Announcements

- Lab today (Feb 5):
  - **Compulsory presence** at 14:00
  - Brief tutorial on how to operate real Turtlebot.
  - Sign-up (3 students per Turtlebot).
- Turtlebot sharing:
  - 3 students per robot; each robot in a dedicated locker in Intel lab
  - Key to be stored at the **reception** - note weekend closure!
- **Don’t break the robots.**

Mobile Robot Systems

Lecture 6: Navigation & Path Planning

Dr. Amanda Prorok

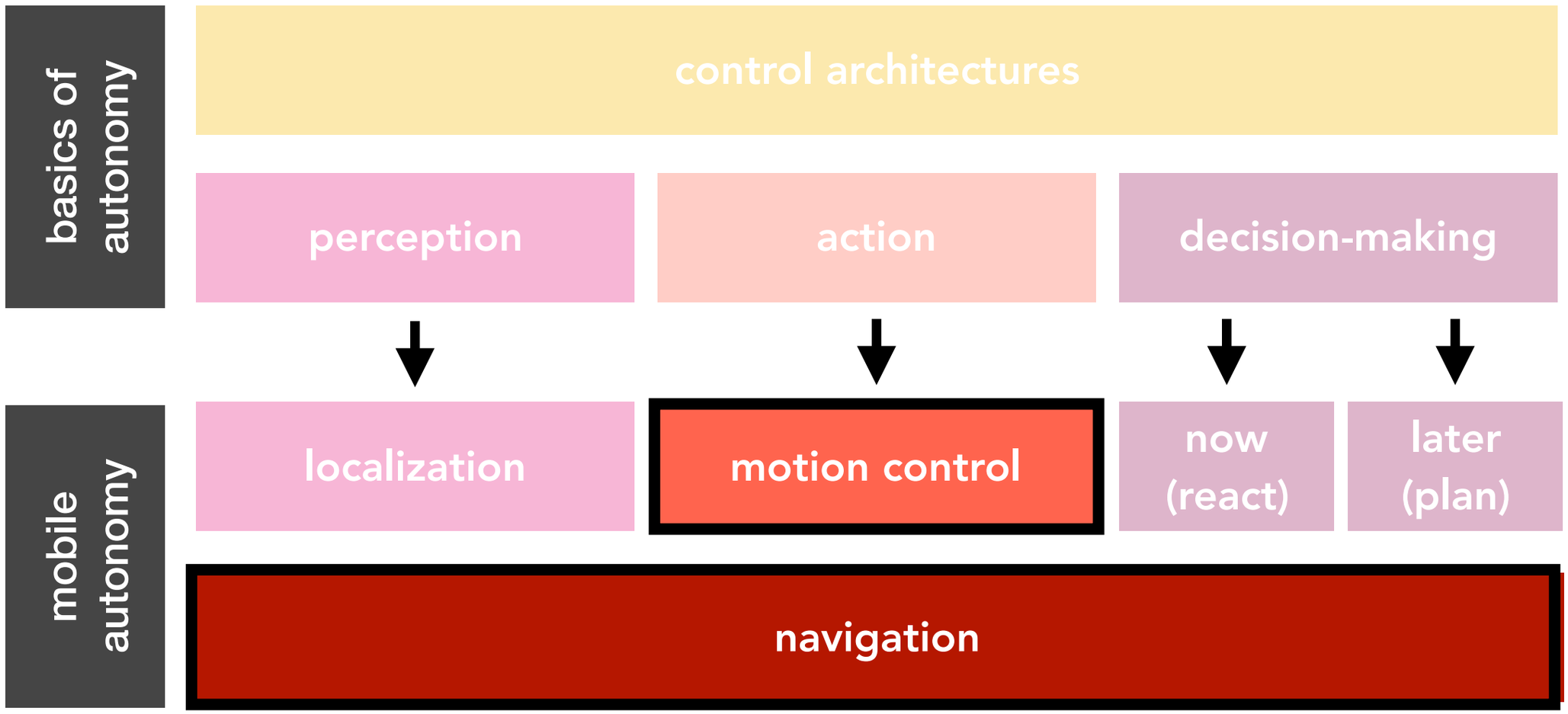
asp45@cam.ac.uk

www.proroklab.org

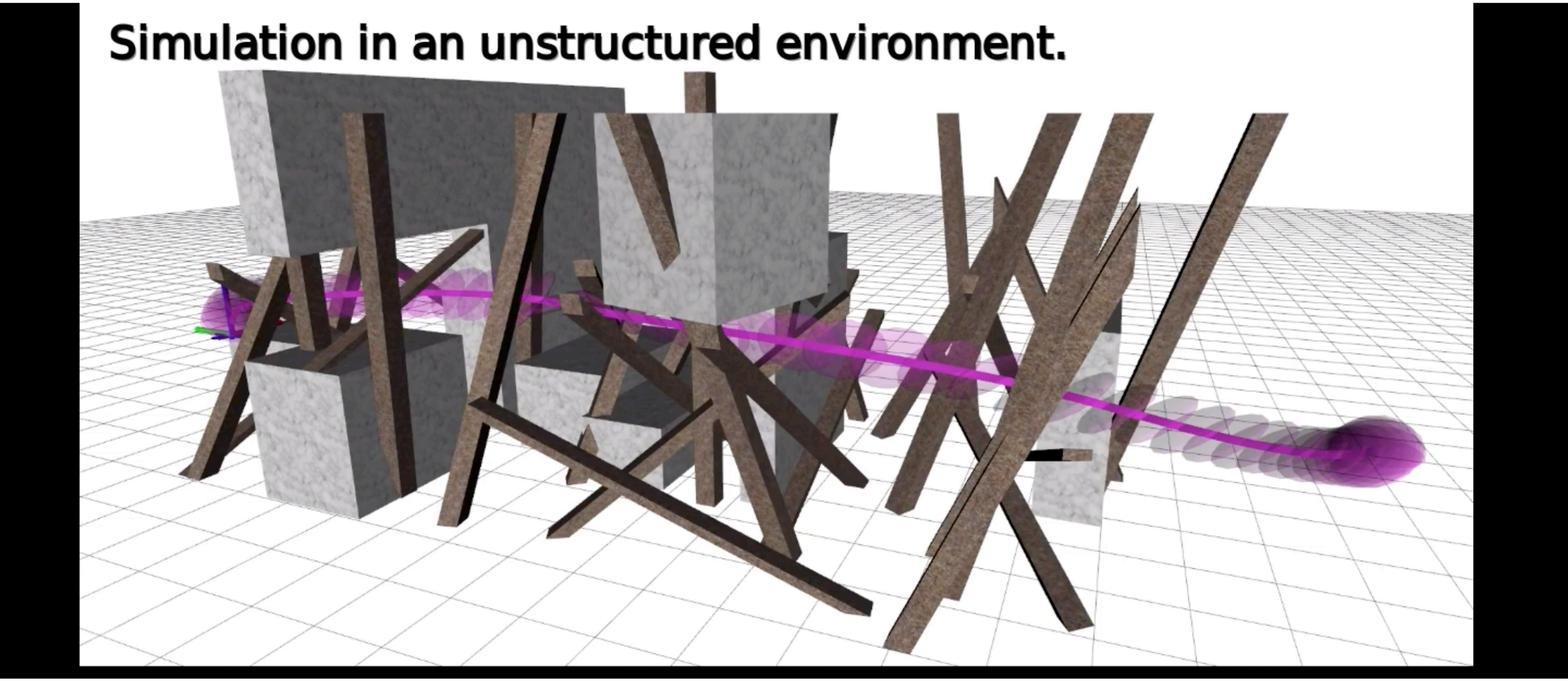
In this Lecture

- Navigation and path planning
- Configuration space
- 3 general method classes
  - Combinatorial
  - Sampling-based
  - Potential fields

Architecture of an Autonomous Mobile Robot



Motion Planning



Configuration Space

The ‘world’ has two entities: **robots and obstacles**. Both considered as closed subset of the world (or workspace): $\mathcal{O} \subset \mathcal{W}$, $\mathcal{R} \subset \mathcal{W}$

The ‘space’ for motion planning is the set of possible transformations that could be applied to the robot (considered as a rigid body).

We refer to this as the **configuration space**.

Important **abstraction** that allows to use the same motion planning algorithms to problems that differ in geometry and kinematics.

[Two common views of the configuration space:
metric space, or topological manifold]

Configuration Space

The robot is mapped to a **single point** in C-space $\mathscr{C}$

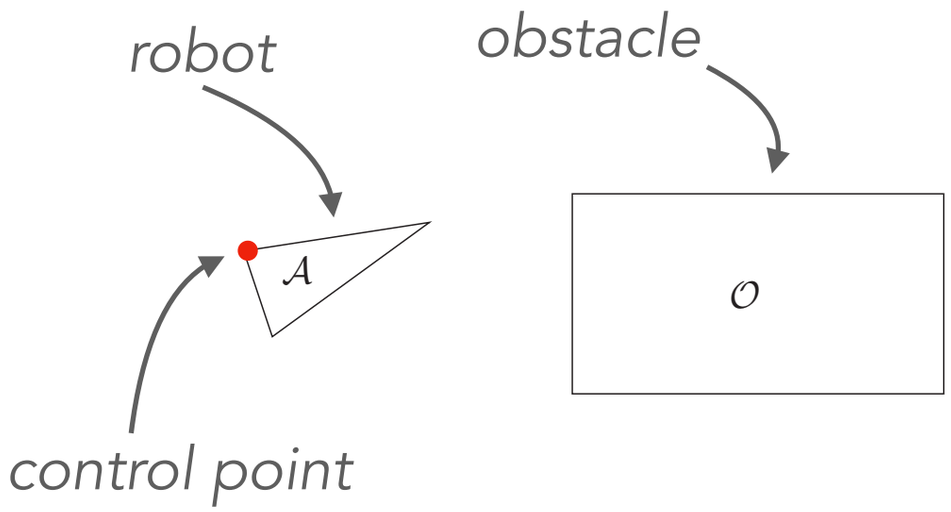
Complete specification of robot configuration: $\mathbf{q} \in \mathscr{C}$

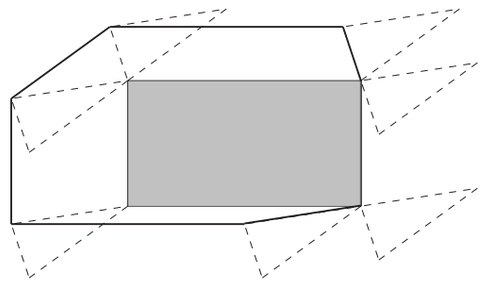
Robot's workspace: $\mathscr{W} = \mathbb{R}^N$ and points occupied by robot: $A(\mathbf{q}) \subset \mathscr{W}$

Obstacle region: $\mathcal{O} \subset \mathscr{W}$

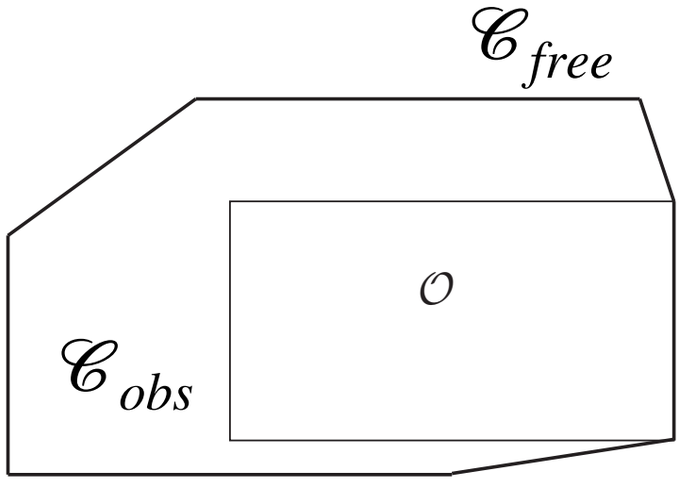
Set of configurations that avoid collision: $\mathscr{C}_{free} = \mathscr{C} \backslash \mathscr{C}_{obs}$

where we have: $\mathscr{C}_{obs} = \{\mathbf{q} \in \mathscr{C} \,|\, A(\mathbf{q}) \cap \mathcal{O} \neq \emptyset\}$



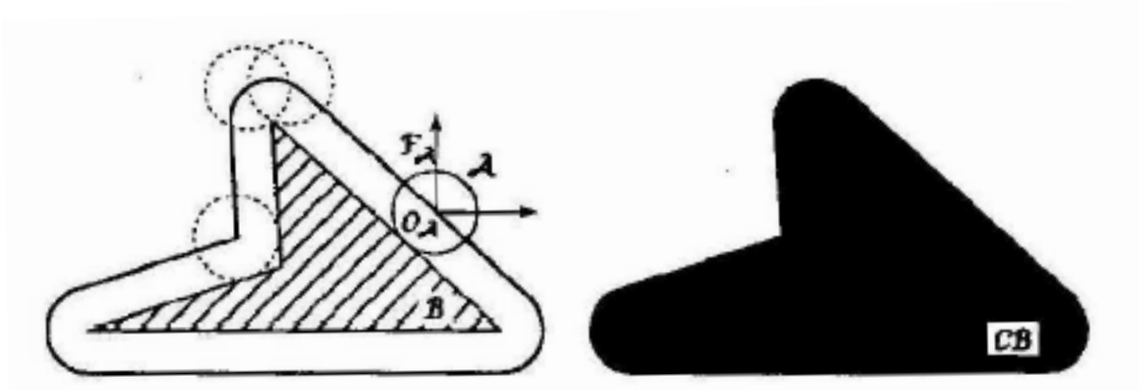


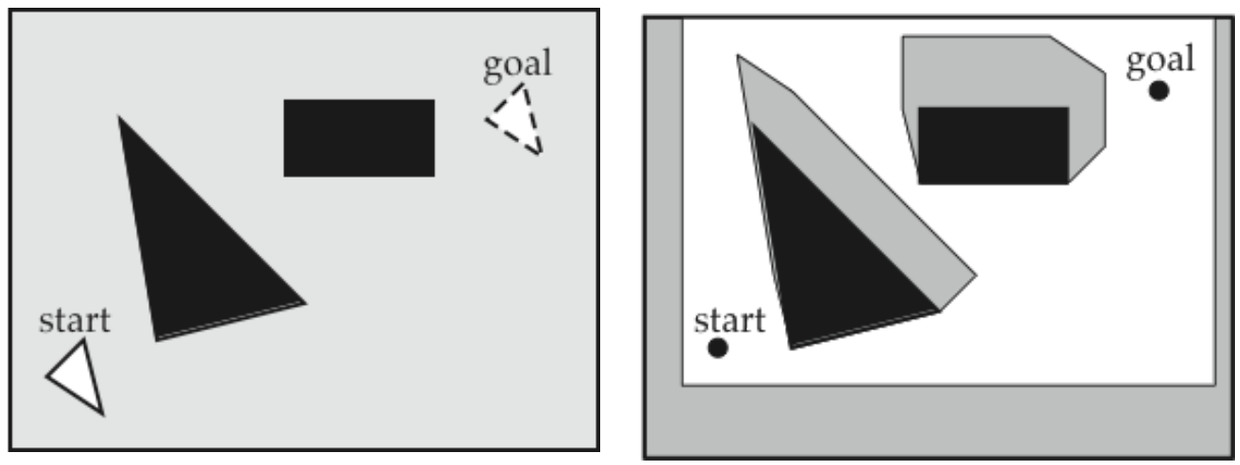
planar translation



Configuration Space

How to compute $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$?



