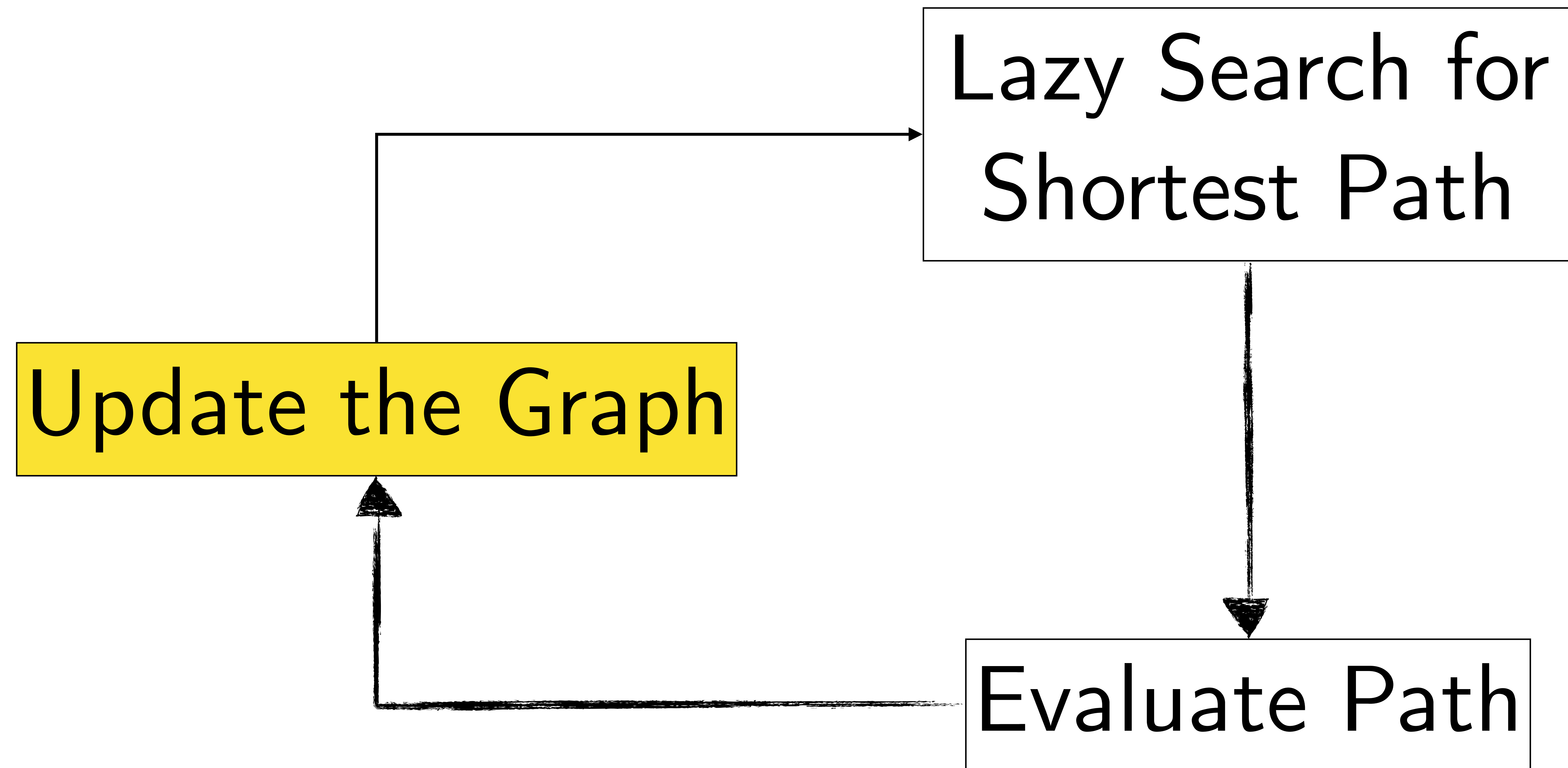
Optimism Under Uncertainty





LazySP

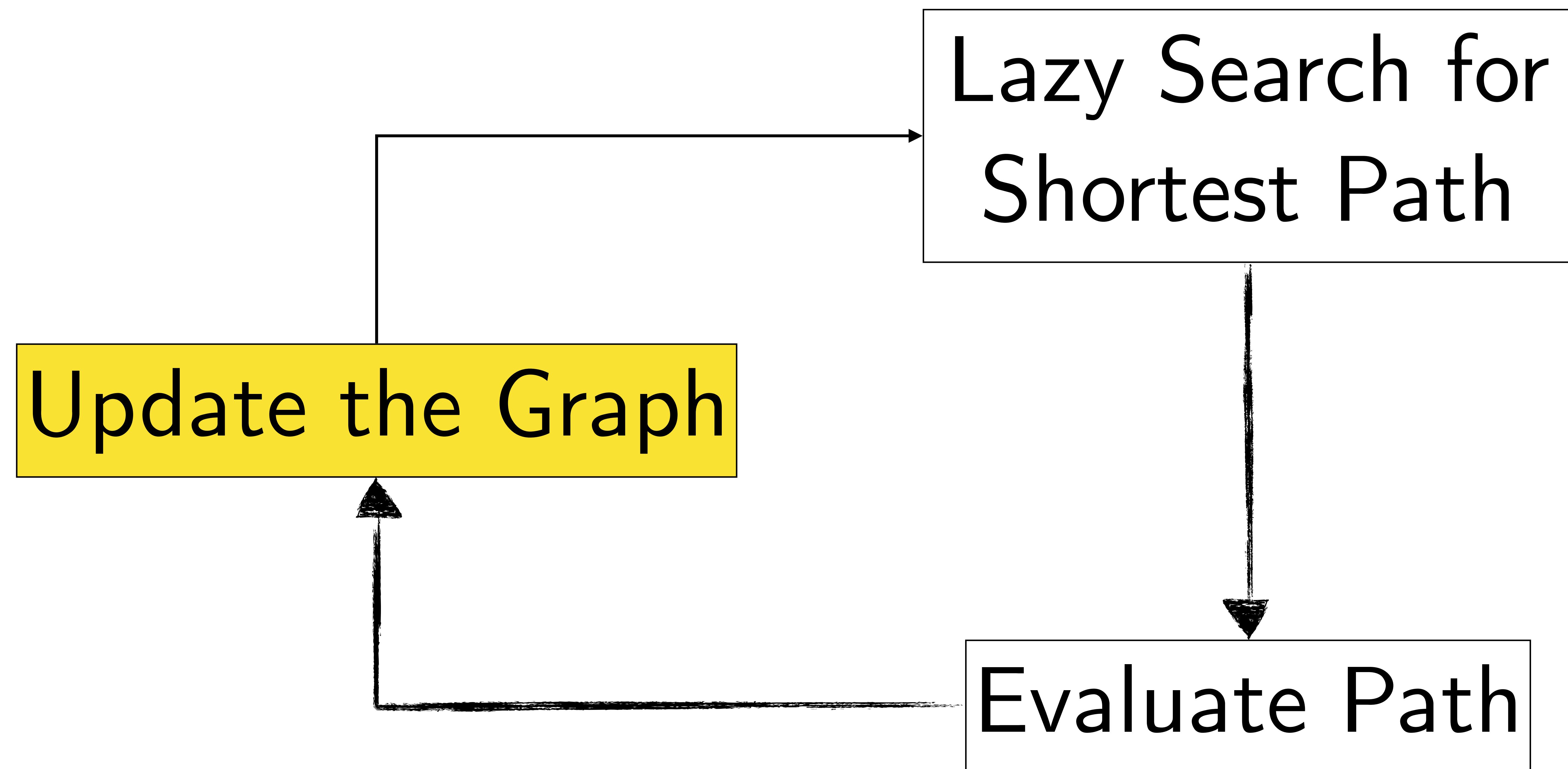
Optimism Under Uncertainty

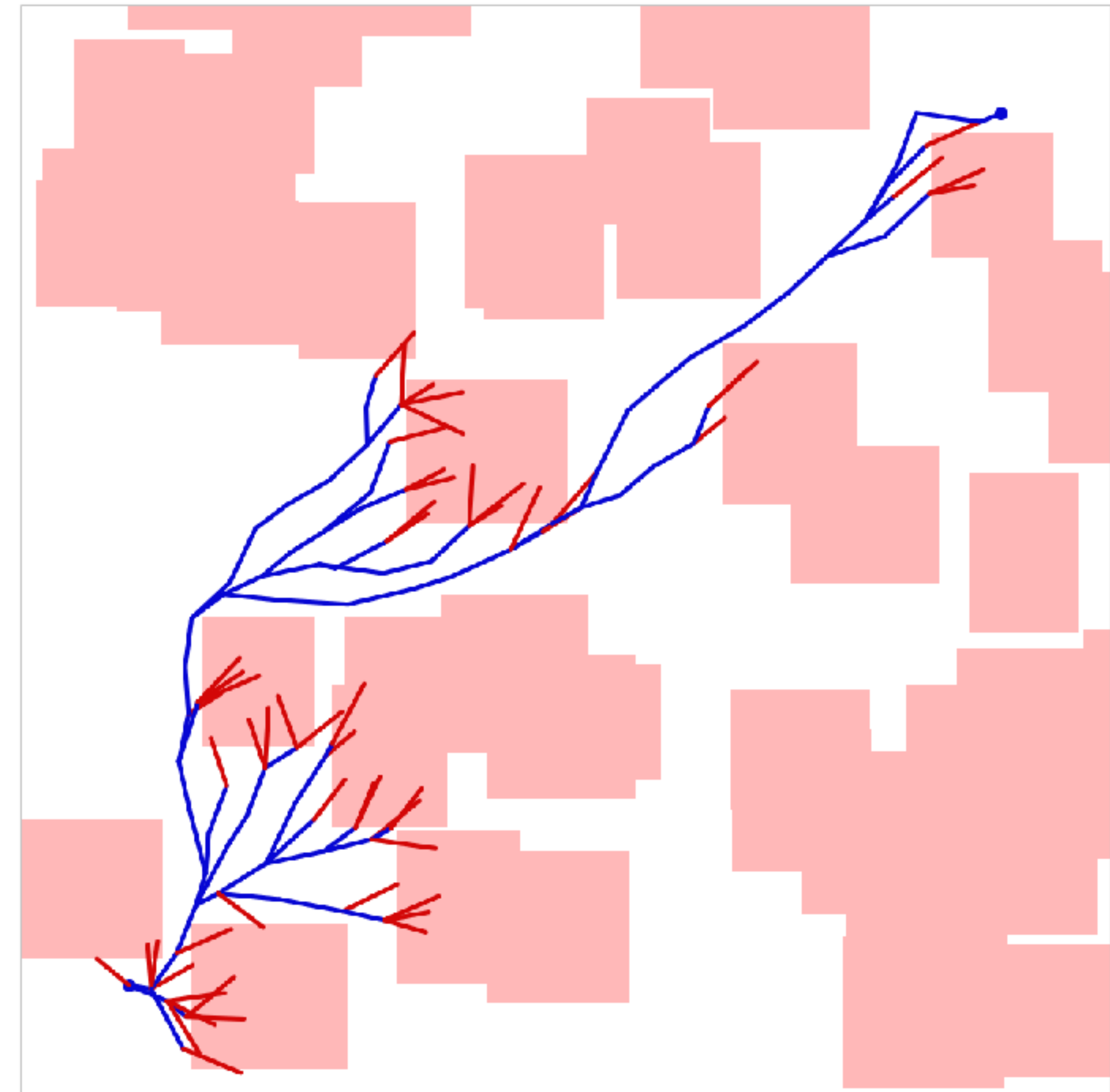


