

L04: Sampling-Based Planning

Planning Algorithms in AI

Prof. Gonzalo Ferrer,

Skoltech, 15 November 2021

Summary L04, LaValle 5.2-5.6

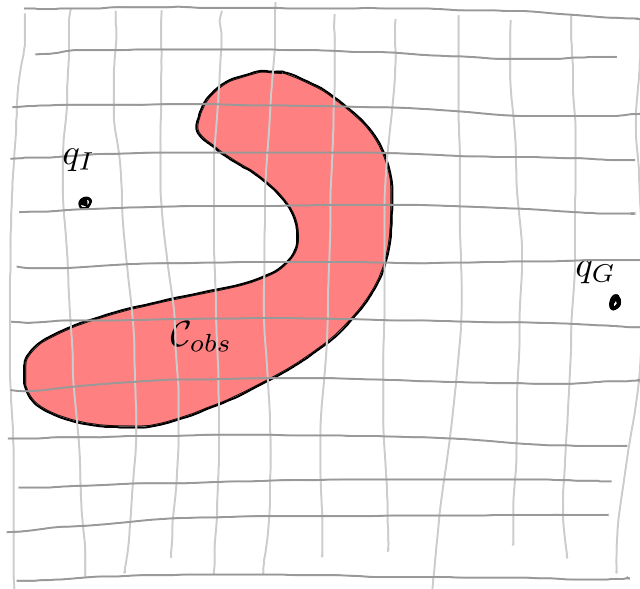
- ➔ Continuous state space and configuration space.
- ➔ Probabilistic completeness.
- ➔ Collision detection, convex hull, inflated points, etc.
- ➔ Probabilistic Roadmap (PRM).
- ➔ Rapidly-Exploring Random Trees (RRT).
- ➔ RRT*
- ➔ Other variants of RRT

Continuous state space

We will consider now **continuous** state spaces. As discussed in L03, the resultant topological spaces are very diverse.

What if we want to treat them as continuous variables? How can we plan?

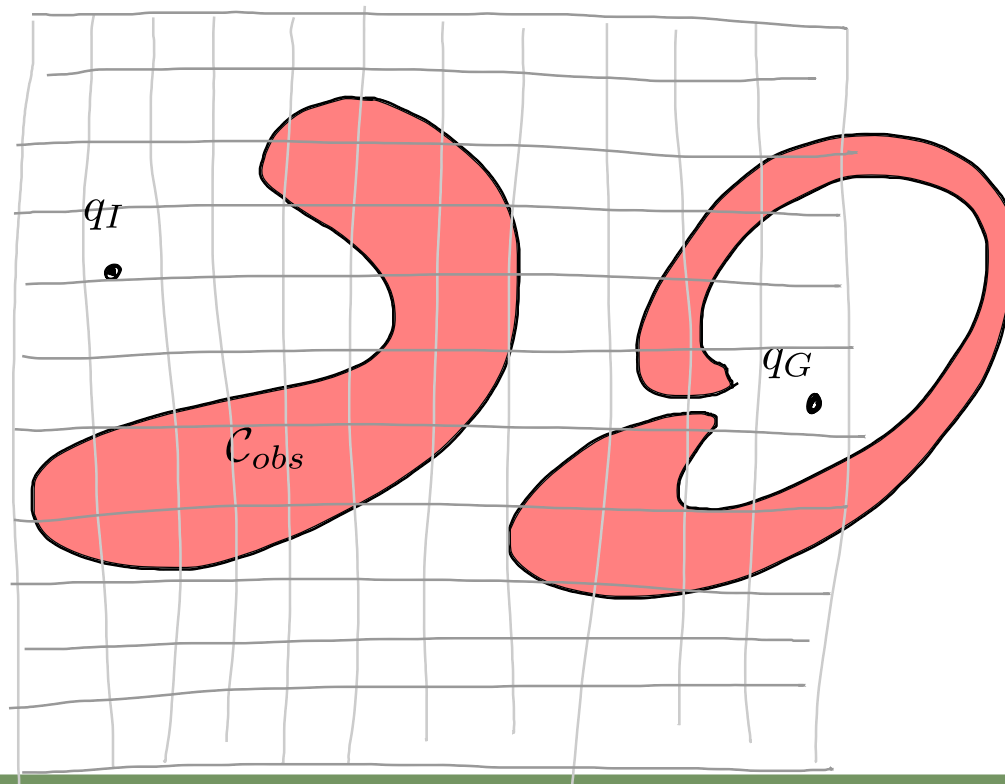
The first solution to try is **discretize** and apply algorithms from L02



