Dijsktra's Algorithm & Sampling-based Algorithms

Recall: Computing Shortest Paths

- **Shortest Paths:**
  - Given:
    - A graph with weights on the edges:

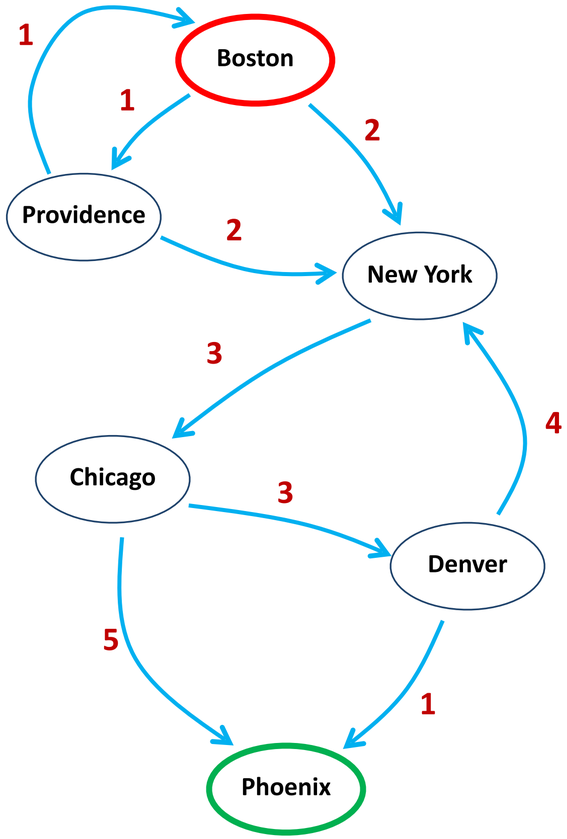
$$G = (V, E) \quad c : V \to \mathbb{R}$$

    - A start vertex and a goal vertex:

$$v_s, v_g \in V$$

  - Find the shortest path that connects start vertex to the goal vertex

$$(v^{(1)}, v^{(2)}, \ldots, v^{(k)}) = \arg\min \left(\sum_{i=1}^{n} c\left(v^{(i)}\right) \right)$$

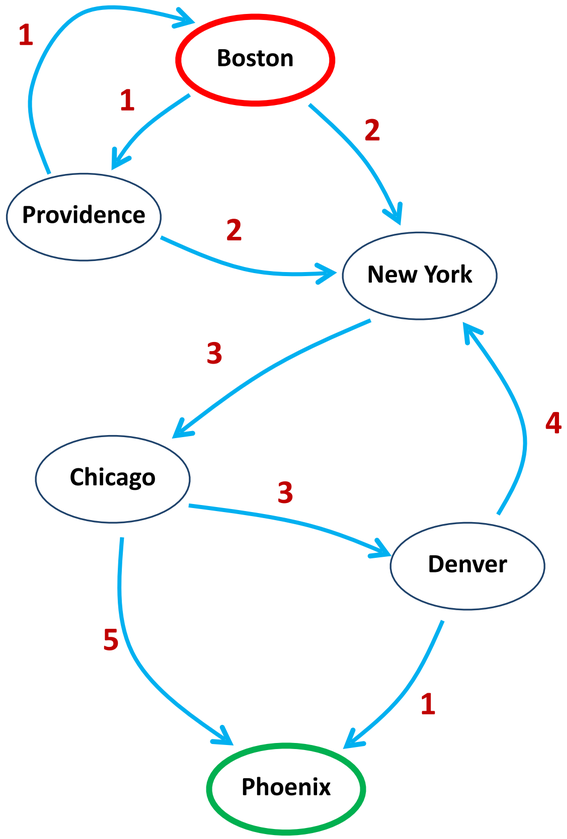


Recall: Quick introduction to Dijkstra's algorithm

Dijsktra's algorithm:

- Keep track of the cost to get to each vertex, and keep track of the 'parent' that gets to the vertex at the lowest cost.
- Allow revisiting a vertex, if it lowers the cost to get to that vertex. *Otherwise, operates much like the BFS.*

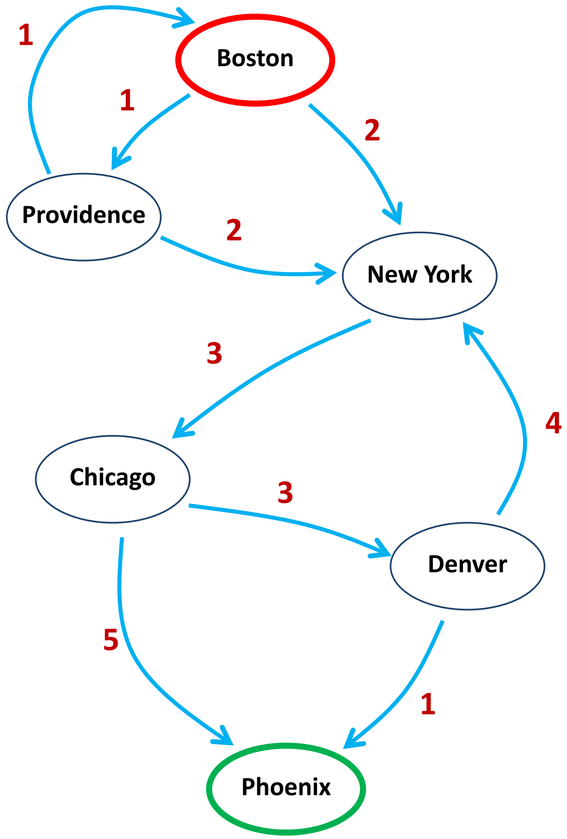
- *Notice: The algorithm computes the shortest path from start vertex to every other vertex (not just the goal).*



Recall: Quick introduction to Dijkstra's algorithm

Dijsktra's algorithm:

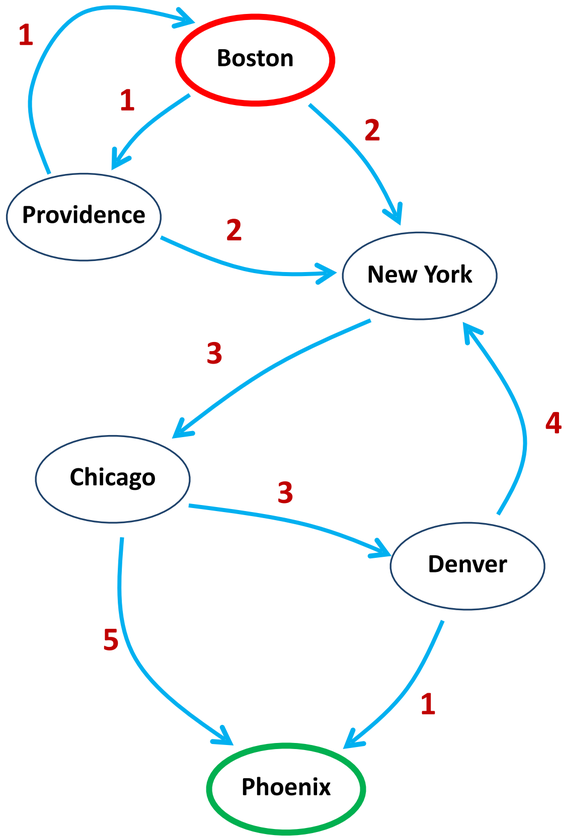
- We will consider a "priority queue" where elements are according to cost-to-get-to-that-vertex
- Initialize with adding all vertices to the queue with
  - Start vertex cost = 0
  - All other vertex cost = infinity
- Repeat until the queue is empty:
  - Pull from the queue (the vertex with shortest cost), say vertex *u*
  - For each edge *(u,v)* going out from the vertex, compute the cost to get to v through u, and update the cost-to-get-to v, if the new route has lower cost.



Recall: Quick introduction to Dijkstra's algorithm

Dijsktra's algorithm (in pseudo code)

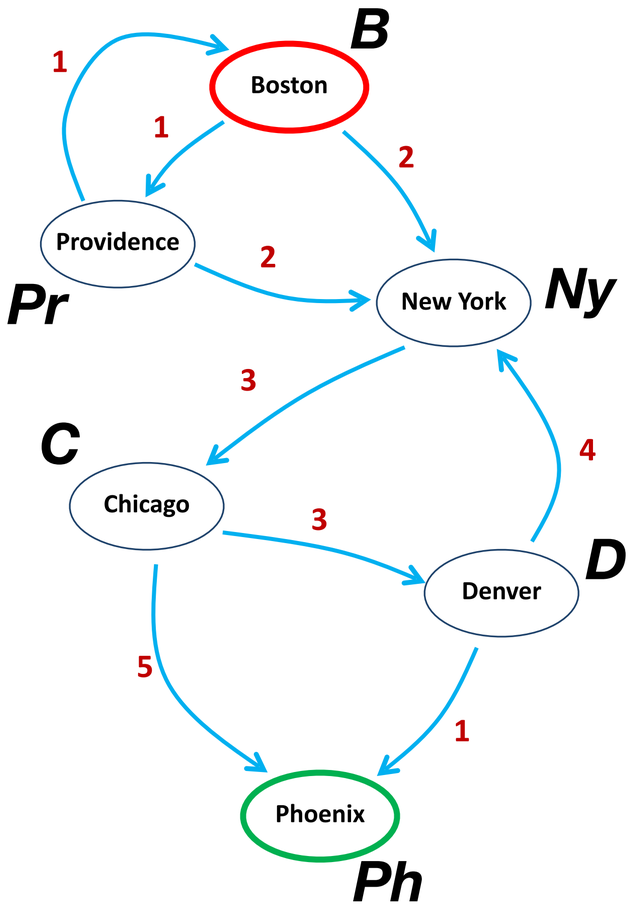
- Q: denotes the priority queue
- Cost: denotes the cost to get to a vertex
- Parent: denotes the parent vertex to get to a vertex through the shortest route
- Initially, Cost and Parent will be incorrect. The algorithm will update Cost and Parent until convergence, and Cost will only decrease as the algorithm progresses.
- Initialize:
  - For each vertex v in V: Insert v into Q with
    - Cost[v] = infinity
    - Parent[v] = Null
  - Cost[v] = 0
  - While Q is not empty:
    - u = Extract from Queue (Q)
    - For each edge (u,v):
      - If Cost[u] + c((u,v)) < Cost[v]
        - Parent[v] = u
        - Cost[v] = Cost[u] + c((u,v))



Recall: Quick introduction to Dijkstra's algorithm

Example

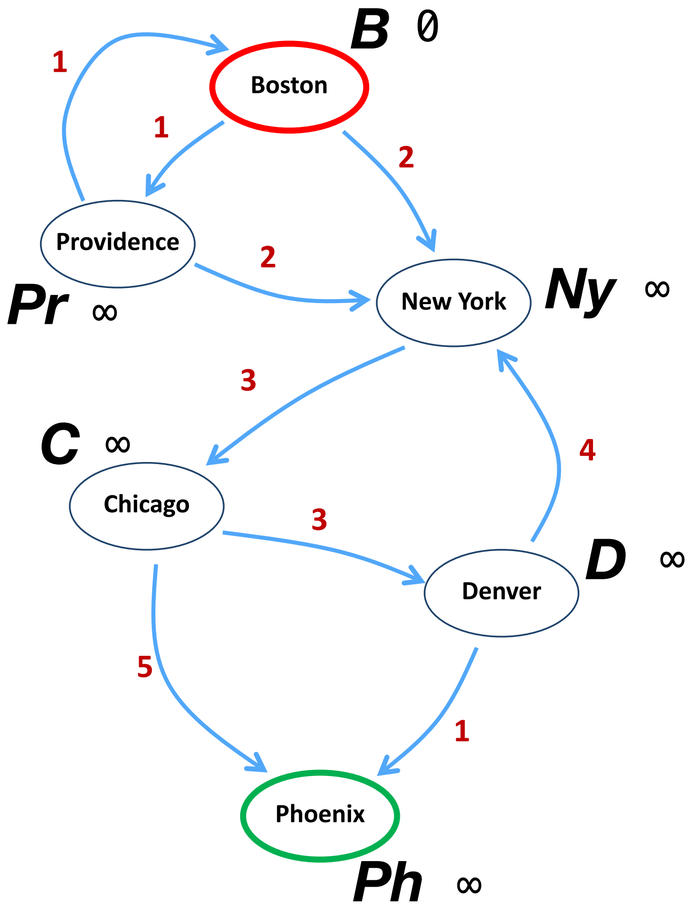
Let's do it on the board!



Recall: Quick introduction to Dijkstra's algorithm

Example

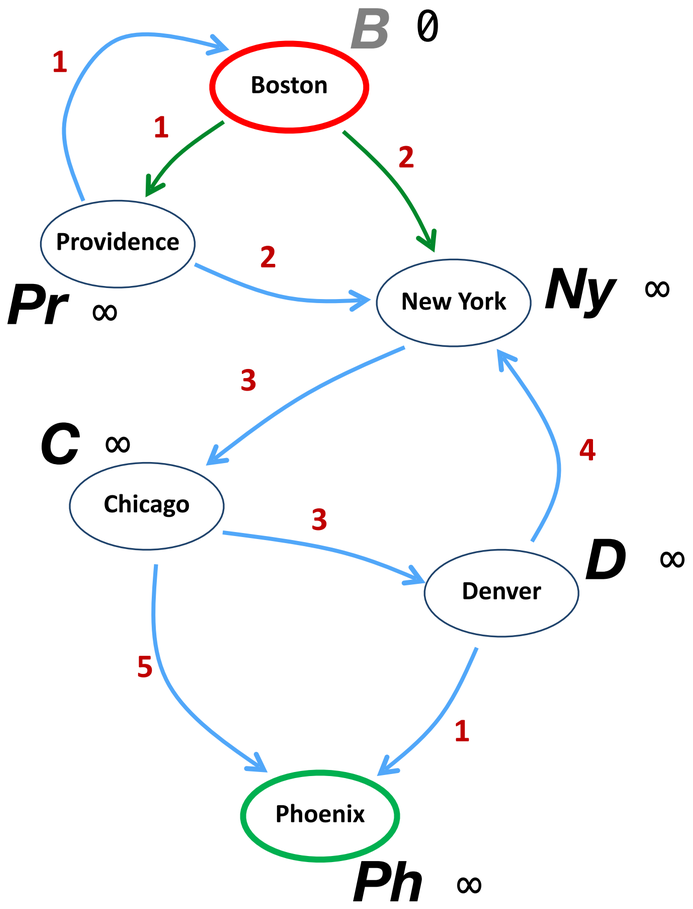
Add all vertices to the queue



Recall: Quick introduction to Dijkstra's algorithm

Example

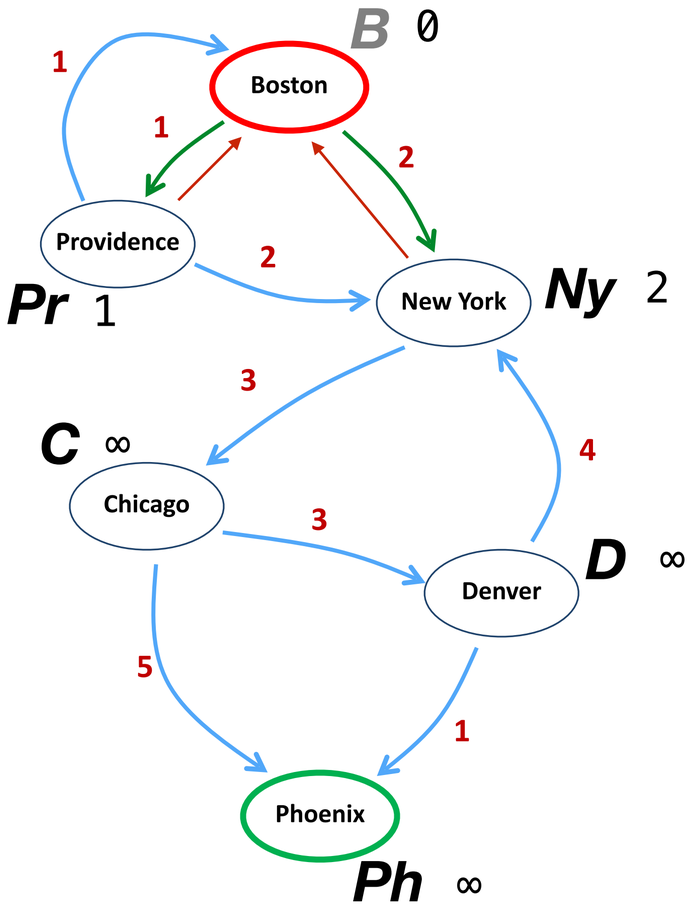
Pull B from the Queue



Recall: Quick introduction to Dijkstra's algorithm

Example

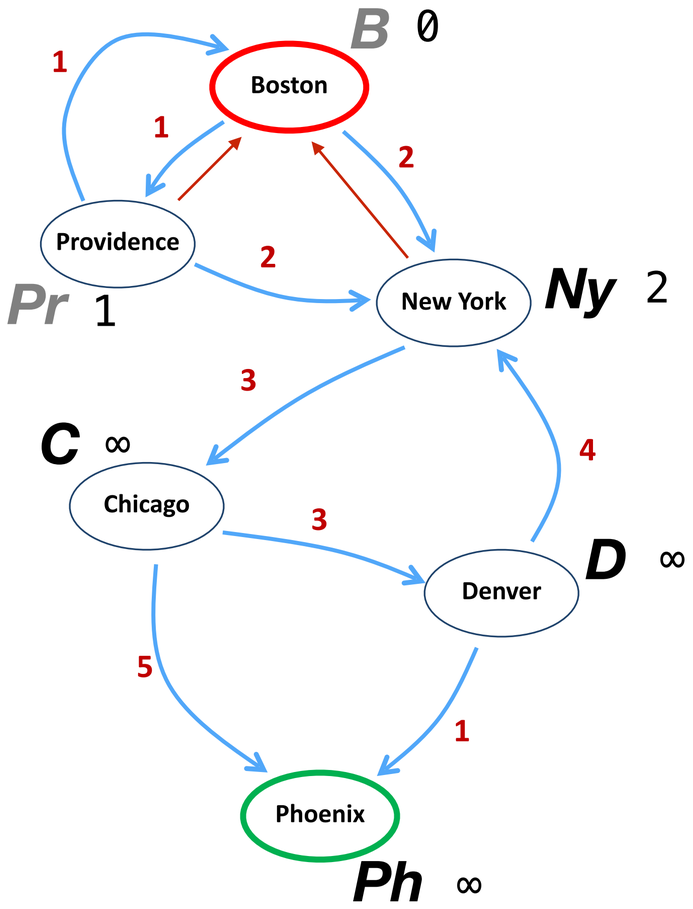
Pull B from the Queue
Consider updating cost to get to Pr and Ny



