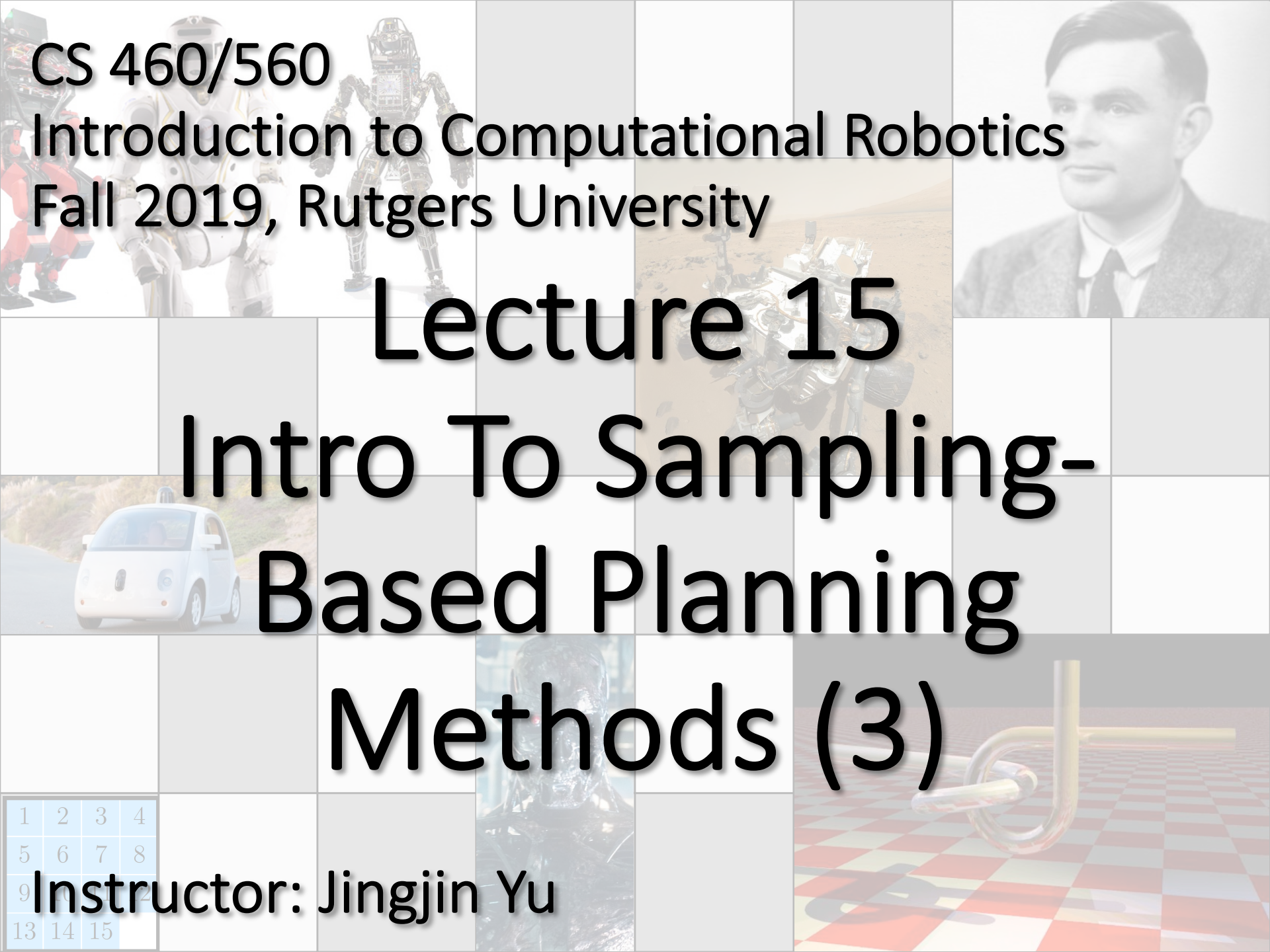


CS 460/560
Introduction to Computational Robotics
Fall 2019, Rutgers University

Lecture 15
Intro To Sampling-Based Planning Methods (3)

Instructor: Jingjin Yu

Outline (for 3 Lectures)

Drawbacks of combinatorial motion planning methods

Probabilistic roadmap (PRM) introduction

Components of sampling-based motion planning methods

- ⇨Sampling
- ⇨k-d tree and nearest neighbor search
- ⇨Distance metric
- ⇨Collision detection

PRM in more detail

A new notion of completeness

Rapidly-exploring random trees (RRT)

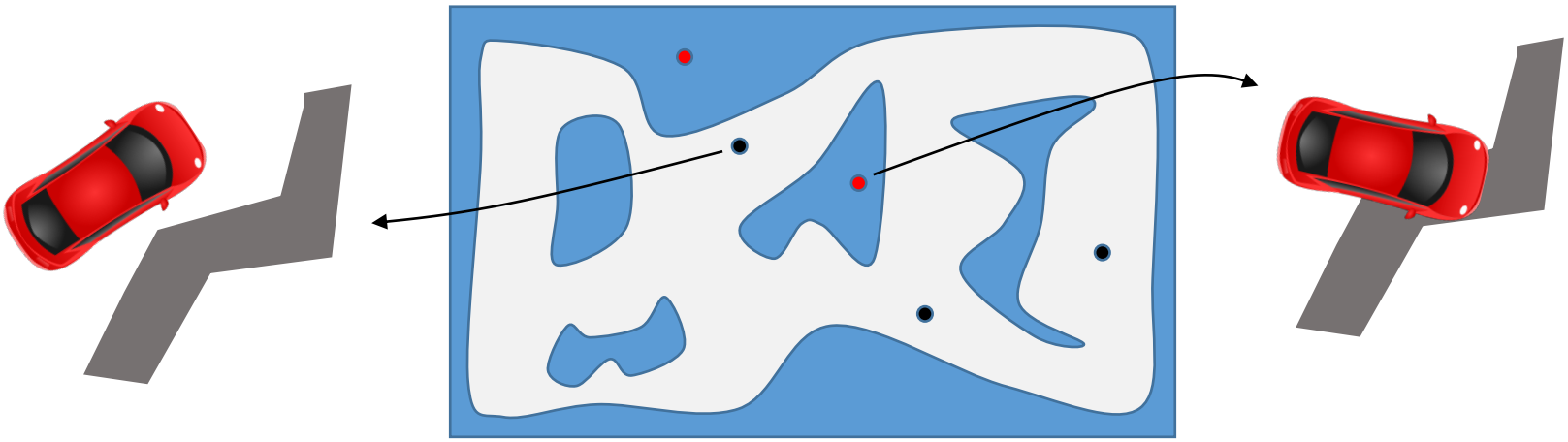
When would sampling-based method work well?

Optimality issues

Key Components of Sampling-Based Planning

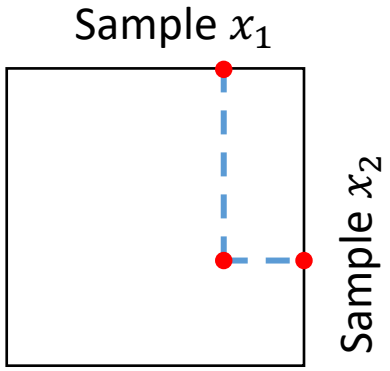
Sampling-based planning requires several important subroutines

- ⇨An **efficient sampling routine** is needed to generate the samples. These samples should **cover** C_{free} well in order to be effective
- ⇨**Efficient nearest neighbor search** is necessary for quickly building the roadmap: for each sample in C_{free} we must find its k-nearest neighbors
- ⇨The neighbor search also requires a **distance metric** to be properly defined so we know the distance between two samples
  - ⇨ This can be tricky for certain spaces, e.g., $SE(3)$
- ⇨**Collision checking** - Note that C_{free} is not computed explicitly so we actually are checking collisions between a complex robot and a complex environment



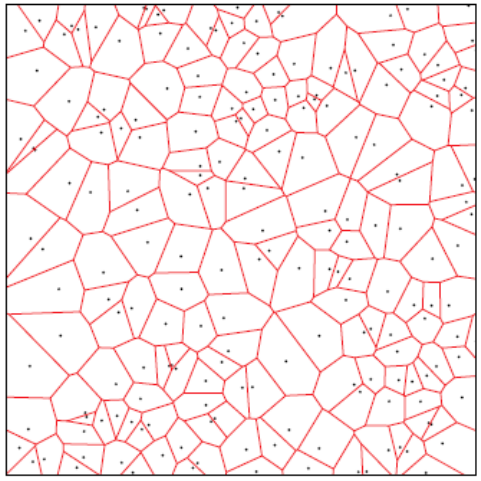
Sampling Routine

The simplest way of achieving this: **uniformly random sampling**

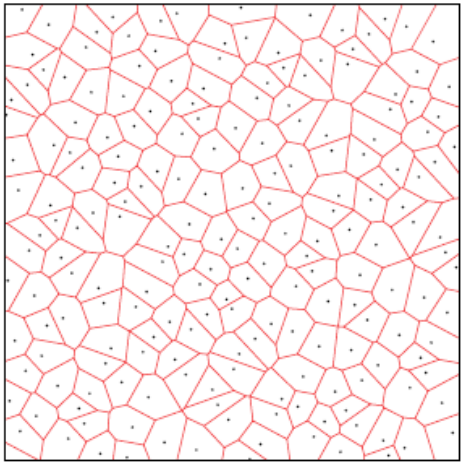


➡ A sample $(x_1, x_2) \in \mathbb{R}^2$

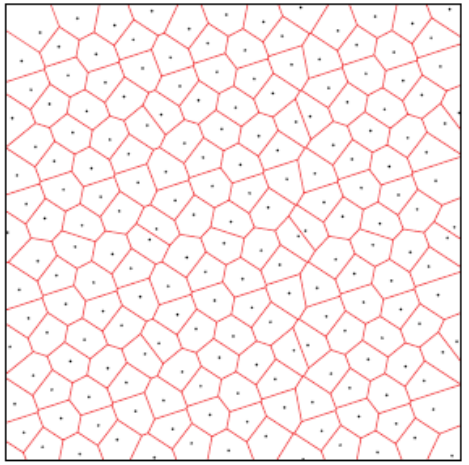
Generally, **incremental, dense** sampling w/ good **dispersion**



(a) 196 pseudorandom samples



(a) 196 Halton points

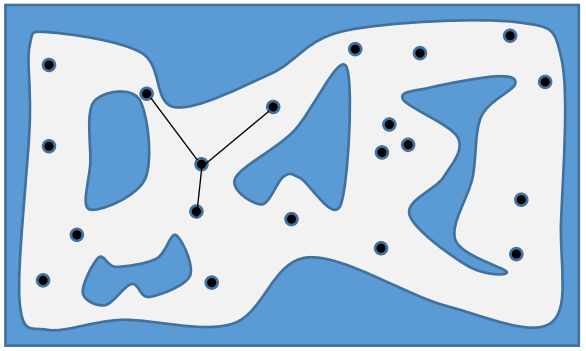


(b) 196 Hammersley points

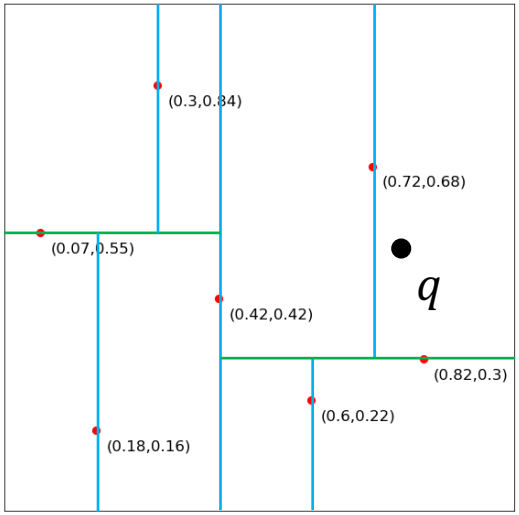
Nearest Neighbor Search w/ k-d Tree

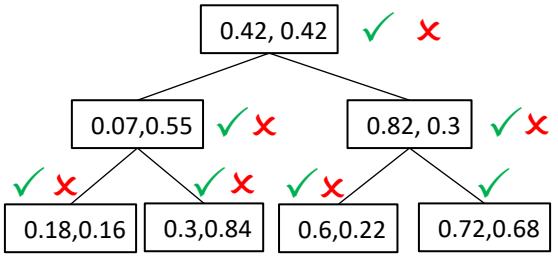
Connecting the samples

- ⇨Building the graph requires connecting the samples
- ⇨Need efficient methods for this