....

LazySP

Optimism Under Uncertainty



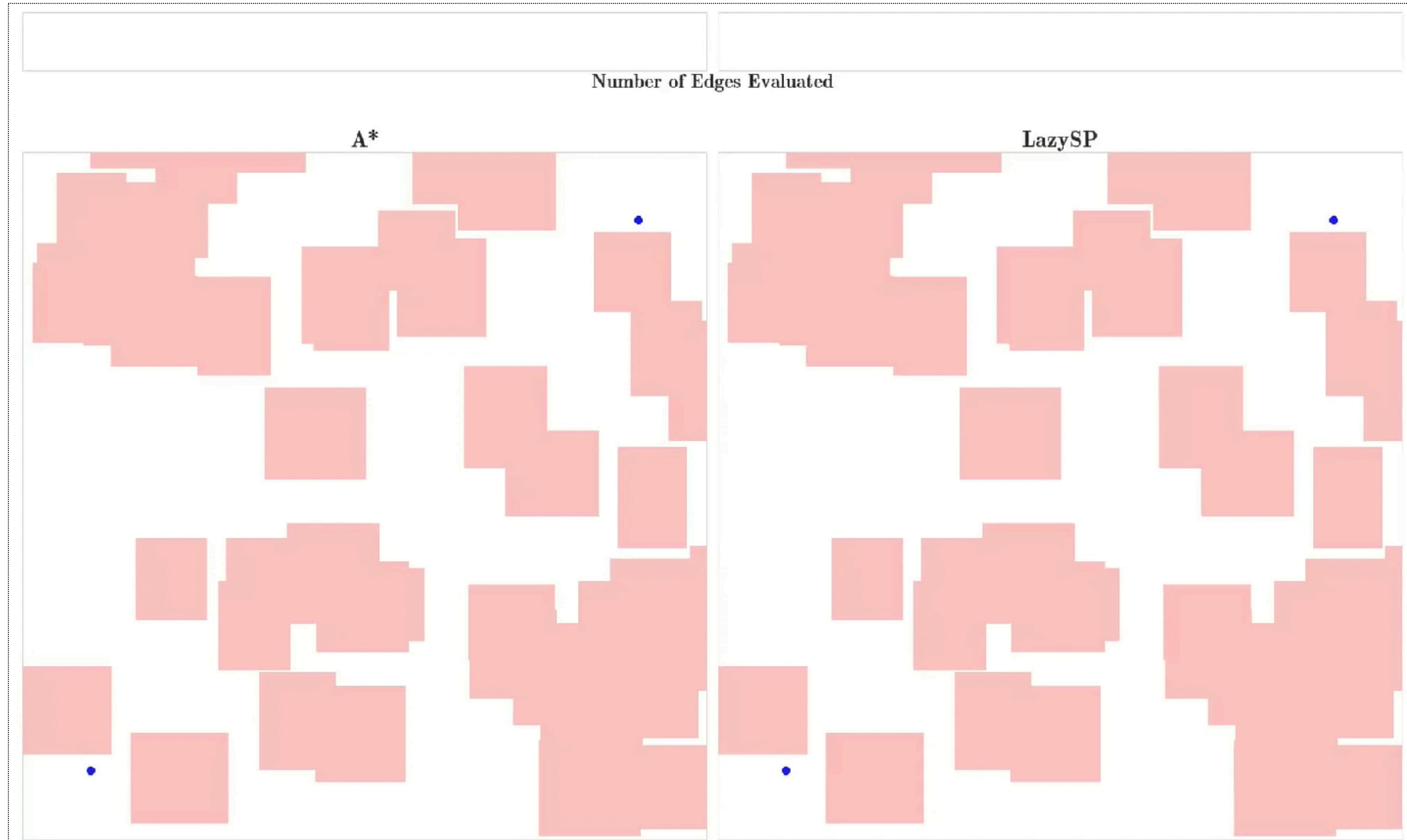


LazySP

Optimism Under Uncertainty

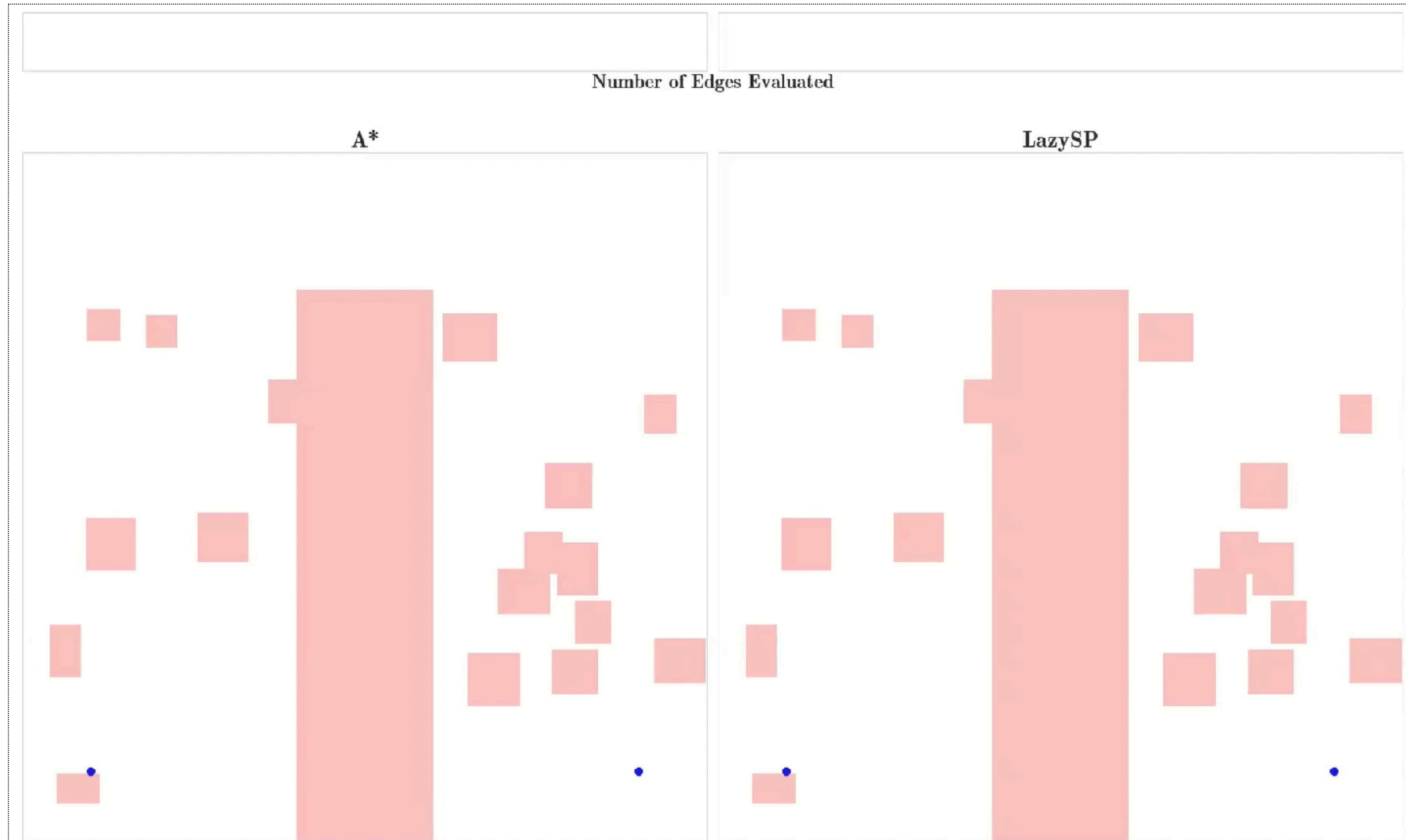
Return shortest feasible path!