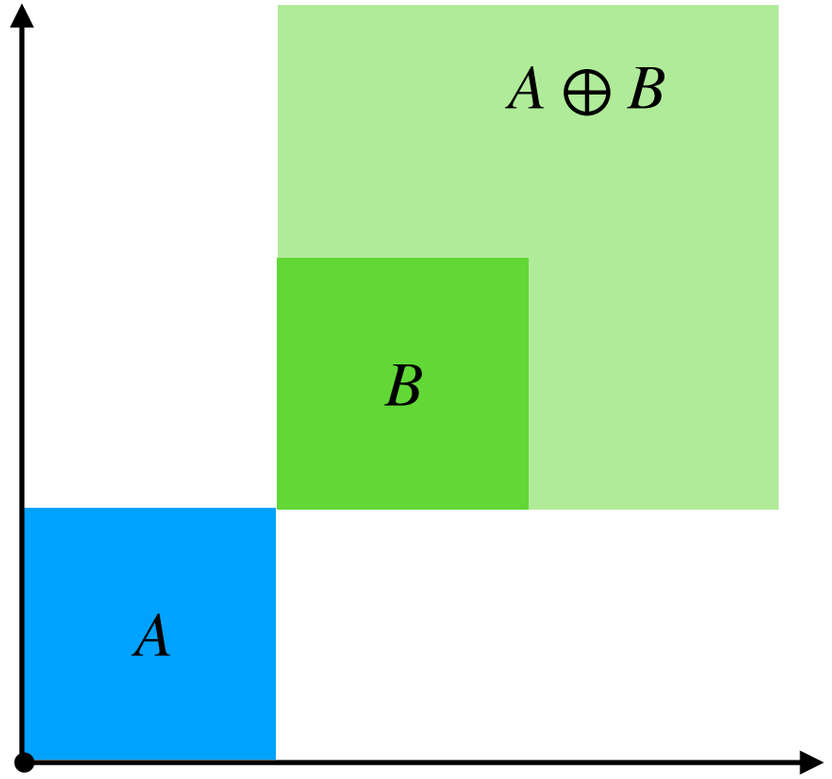


Various methods, e.g. reflect points, Minkowski sum, convex Hull.

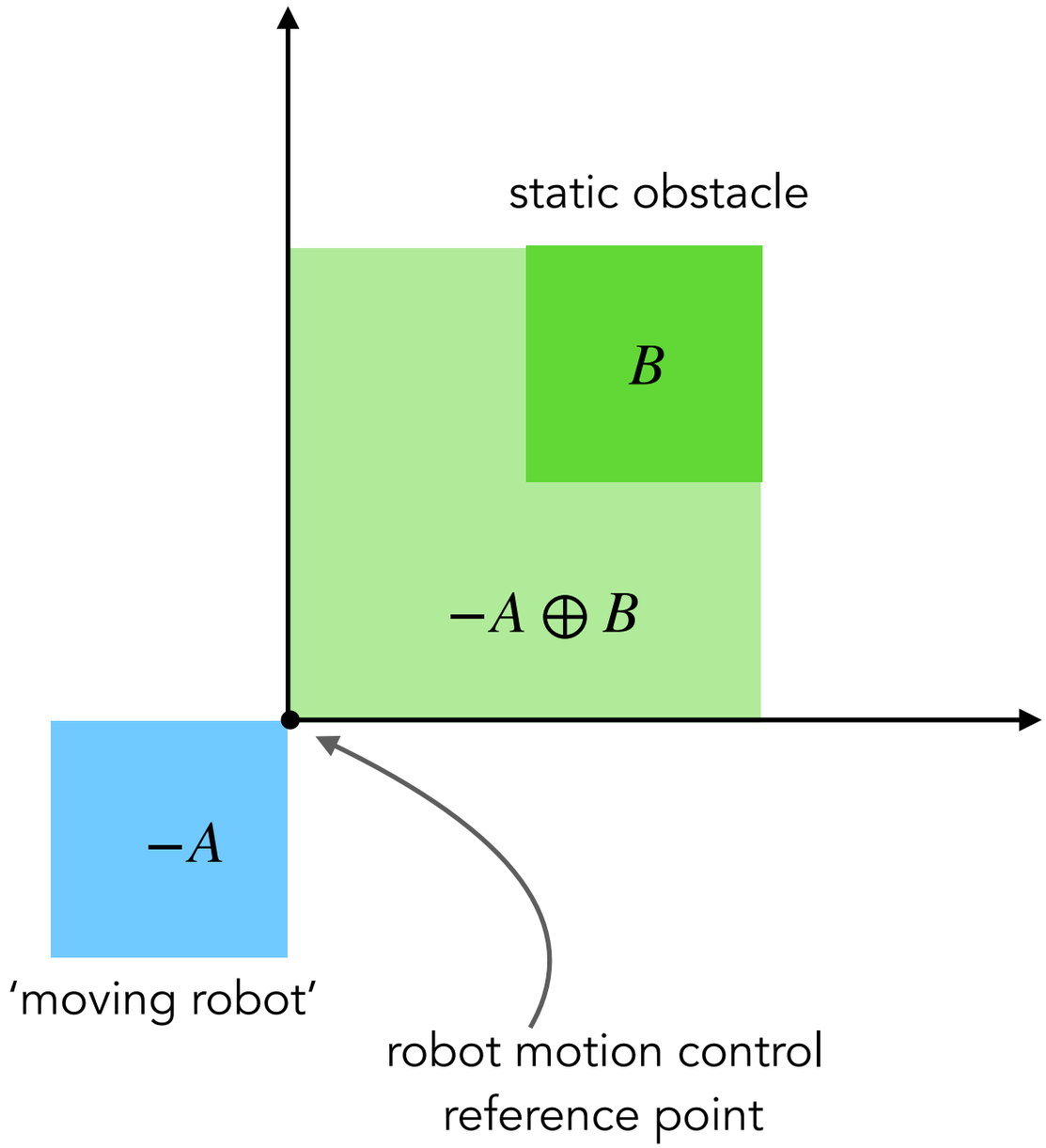
Minkowski Sum

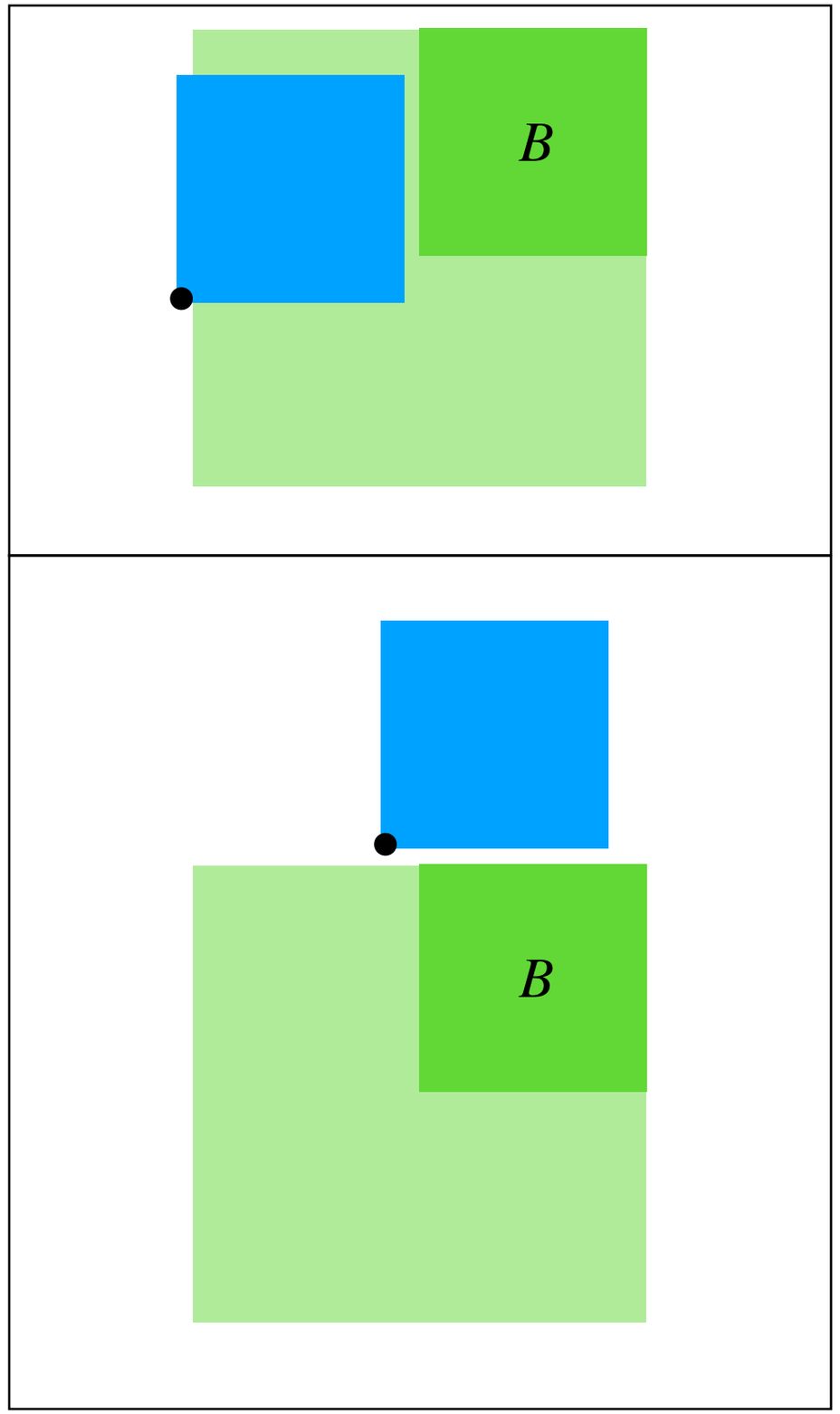
- In geometry, the Minkowski sum (also known as dilation) of two sets of position vectors $\boldsymbol{A}$ and $\boldsymbol{B}$ in Euclidean space is formed by adding each vector in $\boldsymbol{A}$ to each vector in $\boldsymbol{B}$, i.e., the set:

$$A \oplus B = \{\mathbf{a} + \mathbf{b} \,|\, \mathbf{a} \in A, \mathbf{b} \in B\}$$



Minkowski Sum





As long as reference point stays outside dilated area, there will be no collisions.