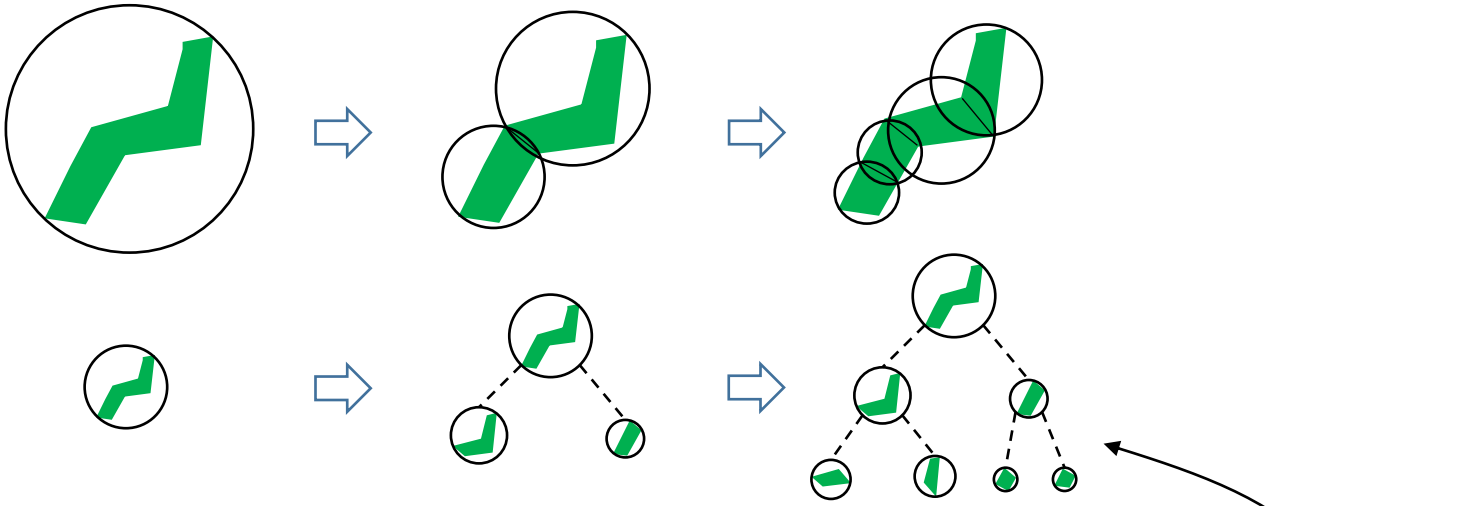
k-d Tree





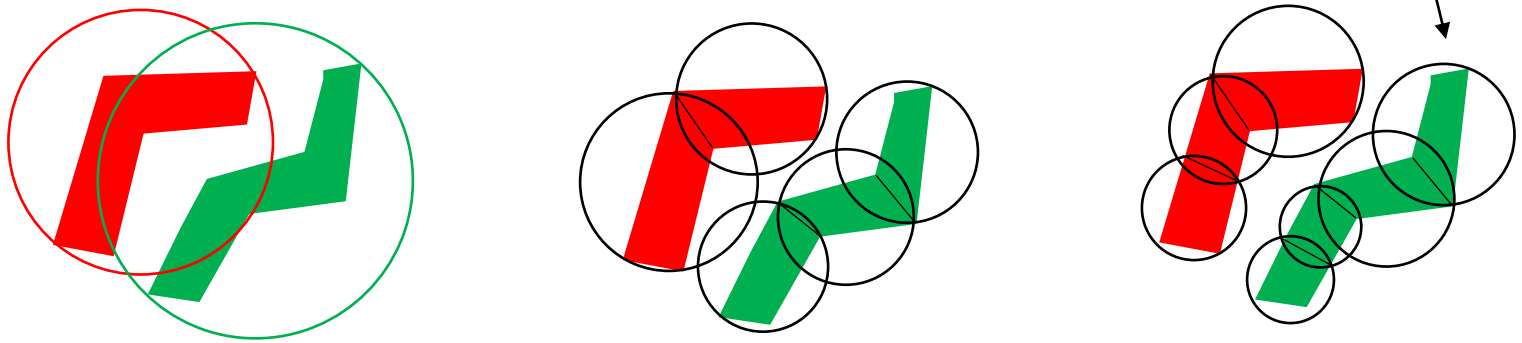
BVH for Collision Checking

BVH (Bounded Volume Hierarchy) breaks complex objects into pieces

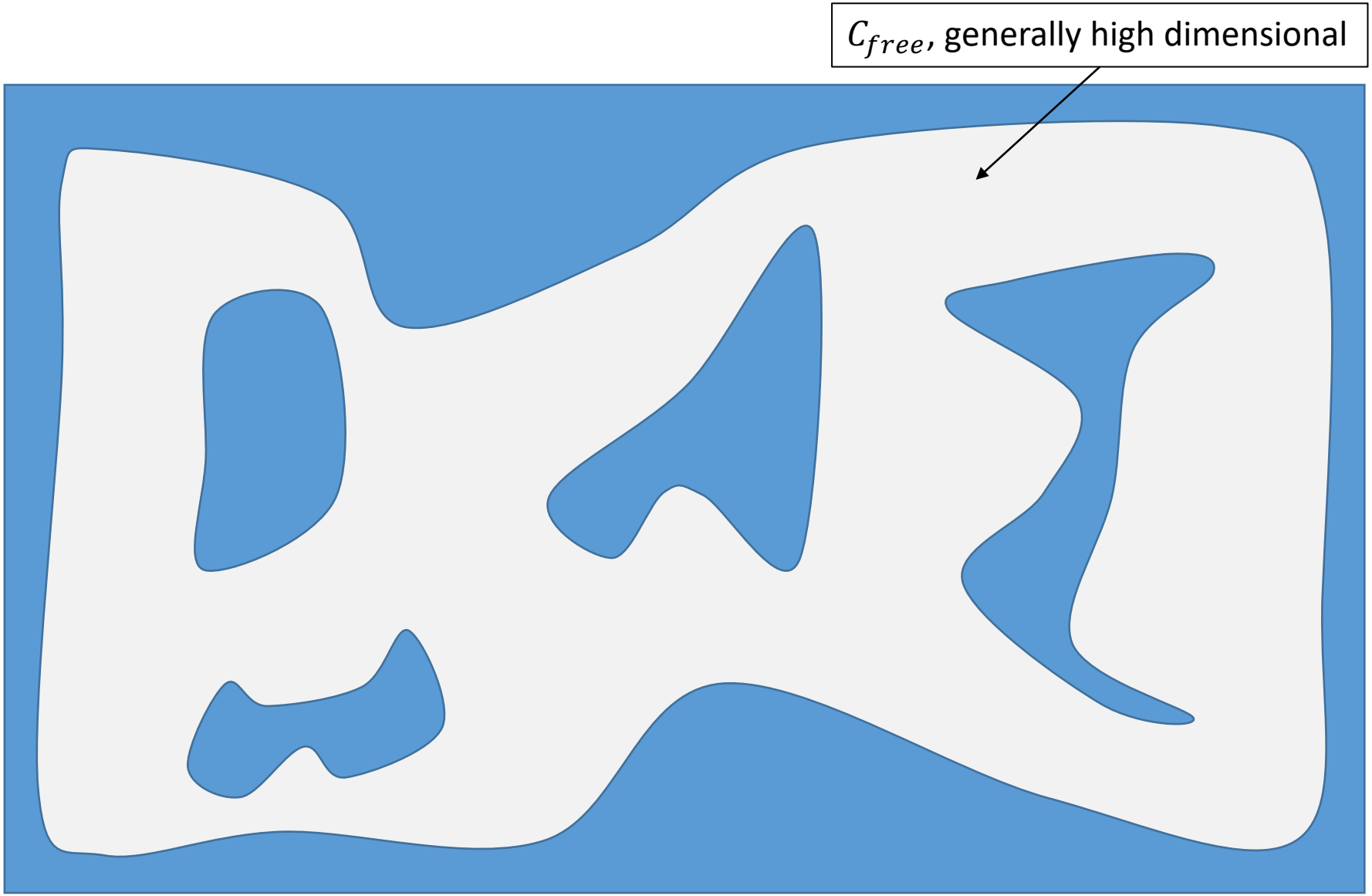


For collision checking, it works with two BVHs

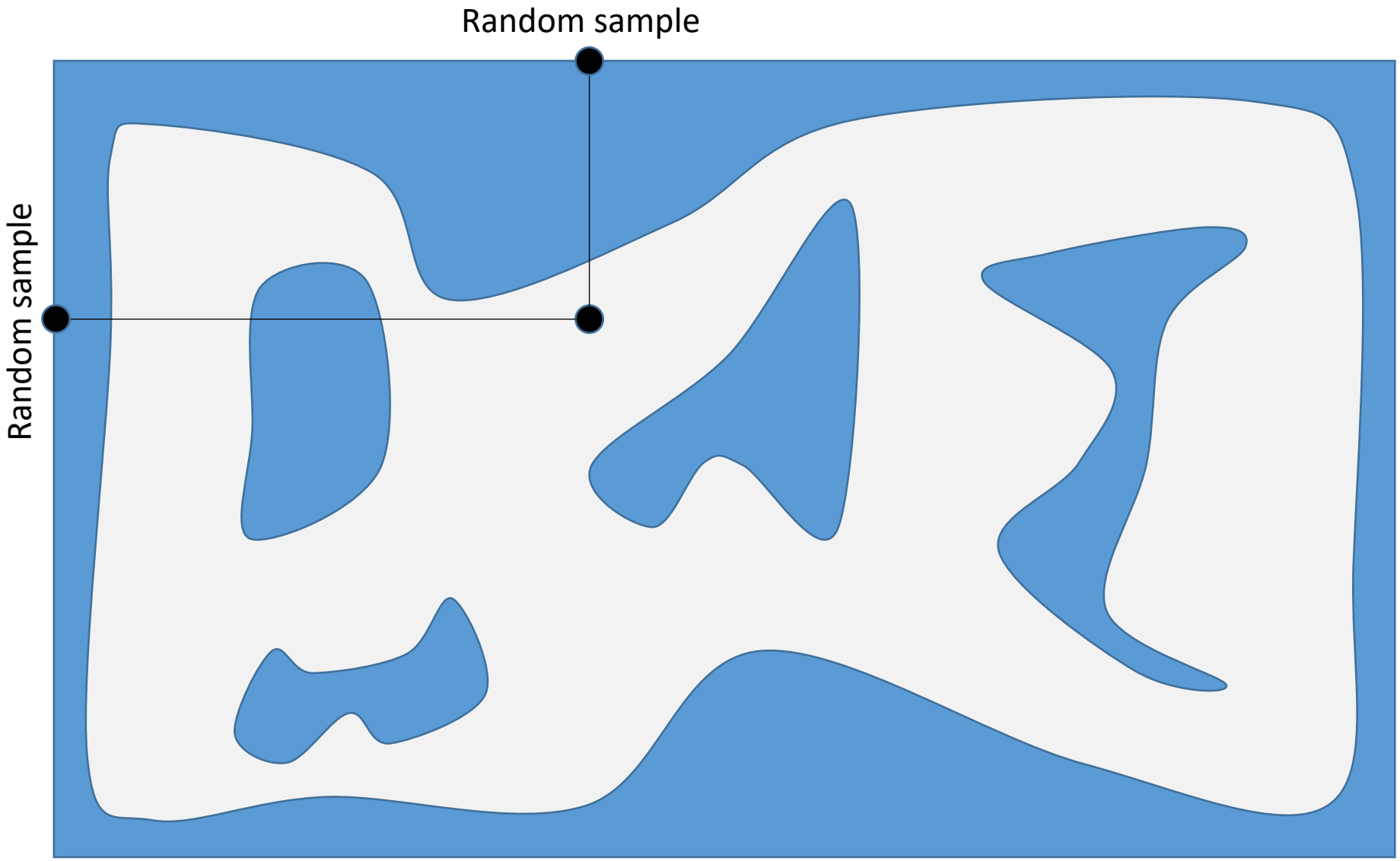
- ⇨BVs collide ⇒ possible collision
- ⇨BVs not colliding ⇒ no collision



