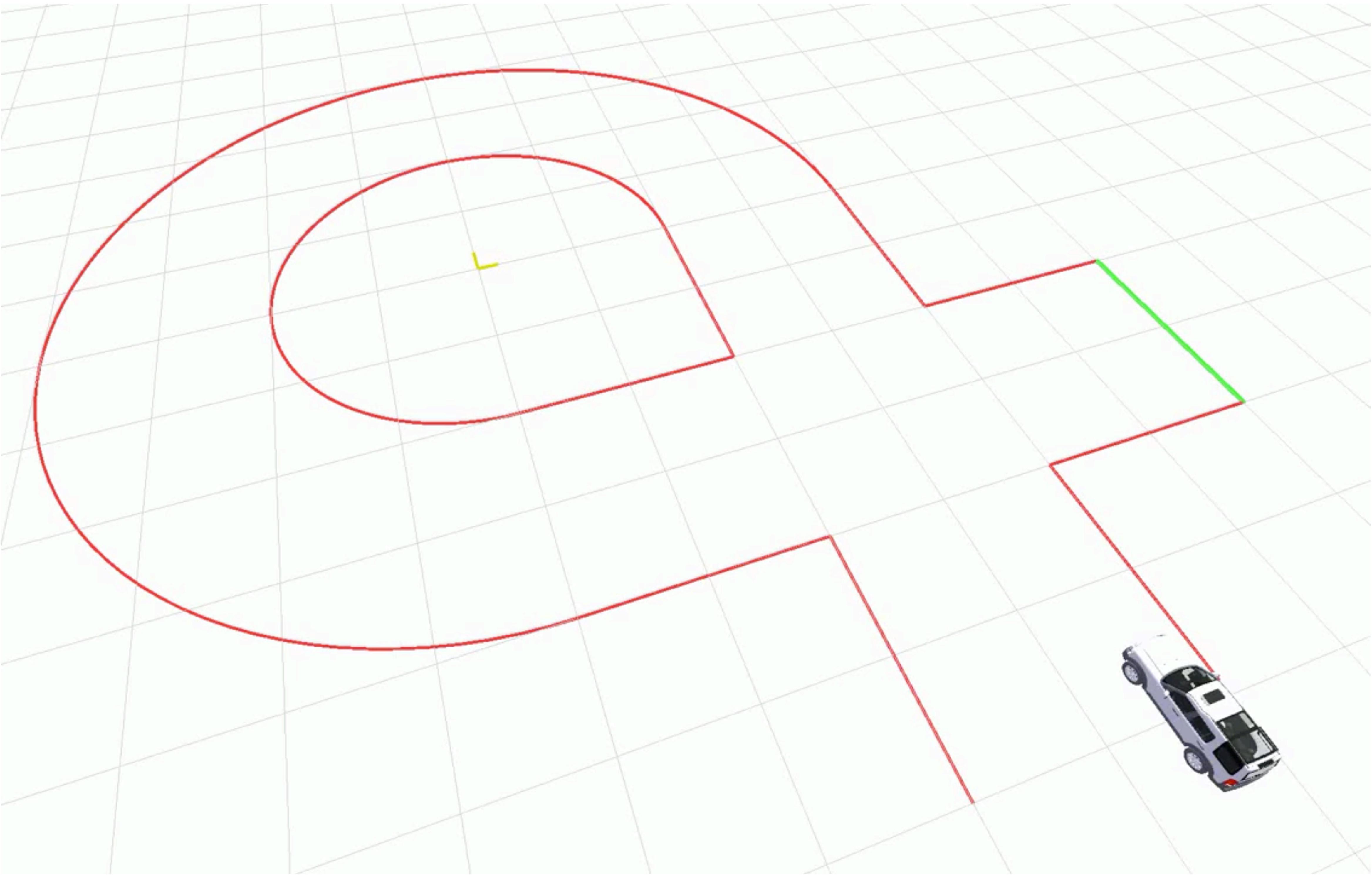
Incremental sampling-based planning methods