The Path Planning Problem

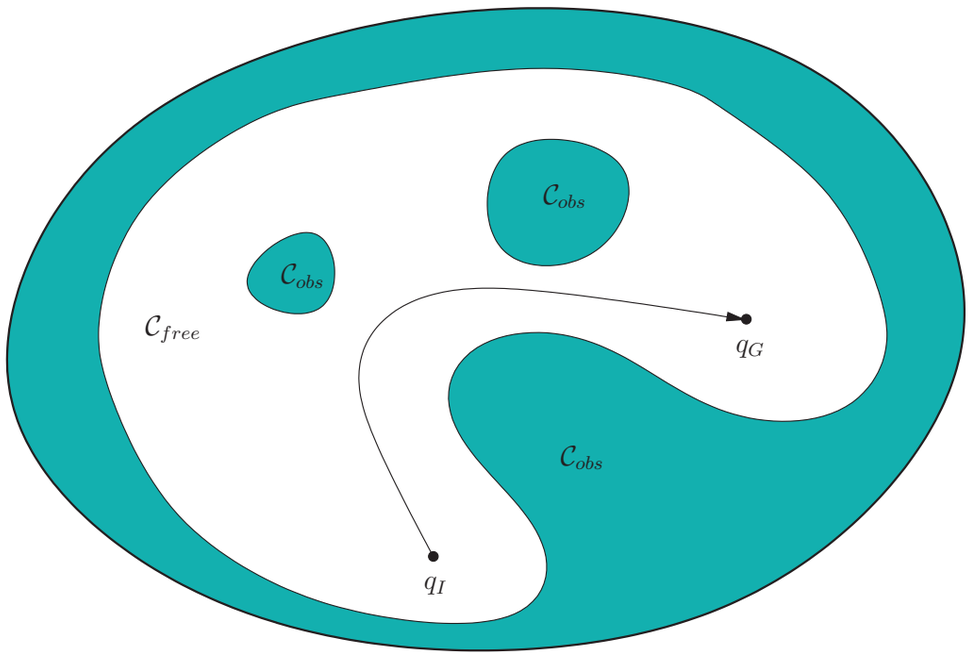
Assume a workspace, obstacle region, and configuration space, with definitions of free and occupied C-space $\mathscr{C}_{free}$ and $\mathscr{C}_{obs}$

Given a *query* $(\mathbf{q}_I, \mathbf{q}_G)$ with

an initial configuration $\mathbf{q}_I \in \mathscr{C}_{free}$

a desired goal configuration $\mathbf{q}_G \in \mathscr{C}_{free}$

compute a path $\tau : [0,1] \mapsto \mathscr{C}_{free}$ such that $\tau(0) = \mathbf{q}_I$ and $\tau(1) = \mathbf{q}_G$



* image credit: Pallotino

Approaches and Guarantees

1. **Combinatorial** planning (exact)
2. **Sampling-based** planning (probabilistic)
3. **Potential-field** methods

These methods use the configuration space abstraction (hence, require preliminary computation of C_{free} and C_{obs})

Guarantees:

- **Complete**: if a solution exists, it finds one, otherwise returns failure
- **Semi-complete**: if a solution exists, if finds one; otherwise may run forever
- **Resolution complete**: If a solution exists, it finds one; otherwise, it terminates and reports that no solution within a specified resolution exists.
- **Probabilistically complete**: If a solution exists, the probability that it will be found tends to one as the number of iterations tends to infinity

Overview of Methods

- **Combinatorial methods**
  - Exact and *complete*
  - Underlying principle: Requires a-priori computation of C-space and generation of a roadmap (e.g., a visibility graph). Then, use a discrete graph-search method on the roadmap.
- **Sampling-based methods**
  - Approximate and not complete; but can be *resolution complete*, or *probabilistically complete*
  - Underlying principle: Avoid explicit construction of roadmaps in C-space; instead, sample a path segment and check for collisions
- **Potential-field methods**
  - Approximate and not complete (suffer from local minima)
  - Underlying principle: Integrate over a vector field to obtain a trajectory (pose as a function of time). Can also be done in reactive mode.

Combinatorial Methods

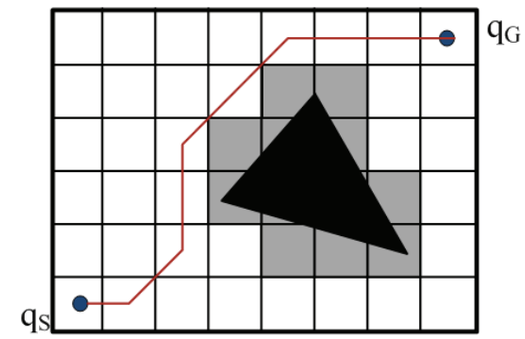
- Recipe:

  1. Compute **C-space** (C_{free} and C_{obs})

  2. Generate a **roadmap** (i.e., a **graph**) in C_{free}

     - Cell decomposition methods: visibility graphs / Voronoi cells / occupancy grid maps
     - Or: maximum clearance roadmaps (direct construction)

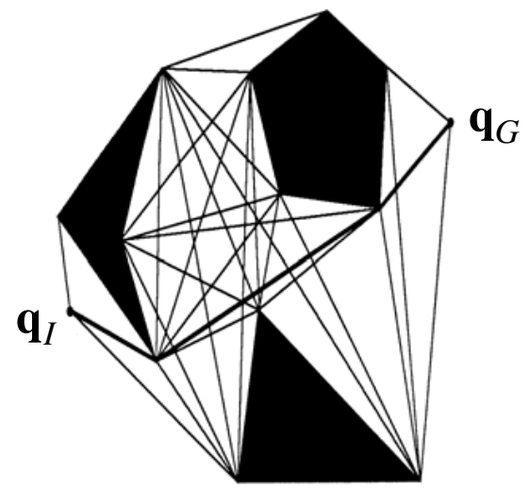
*A valid **roadmap** guarantees accessibility and is connectivity preserving w.r.t. C_{free}*



3. Compute the **minimum-cost path** from initial to goal configuration (cast as a graph search algorithm)

4. Result:

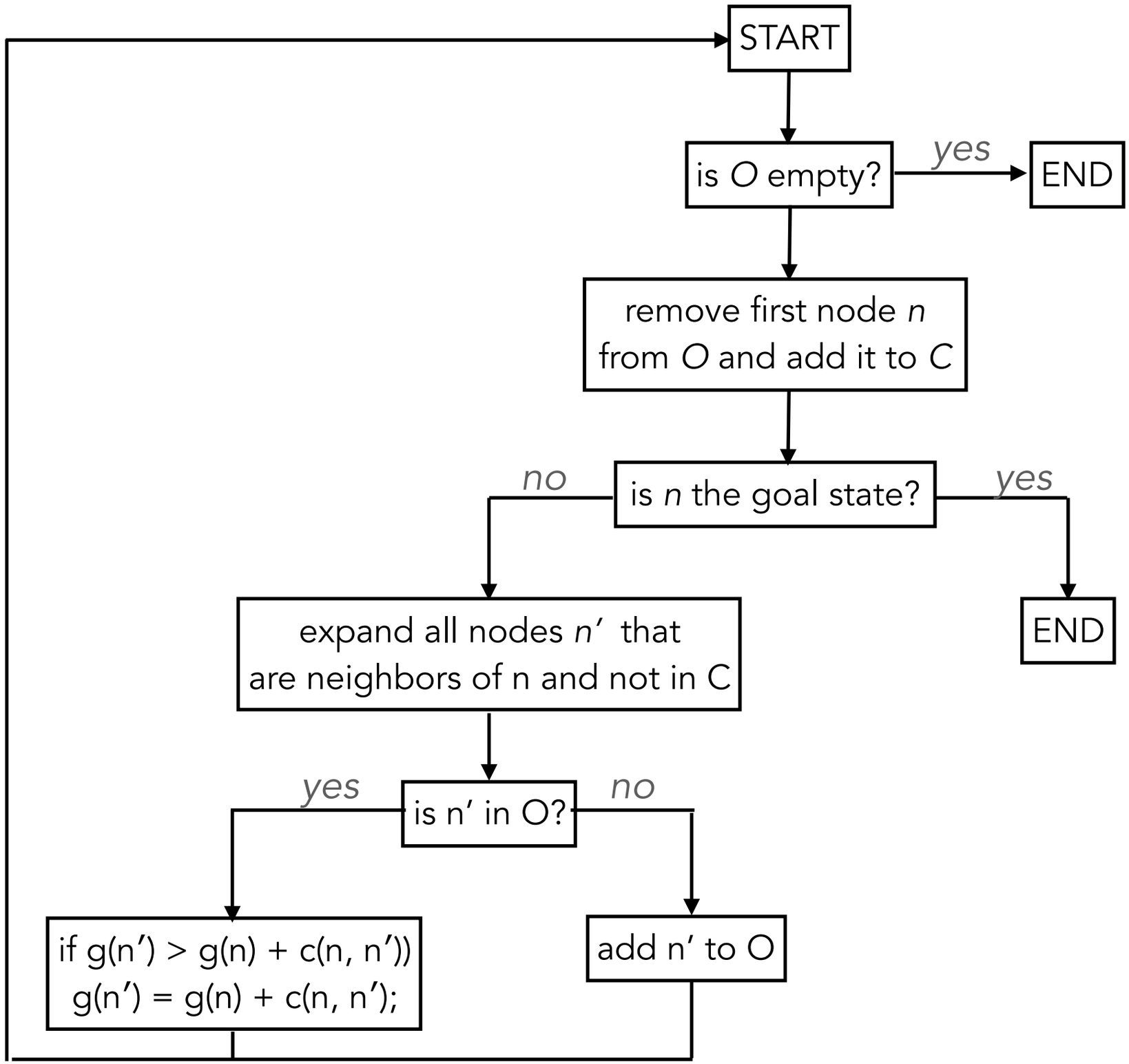
$\tau : [0,1] \mapsto \mathscr{C}_{free}$ such that $\tau(0) = \mathbf{q}_I$ and $\tau(1) = \mathbf{q}_G$



Discrete Search Methods

- How to search roadmap for minimum-cost path?
- One well-known example: **A*** algorithm
- Extension of Dijkstra' search algorithm, to reduce number of states explored (exploiting an informed search using a heuristic)
- Forward search, applied to path planning:
  - Evaluation function: $f(n) = g(n) + h(n)$
  - Operating cost function $g(n)$; cost of path already traversed.
  - Heuristic function $h(n)$; information used to find promising nodes to traverse; heuristic must be **admissible** (i.e., must **underestimate** true cost)!

A*



O: ordered set / open list (priority queue)
C: closed set

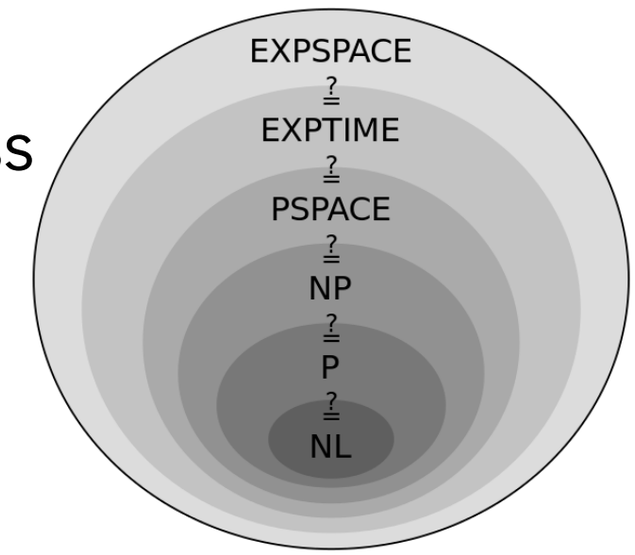
A*

- Requirements
  - Preprocessing to generate roadmap (connected graph) that represents C_{free}
- Pros
  - Optimal path cost and complete
- Cons
  - Memory inefficient (see IDA*)
  - Curse of dimensionality (exponential growth of search space w.r.t. length of solution)

Variants:
replanning algorithms (e.g. D)*
anytime algorithms (e.g. ARA),*
anytime re-planning (e.g. AD)*

Complexity of Path Planning

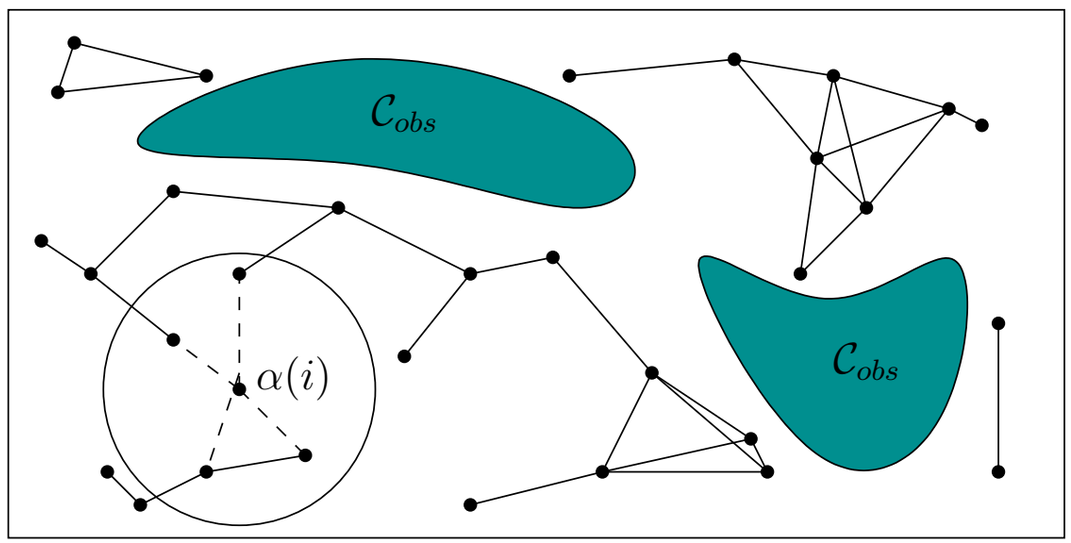
- The general motion planning problem is **PSPACE-hard**
  [Reif 1979; Hopcroft et al. 1984]
- NP ⊆ PSPACE (polynomial amount of memory)
- Challenges:
  - C-space has high dimensionality: E.g.: A rigid body in 3D space has a C-space with 6 dimensions ($\mathscr{C} = \mathbb{R}^3 \times SO(3)$)
  - Simple obstacles have complex C-obstacles; impractical to compute explicit representation of free space for high DOF robots
- Attention has turned to approximation and **randomized** algorithms which trade full completeness of the planner for a major gain in efficiency
  → sampling-based approaches