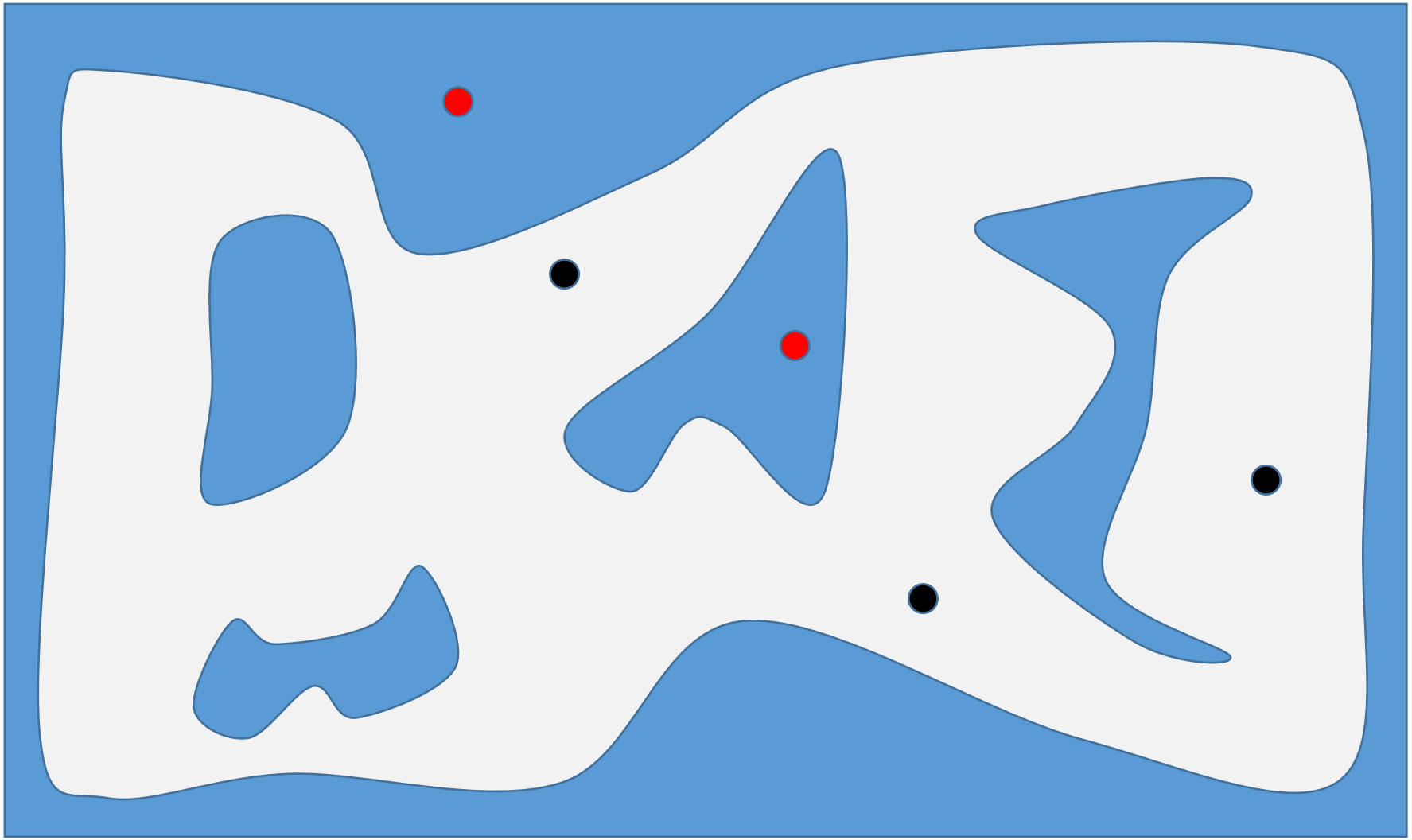
Probabilistic Roadmap in More Detail



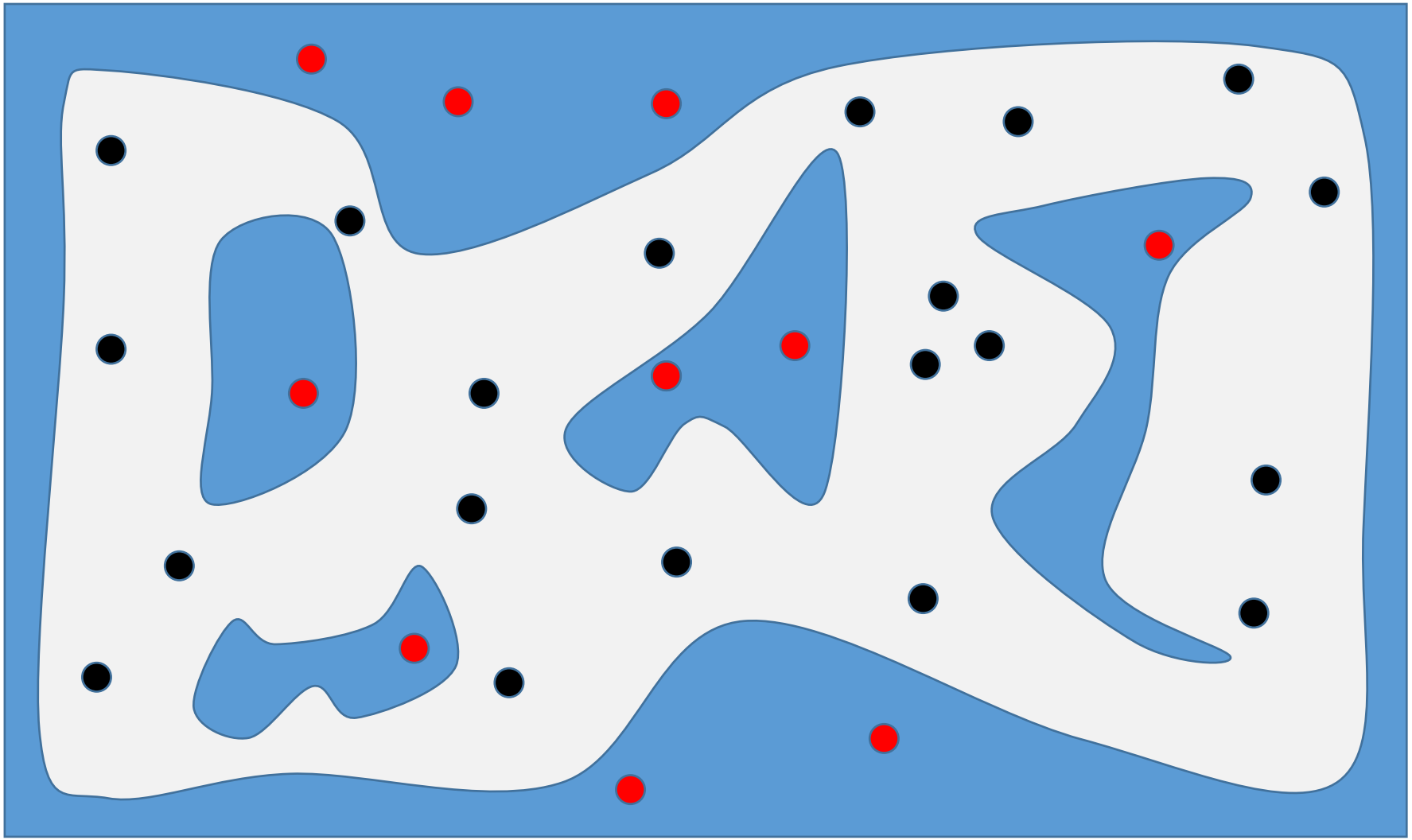
Generating Random Samples



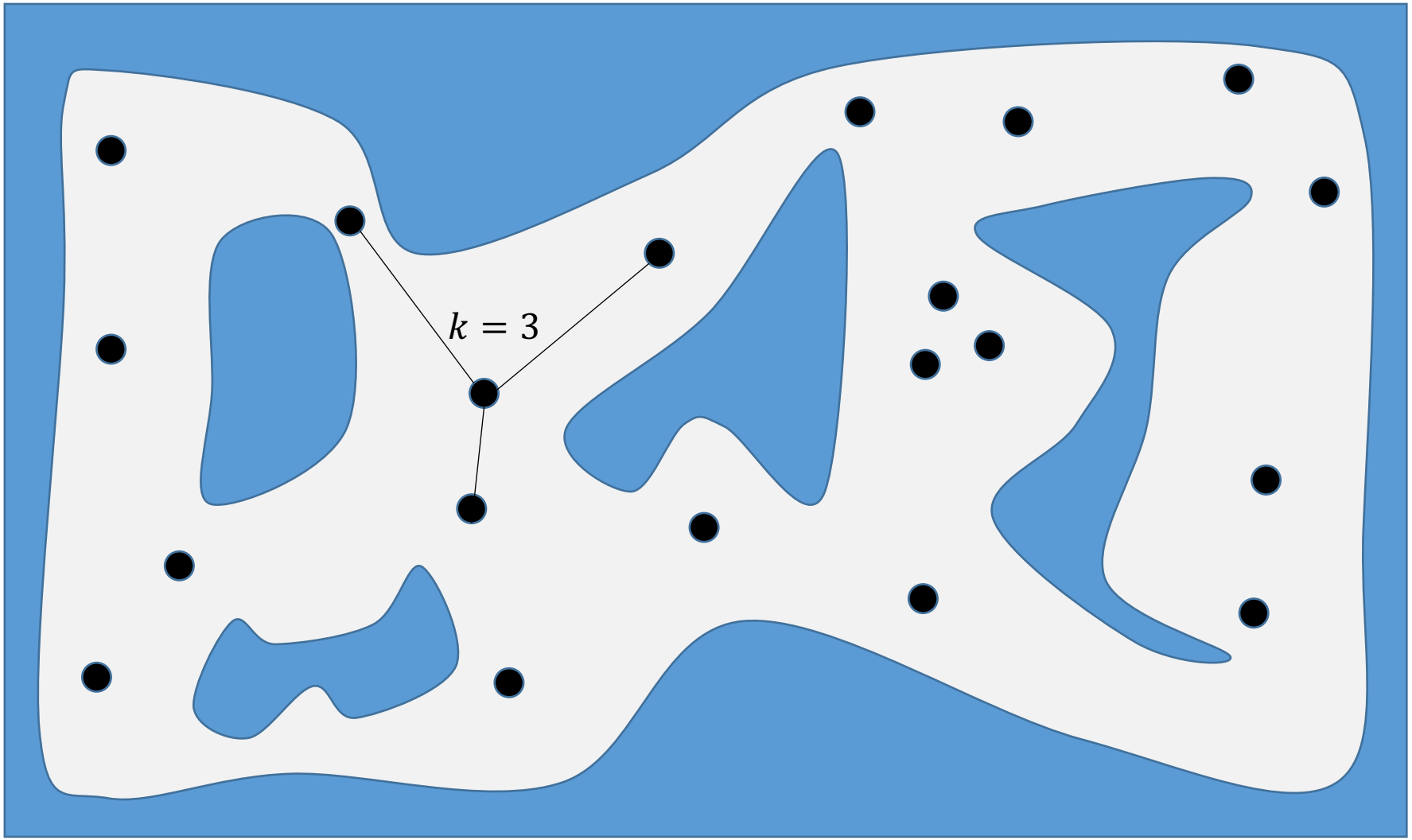
Rejecting Samples Outside C_{free}



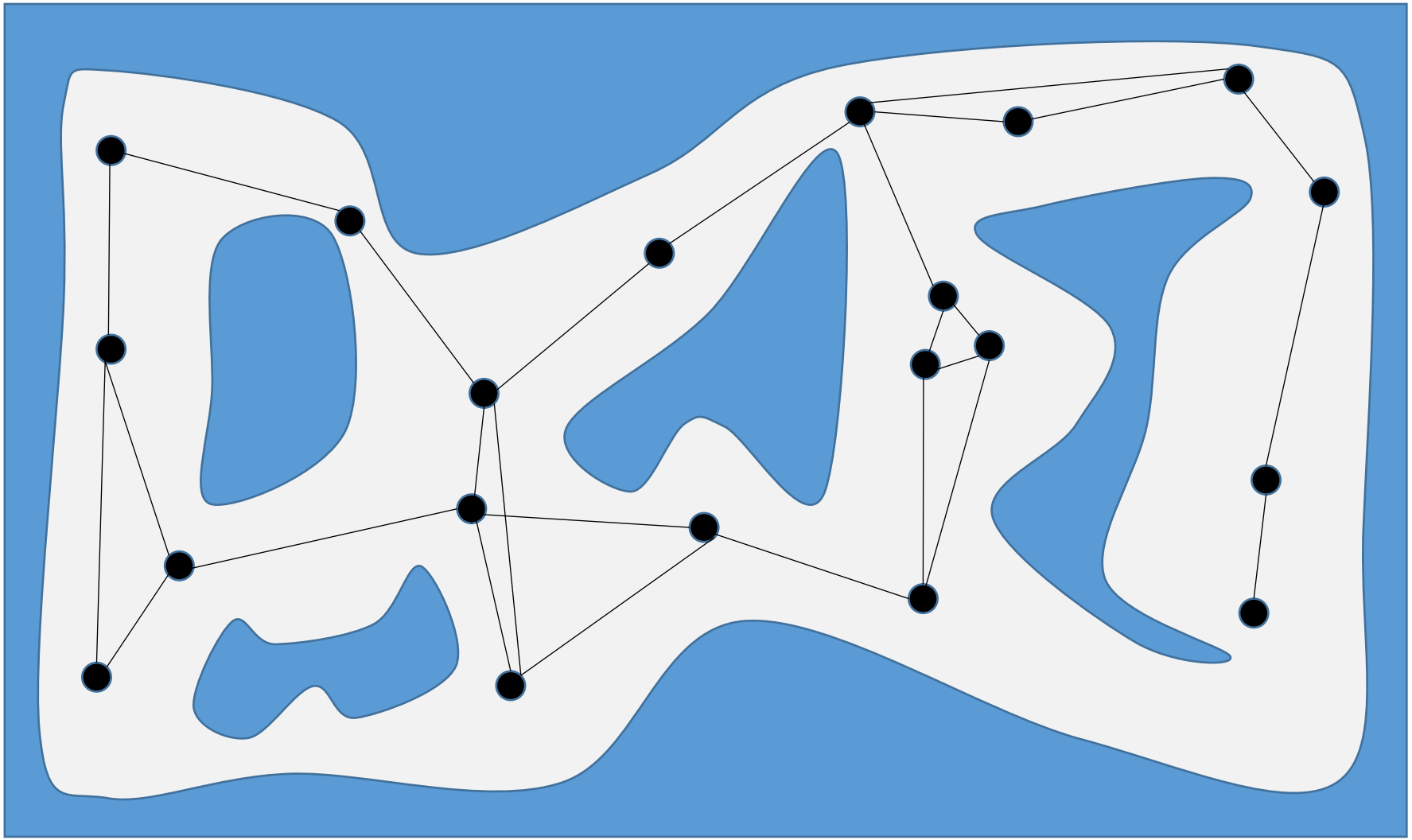
Collecting Enough Samples in C_{free}



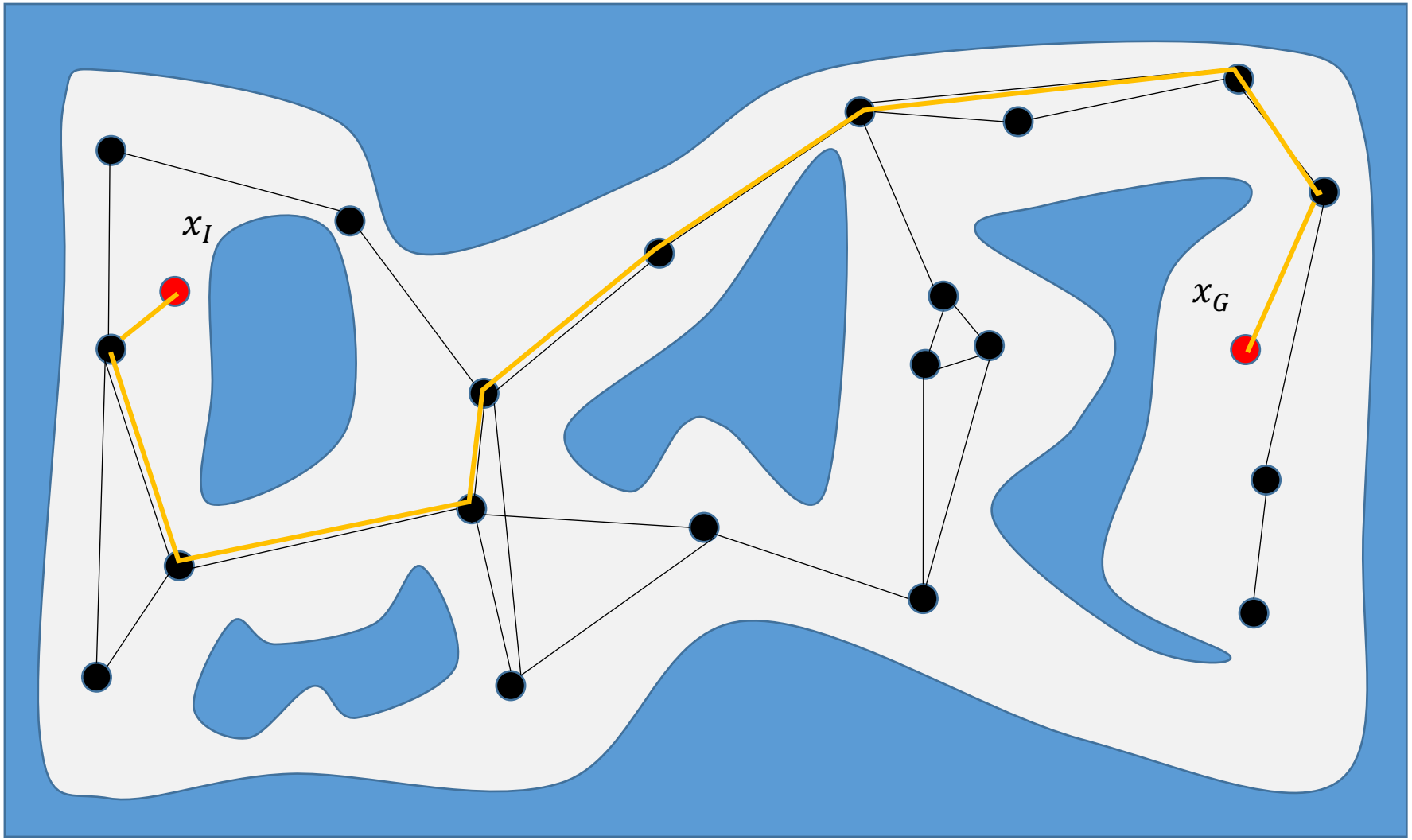
Connect to k Nearest Neighbors (If Possible)



Connect to k Nearest Neighbors (If Possible)

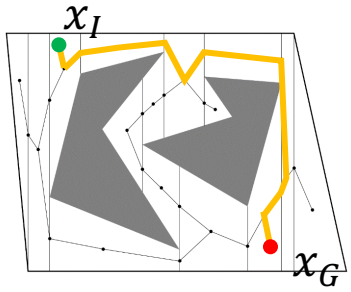


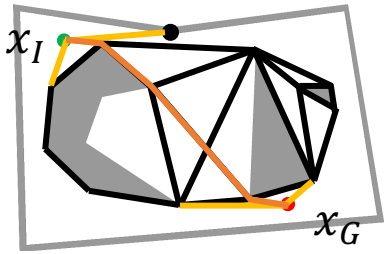
Query Phase

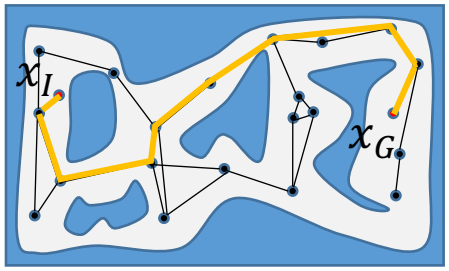


Some Notes

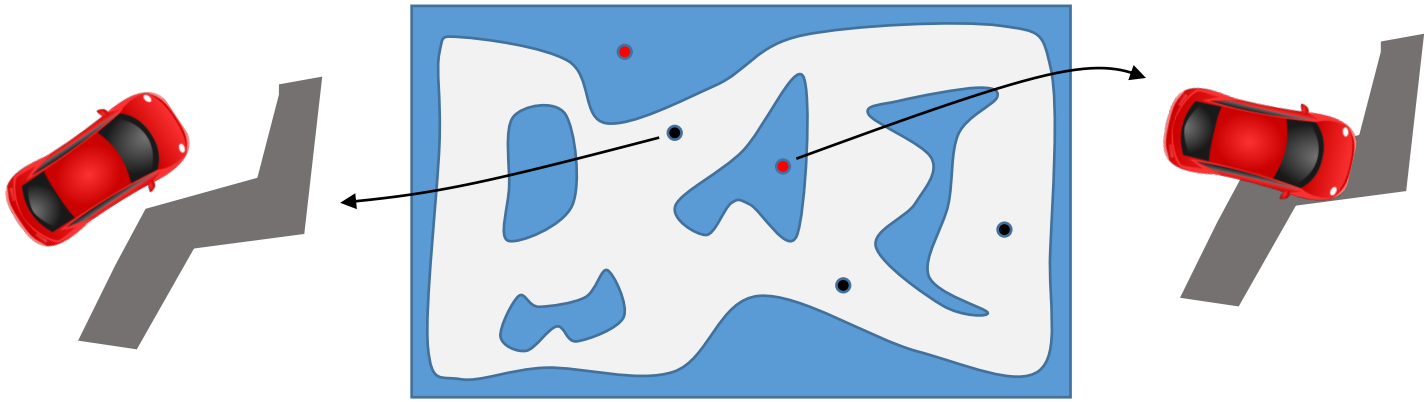
- Main planning methods work on the **configuration space**
  - Combinatorial methods
  - Sampling-based methods







- Collision checking works with 2D or 3D physical space