Continuous state space

There is a problem with the parameters we choose for discretization:

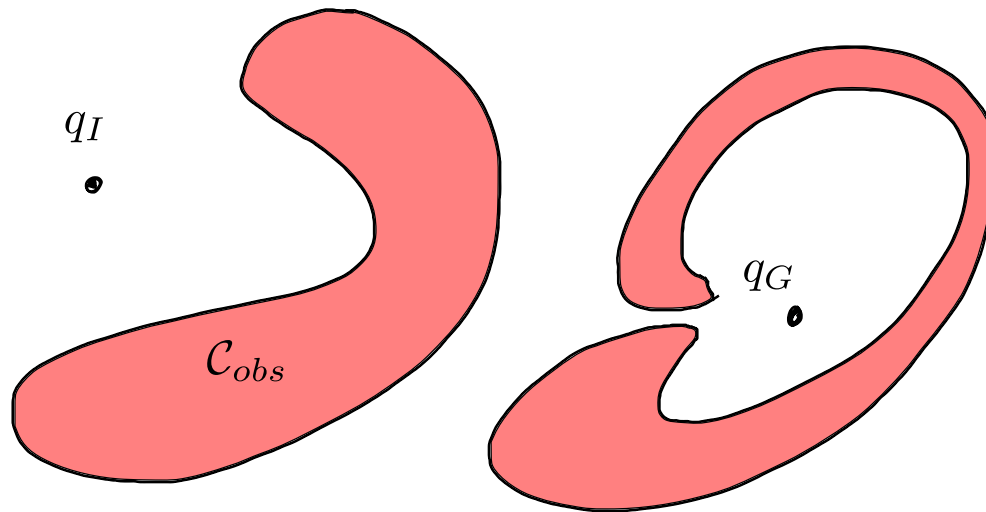
- Too small discretionary leads to visiting an unnecessary number of states
- Too large might miss important details



Continuous state space, second approach

Discretization has some issues, and needs parameters to be chosen...

What if now we simply sample the space?? The resolution adapts with the number of samples.




Completeness of sampling-based algorithms

Discrete planning guaranteed **completeness** of the algorithms: if there was a solution, it is guaranteed to be found in finite time. Recall that we justified by saying that graph search is a *systematic* algorithm.

Instead, the notion of a **dense** sequences becomes necessary: samples come arbitrarily close to any configuration as the number of iterations goes to infinity.

No we can define a weaker completeness: **probabilistic completeness**.
With enough samples, the probability to find an existing solution converges to one.

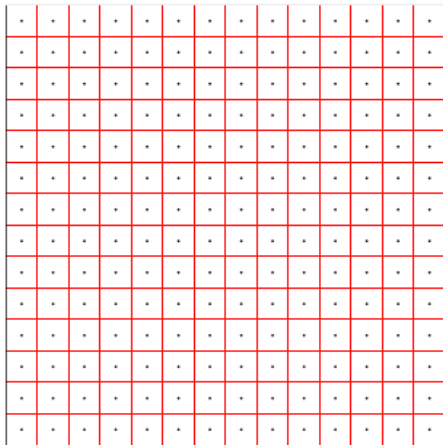


On discretization

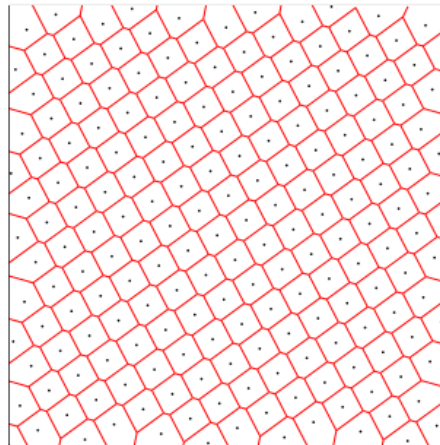
Other alternatives to sampling include:

Grids, Lattices, van der Corput sequences or random sequences from different distributions.

