Autonomous Robotics
Winter 2024

Abhishek Gupta

TAs: Karthikeya Vemuri, Arnav Thareja

Marius Memmel, Yunchu Zhang

Slides borrowed from many sources – Sidd Srinivasa, Sanjiban Choudhury, Dieter Fox

Class Outline

State Estimation

- Robotic System Design
- Filtering
- Localization
- SLAM

Control

- Feedback Control
- PID Control
- MPC
- LQR

Planning

- Search
- Heuristic Search
- Motion Planning
- Lazy Search

Learning

- Imitation Learning
- Policy Gradient
- Actor-Critic
- Model-Based RL

Logistics

- HW 3 due Feb 14
- Paper commentaries due Wednesday 2/14

Lecture Outline

Why is the problem hard?

↓

A recipe for solving motion planning problems

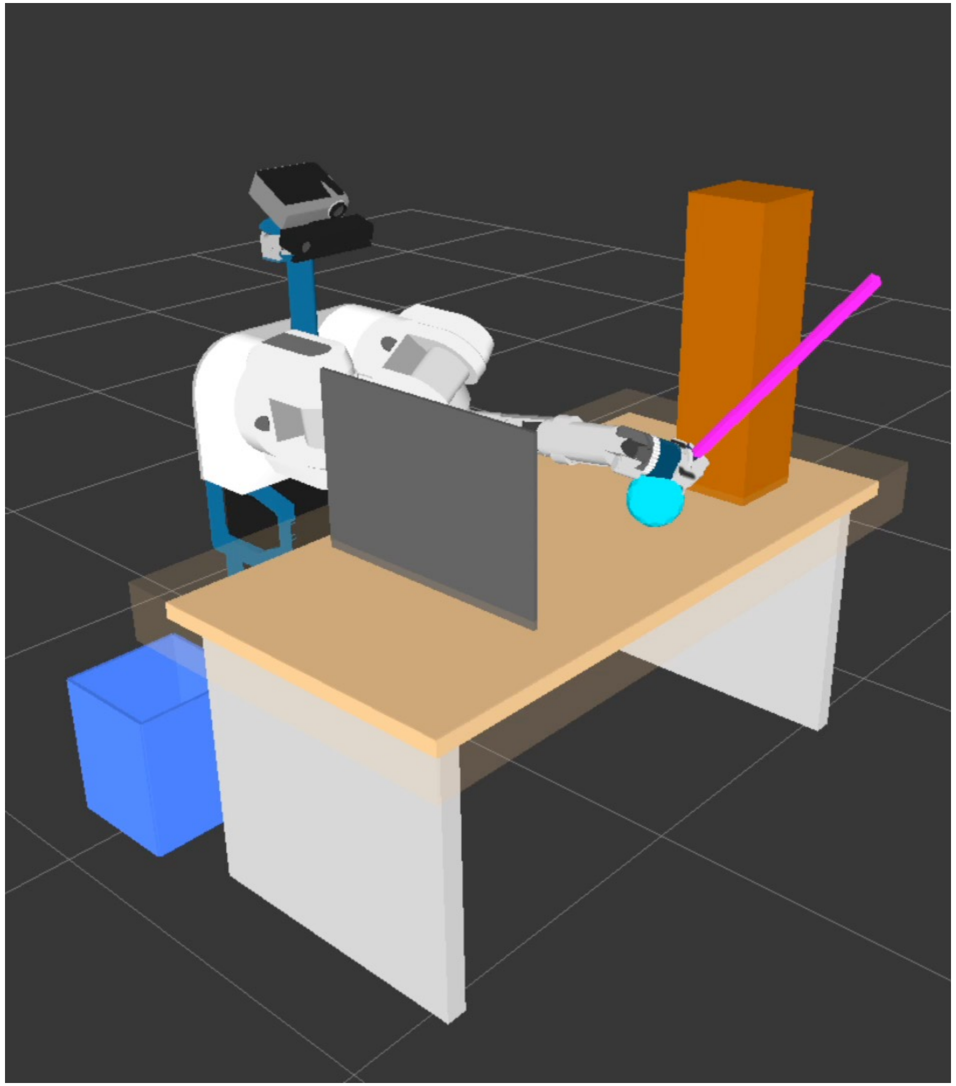
↓

Graph Construction Techniques

↓

Planning via Explicit Search

Geometric Path Planning Problem



Also known as
Piano Mover's Problem (Reif 79)

Given:

1. A *workspace* $\mathcal{W}$, where either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$.
2. An *obstacle region* $\mathcal{O} \subset \mathcal{W}$.
3. A *robot* defined in $\mathcal{W}$. Either a rigid body $\mathcal{A}$ or a collection of m links: $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m$.
4. The *configuration space* $\mathcal{C}$ (C_{obs} and C_{free} are then defined).
5. An *initial configuration* $\boldsymbol{q_I} \in \mathcal{C}_{free}$.
6. A *goal configuration* $\boldsymbol{q_G} \in \mathcal{C}_{free}$. The initial and goal configuration are often called a *query* $(\boldsymbol{q_I}, \boldsymbol{q_G})$.

Compute a (continuous) path, $\tau : [0,1] \rightarrow \mathcal{C}_{free}$, *such that* $\tau(0) = \boldsymbol{q_I}$ *and* $\tau(1) = \boldsymbol{q_G}$.

Also may want to minimize cost $c(\tau)$

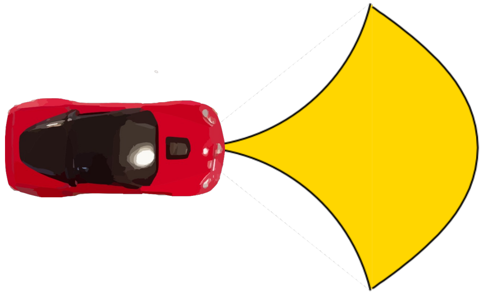
Differential constraints

In geometric path planning, we were only dealing with C-space

$$q \in \mathcal{C}$$

We now introduce differential constraints

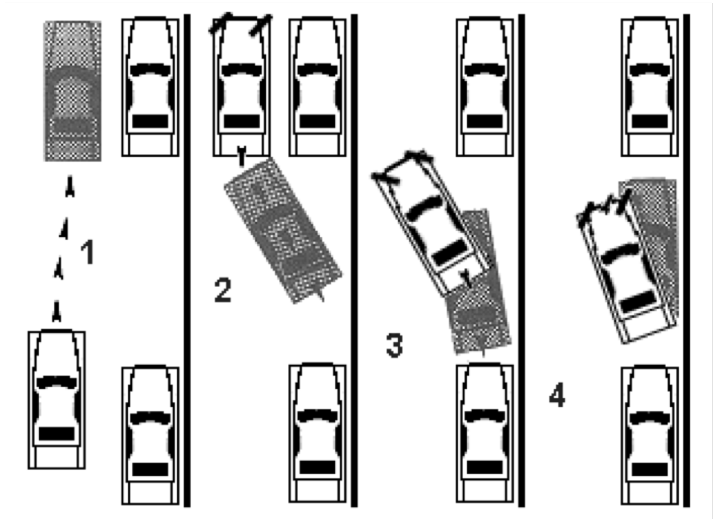
$$\begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = f\left(\begin{bmatrix} q \\ q \end{bmatrix}, u \right)$$

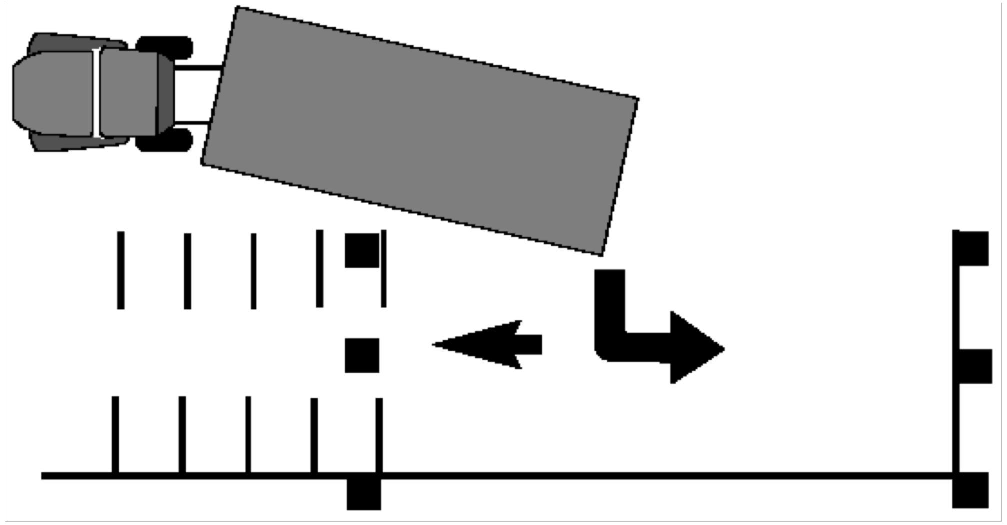


Let the state space *x* be the following augmented C-space

$$x = (q, \dot{q}) \qquad \dot{x} = f(x, u)$$

Differential constraints make things even harder





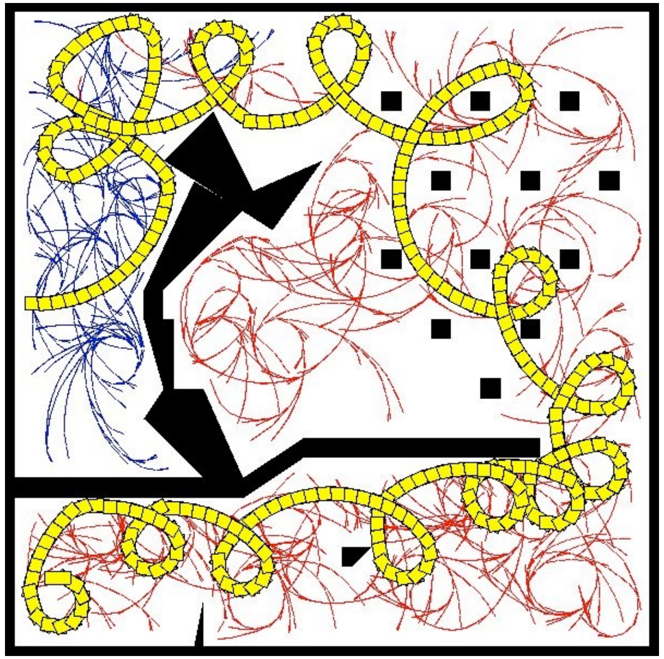
These are examples of non-holonomic system

non-holonomic differential constraints are not completely integrable

i.e. the system is trapped in some sub-manifold of the config space

Differential constraints make things even harder

Emergency landing where
UAV can only turn left



“Left-turning-car”

These are examples of non-holonomic system

non-holonomic differential constraints are not completely integrable

i.e. the system is trapped in some sub-manifold of the config space

Motion planning under differential constraints

1. Given world, obstacles, C-space, robot geometry (same)

2. Introduce state space *X*. Compute free and obstacle state space.

3. Given an action space U

4. Given a state transition equations $\dot{x} = f(x, u)$

5. Given initial and final state, cost function $J(x(t), u(t)) = \int c(x(t), u(t))dt$

6. Compute action trajectory that satisfies boundary conditions, stays in free state space and minimizes cost.

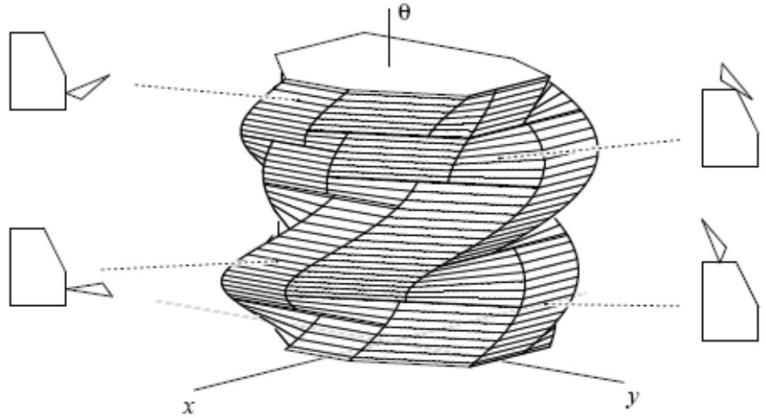
Challenges in Motion Planning

Computing configuration-space obstacles **HARD!**

Planning in continuous high-dimensional space **HARD!**

Underactuated dynamics/constrained system does not allow direct teleportation **HARD!**

Goal: tractable approximations with provable guarantees!



(EXAMPLE FROM HOWIE CHOSET)

Lecture Outline

Why is the problem hard?