Recall: Quick introduction to Dijkstra's algorithm

Example

Pull Pr from the Queue

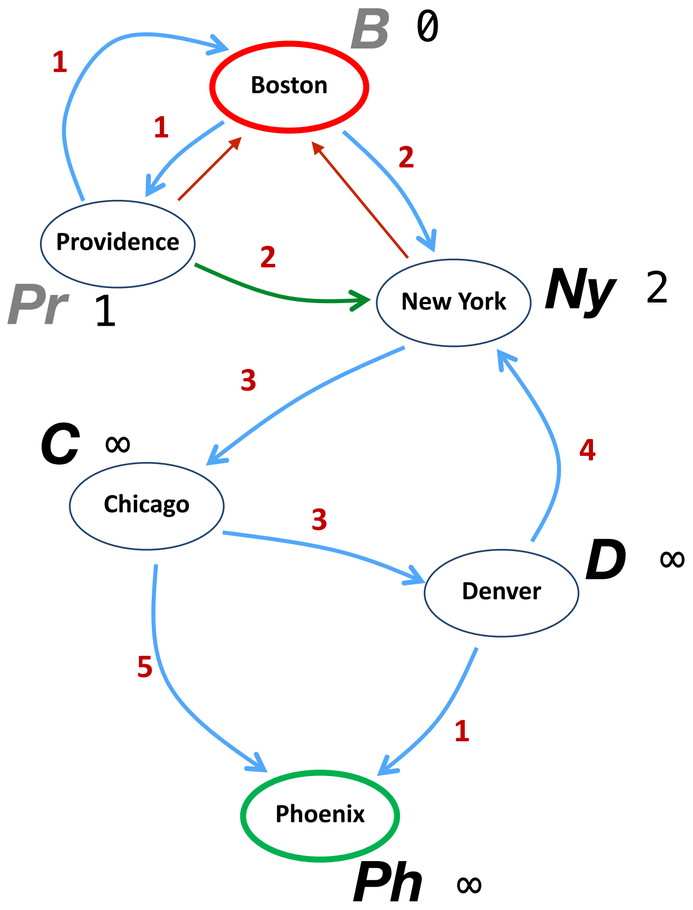


Recall: Quick introduction to Dijkstra's algorithm

Example

Pull Pr from the Queue
Consider updating cost to get to Ny

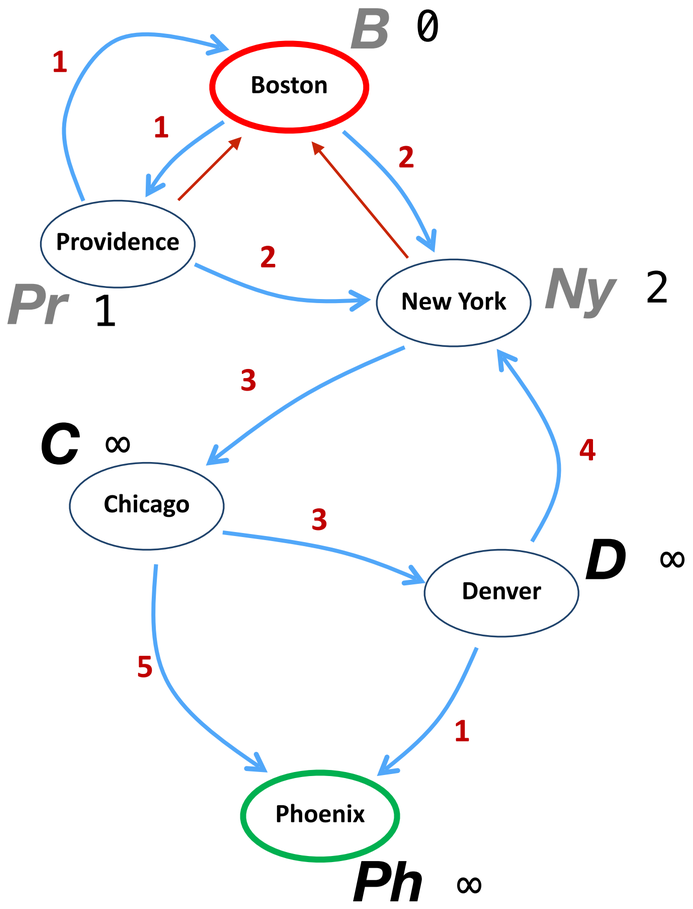
(Notice that the cost to get to Ny does not change. Going from Boston to Ny directly costs 2 units which is less than the cost 3 units of going from Boston to Ny through Pr .)



Recall: Quick introduction to Dijkstra's algorithm

Example

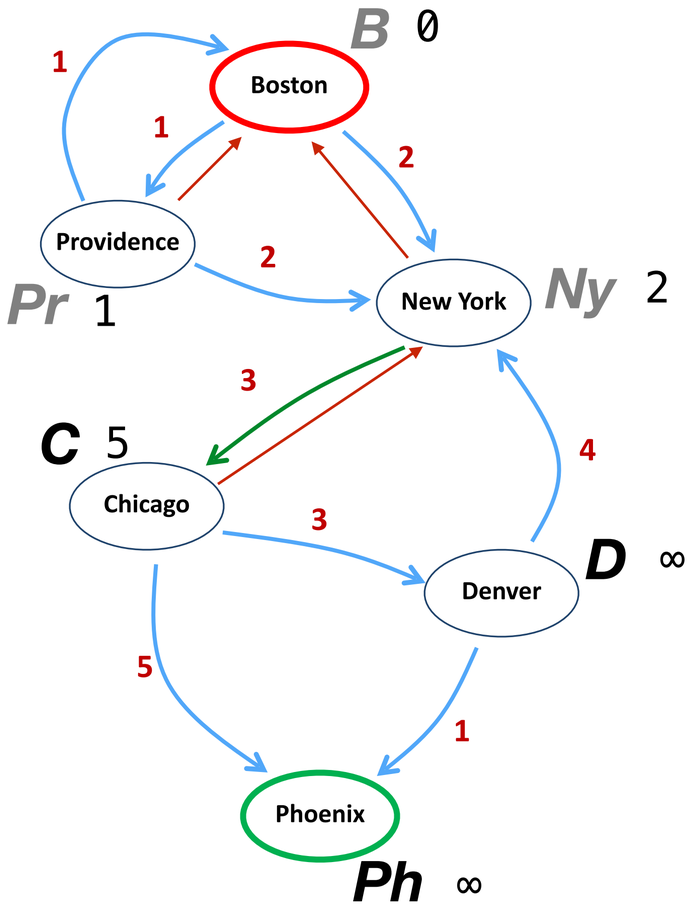
Pull Ny from the Queue



Recall: Quick introduction to Dijkstra's algorithm

Example

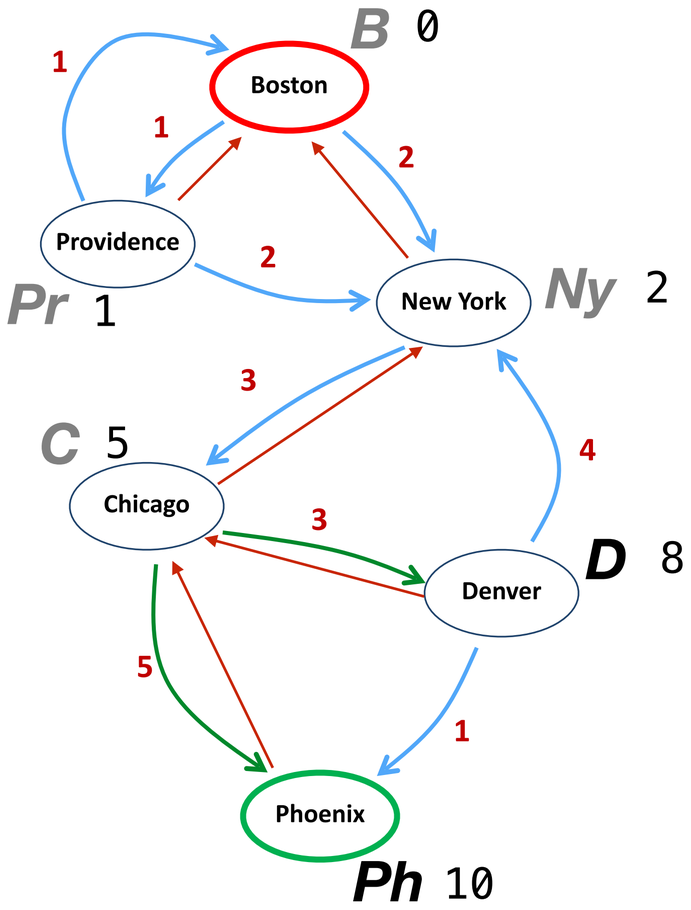
Pull Ny from the Queue
Consider updating cost to get to C



Recall: Quick introduction to Dijkstra's algorithm

Example

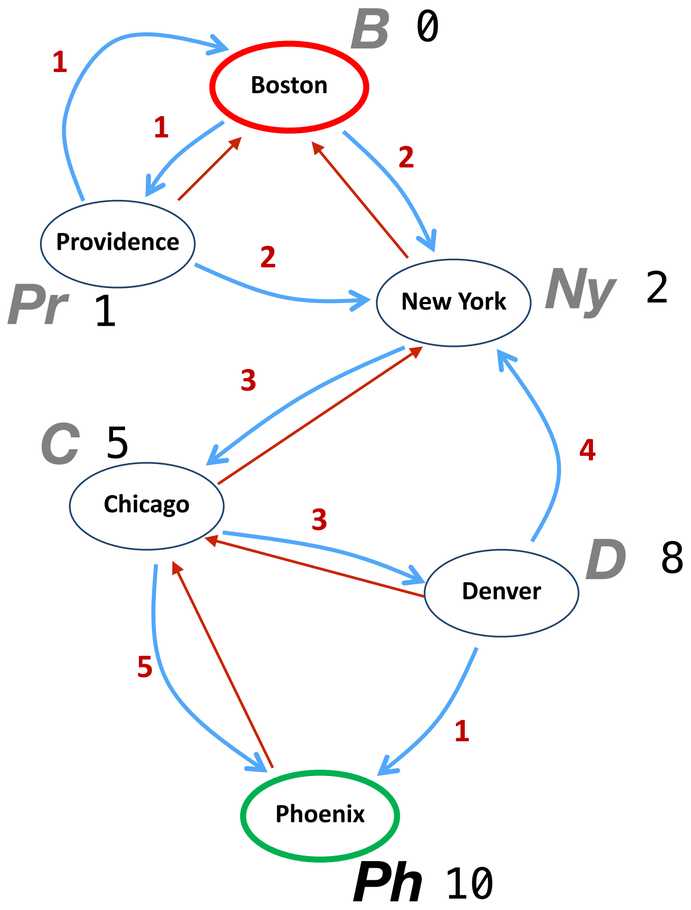
Pull C from the Queue
Consider updating cost to get to D and Ph



Recall: Quick introduction to Dijkstra's algorithm

Example

Pull D from the Queue



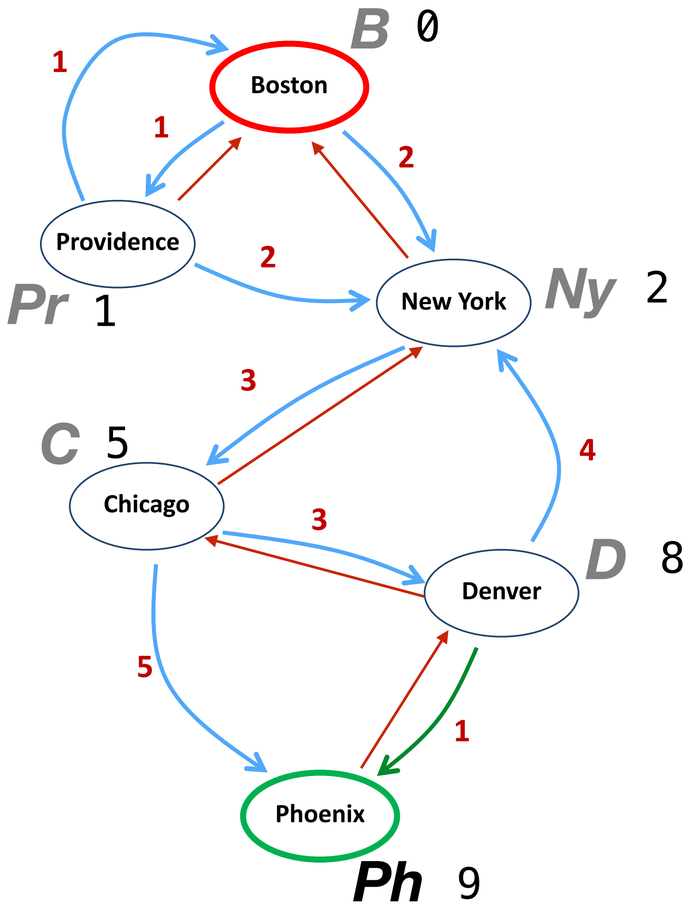
Recall: Quick introduction to Dijkstra's algorithm

Example

Pull D from the Queue
Consider updating cost to get to Ph and Ny

(Notice that the cost to get to Ph goes down. Before the update: Cost to get to Ph through C is 10. After the update: Cost to get to Ph through D is 9. Hence, we update the cost to get to Ph and also update the Parent of P to D.)

(Notice that the cost to get to Ny is more when we go through D (12 units) as opposed to going directly from B (2 units). So the parent for Ny does not change.)

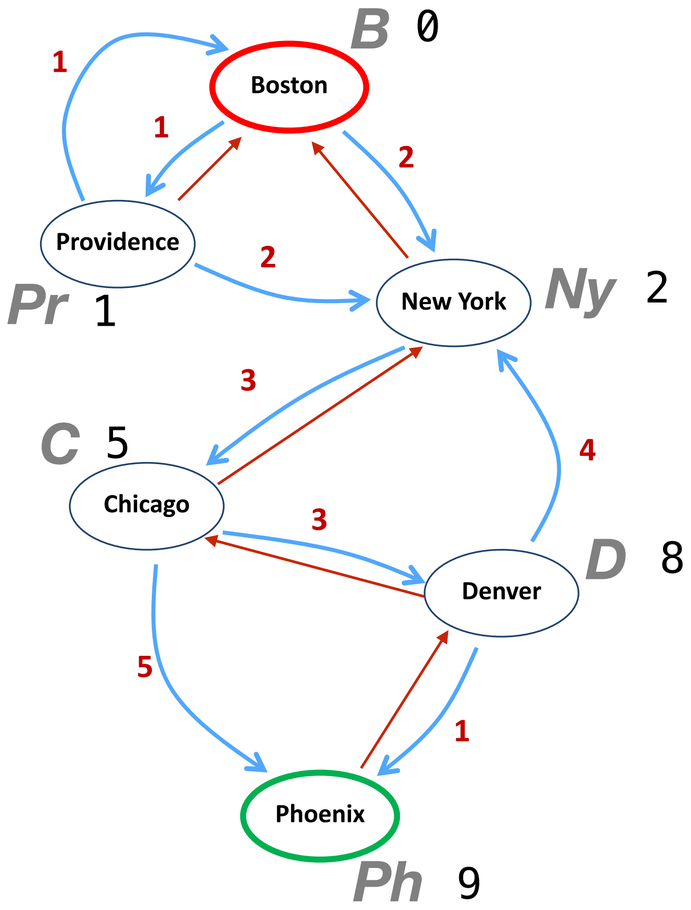


Recall: Quick introduction to Dijkstra's algorithm

Example

Pull Ph from the Queue

(It has no outgoing edges. So there is nothing to do.)



