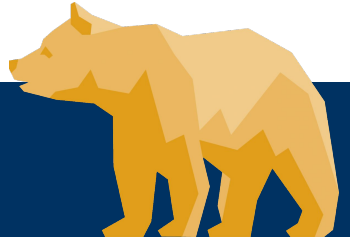
Motion Planning

Laura Hallock

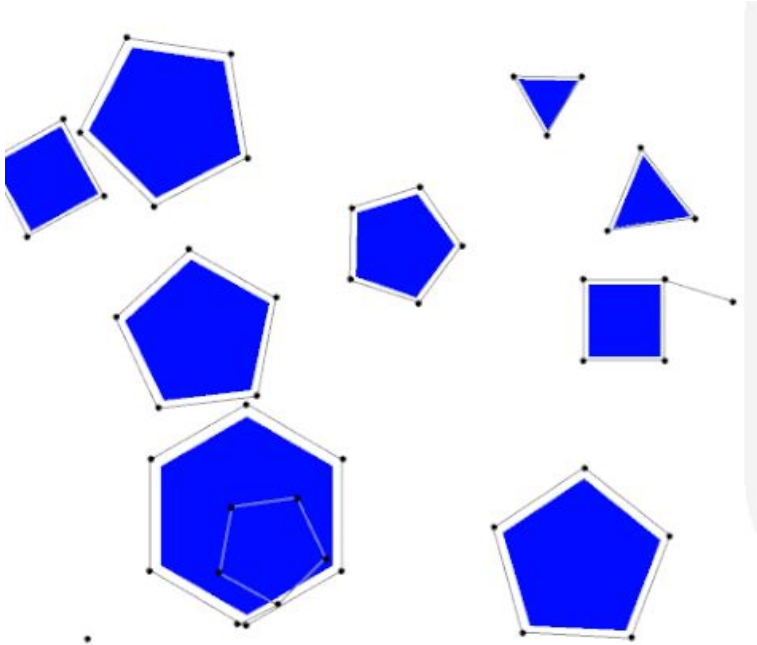
EECS 106A/206A / BIOE 106A

Oct. 24, 2017



Roadmap

- Problem definition
- Intro: **A*** on a grid
- Configuration space
- Solution 1: Sampling-based planners
  - **PRM** (+ A*)
  - **RRT**
  - **RRT***
- Solution 2: Optimization-based planners
  - Problem formulation
  - **Shooting** methods
  - **Direct transcription** & **collocation** methods



(Eric O. Scott for Wikipedia)

Introduction

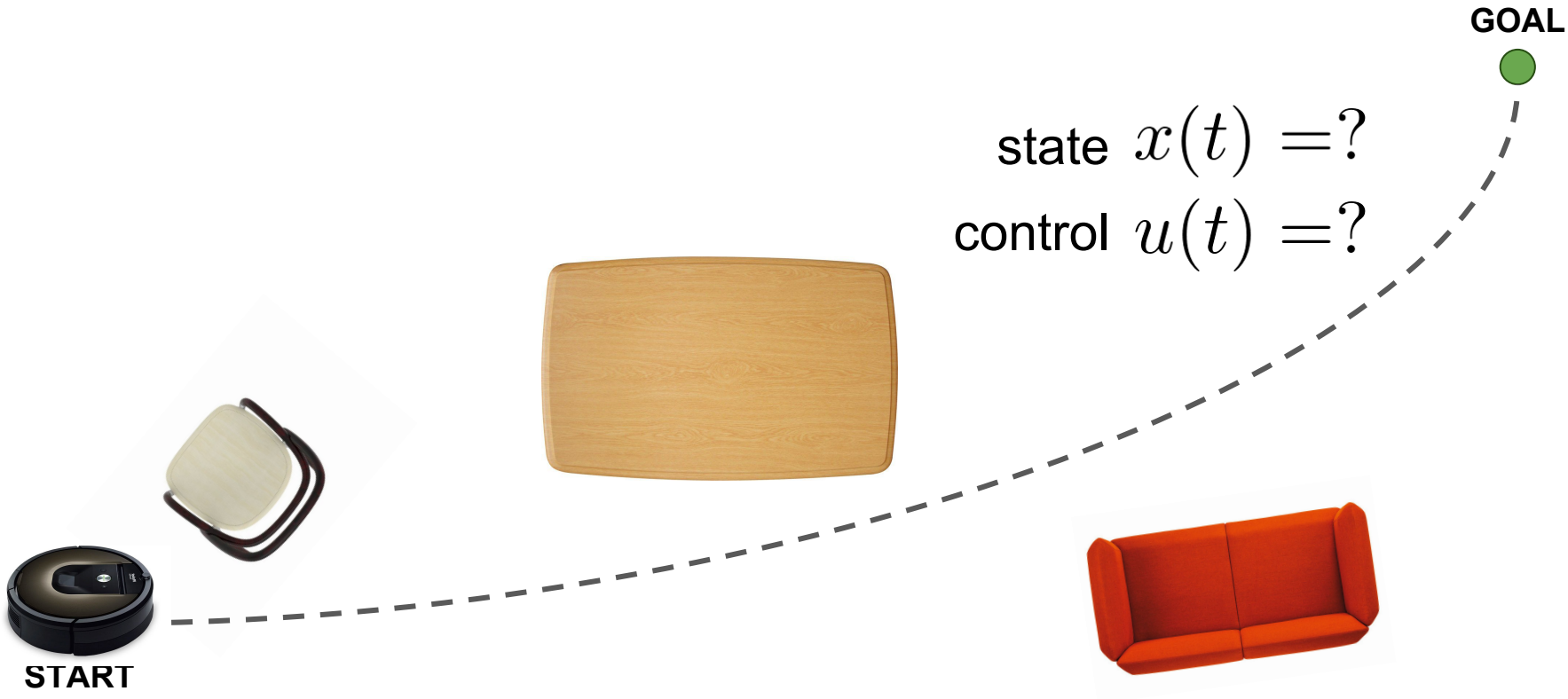
Problem Formulation, A*

What is motion planning?

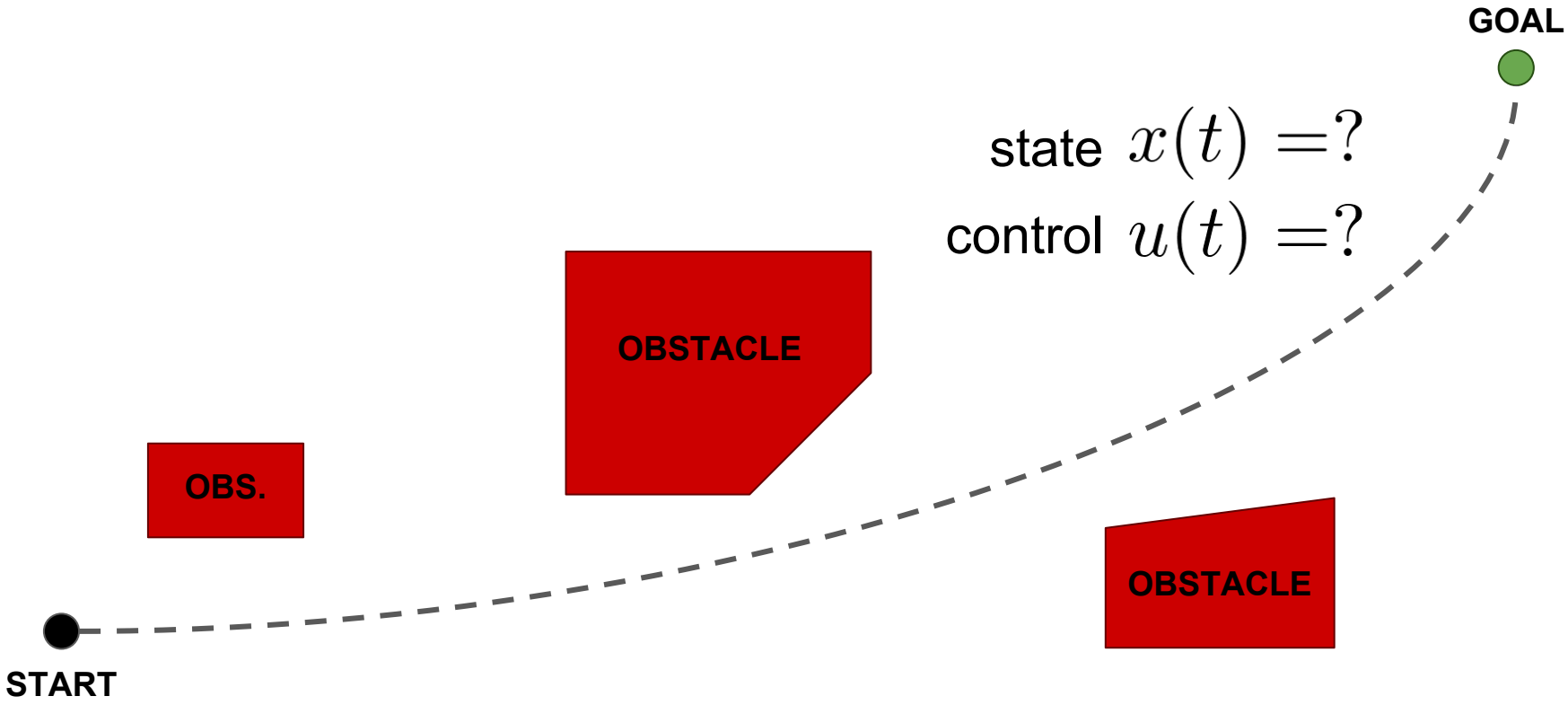
GOAL

START

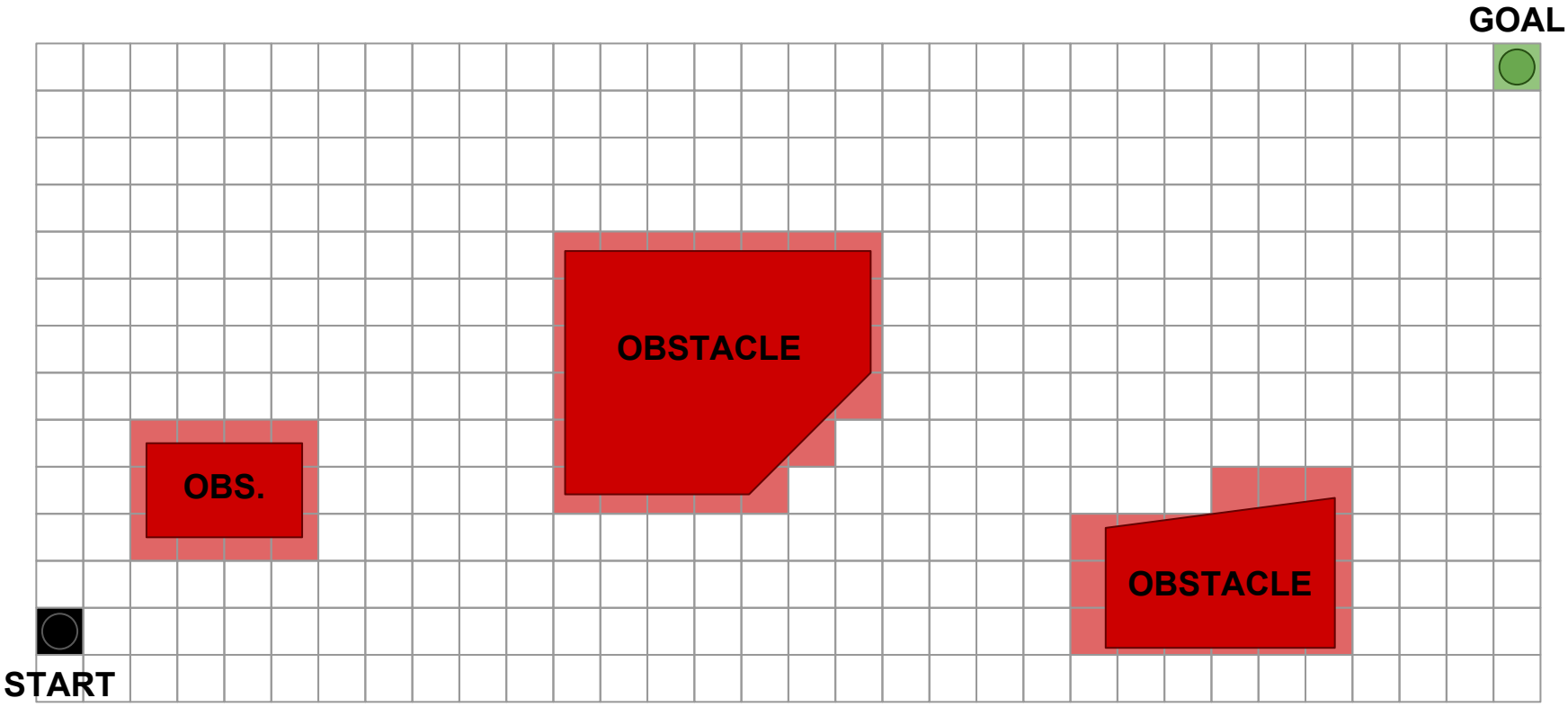
What is motion planning?



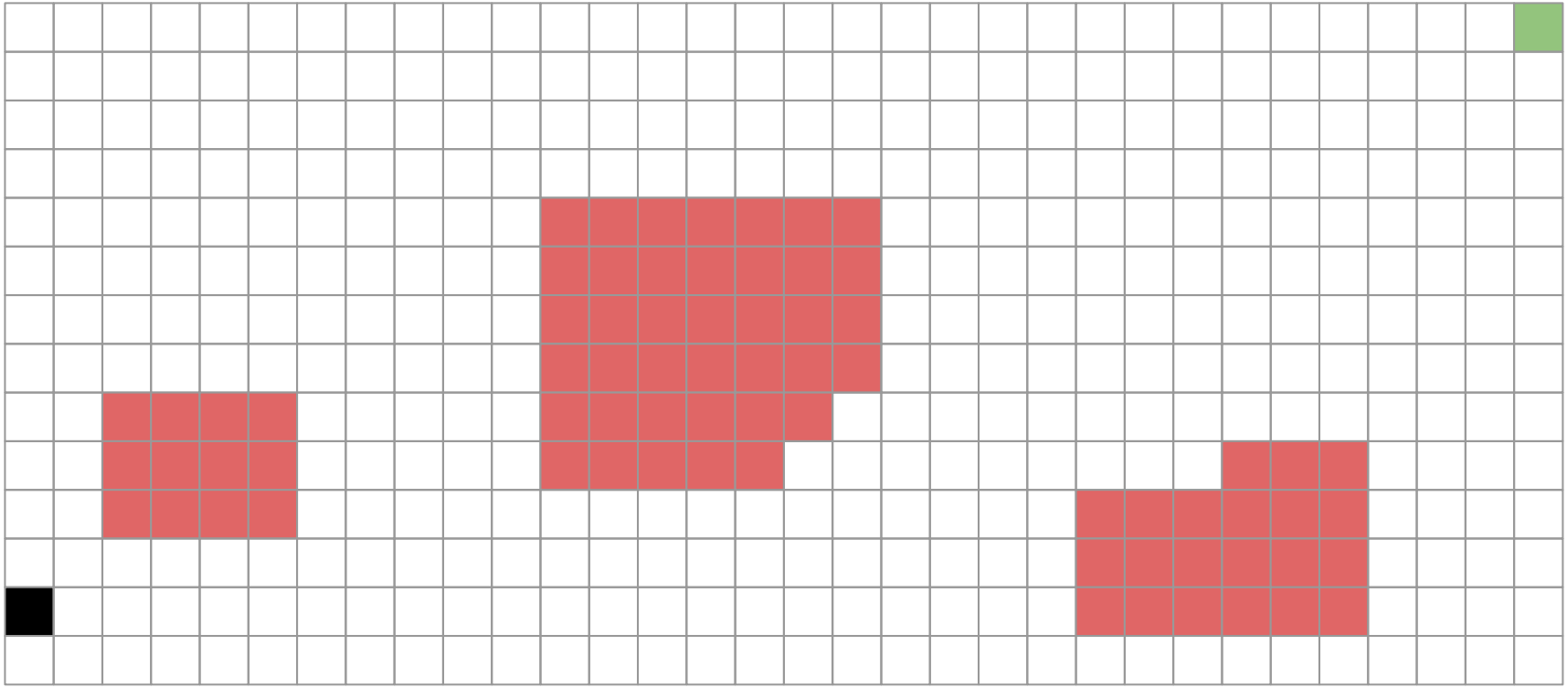
What is motion planning?