↓

A recipe for solving motion planning problems

↓

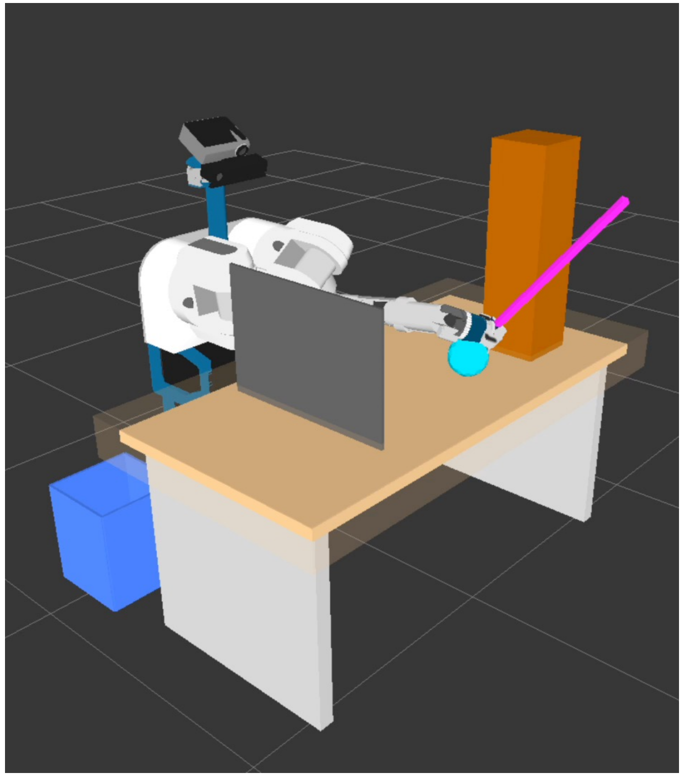
Graph Construction Techniques

↓

Planning via Explicit Search

How might we tackle this problem?

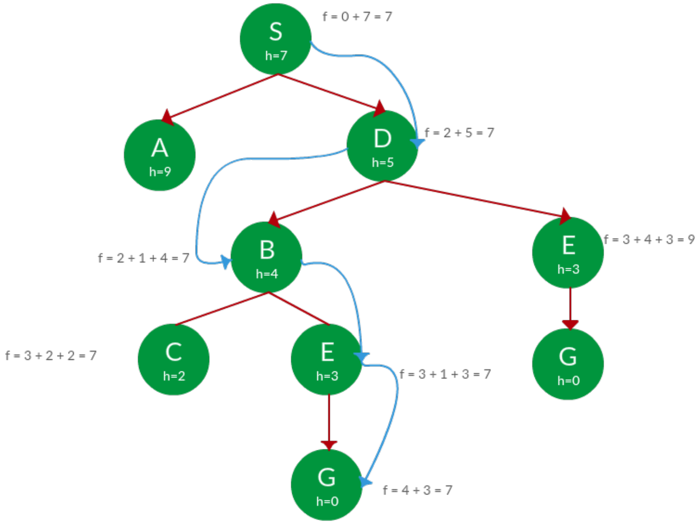
Lets use ideas from search!



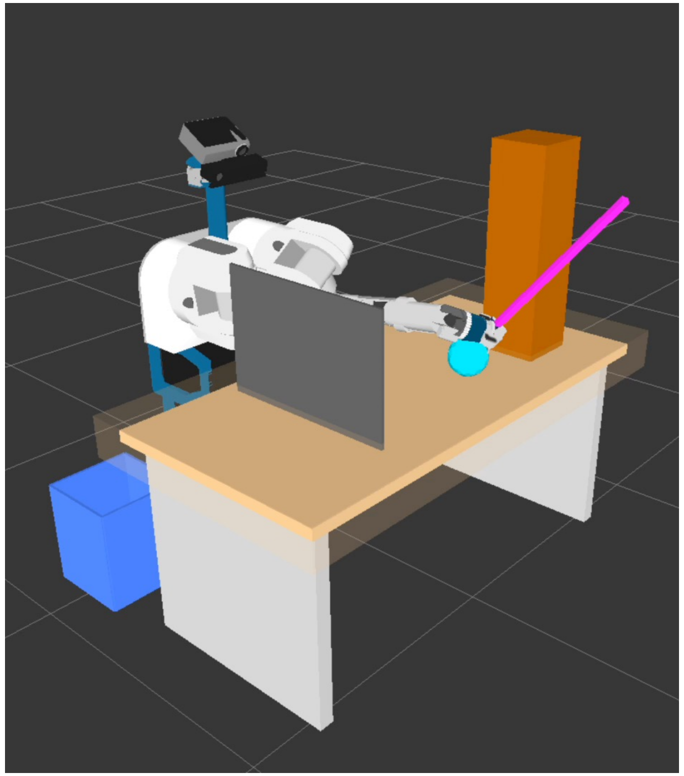
Given:

1. A *workspace* $\mathcal{W}$, where either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$.
2. An *obstacle region* $\mathcal{O} \subset \mathcal{W}$.
3. A *robot* defined in $\mathcal{W}$. Either a rigid body $\mathcal{A}$ or a collection of m links: $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m$.
4. The *configuration space* $\mathcal{C}$ (C_{obs} and C_{free} are then defined).
5. An *initial configuration* $\boldsymbol{q_I} \in \mathcal{C}_{free}$.
6. A *goal configuration* $\boldsymbol{q_G} \in \mathcal{C}_{free}$. The initial and goal configuration are often called a *query* $(\boldsymbol{q_I}, \boldsymbol{q_G})$.

Compute a (continuous) path, $\tau : [0,1] \to \mathcal{C}_{free}$, *such that* $\tau(0) = \boldsymbol{q_I}$ *and* $\tau(1) = \boldsymbol{q_G}$.



How might we tackle this problem?



Given:

1. A *workspace* $\mathcal{W}$, where either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$.
2. An *obstacle region* $\mathcal{O} \subset \mathcal{W}$.
3. A *robot* defined in $\mathcal{W}$. Either a rigid body $\mathcal{A}$ or a collection of m links: $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m$.
4. The *configuration space* $\mathcal{C}$ (C_{obs} and C_{free} are then defined).
5. An *initial configuration* $\boldsymbol{q_I} \in \mathcal{C}_{free}$.
6. A *goal configuration* $\boldsymbol{q_G} \in \mathcal{C}_{free}$. The initial and goal configuration are often called a *query* $(\boldsymbol{q_I}, \boldsymbol{q_G})$.

Compute a (continuous) path, $\tau : [0,1] \to \mathcal{C}_{free}$, *such that* $\tau(0) = \boldsymbol{q_I}$ *and* $\tau(1) = \boldsymbol{q_G}$.

Continuous space

Hard to characterize obstacles

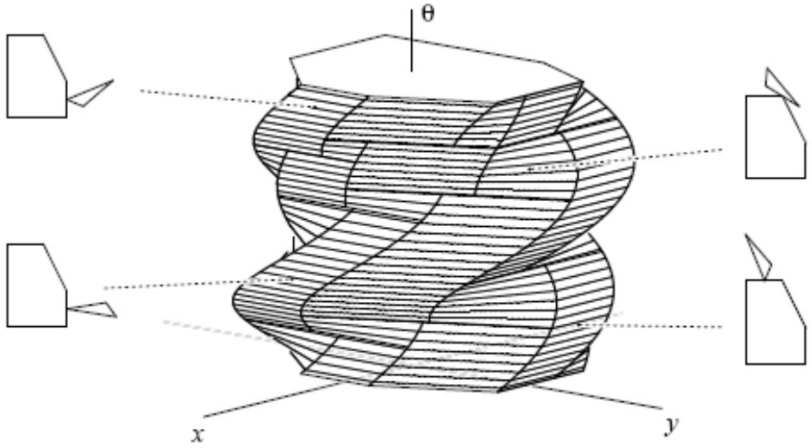
Sampling-Based Motion Planning

Computing configuration-space obstacles is hard

- Use a collision checker instead!

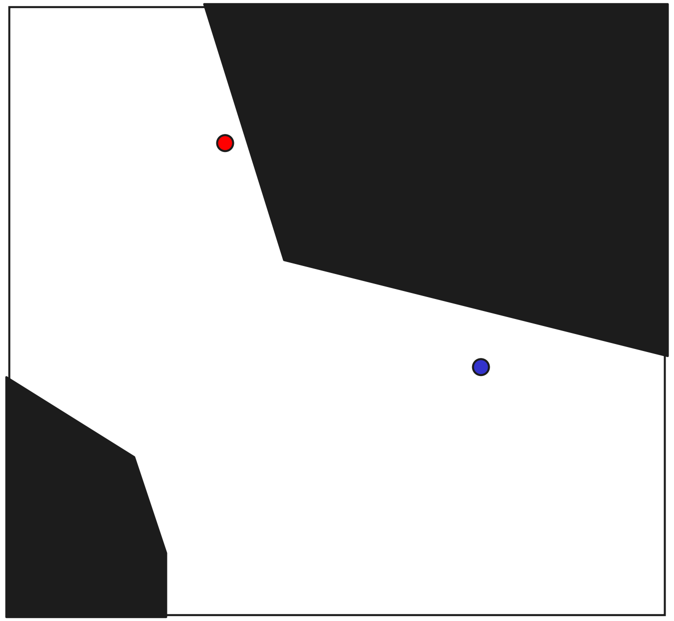
Planning in continuous high-dimensional space is hard

- Construct a discrete graph approximation of the continuous space!



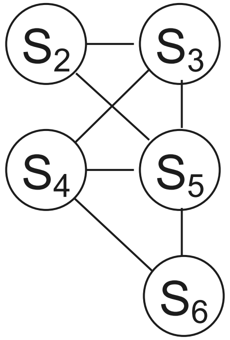
(EXAMPLE FROM HOWIE CHOSET)

Planning as Search



planning map

Convert into a search problem



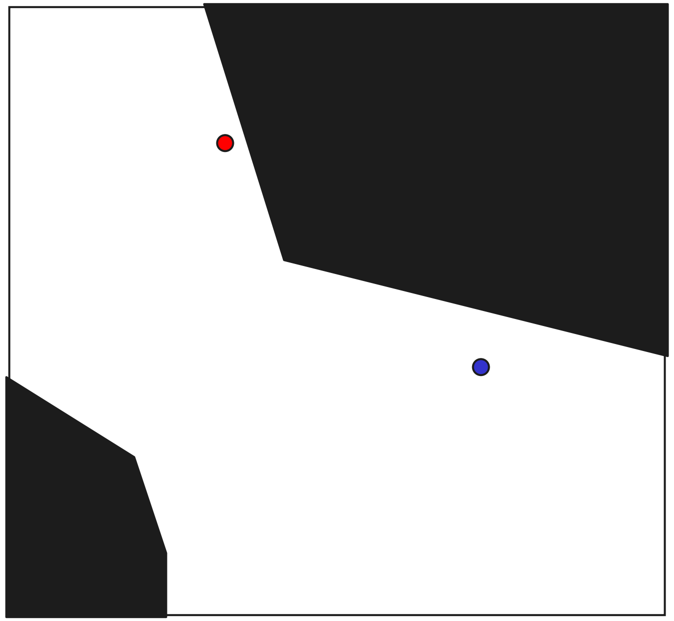
search the graph
for a least-cost path
from s_{start} to s_{goal}

Can use efficient techniques for **discrete** graph search

Explicit graph search

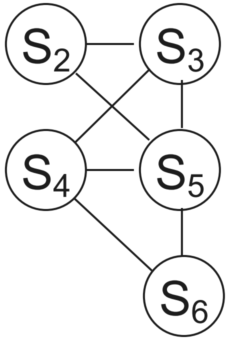
Implicit sampling-based search

Recasting Planning as Search



Convert into a search problem

How?



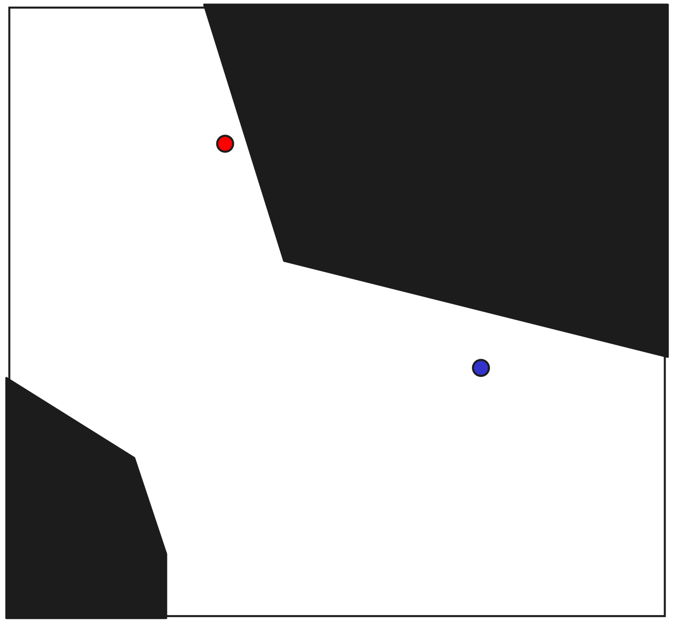
planning map

search the graph
for a least-cost path
from s_{start} to s_{goal}

Can use efficient techniques for **discrete** graph search

Which ones?