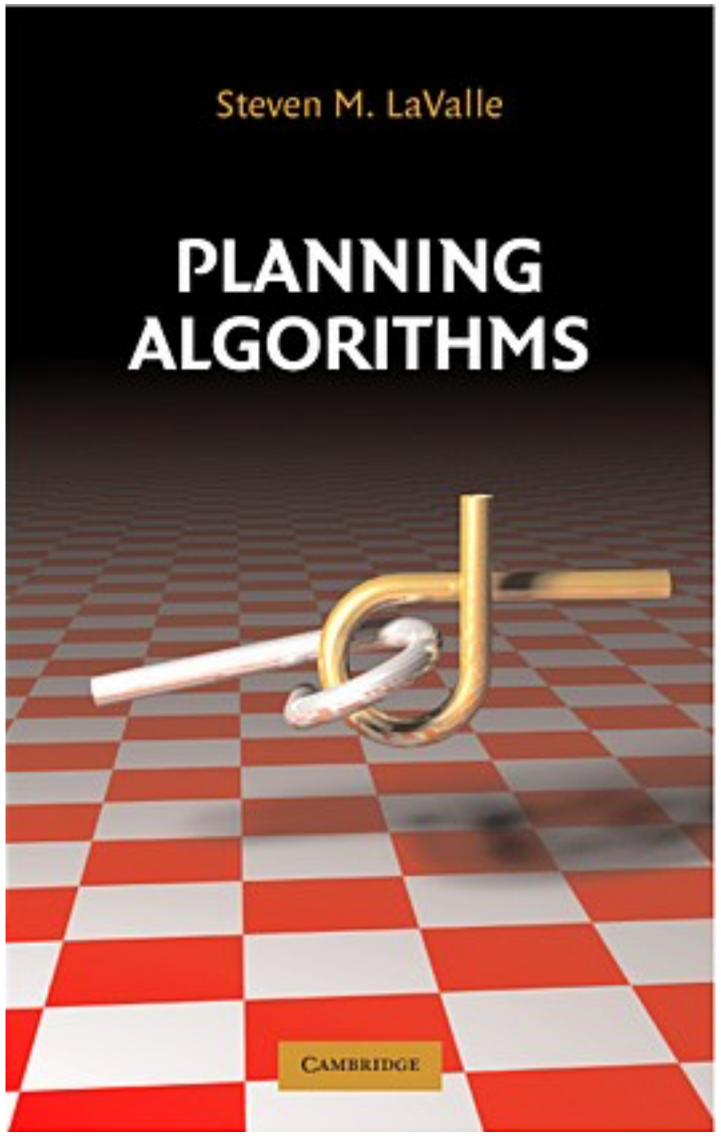


Sertac Karaman, Emilio Frazzoli

Overview

- **Incrementally building a graph**
  - Steering function as local planning
  - Collision checking
- **Optimality, completeness properties**
- Algorithms:
  - **Probabilistic roadmaps (PRMs)**
  - **Rapidly-exploring random trees (RRT)**
  - **RRT*** - asymptotically optimal variant
- **Conclusions on motion planning**

http://planning.cs.uiuc.edu/



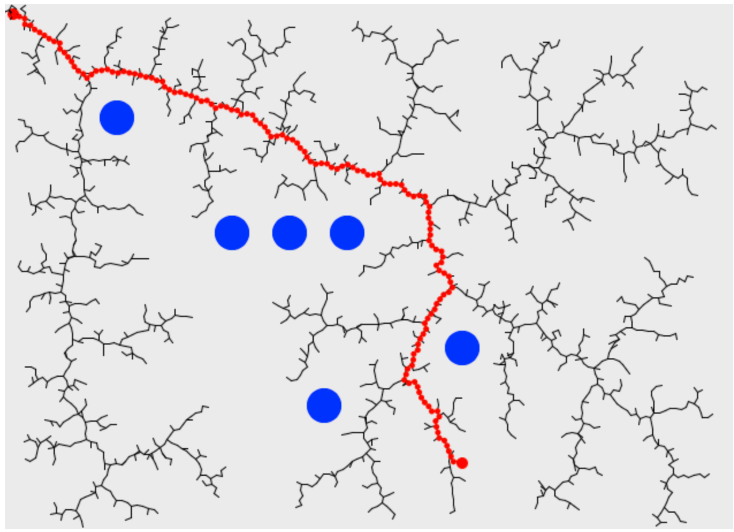
Chapters 5, 14

Incrementally building a graph for planning

- The methods we consider are a **random process in the space of graphs**.

$$\boxed{\text{graph building}} \longrightarrow G_1, G_2, G_3, \ldots$$

$$G = (\mathcal{N}, \mathcal{E})$$



- Some of the methods are **monotone**: only **add** nodes and edges to the graph.

$$\mathcal{N}_i \subset \mathcal{N}_{i+1} \qquad \mathcal{E}_i \subset \mathcal{E}_{i+1}$$