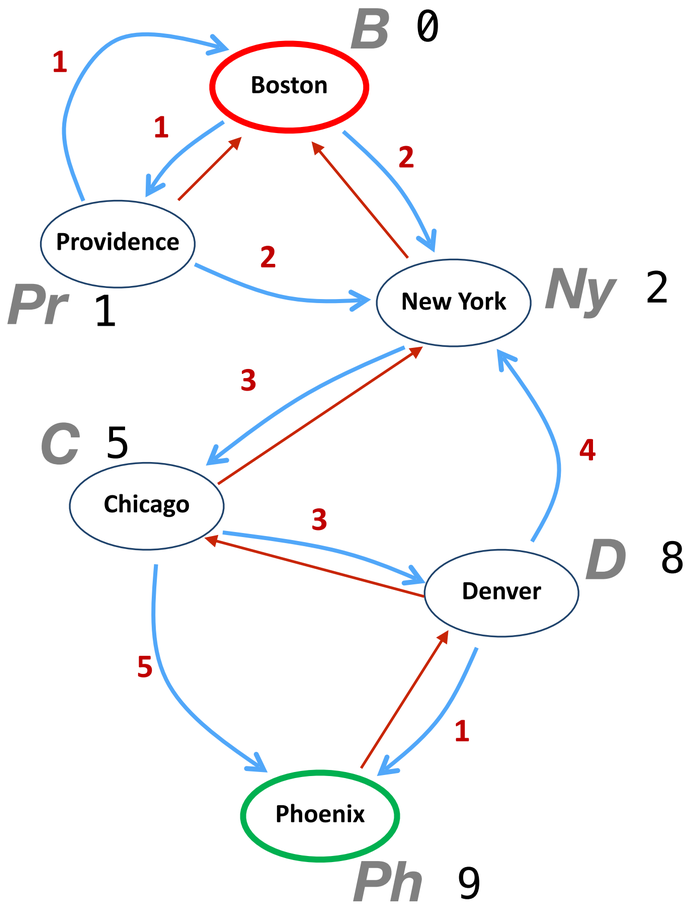
Recall: Quick introduction to Dijkstra's algorithm

Example

The queue is empty. We are done!

Notice that we computed the shortest path to go from vertex B to all other vertices, including Ph.

To find the the shortest path from B to any vertex v, you can start from v and then trace back the trajectory following the parents

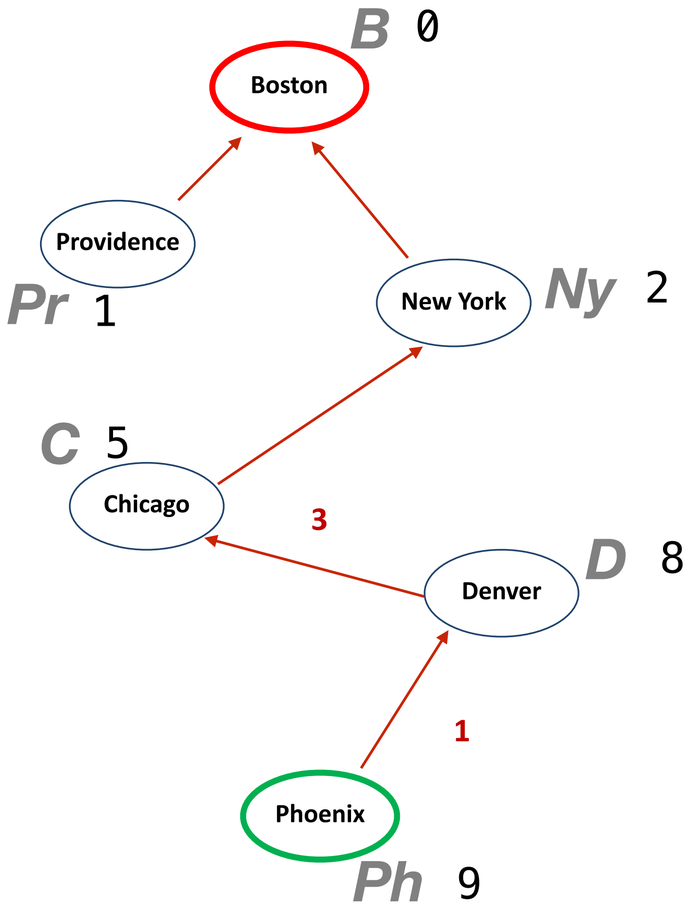


Recall: Quick introduction to Dijkstra's algorithm

Example

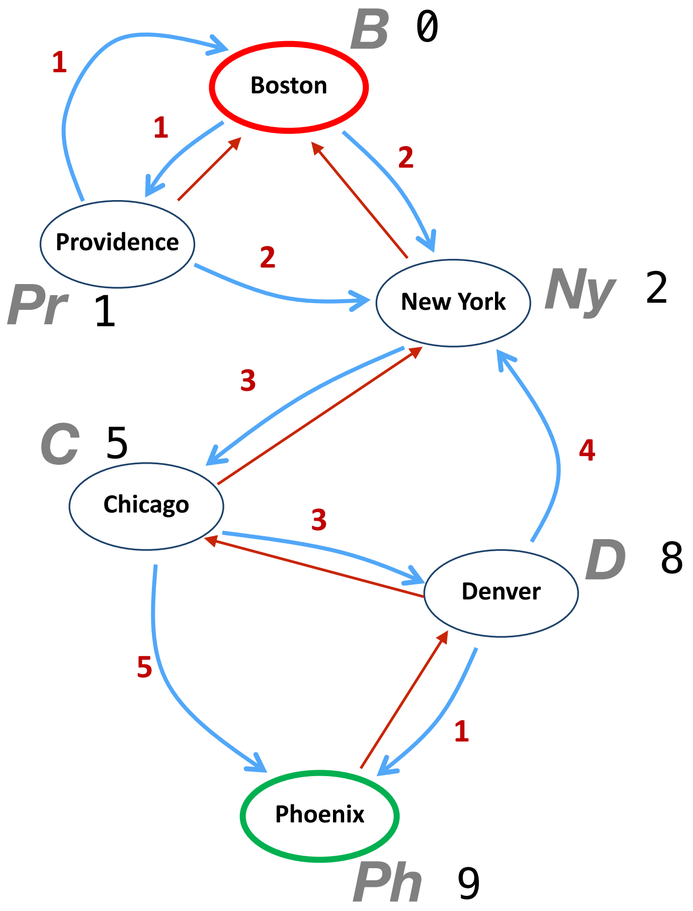
Notice that the Dijsktra's algorithm ultimately turns the graph into a tree that encodes.

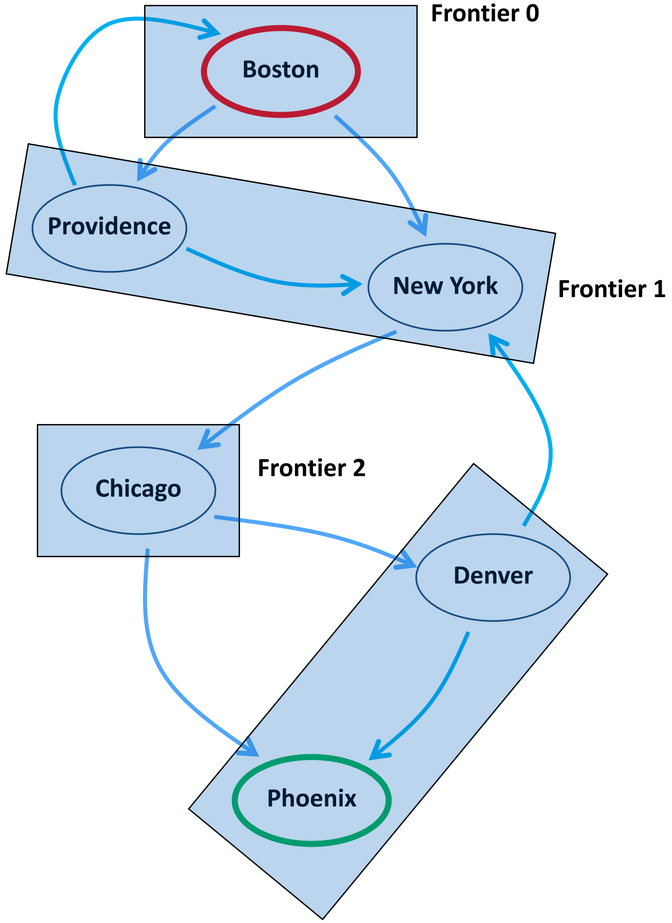
This structure has no directed cycles!



Recap: Shortest Paths and Dijsktra's algorithm

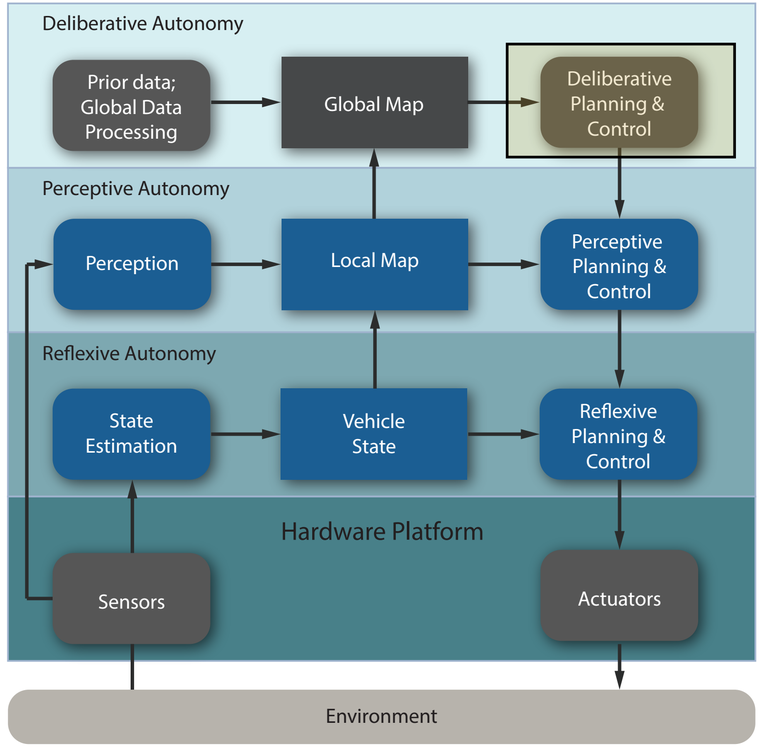
- Shortest path computation is a generalized version of Breadth first search, as it searches the "frontier" in increasing cost to go.

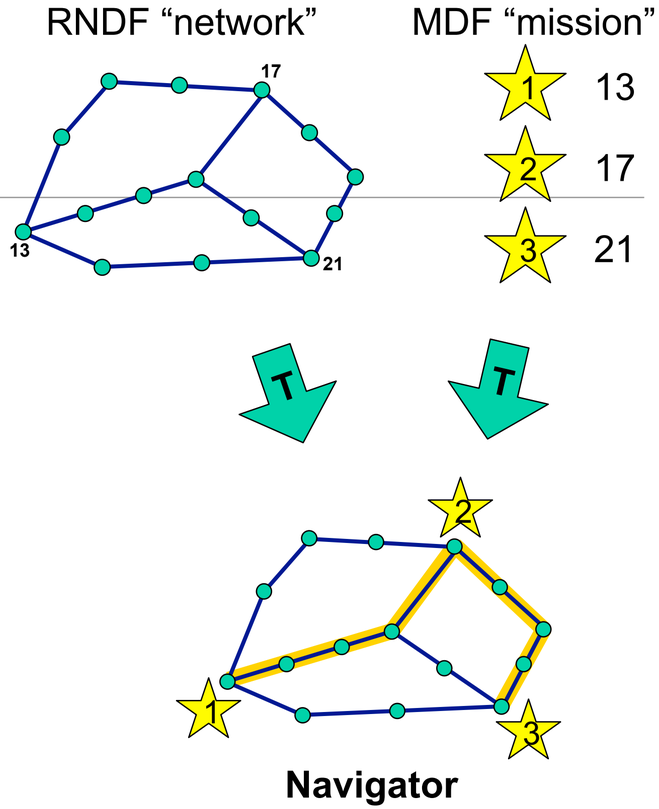




Recap: Shortest Paths and Dijsktra's algorithm

- Shortest path computation is embedded and essential in numerous problems in autonomy and robotics (as well as far outside of these fields).

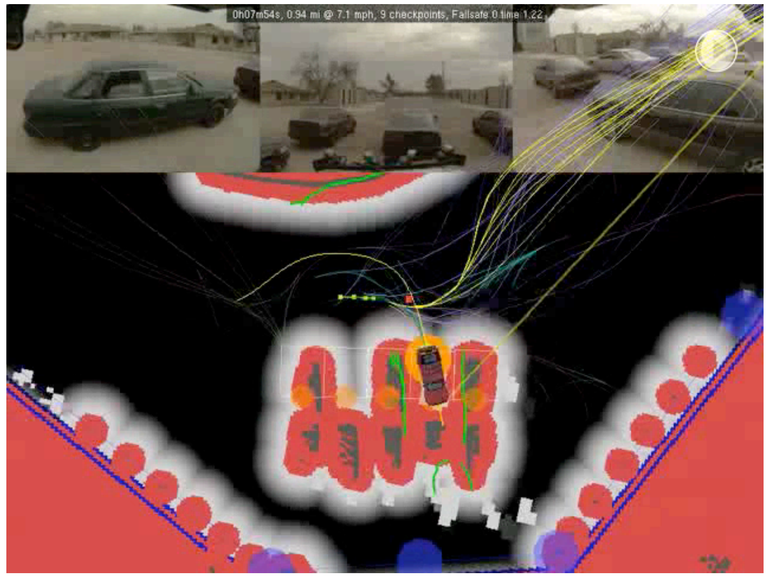


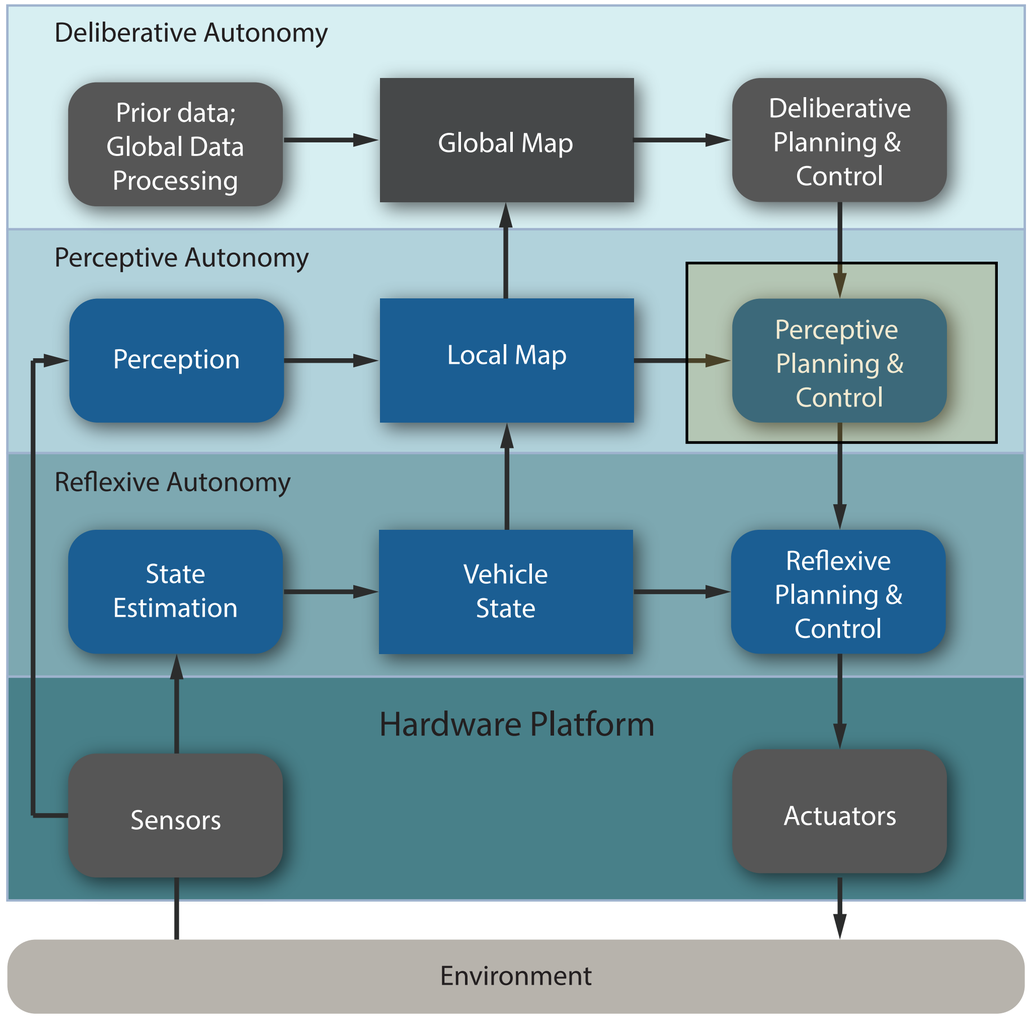


Sampling-based Planning Algorithms

Planning on a Local Map

- The “deliberative planning” process plans across discrete spaces or tasks. There are a finite amount of options (vertices) that we need to consider.
- When we think about “perceptive planning” we need to consider “continuous spaces”. The vehicle can be anywhere on the map.