A* vs LazySP

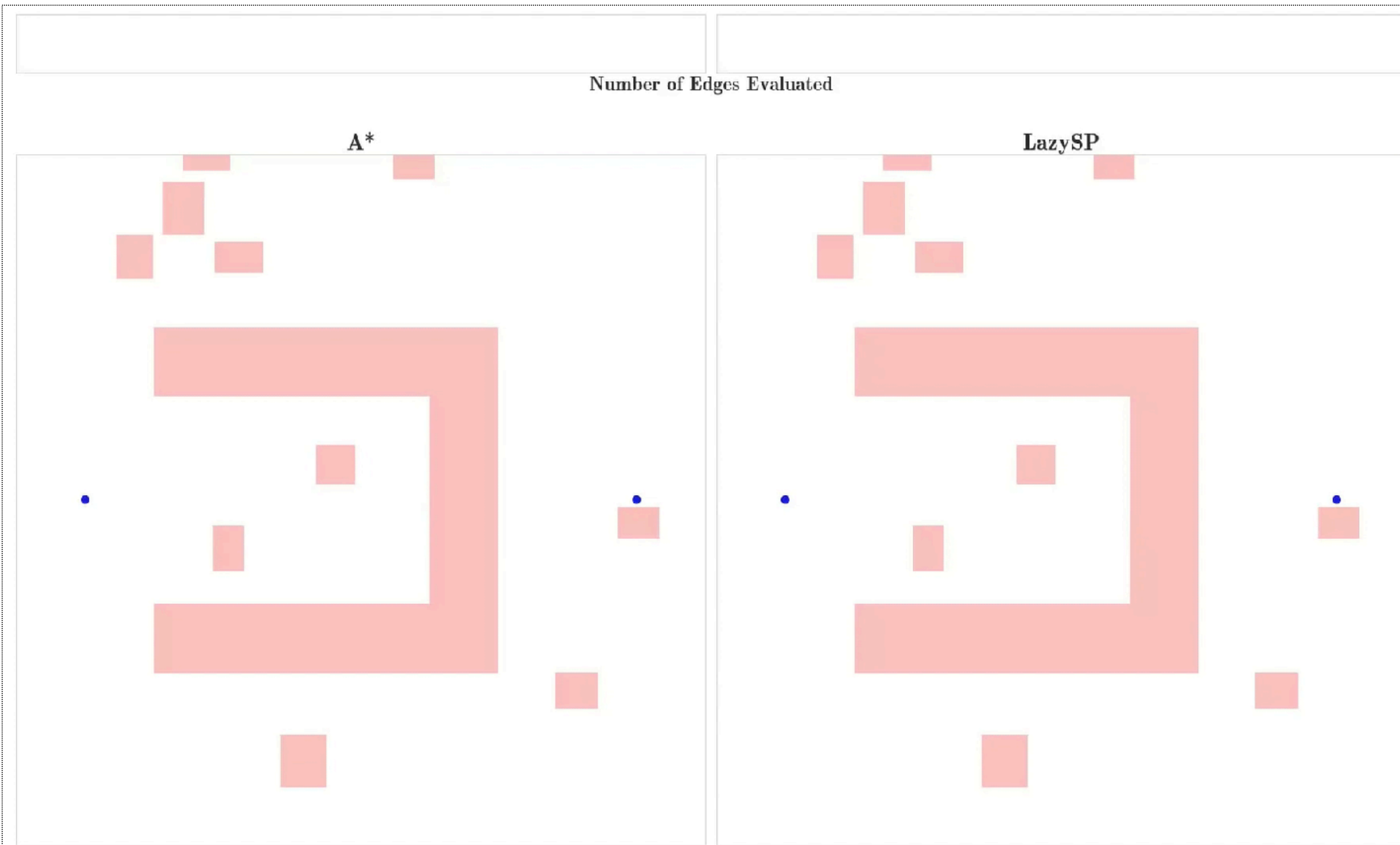


A* vs LazySP

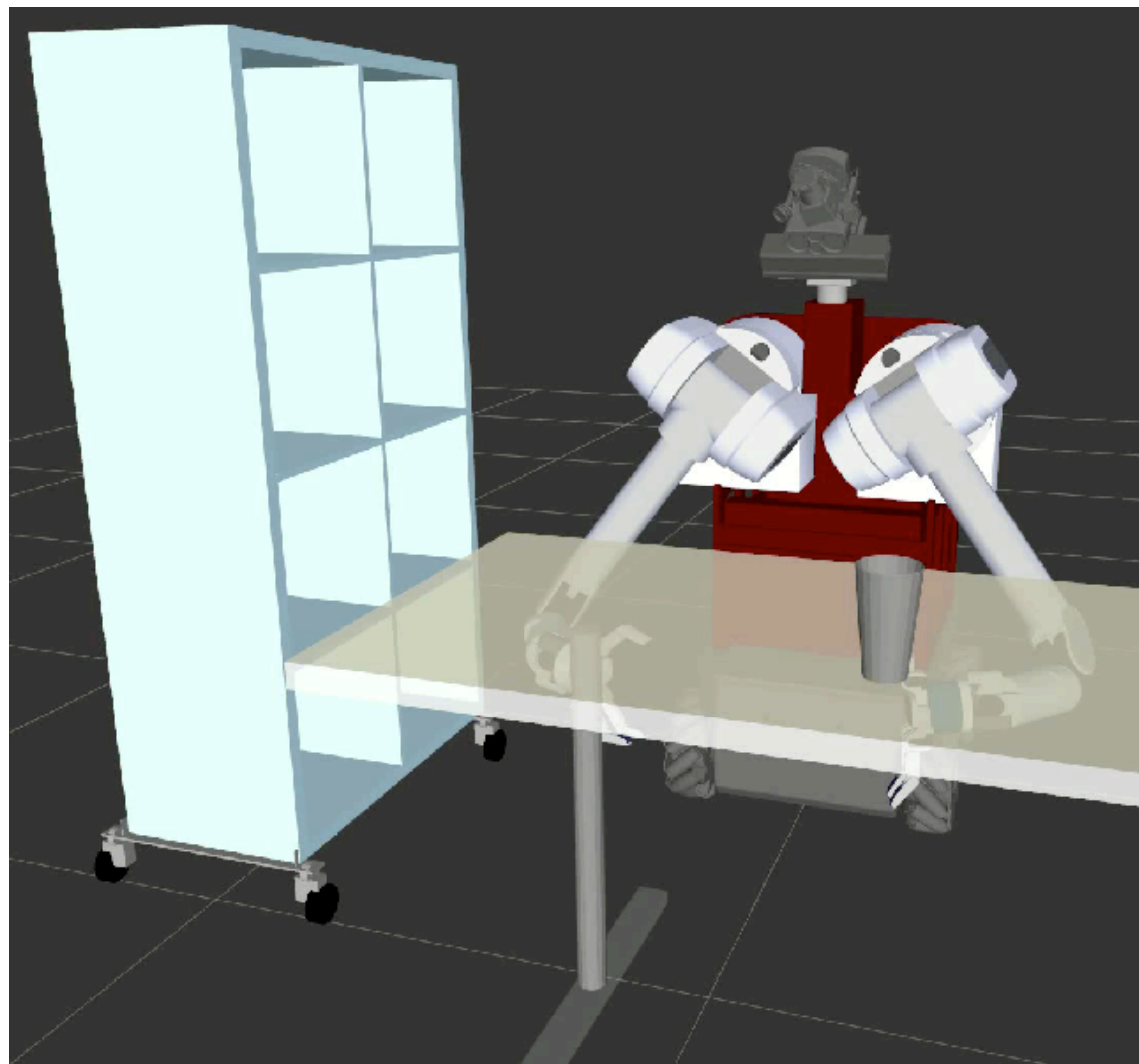
A* vs LazySP



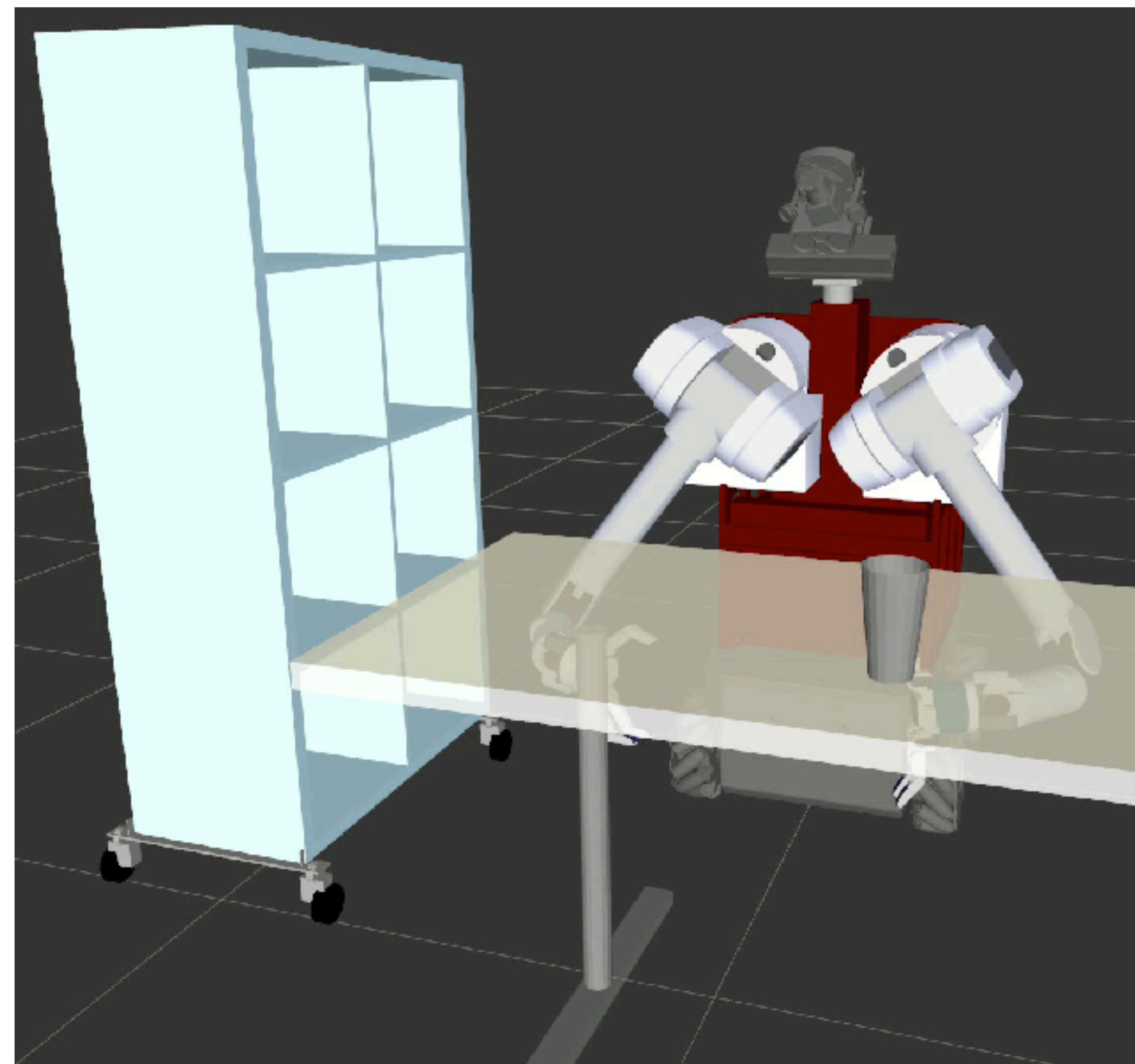
A* vs LazySP



A* vs LazySP



A* (191 edges)



LazySP (38 edges)

What can we prove about Lazy SP?