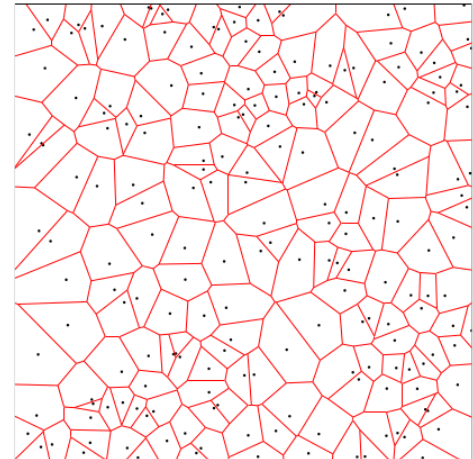
All these alternatives are dense or probably dense



Grid



Lattice



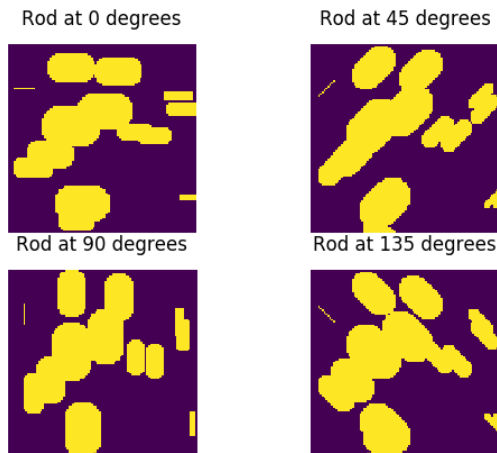
Uniform

Collision detection

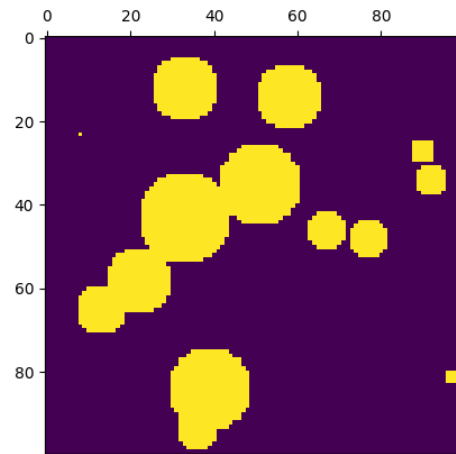
The second required ingredient for sampling-based planning is **Collision detection**. In L03, we defined the *obstacle region* as the set of all configurations what are not in collision:

$$\mathcal{C}_{obs} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}$$

An advantage of sampling is that we can avoid the *explicit* construction of the \mathcal{C}_{obs} and build an **implicit representation** as we sample. To do that, we need a collision detection at each configuration point.



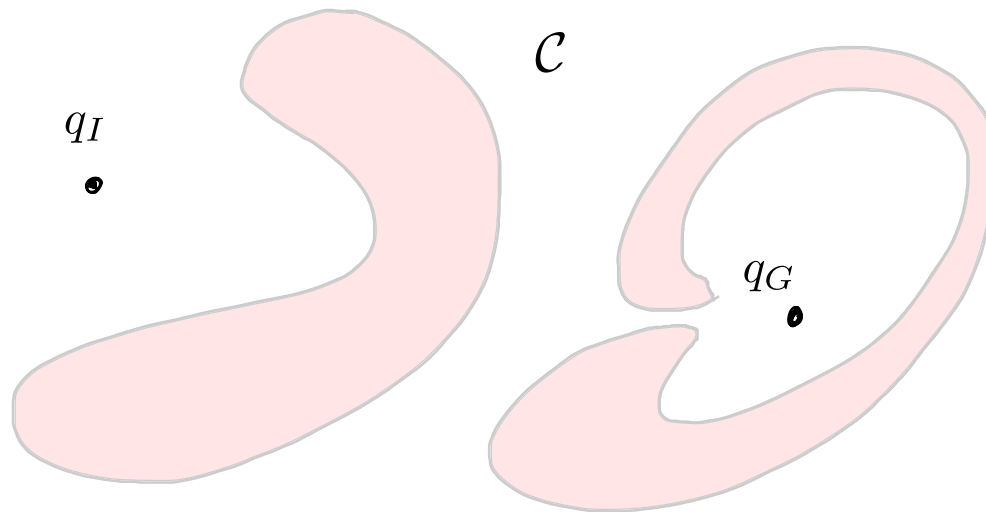
Explicit representation



Implicit representation

Collision detection, Implicit representation

The explicit construction of the *Cobs* is too costly to build. The sampling strategy builds an **implicit representation** as we sample:



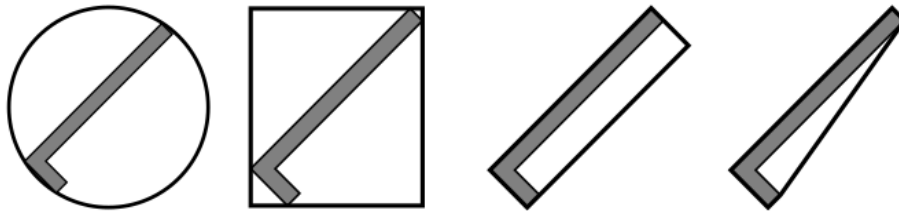
Collision detection: methods

- There are multiple ways to represent objects: Polygons, Polyhedral, Algebraic models, 3D triangles, meshes, NURBS, bitmaps, SDFs, etc.
- For each of the object representations, there is an exact collision checker, however it might be tremendously complex and calculating the full C_{obs} would be impractical. This is just a brief introduction to the topic.
- Regardless of the method, we can follow a 2-phase strategy:
 - Broad Phase: to avoid expensive calculations, we broadly describe objects as boxes or spheres.
 - Narrow Phase: details are checked. For some configurations (close to obstacles) this can be the only way.

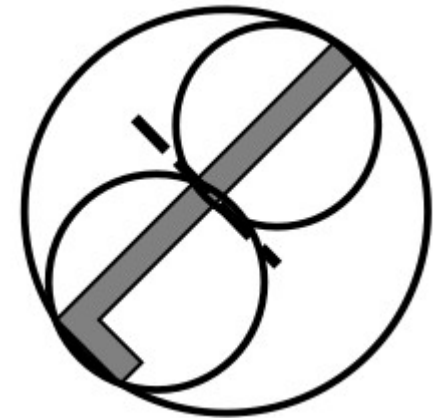
Collision detection: Hierarchical methods

For non-convex objects, one can define a hierarchy of sub-parts of the object and form a tree.

The bound region should fit the object (or sub-part) and the intersections should be as efficient as possible.

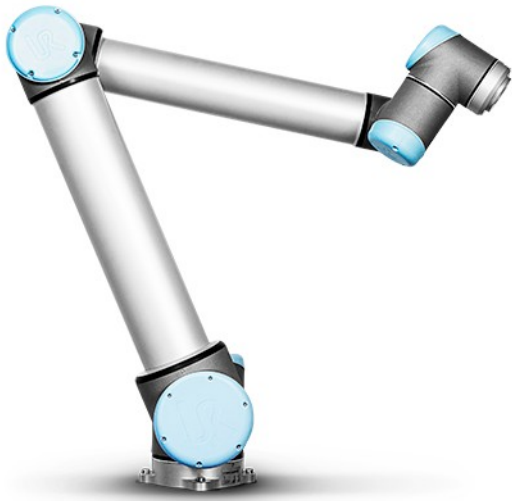


Four different bounding regions: a sphere, axis-aligned box, oriented bounding box and convex hull.



Tree of spheres

Collision detection: Hierarchical methods



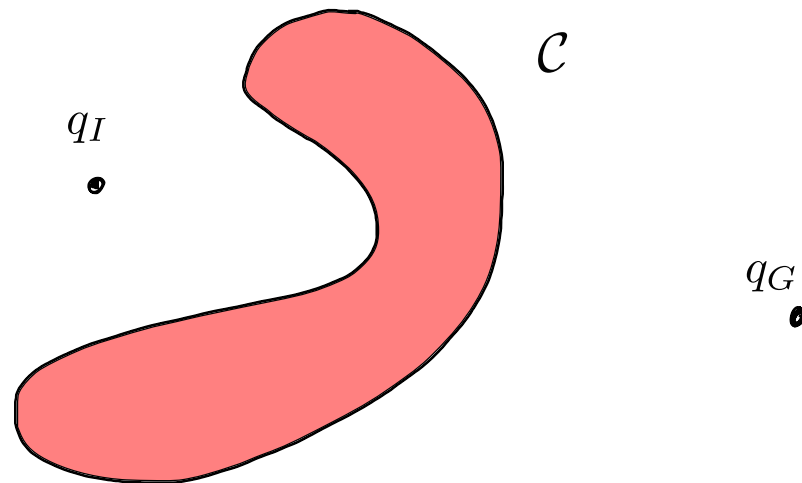
UR serial manipulator.