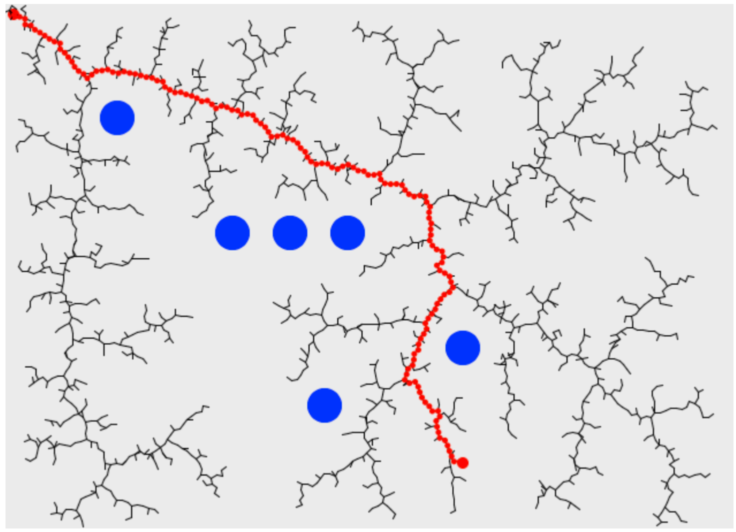
- But, we will see that an optimality result requires "rewiring" the graph, always adding nodes, but sometimes changing the edges.

Incrementally building a graph for planning

- The methods we consider are a **random process in the space of graphs**.