LazySP finds the optimal path

LazySP evaluates the minimal number of edges

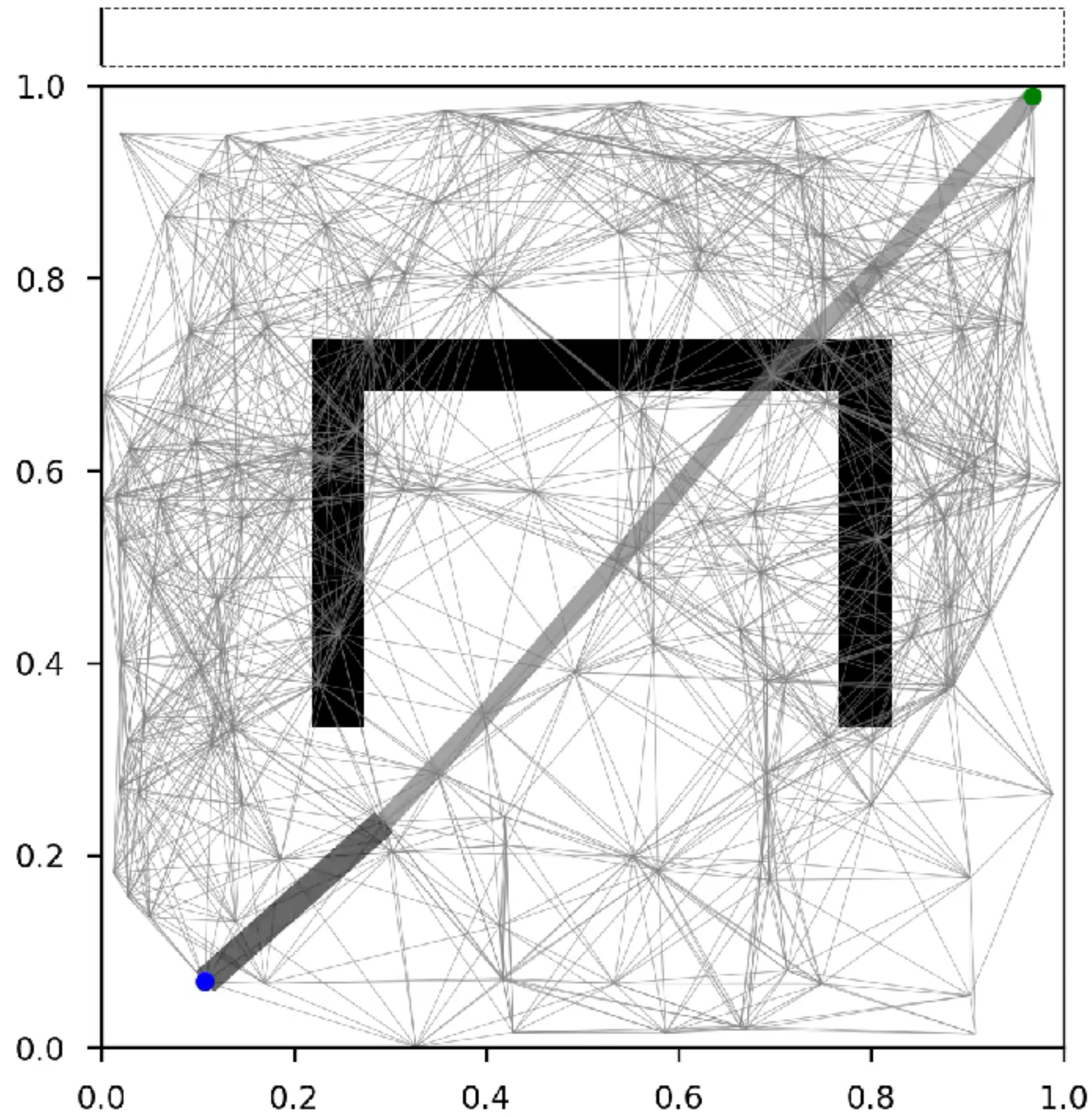
(For a given edge selector policy)

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

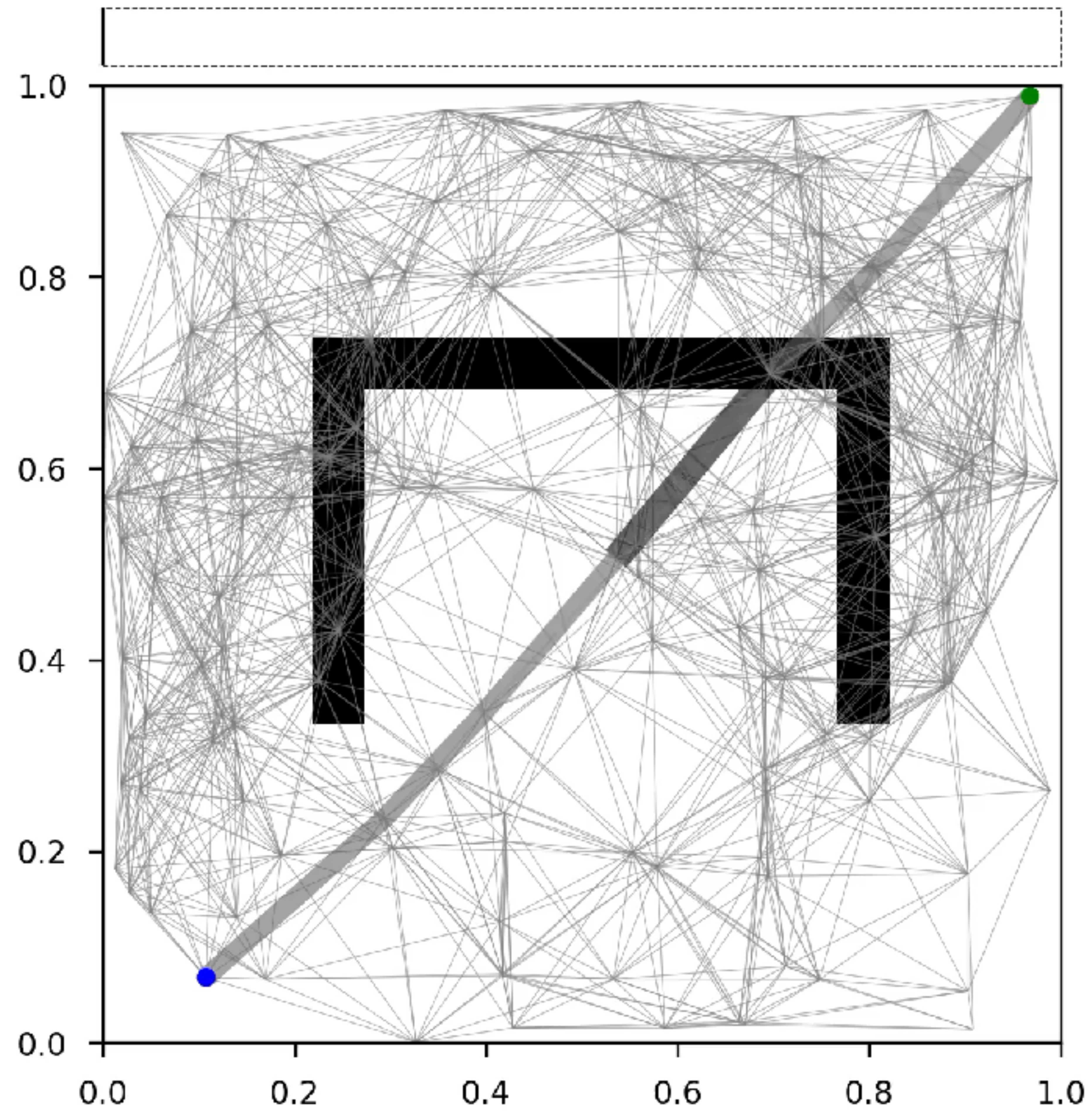
Leveraging Experience in Lazy Search

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

Learn which edges to evaluate (STROLL)