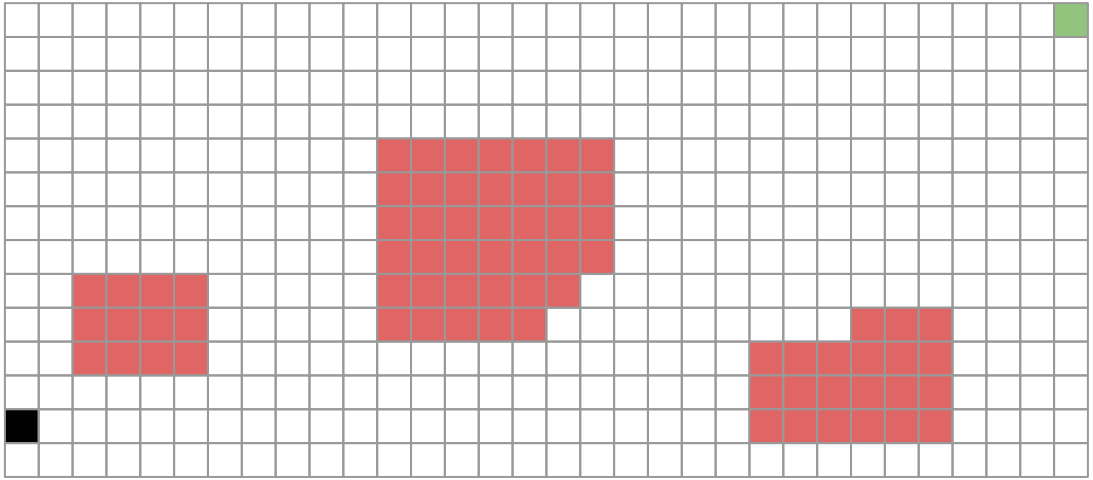
Simple idea: grid + search



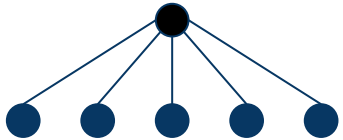
Simple idea: grid + search

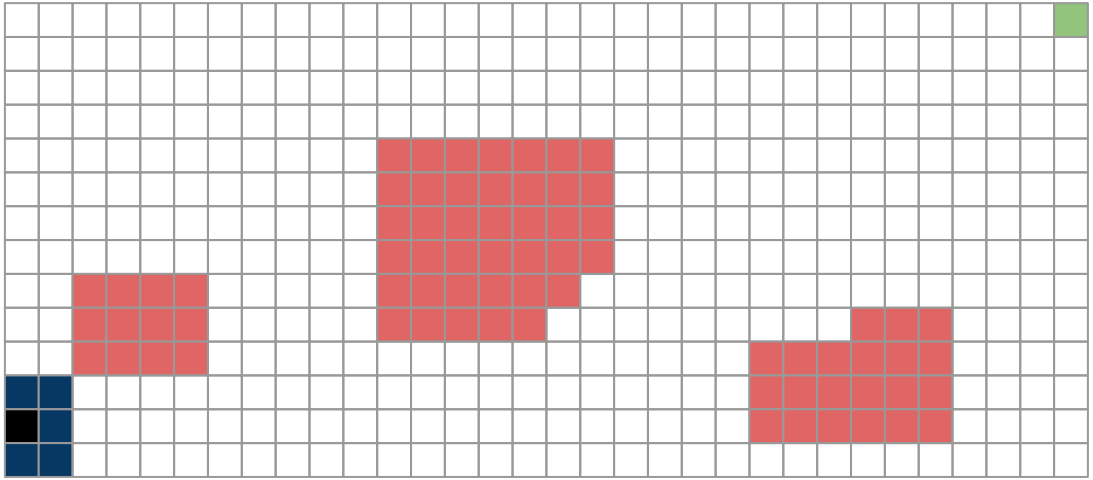


Idea: create a tree of possible paths

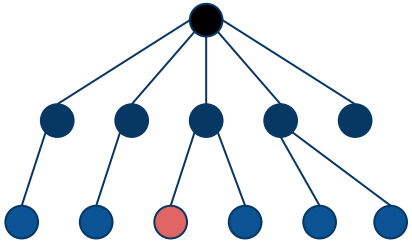


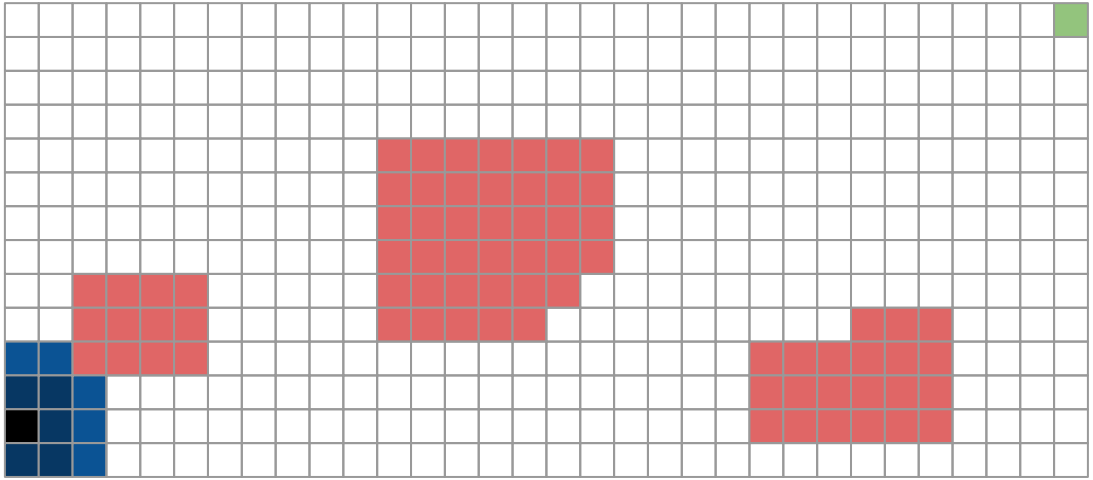
Idea: create a tree of possible paths



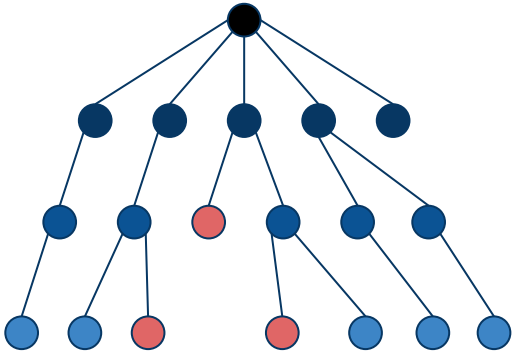


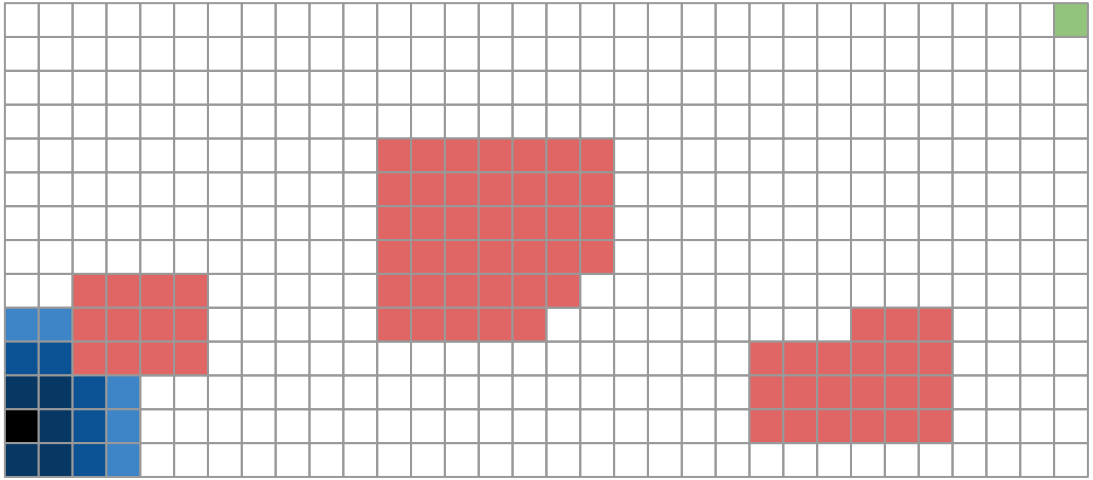
Idea: create a tree of possible paths



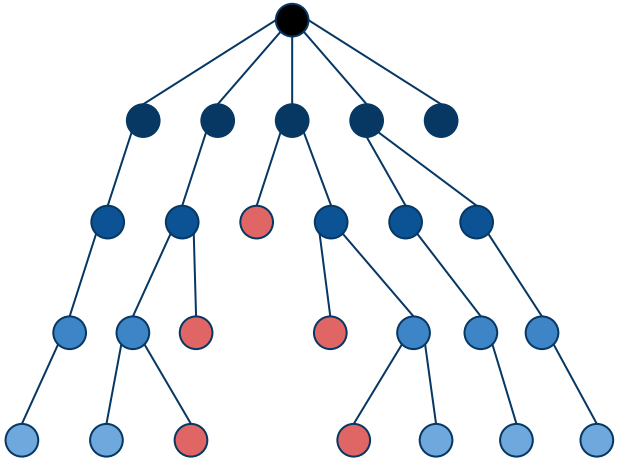


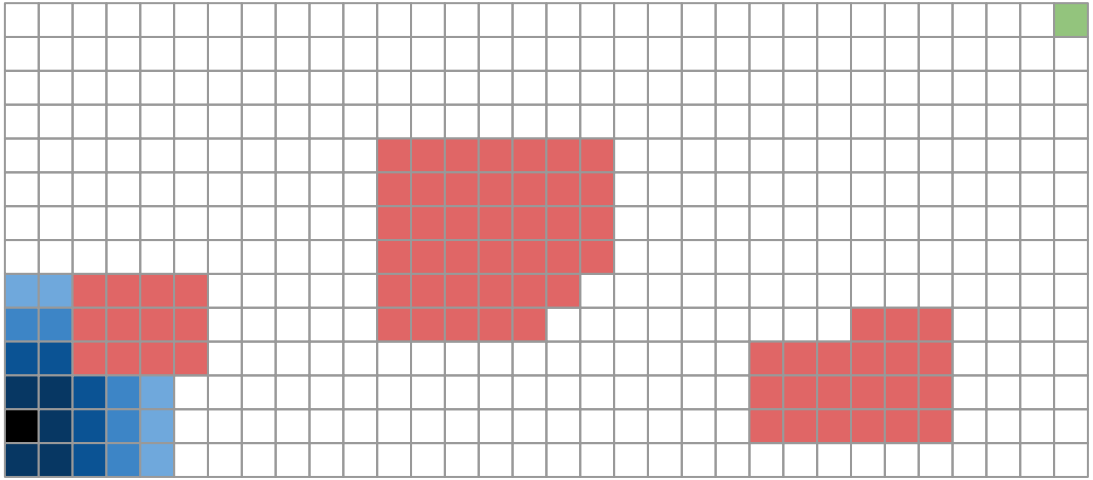
Idea: create a tree of possible paths



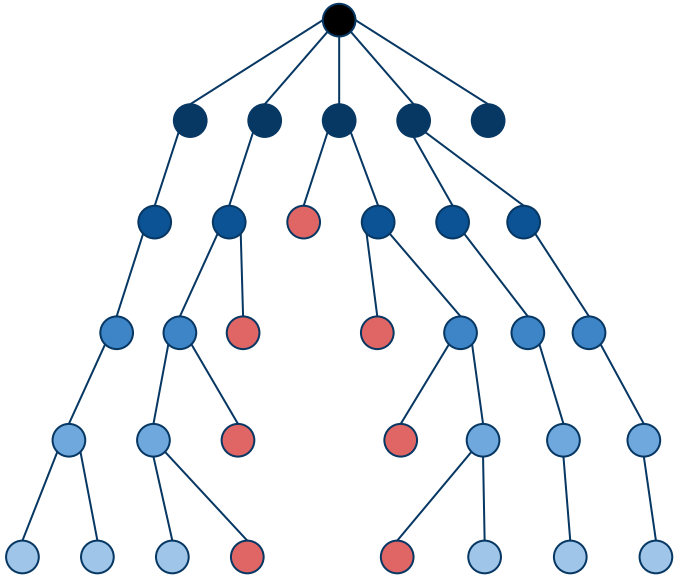


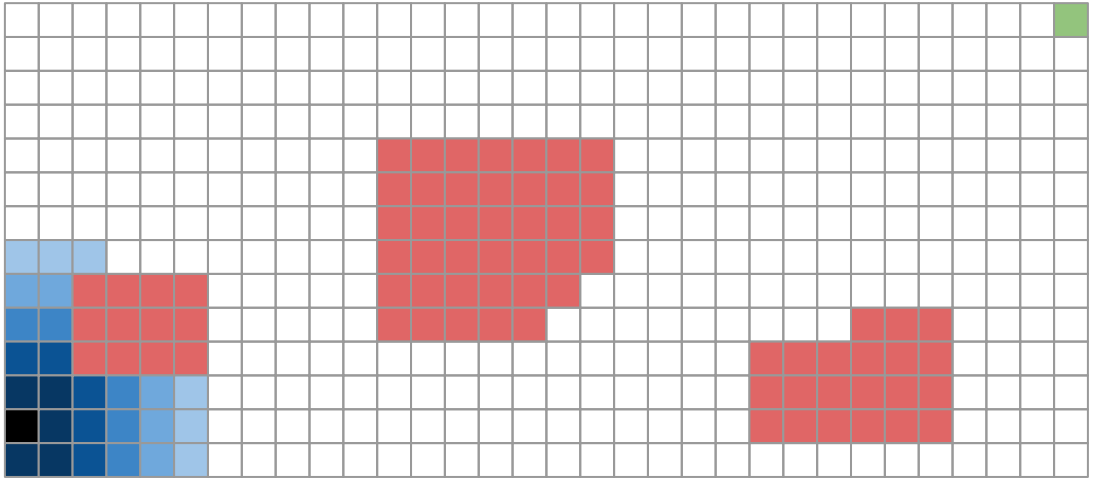
Idea: create a tree of possible paths



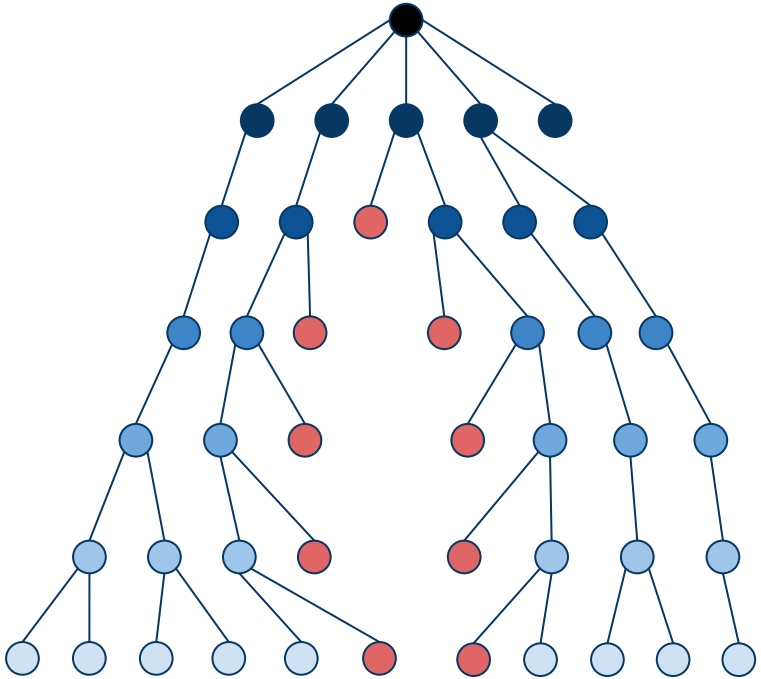


Idea: create a tree of possible paths

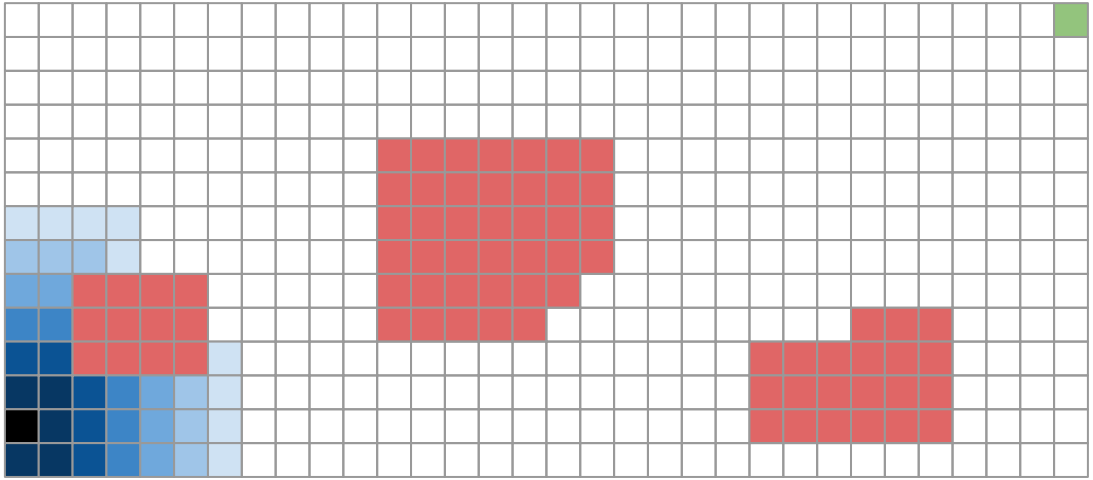




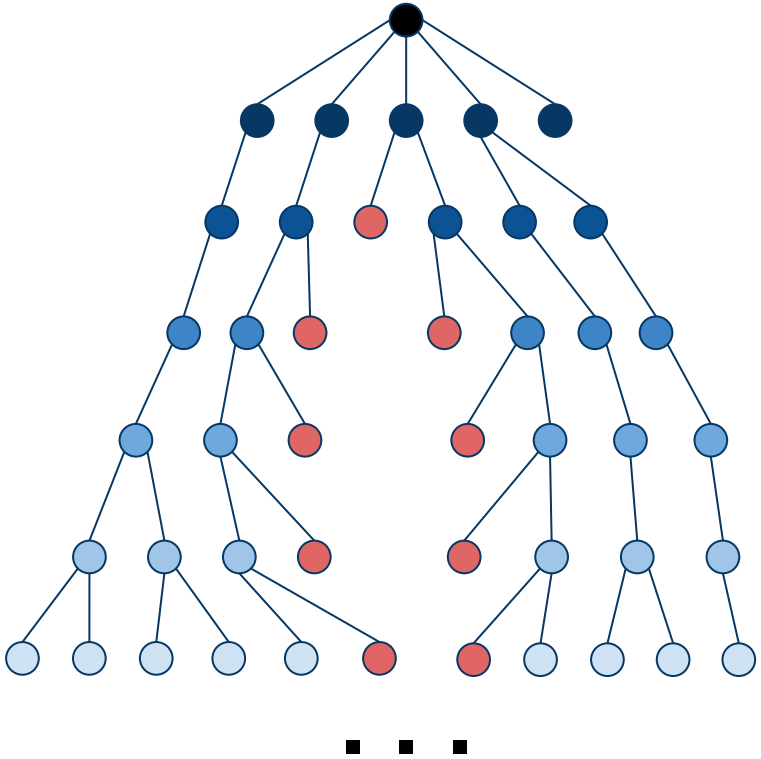
Idea: create a tree of possible paths

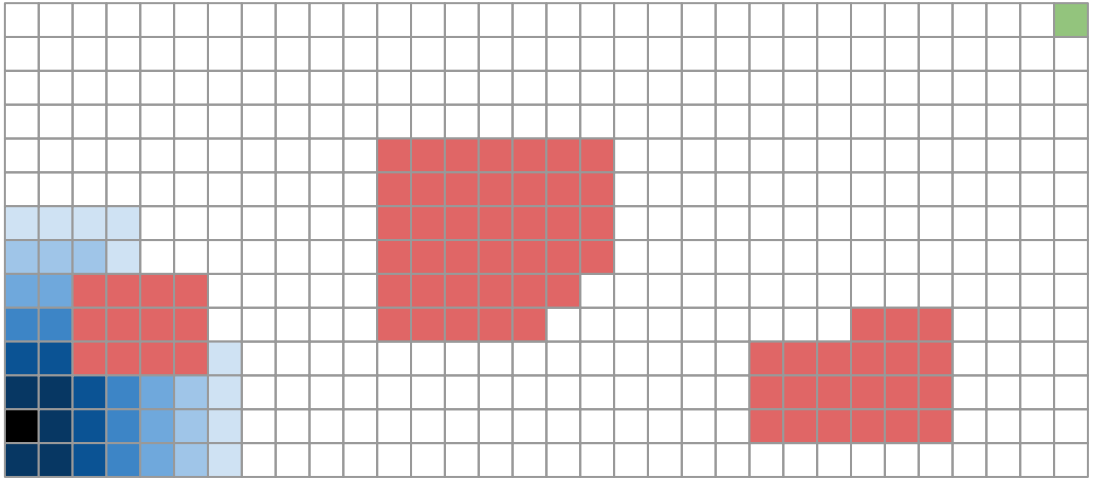


. . .

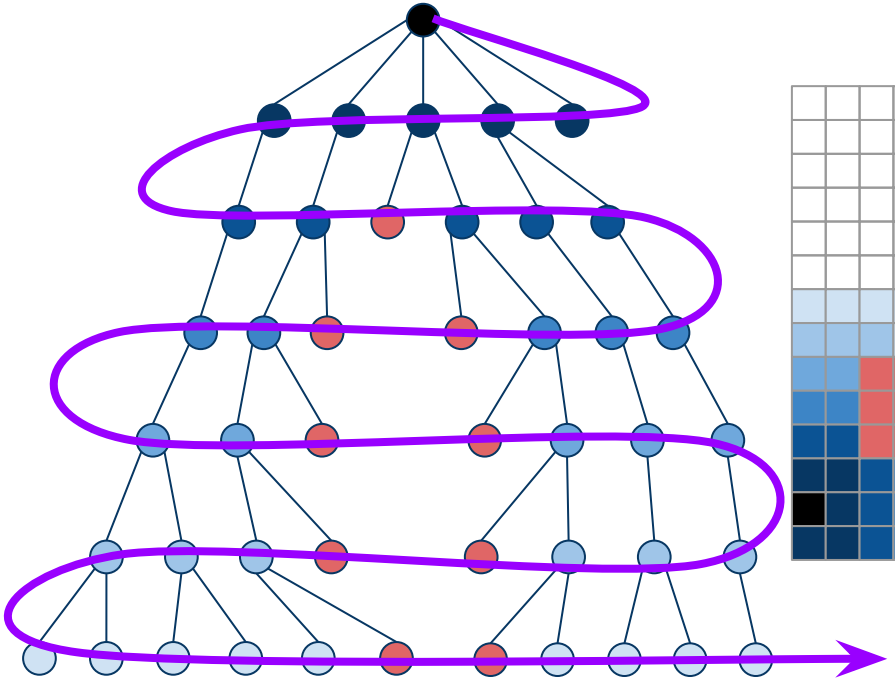


How can we search possible paths efficiently?

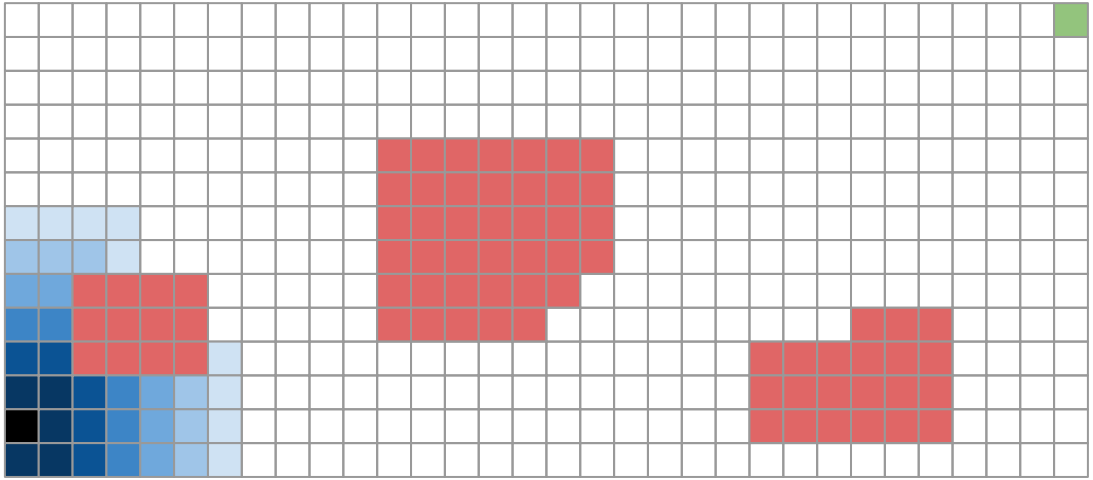




How can we search possible paths efficiently?

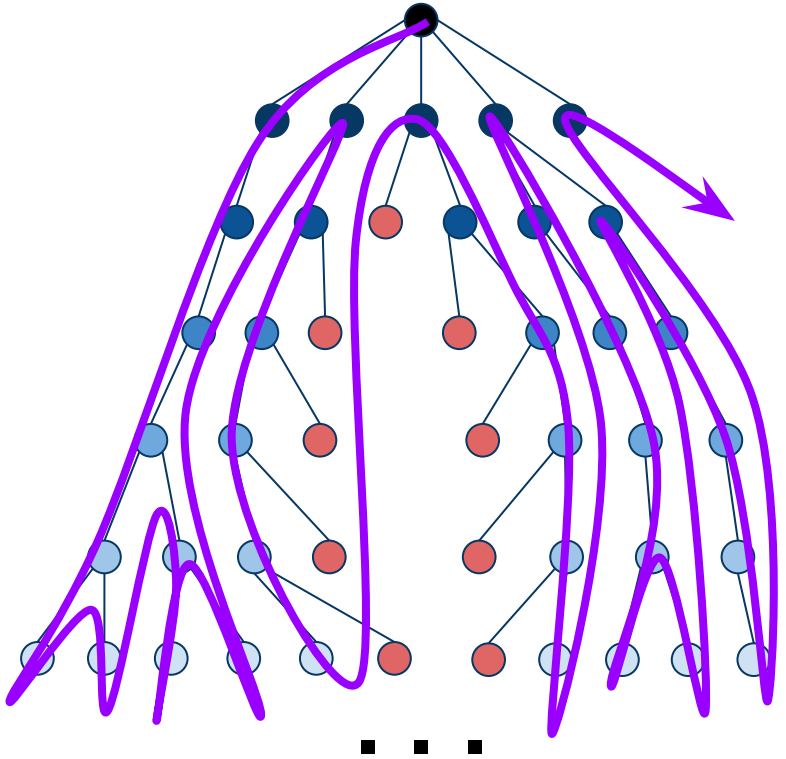


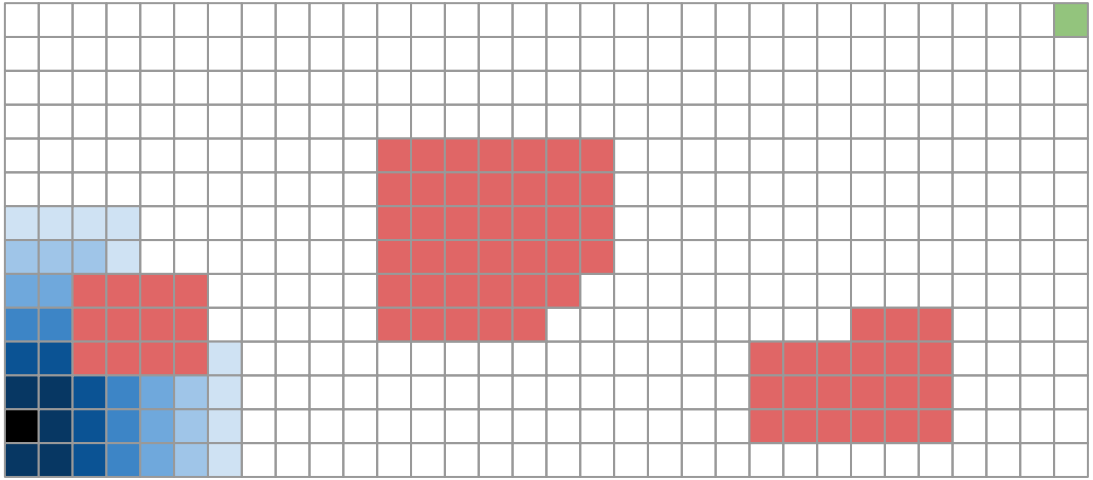
. . .



Breadth-First Search (BFS)

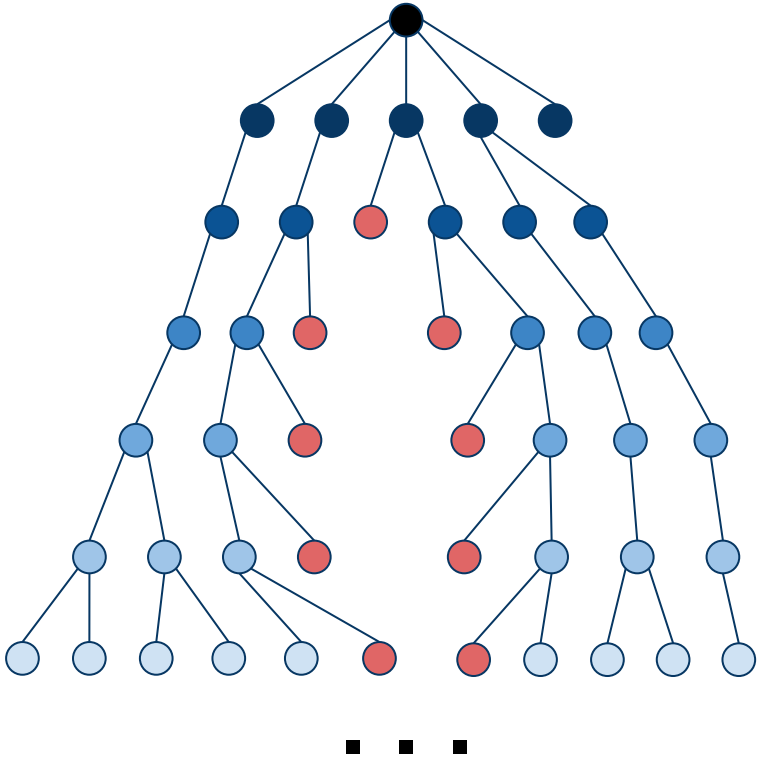
How can we search possible paths efficiently?

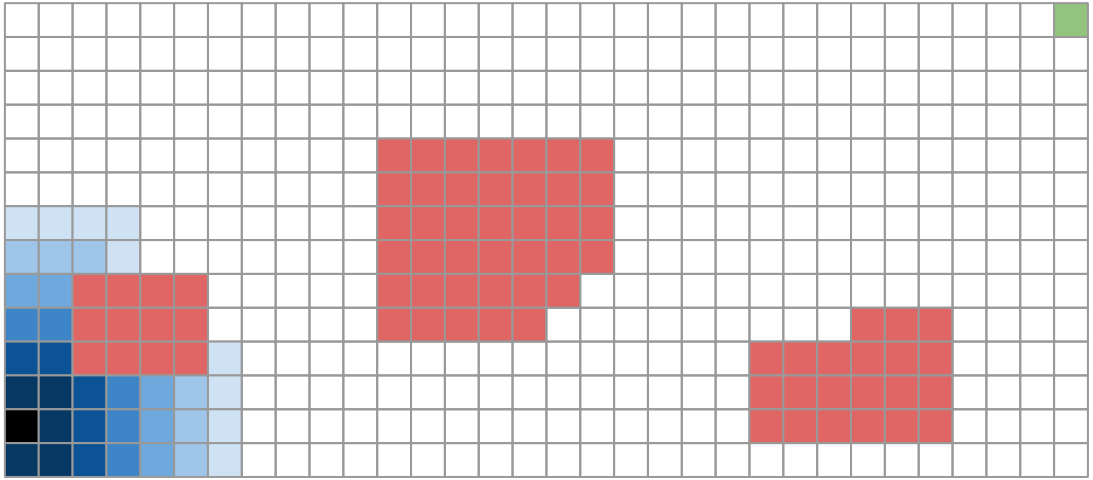




Depth-First Search (DFS)

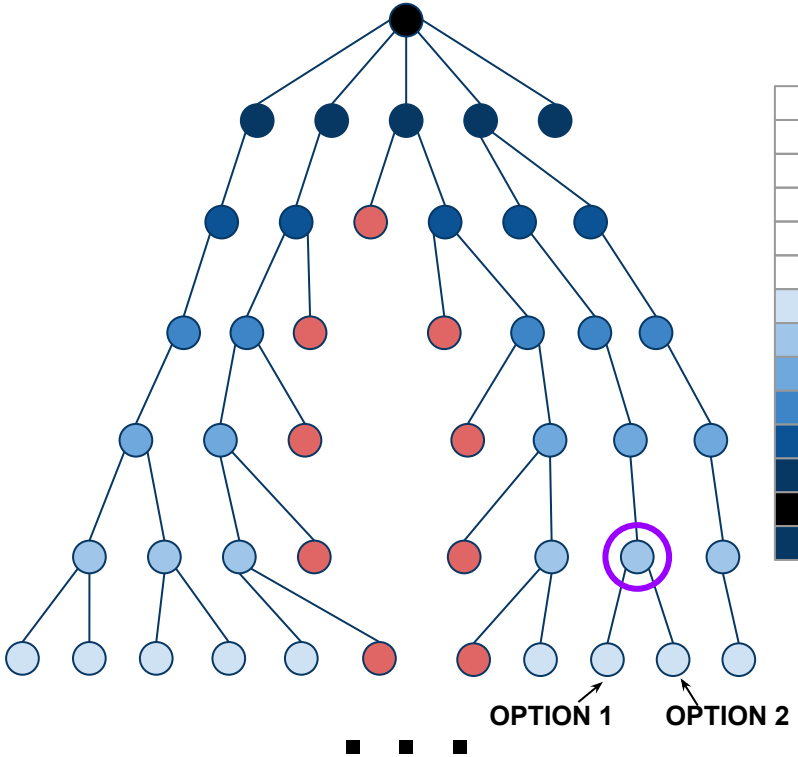
How can we search possible paths efficiently?

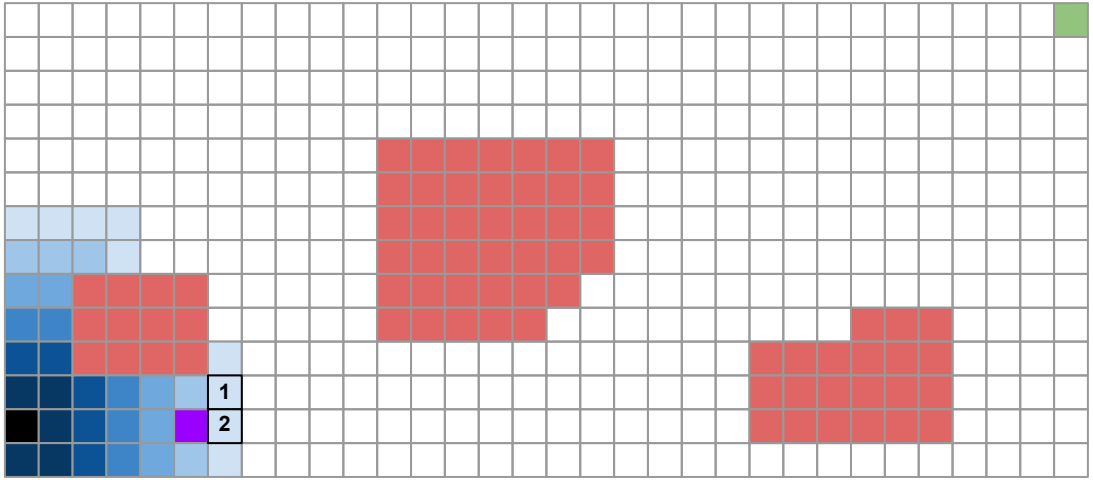




Can we do better?

Reframing the problem: where should we go next?





If robot is here, where should we search next?