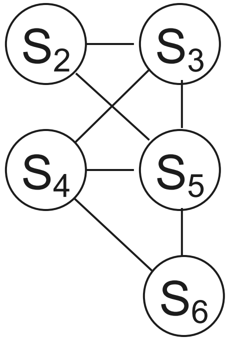
Recasting Planning as Search



planning map

Convert into a search problem

How? = Sampling

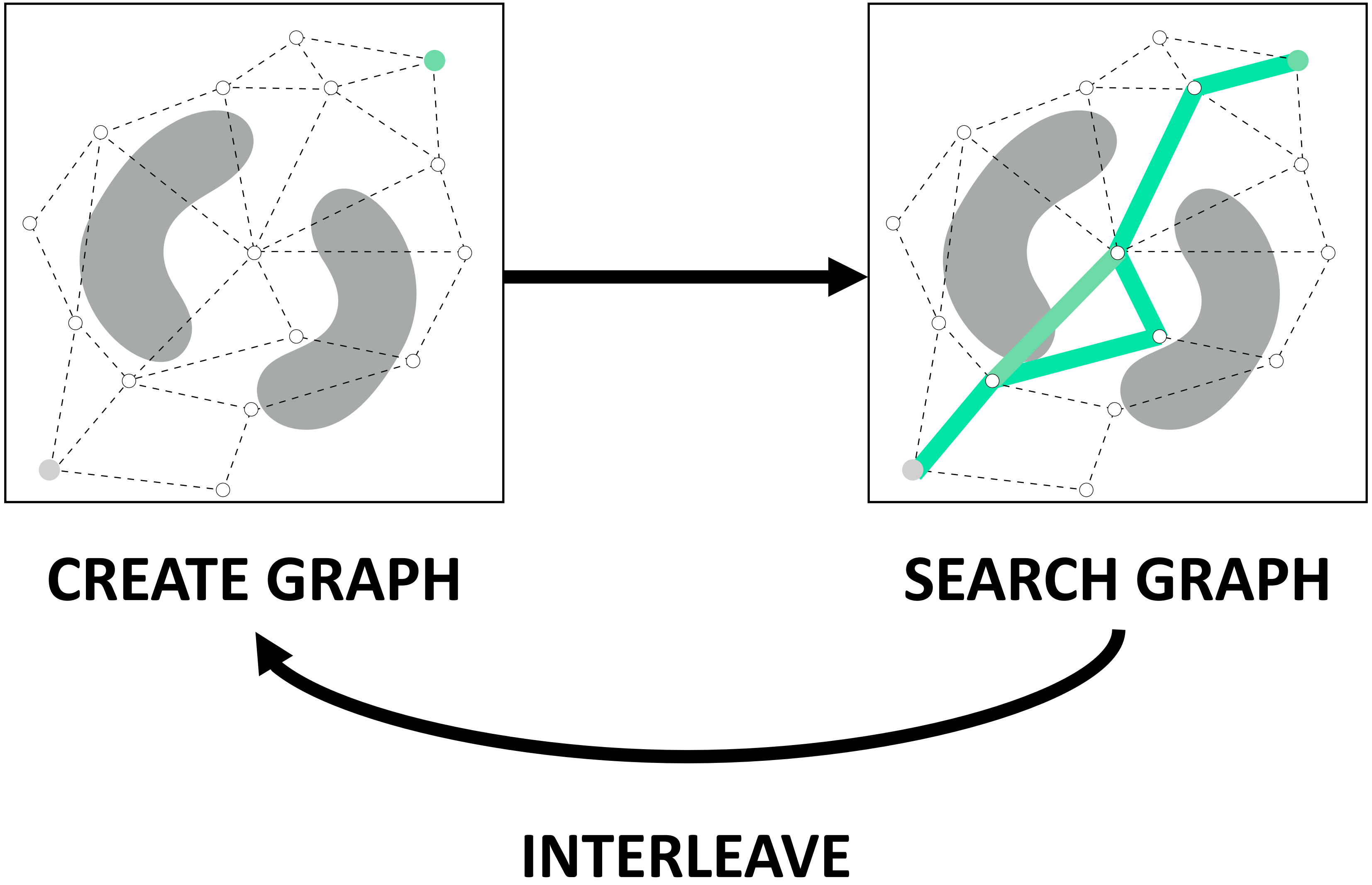


search the graph for a least-cost path from s_{start} to s_{goal}

Can use efficient techniques for **discrete** graph search

Which ones? = Best-first explicit search or Implicit sampling-based graph search

Sampling-Based Motion Planning



Sampling-Based Motion Planning

NEW PLANNING ALGORITHM	**=**	**GRAPH CONSTRUCTION**	**×**	**FANCY SEARCH ALGORITHM**	**×**	**++ for efficiency**

Lecture Outline

Why is the problem hard?

↓

A recipe for solving motion planning problems

↓

Graph Construction Techniques

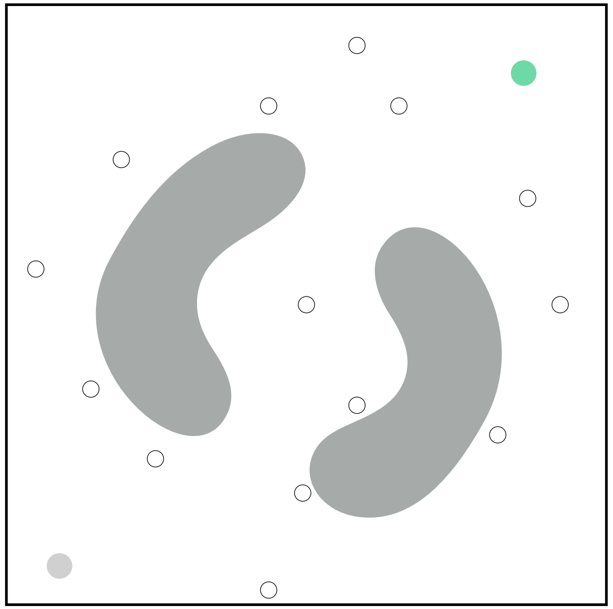
↓

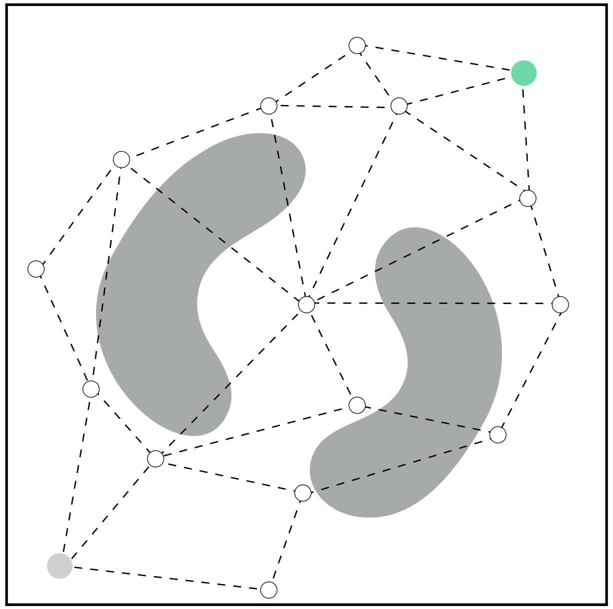
Planning via Explicit Search

Creating a Graph

$$G = (V, E)$$

1. Sample collision-free configurations as vertices (including start and goal)
2. Connect neighboring vertices with simple movements as edges

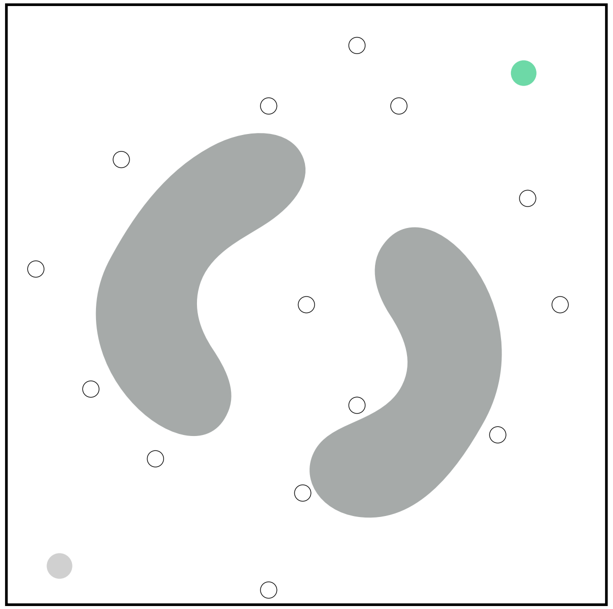




Creating a Graph

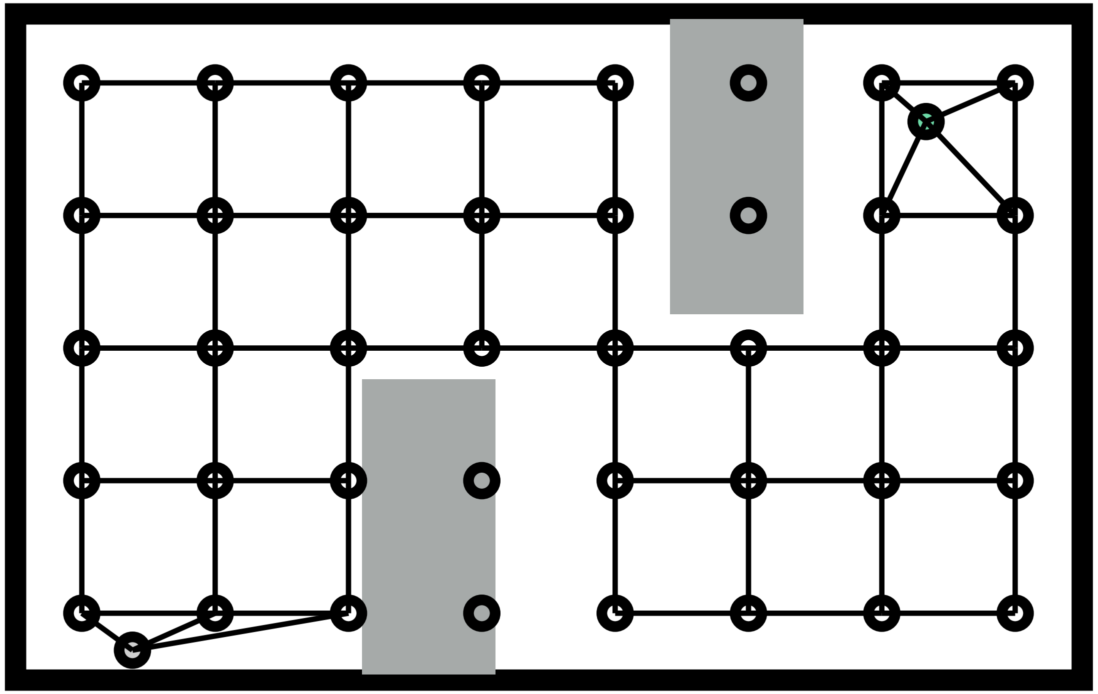
$$G = (V, E)$$

1. Sample collision-free configurations as vertices (including start and goal)
2. Connect neighboring vertices with simple movements as edges



Strategy 1: Lattice Sampling / Discretization

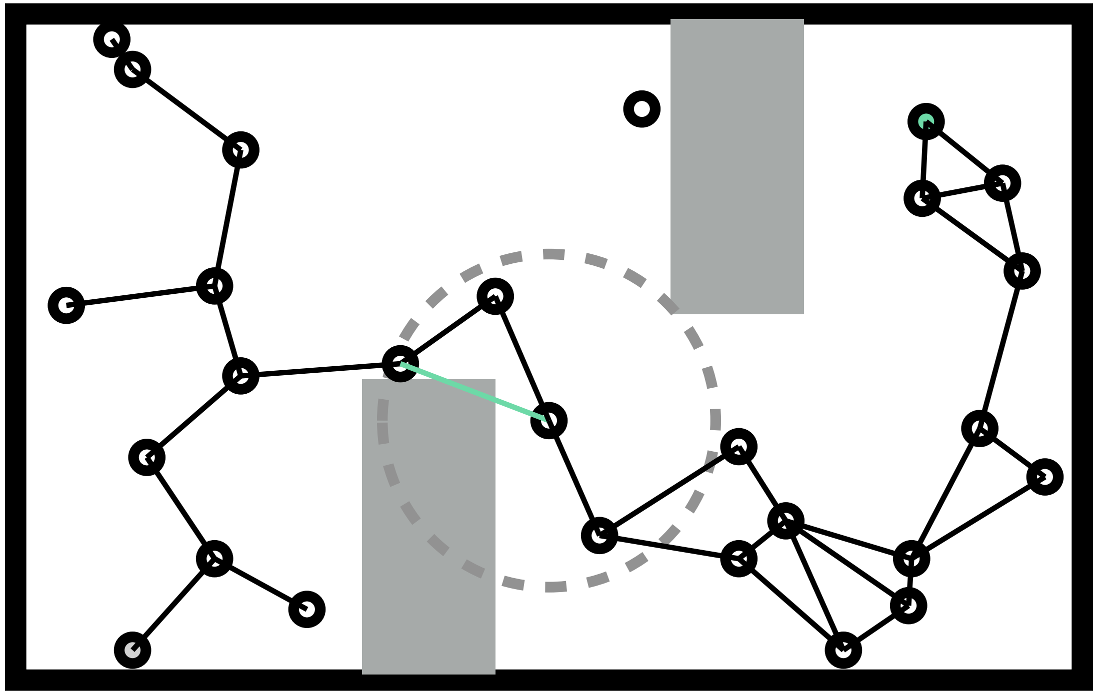
Main idea: create a grid, and connect neighboring points (4-conn, 8-conn, …)



Pros/Cons?