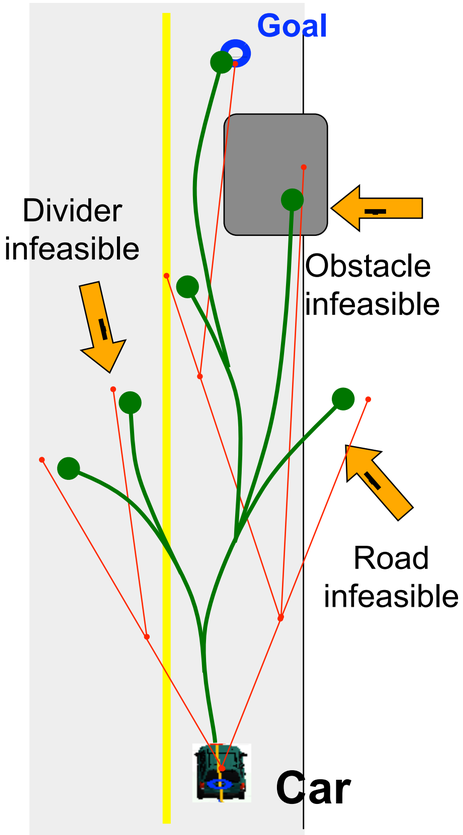




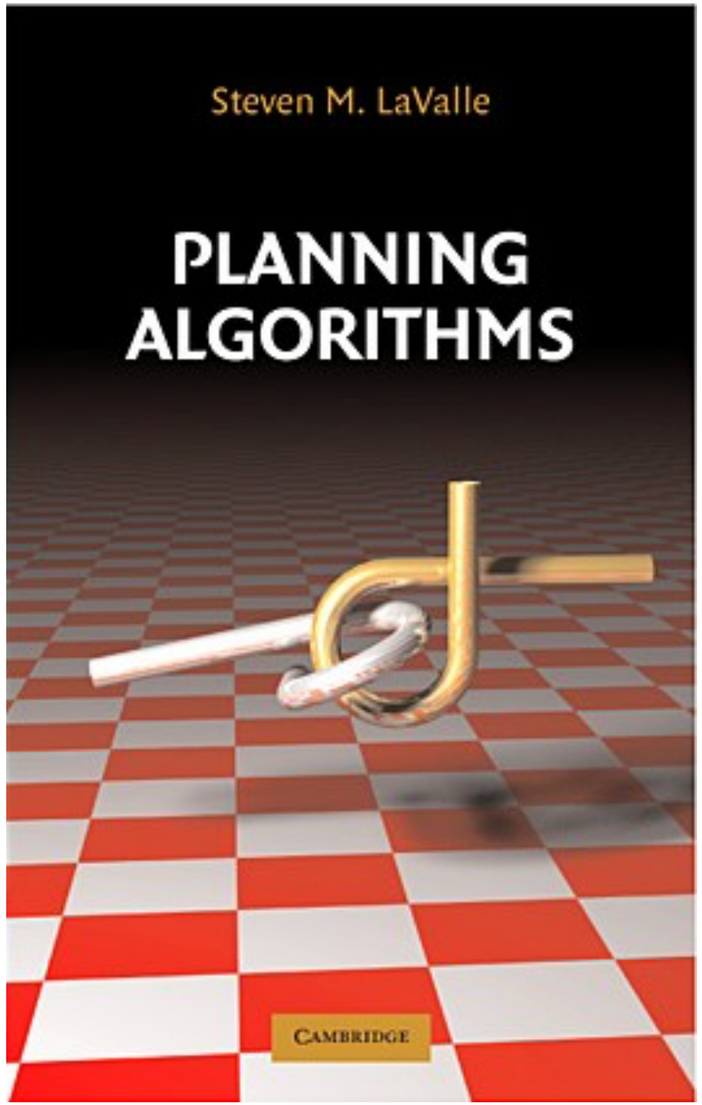
Searching continuous spaces

- The key question for this session will be regarding how we search continuous spaces!
- Recall our animation from the first lecture (on the right)
  - Lets formalize this set of randomized algorithms.
- Note that there are several other algorithms that can operate on continuous spaces. We are going to look at sampling-based algorithms today, mostly because they are relatively easier to implement

Rapidly-exploring Random Tree (RRT) for Motion Planning



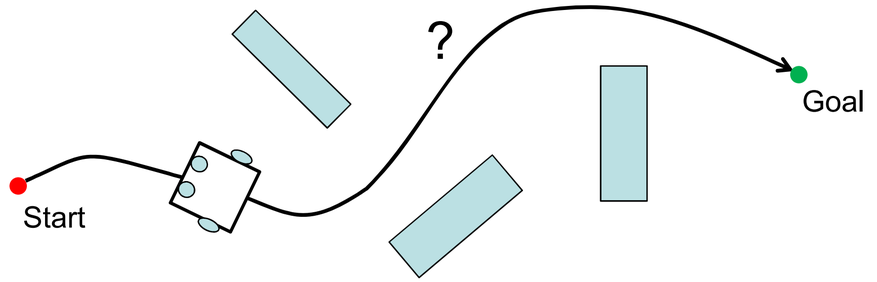
More on Motion Planning Algorithms



- Extensive coverage of the literature (until 2007).
- Includes problem definitions, computational hardness results, and algorithmic approaches.
- The book is freely available online at: http://planning.cs.uiuc.edu

Intuition for Motion Planning

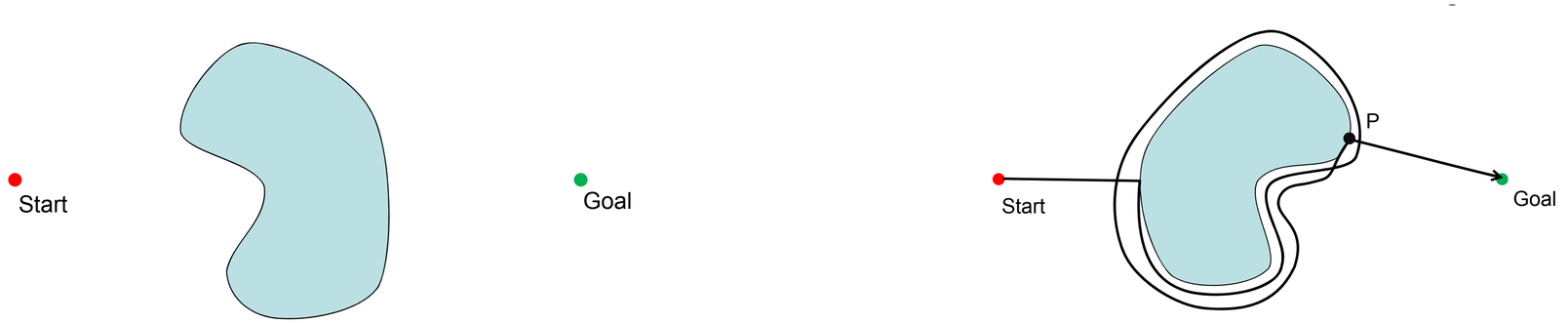
- What series of motions will get the robot to the goal?



- Are there cases in which no such motion exists?

Algorithm 1: The Bug Algorithm

- **Algorithm:**
  - Repeatedly advance toward goal
  - Upon encountering an obstacle: Completely circumnavigate it counter-clockwise, then depart from point P that minimizes distance to goal.



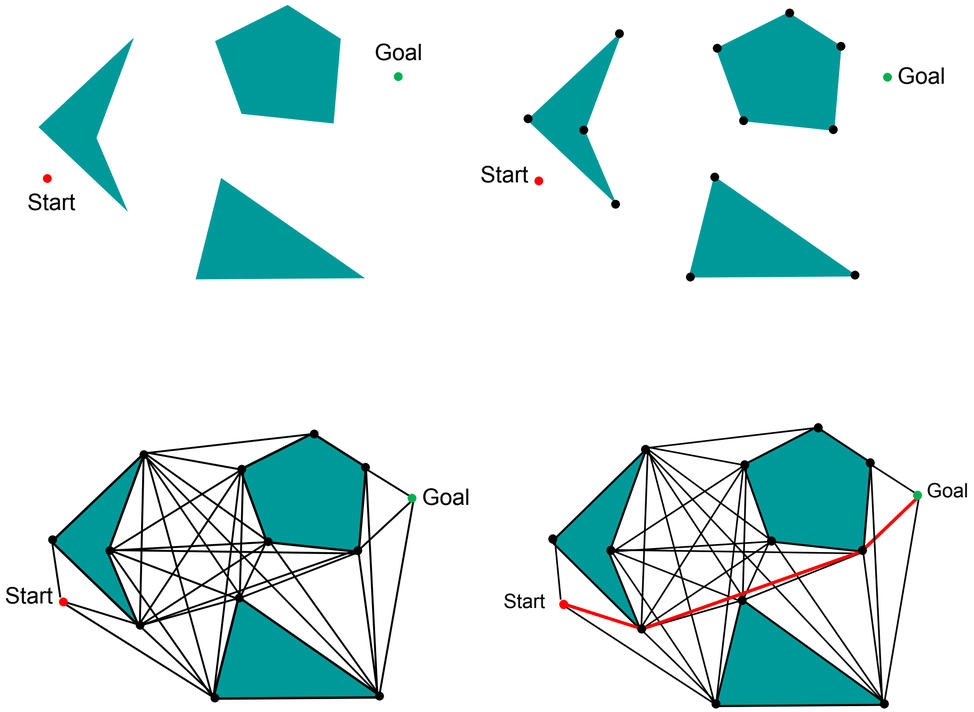
Are there any problems with this approach?

MIT AEROASTRO

Algorithm 2: Visibility Graphs

- Visibility graphs can be utilized in 2D.

- **Algorithm:**
  - Create a graph:
    - Vertex set: all vertices on polygonal obstacles
    - Edge set: all vertex pairs that can be connected by a straight collision free path
  - Return the shortest path on this graph.



This algorithm is returns the shortest path!

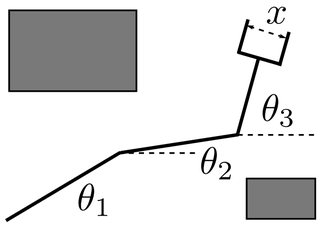
MIT AEROASTRO

Reducing rigid bodies to a point:
A brief introduction to Configuration Spaces

- We will focus on point robots (rather than rigid body), without loss of any generality.
- **Configuration Space:** Set of all configuration of the robot.
- *What are the configuration spaces of the following robots?*

For most mobile robots, the set of all robot poses will be the configuration space:

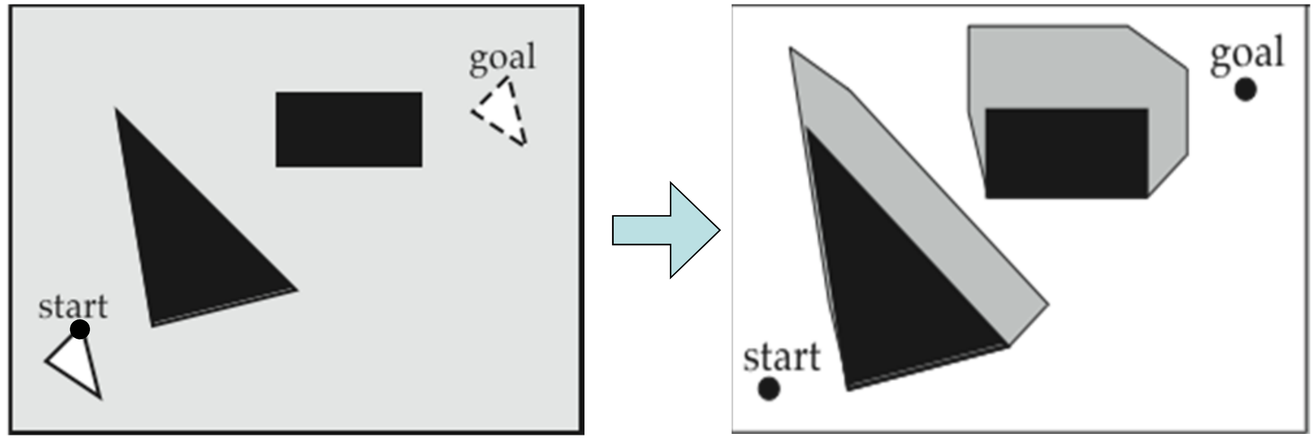
- *In a number of examples (e.g., manipulators) the configuration space may be more complex*



MIT AEROASTRO

Motion Planning in C-space

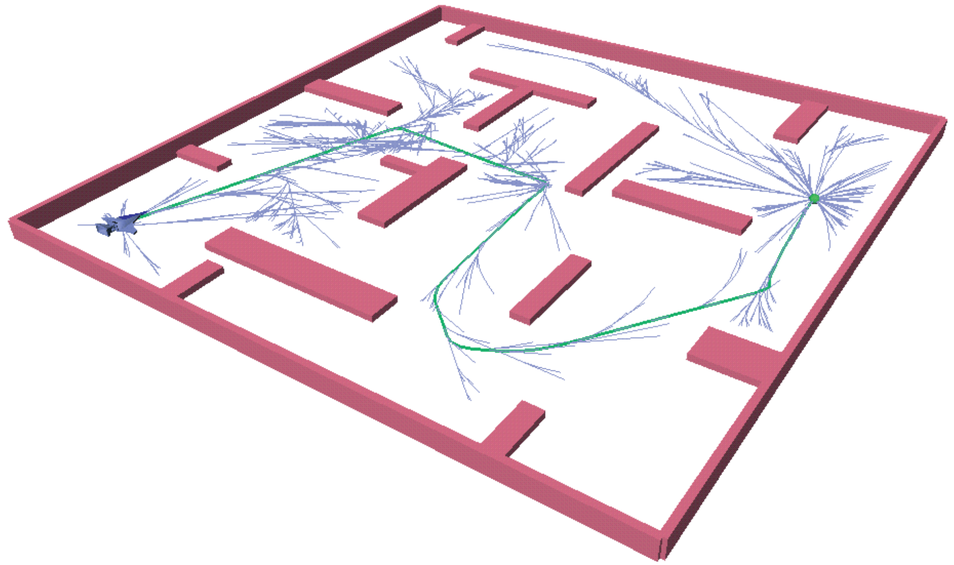
- Given robot and set of obstacles:
  - Compute C-space representation of obstacles
  - Find a (continuous) path from initial configuration to the goal configuration.
- The problem is simpler geometrically. But, we are not done.
  We still need to find a continuous path for a point robot.



- The idea of configuration space is attributed to Prof. Lozano-Perez.