Idea: search paths we think are good

Good path: a path with *low cost*

- fewest number of squares traversed, or
- fewest number of turns, or
- farthest from obstacles, or
- . . . , or
- some combination of the above

Idea: search paths we think are good

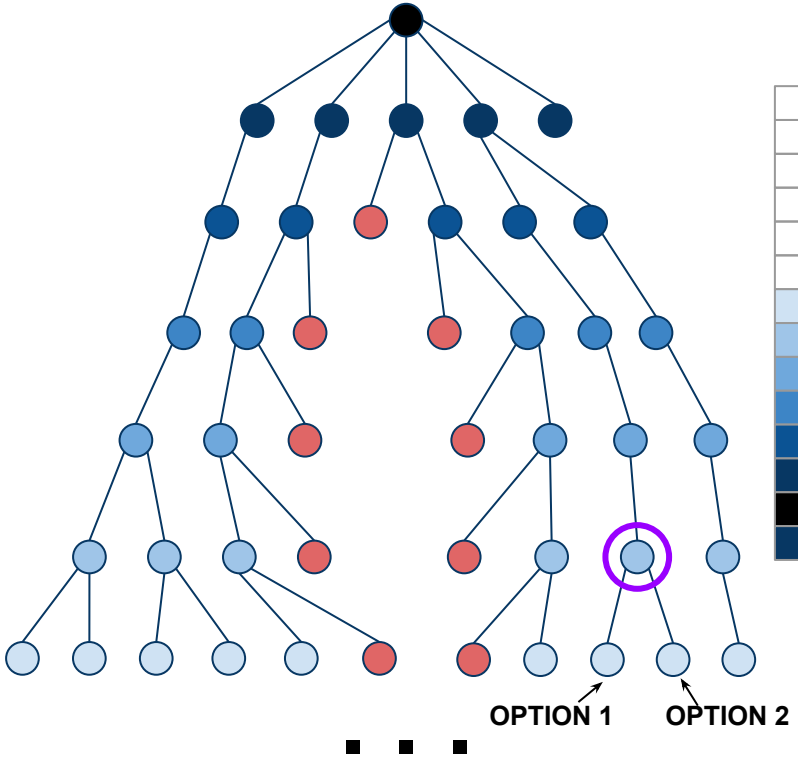
For our application, we will **assign each grid square a cost** and declare that **the cost of the path is the sum of the cost of the squares it traverses**.

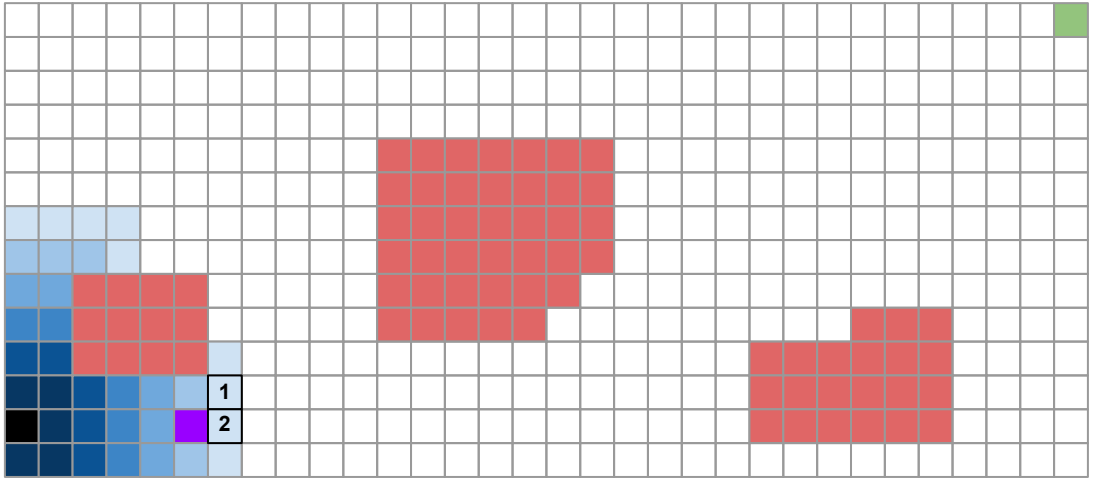
Idea: search paths we think are good

For our application, we will **assign each grid square a cost** and declare that **the cost of the path is the sum of the cost of the squares it traverses**.

Key idea: when deciding where to search next, choose the square with the lowest cost

Our example: choose lower-cost square (1 or 2)





If robot is here, where should we search next?

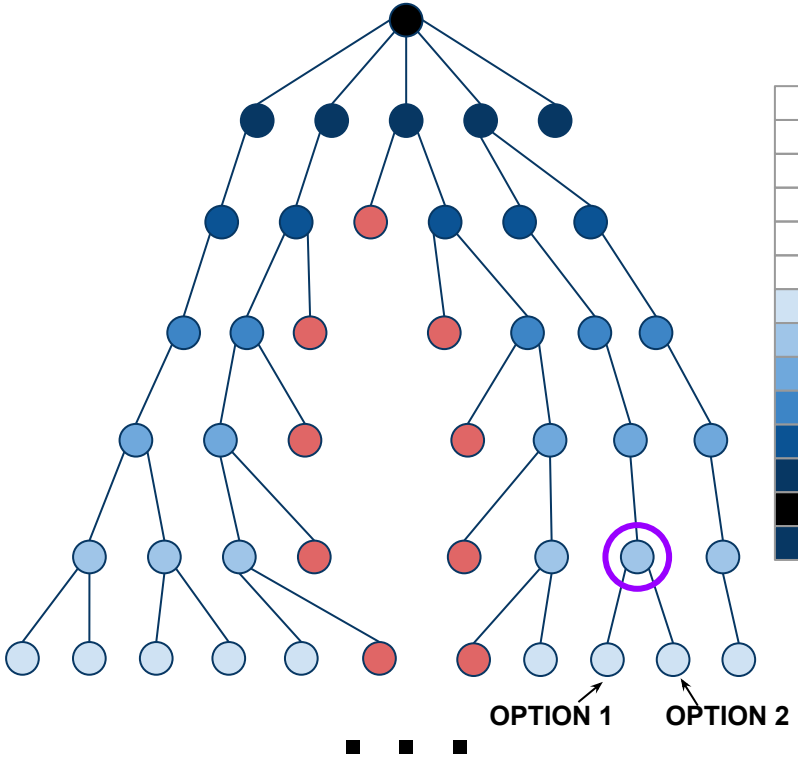
Assigning Cost Functions

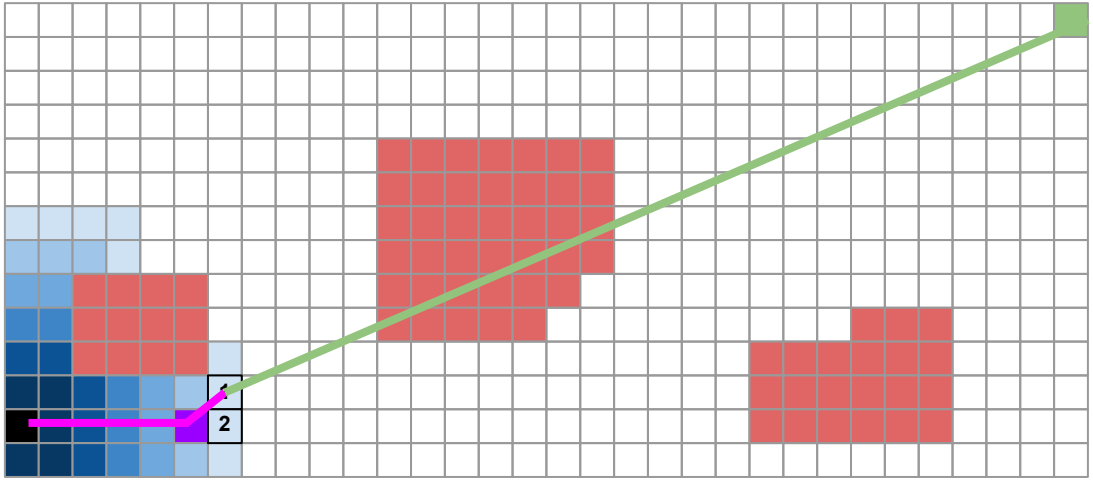
We will use the **cost function**

$$\underbrace{c(x)}_{\text{total cost}} = \underbrace{g(x)}_{\text{distance traveled to get there}} + \underbrace{h(x)}_{\text{estimated (**heuristic**) distance to goal}}$$

When we use this cost function, the algorithm is called **A***.

A*: option 1

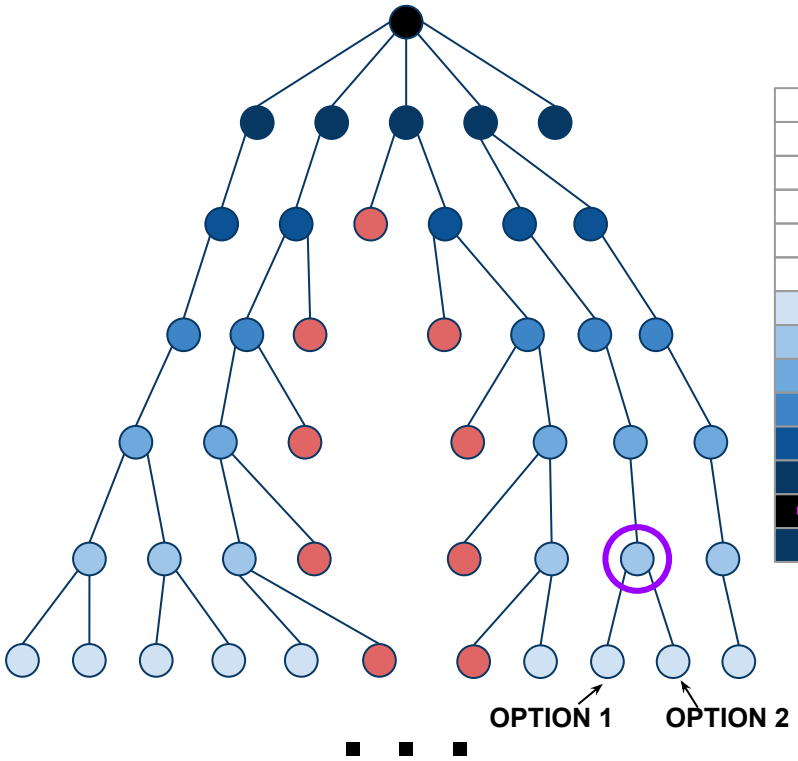


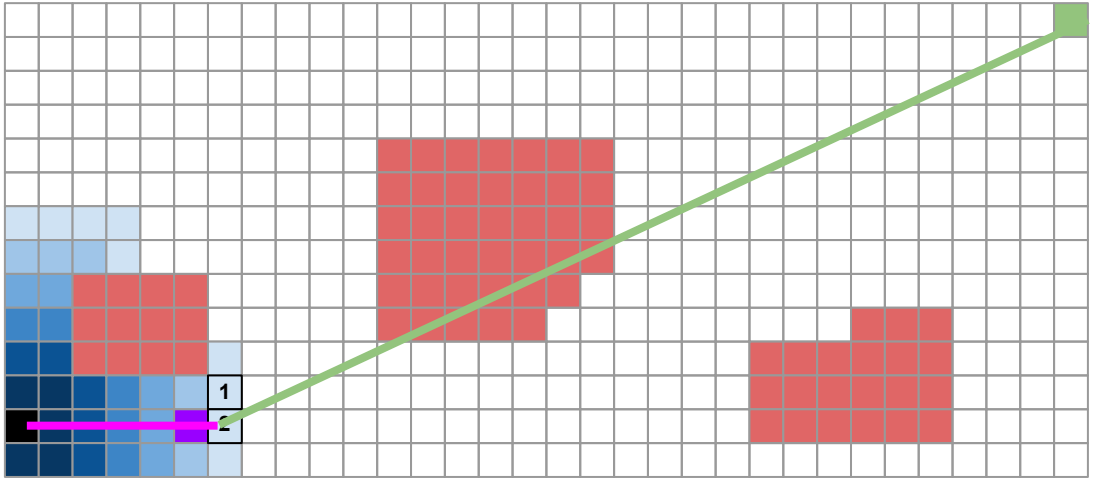


$$c(x) = g(x) + h(x)$$

total cost — distance traveled to get there — estimated (**heuristic**) distance to goal

A*: option 2

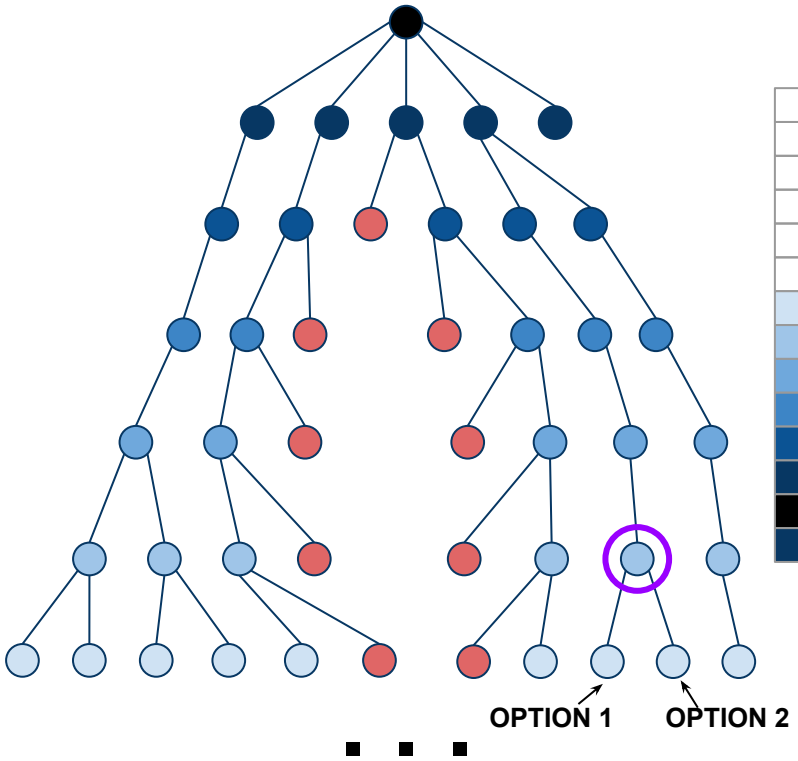


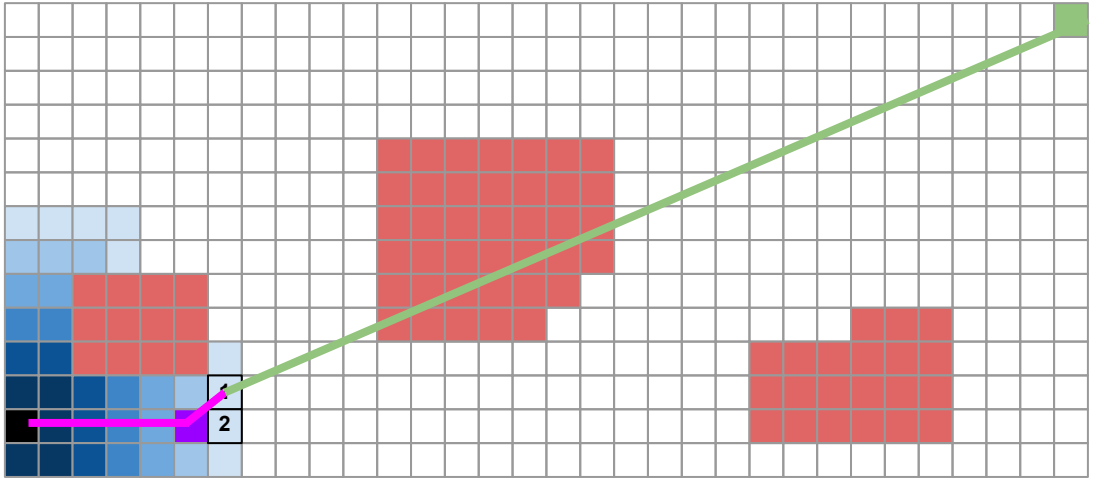


$$c(x) = g(x) + h(x)$$

total cost — distance traveled to get there — estimated (**heuristic**) distance to goal

A*: option 1 ✓

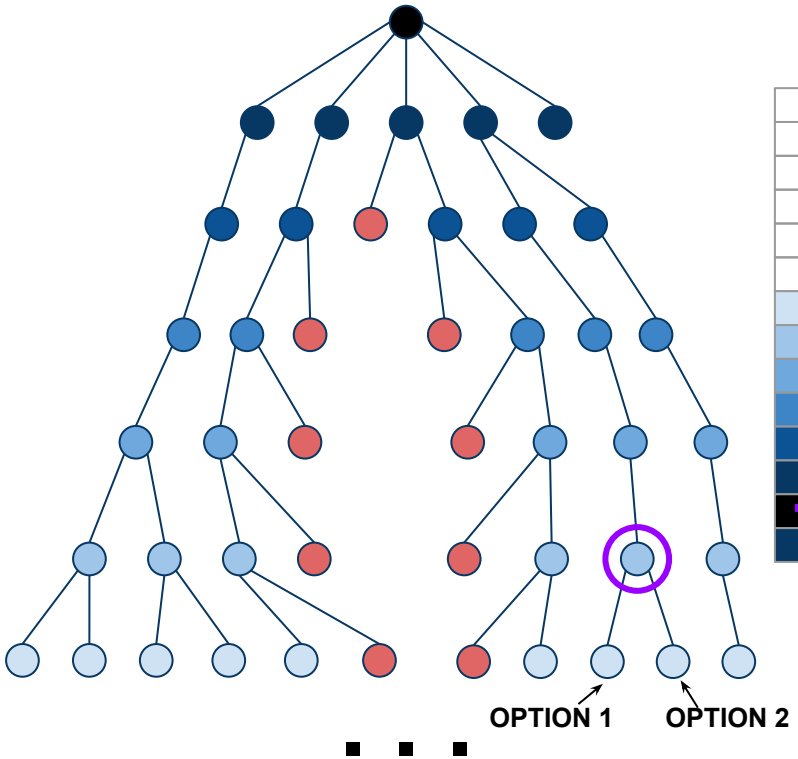


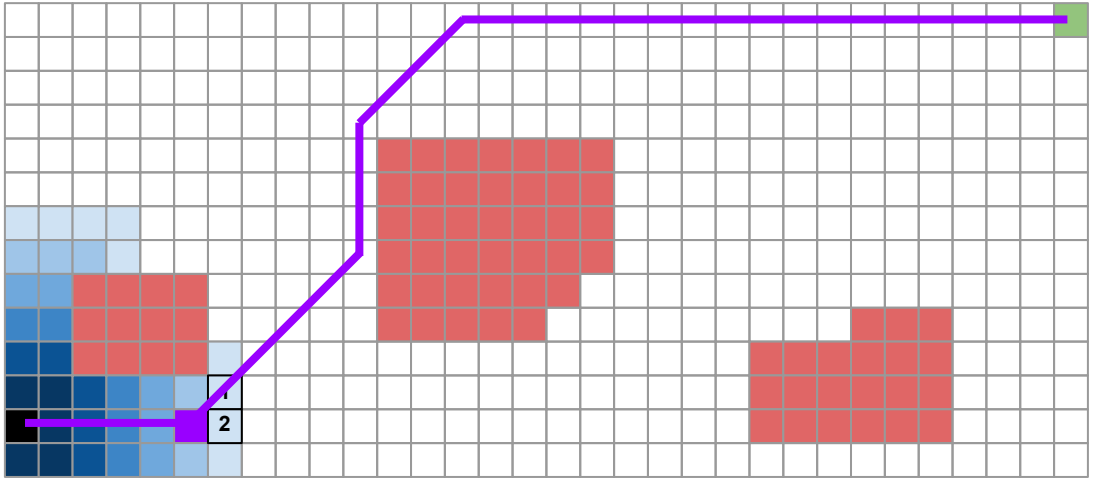


$$c(x) = g(x) + h(x)$$

total cost — distance traveled to get there — estimated (**heuristic**) distance to goal

A*: complete path



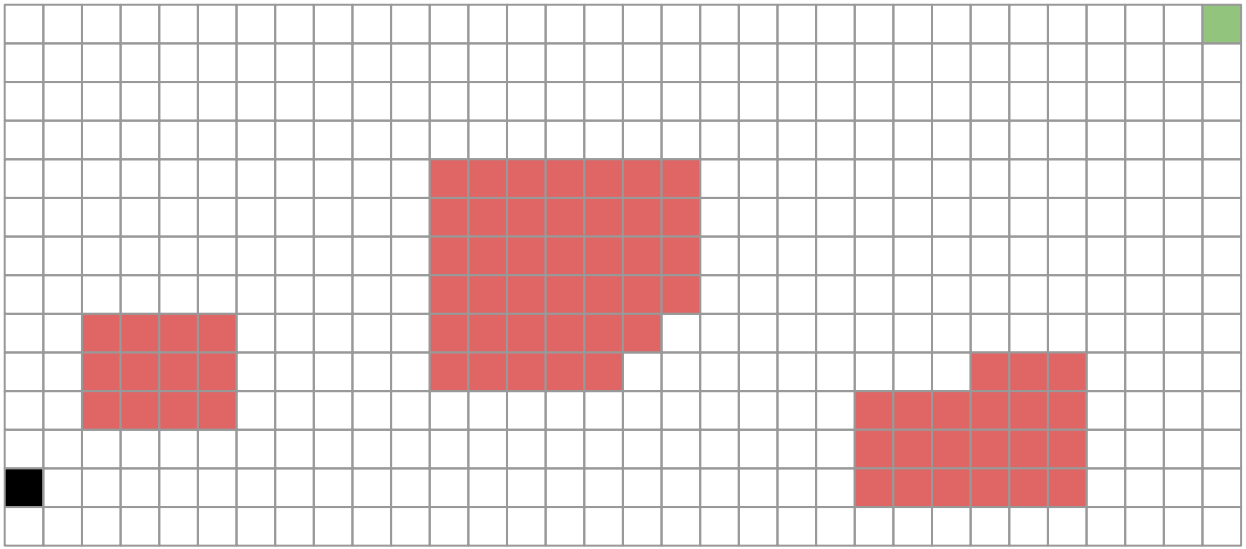


$$c(x) = g(x) + h(x)$$

total cost — distance traveled to get there — estimated (**heuristic**) distance to goal

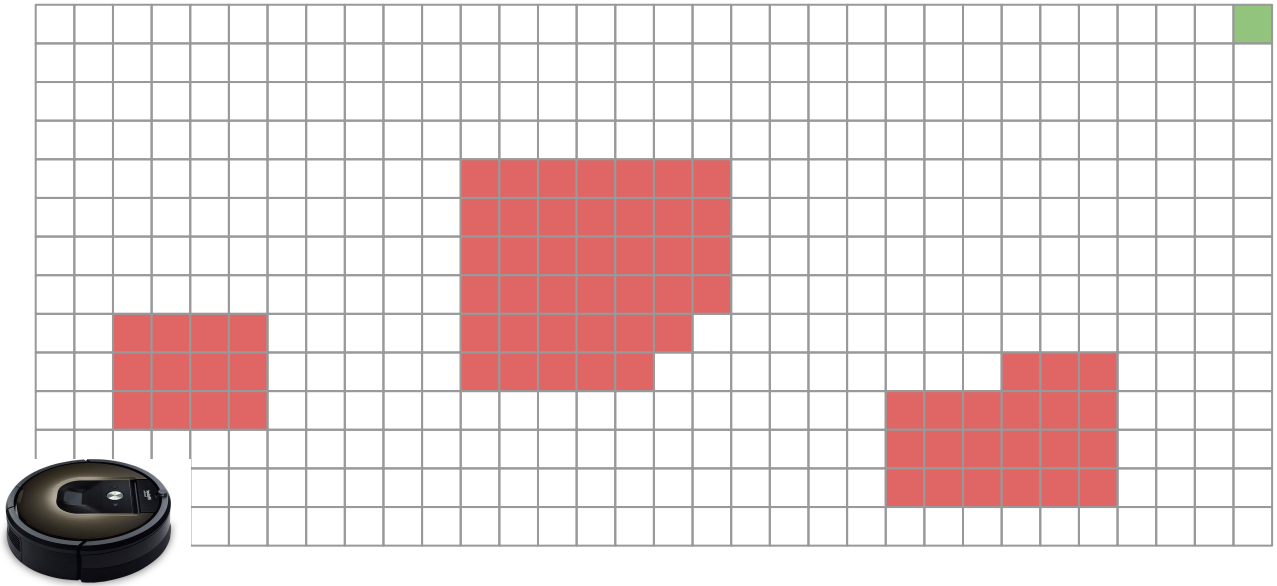
Limitations of A* *(on a rectilinear grid)*

Problem 1: Your robot isn't actually a point.



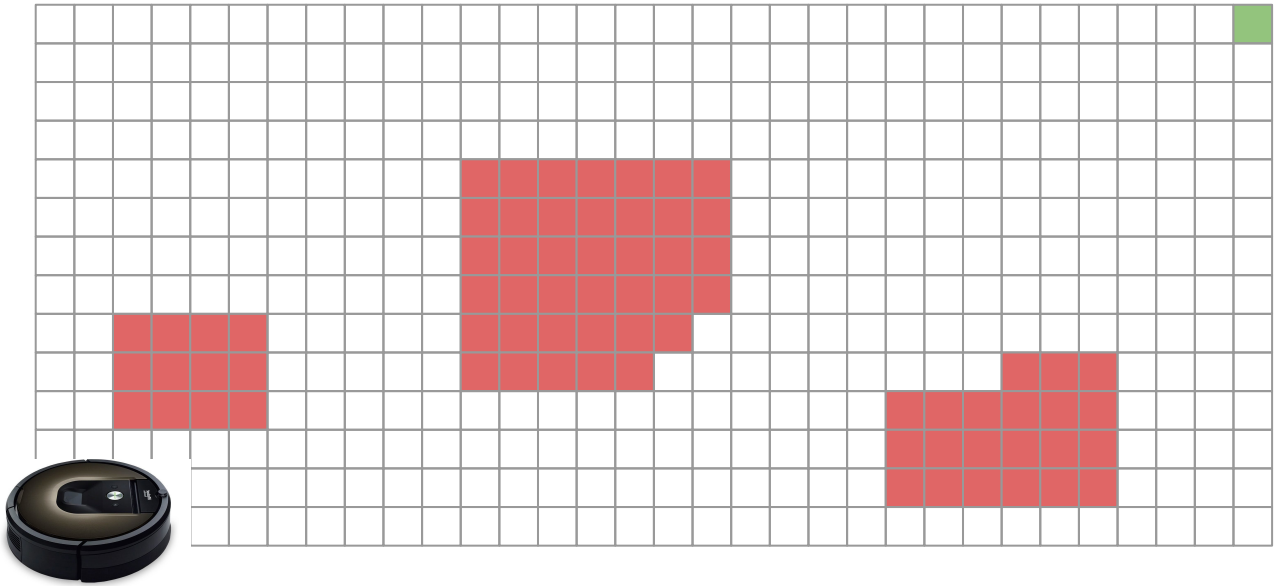
Limitations of A* *(on a rectilinear grid)*

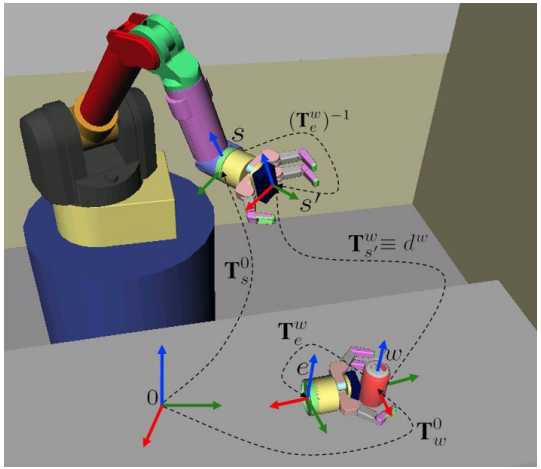
Problem 1: Your robot isn't actually a point.