Strategy 2: Uniform Random Sampling

Main idea: sample uniformly between each dimension's lower/upper bounds
Connect vertices within radius (r-disc) or k nearest neighbors



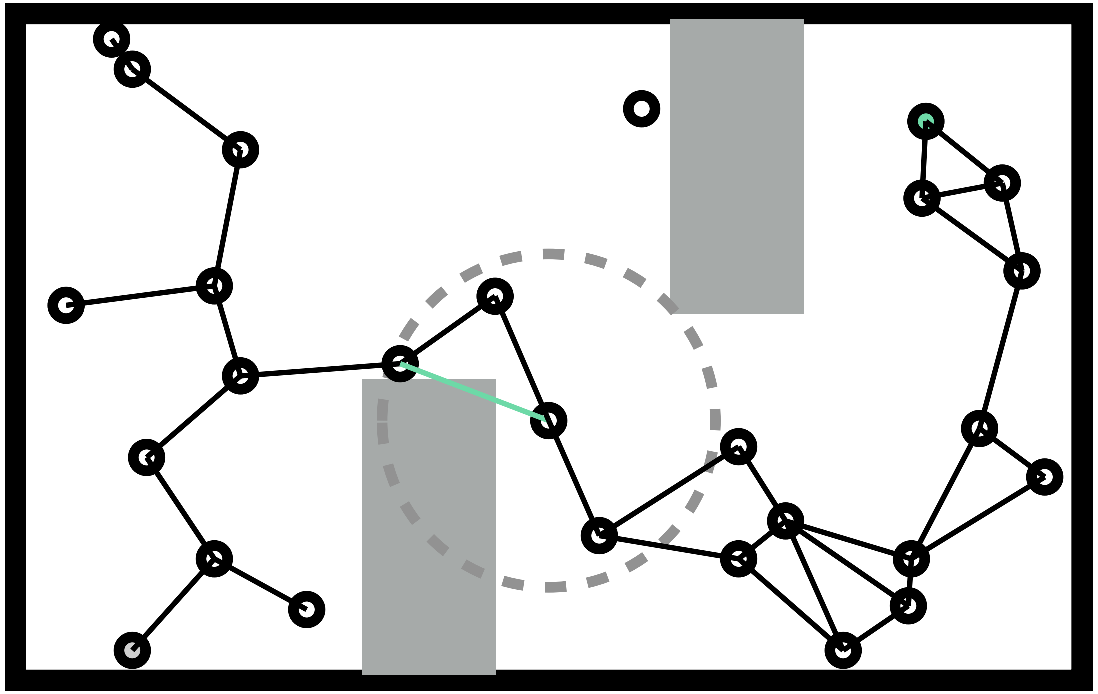
Pros/Cons?

KAVRAKI ET AL., 1996

Probabilistic Roadmap (PRM)

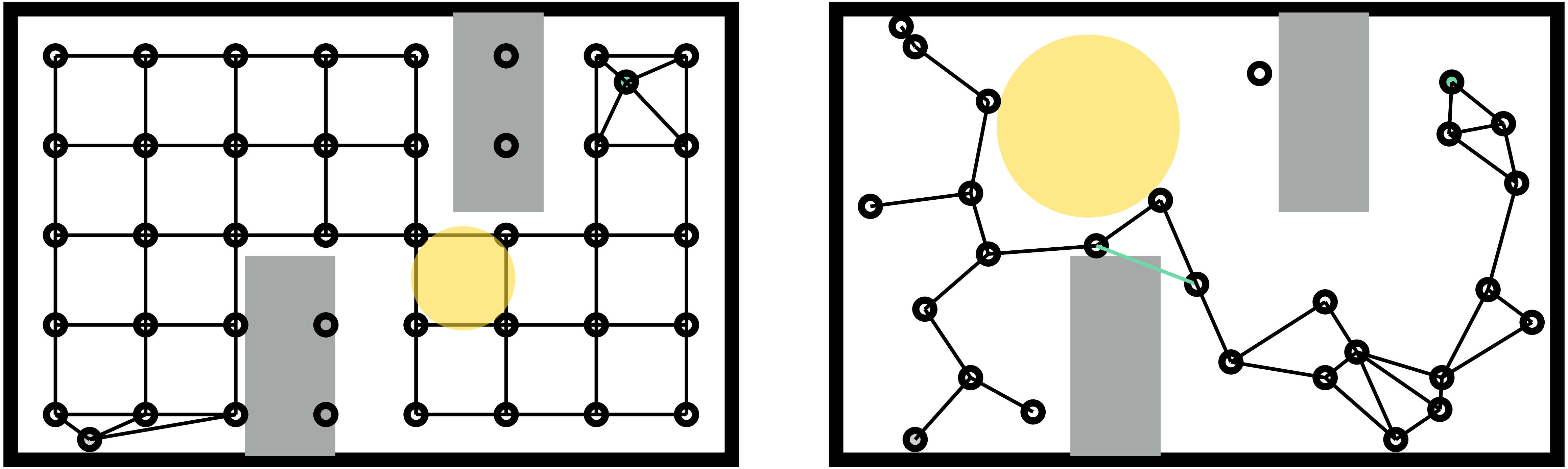
When should we collision-check edges?

What is the optimal radius? (PRM with optimal radius = PRM*)



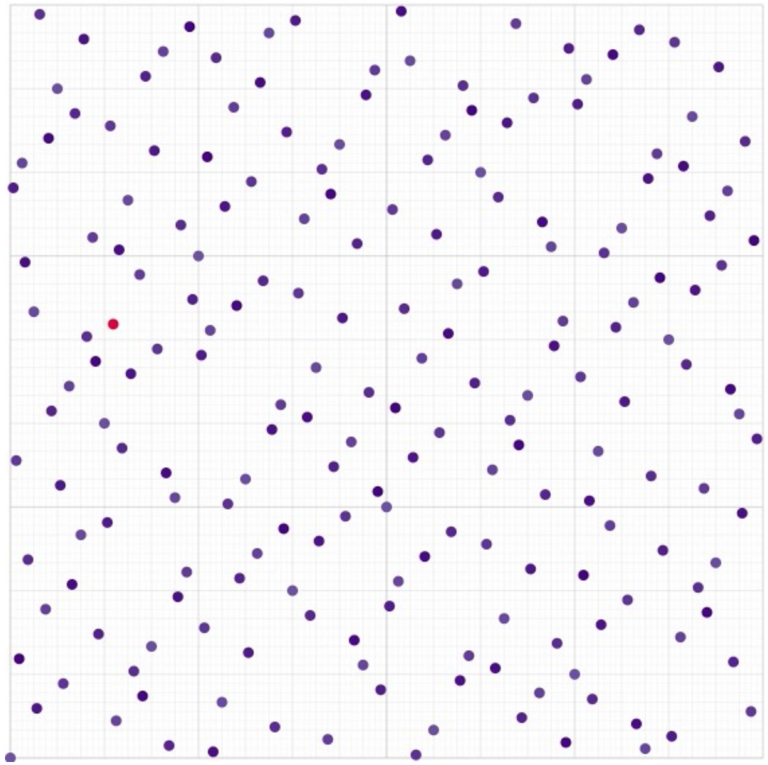
KAVRAKI ET AL., 1996

Alternatives to Random Sampling

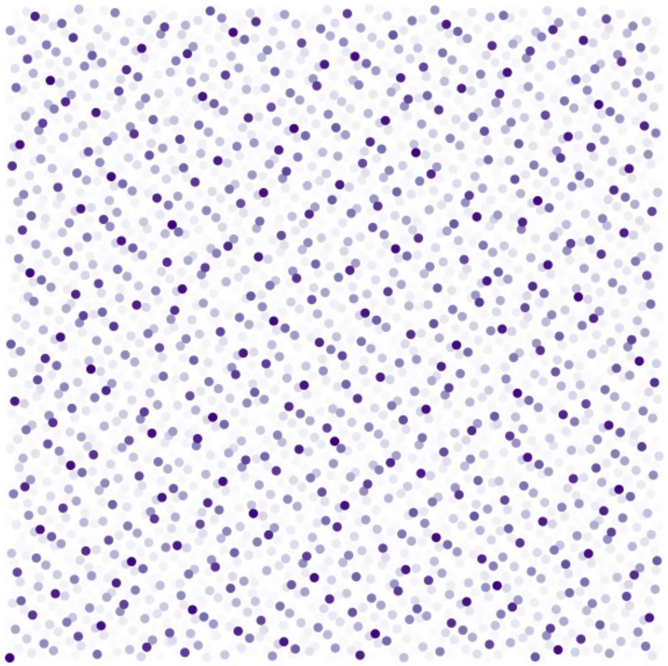


Strategy 3: Low-Dispersion Sampling

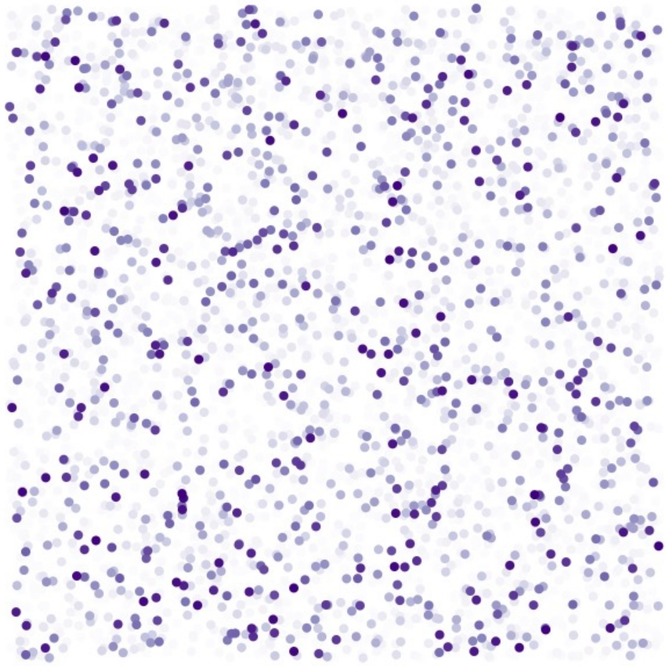
Main idea: Halton sequence uniformly densifies the space