Fetch robot

Probabilistic Roadmaps

The idea is very simple: we sample the C-space and connect the new configurations as we are getting them.



PRM Algorithm

(I) Build roadmap:

```
1  G.init()
2  for i in [1,N]:
3       $q_{rand} = \text{sample}(C_{free})$ 
4      if  $q_{rand}$  in  $C_{free}$ :          (collision detection)
5          G.add_vertex( $q_{rand}$ )
6          for q in G.neighbourhood( $q_{rand}$ ):
7              if q in some_criteria:
8                  G.add_edge(q,  $q_{rand}$ )
9                  check_collision_segment(q,  $q_{rand}$ )
10 return G
```

(II) Query:

```
1  graph_search(G,  $q_I$ ,  $q_G$ )
```

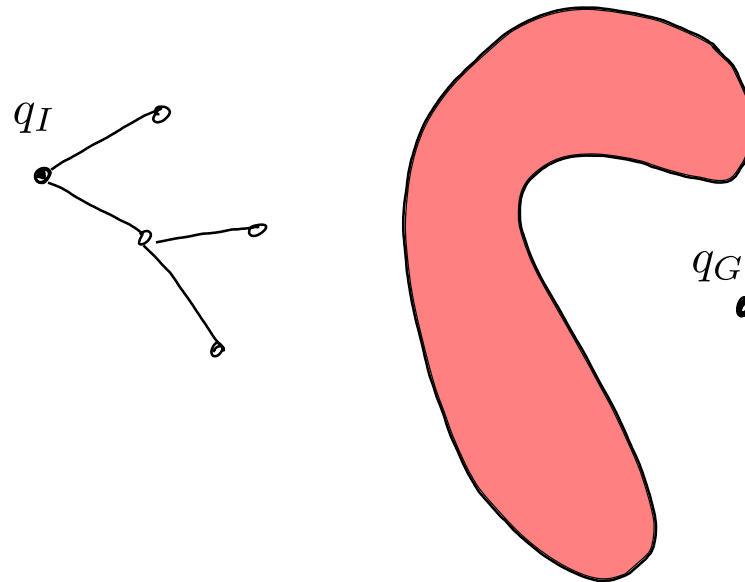
Therefore, the PRM can be considered a multiquery algorithm, once the roadmap is created, one can query as many times as needed.

Lazy PRMs

- The idea is to create a graph over the C-space and then evaluate for collision only as traversing the graph.
- This is a clever strategy to reduce the number of expensive collision checks.
- In case there is no obstacle, the graph search will bring you directly to the goal.
- In case there is a collision detected, the graph will be updated by removing the edge and search will be retaken.
- In practice works well.

Rapidly-exploring Random Trees (RRT)

The main idea is to build a tree and extend it to explore the C-space



RRT Algorithm

RRT:

```
1   $G.init(q_I)$ 
2  for  $i$  in  $[1, N]$ :
3       $q_{rand} = sample(C)$ 
4       $q_{near} = find\_nearest(q_{rand})$ 
4       $q_{new} = steer(q_{near}, q_{rand})$ 
5      if  $q_{new}$  in  $C_{free}$ :
6           $G.add\_edge(q_{near}, q_{new})$ 
```