Limitations of A* *(on a rectilinear grid)*

Problem 1: Your robot isn't actually a point.

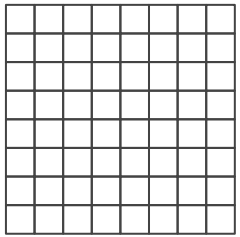


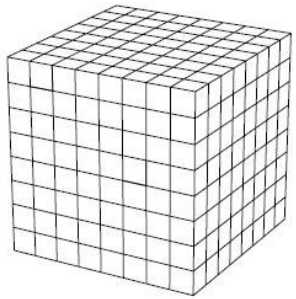


Limitations of A* *(on a rectilinear grid)*

Problem 1: Your robot isn't actually a point.

Problem 2: the curse of dimensionality





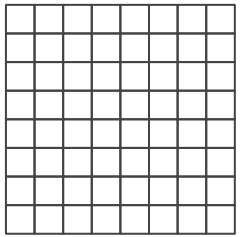
$$8^2 = 64$$

$$8^3 = 512$$

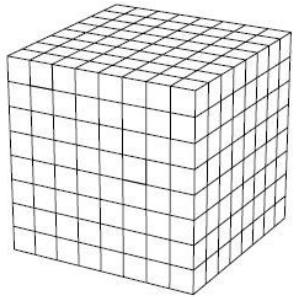
Limitations of A* *(on a rectilinear grid)*

Problem 1: Your robot isn't actually a point.

Problem 2: the curse of dimensionality

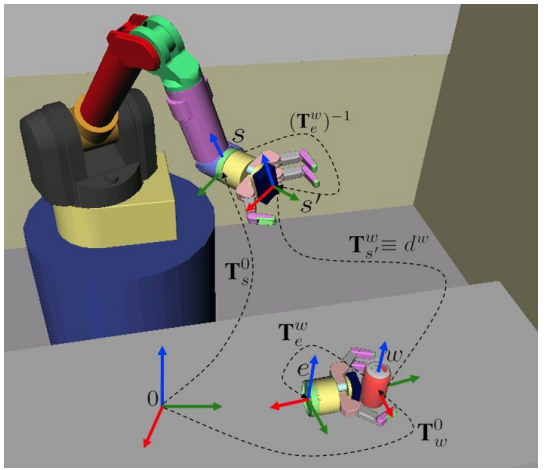


$$8^2 = 64$$



$$8^3 = 512$$

. . .



$$8^6 = 262144$$

Limitations of A* *(on a rectilinear grid)*

Problem 1: Your robot isn't actually a point.

Problem 2: the curse of dimensionality

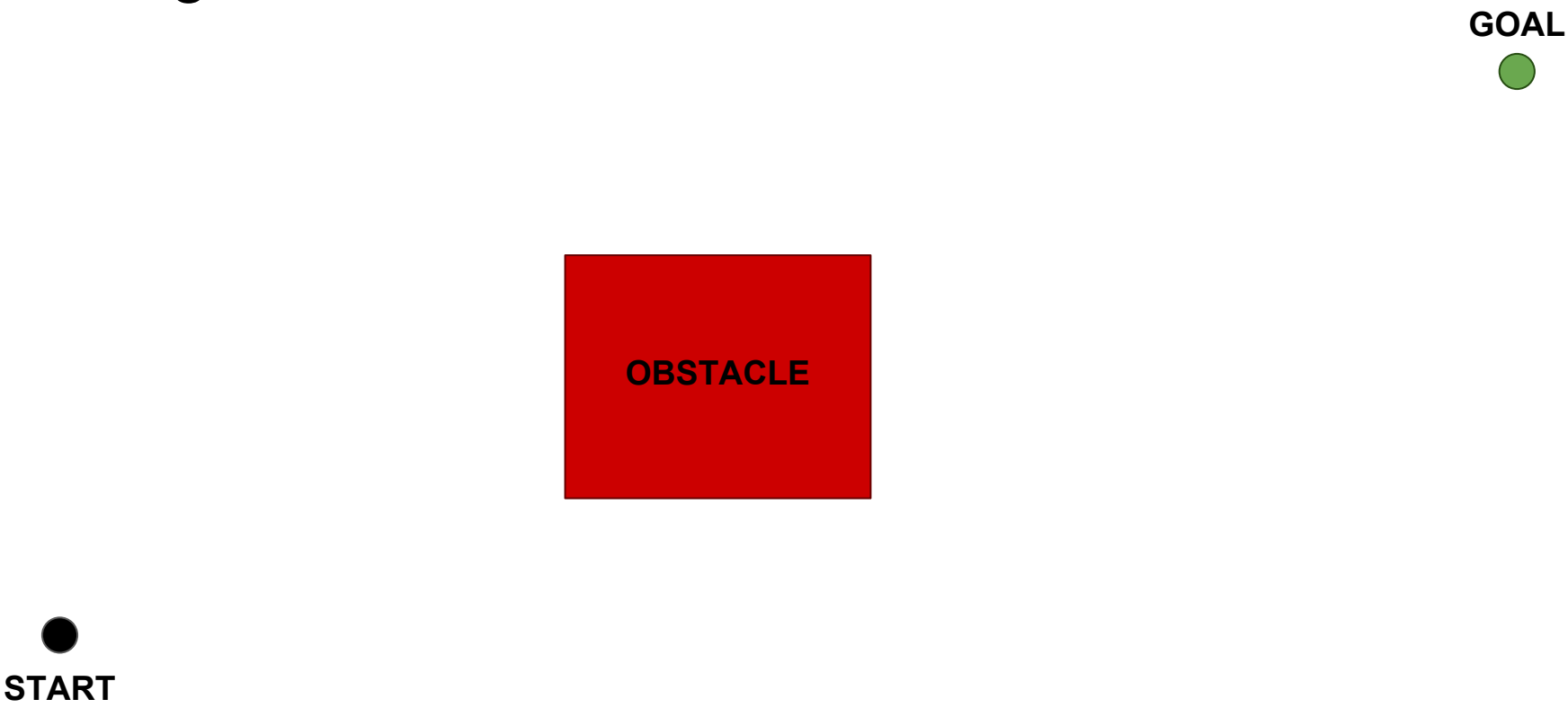
Limitations of A* *(on a rectilinear grid)*

Problem 1: Your robot isn't actually a point. → configuration space

Problem 2: the curse of dimensionality → **1**: sampling-based planning
2: optimization-based planning

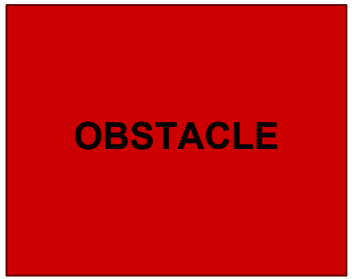
configuration space

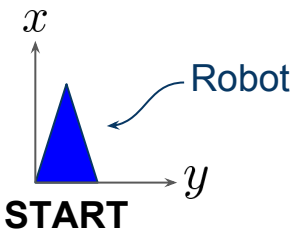
Making the Robot “Robot-Sized”



Making the Robot “Robot-Sized”

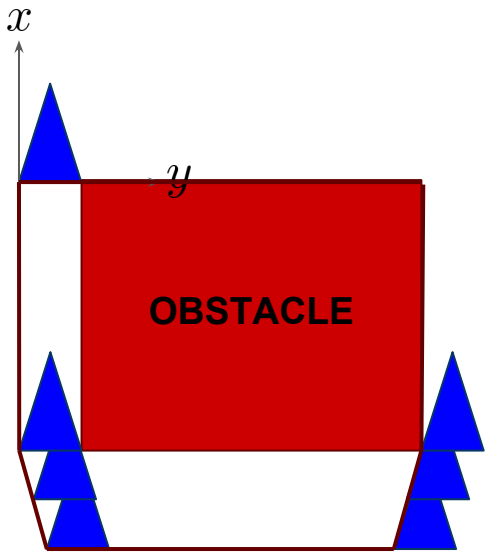
GOAL





Making the Robot “Robot-Sized”

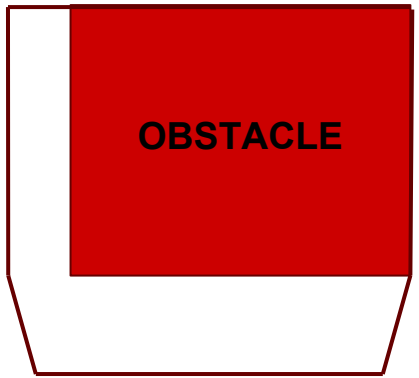
GOAL



START

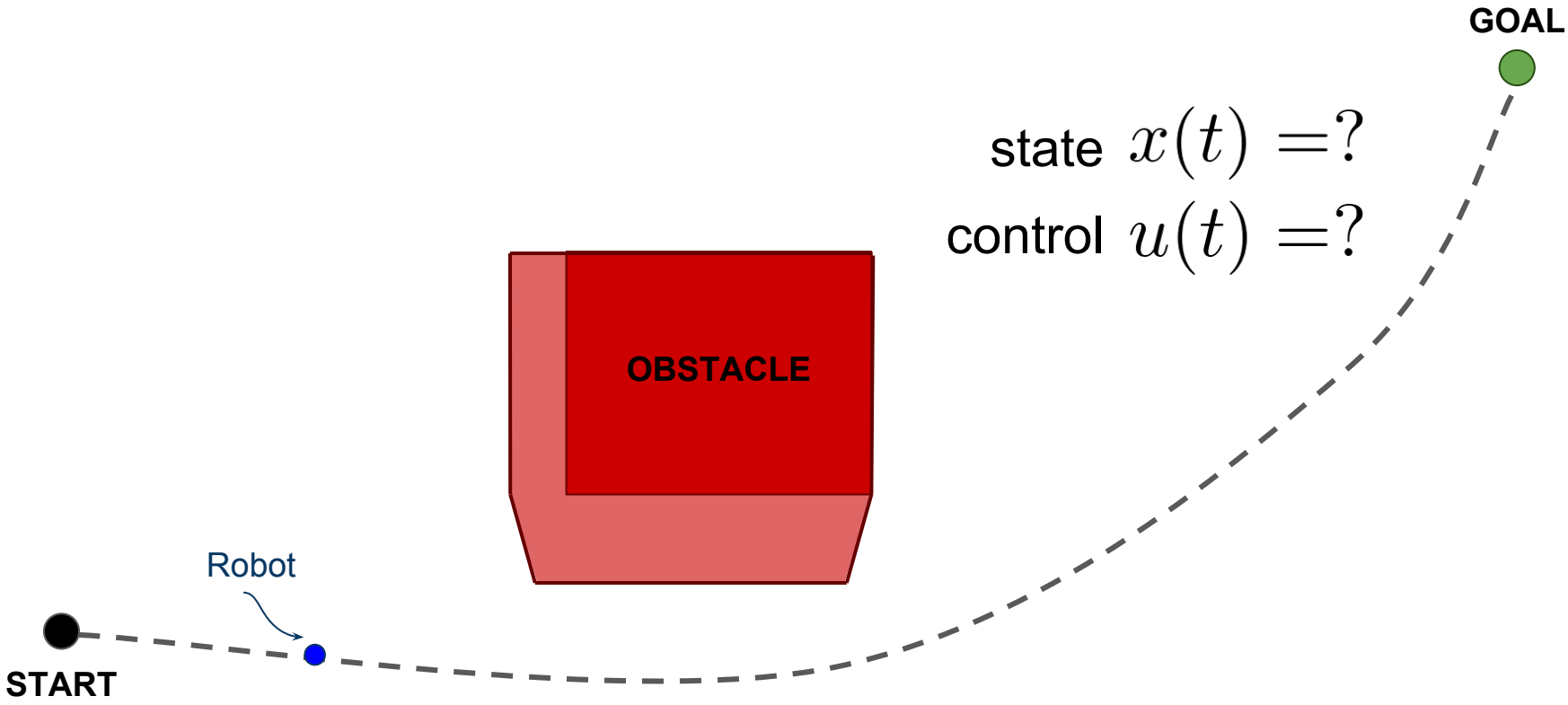
Making the Robot “Robot-Sized”

GOAL

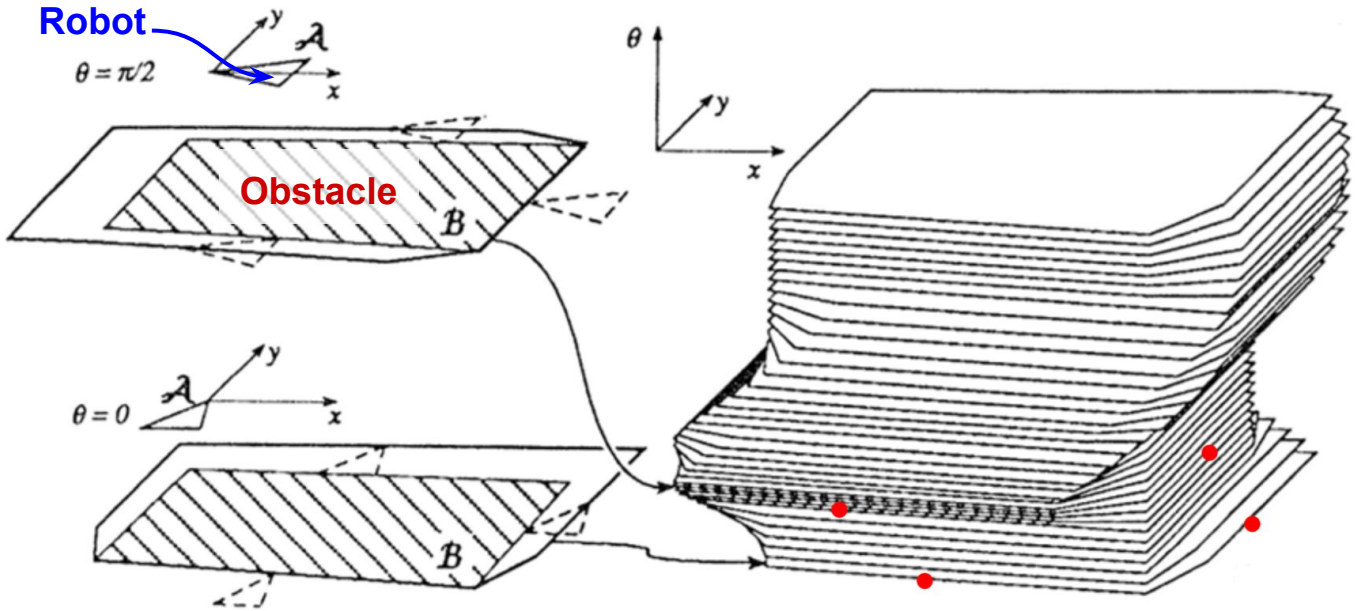


START

Configuration Space

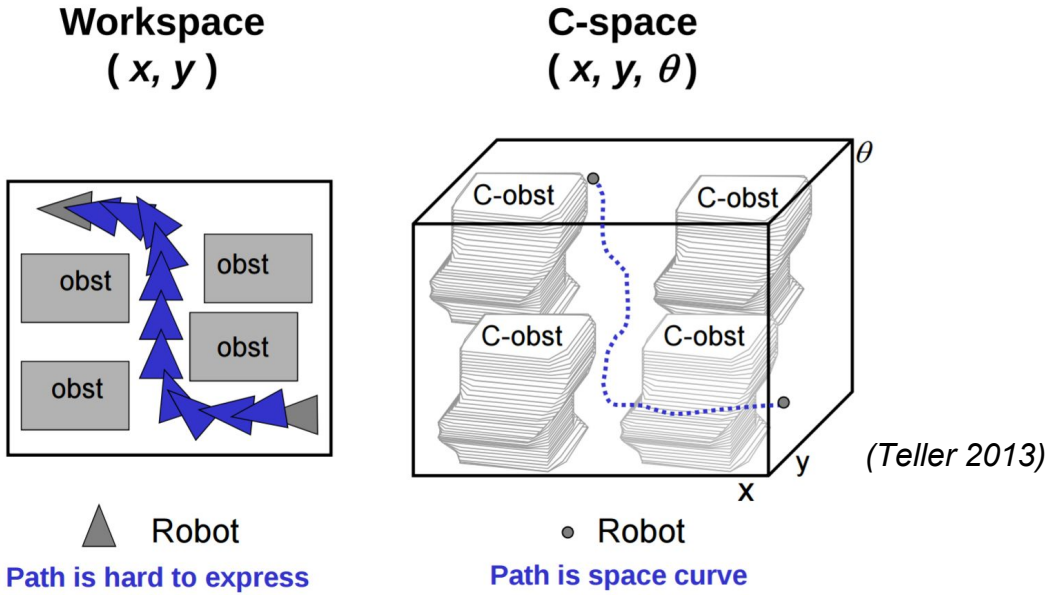


What if the robot can rotate?



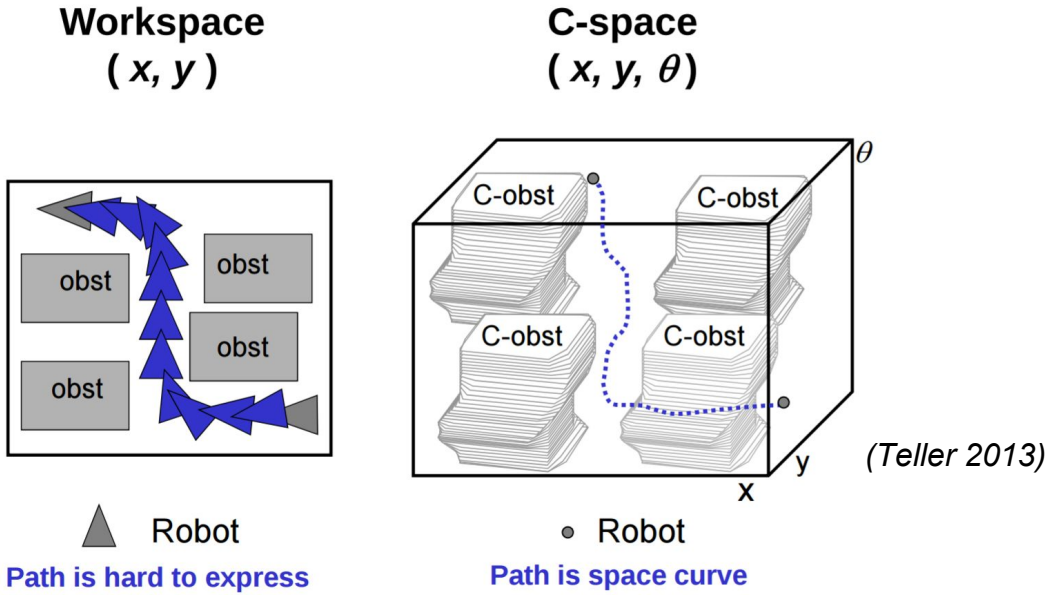
(Latombe 1991)

Multi-Obstacle C-Space



(Teller 2013)

Multi-Obstacle C-Space



(Teller 2013)

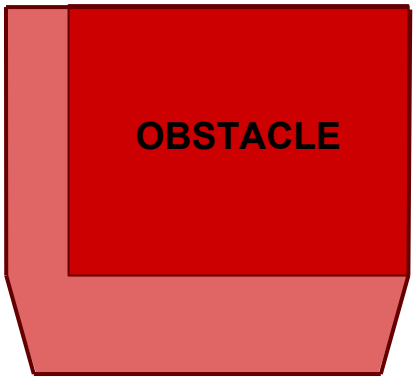
. . . but we still need to tackle the curse of dimensionality.

Solution 1:

Sampling-based planners

Idea: instead of gridding, sample randomly!

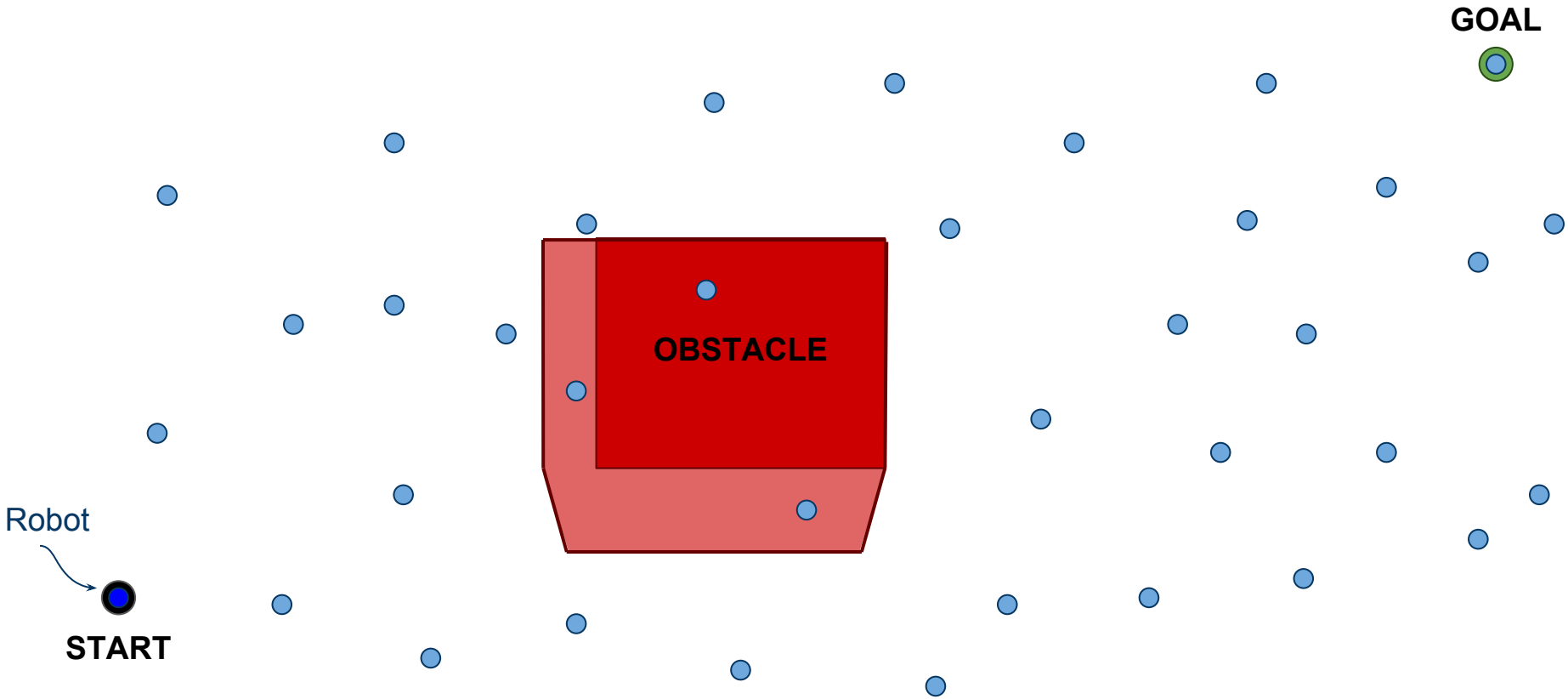
GOAL



Robot

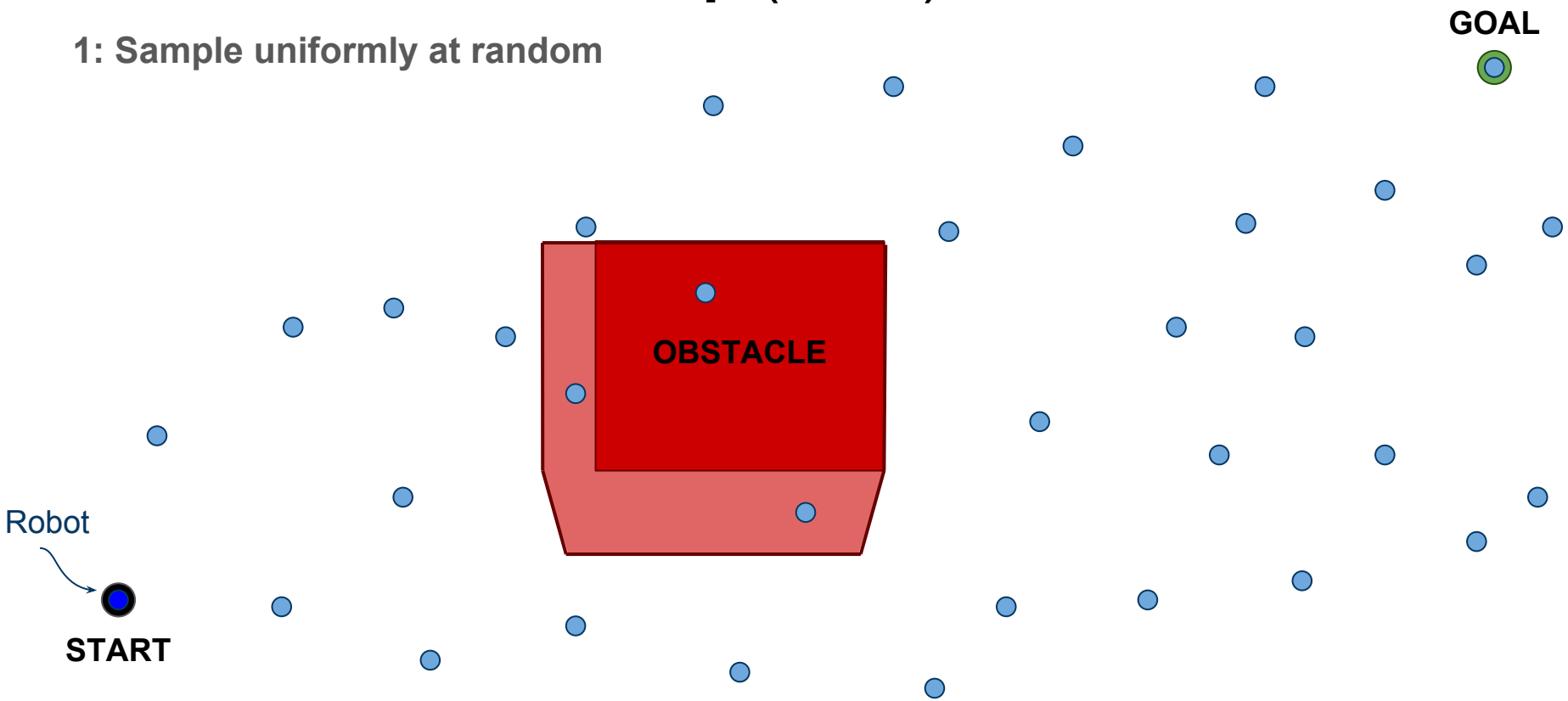
START

Idea: instead of gridding, sample randomly!