  - Move a robot to the sampled pose/configuration (rigid body transformation)
  - Then check for collision in 2D (e.g., car) or 3D (e.g., quadcopter)

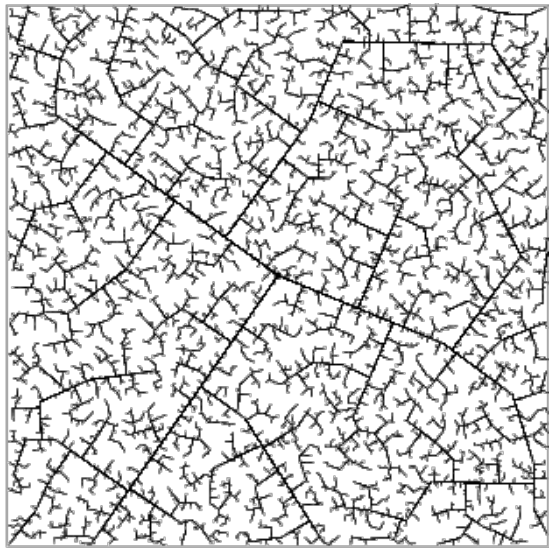


Drawbacks of Multi-Query Methods

PRM is known as a “multi-query” sampling-based method because after initial roadmap is built, multiple queries can be executed on the same roadmap

- ⇨But, this also means that the roadmap is likely to have a lot of useless information stored if we want to run a single query
- ⇨People developed **single-query** methods to handle such situations
- ⇨One method is the rapidly-exploring random trees (RRT, by LaValle & Kuffner)

45 iterations



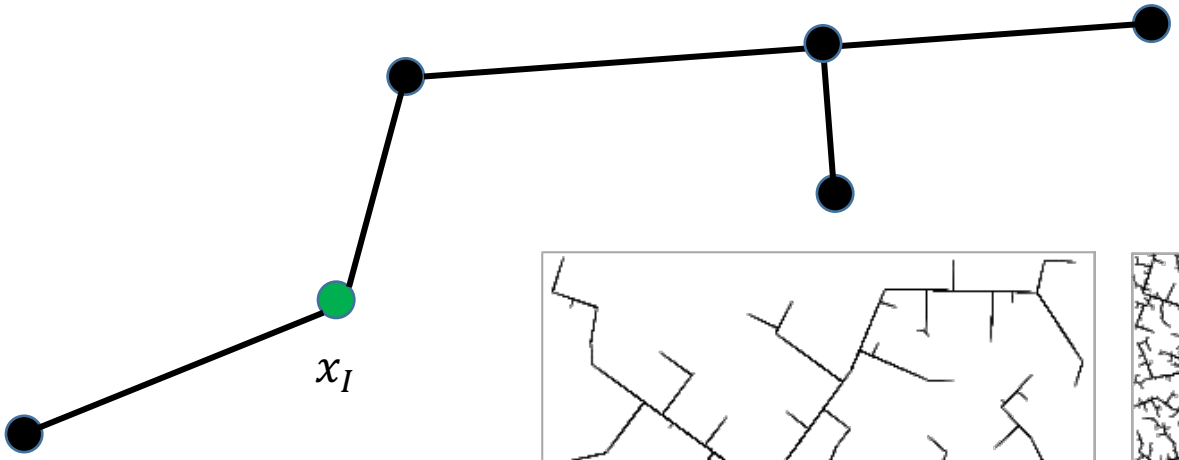
2345 iterations

Image sources: Planning Algorithms by LaValle

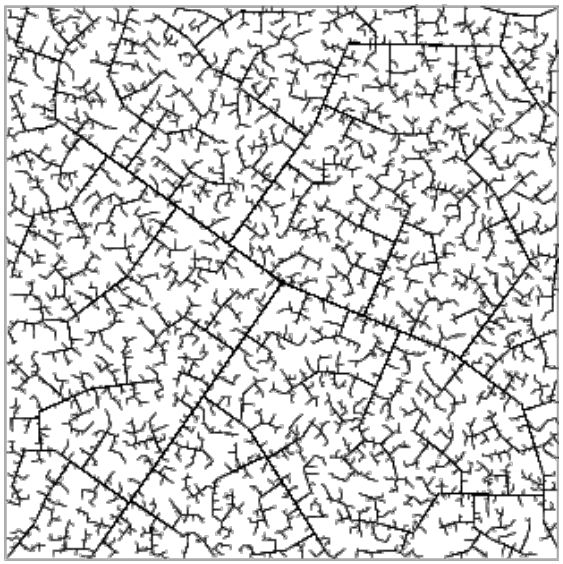
Rapidly-Exploring Random Trees w/o Obstacle