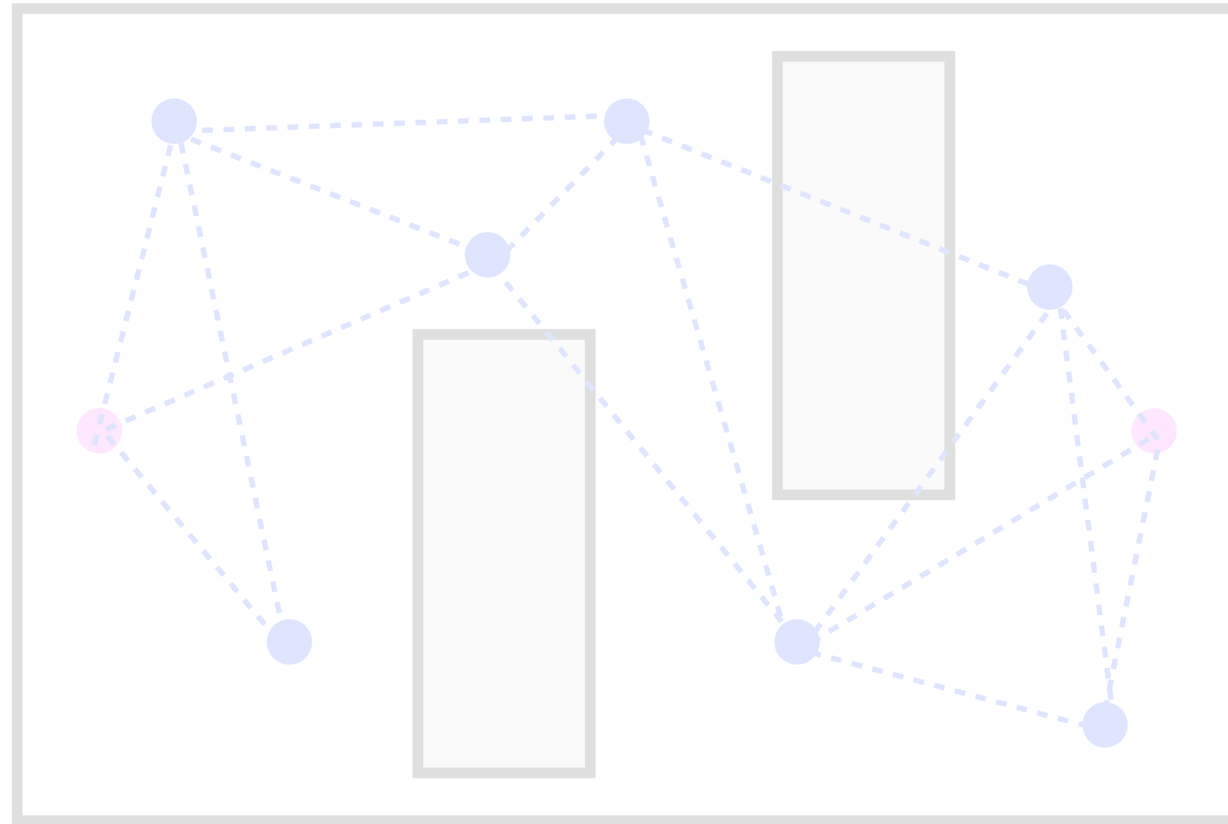


LazySP

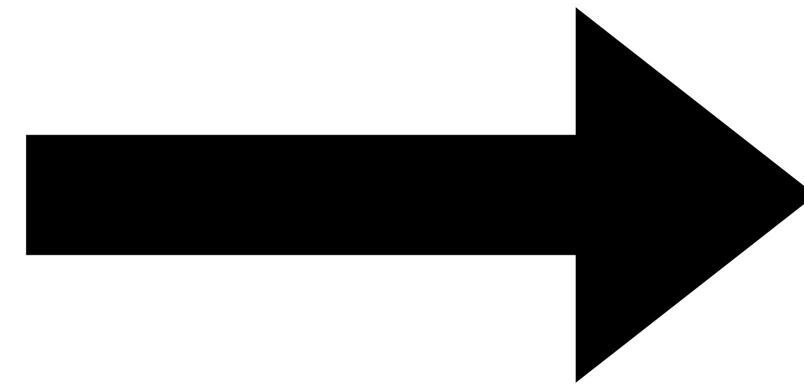


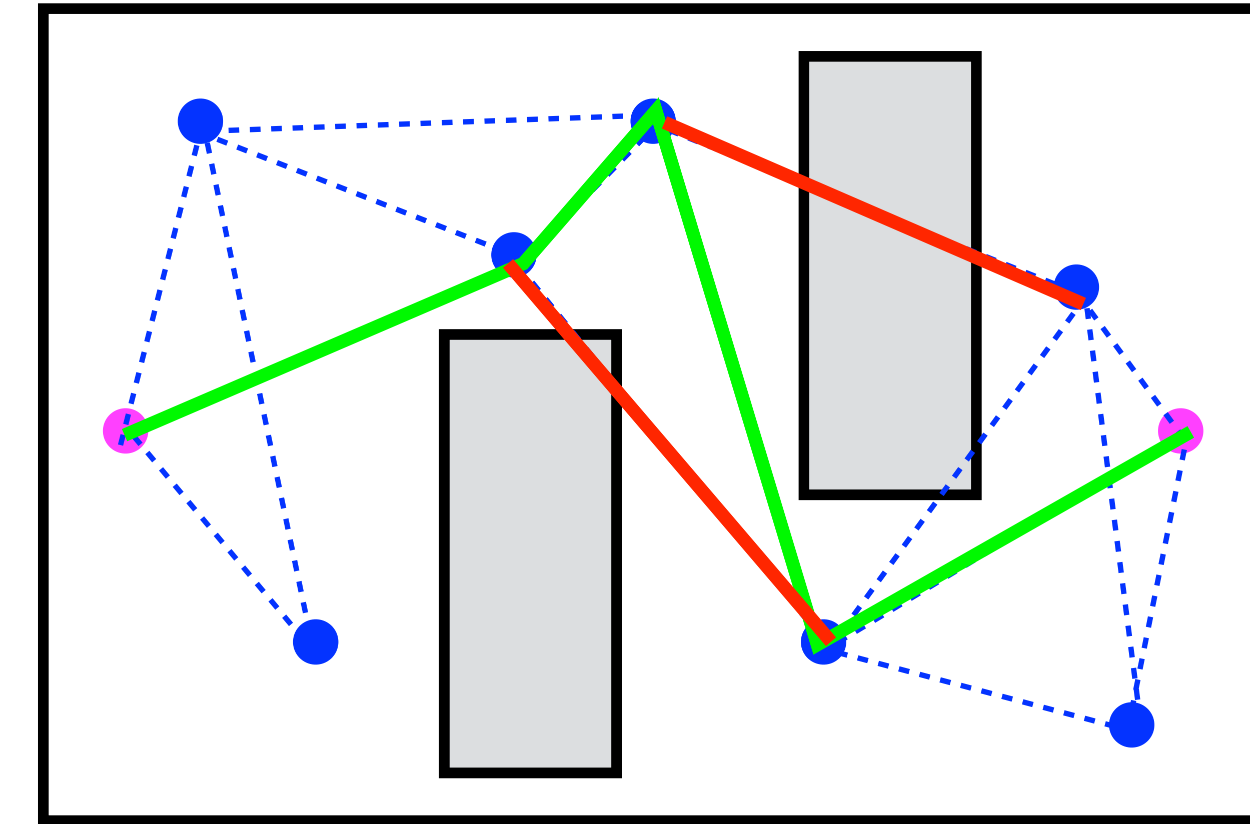
STROLL

General framework for motion planning



Create a graph

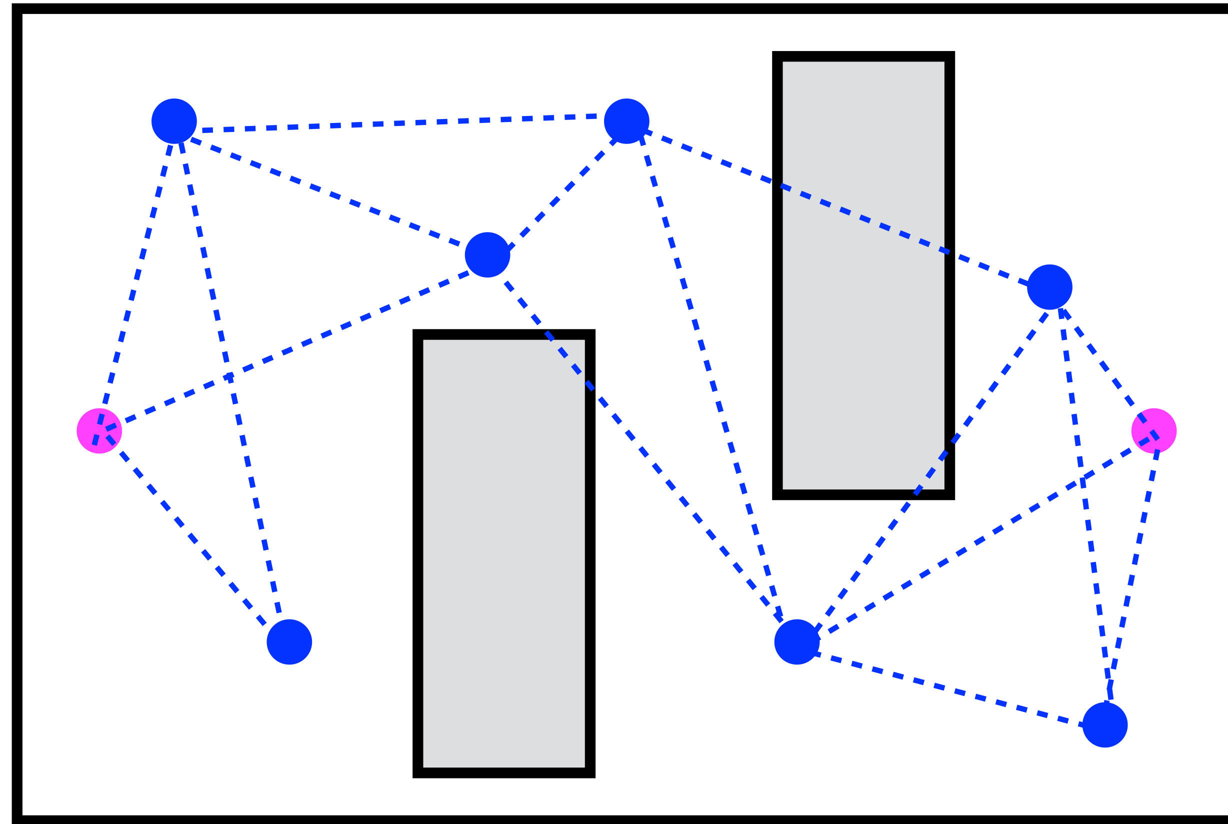




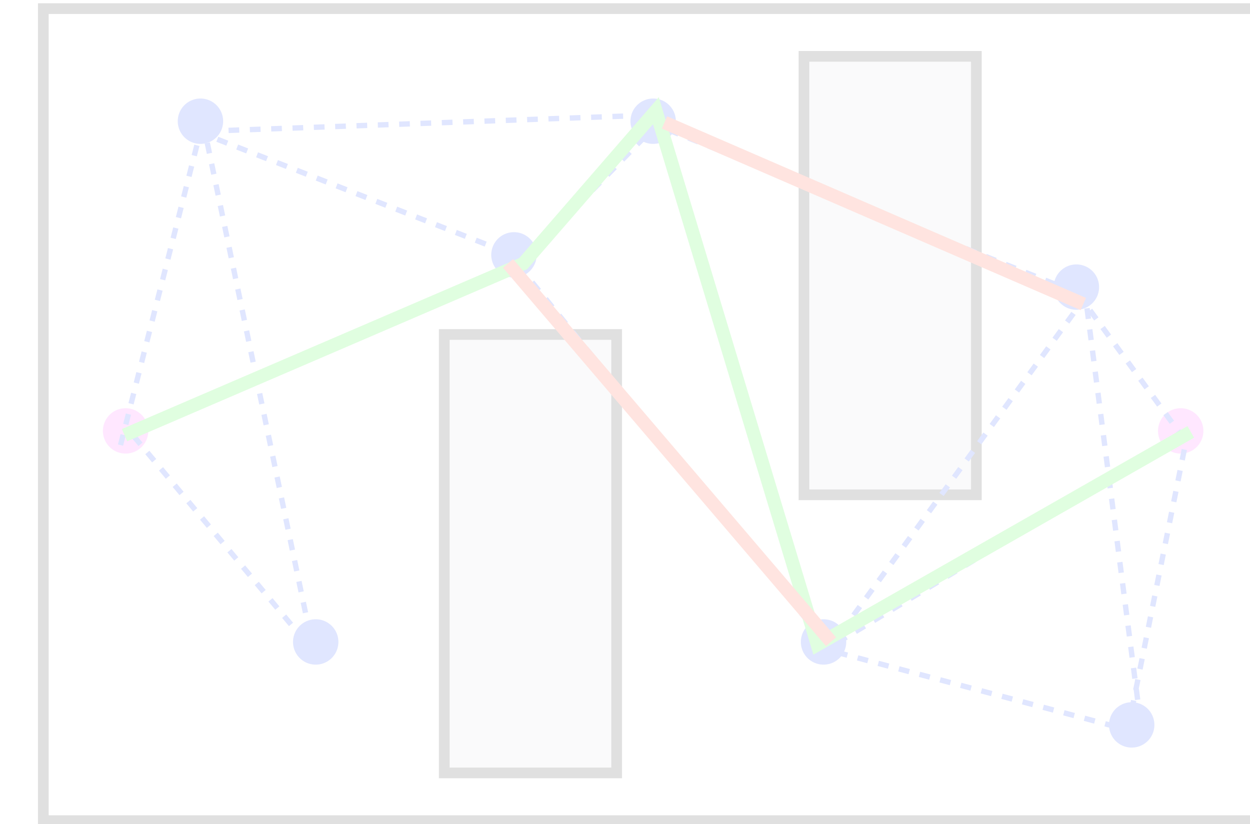
Search the graph

Interleave

General framework for motion planning



Create a graph



Search the graph

Interleave