Probabilistic Roadmap (PRM)

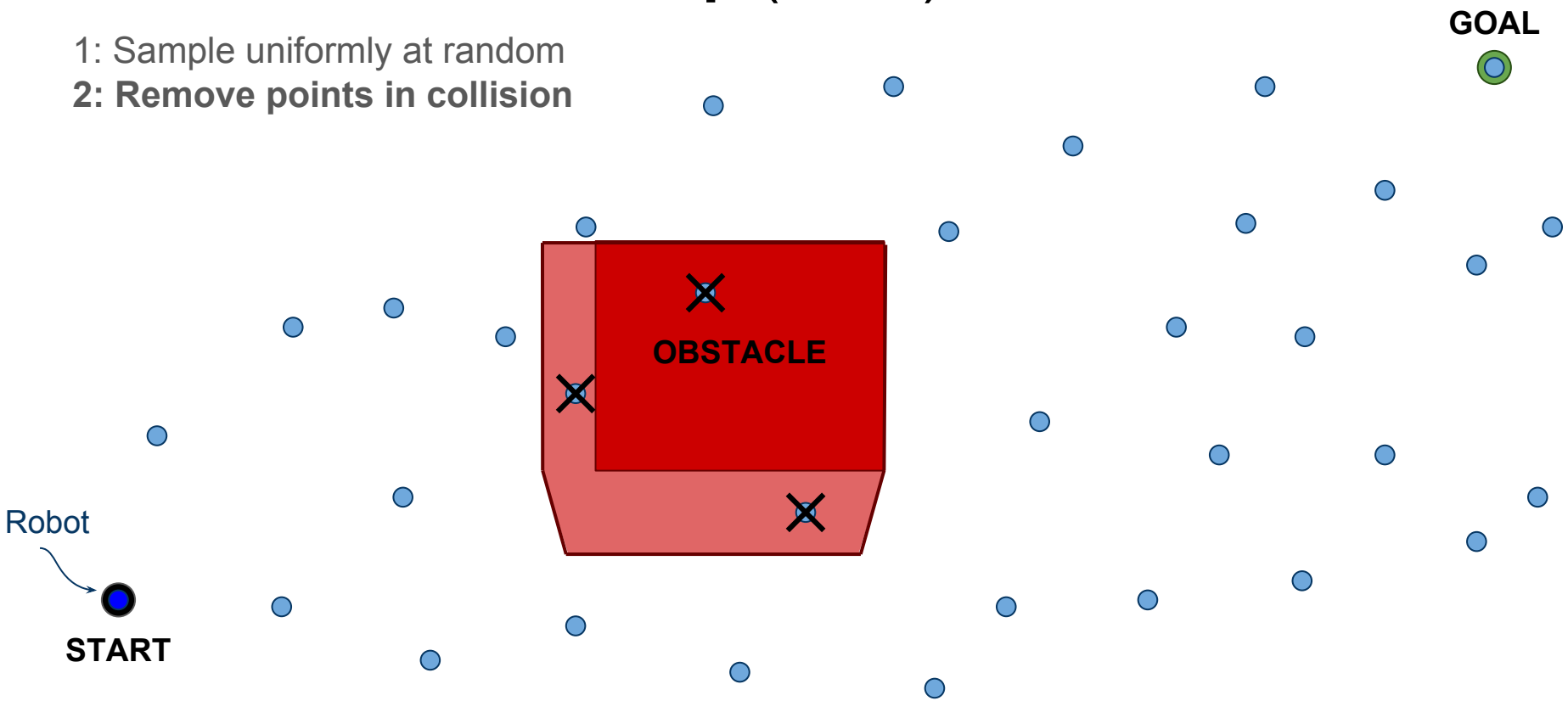
1: Sample uniformly at random



Probabilistic Roadmap (PRM)

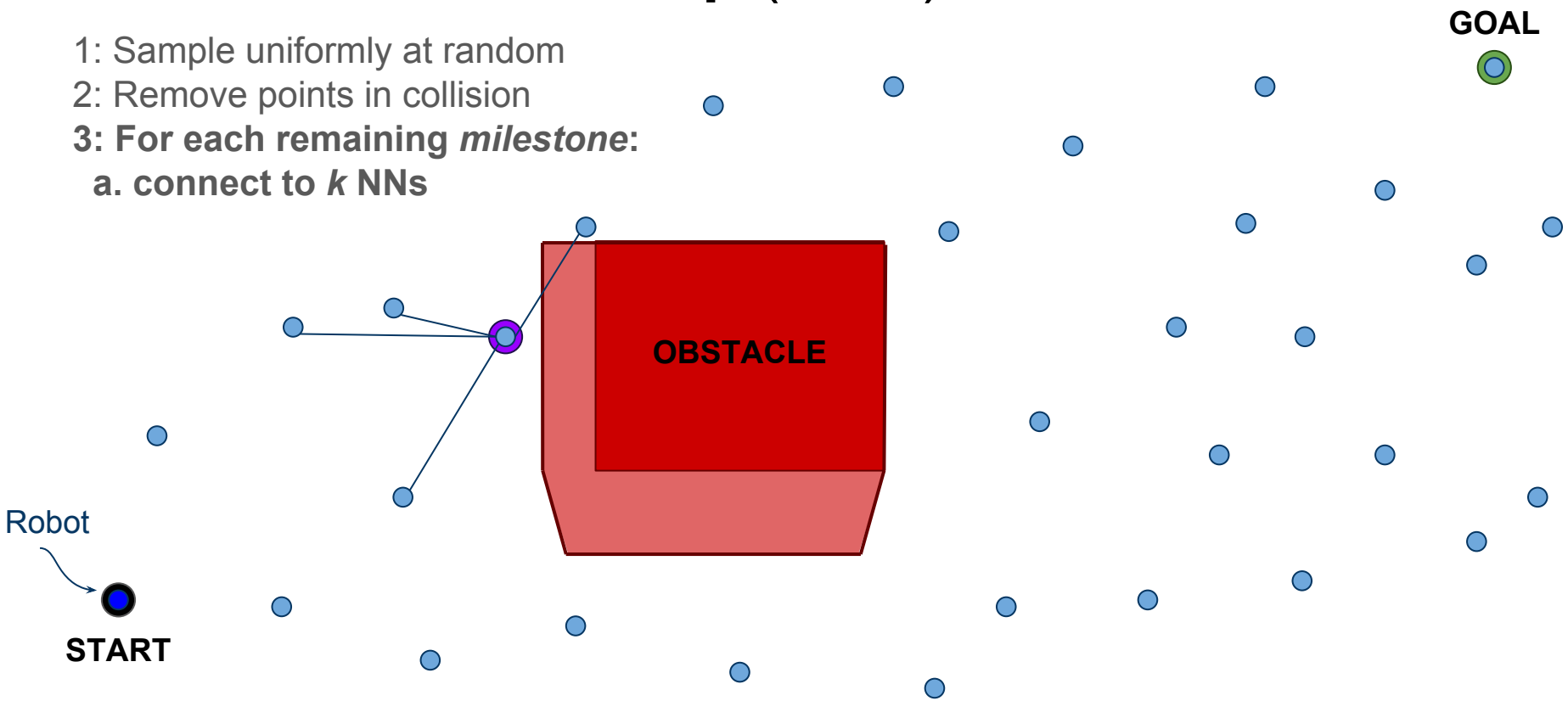
1: Sample uniformly at random

2: Remove points in collision



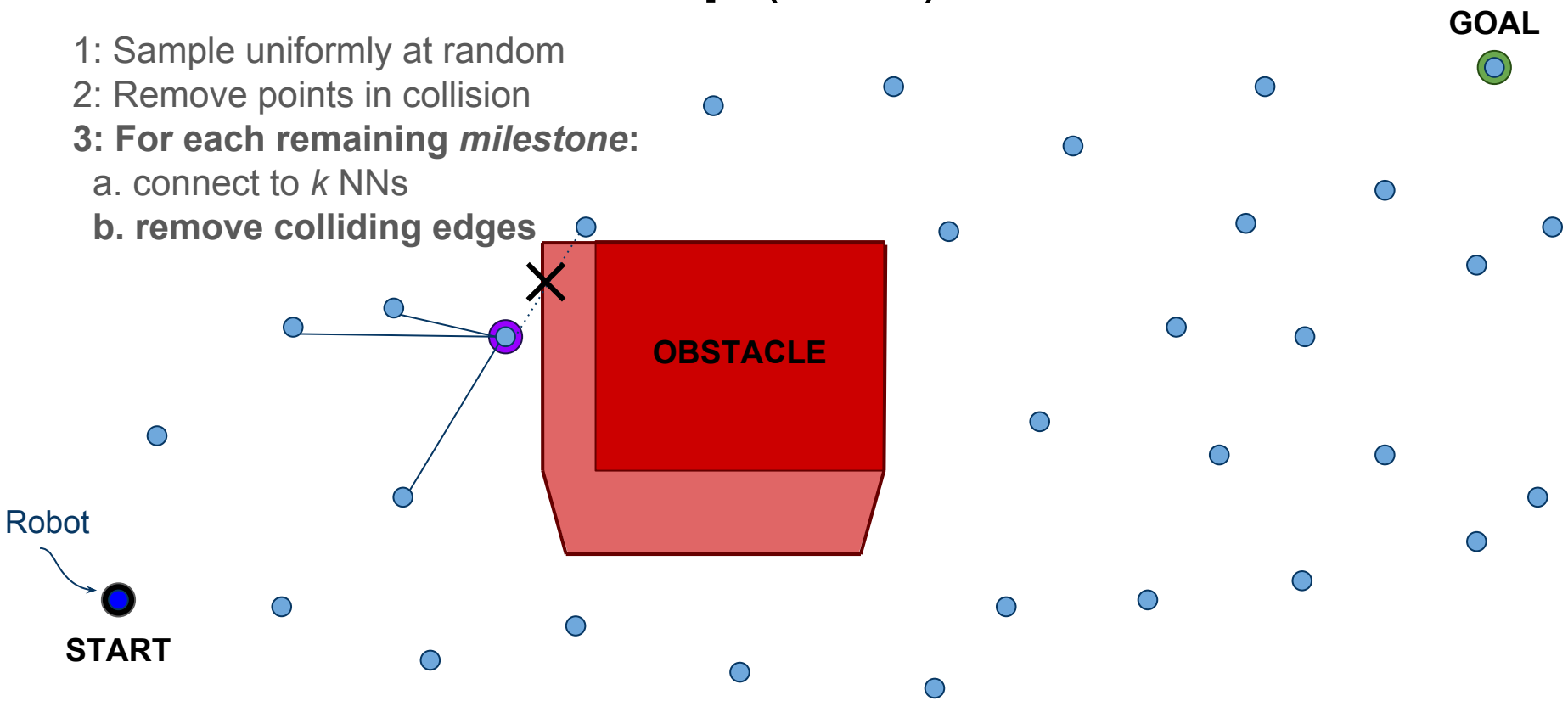
Probabilistic Roadmap (PRM)

1: Sample uniformly at random
2: Remove points in collision
3: For each remaining *milestone*:
a. connect to *k* NNs



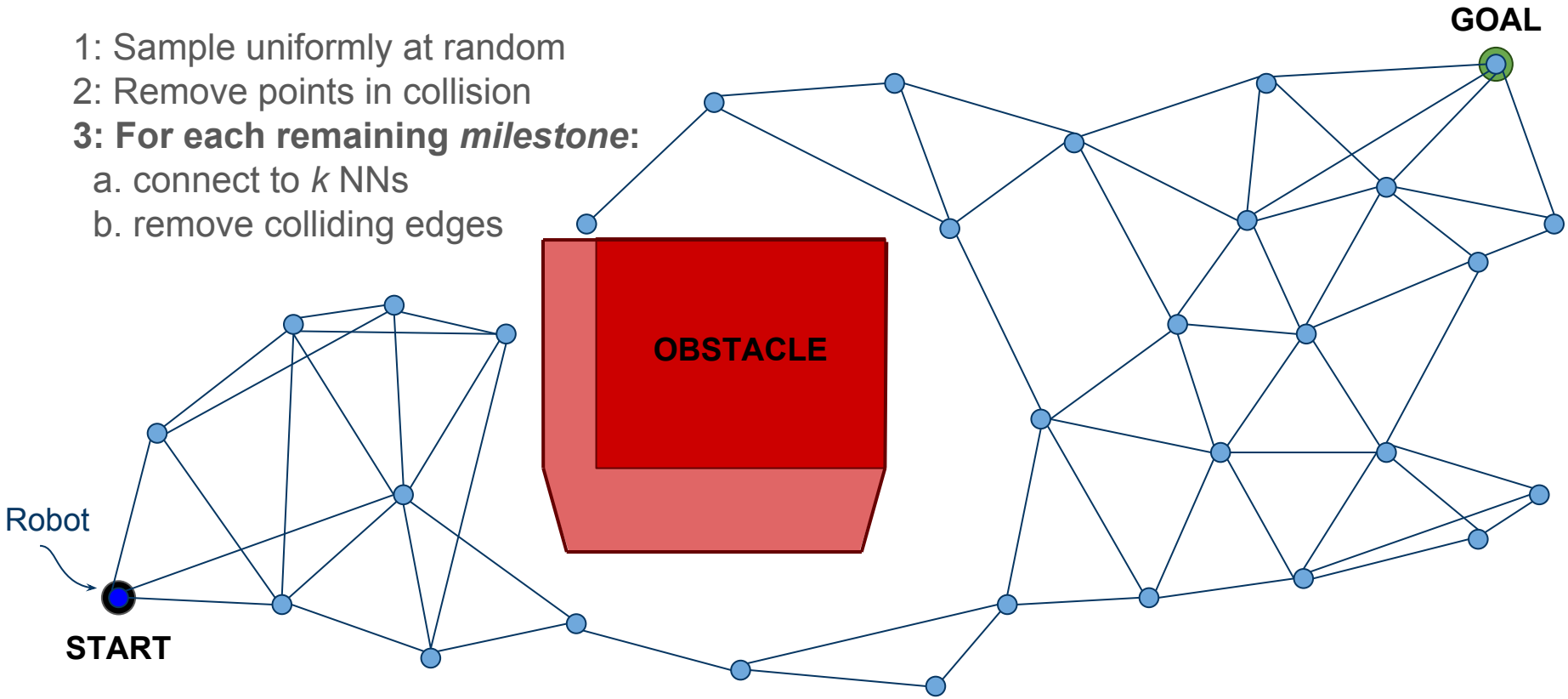
Probabilistic Roadmap (PRM)

1: Sample uniformly at random
2: Remove points in collision
3: For each remaining *milestone*:
  a. connect to *k* NNs
  b. remove colliding edges



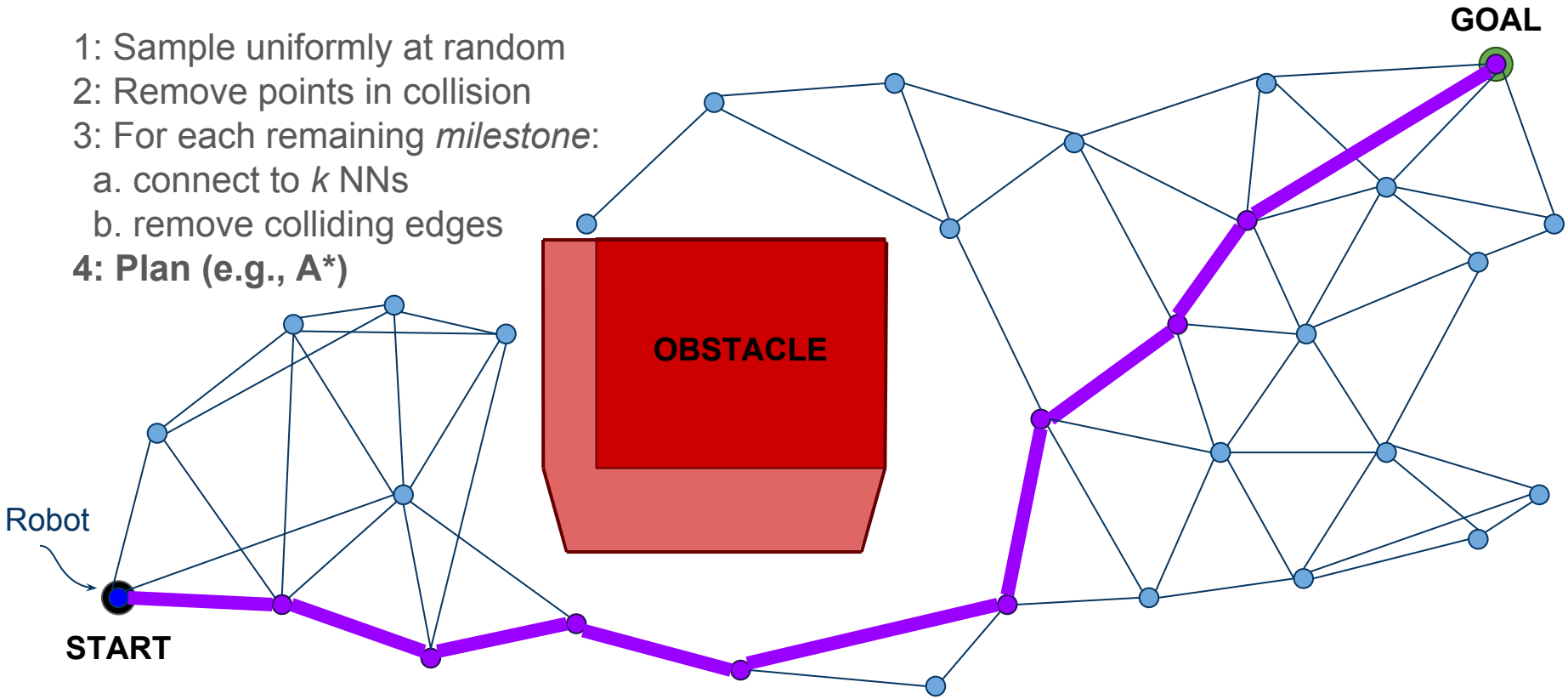
Probabilistic Roadmap (PRM)

1: Sample uniformly at random
2: Remove points in collision
3: For each remaining *milestone*:
  a. connect to *k* NNs
  b. remove colliding edges



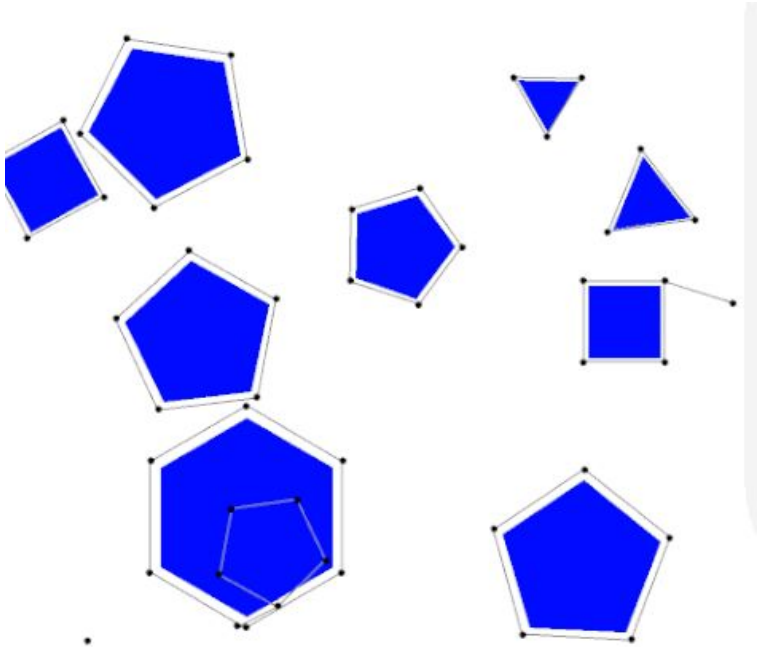
Probabilistic Roadmap (PRM)

1: Sample uniformly at random
2: Remove points in collision
3: For each remaining *milestone*:
  a. connect to *k* NNs
  b. remove colliding edges
4: Plan (e.g., A*)



Probabilistic Roadmap (PRM)

+ Probabilistically complete
+ Multiple start states / queries



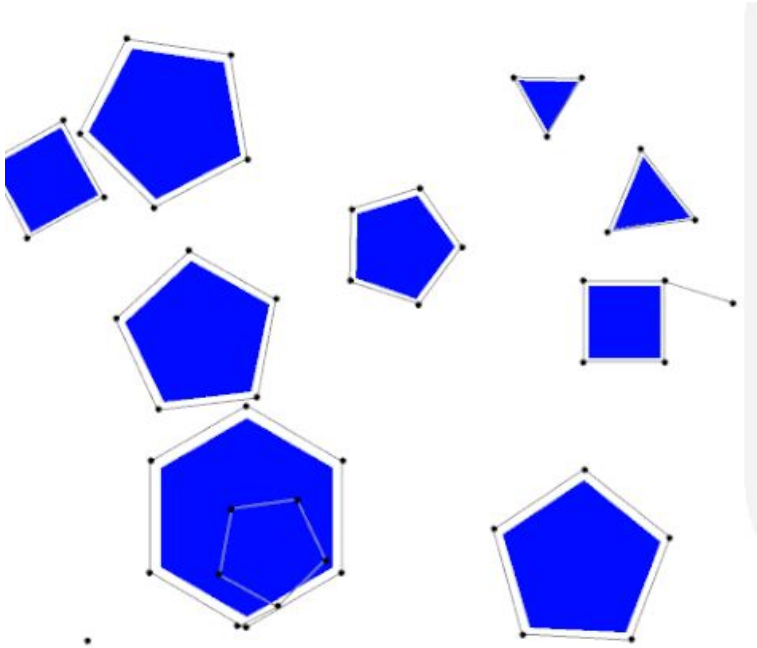
(Eric O. Scott for Wikipedia)

Probabilistic Roadmap (PRM)

+ Probabilistically complete

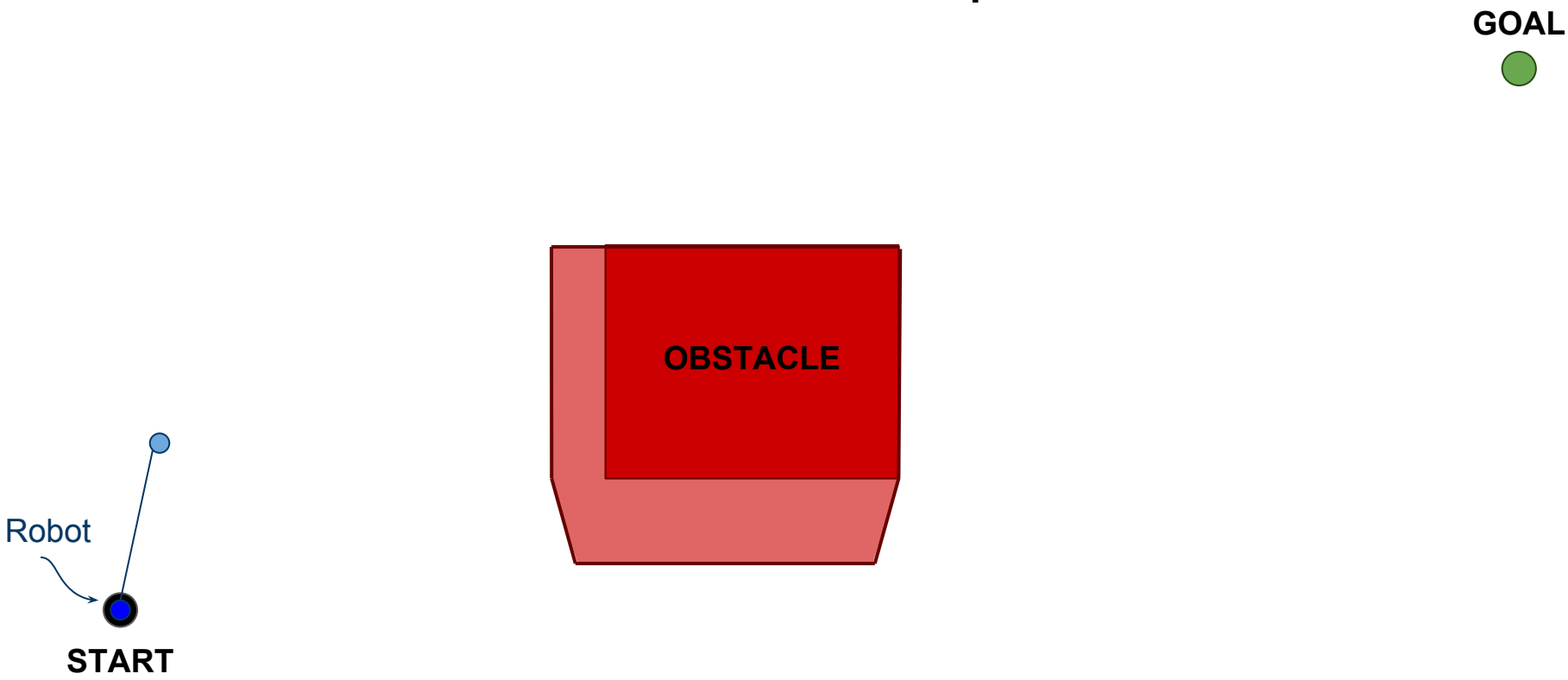
+ Multiple start states / queries

- To connect 2 nodes, must solve the **two-point boundary value problem** (2PBVP) *(board)*