Sampling-Based Methods

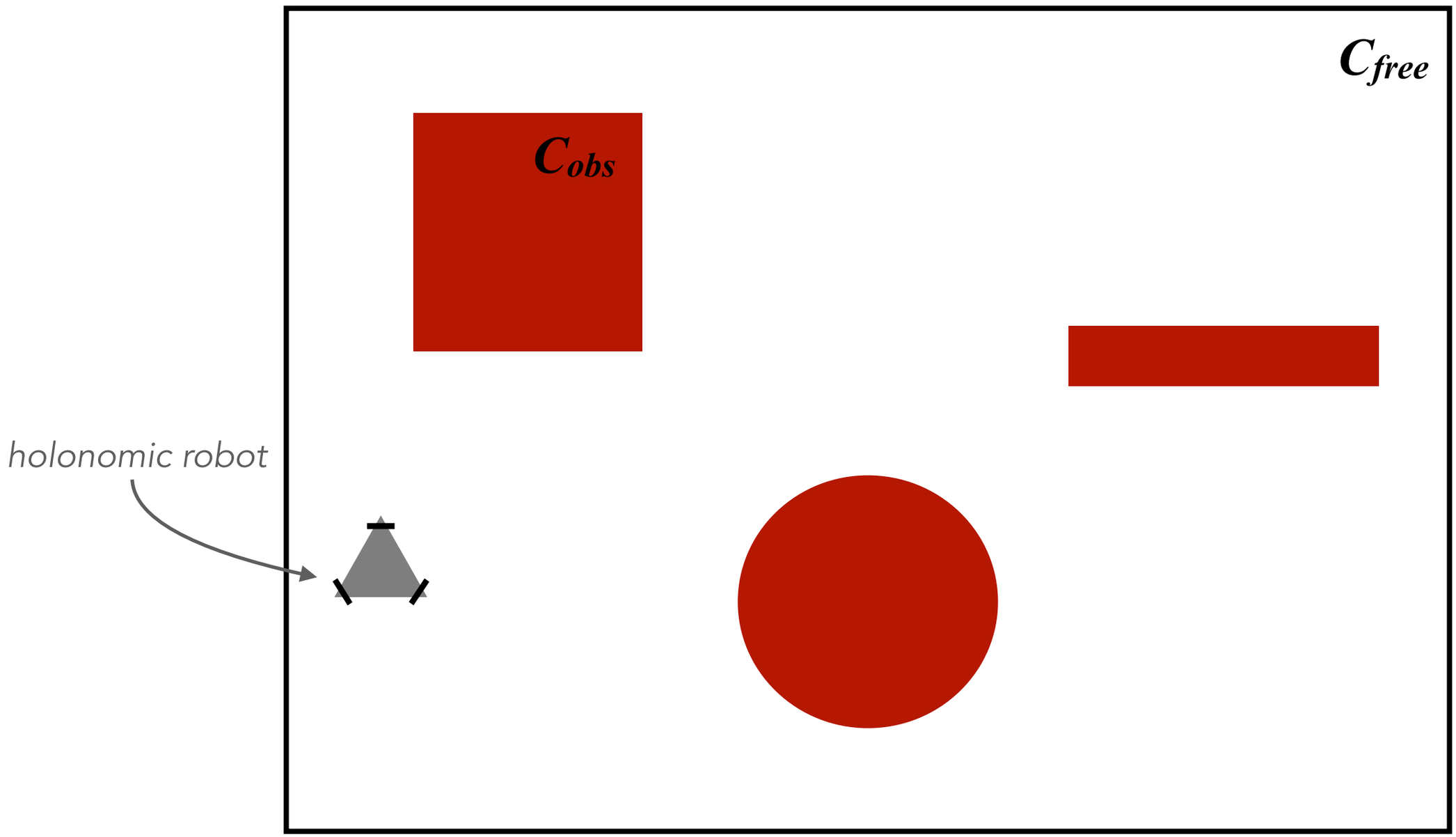
- **Probabilistic Roadmaps** (PRMs) (Kavraki, LaValle, et al.)

1. Initialize an empty graph G.
   Vertices will correspond to configurations, edges to collision-free paths.
2. **Sample** configurations $a(i)$ in C_{free}
3. Use a metric defined in C-space to compute **neighborhood** set of $a(i)$, of vertices $\boldsymbol{q}$ already in G.
4. **Local planner**: check if $a(i)$ can be connected to points $\boldsymbol{q}$ in neighborhood set. Add to edge $(a(i), \boldsymbol{q})$ to edge set if no collision detected.
5. Terminate when N edges added to roadmap.



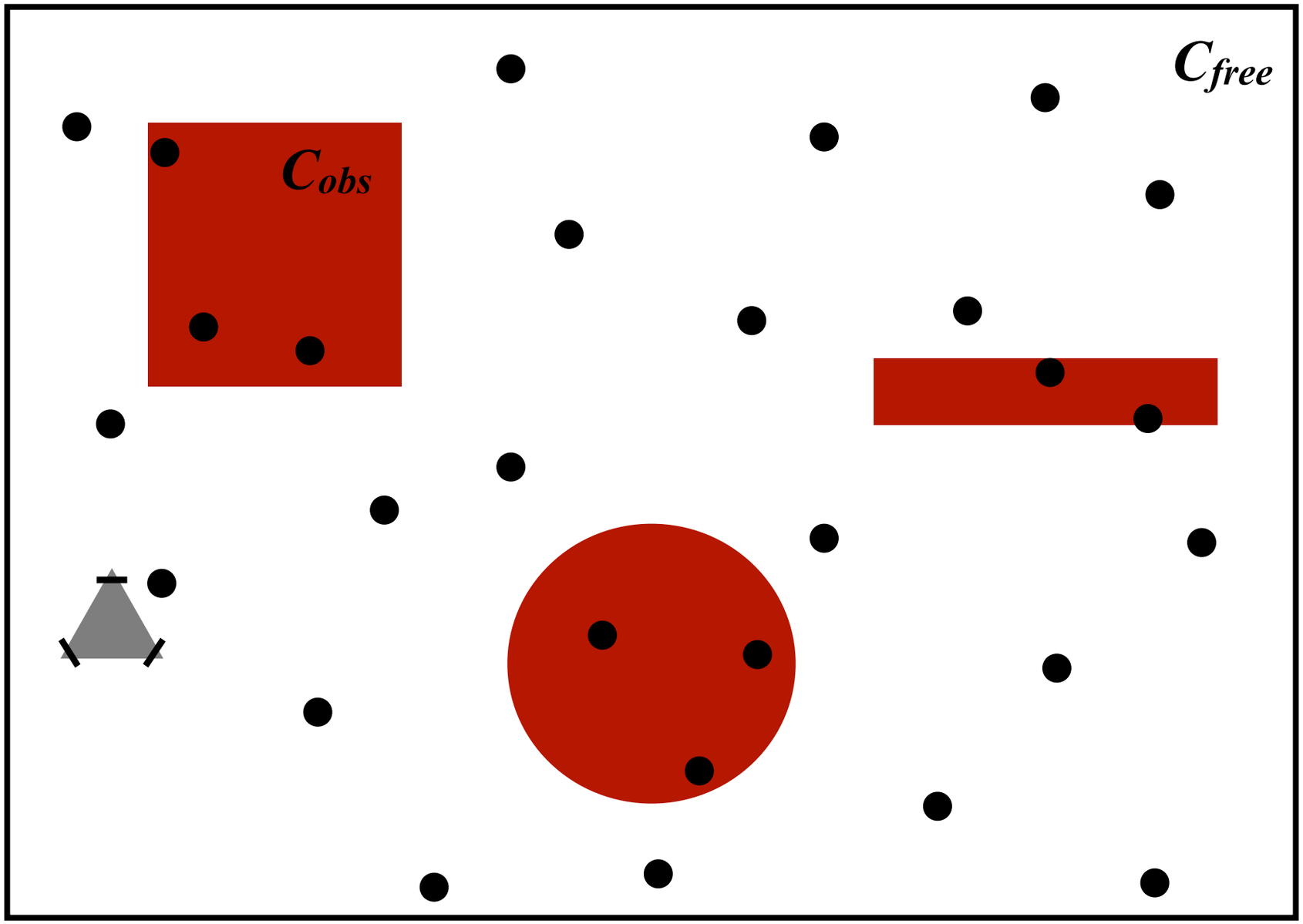
* image credit: Kavraki, 2008

Probabilistic Roadmaps (PRMs)



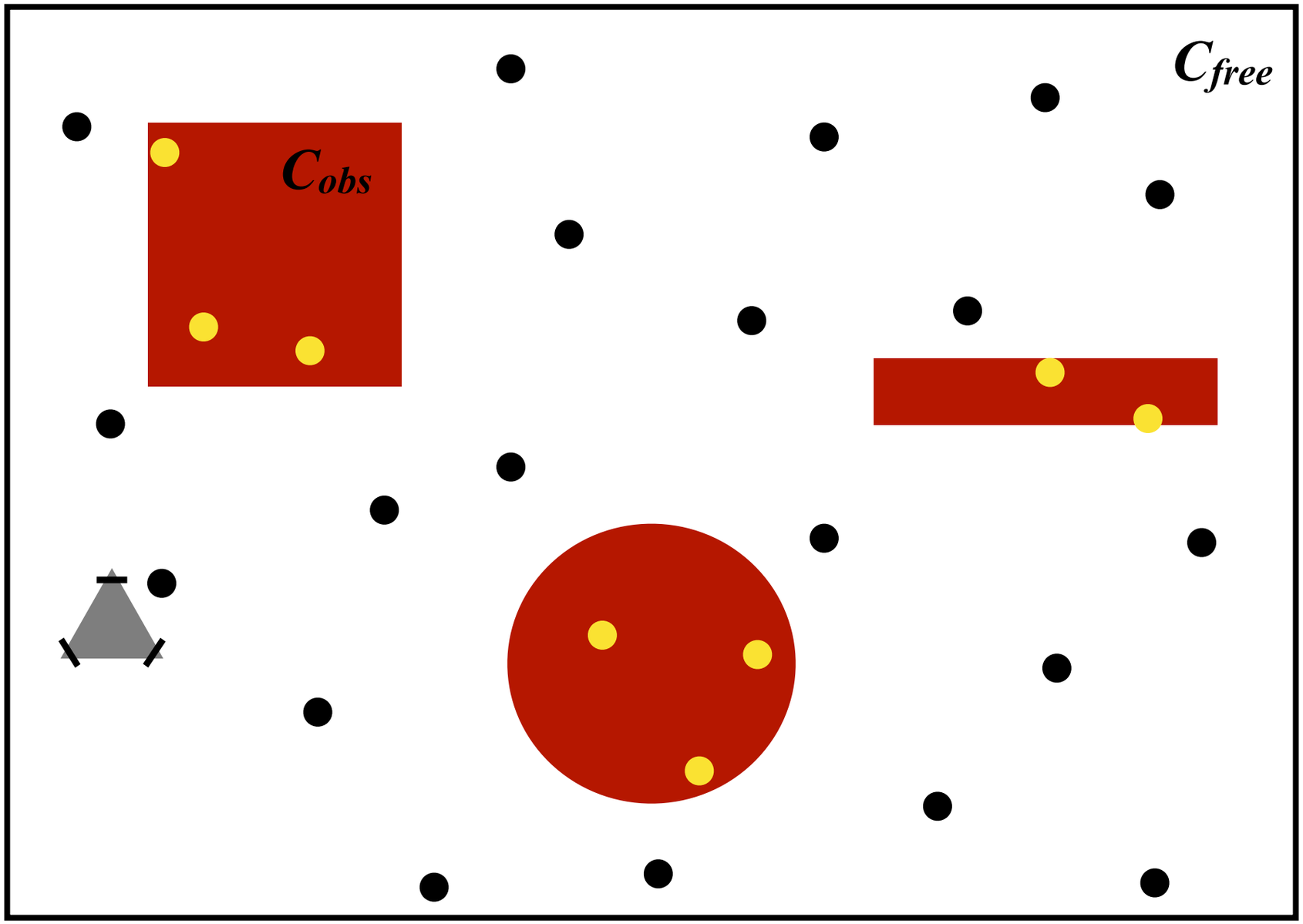
initialize empty graph and C-space

Probabilistic Roadmaps (PRMs)