RRT without obstacle simply grows a tree from a point

⇨Basically, tries to connect new points to the closest part of the existing tree



45 iterations

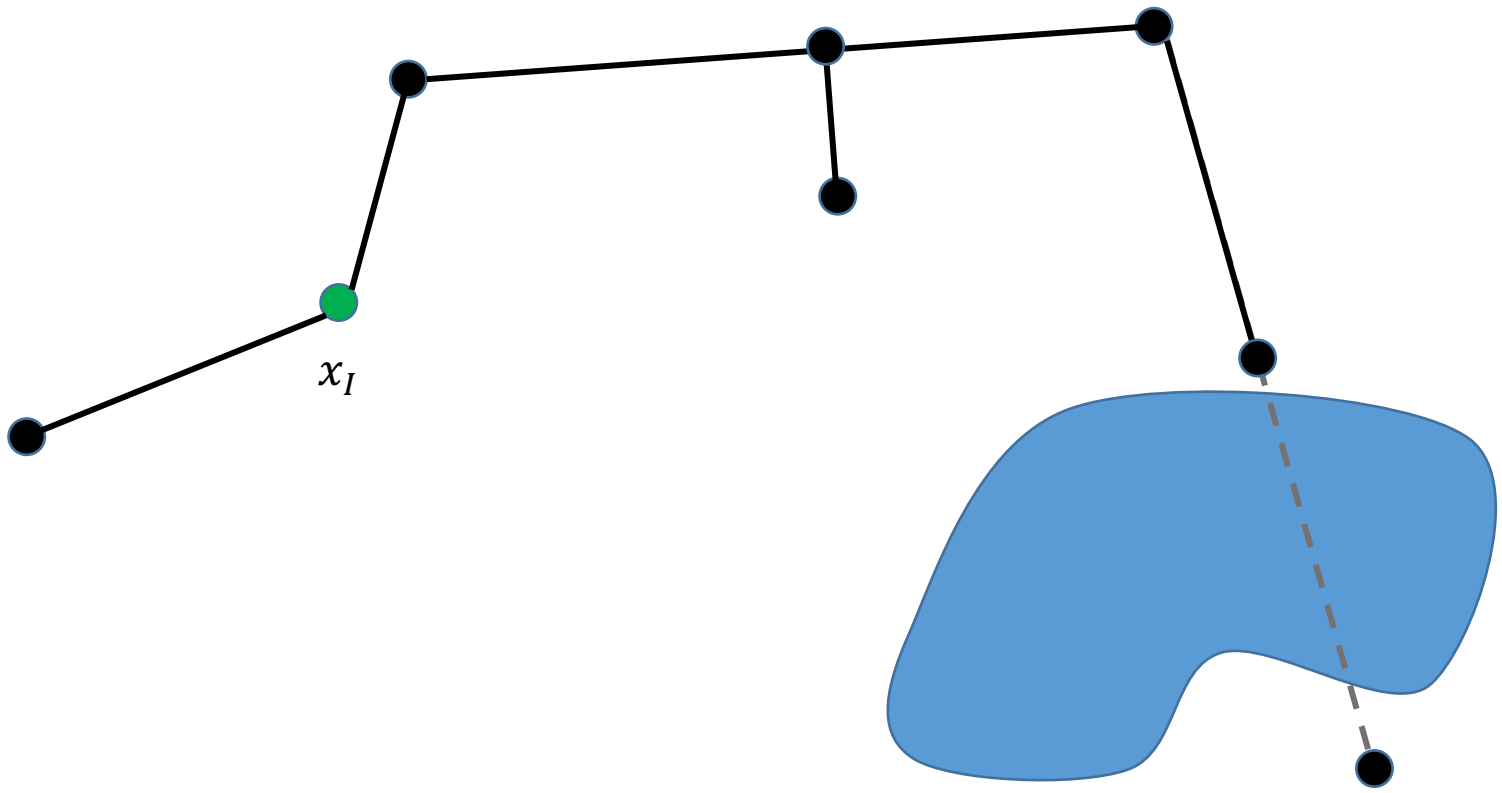


2345 iterations

Image sources: Planning Algorithms by LaValle

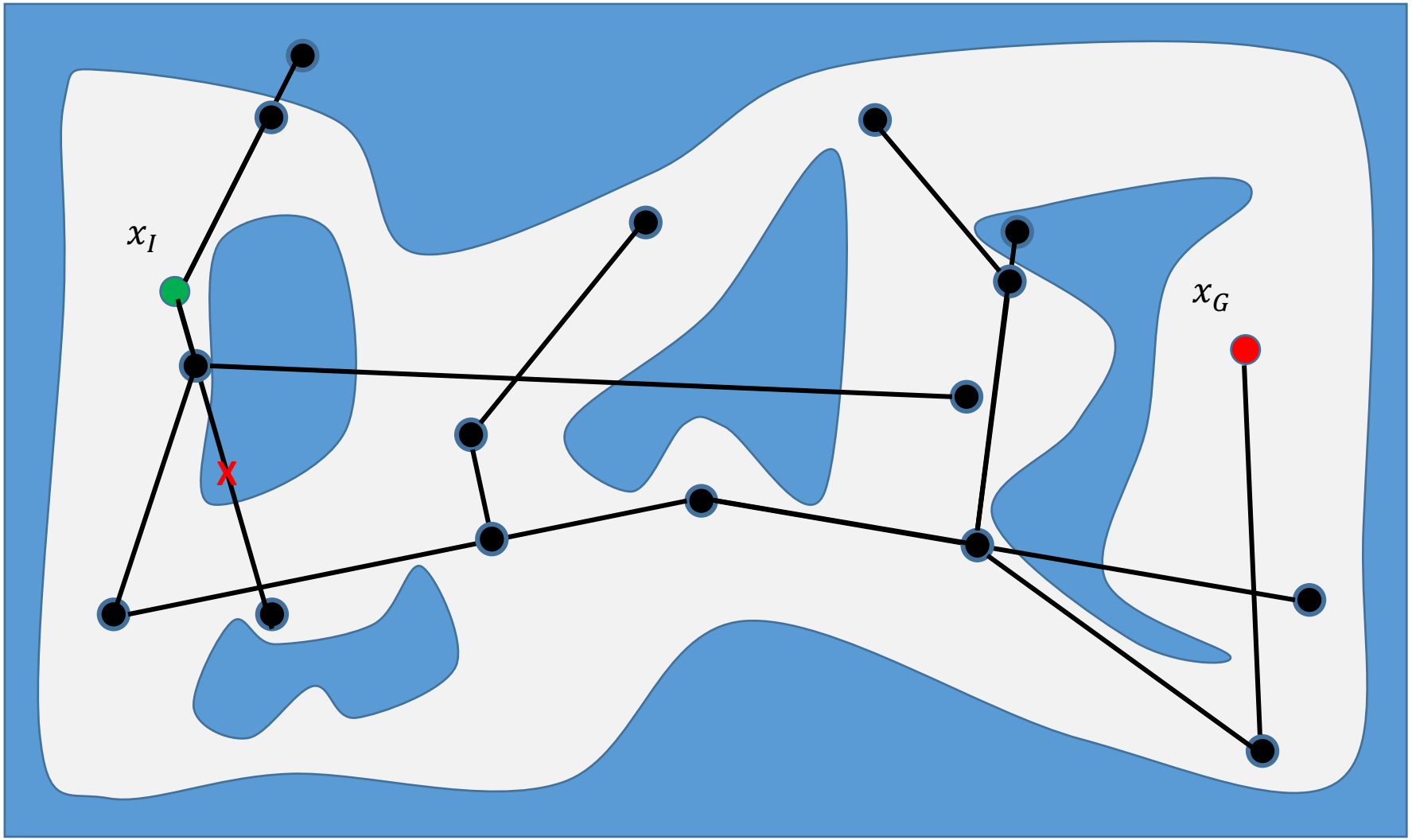
RRT with Obstacles

When there are obstacles, try to extend the tree as much as possible



Same procedure if sample falls inside an obstacle

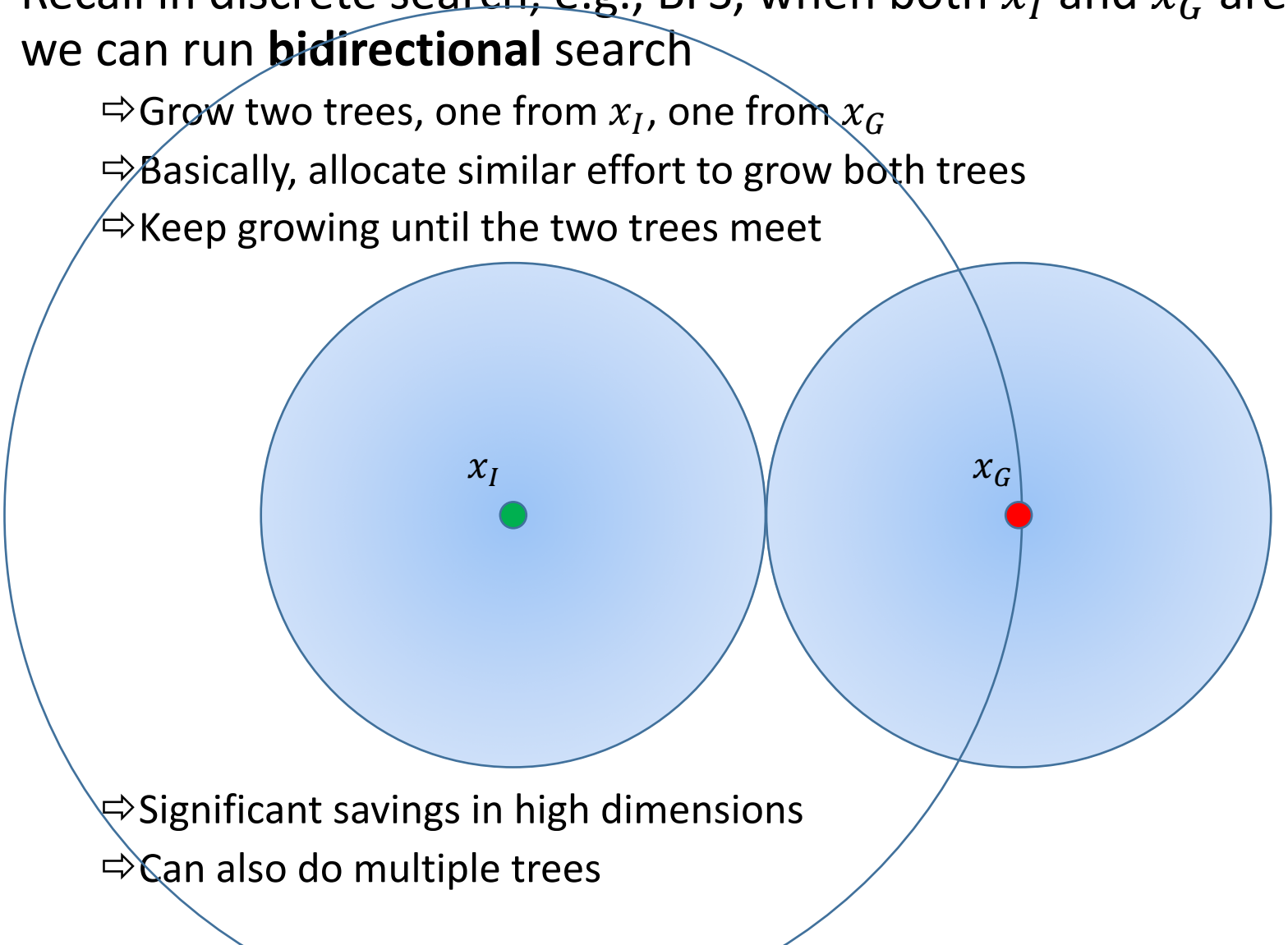
Tree Building Example



Improving Efficiency with Bidirectional Search

Recall in discrete search, e.g., BFS, when both x_I and x_G are known, we can run **bidirectional** search

- ⇨Grow two trees, one from x_I, one from x_G
- ⇨Basically, allocate similar effort to grow both trees
- ⇨Keep growing until the two trees meet



- ⇨Significant savings in high dimensions
- ⇨Can also do multiple trees

Difficult Cases

Recall that sampling based methods work well when the problems are relatively easy to solve. Sometimes, problems are just hard.

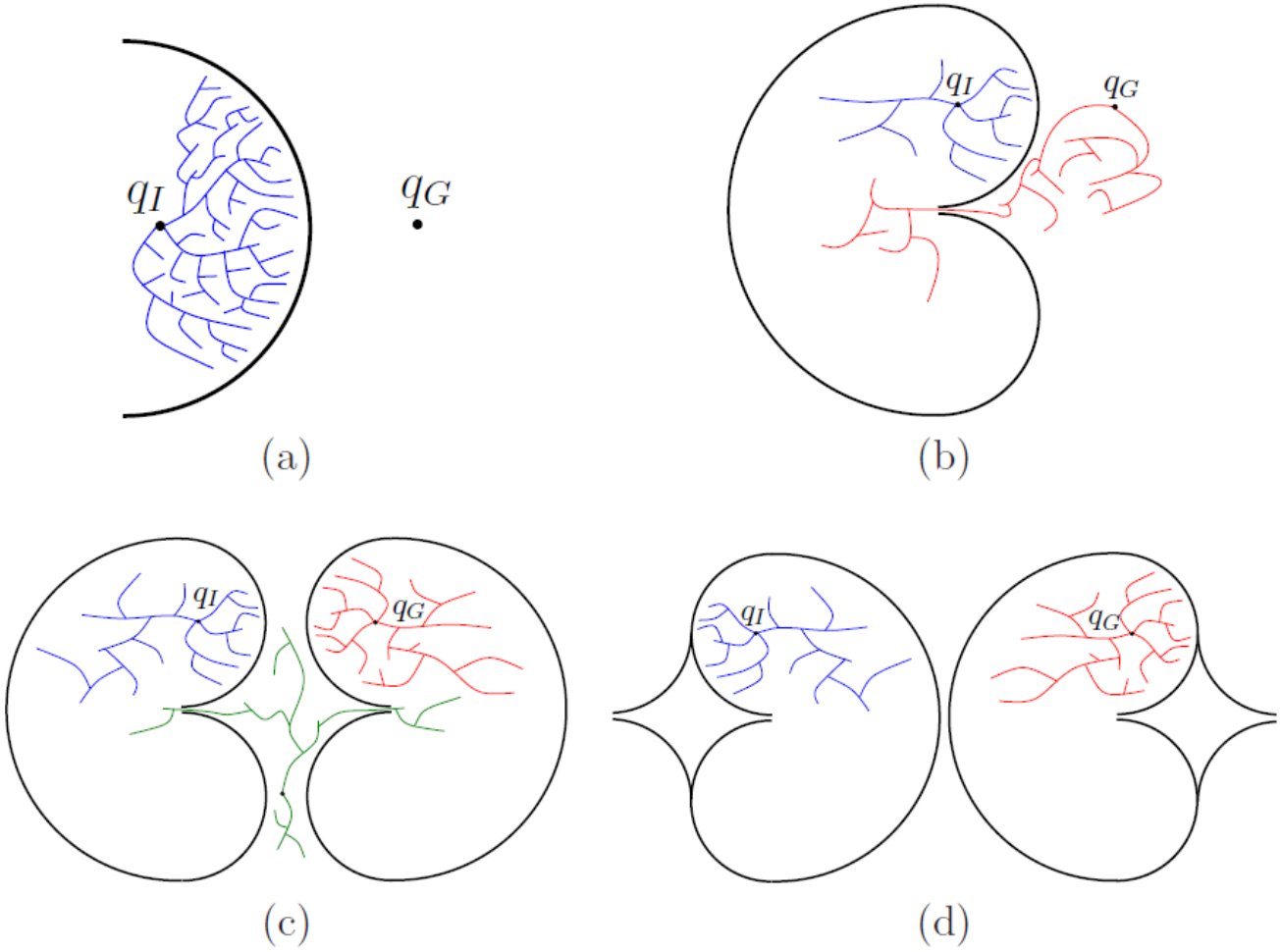
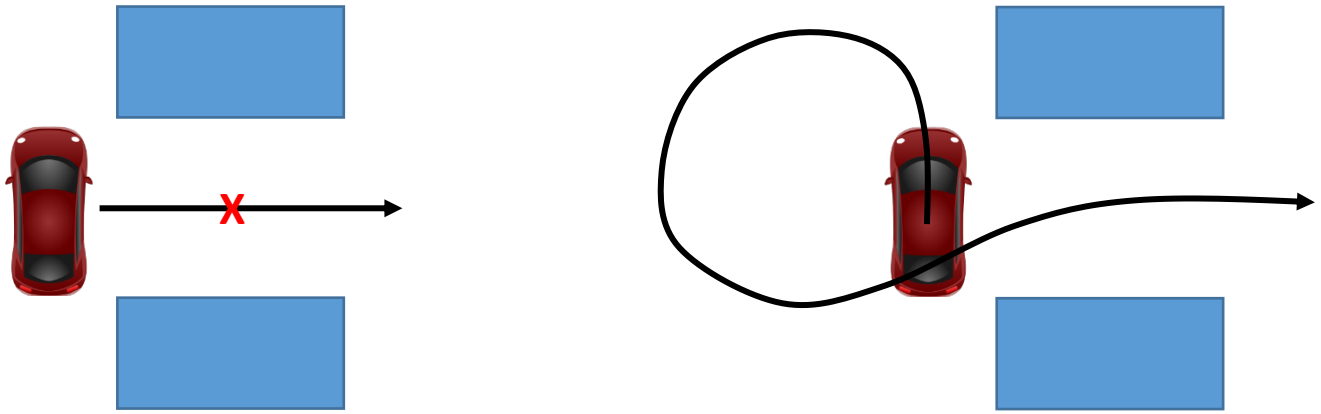