$$\boxed{\text{graph building}} \longrightarrow G_1, G_2, G_3, \ldots$$

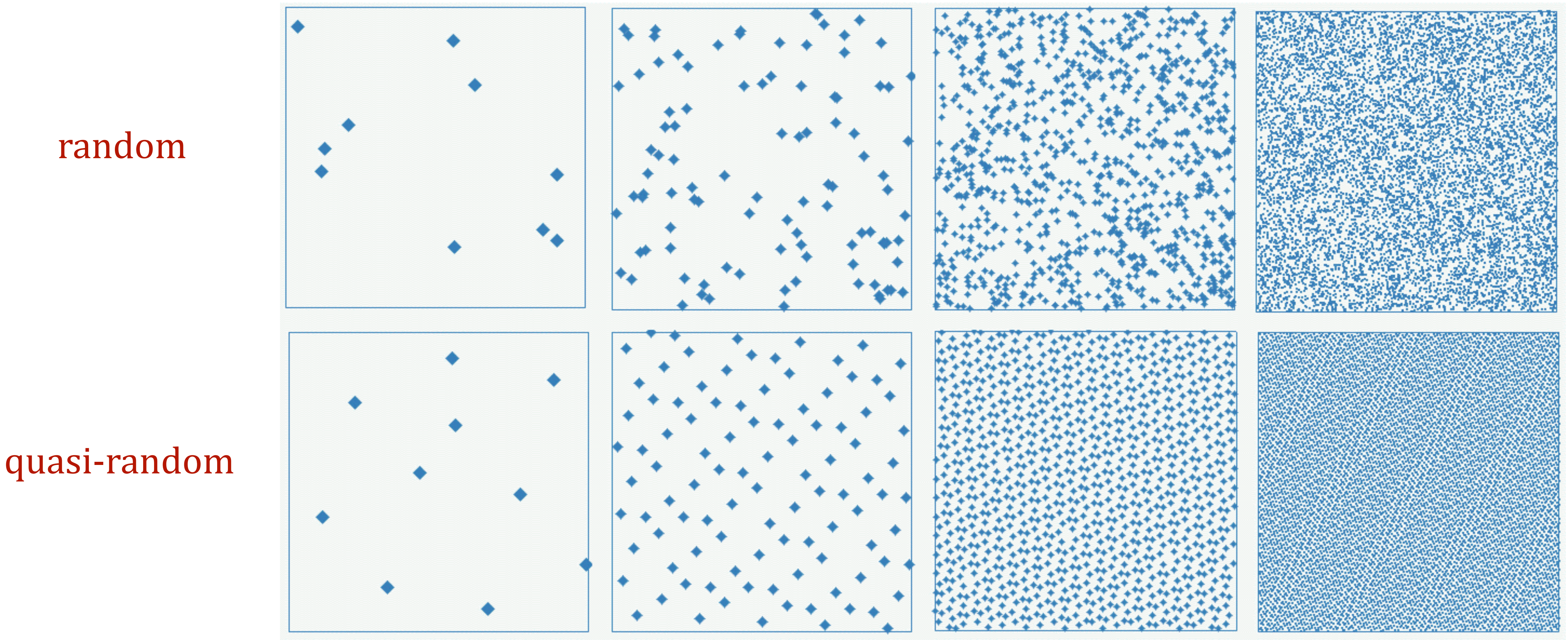
$$G = (\mathcal{N}, \mathcal{E})$$



- The main ingredients:
  1. How to “seed” the graph
  2. How to sample a new node
  3. How to choose which other node it might connect to
  4. How to decide which edges to add
  5. How to decide which edges to remove

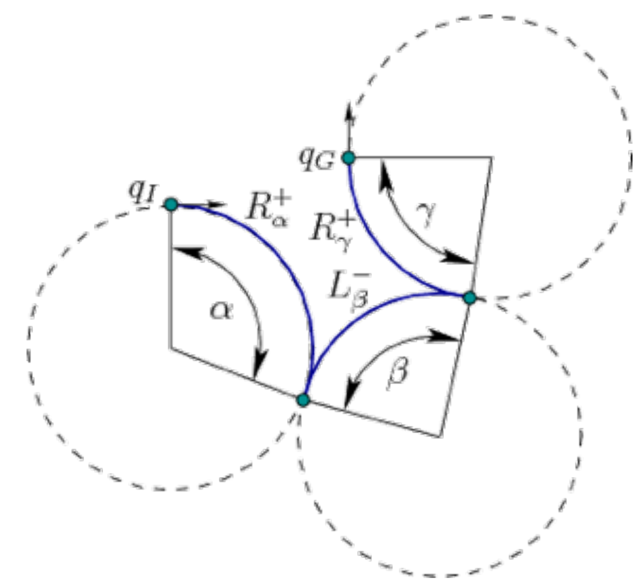
Sampling sequences

- We need to sample a sequence of points in configuration space
  - **Not necessarily random.**
  - We want it to have **low discrepancy**
  - **Not aligned** with the coordinate axes

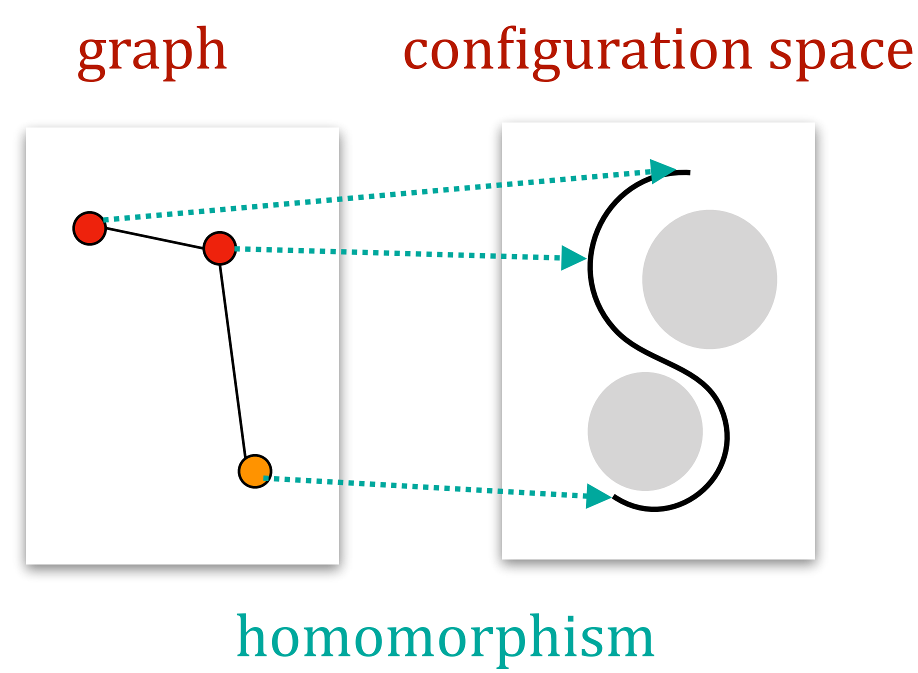


Steering functions as local planning methods

- Recall: the **steering function computes a feasible path given two nodes**.
    - Closed form solutions for Dubins, Reeds-Shepp, differential drive.
    - Otherwise: solve a boundary value problem.