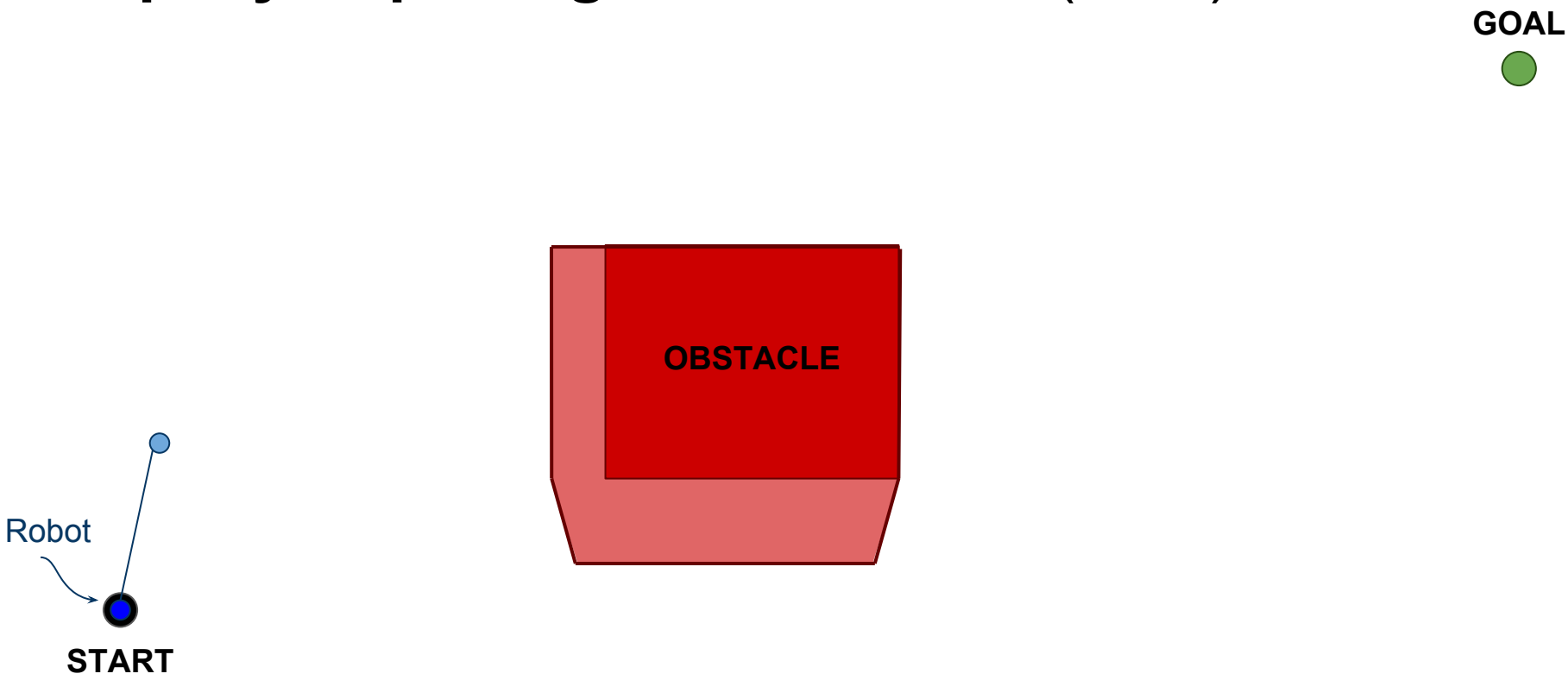


(Eric O. Scott for Wikipedia)

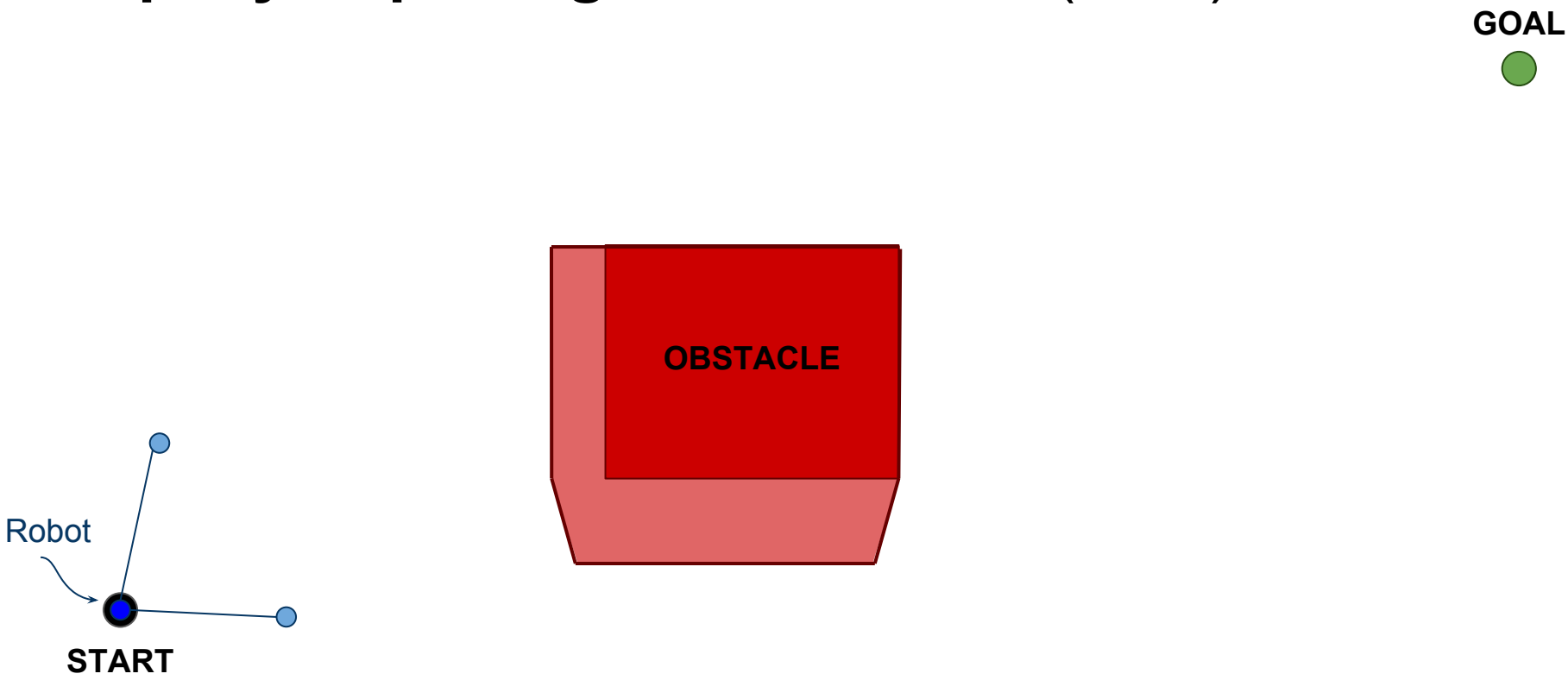
Idea: Execute random control inputs instead!



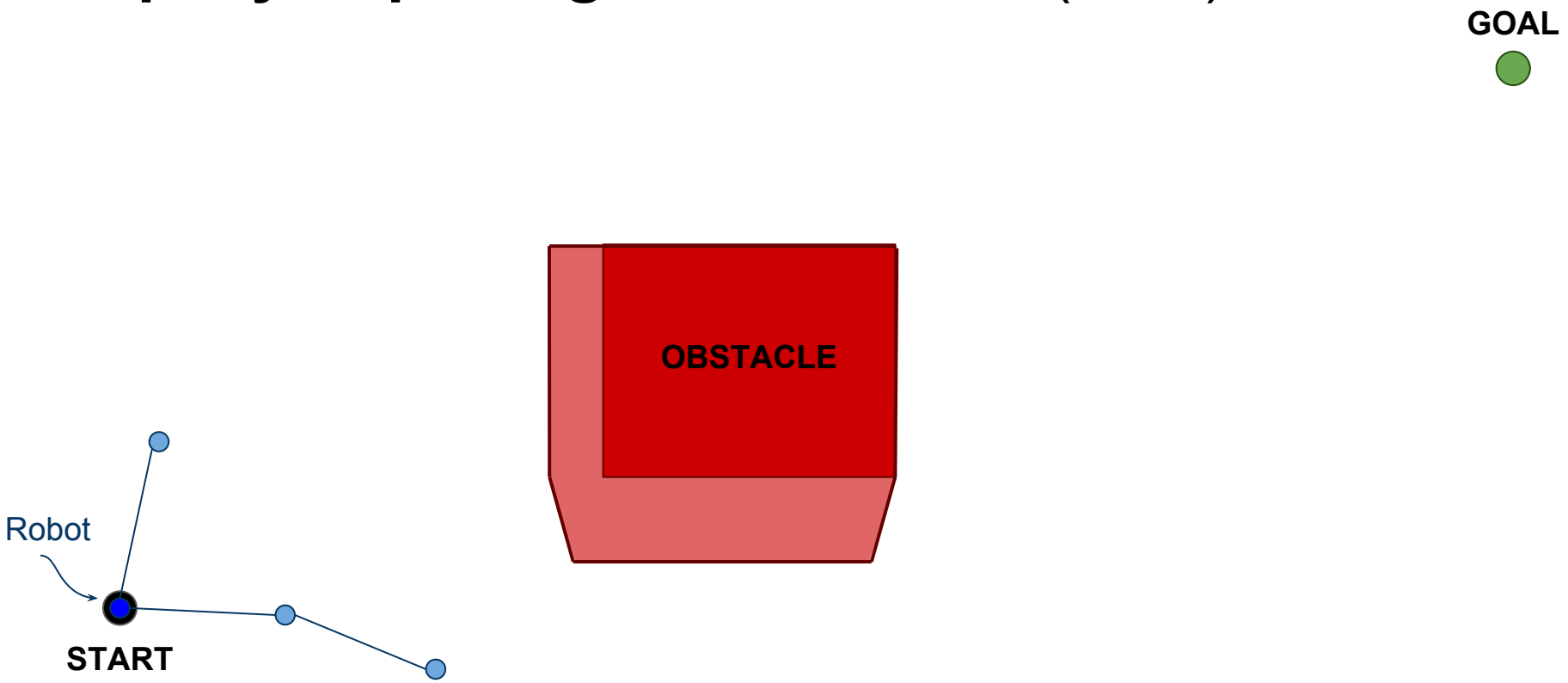
Rapidly-Exploring Random Tree (RRT)



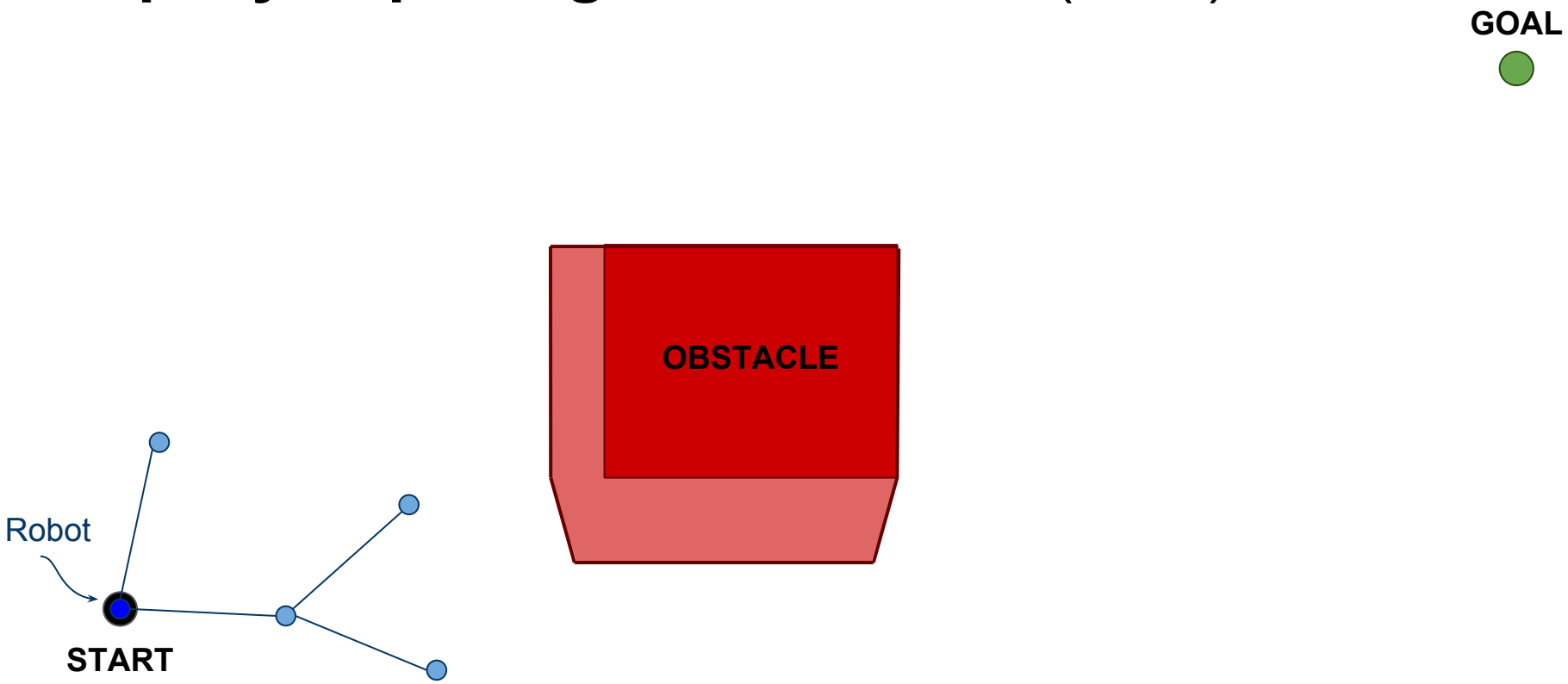
Rapidly-Exploring Random Tree (RRT)



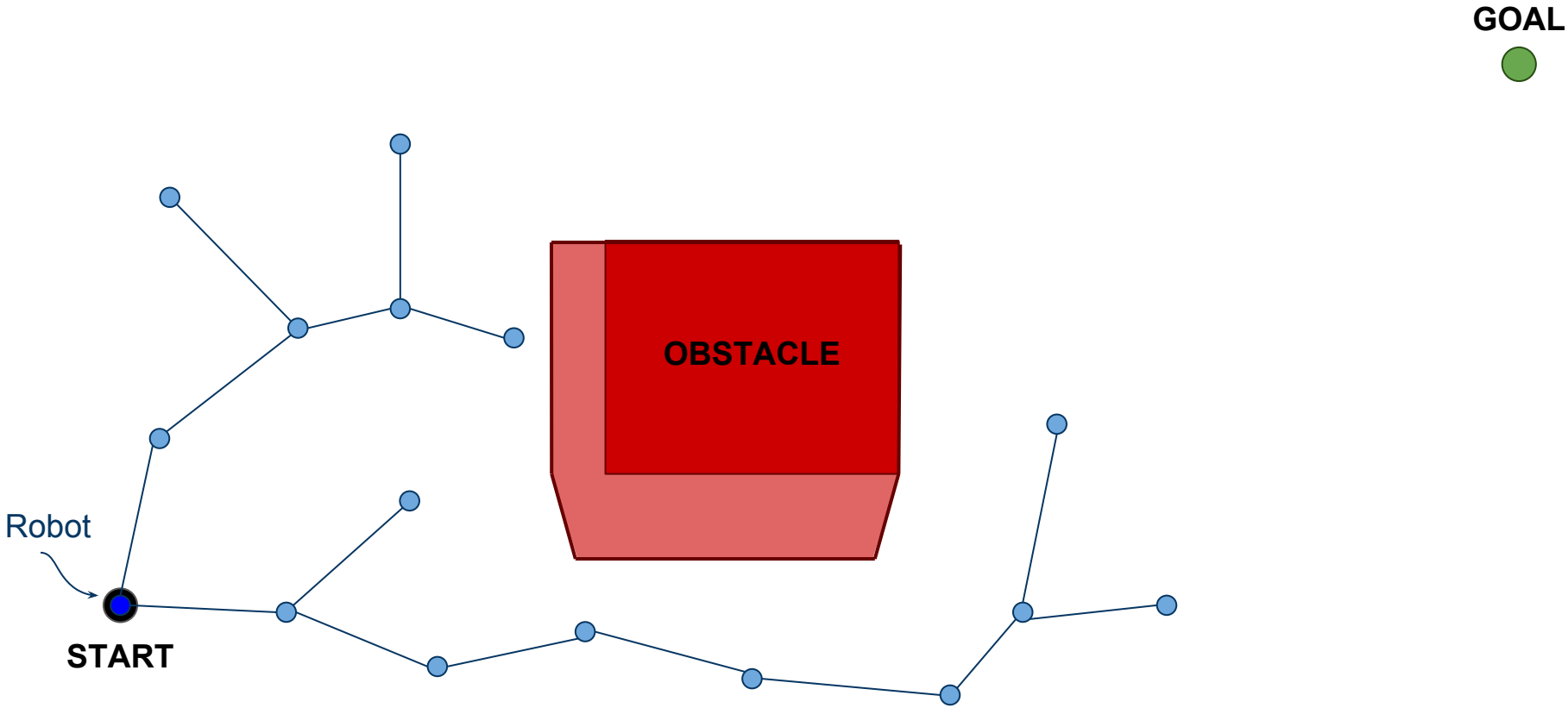
Rapidly-Exploring Random Tree (RRT)



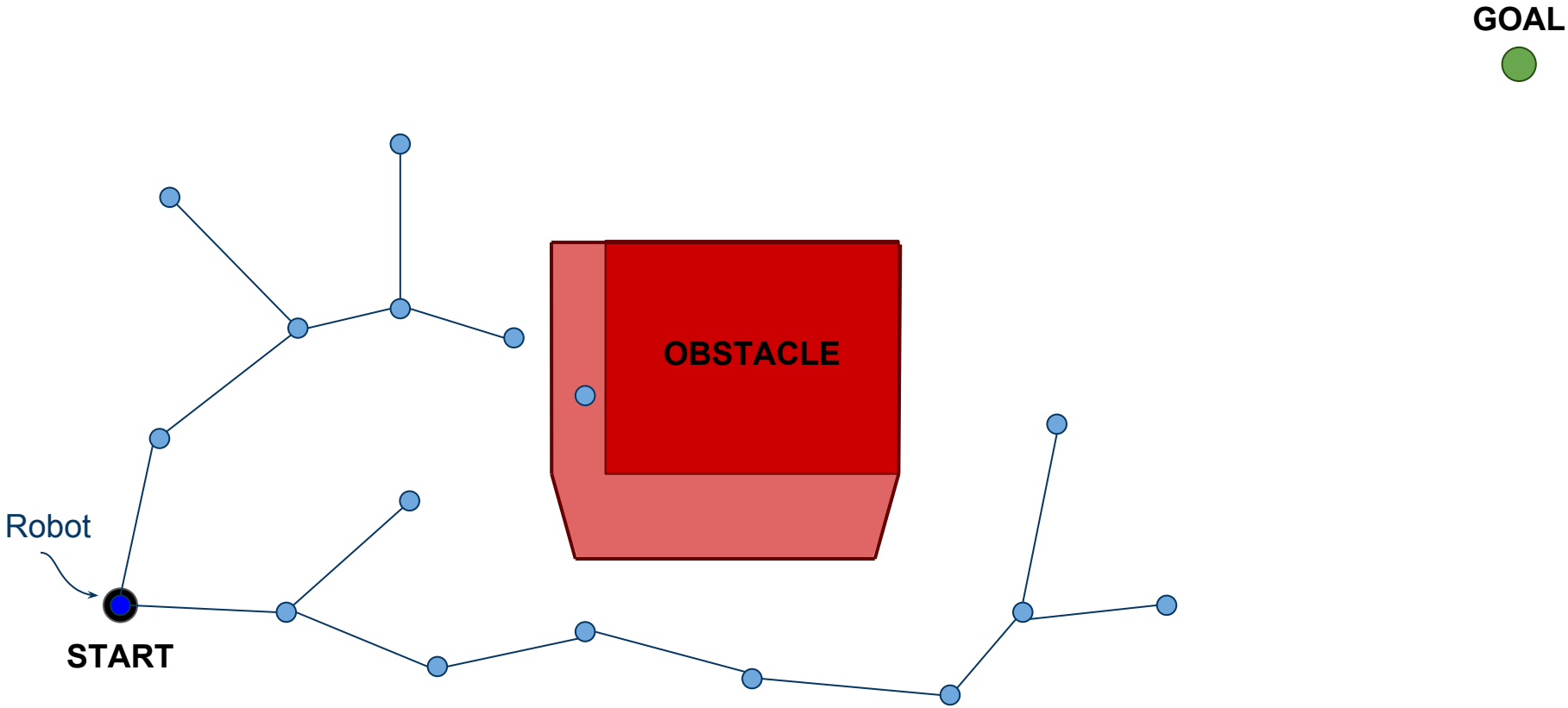
Rapidly-Exploring Random Tree (RRT)



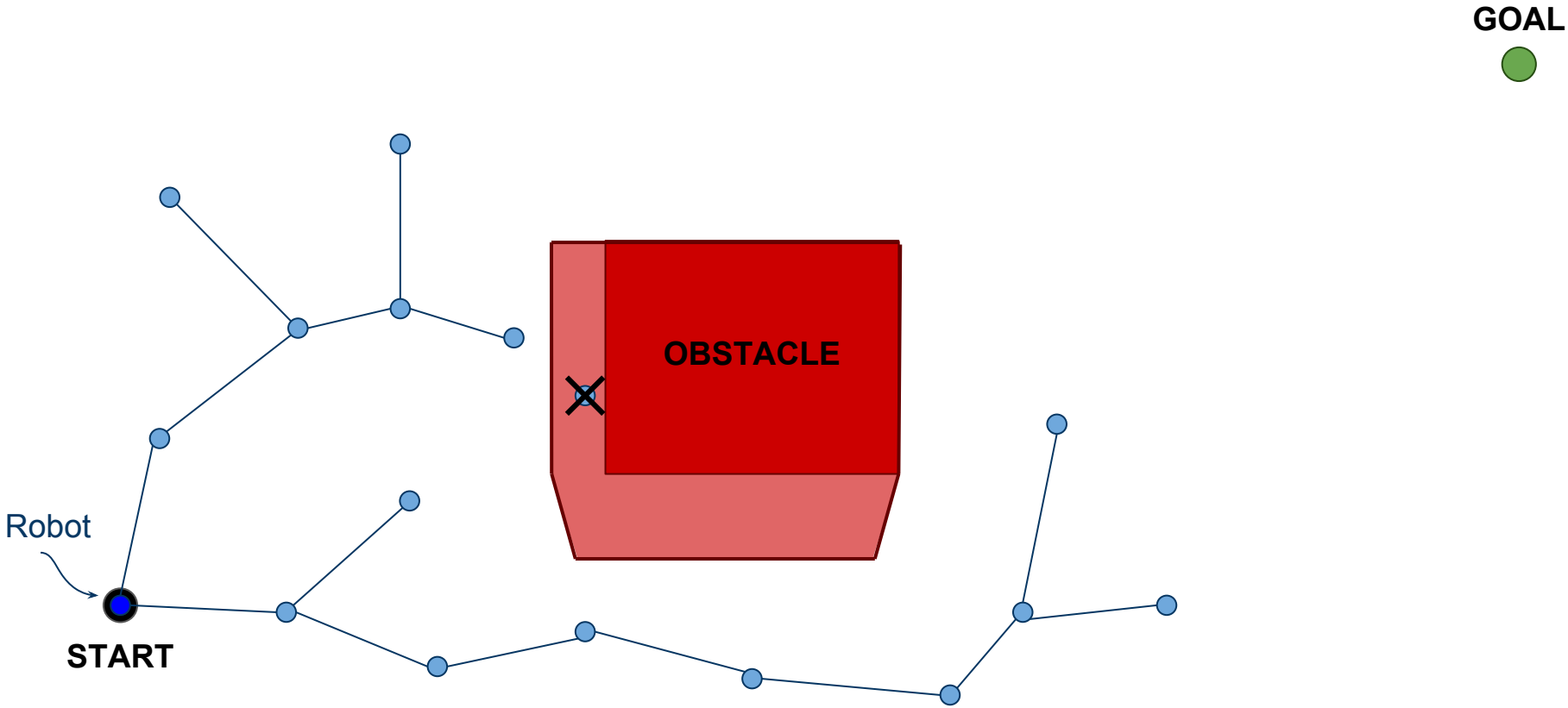
Rapidly-Exploring Random Tree (RRT)



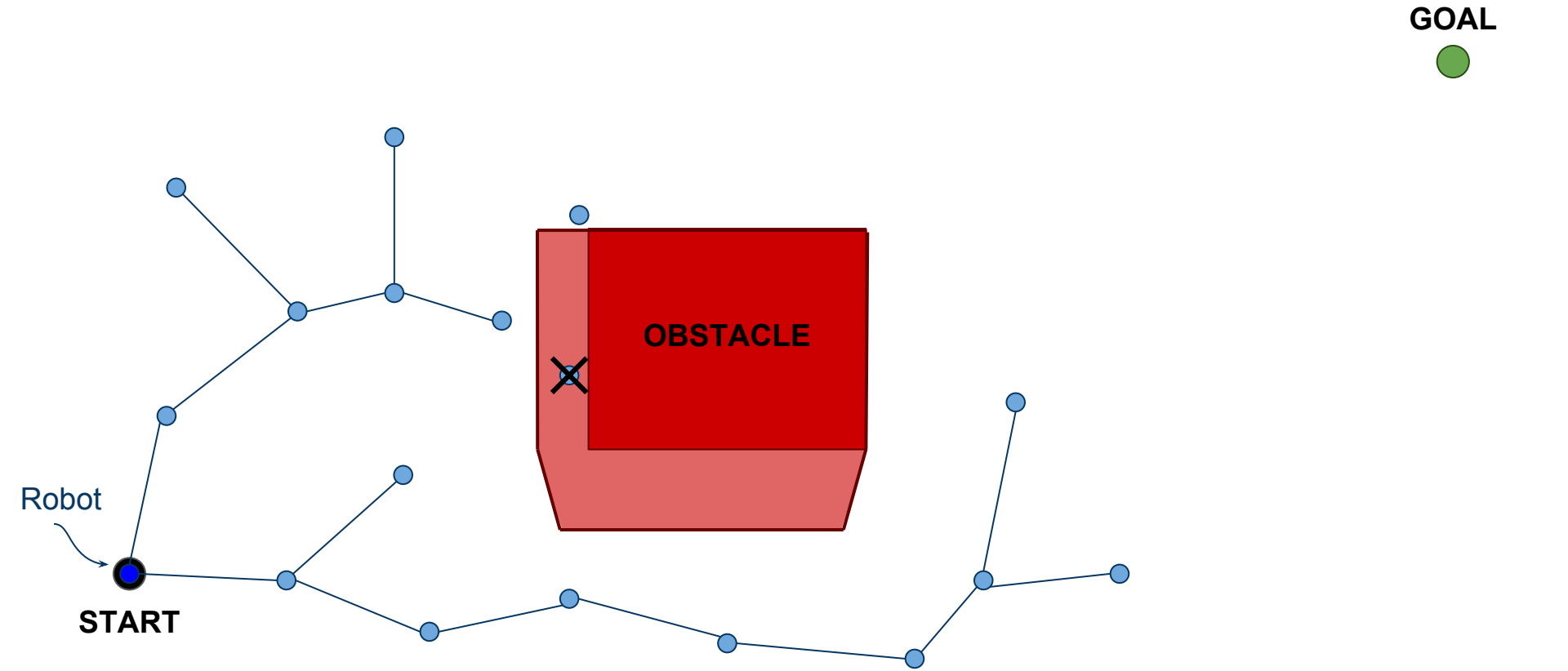
Rapidly-Exploring Random Tree (RRT)