Creating a graph: Abstract algorithm

$$G = (V, E)$$

Vertices: set of configurations

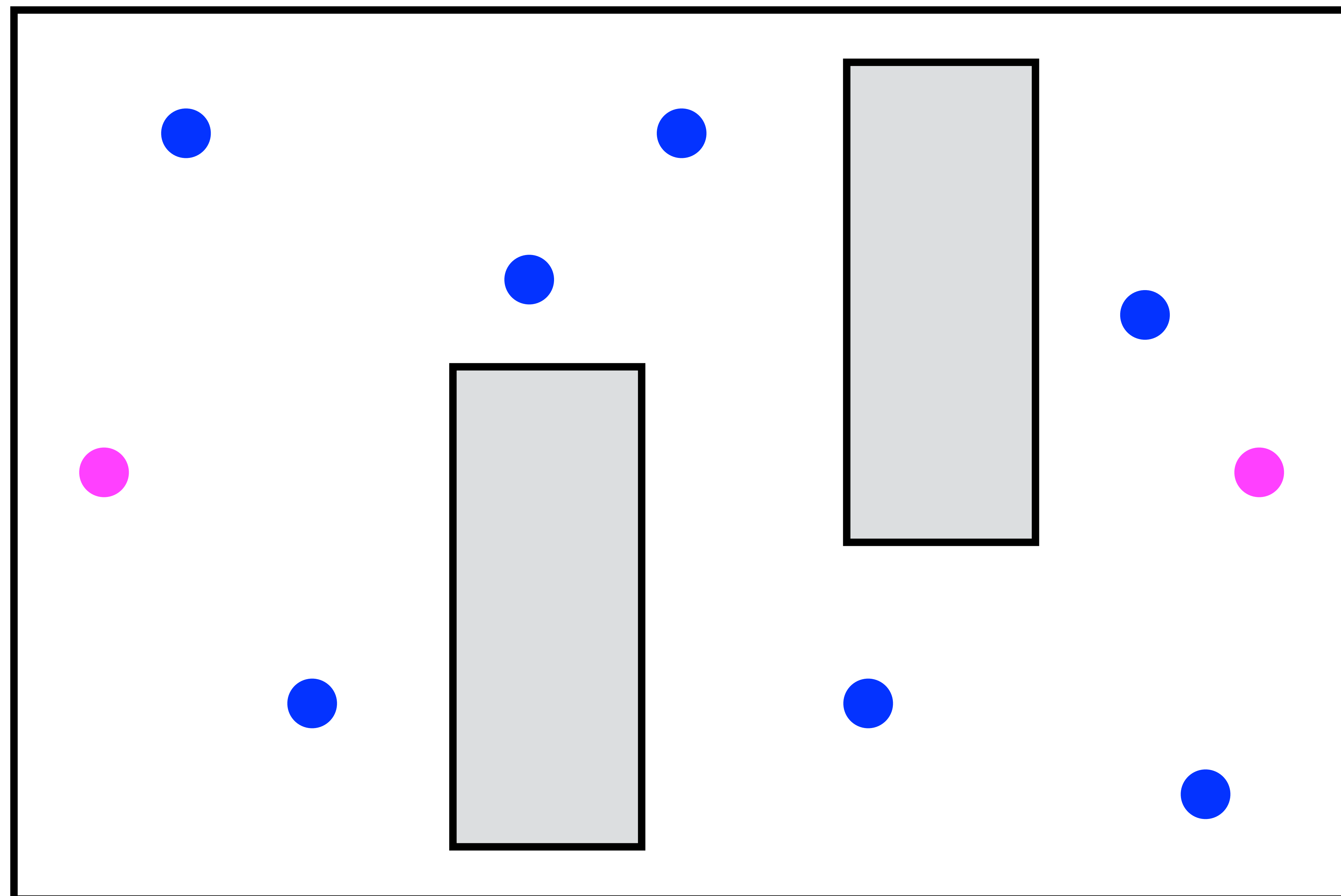
Edges: paths connecting configurations

Creating a graph: Abstract algorithm

$$G = (V, E)$$

Vertices: set of configurations

Edges: paths connecting configurations



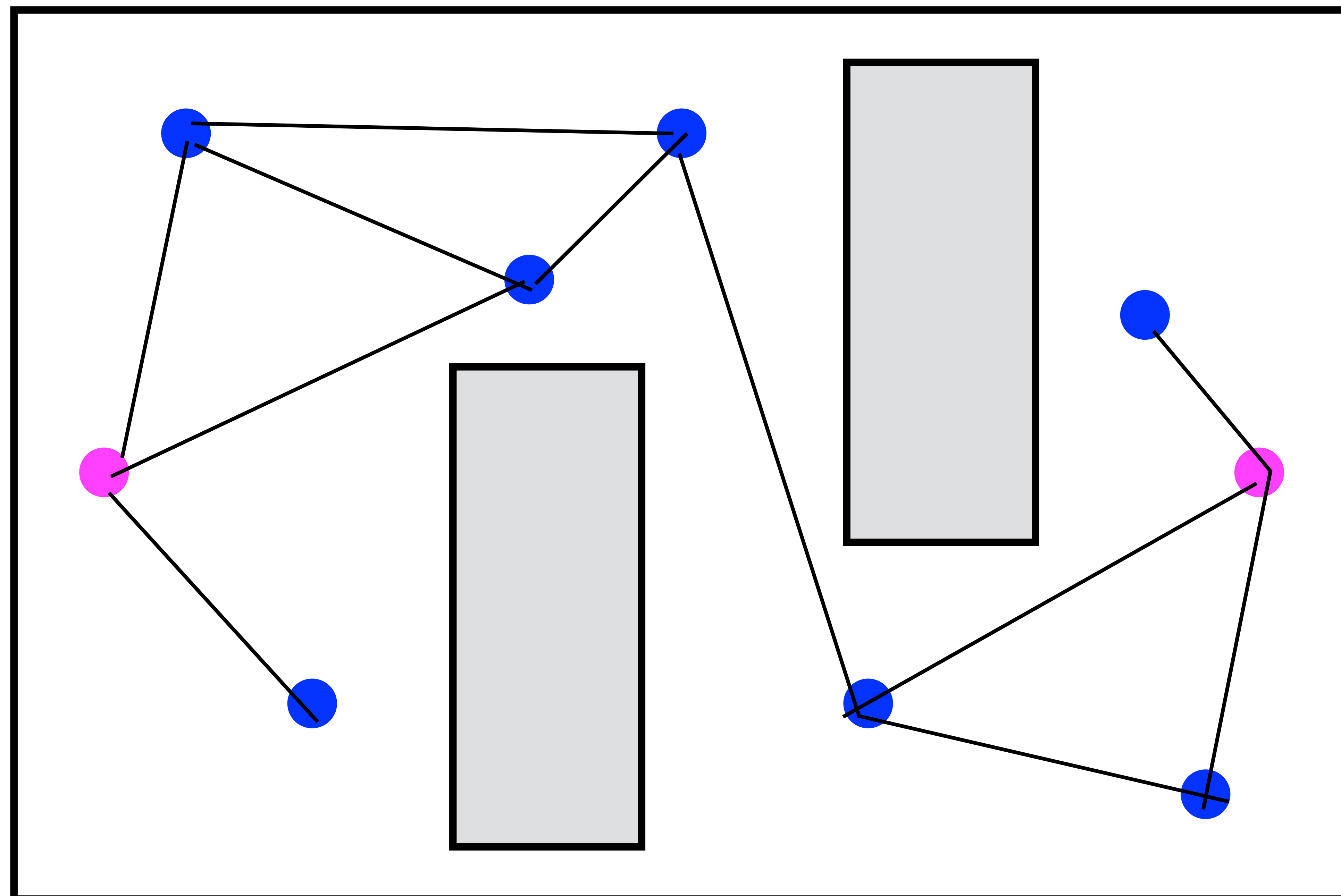
1. Sample a set of collision free vertices V (add start and goal)