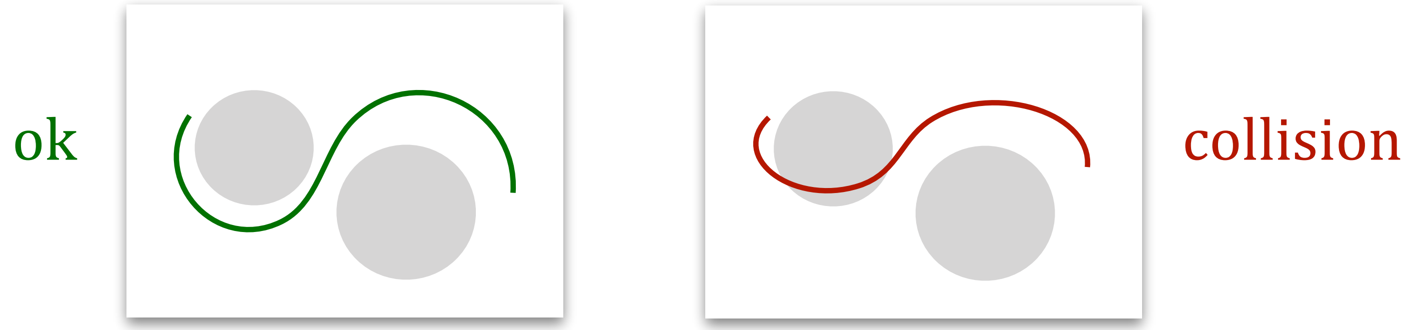
- The **feasibility invariant** in graph construction:
  If two nodes are connected by a path,
  there is a feasible path between
  their corresponding points in configuration space
    - We will not need to remember *which* path.



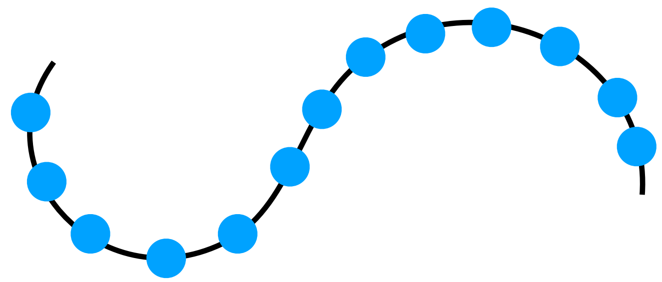
- It's ok if the steering function is not complete, though it will make the overall algorithm slower.

Collision checking

- We need a way to check if a path belongs to the free configuration space.



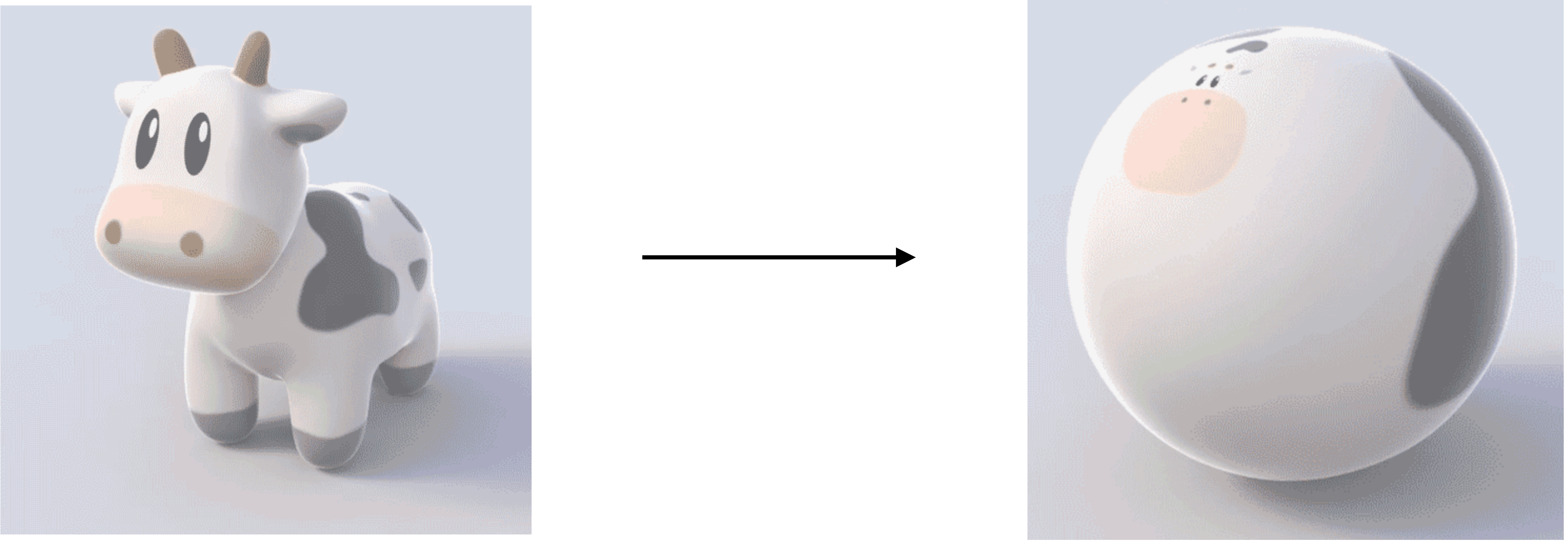
- If you know how to check if a **point** is in free configuration space, then you can check a path by checking its points at a given interval.