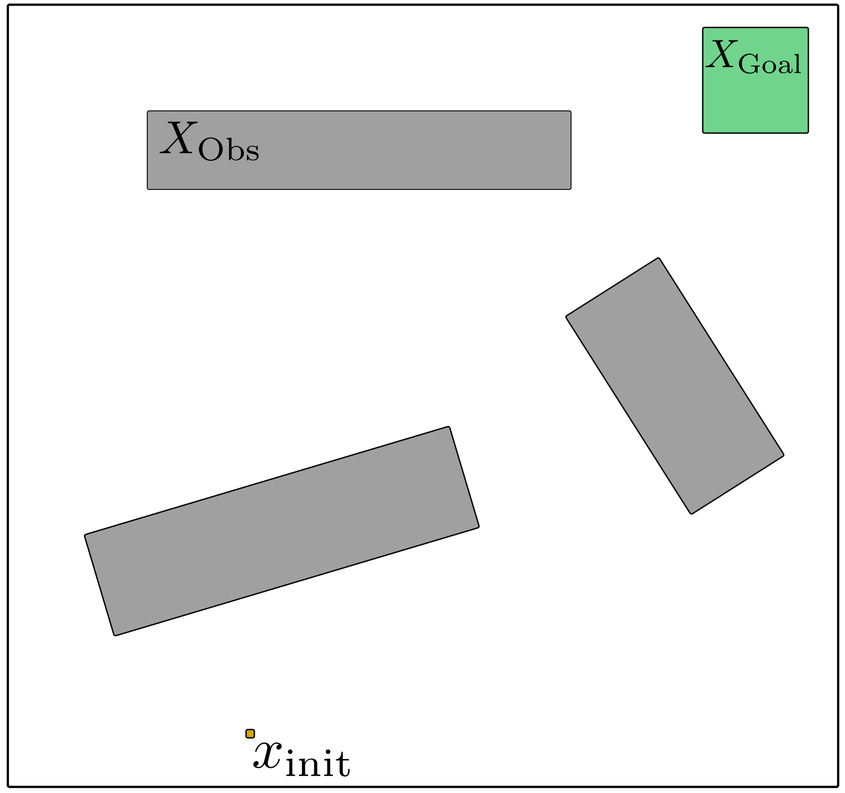
Sampling-based Methods

- **Key idea:** Sample the configuration space and connect samples with trajectories to infer the connectivity of the free space.
- Probabilistic RoadMap (PRM) and the Rapidly-exploring Random Trees (RRT) are two of the widely used variants.



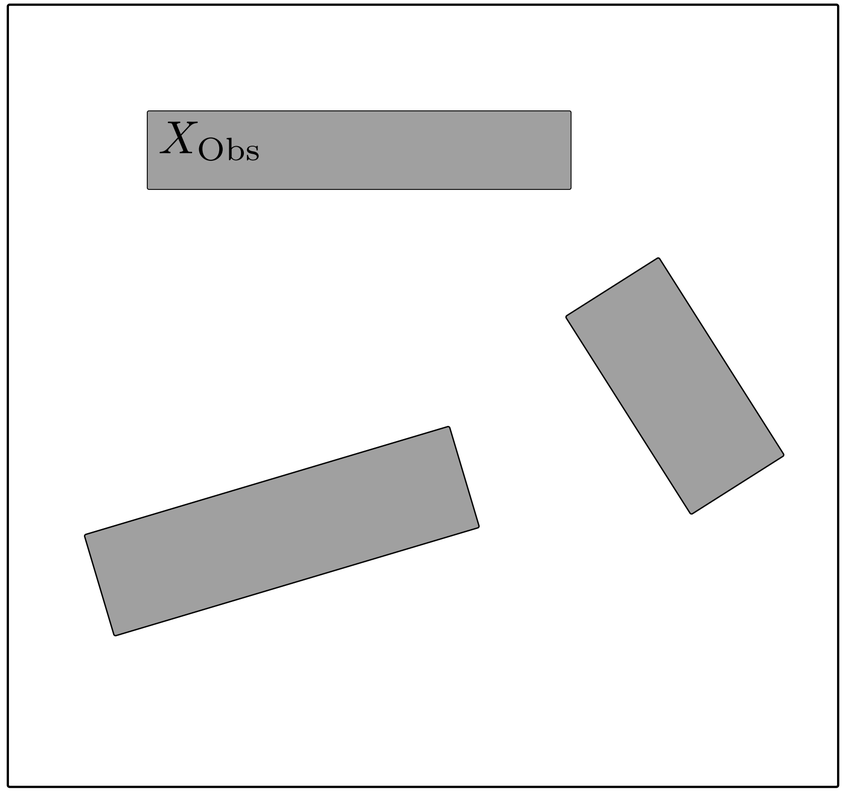
Probabilistic RoadMap (PRM) Algorithm

1. Sample N points uniformly at random from $\mathcal{C}$;
2. Connect each pair with a straight trajectory;
3. Delete all vertices and edges that lies on the obstacle set $\mathcal{C}_{\text{obs}}$;
4. Return the remaining roadmap $\mathcal{G} = (V, E)$.



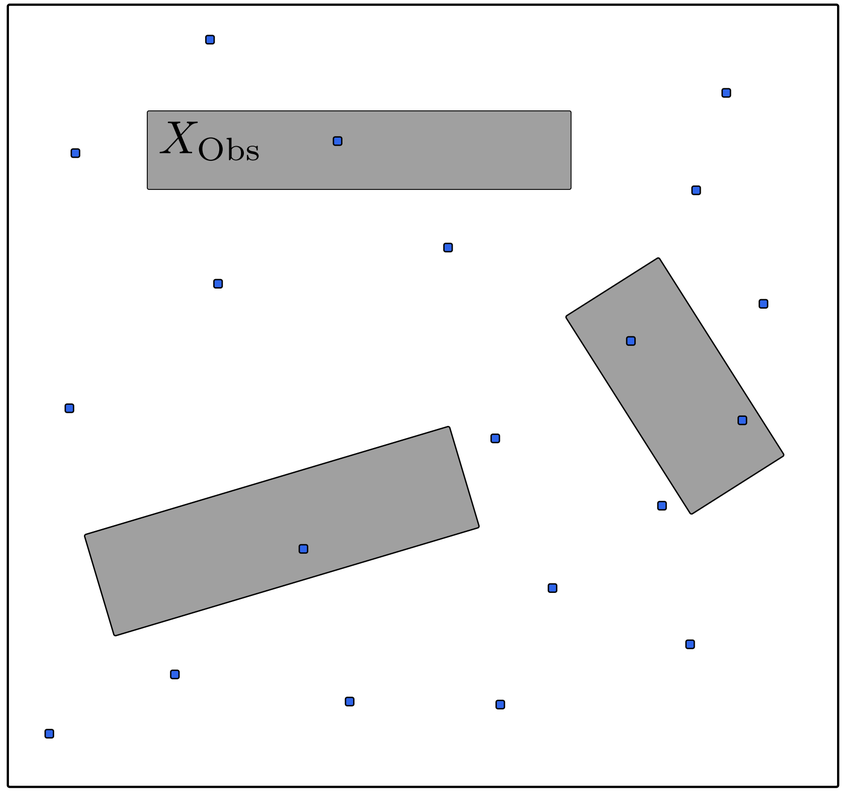
Probabilistic RoadMap (PRM) Algorithm

1. Sample N points uniformly at random from $\mathcal{C}$;
2. Connect each pair with a straight trajectory;
3. Delete all vertices and edges that lies on the obstacle set $\mathcal{C}_{\text{obs}}$;
4. Return the remaining roadmap $\mathcal{G} = (V, E)$.



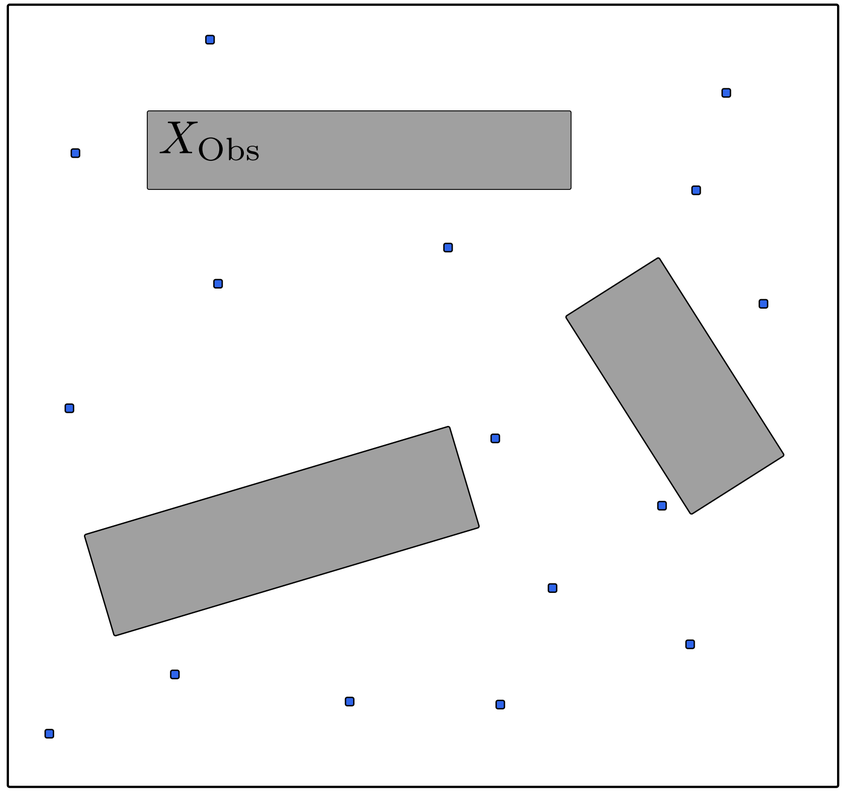
Probabilistic RoadMap (PRM) Algorithm

1. Sample N points uniformly at random from $\mathcal{C}$;
2. Connect each pair with a straight trajectory;
3. Delete all vertices and edges that lies on the obstacle set $\mathcal{C}_{\text{obs}}$;
4. Return the remaining roadmap $\mathcal{G} = (V, E)$.



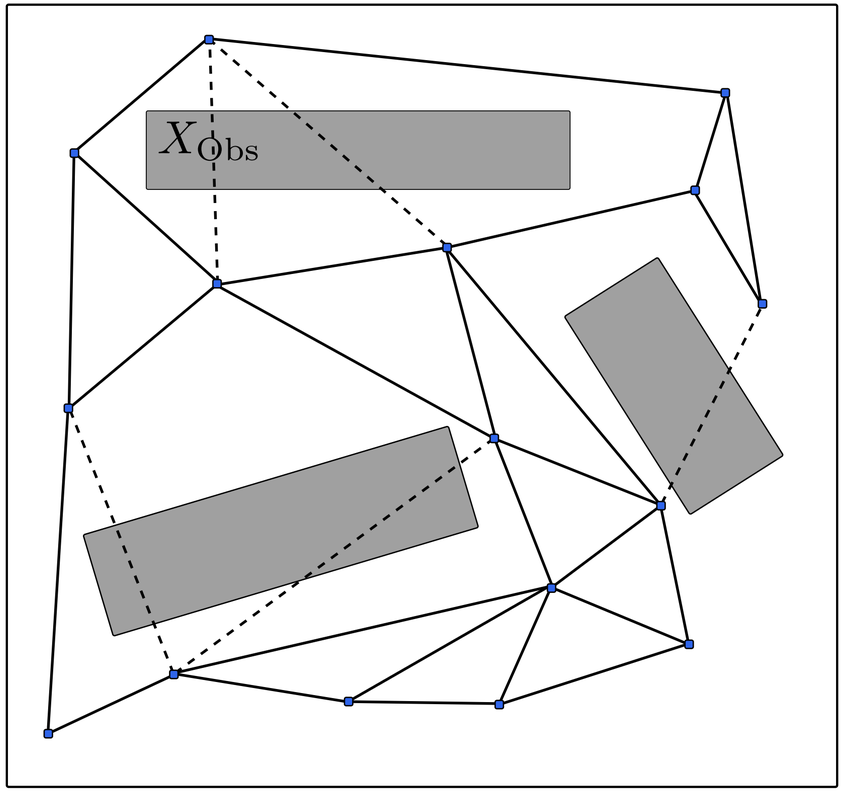
Probabilistic RoadMap (PRM) Algorithm

1. Sample N points uniformly at random from $\mathcal{C}$;
2. Connect each pair with a straight trajectory;
3. Delete all vertices and edges that lies on the obstacle set $\mathcal{C}_{\text{obs}}$;
4. Return the remaining roadmap $\mathcal{G} = (V, E)$.



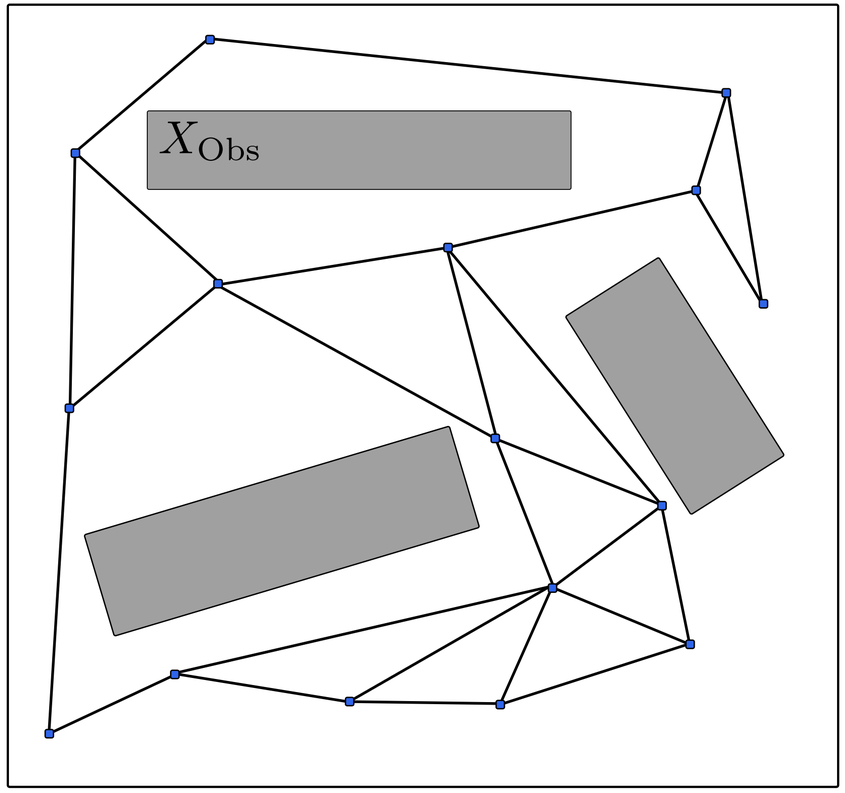
Probabilistic RoadMap (PRM) Algorithm

1. Sample N points uniformly at random from $\mathcal{C}$;
2. Connect each pair with a straight trajectory;
3. Delete all vertices and edges that lies on the obstacle set $\mathcal{C}_{\text{obs}}$;
4. Return the remaining roadmap $\mathcal{G} = (V, E)$.



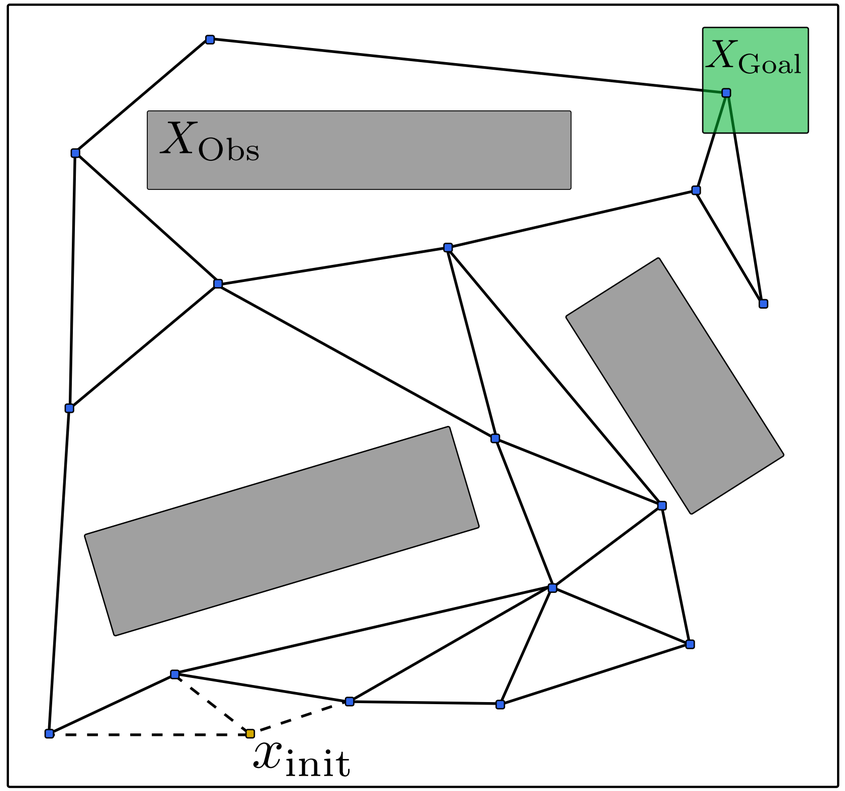
Probabilistic RoadMap (PRM) Algorithm

1. Sample N points uniformly at random from $\mathcal{C}$;
2. Connect each pair with a straight trajectory;
3. Delete all vertices and edges that lies on the obstacle set $\mathcal{C}_{\text{obs}}$;
4. Return the remaining roadmap $\mathcal{G} = (V, E)$.