Halton sequence



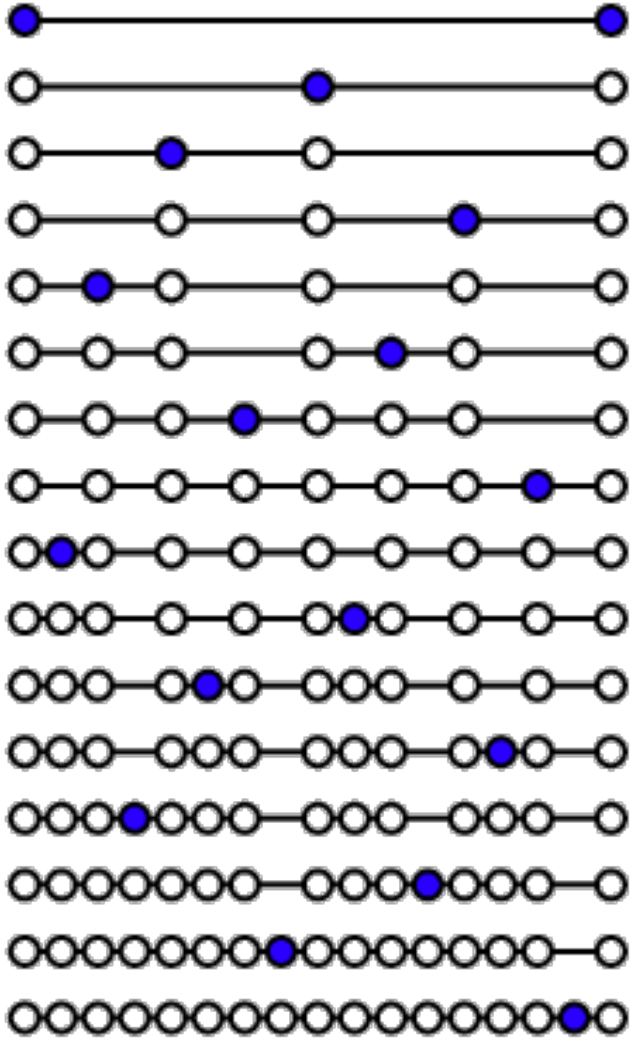
Uniformly randomly sample



HTTPS://OBSERVABLEHQ.COM/@JRUS/HALTON

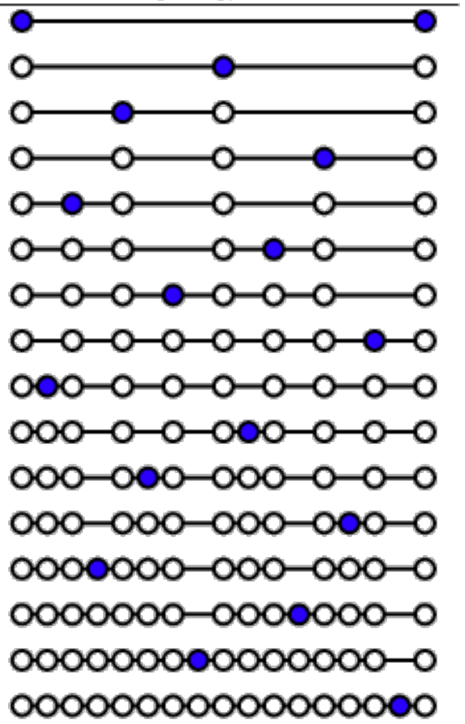
Detour: Van der Corput sequence

i	Naive Sequence	Binary	Reverse Binary	Van der Corput	Points in $[0,1]/\sim$
1	0	.0000	.0000	0	
2	1/16	.0001	.1000	1/2	
3	1/8	.0010	.0100	1/4	
4	3/16	.0011	.1100	3/4	
5	1/4	.0100	.0010	1/8	
6	5/16	.0101	.1010	5/8	
7	3/8	.0110	.0110	3/8	
8	7/16	.0111	.1110	7/8	
9	1/2	.1000	.0001	1/16	
10	9/16	.1001	.1001	9/16	
11	5/8	.1010	.0101	5/16	
12	11/16	.1011	.1101	13/16	
13	3/4	.1100	.0011	3/16	
14	13/16	.1101	.1011	11/16	
15	7/8	.1110	.0111	7/16	
16	15/16	.1111	.1111	15/16	



Detour: Van der Corput sequence

i	Naive Sequence	Binary	Reverse Binary	Van der Corput	Points in $[0,1]/\sim$
1	0	.0000	.0000	0	
2	1/16	.0001	.1000	1/2	
3	1/8	.0010	.0100	1/4	
4	3/16	.0011	.1100	3/4	
5	1/4	.0100	.0010	1/8	
6	5/16	.0101	.1010	5/8	
7	3/8	.0110	.0110	3/8	
8	7/16	.0111	.1110	7/8	
9	1/2	.1000	.0001	1/16	
10	9/16	.1001	.1001	9/16	
11	5/8	.1010	.0101	5/16	
12	11/16	.1011	.1101	13/16	
13	3/4	.1100	.0011	3/16	
14	13/16	.1101	.1011	11/16	
15	7/8	.1110	.0111	7/16	
16	15/16	.1111	.1111	15/16	



The b-ary representation of the positive integer $n \geq 1$ is

$$n = \sum_{k=0}^{L-1} d_k(n) b^k = d_0(n) b^0 + \cdots + d_{L-1}(n) b^{L-1},$$

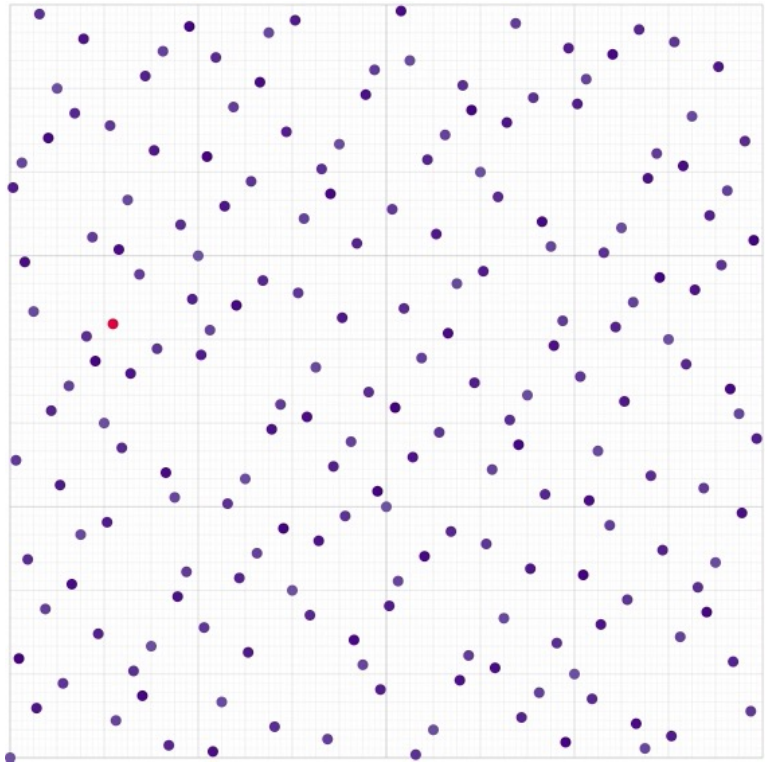
where b is the base in which the number n is represented, and $0 \leq d_k(n) < b$; that is, the k-th digit in the b-ary expansion of n. The n-th number in the van der Corput sequence is

$$g_b(n) = \sum_{k=0}^{L-1} d_k(n) b^{-k-1} = d_0(n) b^{-1} + \cdots + d_{L-1}(n) b^{-L}.$$

Whiteboard

Strategy 3: Low-Dispersion Sampling

Halton sequence – multi-dimensional van der corput sequence, co-prime bases



`positional(1234, 10)` $\longrightarrow [1, 2, 3, 4]$

`halton(1234, 10)` $\longrightarrow \frac{4}{10} + \frac{3}{100} + \frac{2}{1000} + \frac{1}{10000}$

`positional(1234, 2)` $\longrightarrow [1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0]$

`halton(1234, 2)` $\longrightarrow \frac{1}{4} + \frac{1}{32} + \frac{1}{128} + \frac{1}{256} + \frac{1}{2048}$

`positional(1234, 3)` $\longrightarrow [1, 2, 0, 0, 2, 0, 1]$

`halton(1234, 3)` $\longrightarrow \frac{1}{3} + \frac{2}{27} + \frac{2}{729} + \frac{1}{2187}$

`positional(0x4d2, 16)` $\longrightarrow [4, 13, 2]$

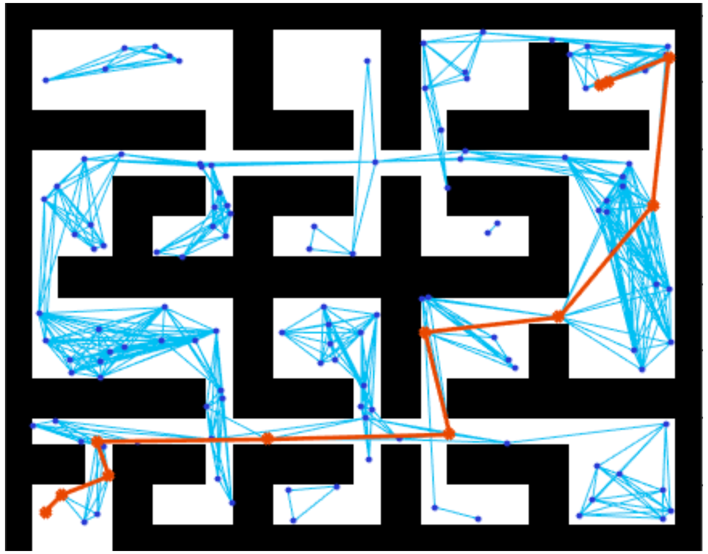
`halton(0x4d2, 16)` $\longrightarrow \frac{2}{16} + \frac{13}{256} + \frac{4}{4096}$

HTTPS://OBSERVABLEHQ.COM/@JRUS/HALTON

What Graphs Are Good?

A good graph must be sparse (both in vertices and edges)

A good graph must have good free-space coverage
For every configuration in the free space, there's a vertex in the graph that can be connected to it.



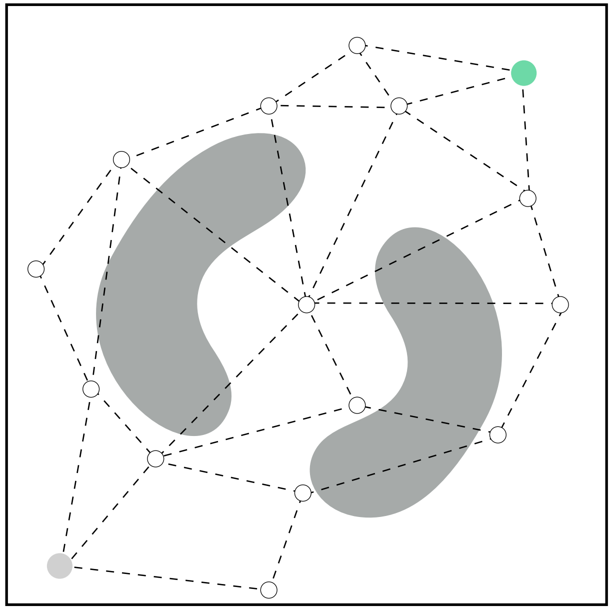
A good graph must have good free-space connectivity

For every connected pair of points in the free space, there's a path on the graph between them.

Creating a Graph

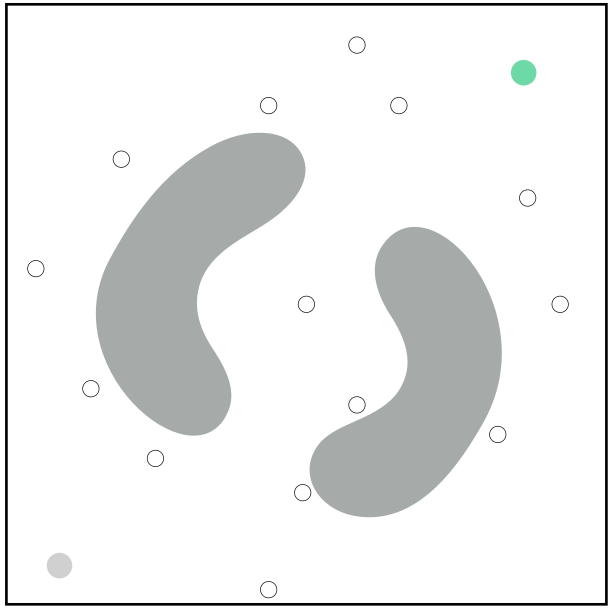
$$G = (V, E)$$

1. Sample collision-free configurations as vertices (including start and goal)
2. Connect neighboring vertices with simple movements as edges

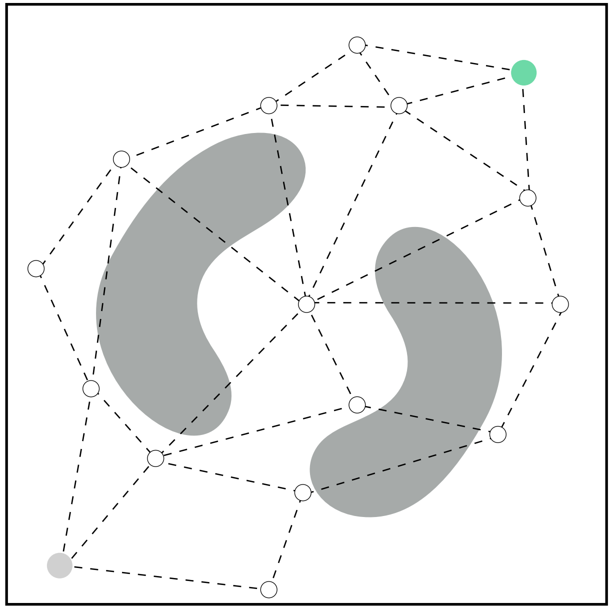


Creating a Graph

$$G = (V, E)$$



Connect collision free edges



API for Graph Construction

Input

1. A collision checker

$\texttt{coll}(q)$

2. Steering method

$\texttt{steer}(q_1, q_2)$

Graph Construction

Output

Connected graph for search

Let's take a look at the inputs

We need to give the planner a collision checker

$$\texttt{coll}(q) = \begin{cases} 0 & \text{in collision, i.e. } q \in \mathcal{C}_{obs} \\ 1 & \text{free, i.e. } q \in \mathcal{C}_{free} \end{cases}$$

What work does this function have to do?