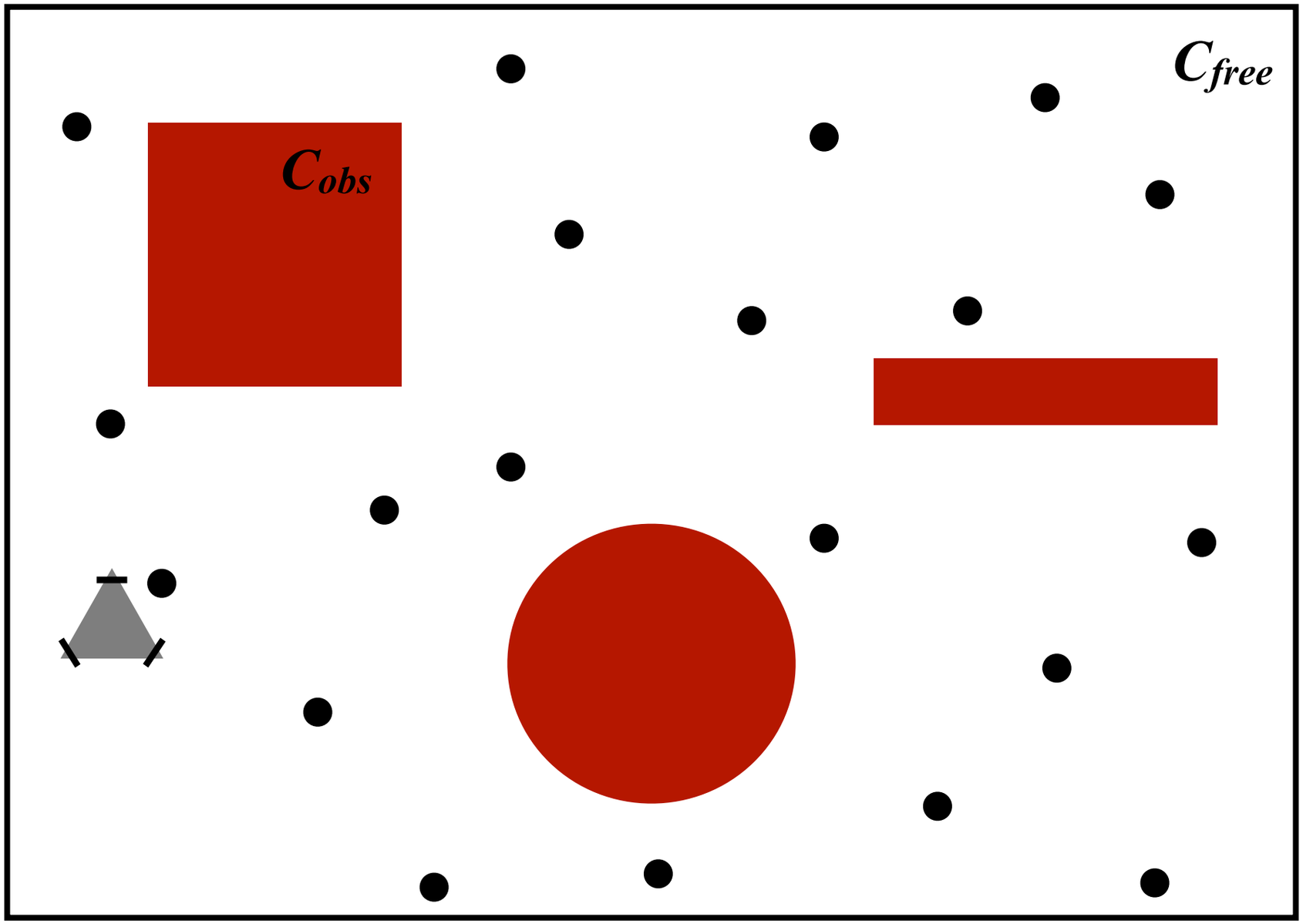
sample random configurations in C-space

Probabilistic Roadmaps (PRMs)



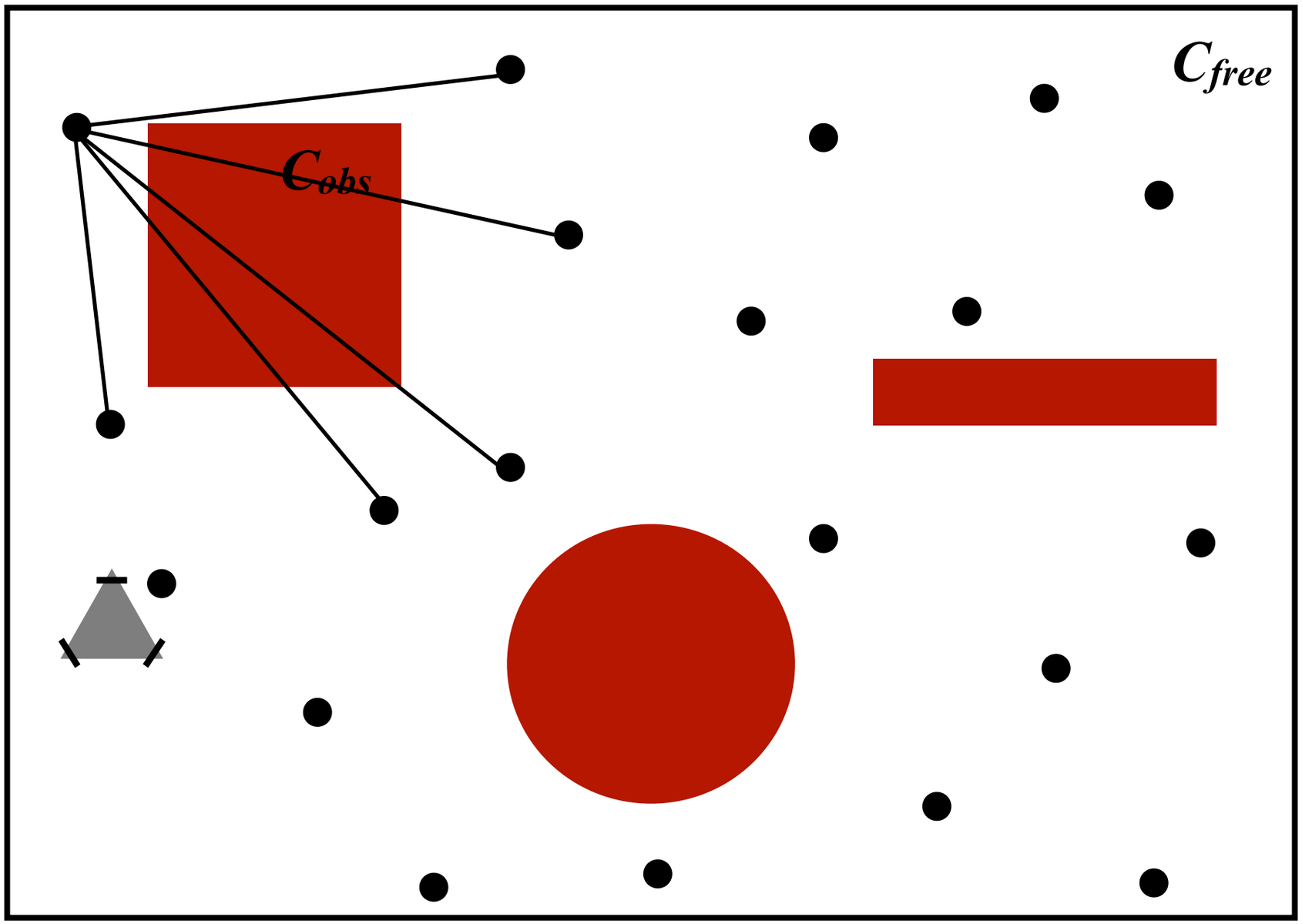
test for collisions

Probabilistic Roadmaps (PRMs)



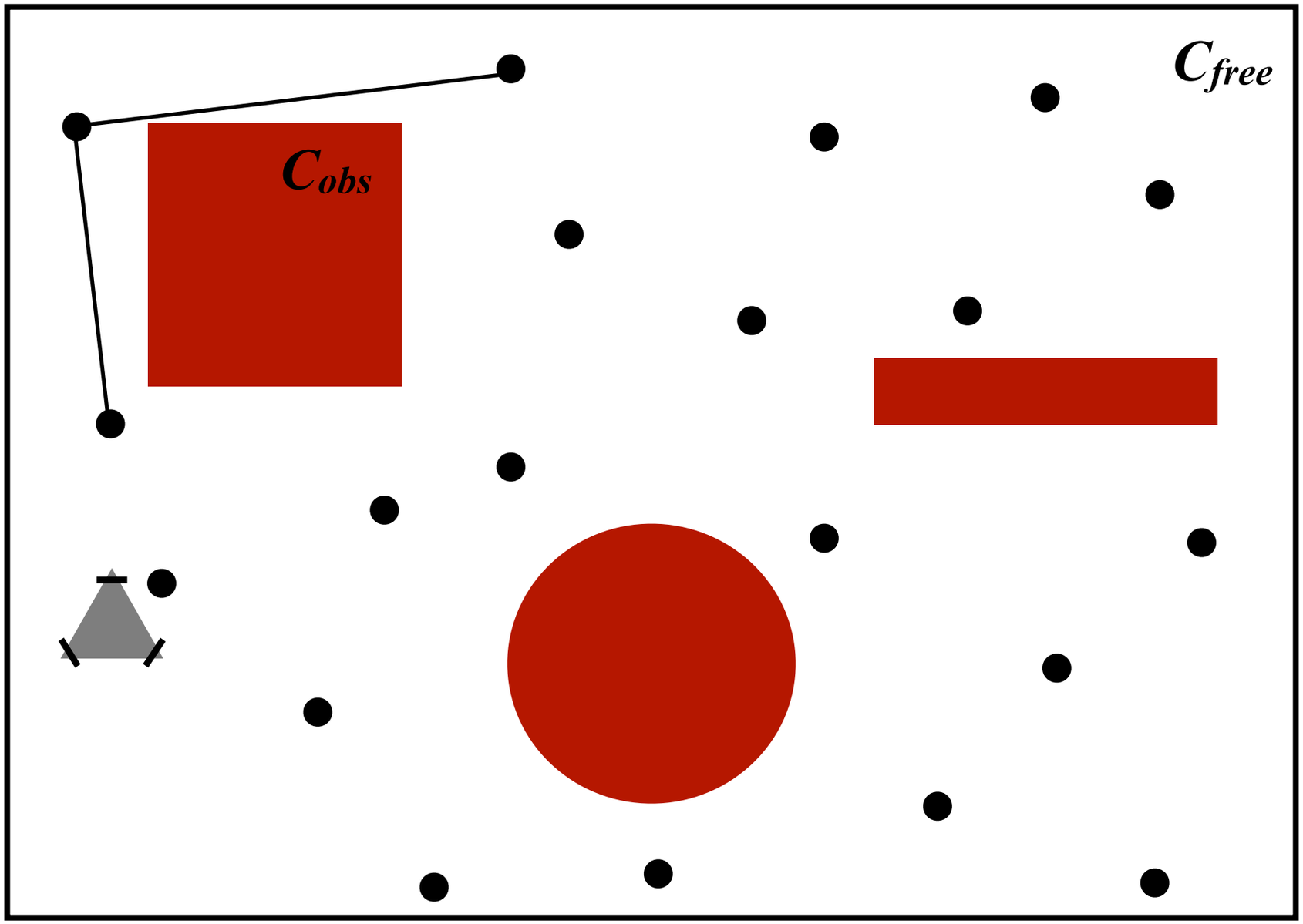
retain collision-free configurations

Probabilistic Roadmaps (PRMs)



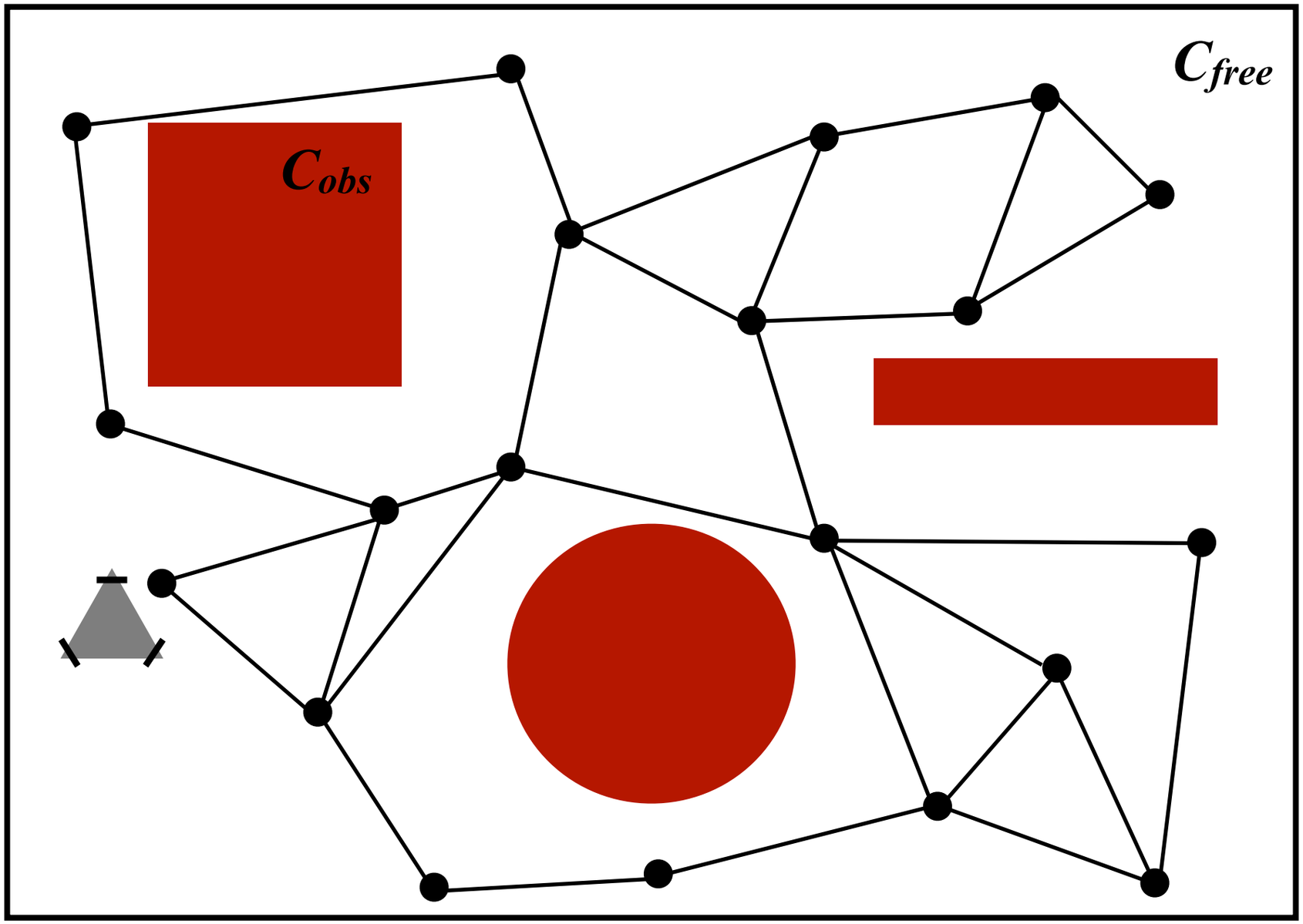
create edges to nearest neighbors

Probabilistic Roadmaps (PRMs)