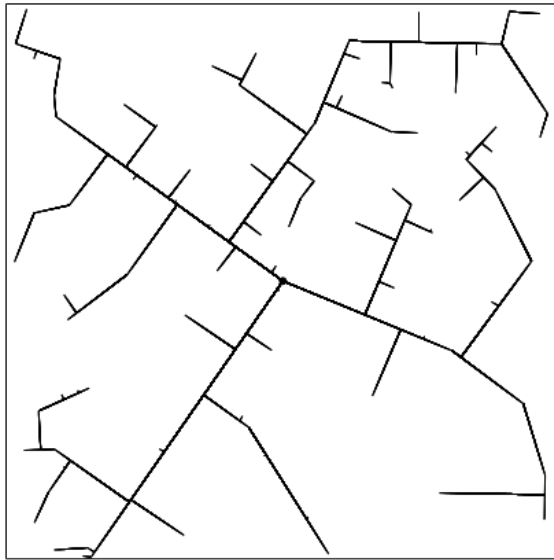
The *steer* function is usually a fixed step size, going on the direction of the random sample, but not all the way down.

If dynamics or other constraints are considered, it is the result of executing an action u and propagate the state through the **transition function**.

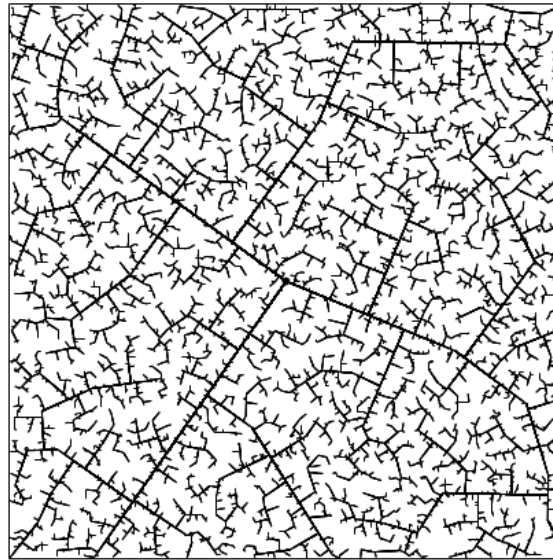
RRT, efficient exploration of the C-space

By expanding the tree it explores the space very efficiently.

On the other side, it is highly influenced by the order of the sampling sequence.