Collision checking is expensive!

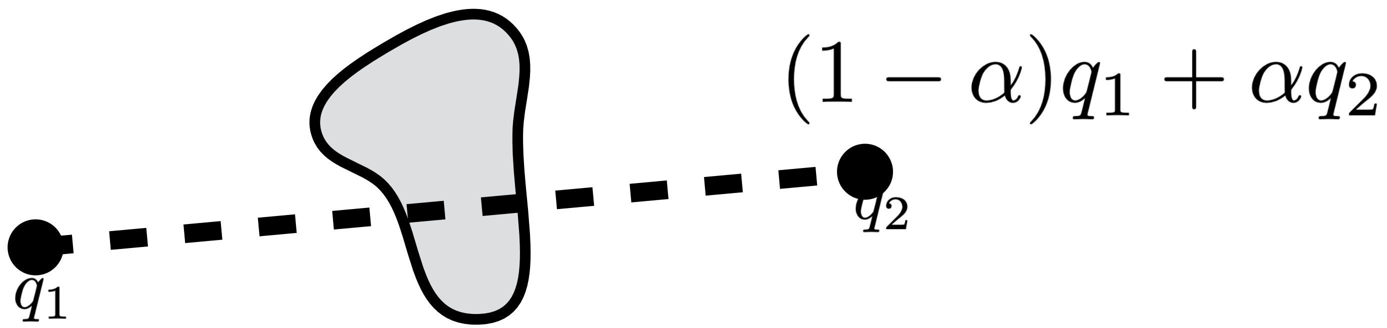
Let's take a look at the inputs

We need to give the planner a steer function

$$\texttt{steer}(q_1, q_2)$$

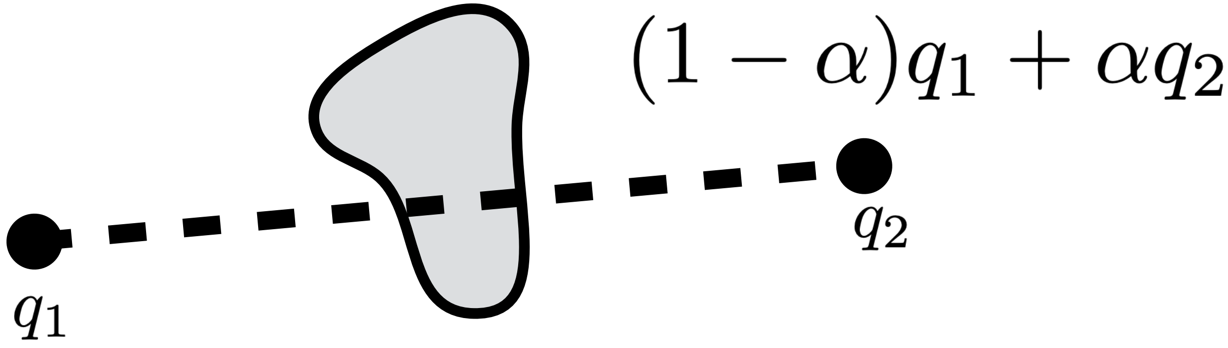
A steer function tries to join two configurations with a feasible path

Computes simple path, calls coll(q), and returns success if path is free



Example: Connect them with a straight line and check for feasibility

Can steer be smart about collision checking?



$\texttt{steer}(q_1, q_2)$ has to assure us line is collision free (upto a resolution)

Things we can try:

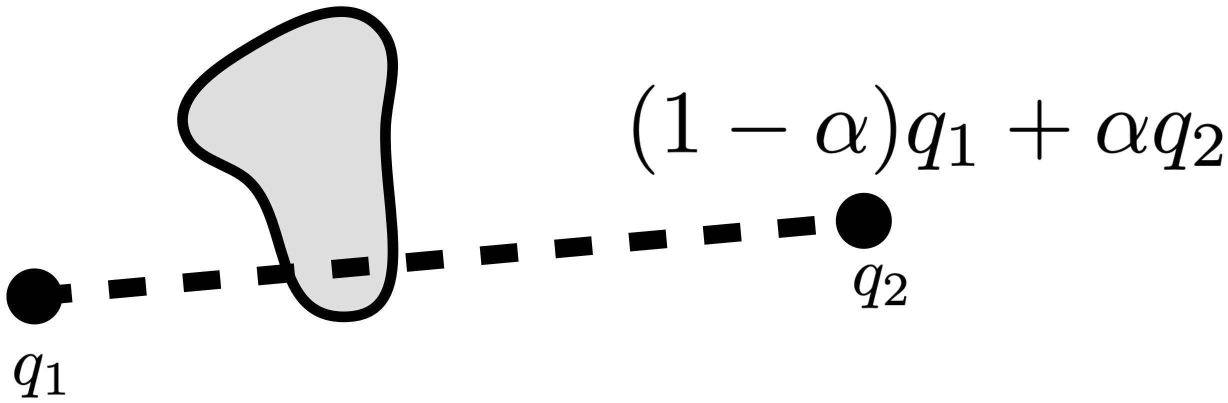
1. Step forward along the line and check each point

2. Step backwards along the line and check each point

…….

Can steer be smart about collision checking?

Say we chunk the line into 16 parts



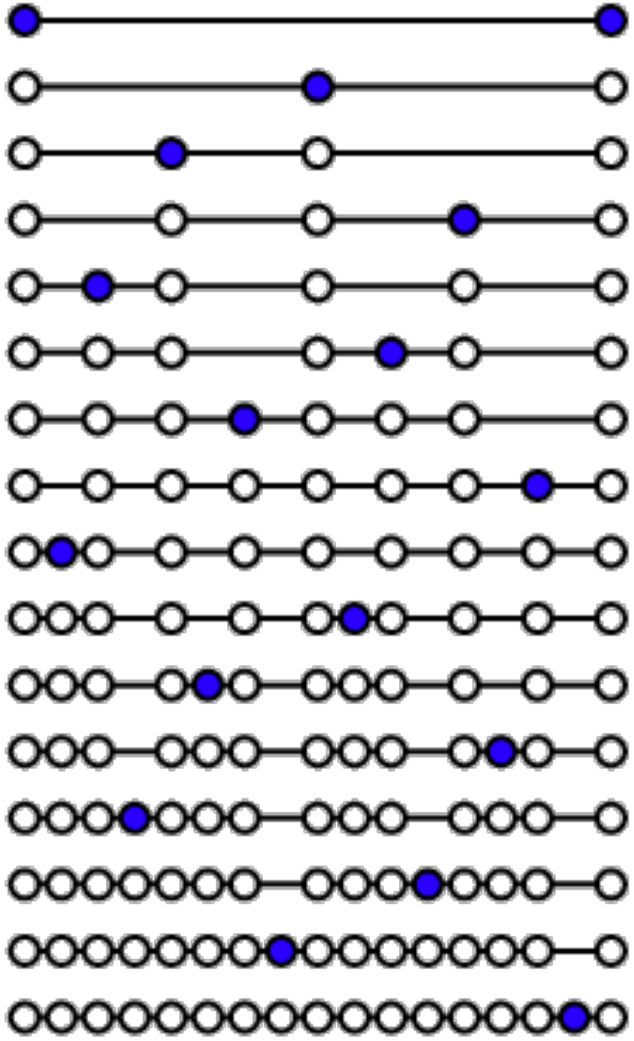
Any collision checking strategy corresponds to sequence

(Naive) $\alpha = 0, \frac{1}{16}, \frac{2}{16}, \frac{3}{16}, \cdots, \frac{15}{16}$

(Bisection) $\alpha = 0, \frac{8}{16}, \frac{4}{16}, \frac{12}{16}, \cdots, \frac{15}{16}$

Ans: Van der Corput sequence

i	Naive Sequence	Binary	Reverse Binary	Van der Corput	Points in $[0,1]/\sim$
1	0	.0000	.0000	0	
2	1/16	.0001	.1000	1/2	
3	1/8	.0010	.0100	1/4	
4	3/16	.0011	.1100	3/4	
5	1/4	.0100	.0010	1/8	
6	5/16	.0101	.1010	5/8	
7	3/8	.0110	.0110	3/8	
8	7/16	.0111	.1110	7/8	
9	1/2	.1000	.0001	1/16	
10	9/16	.1001	.1001	9/16	
11	5/8	.1010	.0101	5/16	
12	11/16	.1011	.1101	13/16	
13	3/4	.1100	.0011	3/16	
14	13/16	.1101	.1011	11/16	
15	7/8	.1110	.0111	7/16	
16	15/16	.1111	.1111	15/16	



Boundary Value Problem

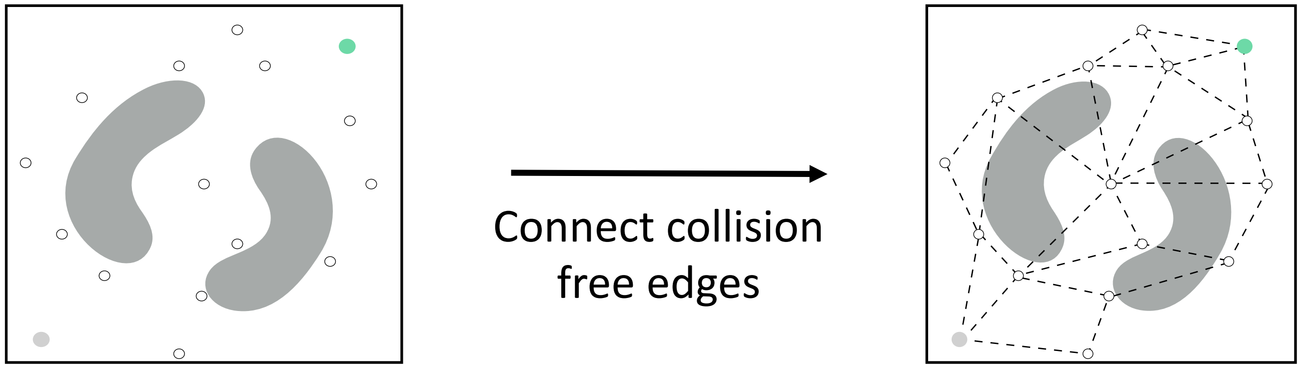
How can we move from one configuration to another?
→Hard in general!

Define a steering function that is tasked with connecting two configurations
→Previously, steering function was trivial (straight line)

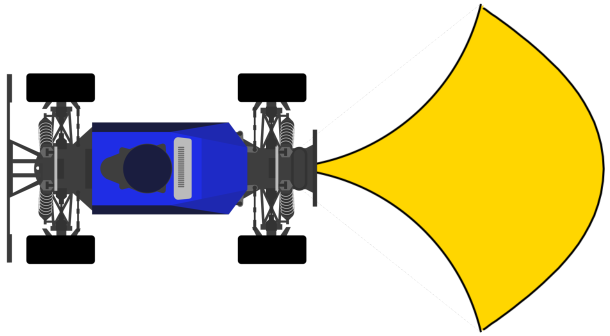
Differential Constraints on Graphs

To construct a graph under differential constraints:

1. Sample collision free configuration states (check with collision checker)
2. Solve boundary-value problem to see if states can be connected
3. If connectable, add an edge, otherwise no edge
4. Benefit!