Image sources: Planning Algorithms by LaValle

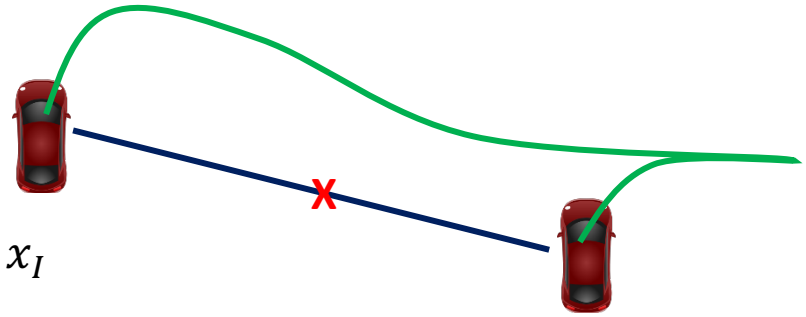
Solving Kinodynamic Problems

Besides solving single-query problems faster, RRT is suitable for solving problems for systems with **differential constraints**

⇨For example, a normal car is such a system



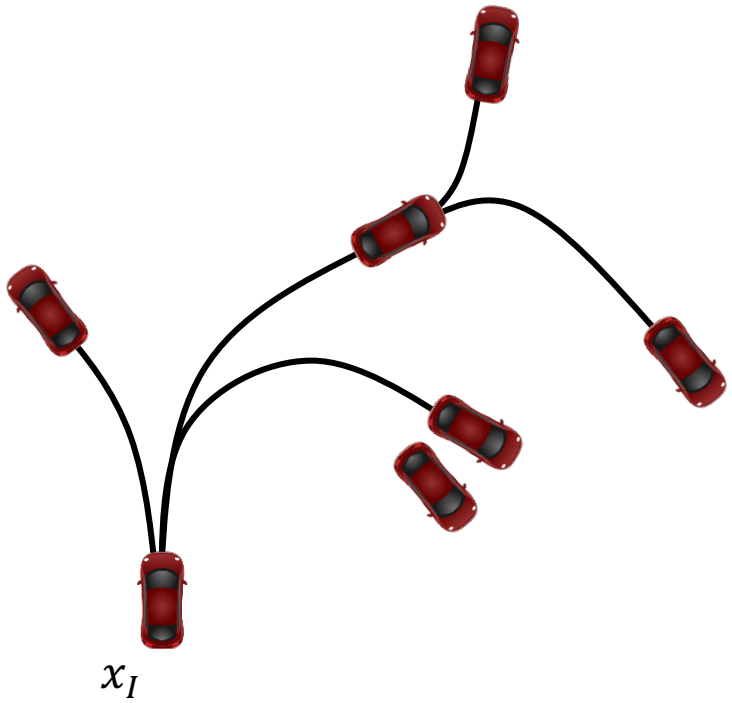
⇨Standard PRM and RRT cannot be applied!



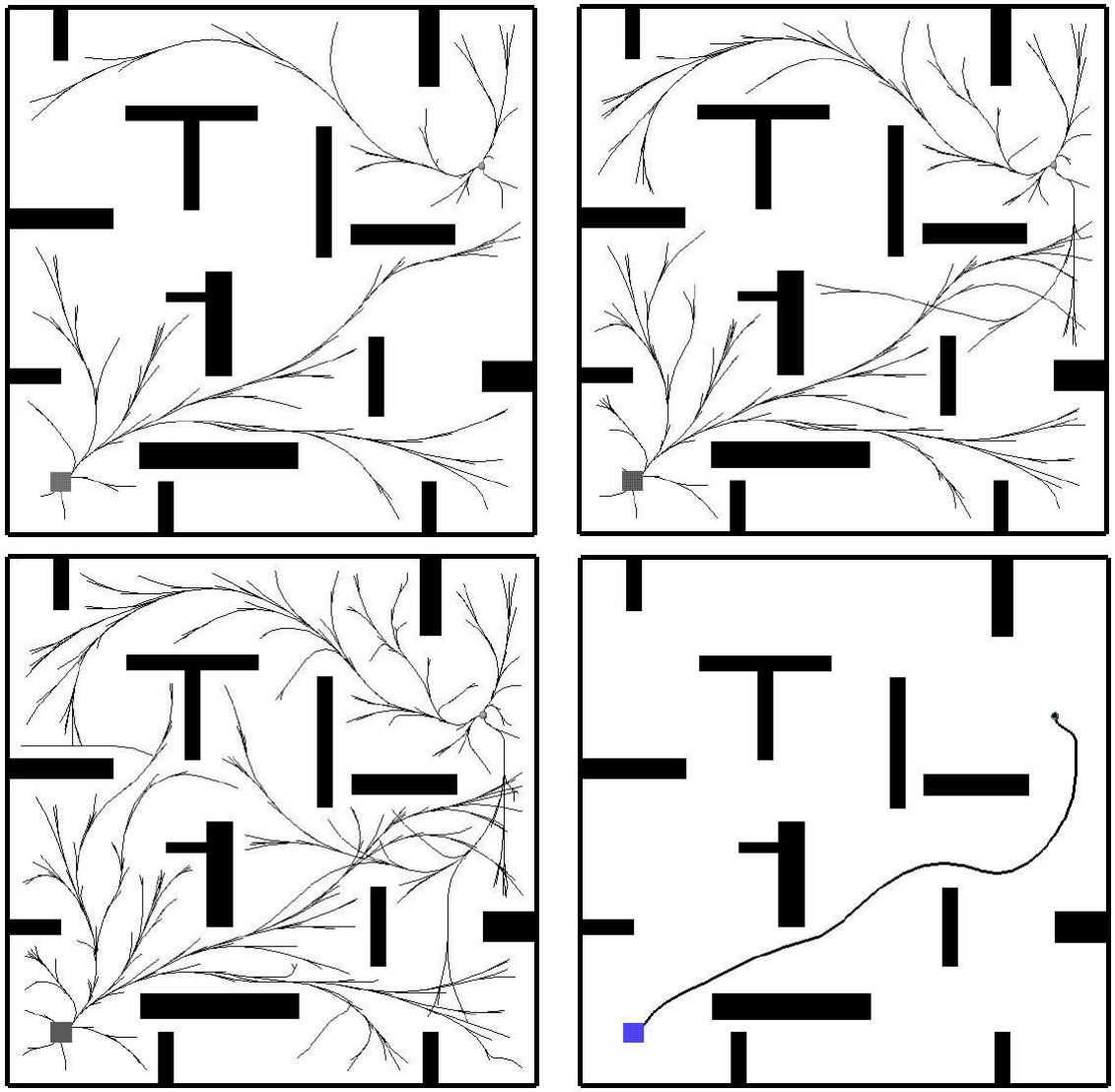
Kinodynamic RRT

We can grow an RRT respecting the differential constraints

- ⇨Standard PRM and RRT cannot be applied!
- ⇨Need to compute path more carefully
  - ⇨ Needs to solve a boundary value problem (differential equations)
- ⇨Example w/o obstacles