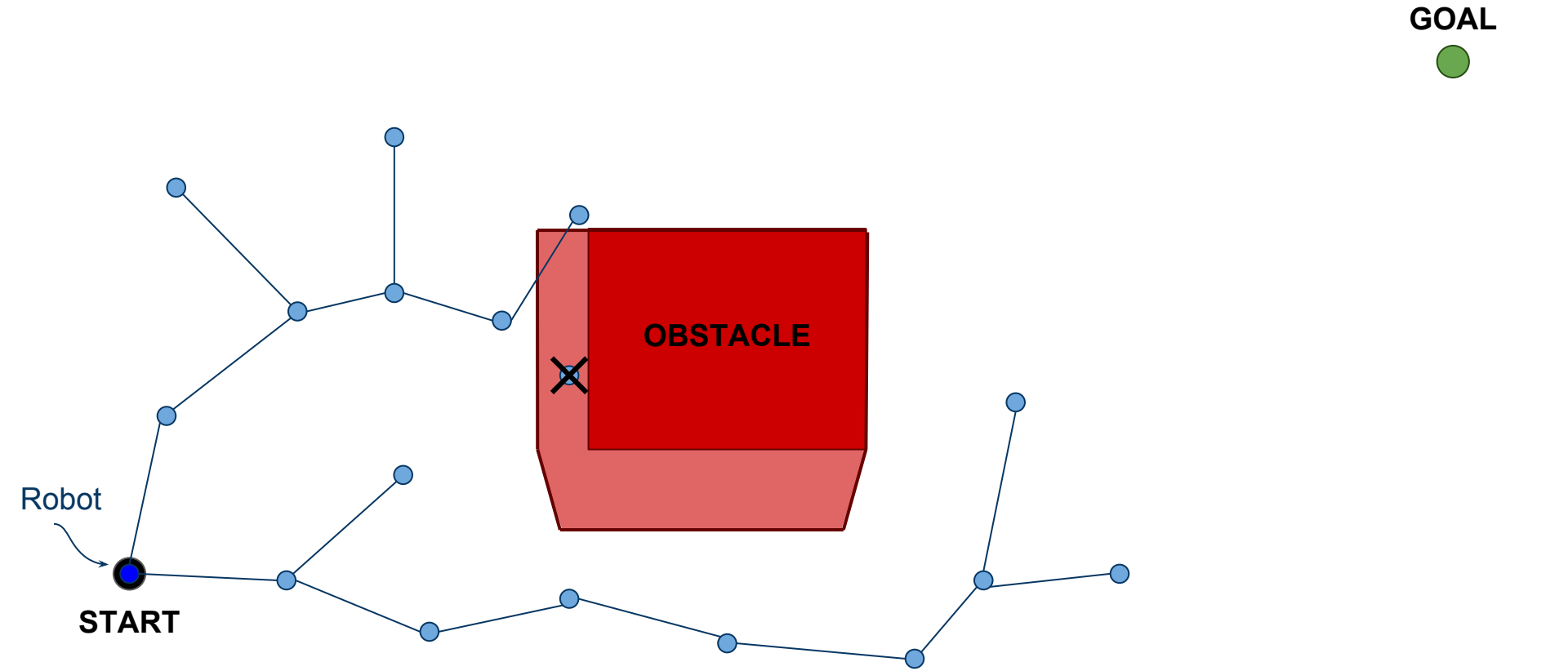
Rapidly-Exploring Random Tree (RRT)



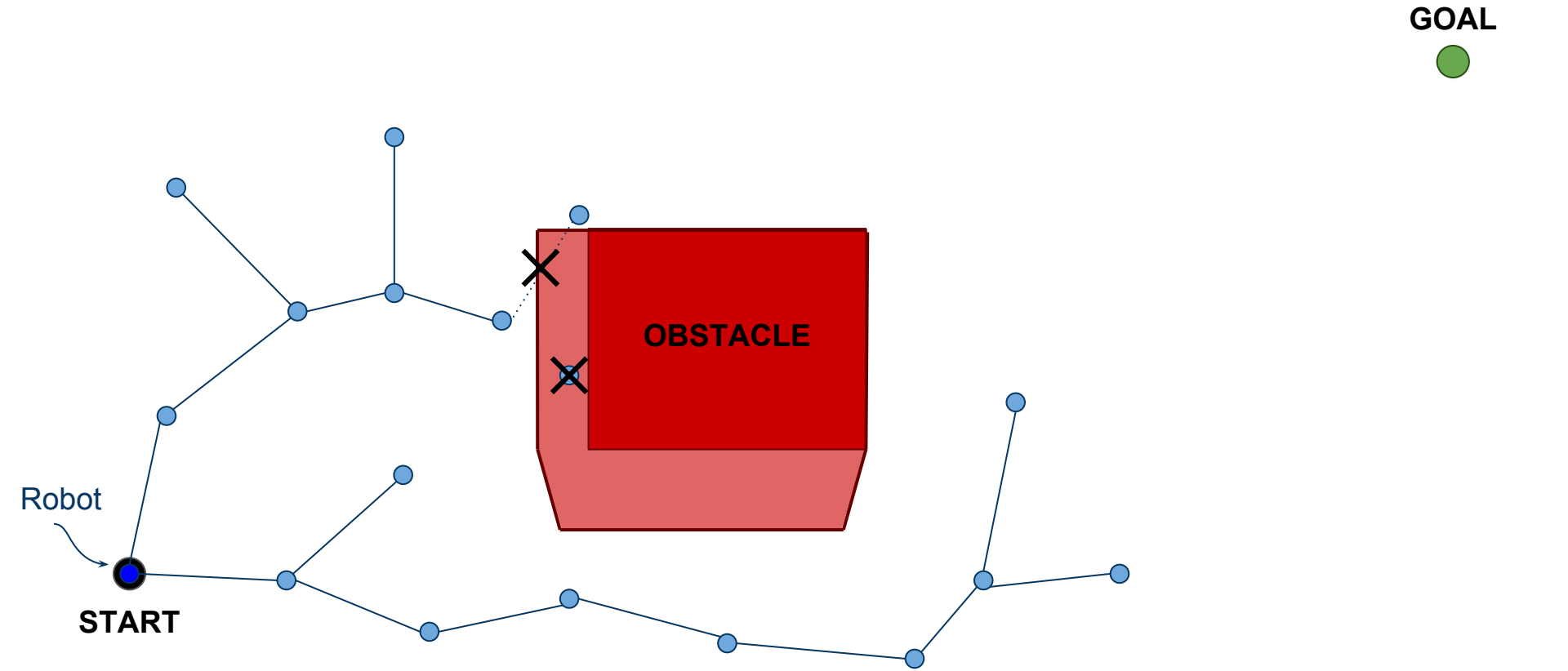
Rapidly-Exploring Random Tree (RRT)



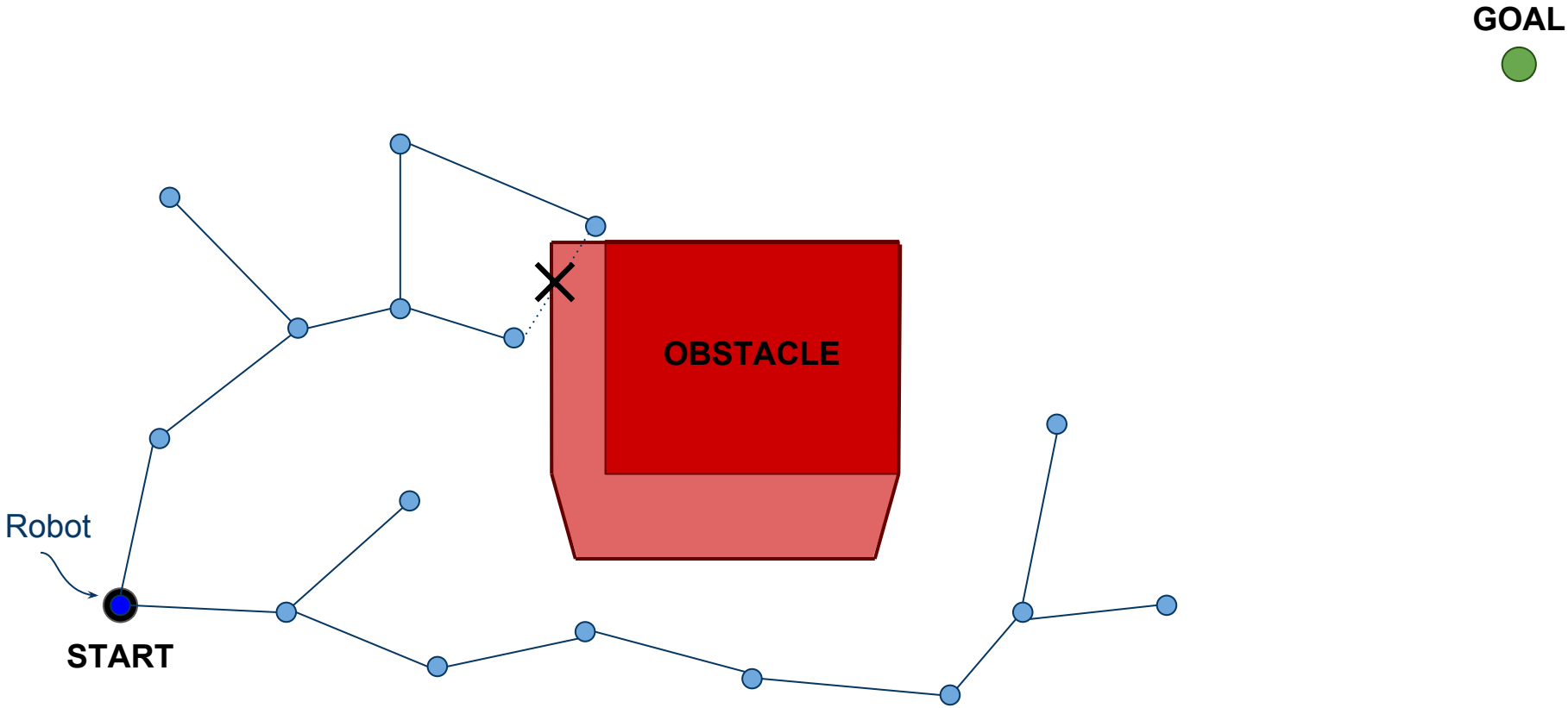
Rapidly-Exploring Random Tree (RRT)



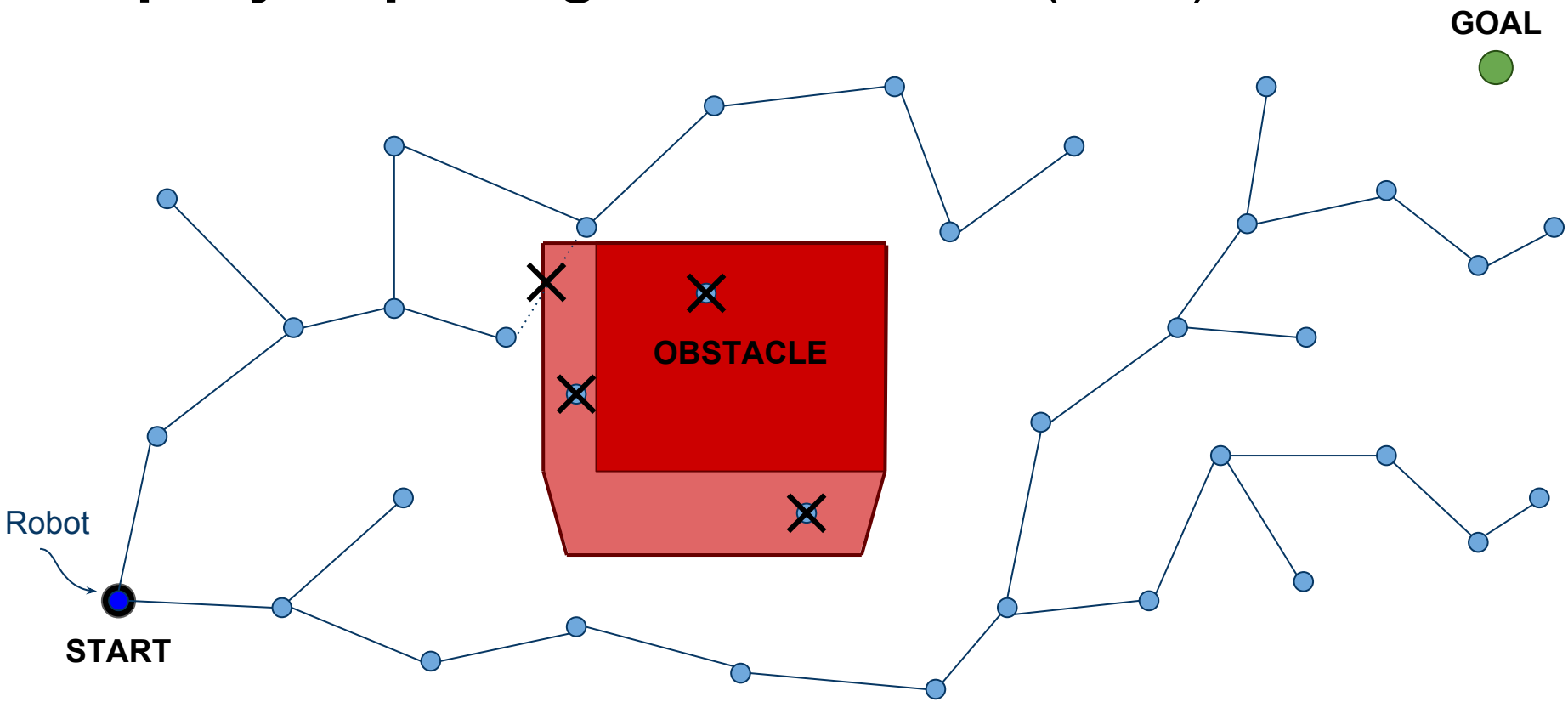
Rapidly-Exploring Random Tree (RRT)



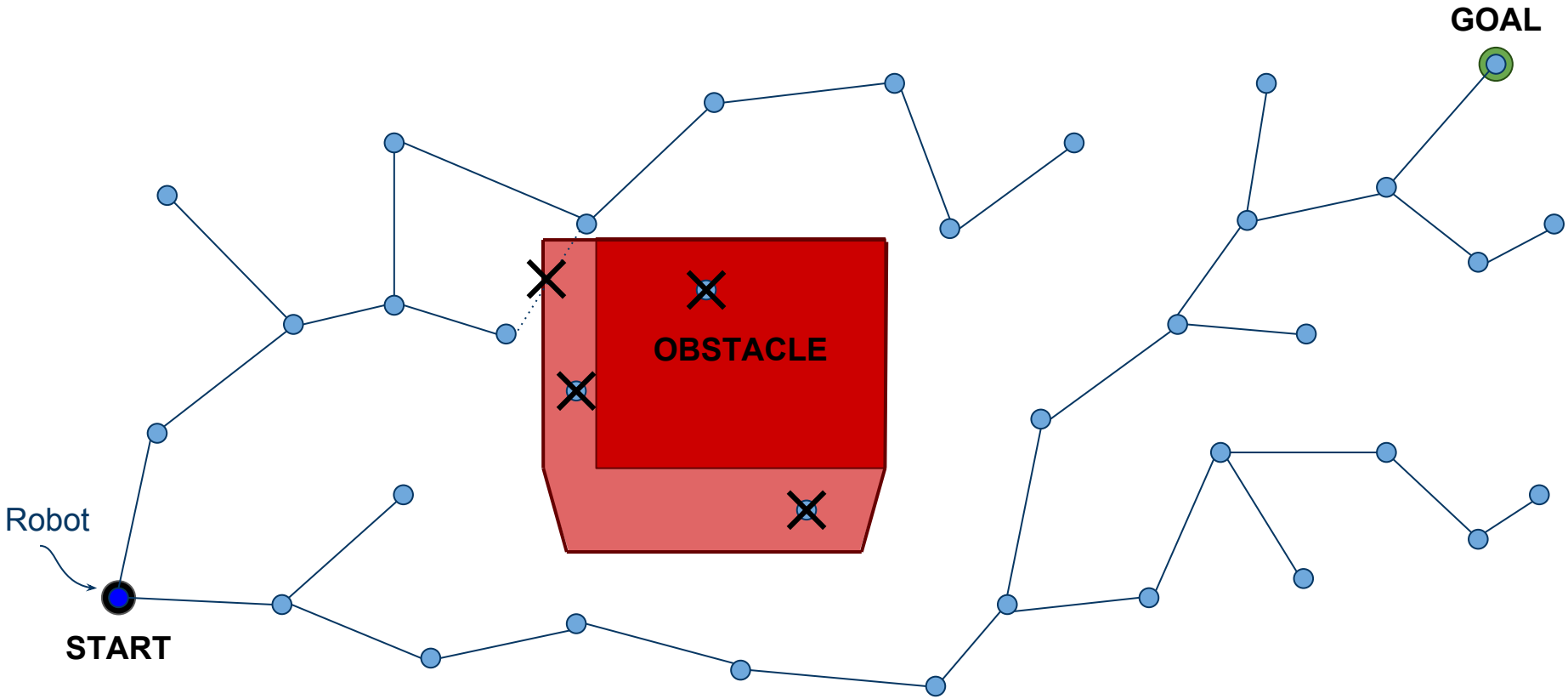
Rapidly-Exploring Random Tree (RRT)



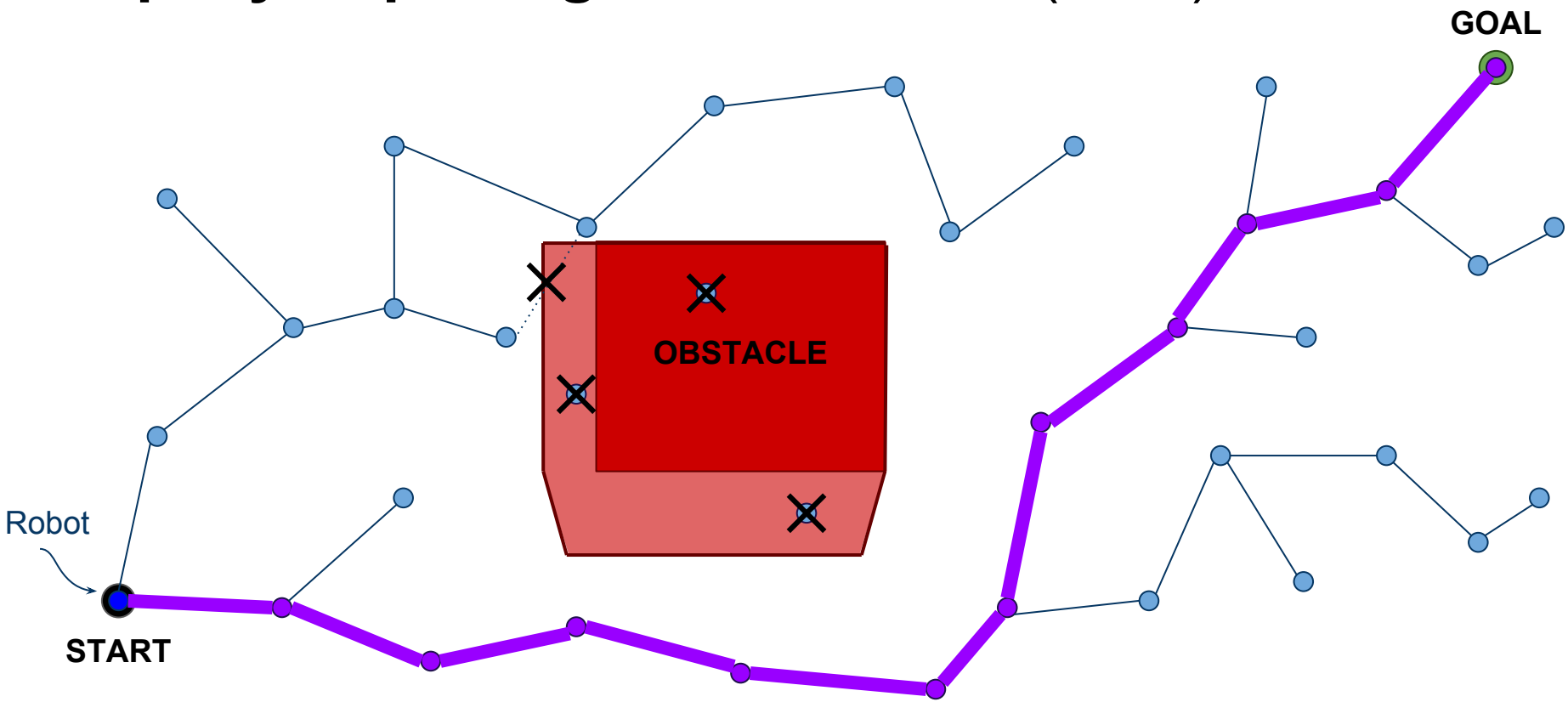
Rapidly-Exploring Random Tree (RRT)



Rapidly-Exploring Random Tree (RRT)



Rapidly-Exploring Random Tree (RRT)



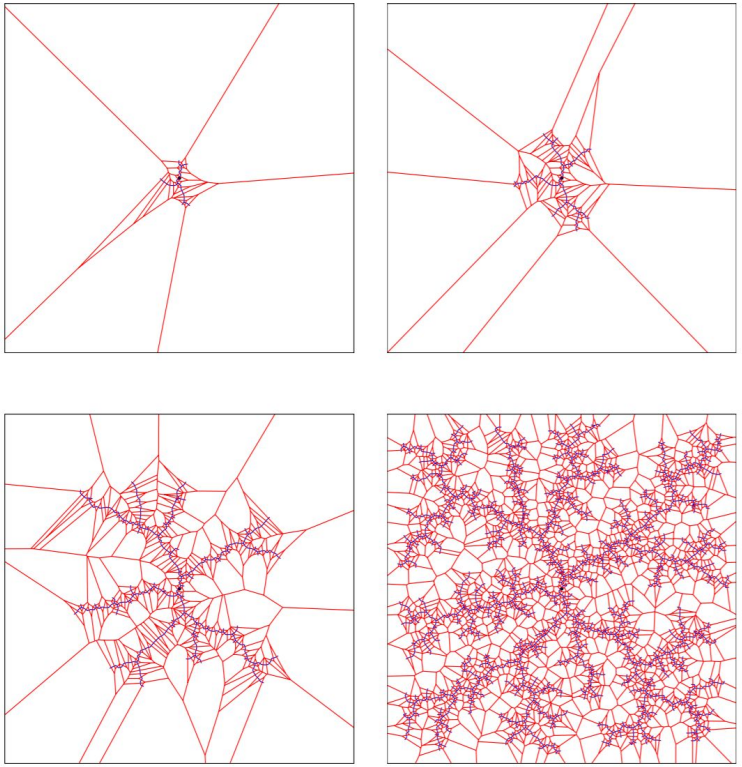
Rapidly-Exploring Random Tree (RRT)

+ Probabilistically complete

+ Handles high-DOF systems well

+ No 2PBVP

+ Exploration biased toward unexplored regions (**Voronoi bias**)



(Dellaert 2011)

Rapidly-Exploring Random Tree (RRT)

+ Probabilistically complete

+ Handles high-DOF systems well

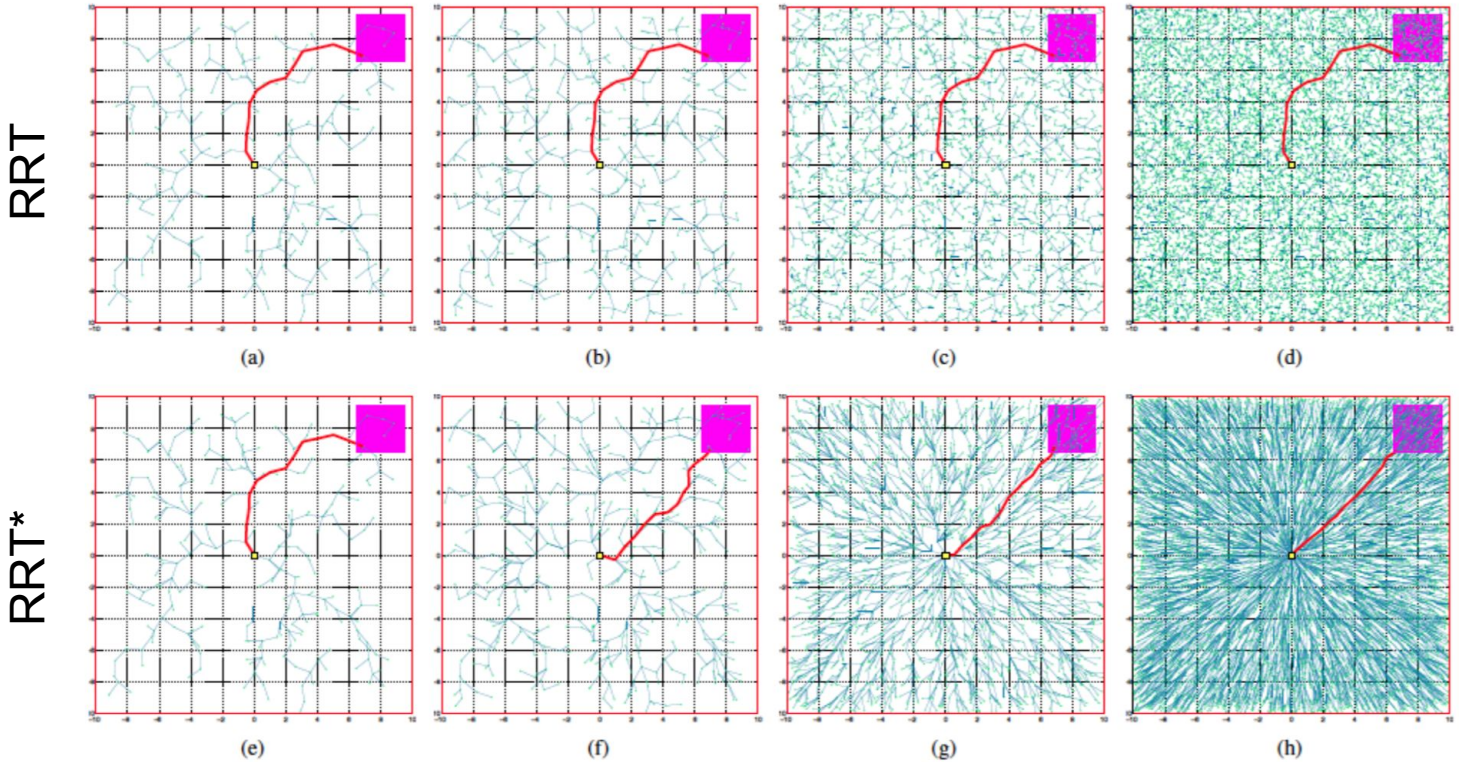
+ No 2PBVP

+ Exploration biased toward unexplored regions (**Voronoi bias**)