Solving the Boundary Value Problem



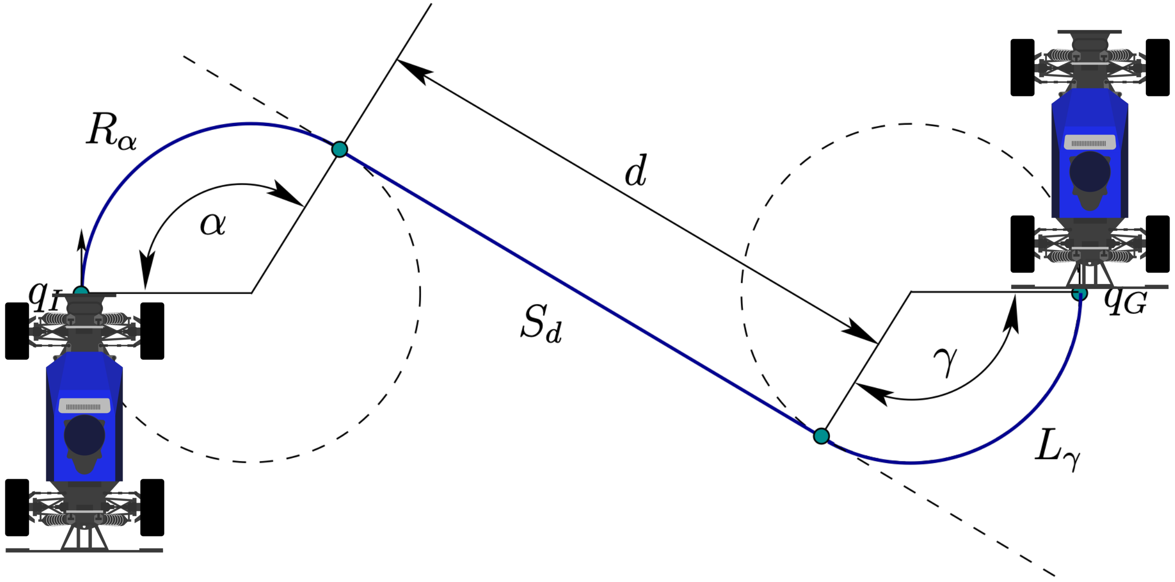
$$q_1 = (x_1, y_1, \theta_1)$$

$$q_2 = (x_2, y_2, \theta_2)$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \frac{v\tan\delta}{L} \end{bmatrix}$$

$$0 \leq v \leq v_{\max}, |\delta| \leq \delta_{\max}$$

Dubins Curves



$R_\alpha S_d L_\gamma$

RIGHT-STRAIGHT-LEFT

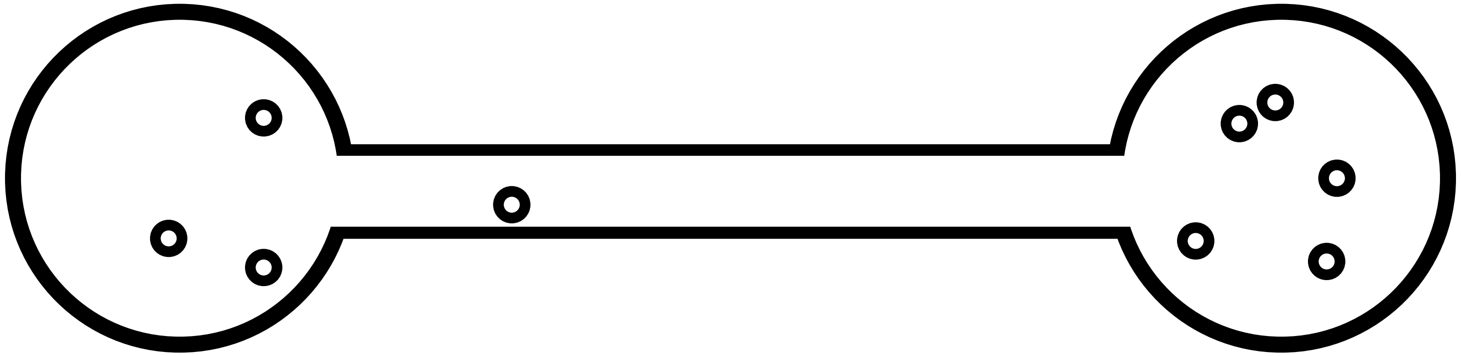
Dubins showed that all solutions had to be one of six classes

$\{LRL, RLR, LSL, LSR, RSL, RSR\}$

Given two configurations to connect, evaluate all six options, return shortest one

Car has fixed forward velocity; Reeds-Shepp curves may include backward velocity

What Environments Are Hard?



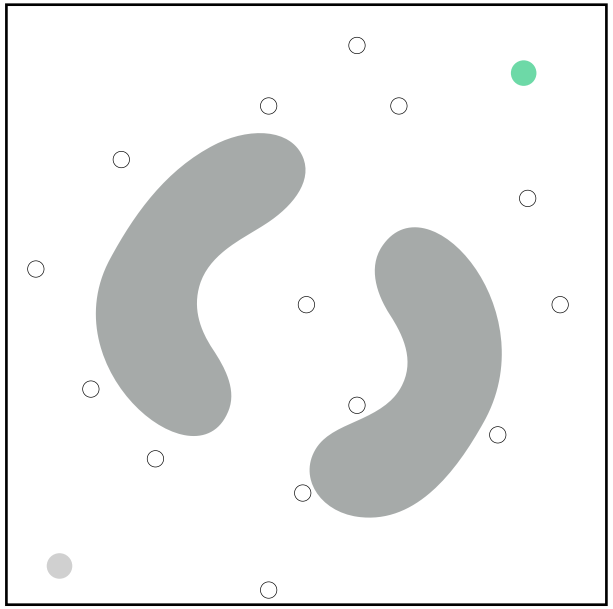
Sampling-based methods struggle with narrow passages
Probability of sampling an edge in the passage is very small, so with a finite number of samples, the two halves of the roadmap may not be connected

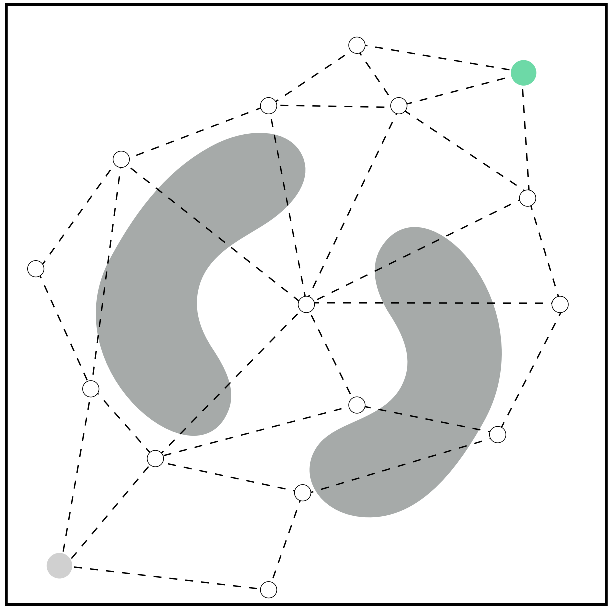
Practical solutions: sample near obstacle surface, bridge test to add samples between two obstacles, train ML algorithm to detect narrow passages

Creating a Graph

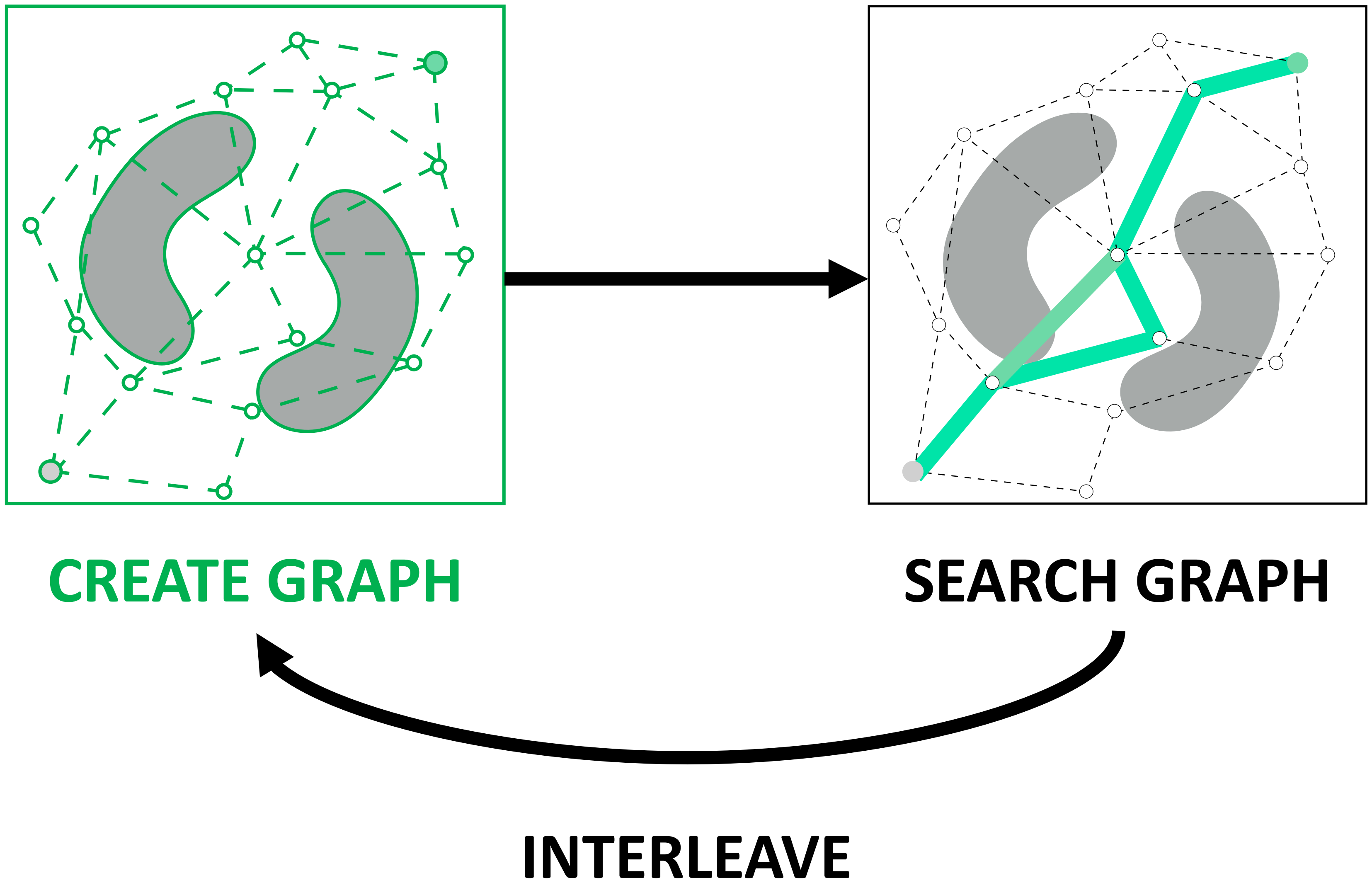
$$G = (V, E)$$

1. **Sample collision-free configurations as vertices (including start and goal)**
2. **Connect neighboring vertices with simple movements as edges**





Sampling-Based Motion Planning



Lecture Outline

Why is the problem hard?

↓

A recipe for solving motion planning problems

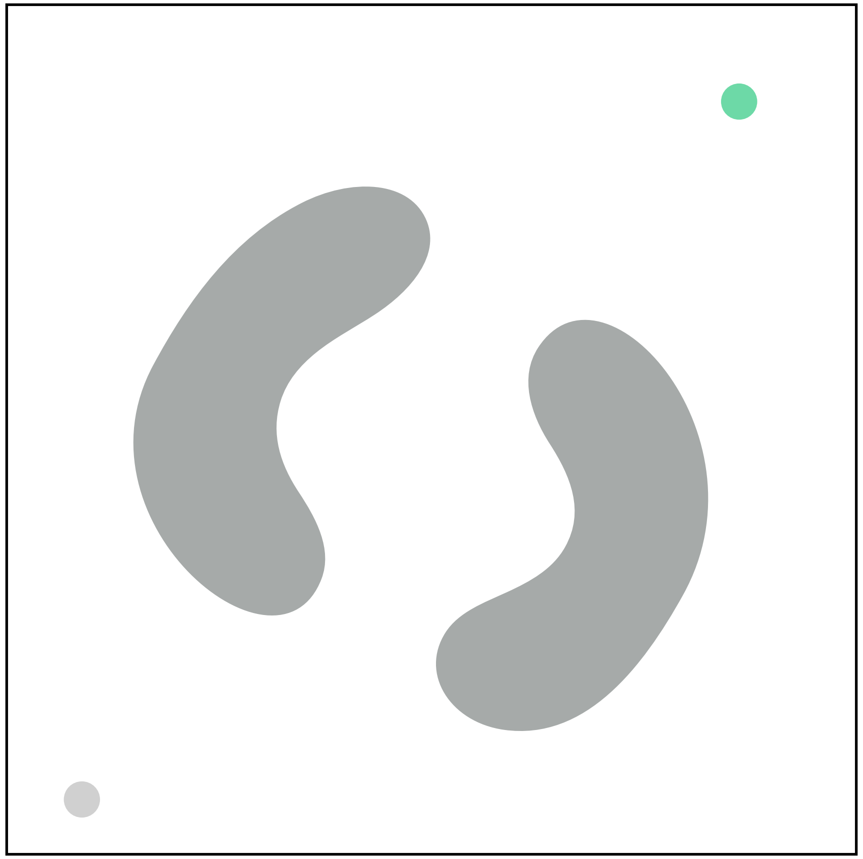
↓

Graph Construction Techniques

↓

Planning via Explicit Search

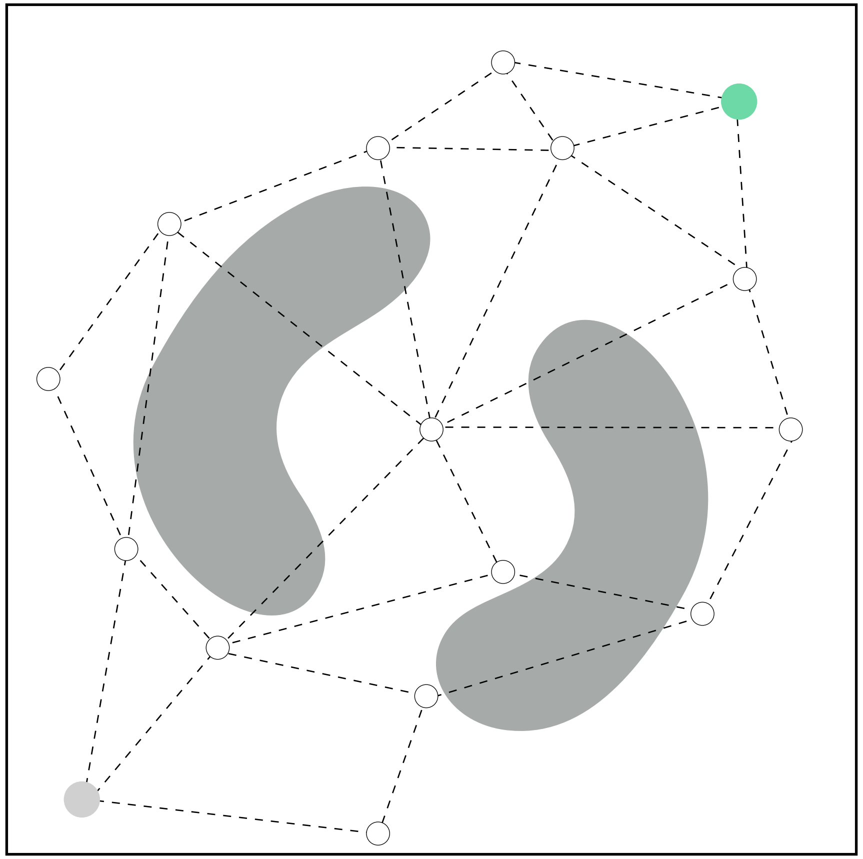
Minimal Cost Path on a Graph



START, GOAL

COST (E.G. LENGTH)