Creating a graph: Abstract algorithm

$$G = (V, E)$$

Vertices: set of configurations

Edges: paths connecting configurations

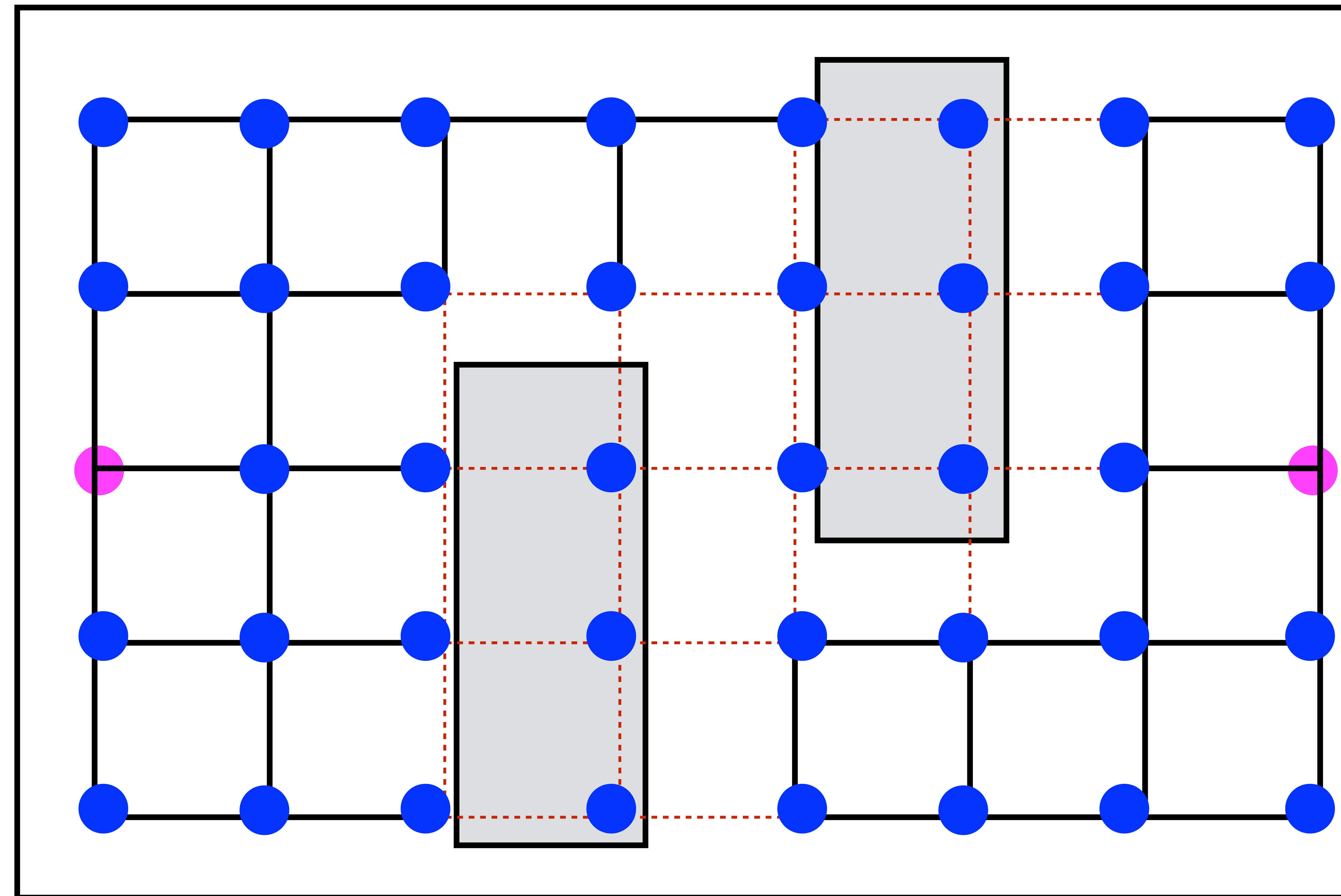


1. Sample a set of collision free vertices V (add start and goal)

2. Connect "neighboring" vertices to get edges E

Strategy 1: Discretize configuration space

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

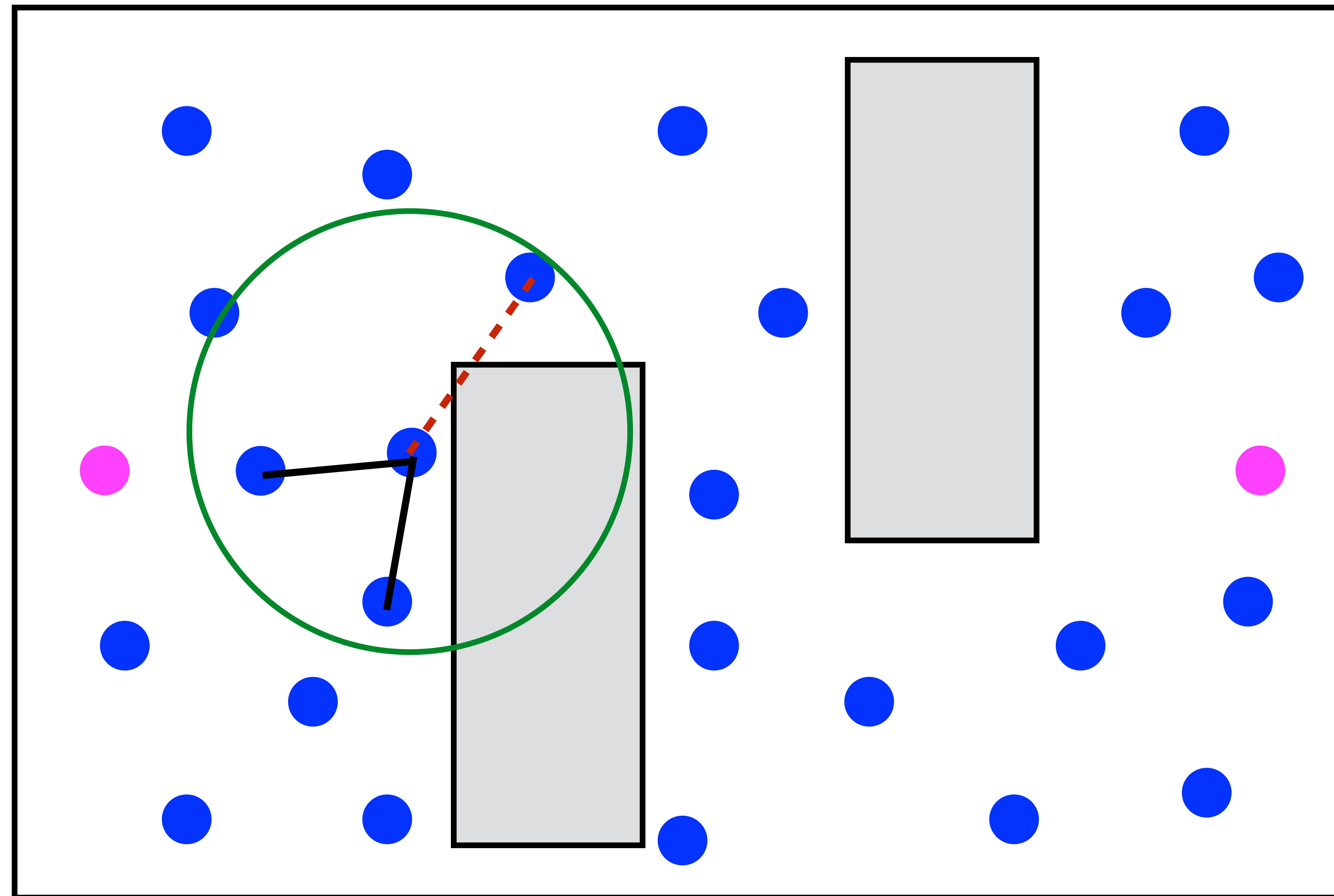


Theoretical guarantees: Resolution complete

What are the pros? What are the cons?