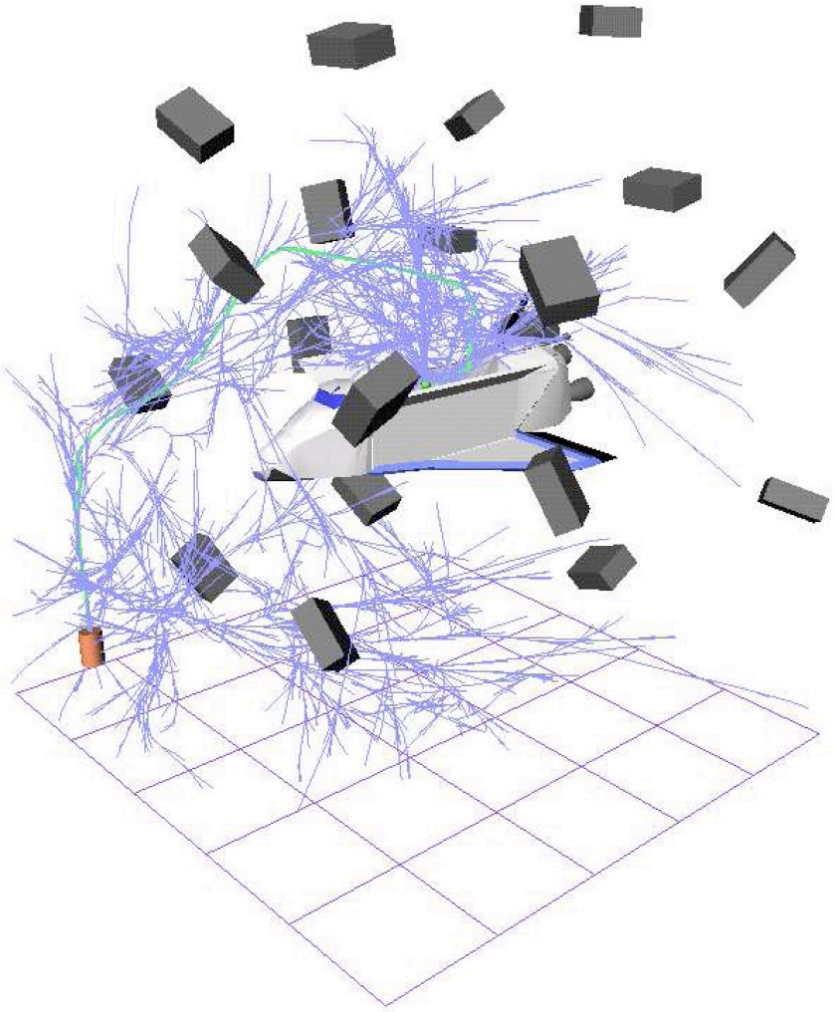


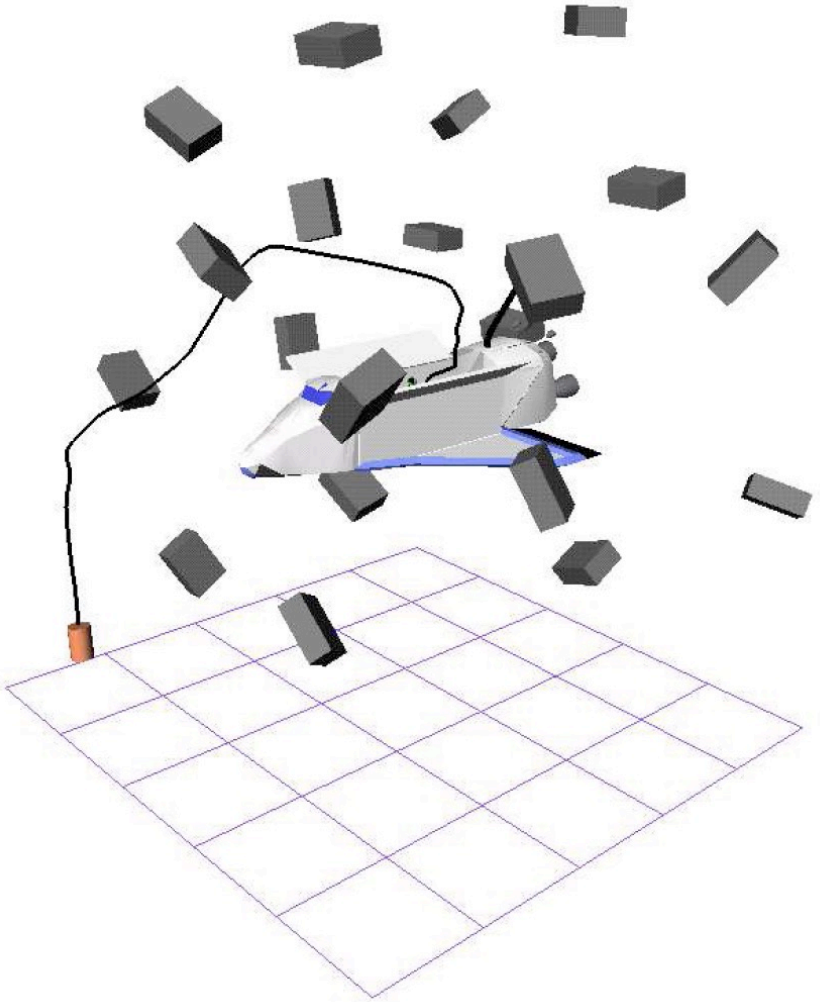
Kinodynamic RRT Examples



RRT after 500, 1000, 1582 nodes

Kinodynamic RRT Examples, Continued

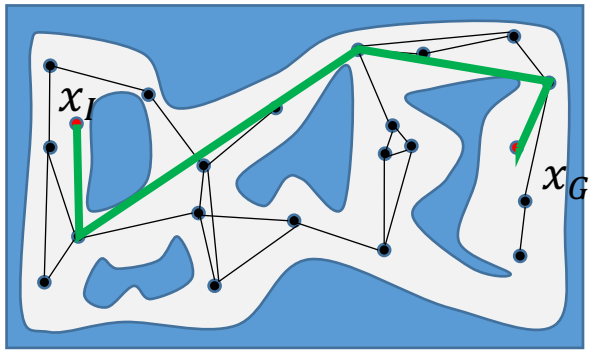




When Does PRM and RRT Work Well?

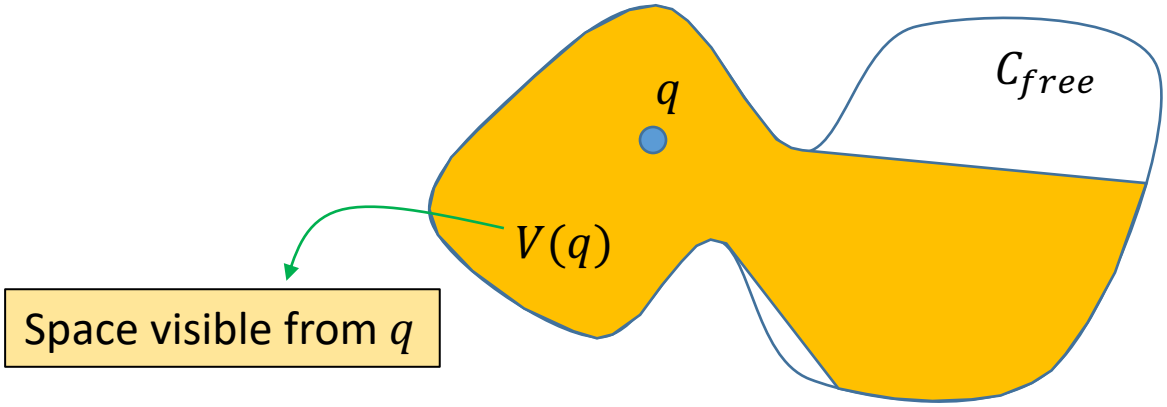
PRM tends to work well for reasonably high-dimensional problems. How comes throwing some points in C_{free} and connect them simply work?

- ⇨This means the problems are “easy” in a sense
- ⇨Such environments are known as “expansive”
- ⇨More precisely, $(\varepsilon, \alpha, \beta)$-expansive



ε-goodness:

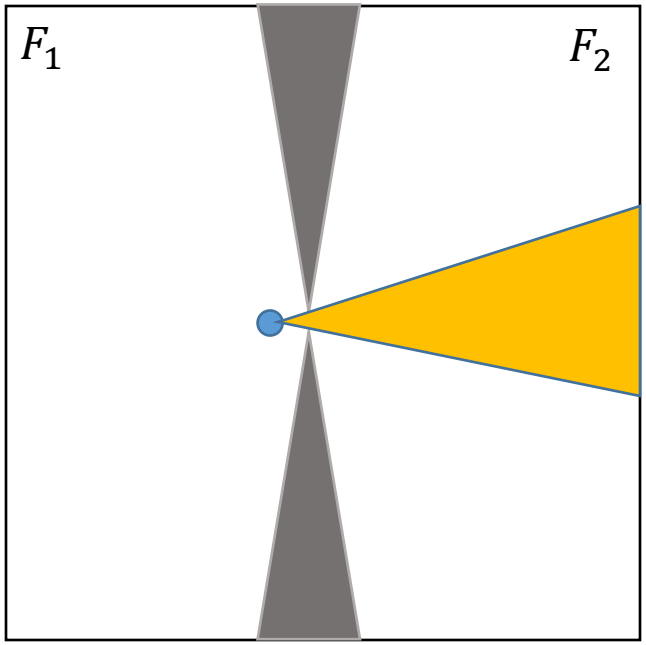
- ⇨$F = C_{free}$ is ε-good if given any point $q \in F$, $\mu(V(q)) \geq \varepsilon\mu(F)$
- ⇨Here, $\mu(\cdot)$ measures the volume of a region



When Does PRM and RRT Work Well? Cont.

β-lookout

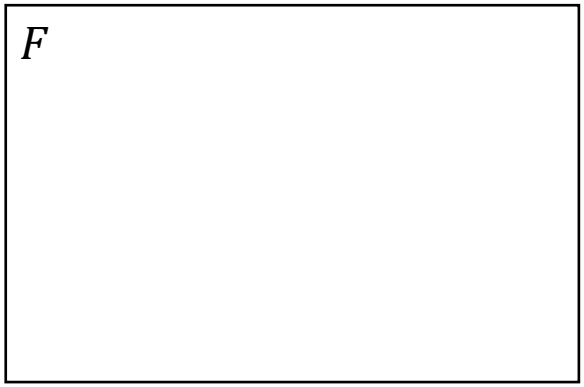
⇨The β-lookout of a subset (e.g., F_2) is set of all configurations $q \in F \backslash F_2$ such that $\mu(V(q) \cap F_2) \geq \beta\mu(F_2)$



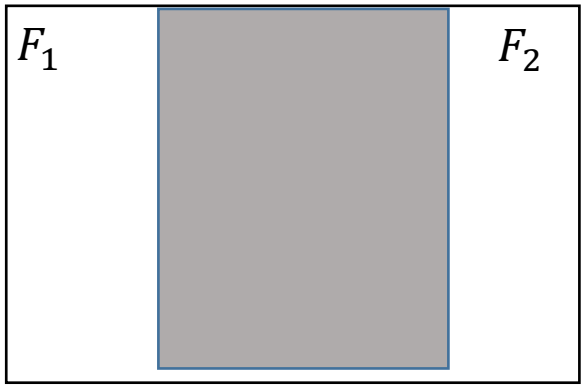
F is $(\varepsilon, \alpha, \beta)$-expansive if it is ε-good and each one of its subsets F_i has a β-lookout with volume at least $\alpha\mu(F_i)$

When Does PRM and RRT Work Well? Cont.

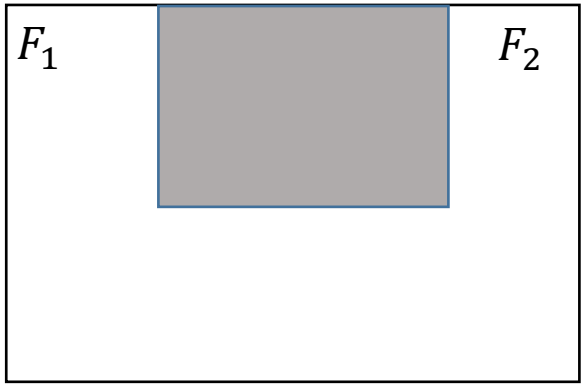
Some examples



$\varepsilon = \alpha = \beta = 1$



$\varepsilon \approx 0.5$, small α and β



Good $\varepsilon, \alpha, \beta$

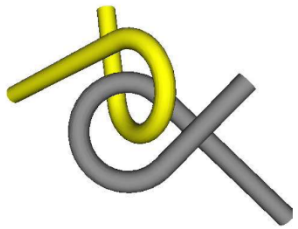
When Does PRM and RRT Work Well? Cont.

Theorem: Let F be $(\varepsilon, \alpha, \beta)$-expansive, and s and g be two configurations in the same component of F. Then BasicPRM(s, g, N) with uniform sampling returns a path connecting s and g with the probability

$$\Pr(fail) \leq \left(\frac{c_1}{\varepsilon\alpha}\right) e^{c_2 \varepsilon\alpha(-N+\frac{c_3}{\beta})}$$

Basically, this says that the if $\varepsilon, \alpha, \beta$ are large, then the PRM problem is "easy".

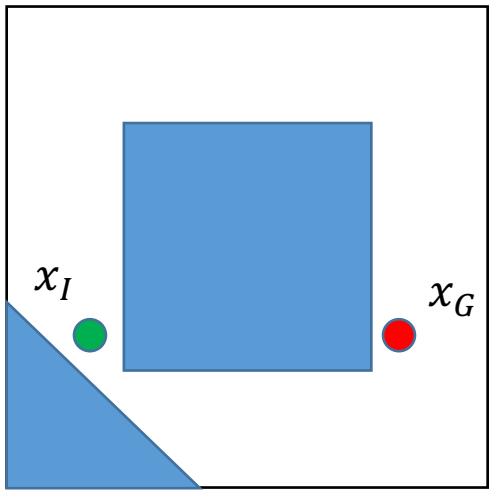
- ⇨This is often the case in practice
- ⇨That is, it is not often the case that there are "narrow passages"
- ⇨Hard instances are sometimes intentionally made



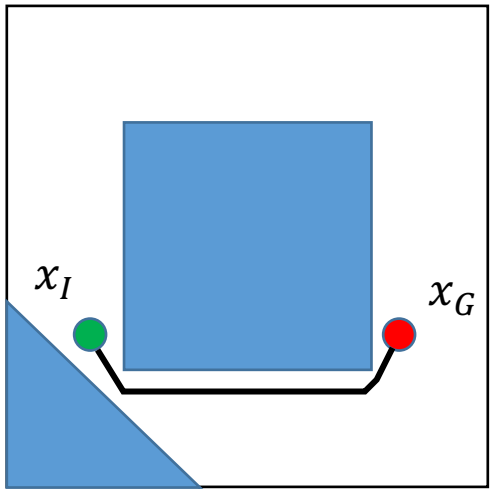
Non-Optimality of PRM and RRT

PRM and RRTs are not optimal

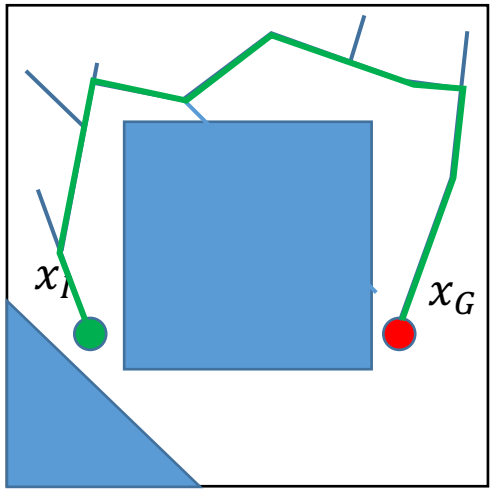
⇨It is possible construct instances to make PRM/RRT produce long paths



Problem



Optimal solution



Likely RRT solution

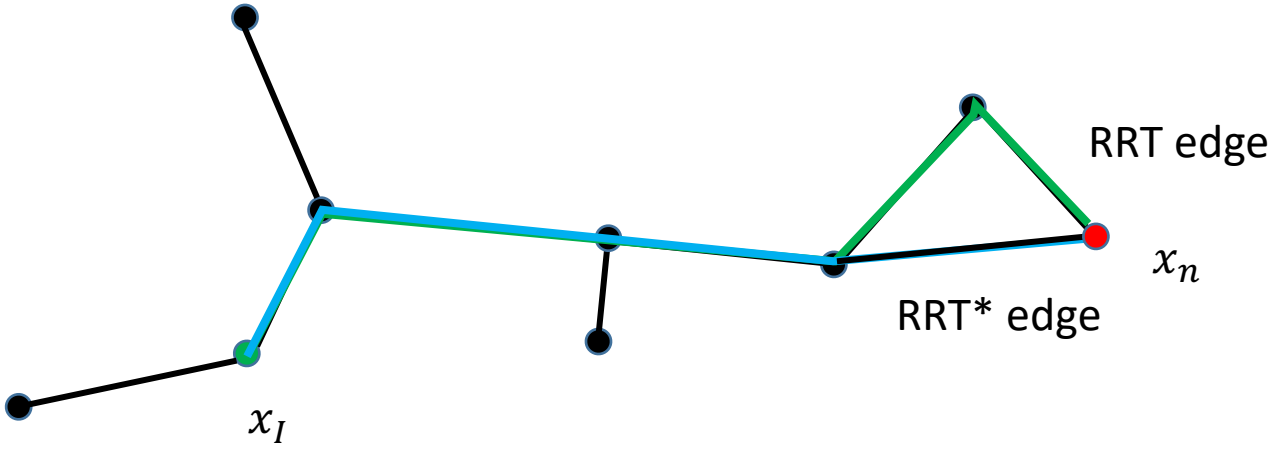
⇨Can we do better?