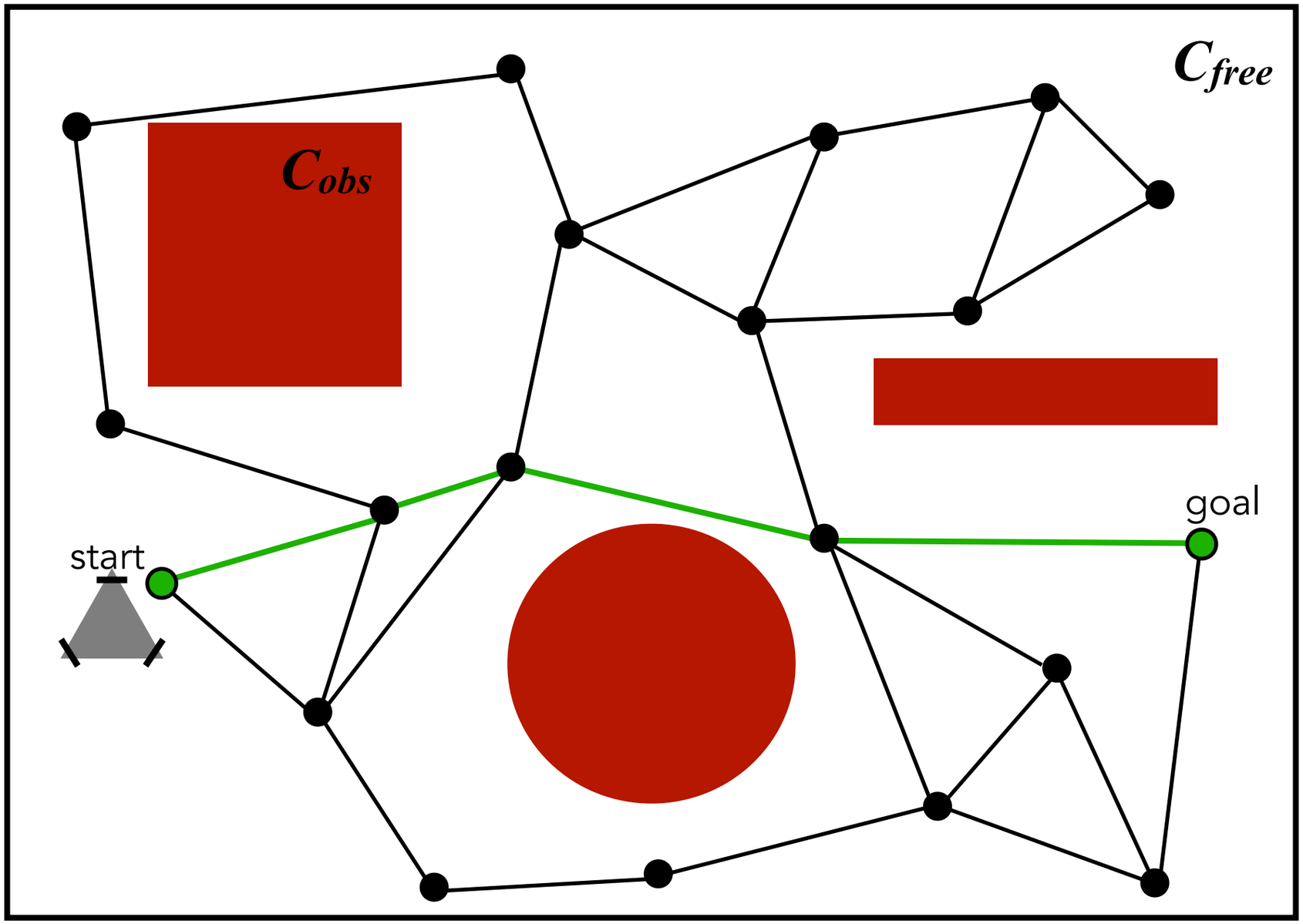
create edges to nearest neighbors

Probabilistic Roadmaps (PRMs)



retain collision-free local paths to generate PRM

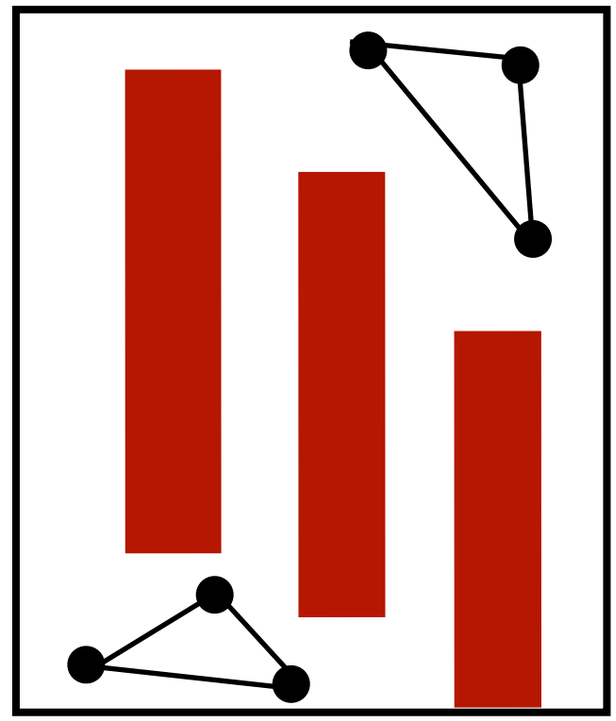
Probabilistic Roadmaps (PRMs)



search PRM for shortest path to goal configuration

Probabilistic Roadmaps (PRMs)

- Primitives required:
  - Method to sample configurations in free C-space
  - Method to check for collisions in C-space
- Pros:
  - Probabilistically complete
  - Apply easily to high-dimensional C-spaces
  - Fast query processing (start, goal)
- Cons:
  - Don’t work well for narrow passages
  - Hard to connect vertices for differential motion constraints (generally requires solution to the Boundary Value Problem)
  - Hard to sample uniformly in configuration space.



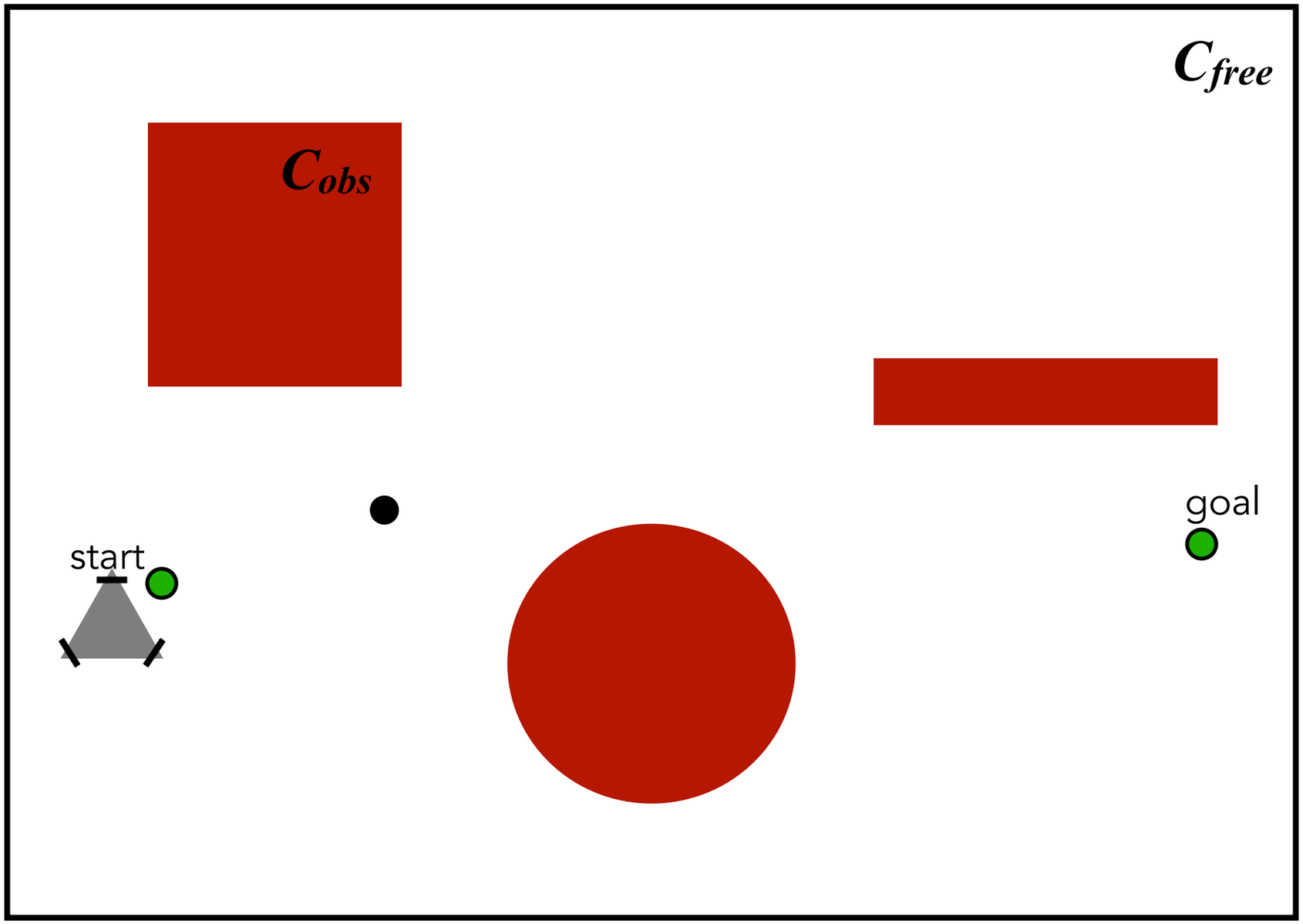
* image credit: Kavraki, 2008

Rapidly Exploring Random Trees (RRTs)

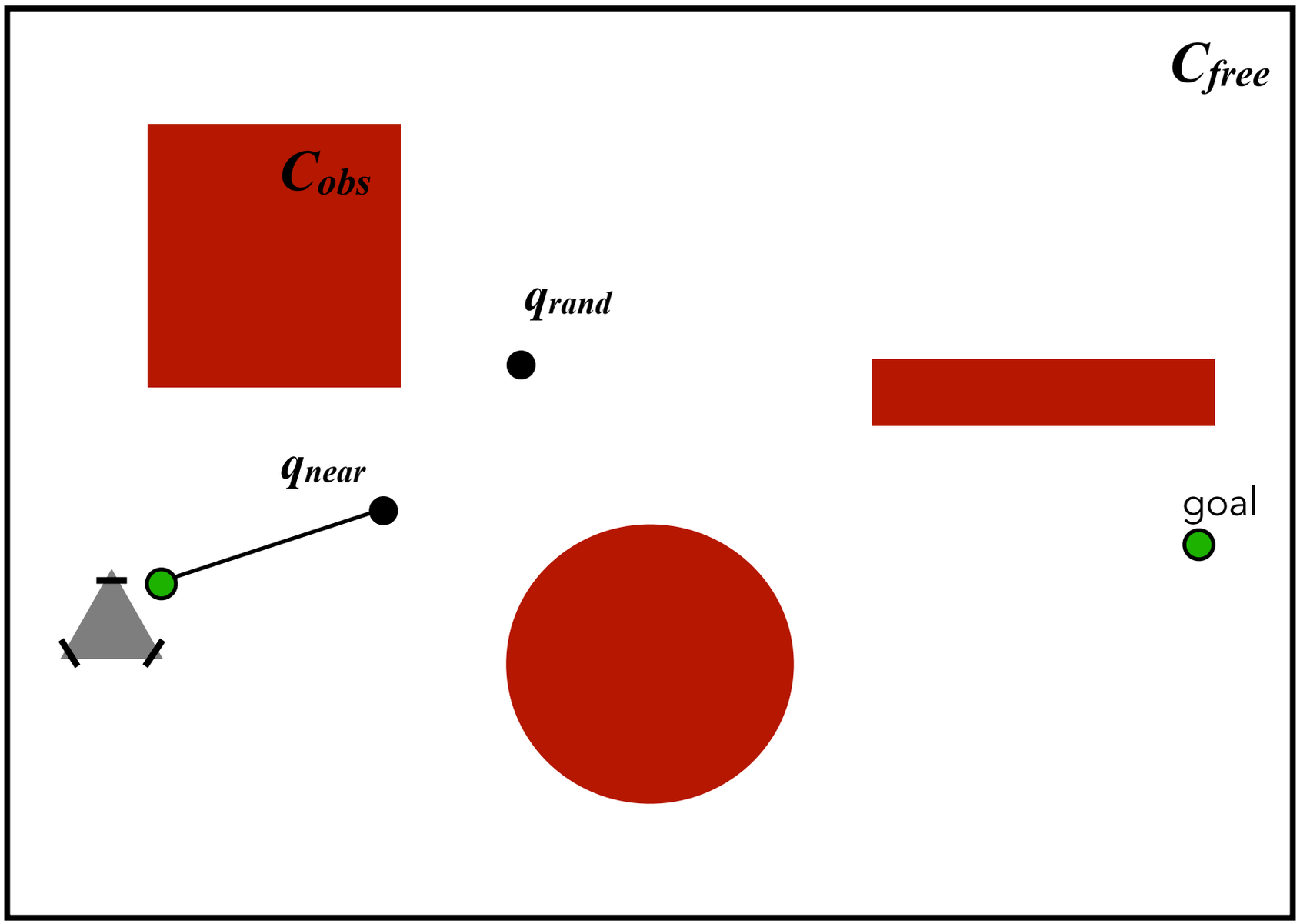
- LaValle and Kuffner, 2000.
- Similar to PRMs, but for single-query problems, a tree is enough.
- Basic idea: Build a tree by generating 'next states' in the tree, by executing random control

1. Initialize a graph G with a vertex corresponding to $\boldsymbol{q_I}$.
   Vertices correspond to configurations, edges to collision-free paths.
2. Sample a random state in C-space $\boldsymbol{q_{rand}}$ (with small probability, sample $\boldsymbol{q_G}$)
3. Find its nearest vertex $\boldsymbol{q_{near}}$ in G.
4. Local planner: For non-holonomic robots, find a new vertex $\boldsymbol{q_{new}}$ in C_{free} close to $\boldsymbol{q_{rand}}$ and check that edge $(\boldsymbol{q_{near}}, \boldsymbol{q_{new}})$ is collision-free. If so, add new edge and new vertex to G *(and discard* $\boldsymbol{q_{rand}}$*)*.
5. Check if G encodes a solution (path from $\boldsymbol{q_I}$ to $\boldsymbol{q_G}$), return G.
6. Return to step 2 or terminate with failure at time-out.