Minimal Cost Path on a Graph

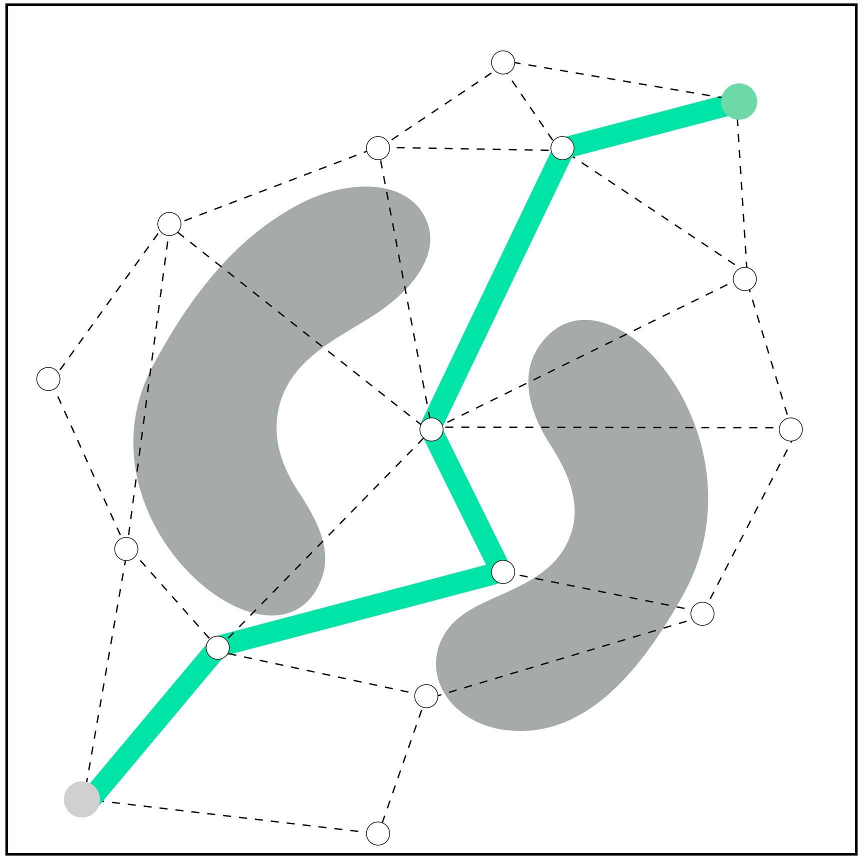


START, GOAL

COST (E.G. LENGTH)

GRAPH (VERTICES, EDGES)

Minimal Cost Path on a Graph

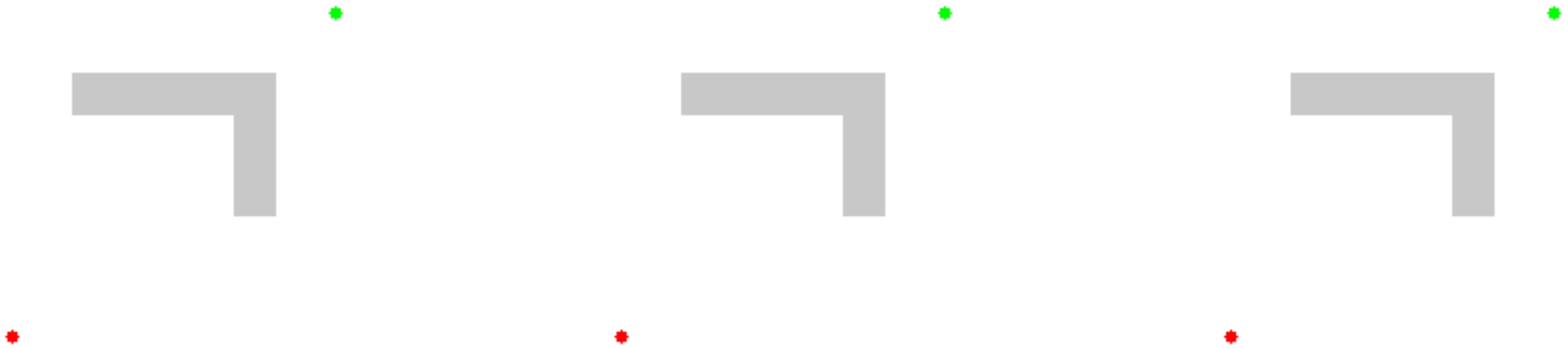


START, GOAL

COST (E.G. LENGTH)

GRAPH (VERTICES, EDGES)

Best-First Search Meta-Algorithm



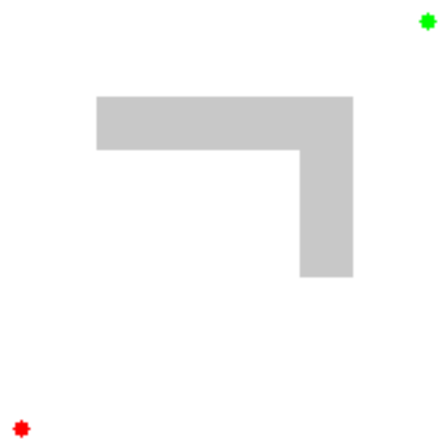
(WIKIPEDIA)

Best-First Search Meta-Algorithm

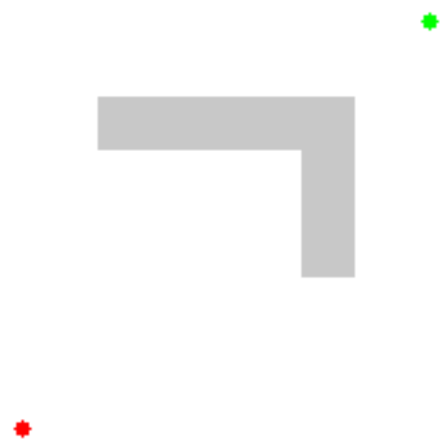
Key insight: maintain a priority queue of promising nodes, ranked by f(s)

-Initialize queue with start node
-While goal isn’t reached
  Pop the most promising node from the queue
  If it’s not the goal, enqueue its neighbors
-When goal is reached, compute path by backtracking to the start

Best-First Search Meta-Algorithm



DIJKSTRA



A*

(WIKIPEDIA)

Best-First Search Implementation

Inputs: graph G = (V, E); cost c(s, s') = c(e); start and goal

Data structures maintained

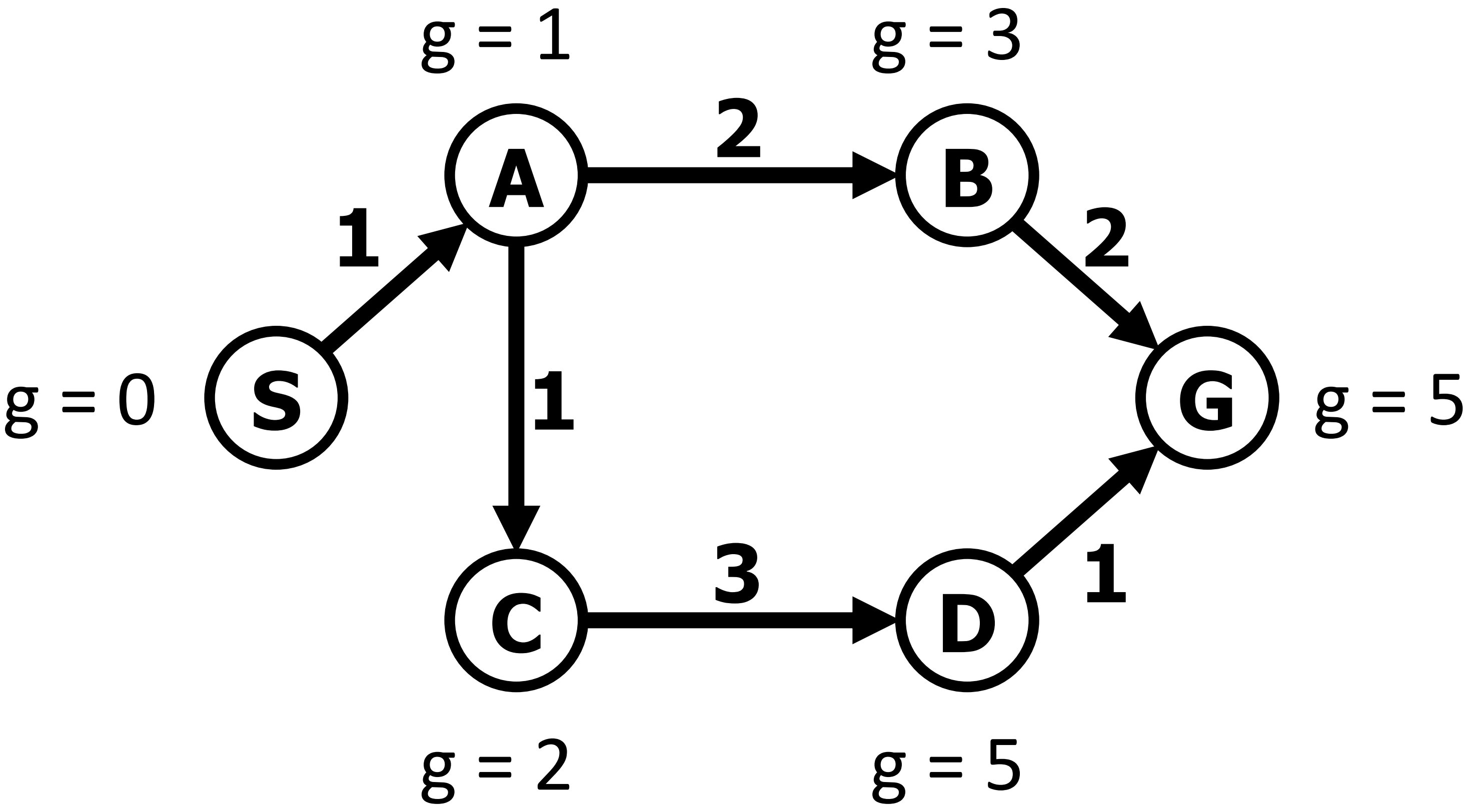
- OPEN: priority queue of nodes that may be expanded (with priority f)
- CLOSED: set of nodes that have been expanded
- g(s): estimated minimum cost from start to node s ("cost-to-come")

Best-First Search Implementation

Initialize g(start) = 0 and all other g-values to infinity
Insert start into OPEN
While goal not in CLOSED
  Remove s with smallest f(s) from OPEN
  Add s to CLOSED
  For every neighbor s'
    If g(s) + c(s, s') < g(s'), update g(s') and add s' to OPEN (with parent s)

Dijkstra's Shortest Path Algorithm

Best-first search with f(s) = g(s)
Only expands nodes with lower cost-to-come than goal!



(WIKIPEDIA)

Class Outline

State Estimation

- Robotic System Design
- Filtering
- Localization
- SLAM

Control

- Feedback Control
- PID Control
- MPC
- LQR

Planning

- Search
- Heuristic Search
- Motion Planning
- Lazy Search

Learning

- Imitation Learning
- Policy Gradient
- Actor-Critic
- Model-Based RL