Strategy 2: Uniformly randomly sample

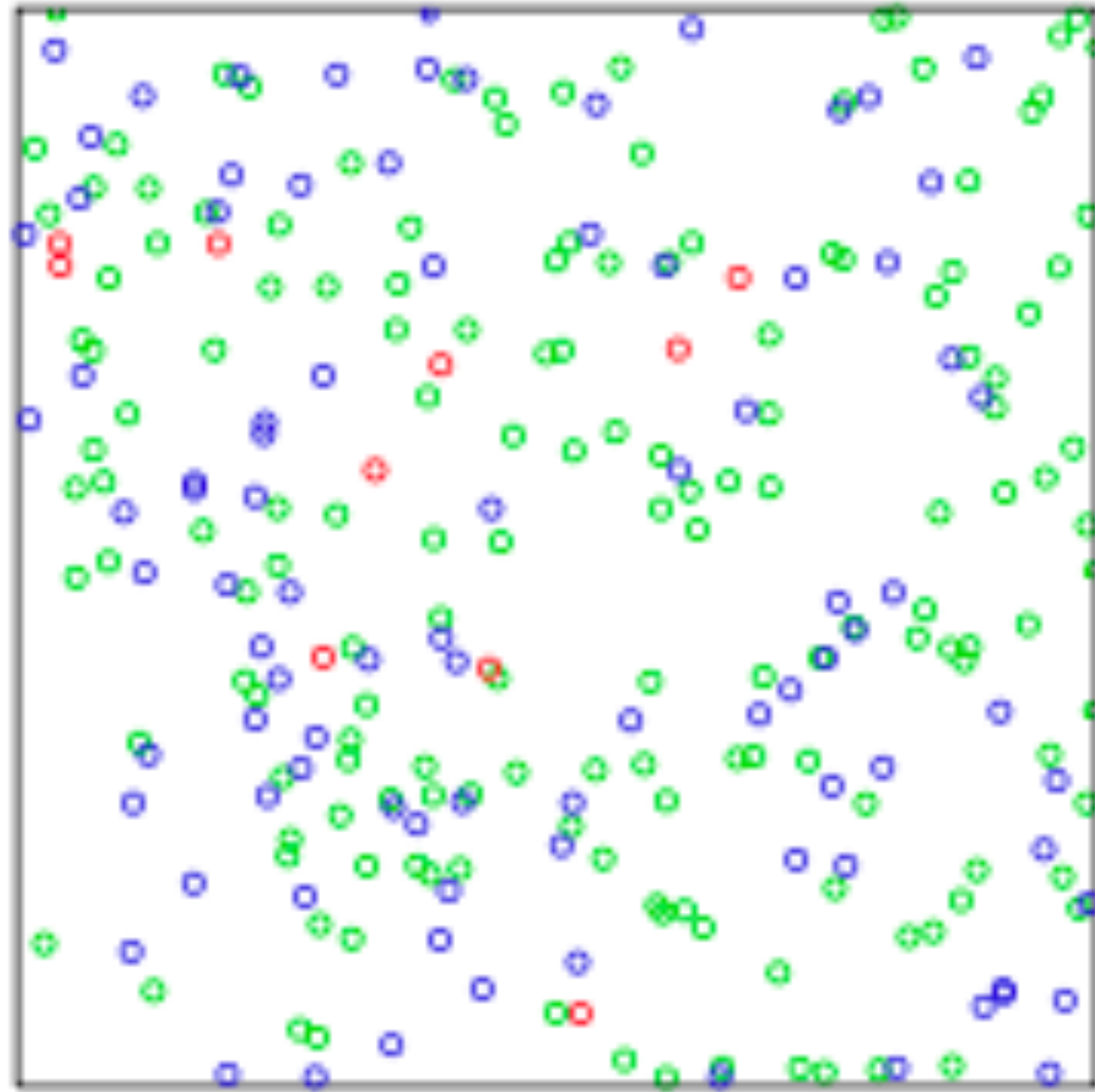
Randomly sample points. Connect all neighbors in a ball!



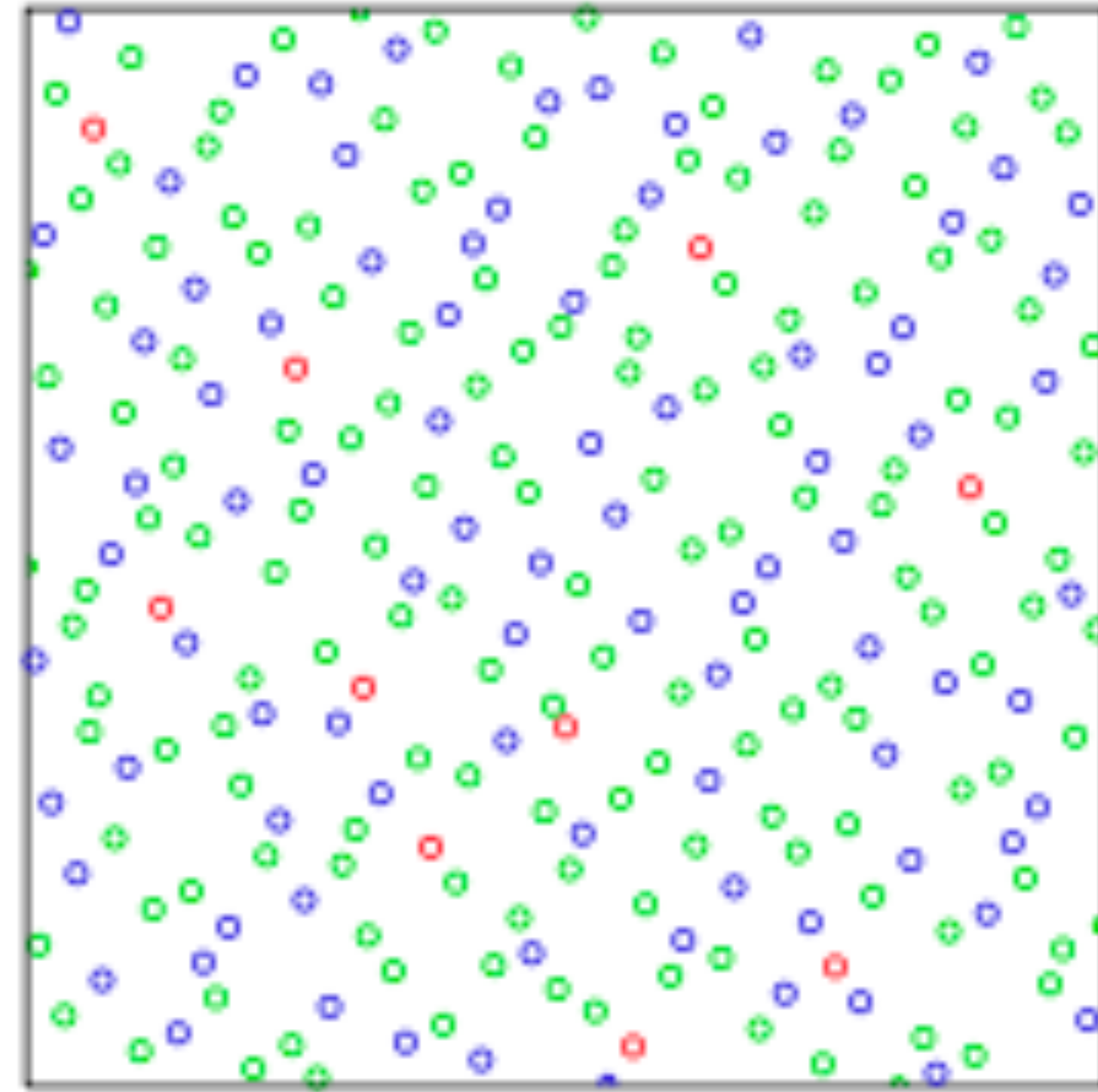
Theoretical guarantees: Probabilistically complete

What are the pros? What are the cons?

Can we do better than random?



Uniform random sampling tends to clump

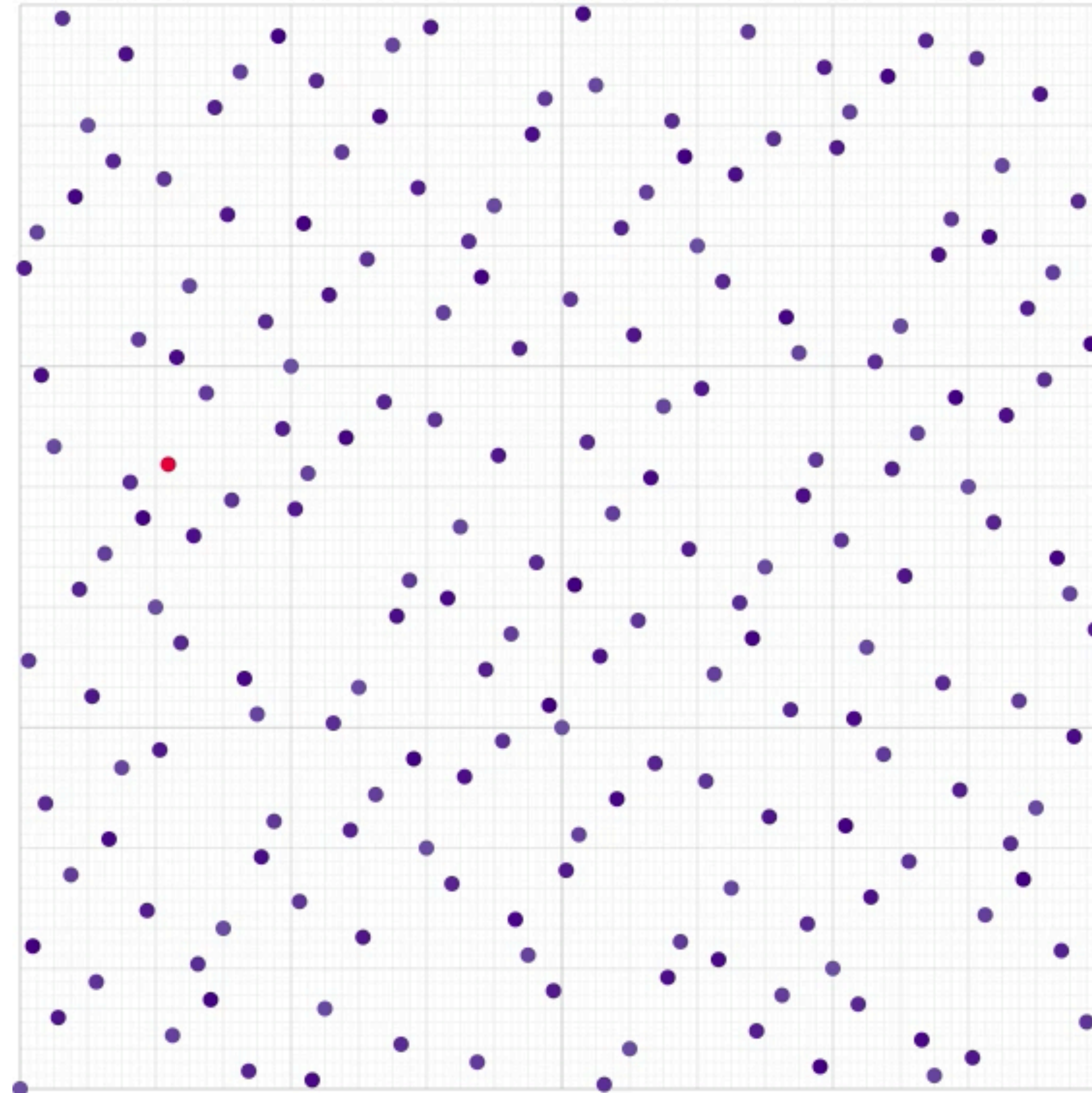


Ideally we would want points to be spread out evenly

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)