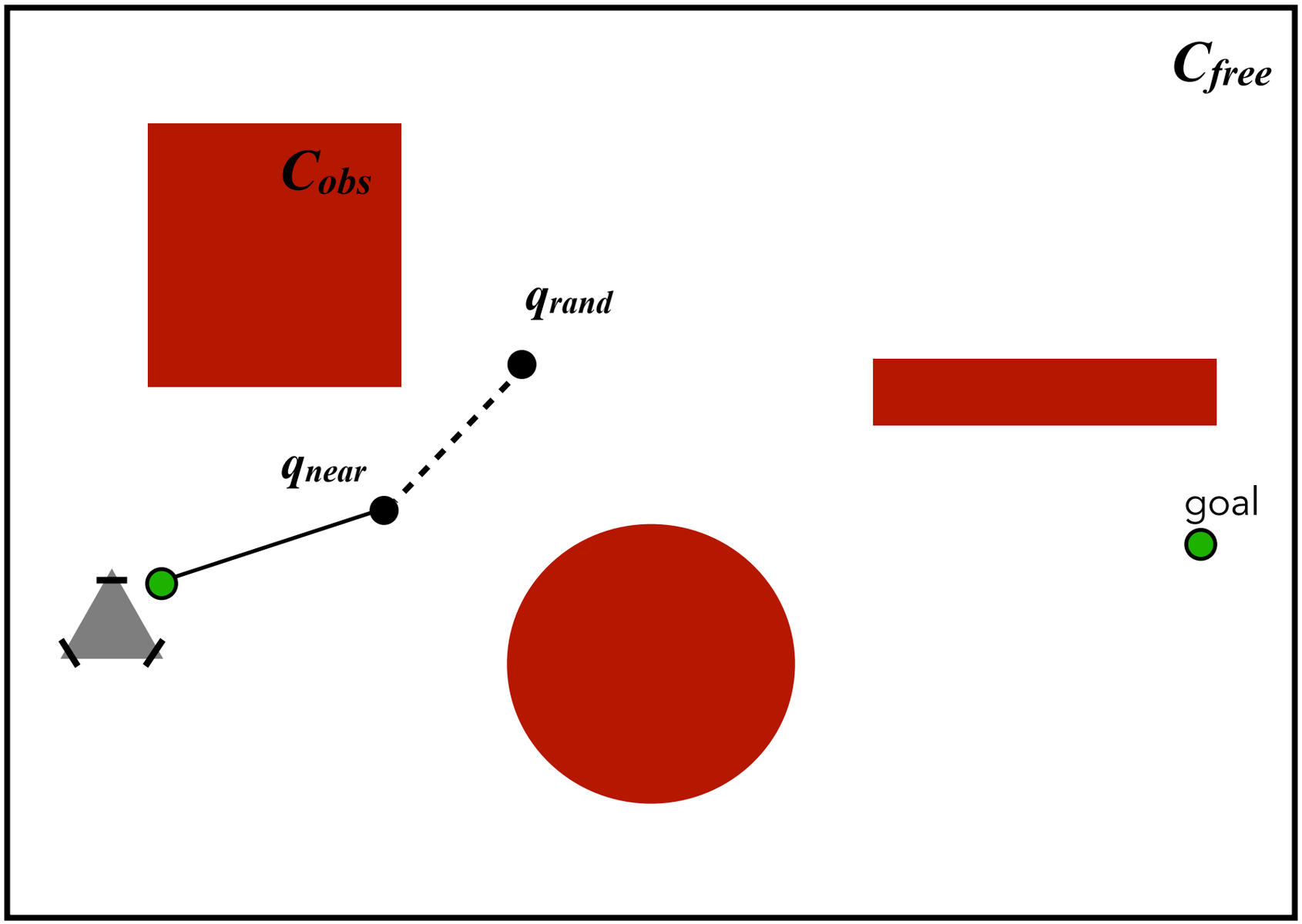
Rapidly Exploring Random Trees (RRTs)



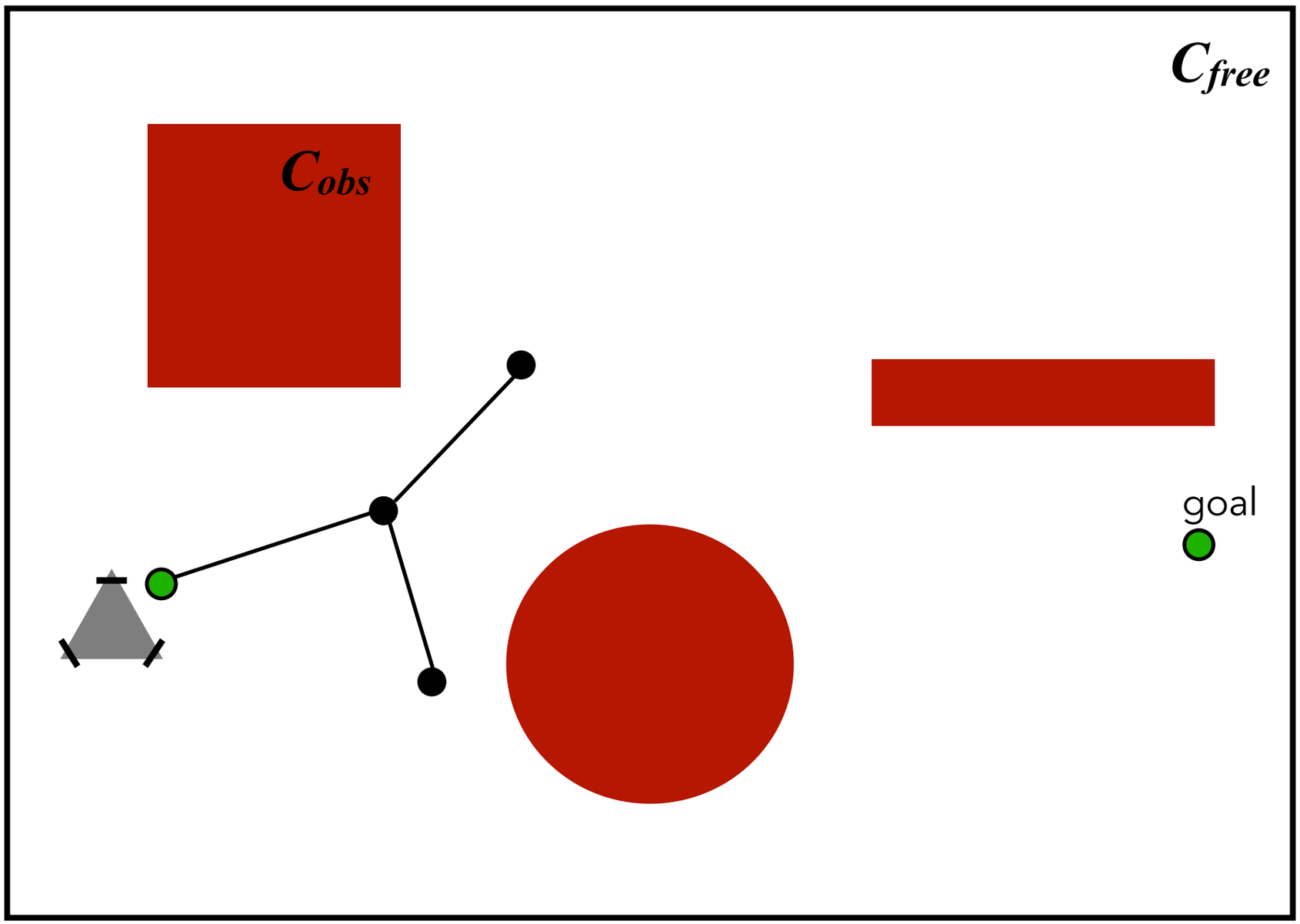
Rapidly Exploring Random Trees (RRTs)



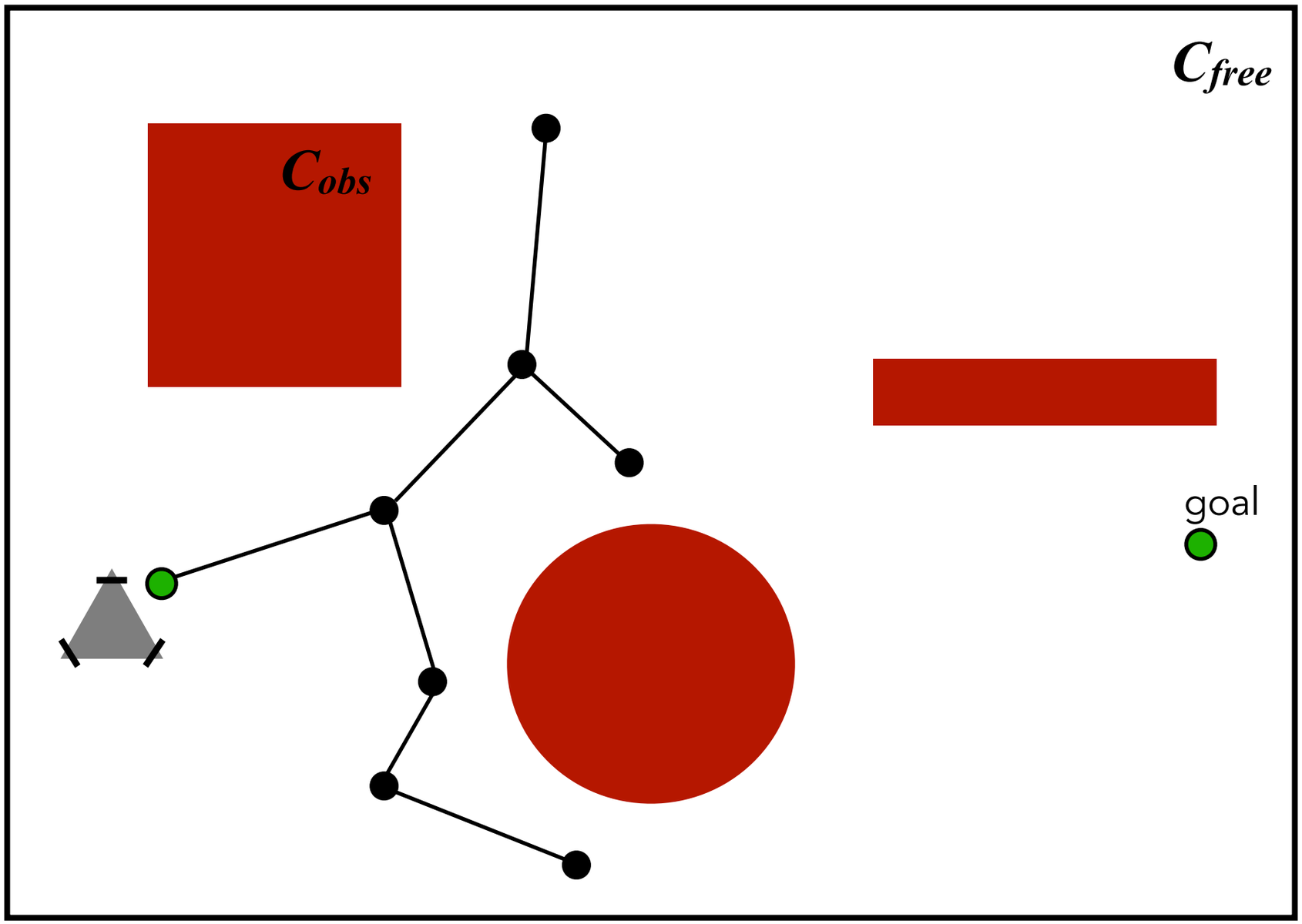
Rapidly Exploring Random Trees (RRTs)