- Single start state / query

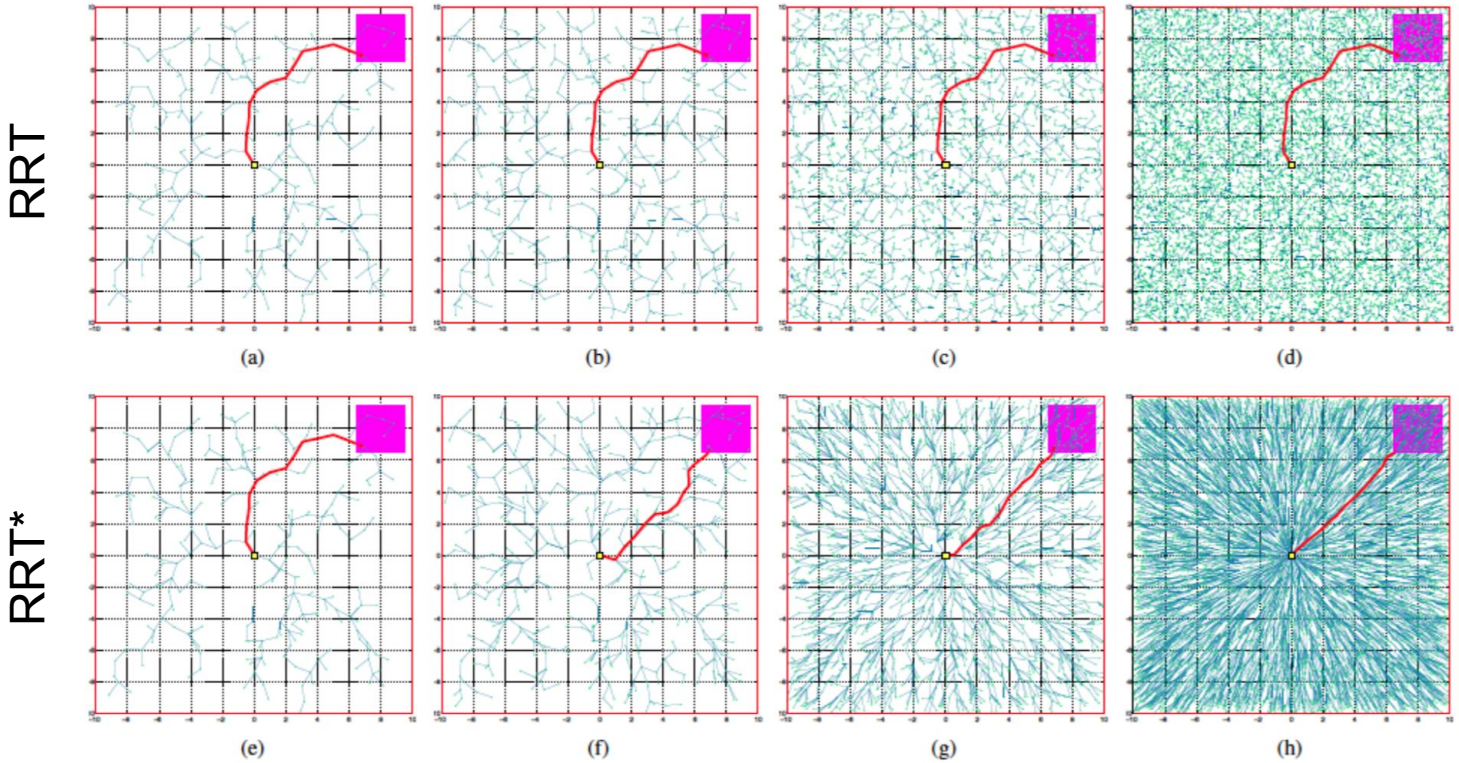
- **Excessively** suboptimal

Idea: Re-wire nodes in the tree → **RRT***



(Dellaert 2011)

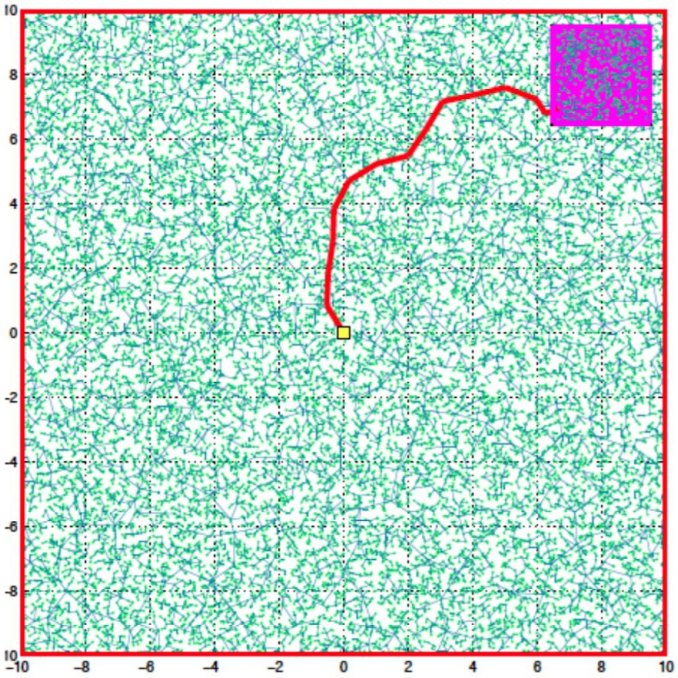
RRT*



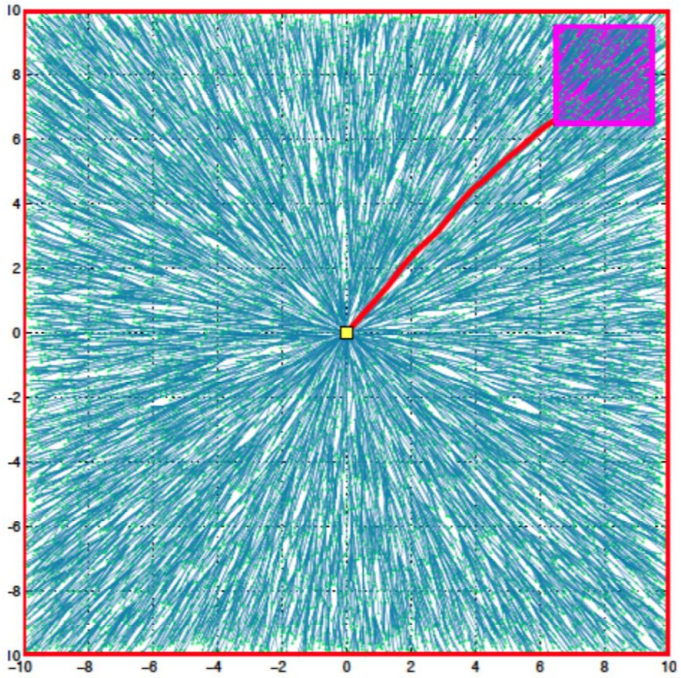
(Dellaert 2011)

RRT*

RRT



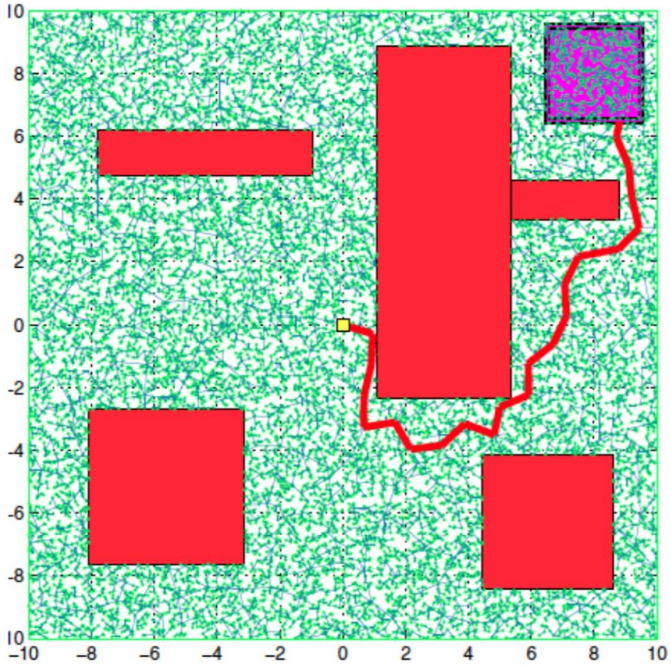
RRT*



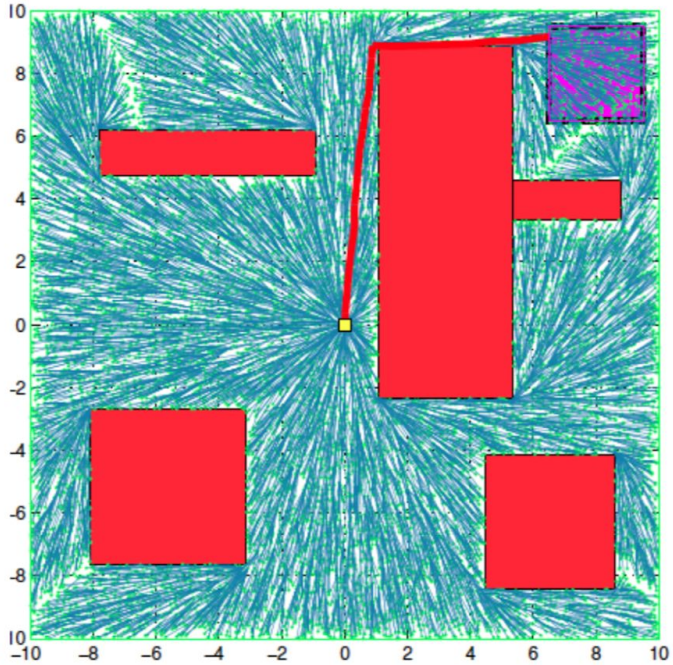
(Dellaert 2011)

RRT*

RRT



RRT*



(Dellaert 2011)

RRT*

+ Probabilistically complete

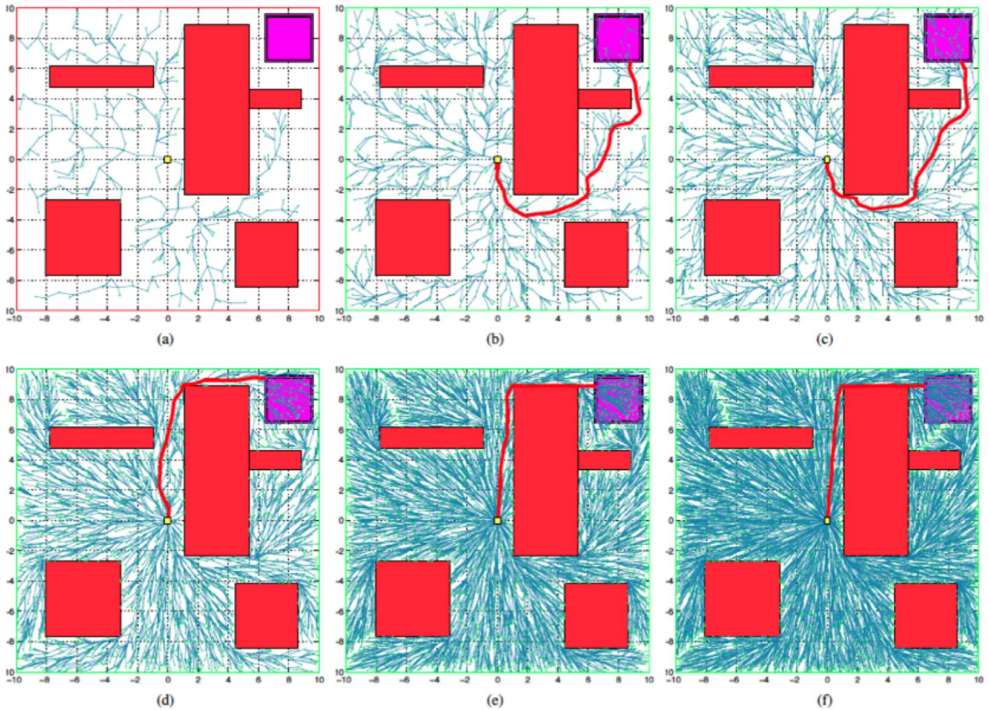
+ Handles high-DOF systems well

+ Exploration biased toward unexplored regions (**Voronoi bias**)

+ **Asymptotically optimal**

+ "**Anytime**" property

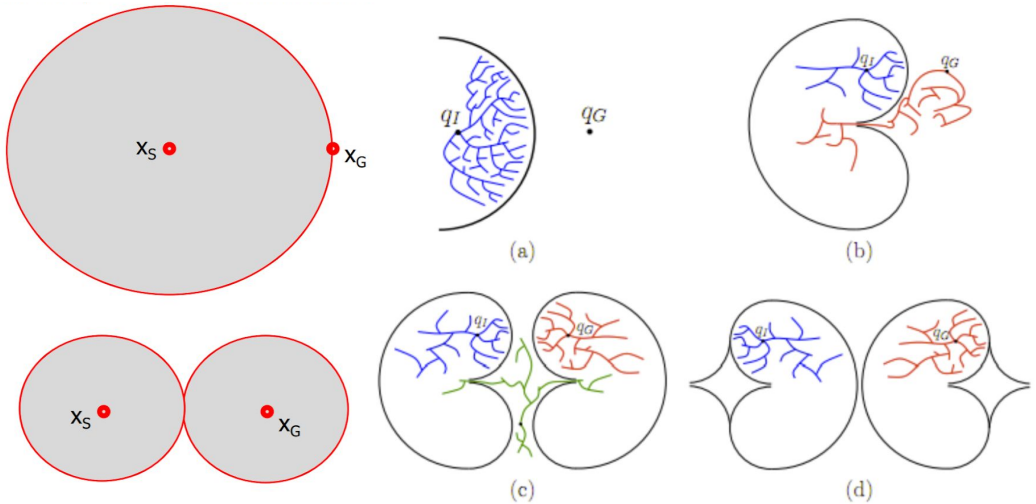
- Back to 2PBVP



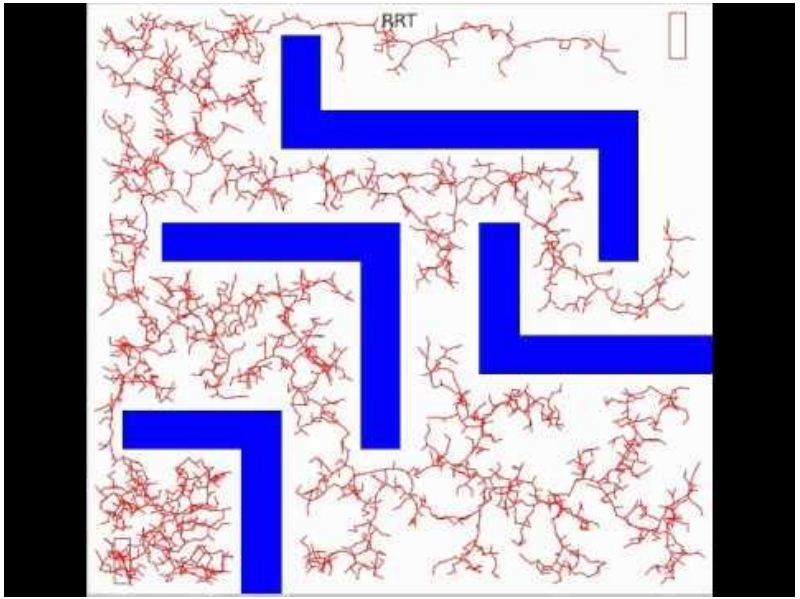
(Dellaert 2011)

Other RRT Variants (connection, smoothing, etc.)

Multi-Directional RRT



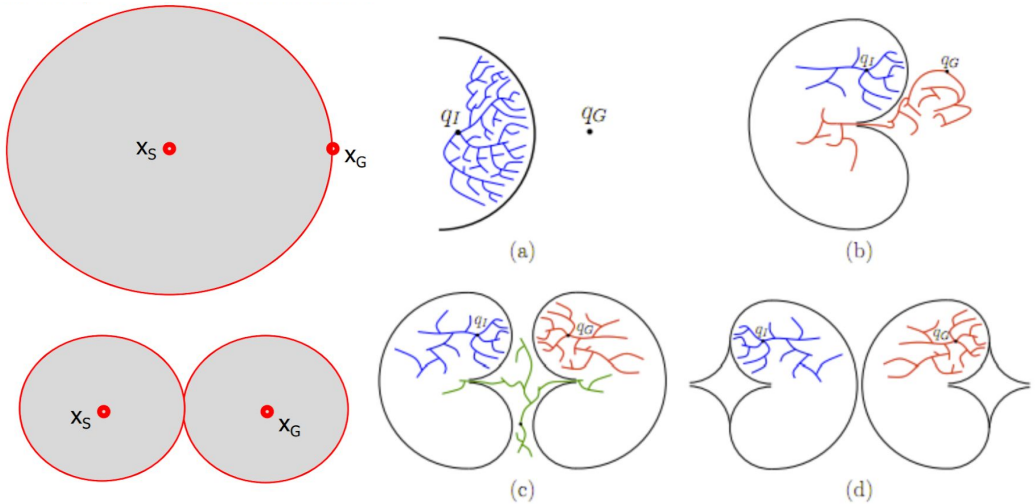
RRT-Connect



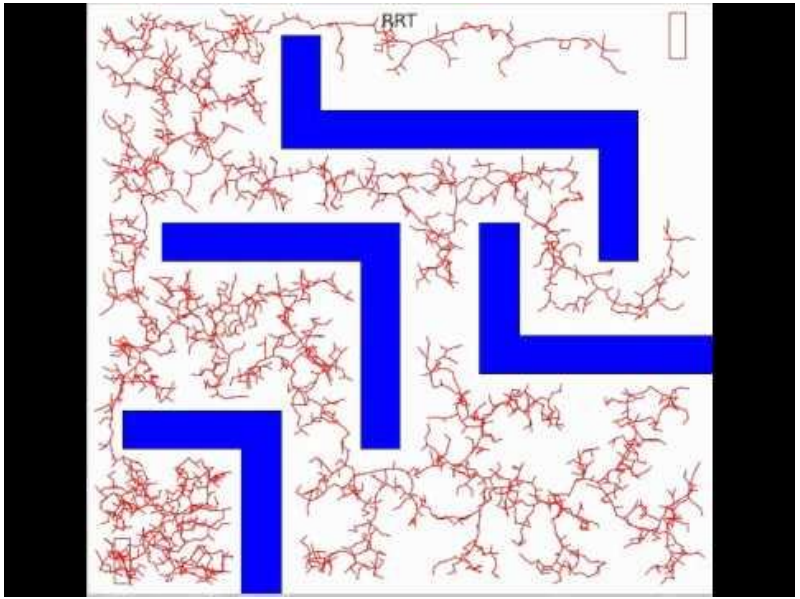
(Abbeel 2011)

Other RRT Variants (connection, smoothing, etc.)

Multi-Directional RRT



RRT-Connect



(Abbeel 2011)

. . . but we're not going to see any quadrotors running RRT(*) anytime soon.

Solution 2:

Optimization-based planners

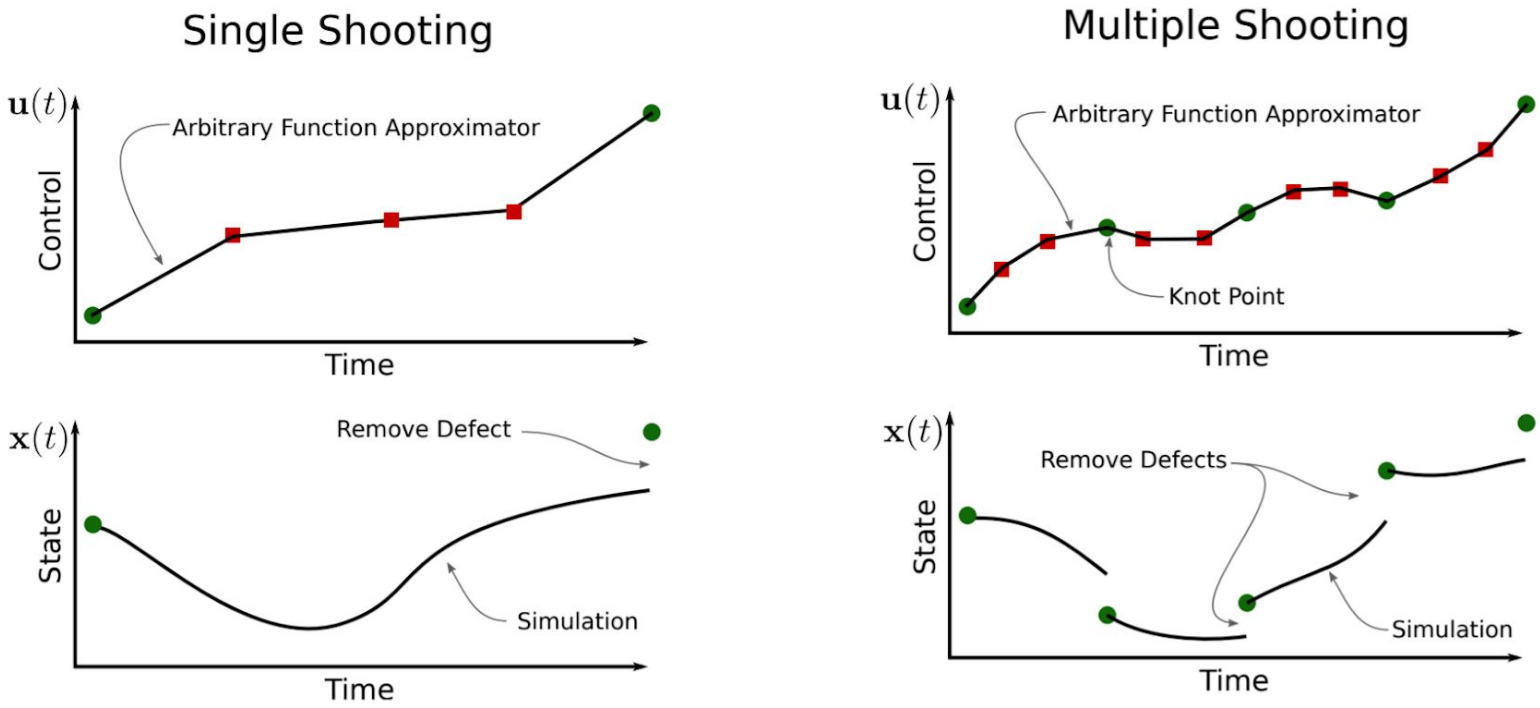
Idea: Sacrifice PC for real-time execution! (ish . . .)

Optimization-based planners / “trajectory optimization”

- Shooting methods
- Direct transcription & collocation methods

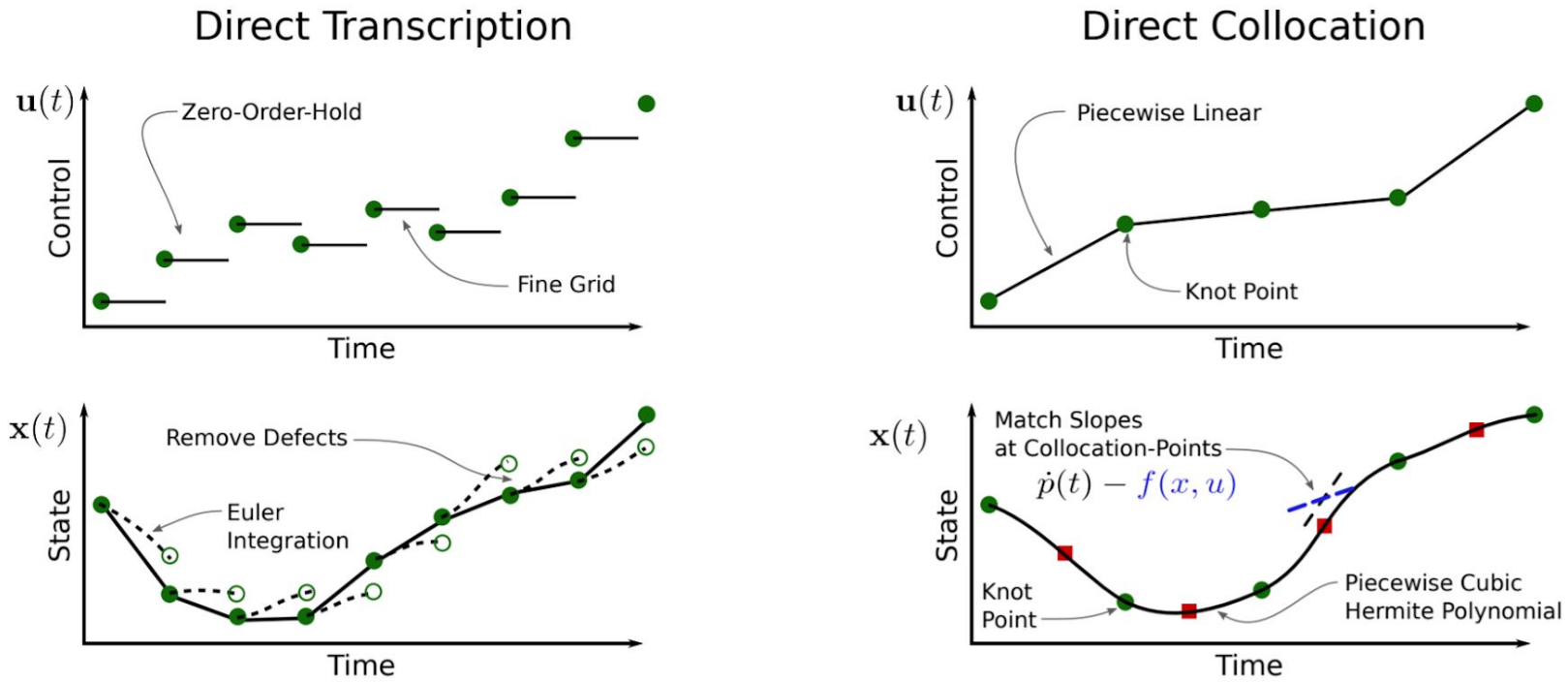
(board)

Shooting



(Kelly 2015)

Direct Transcription & Collocation



(Kelly 2015)

summary

Path Planning Summary

			Can handle . . . (reasonably well in practice)				
	Asymp. Optimal?	**Prob. Complete?**	**Multiple queries**	**Non-holonomy**	**Long planning horizons**	**Complicated geometry**	**High-DOF state space**
A* on rectilinear grid	N	N	ish?	N	Y	Y	N
A* on PRM*	Y	Y	Y	ish?	Y	Y	ish?
RRT	N	Y	N	Y	Y	Y	Y
RRT*	Y	Y	N	ish?	Y	Y	Y
Shooting	N	N	N	Y	N	N	Y
Collocation	N	N	N	Y	N	Y	Y

***Disclaimer**: YMMV. Motion planning researchers are still fighting over every single one of these squares.*

Still lots of work to be done!