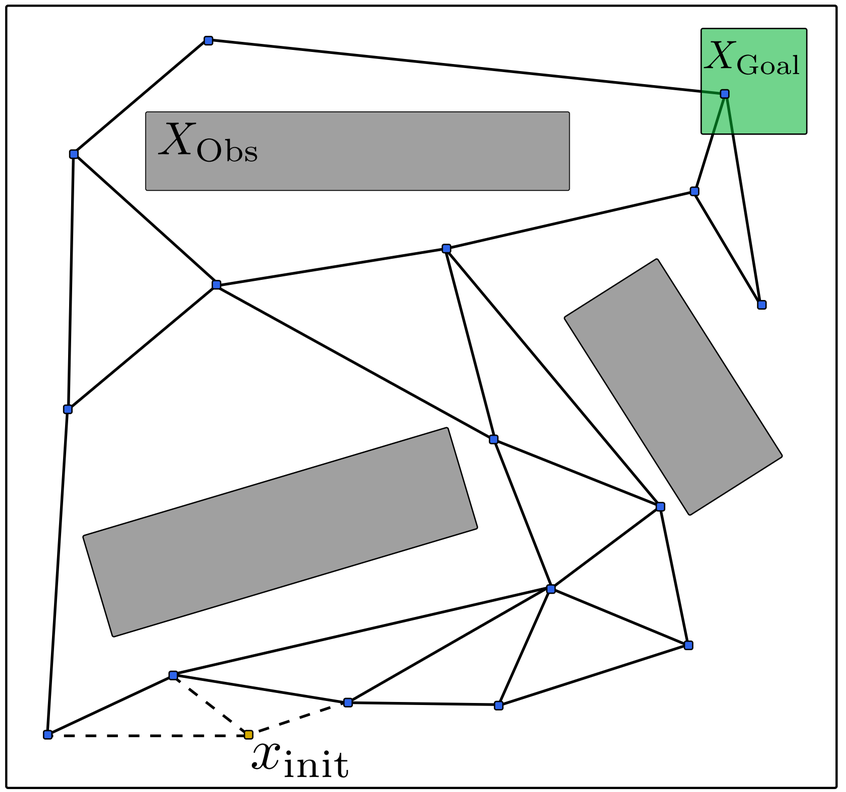
Probabilistic RoadMap (PRM) Algorithm

1. Sample N points uniformly at random from $\mathcal{C}$;
2. Connect each pair with a straight trajectory;
3. Delete all vertices and edges that lies on the obstacle set $\mathcal{C}_{\text{obs}}$;
4. Return the remaining roadmap $\mathcal{G} = (V, E)$.



What is Wrong with the Probabilistic RoadMap (PRM)?

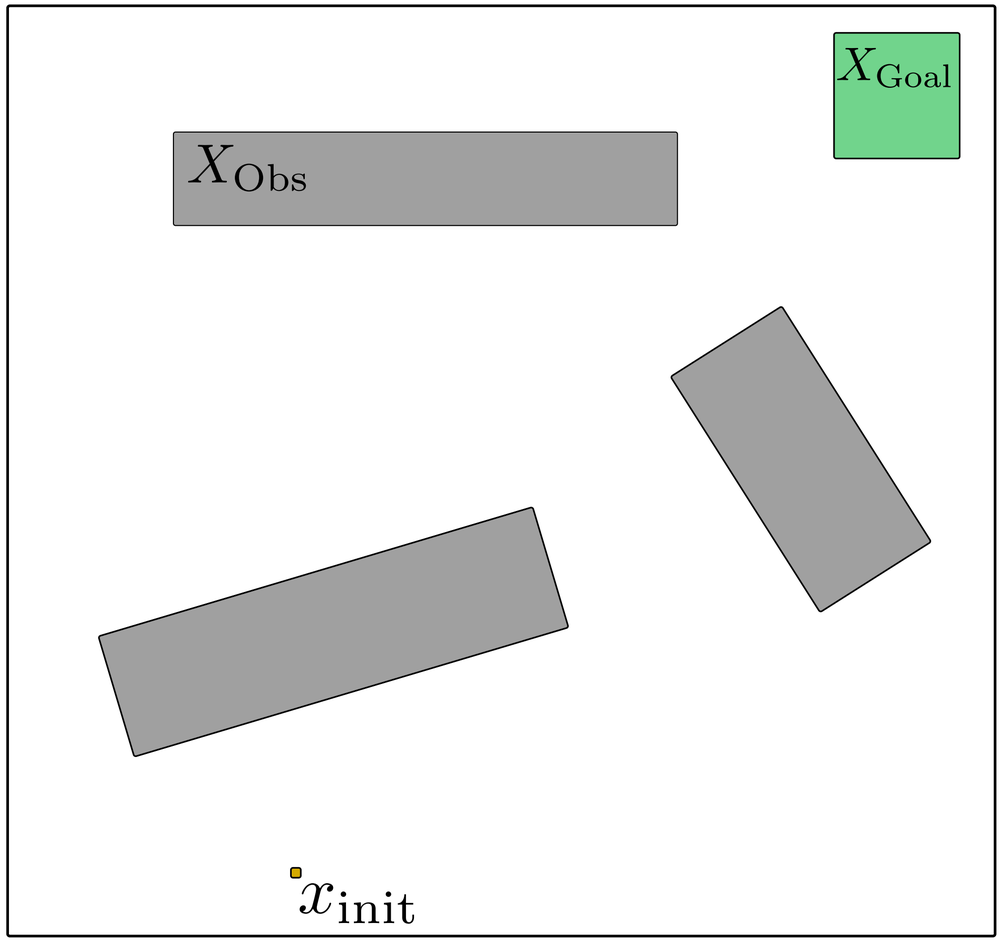
1. Sample N points uniformly at random from $\mathcal{C}$;
2. Connect each pair with a straight trajectory;
3. Delete all vertices and edges that lies on the obstacle set $\mathcal{C}_{\text{obs}}$;
4. Return the remaining roadmap $\mathcal{G} = (V, E)$.



How do we generate dynamically-feasible trajectories that connect two states?

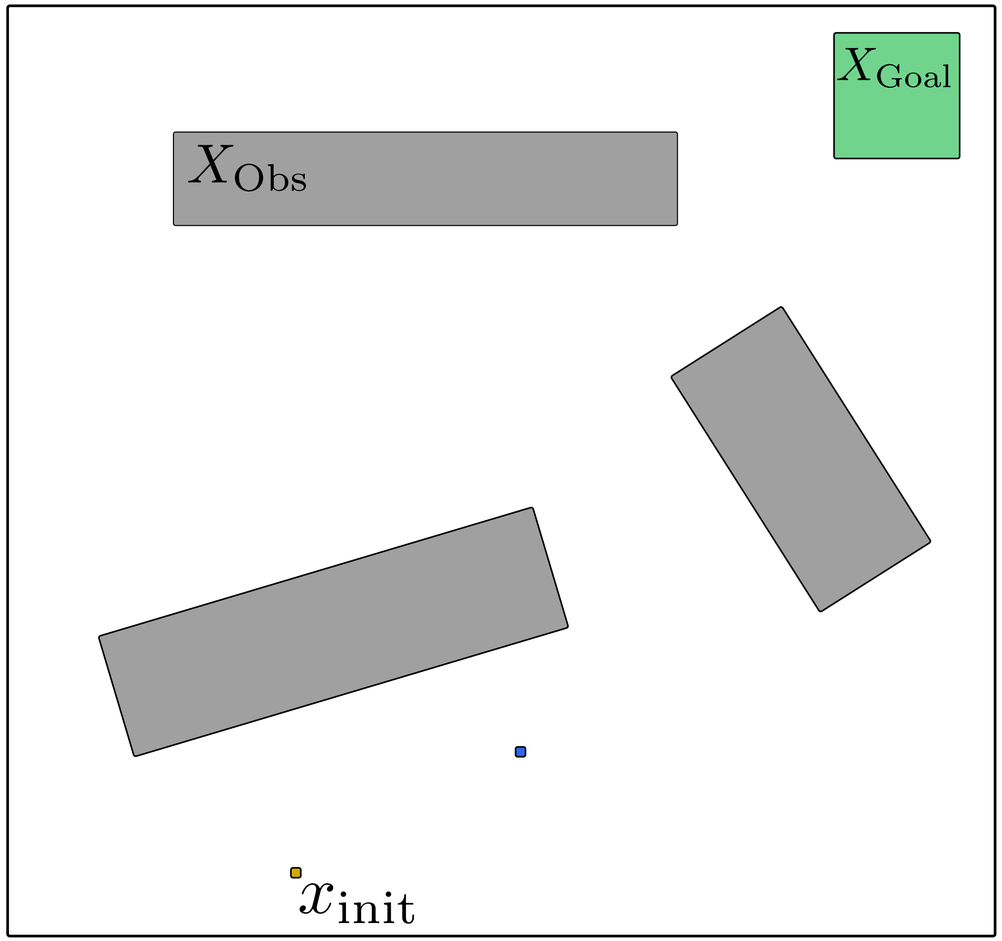
Rapidly-exploring Random Tree (RRT) Algorithm

1. For $i = 1, \ldots, N$ do
2. $\quad z_{\text{rand}} \leftarrow \texttt{Sample}(X)$;
3. $\quad z_{\text{nearest}} \leftarrow \texttt{Nearest}(V, z_{\text{rand}})$;
4. $\quad x_{\text{new}} \leftarrow \texttt{Steer}(z_{\text{nearest}}, z_{\text{rand}})$;
5. $\quad$ If $\texttt{CollisionFree}(x_{\text{new}})$ then
6. $\quad\quad z_{\text{new}} \leftarrow x_{\text{new}}(T)$;
7. $\quad\quad V \leftarrow V \cup \{z_{\text{new}}\}$;
8. $\quad\quad E \leftarrow E \cup \{(z_{\text{nearest}}, z_{\text{new}})\}$;
9. $\quad$ End If
10. End For



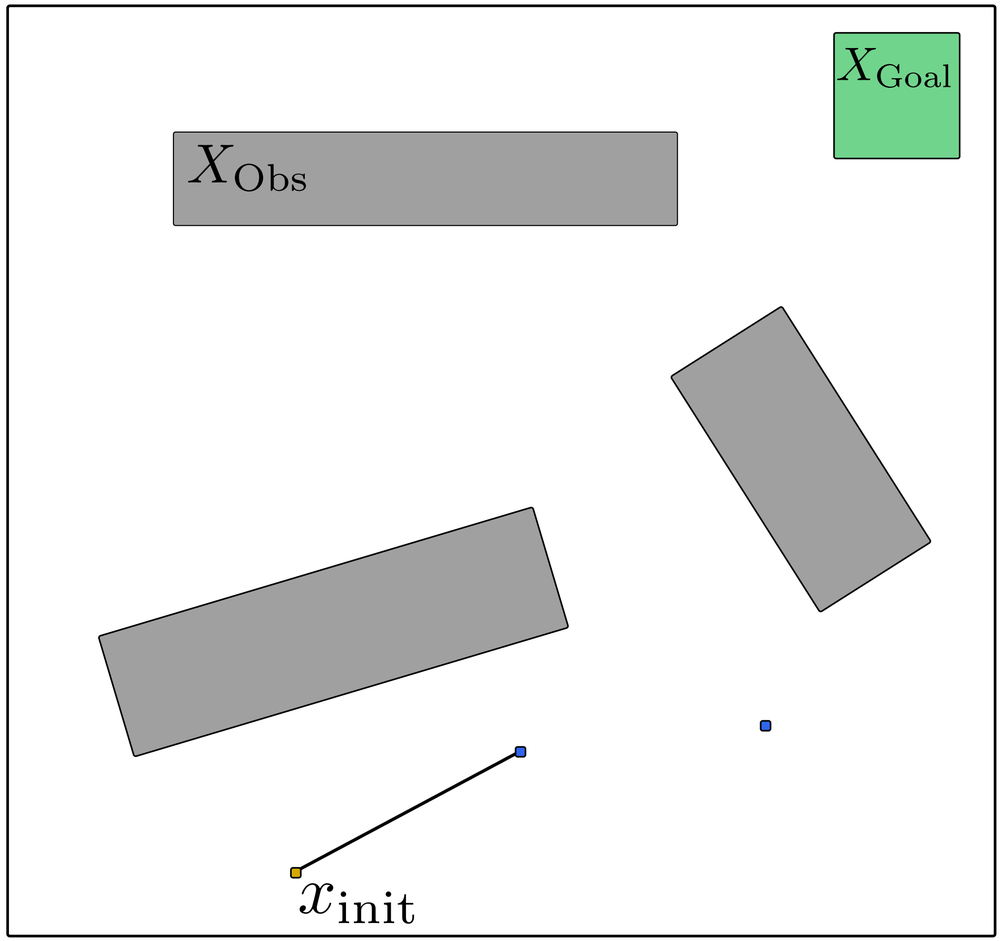
Rapidly-exploring Random Tree Algorithm

1. For $i = 1, \ldots, N$ do
2. $\quad z_{\text{rand}} \leftarrow \texttt{Sample}(X)$;
3. $\quad z_{\text{nearest}} \leftarrow \texttt{Nearest}(V, z_{\text{rand}})$;
4. $\quad x_{\text{new}} \leftarrow \texttt{Steer}(z_{\text{nearest}}, z_{\text{rand}})$;
5. $\quad$ If $\texttt{CollisionFree}(x_{\text{new}})$ then
6. $\quad\quad z_{\text{new}} \leftarrow x_{\text{new}}(T)$;
7. $\quad\quad V \leftarrow V \cup \{z_{\text{new}}\}$;
8. $\quad\quad E \leftarrow E \cup \{(z_{\text{nearest}}, z_{\text{new}})\}$;
9. $\quad$ End If
10. End For



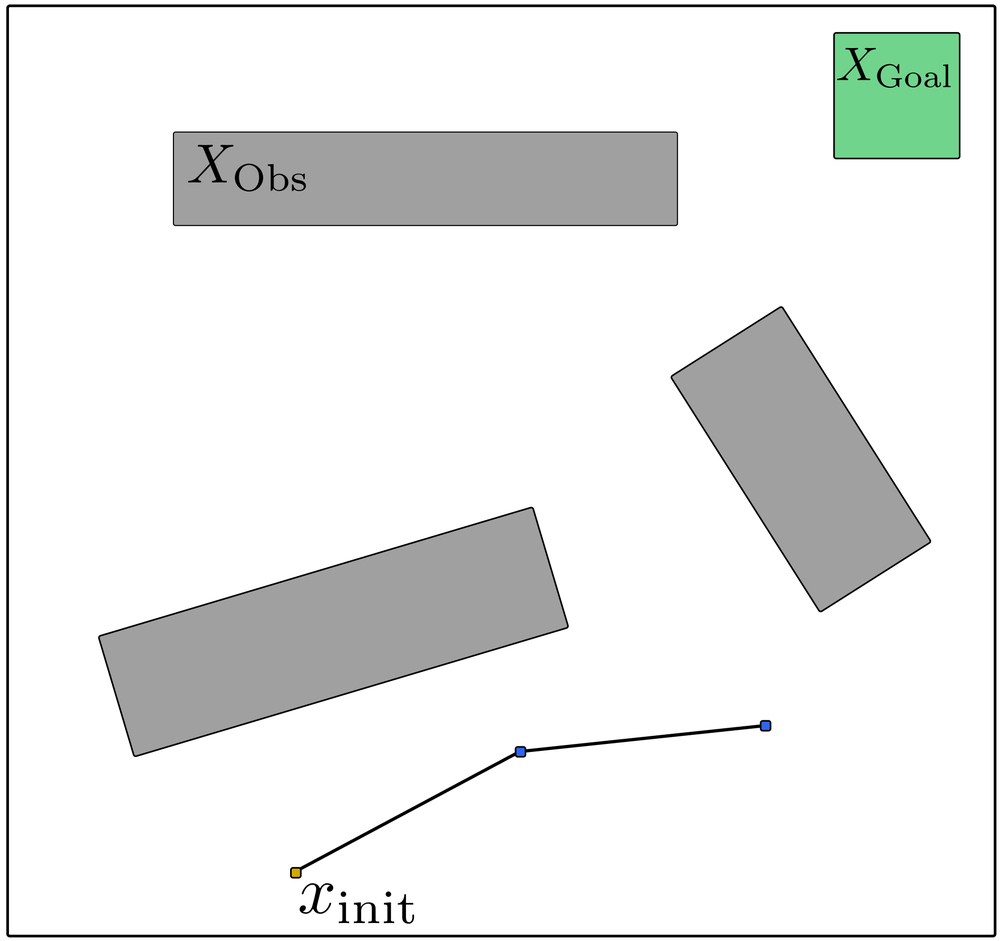
Rapidly-exploring Random Tree Algorithm

1. For $i = 1, \ldots, N$ do
2. $\quad z_{\text{rand}} \leftarrow \texttt{Sample}(X)$;
3. $\quad z_{\text{nearest}} \leftarrow \texttt{Nearest}(V, z_{\text{rand}})$;
4. $\quad x_{\text{new}} \leftarrow \texttt{Steer}(z_{\text{nearest}}, z_{\text{rand}})$;
5. $\quad$ If $\texttt{CollisionFree}(x_{\text{new}})$ then
6. $\quad\quad z_{\text{new}} \leftarrow x_{\text{new}}(T)$;
7. $\quad\quad V \leftarrow V \cup \{z_{\text{new}}\}$;
8. $\quad\quad E \leftarrow E \cup \{(z_{\text{nearest}}, z_{\text{new}})\}$;
9. $\quad$ End If
10. End For