- It's **ok** if your collision checking method is *a bit* conservative (pessimistic), though it reduces the size of the solution set.

Example of conservative collision checking

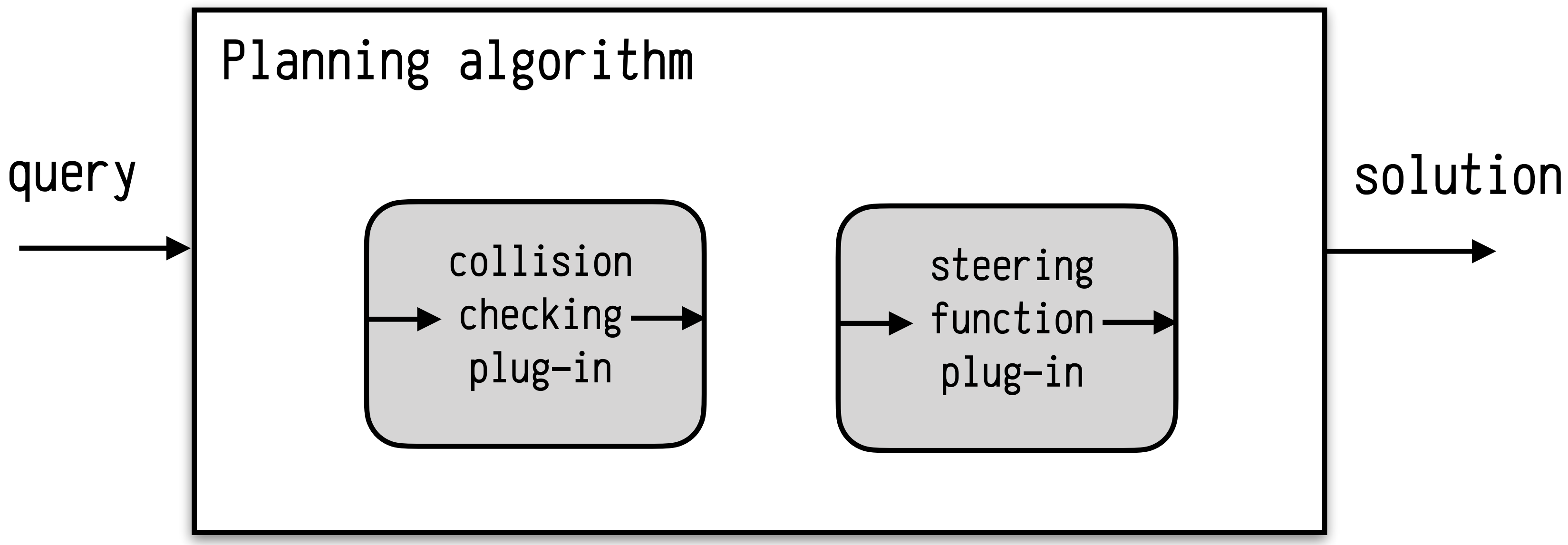
- **Assume all cows are spherical:**



- Now you can collision-check your cows
  by simply computing the distance between their centers.

What's nice (1): working with black boxes

- The **steering** and **collision checking** functions are used as **black boxes** and they are decoupled.