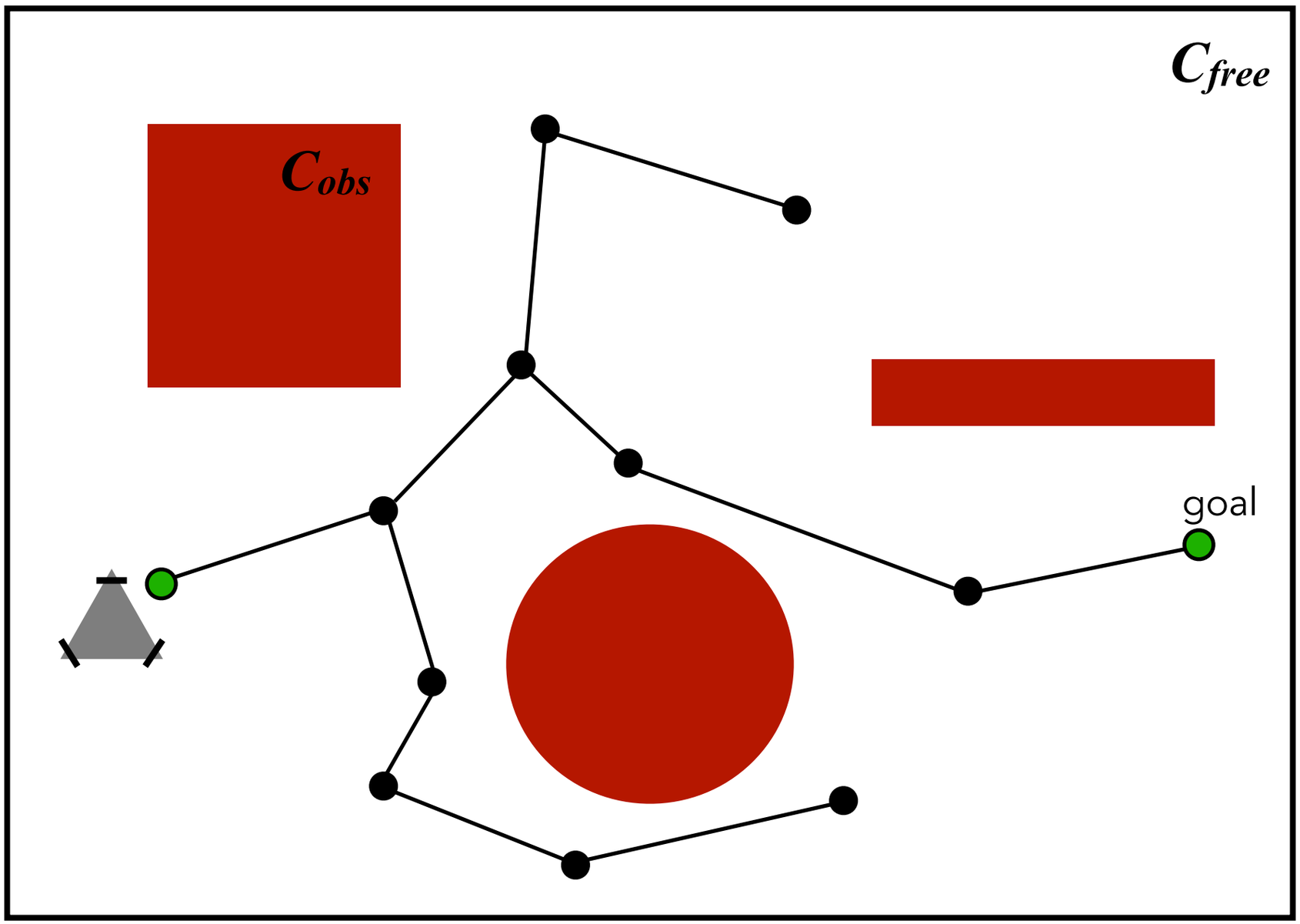
Rapidly Exploring Random Trees (RRTs)



Rapidly Exploring Random Trees (RRTs)



Rapidly Exploring Random Trees (RRTs)



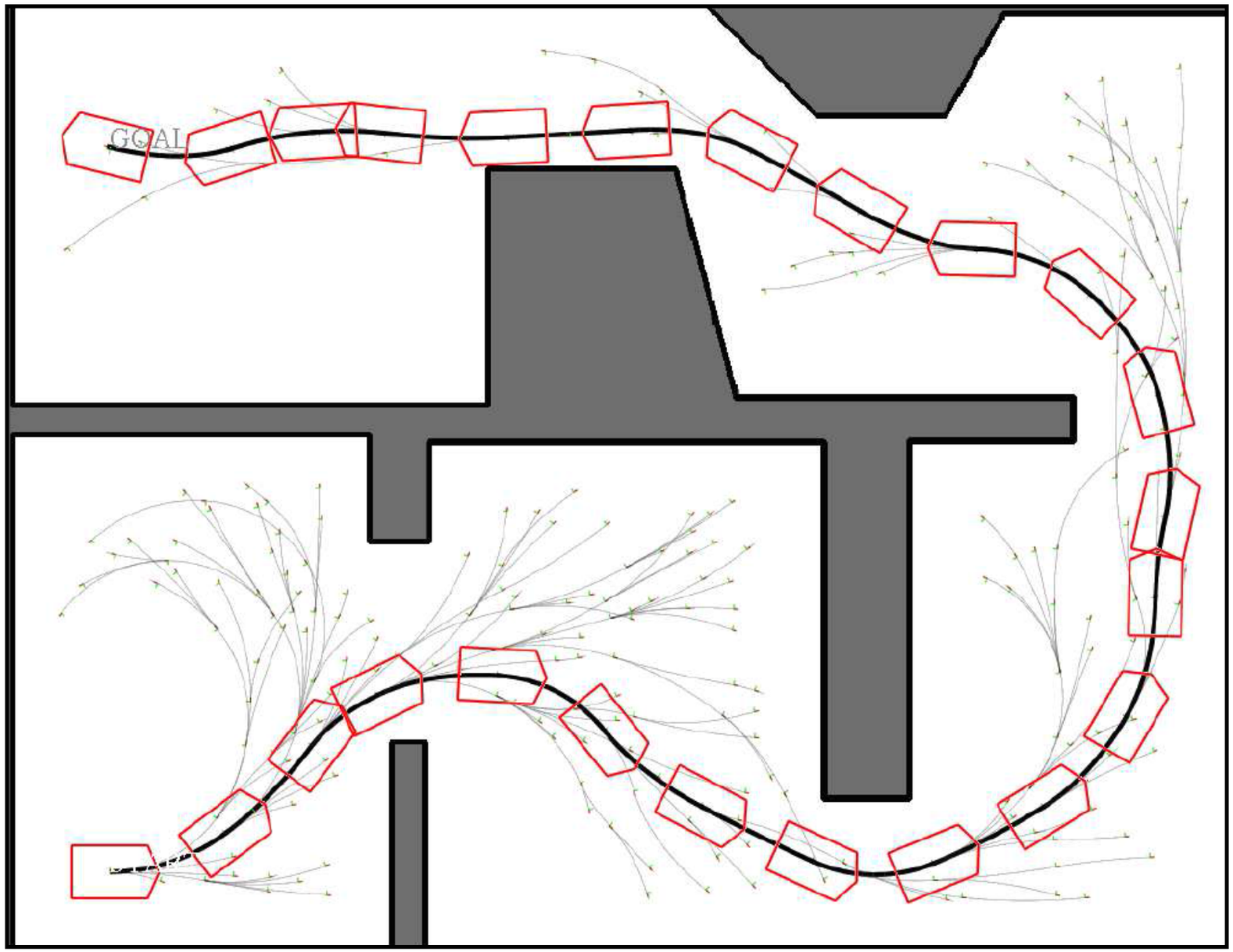
Rapidly Exploring Random Trees (RRTs)

- Primitives required:
  - Method to sample configurations in free C-space
  - Method to check for collisions in C-space
- Pros:
  - Probabilistically complete
  - Exponential rate of decay for the probability of failure
  - Asymptotically optimal (RRT*)
- Cons:
  - For non-holonomic robots, generating edges to sampled configurations requires solution to the *two-point boundary value problem*. (Alternate trick: use **motion primitives**).

* image credit: Kavraki, 2008

Rapidly Exploring Random Trees (RRTs)

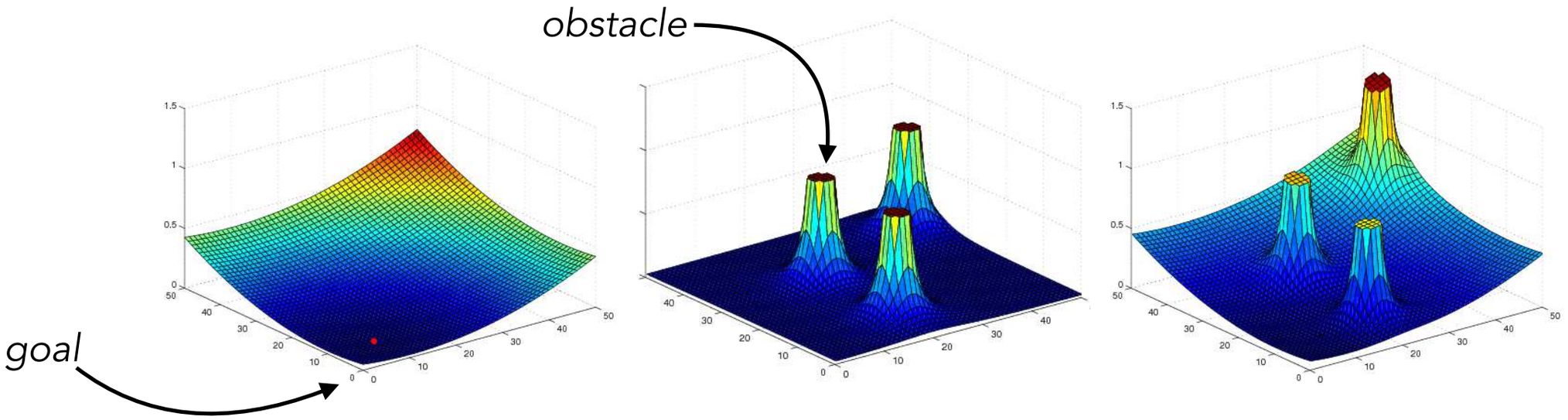
Example: Integration of motion primitives with RRT-based search.



Potential Field Method

- No explicit roadmap; instead, construct a real-valued potential function: $U : \mathbb{R}^m \mapsto \mathbb{R}$

  Attractive potential: $U_a(\mathbf{q})$ Repulsive potential: $U_r(\mathbf{q})$



Differentiate potential $\nabla U(\mathbf{q}) = \left[\frac{\partial U}{\partial \mathbf{q}_1}, \ldots, \frac{\partial U}{\partial \mathbf{q}_m}\right]^\mathsf{T}$

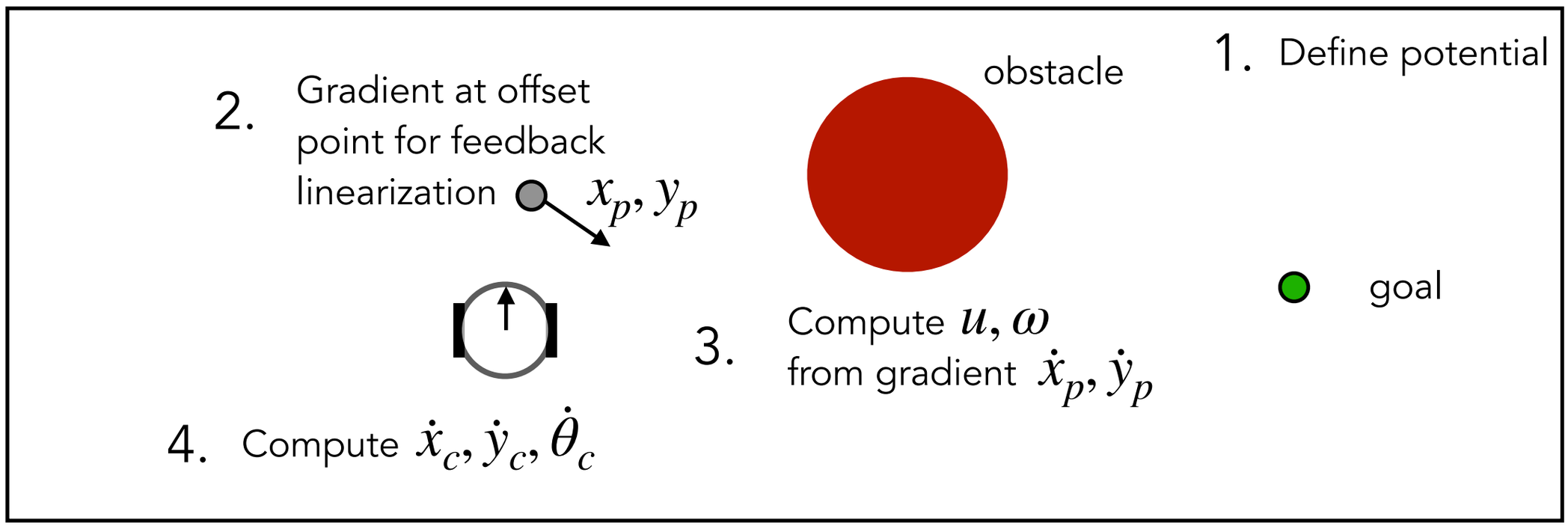
and compute a path via gradient descent (*gradient motion*).

* image credit: Kavraki, 2008

Potential Field Method

- Creating the potential field in C-space is hard.
- One possible strategy:
  1. Create potential field in robot work-space
  2. Compute gradient
  3. Use feedback linearization to get control inputs

4. (Compute gradient in C-space from control inputs)



Vector Fields

- Requirements
  - Representation of work-space / C-space
- Pros
  - Computationally efficient (once C-space is constructed)
  - More than just 1 path: motion strategy (feedback control) is readily computed from any point in field.
- Cons
  - Local minima (require methods to resolve them)
  - Representation of C-space is hard in high dimensions

Further Reading

Books that cover fundamental concepts:

- Steven M. LaValle. "Planning Algorithms". 2006. Cambridge University Press.
- http://planning.cs.uiuc.edu/

Seminal papers:

- S. M. LaValle and J. J. Kuffner; Rapidly-exploring random trees; 2001.
- S. Karaman, E. Frazzoli; Sampling-based Algorithms for Optimal Motion Planning; 2011
- J. Barraquand and J-C. Latombe. Robot motion planning: A distributed representation approach;1991.