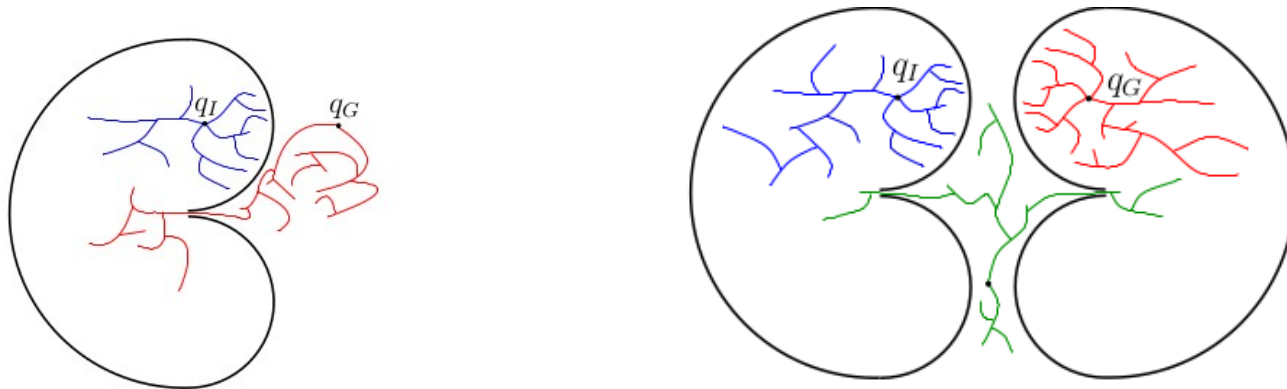
45 iterations



2345 iterations

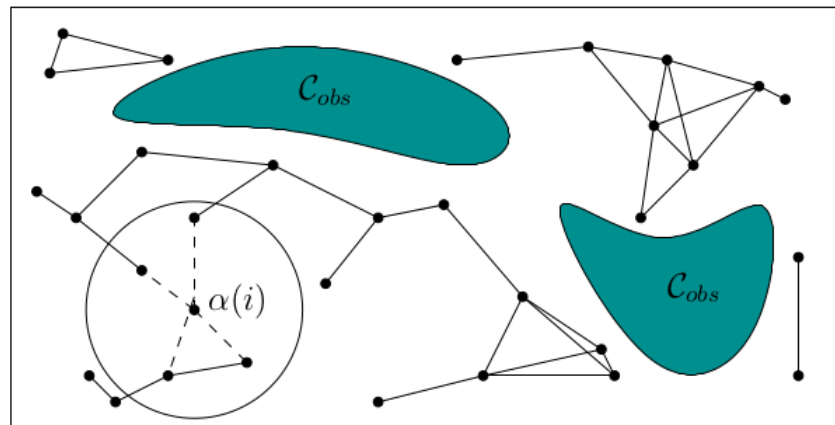
RRT improvements

- Sometimes we might want to sample the goal state with some small probability (1/100). The sampling sequence is still dense, but it might speed up things.
- Headed, connect larger steps if there is direct visibility.
- Changing the sampling sequence depending on the boundary of C_{free} .
- Bidirectional RRT, or multiple RRT's.



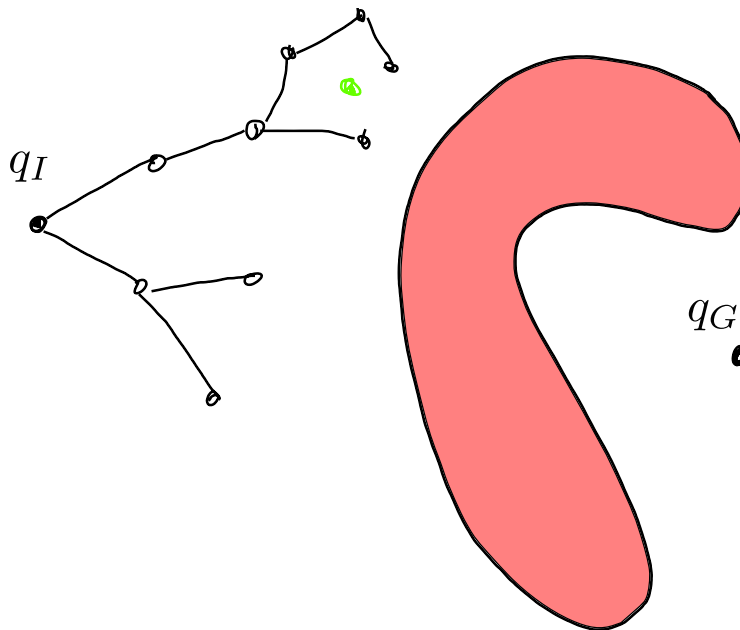
Sampling-based Optimal Planning: PRM*

- This family of algorithms look for optimality while planning. This means each vertex now contains some cost $C(q)$.
- They are provably asymptotically optimal.
- The idea of PRM* is that while we are sampling, we decrease the radius of search as a requisite to add a new edge.
- For the PRMs, the radius of search to connect a new node, decreases with the samples:



Sampling-based Optimal Planning: RRT*

- For the RRT*, optimality requires to change the configuration of the tree.
- Every time there is a new node added to the graph, the topology of the local graph is analyzed and updated if the cost is reduced. **Rewiring** process.



Other RRT variants

- Fast Marching Trees [1]: Also a sampling-based optimal planning technique, improving convergence rate.
- Informed RRT [2]:
- Mixing learning and Sampling-based algorithms, a very promising research direction.

[1] Janson, L., Schmerling, E., Clark, A., & Pavone, M. (2015). "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions." The International journal of robotics research.

[2] Gammell, J. D., Srinivasa, S. S., & Barfoot, T. D. (2020). "Batch informed trees (BIT*): Informed asymptotically optimal anytime search". IJRR

What are the drawbacks of sampling-based planning?

- In practice, sampling-based planning allows to plan in high dimensional spaces, something that limits discrete planning. However we don't have guarantees, only **probabilistic guarantees**.
 - This weak guarantees means in practice algorithms **may fail to converge**, running for an absurd amount of time.
 - The **rate of convergence** and the **complexity** of these algorithms is very hard to establish, even more for sampling-based optimal planning.
 - For really high dimensional spaces, even the sampling-based planners described here would have a hard time finding a solution.
 - Even though, these are the most popular techniques for planning currently used.
- 