- You can make a very generic algorithm and add "plugins" for:
  - new dynamics ➠ new steering function
  - new environments ➠ new collision checking

What's nice (2): robustness

- Within reasonable limits:
    - It's ok if the steering function is not complete.
      e.g. only works if points are "close enough".
    - It's ok if the collision checking is conservative.
- Algorithm will be slower but still complete
  if you have not pruned all feasible paths.

- You can explore the **trade-off space**:
    - More precise steering/collision checking
      but fewer overall iterations.

      $\updownarrow$

    - Faster steering / collision checking
      but overall more iterations

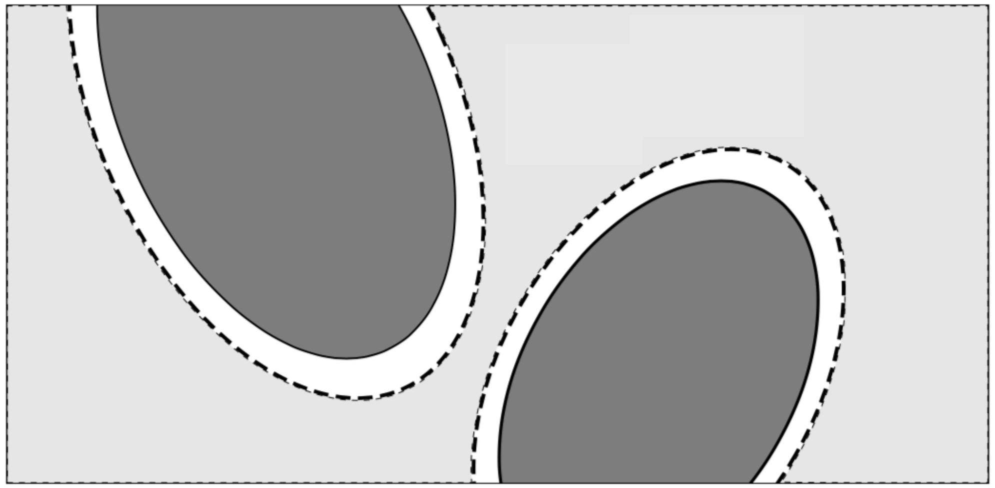
Properties of incremental algorithms

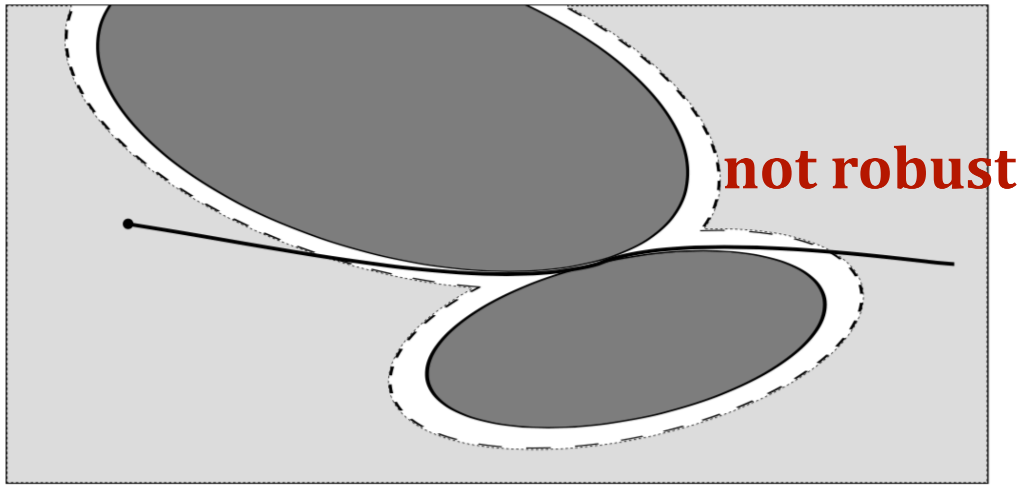
- For **incremental algorithms**, we have two properties of interest:
  - **Probabilistic completeness** ≡ **we are guaranteed to find a solution**, for any robustly feasible motion planning problem.
  - **Asymptotic optimality** ≡ **the solution will be optimal**, for any robustly feasible motion planning problem.

- *Robustly feasible problem* ≡ not a pathological case (definition in the next slide).

Robust problems and solutions

- A robustly feasible problem is one where the solution is robust.
- *Definition:* A **solution is robust** if it remains a solution when the obstacles are infinitesimally dilated by a small δ.





- Equivalent: if a path is a solution, there is a neighbourhood of the path whose points are solutions.
- Robustly optimal problem ≡ the optimal solution can be obtained as a limit of robust solutions.