⇨ Need to keep “re-wiring” the graph structure

Re-Wiring for Gaining Optimality

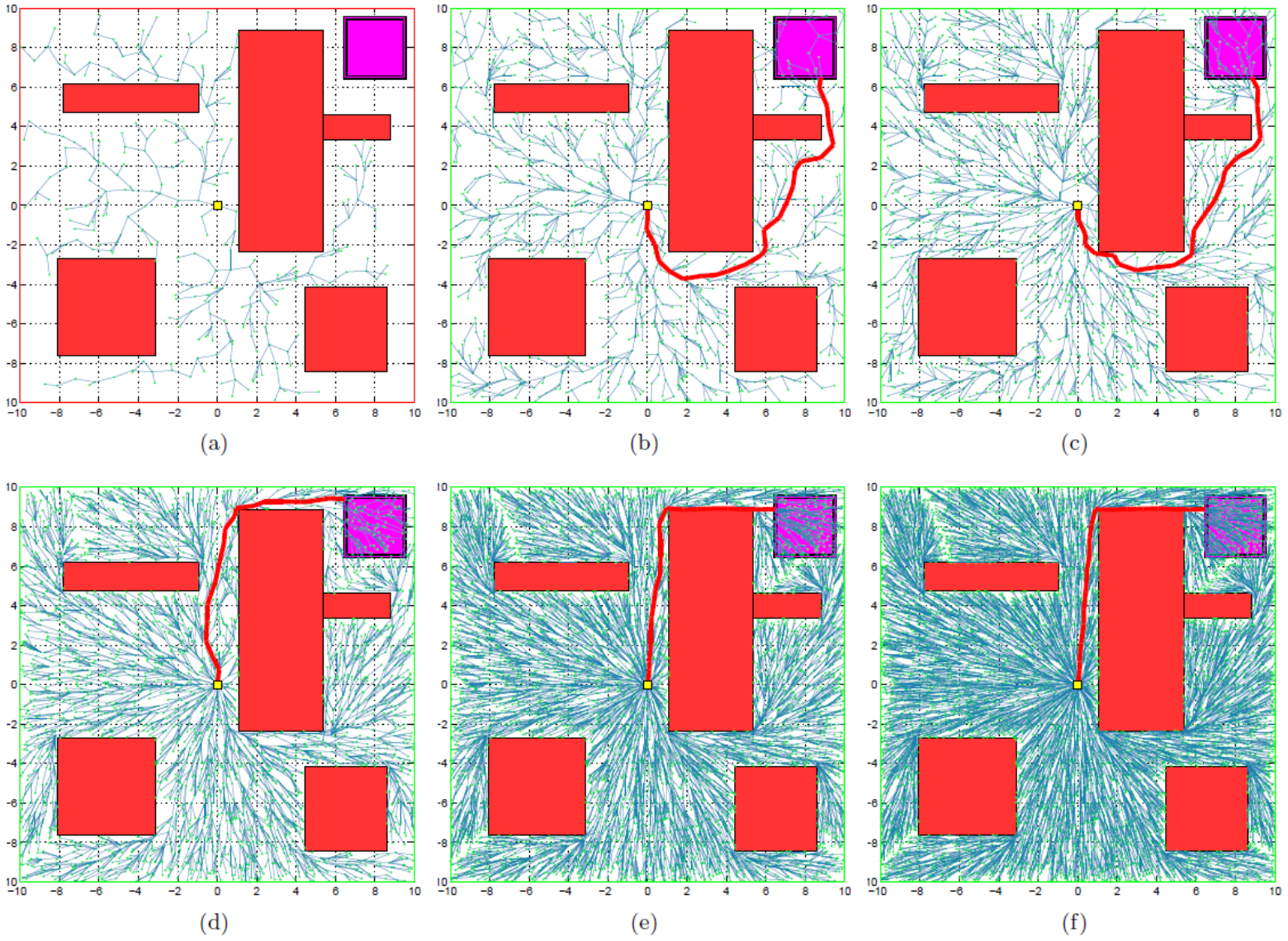
Consider growing an RRT without obstacles

- ⇨For each new sample x_n, check its $\log n$ neighborhood
- ⇨If there are better paths from x_I to x_n, pick that path
- ⇨The resulting algorithm is called RRT* (Karaman & Frazzoli)



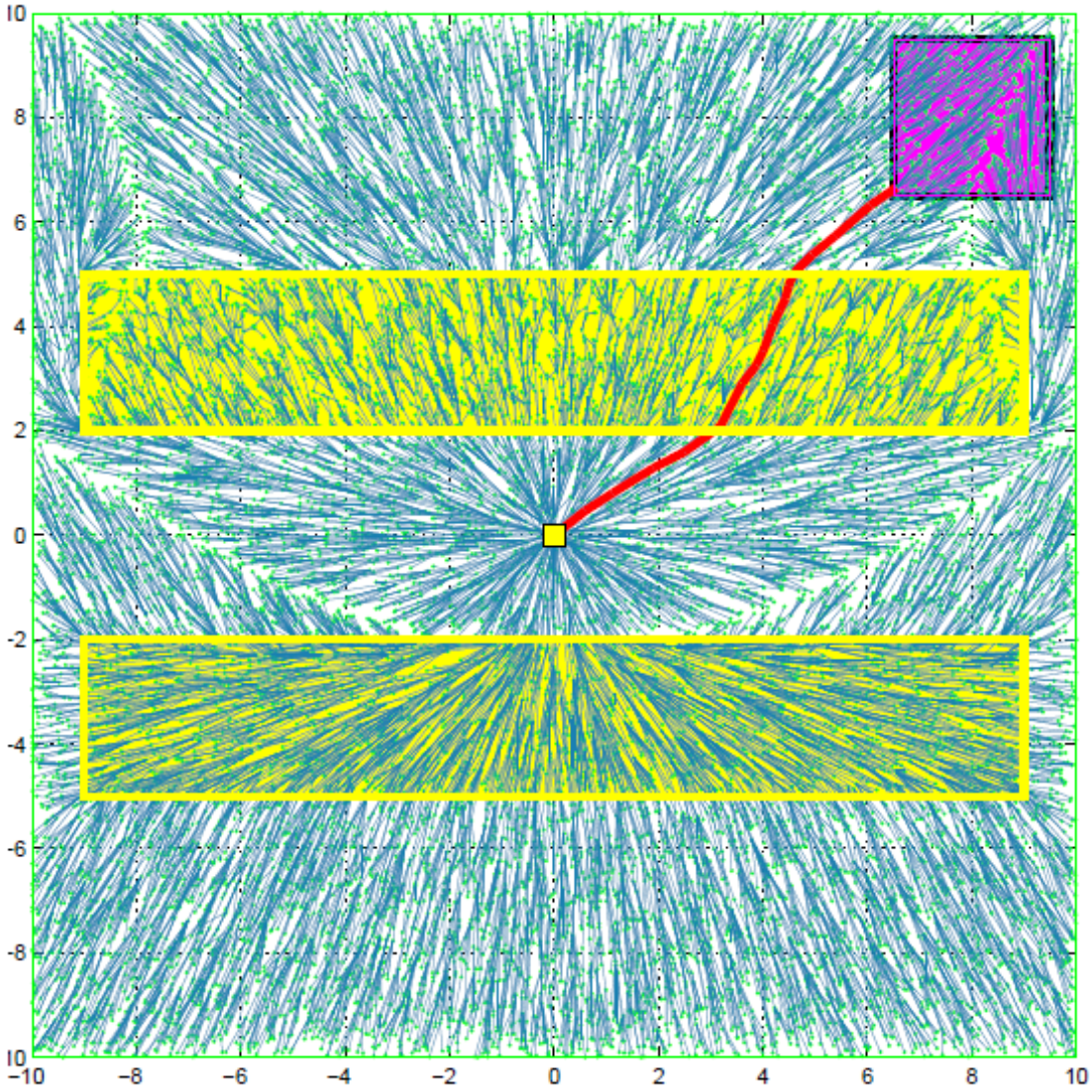
- ⇨RRT* is an asymptotically optimal sampling-based algorithm
  - ⇨ As the number of samples goes to infinity, an optimal path from x_I to x_G is obtained
- ⇨Other optimal sampling-based planners are available as well
  - ⇨ PRM* (Karaman & Frazzoli)
  - ⇨ Stable Sparse RRT (SST) (Bekris)

RRT* Example with Obstacles



RRT* after 500, 1500, 2500, 5000, 10000, 15000 Iterations

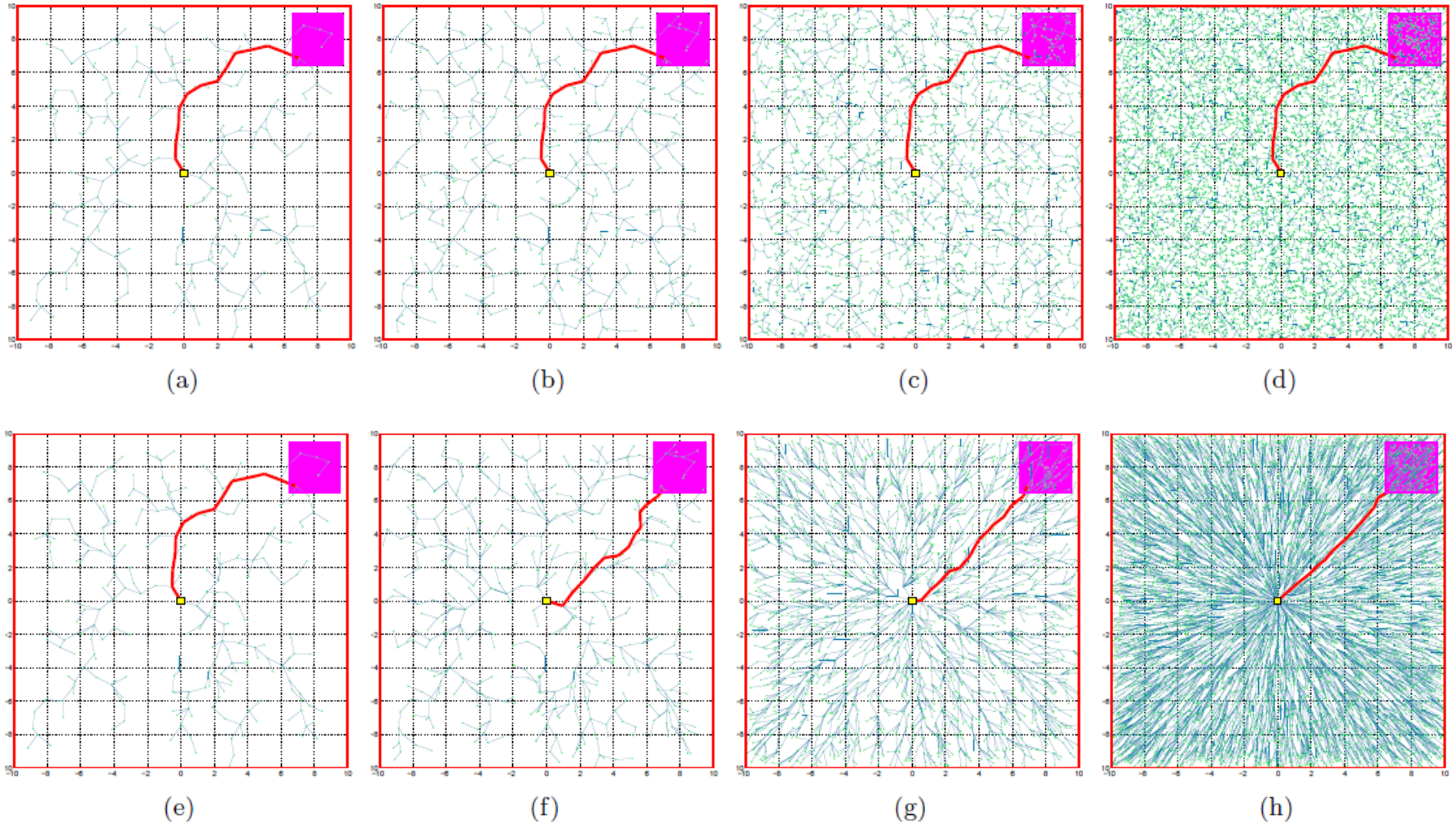
RRT* Example with Cost Variance



High cost region

Low cost region

RRT* Compared to RRT



RRT [top] and RRT*[bottom] after 250, 500, 2500, and 10000 vertices are added