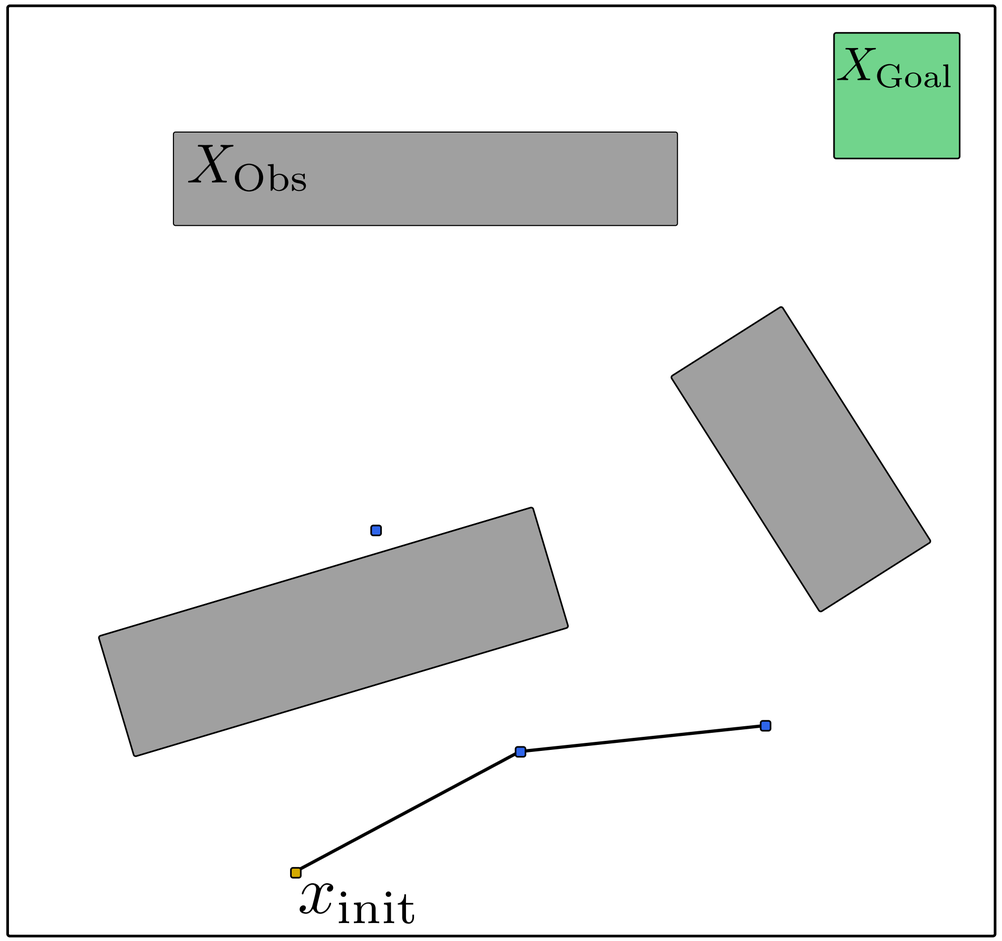
Rapidly-exploring Random Tree Algorithm

1. For $i = 1, \ldots, N$ do
2. $\quad z_{\text{rand}} \leftarrow \texttt{Sample}(X)$;
3. $\quad z_{\text{nearest}} \leftarrow \texttt{Nearest}(V, z_{\text{rand}})$;
4. $\quad x_{\text{new}} \leftarrow \texttt{Steer}(z_{\text{nearest}}, z_{\text{rand}})$;
5. $\quad$ If $\texttt{CollisionFree}(x_{\text{new}})$ then
6. $\quad\quad z_{\text{new}} \leftarrow x_{\text{new}}(T)$;
7. $\quad\quad V \leftarrow V \cup \{z_{\text{new}}\}$;
8. $\quad\quad E \leftarrow E \cup \{(z_{\text{nearest}}, z_{\text{new}})\}$;
9. $\quad$ End If
10. End For



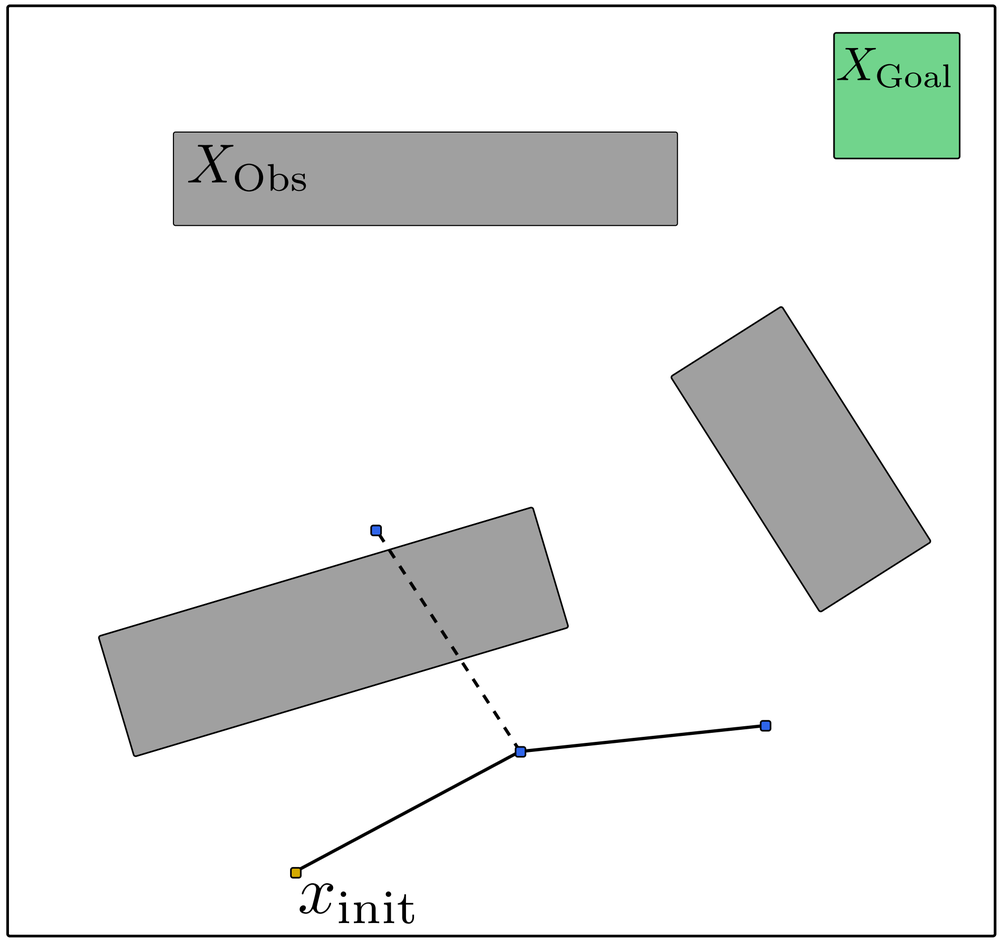
Rapidly-exploring Random Tree Algorithm

1. For $i = 1, \ldots, N$ do
2. $\quad z_{\text{rand}} \leftarrow \texttt{Sample}(X)$;
3. $\quad z_{\text{nearest}} \leftarrow \texttt{Nearest}(V, z_{\text{rand}})$;
4. $\quad x_{\text{new}} \leftarrow \texttt{Steer}(z_{\text{nearest}}, z_{\text{rand}})$;
5. $\quad$ If $\texttt{CollisionFree}(x_{\text{new}})$ then
6. $\quad\quad z_{\text{new}} \leftarrow x_{\text{new}}(T)$;
7. $\quad\quad V \leftarrow V \cup \{z_{\text{new}}\}$;
8. $\quad\quad E \leftarrow E \cup \{(z_{\text{nearest}}, z_{\text{new}})\}$;
9. $\quad$ End If
10. End For



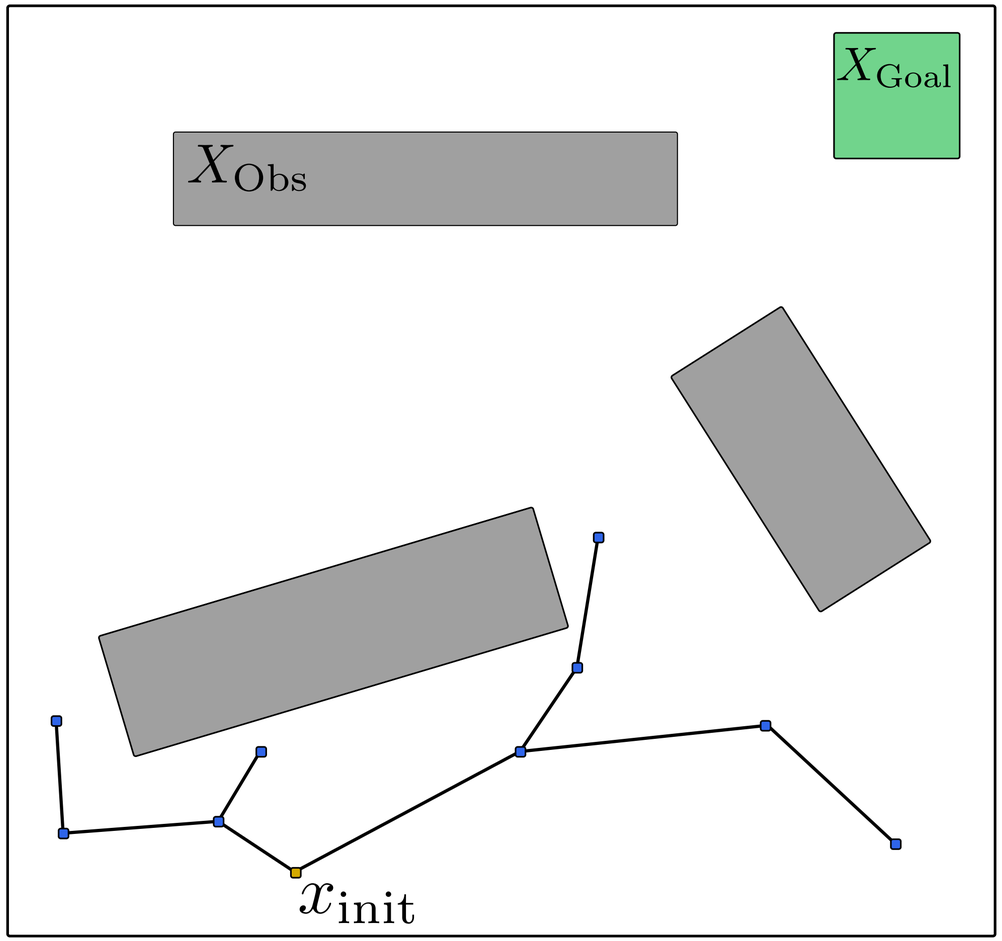
Rapidly-exploring Random Tree Algorithm

1. For $i = 1, \ldots, N$ do
2. $\quad z_{\text{rand}} \leftarrow \texttt{Sample}(X)$;
3. $\quad z_{\text{nearest}} \leftarrow \texttt{Nearest}(V, z_{\text{rand}})$;
4. $\quad x_{\text{new}} \leftarrow \texttt{Steer}(z_{\text{nearest}}, z_{\text{rand}})$;
5. $\quad$ If $\texttt{CollisionFree}(x_{\text{new}})$ then
6. $\quad\quad z_{\text{new}} \leftarrow x_{\text{new}}(T)$;
7. $\quad\quad V \leftarrow V \cup \{z_{\text{new}}\}$;
8. $\quad\quad E \leftarrow E \cup \{(z_{\text{nearest}}, z_{\text{new}})\}$;
9. $\quad$ End If
10. End For



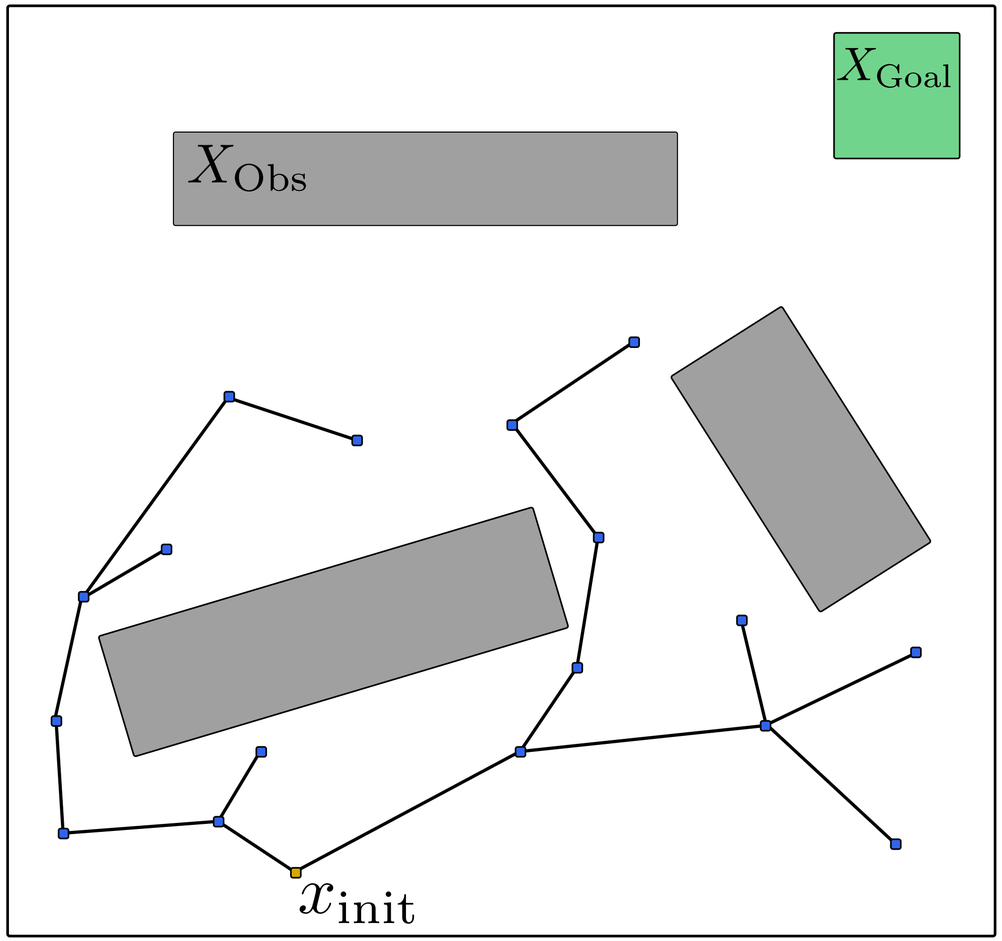
Rapidly-exploring Random Tree Algorithm

1. For $i = 1, \ldots, N$ do
2. $\quad z_{\text{rand}} \leftarrow \texttt{Sample}(X)$;
3. $\quad z_{\text{nearest}} \leftarrow \texttt{Nearest}(V, z_{\text{rand}})$;
4. $\quad x_{\text{new}} \leftarrow \texttt{Steer}(z_{\text{nearest}}, z_{\text{rand}})$;
5. $\quad$ If $\texttt{CollisionFree}(x_{\text{new}})$ then
6. $\quad\quad z_{\text{new}} \leftarrow x_{\text{new}}(T)$;
7. $\quad\quad V \leftarrow V \cup \{z_{\text{new}}\}$;
8. $\quad\quad E \leftarrow E \cup \{(z_{\text{nearest}}, z_{\text{new}})\}$;
9. $\quad$ End If
10. End For



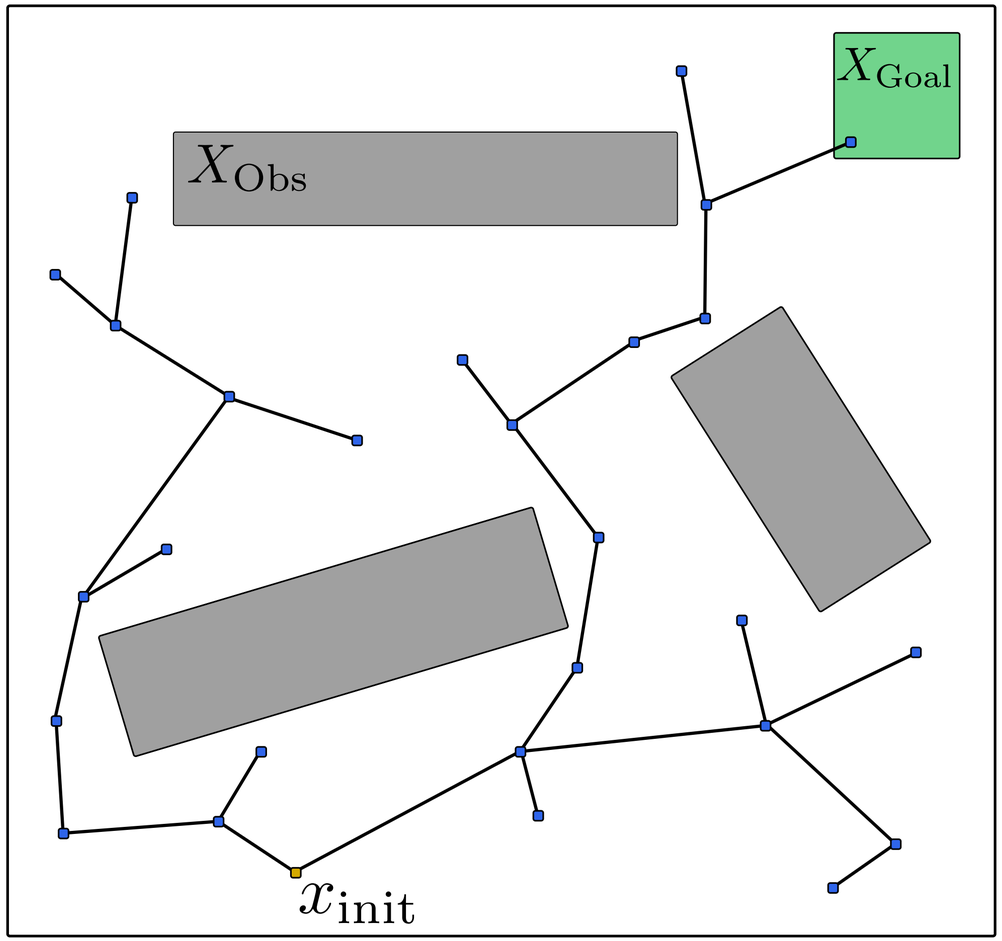
Rapidly-exploring Random Tree Algorithm

1. For $i = 1, \ldots, N$ do
2. $\quad z_{\text{rand}} \leftarrow \texttt{Sample}(X)$;
3. $\quad z_{\text{nearest}} \leftarrow \texttt{Nearest}(V, z_{\text{rand}})$;
4. $\quad x_{\text{new}} \leftarrow \texttt{Steer}(z_{\text{nearest}}, z_{\text{rand}})$;
5. $\quad$ If $\texttt{CollisionFree}(x_{\text{new}})$ then
6. $\quad\quad z_{\text{new}} \leftarrow x_{\text{new}}(T)$;
7. $\quad\quad V \leftarrow V \cup \{z_{\text{new}}\}$;
8. $\quad\quad E \leftarrow E \cup \{(z_{\text{nearest}}, z_{\text{new}})\}$;
9. $\quad$ End If
10. End For



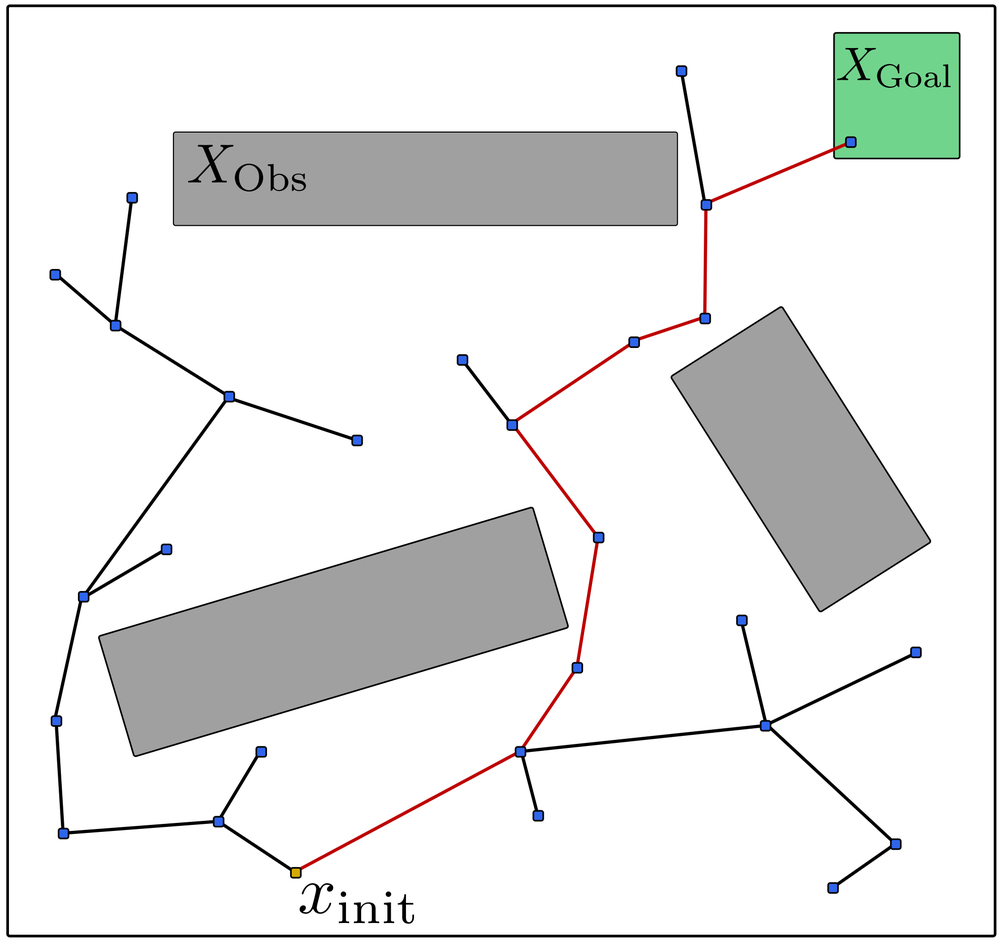
Rapidly-exploring Random Tree Algorithm

1. For $i = 1, \ldots, N$ do
2. $\quad z_{\text{rand}} \leftarrow \texttt{Sample}(X)$;
3. $\quad z_{\text{nearest}} \leftarrow \texttt{Nearest}(V, z_{\text{rand}})$;
4. $\quad x_{\text{new}} \leftarrow \texttt{Steer}(z_{\text{nearest}}, z_{\text{rand}})$;
5. $\quad$ If $\texttt{CollisionFree}(x_{\text{new}})$ then
6. $\quad\quad z_{\text{new}} \leftarrow x_{\text{new}}(T)$;
7. $\quad\quad V \leftarrow V \cup \{z_{\text{new}}\}$;
8. $\quad\quad E \leftarrow E \cup \{(z_{\text{nearest}}, z_{\text{new}})\}$;
9. $\quad$ End If
10. End For



Rapidly-exploring Random Tree Algorithm

1. For $i = 1, \ldots, N$ do
2. $\quad z_{\text{rand}} \leftarrow \texttt{Sample}(X)$;
3. $\quad z_{\text{nearest}} \leftarrow \texttt{Nearest}(V, z_{\text{rand}})$;
4. $\quad x_{\text{new}} \leftarrow \texttt{Steer}(z_{\text{nearest}}, z_{\text{rand}})$;
5. $\quad$ If $\texttt{CollisionFree}(x_{\text{new}})$ then
6. $\quad\quad z_{\text{new}} \leftarrow x_{\text{new}}(T)$;
7. $\quad\quad V \leftarrow V \cup \{z_{\text{new}}\}$;
8. $\quad\quad E \leftarrow E \cup \{(z_{\text{nearest}}, z_{\text{new}})\}$;
9. $\quad$ End If
10. End For

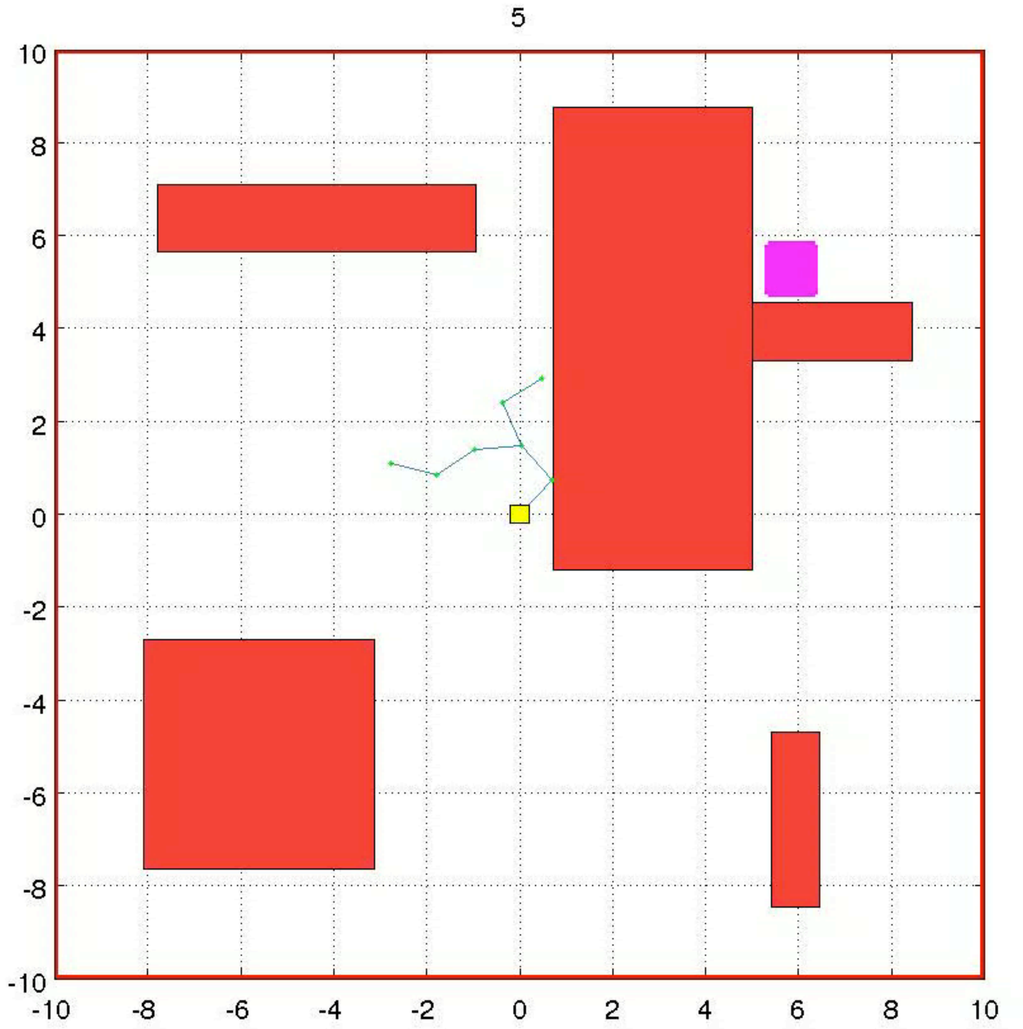


Important note:

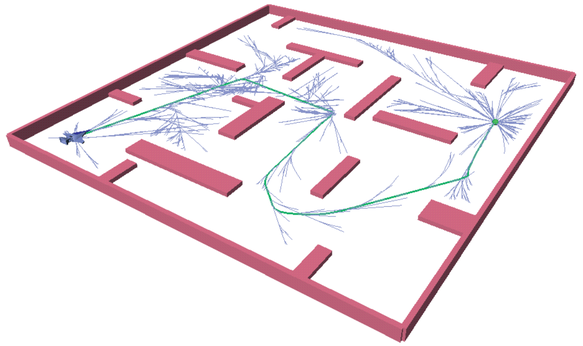
Sampling-based algorithms do not require constructing the configuration space!

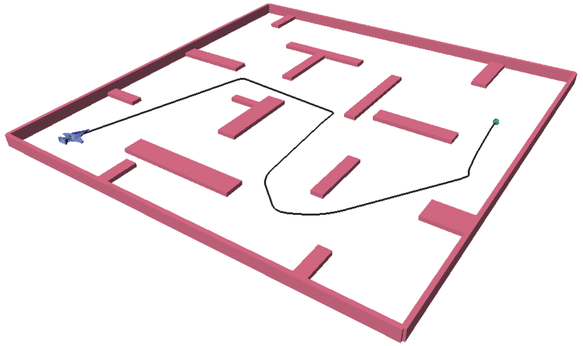
All we need is a “collision checking” function, which is often easy to implement.

RRT on a Simple Example



RRT Algorithm on More Challenging Examples



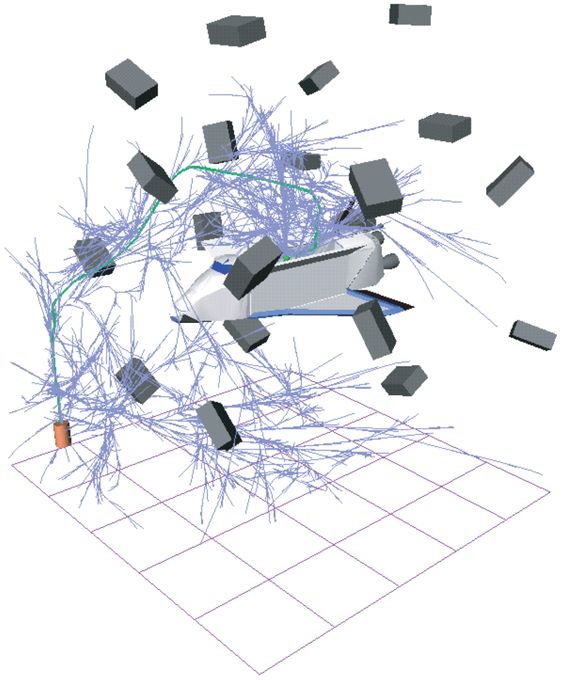


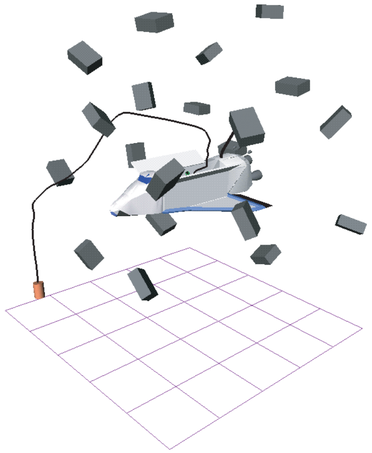
Hovercraft in a maze

Equations of motion:

$$\mathbf{x}(t) = \begin{pmatrix} \mathbf{p}(t) \\ \mathbf{q}(t) \\ \mathbf{v}(t) \\ \omega(t) \end{pmatrix} \qquad \dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) = \begin{pmatrix} \dot{\mathbf{p}}(t) \\ \dot{\mathbf{q}}(t) \\ \dot{\mathbf{v}}(t) \\ \dot{\omega}(t) \end{pmatrix}$$

$$= \begin{pmatrix} \mathbf{v}(t) \\ \frac{1}{2}\hat{\omega}(t) \cdot \mathbf{q}(t) \\ \frac{\mathbf{F}}{M} \\ R(t) I^{-1} R(t)^T \tau \end{pmatrix},$$





Spacecraft navigation through an obstacle field

[LaValle & Kuffner, "Randomized Kinodynamic Planning", International Journal of Robotics Research, 2001]

MIT AEROASTRO

Conclusions

- Motion planning problems are embedded and essential in almost all applications of autonomy
- The key aspect of the problem is its continuous configuration space.
- One effective way to solve continuous-space planning problems is sampling, which creates a discrete abstraction that we can search.