Probabilistic completeness

- *Definition:* An algorithm is **probabilistically complete** if, for any robustly feasible motion planning problem, it **will eventually find a solution** with probability 1 as the iterations *N* grow:

$$\lim_{N \to \infty} \Pr(\text{algorithm finds a solution}) = 1$$

- Note that this does not tell us much about the performance.
  - Example in another domain:

    A very simple sorting algorithm:
    apply a random permutation to the list, then verify if it is sorted.

    probabilistically complete!

Asymptotic optimality

- *Definition:* An algorithm is **asymptotically optimal** if, for any robustly optimal problem, eventually it **will find a solution with the optimal cost** as the iterations N grow:

$$\lim_{N\to\infty} \Pr(\text{cost of solution} = \text{optimum}) = 1$$

- Note that also this doesn't tell us much.
  - Example:

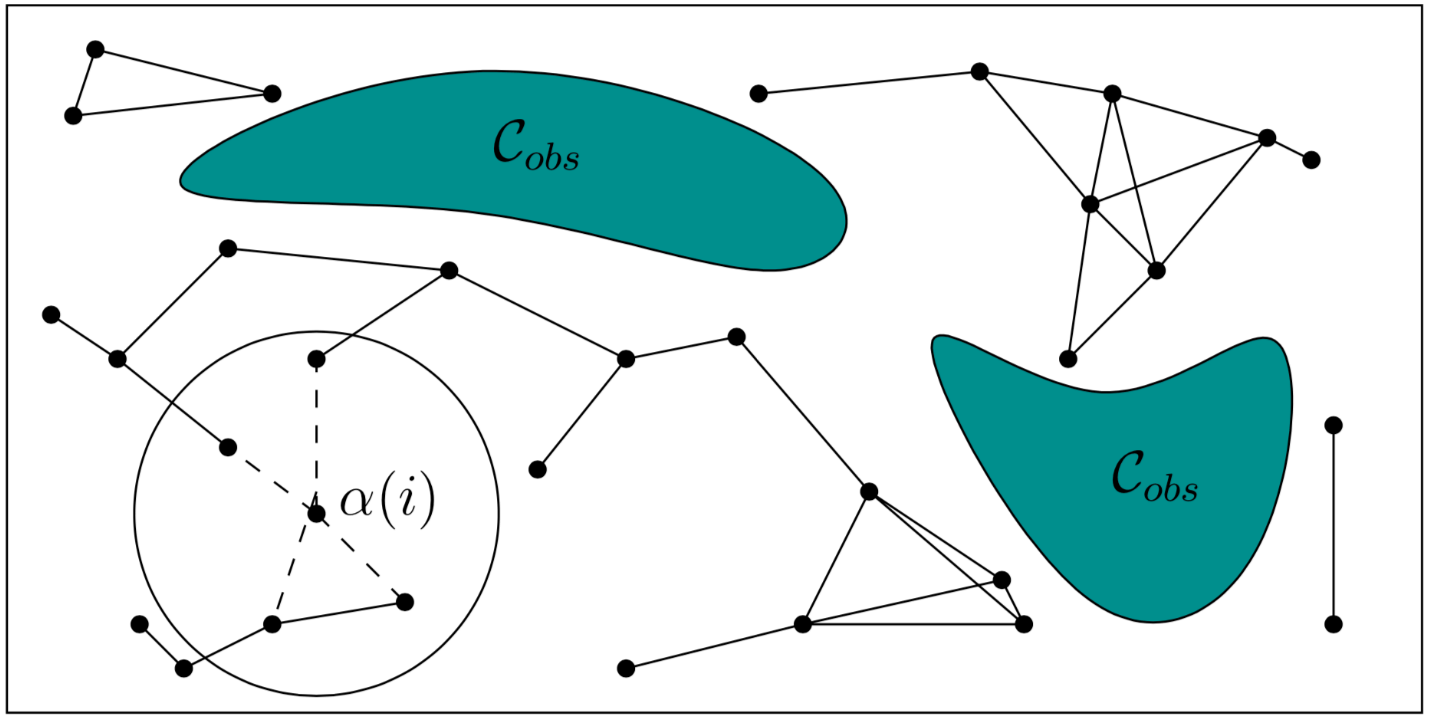
    To minimize any function f(x),
    sample x randomly, and remember the best option.

    Asymptotically optimal!

- Asymptotically optimal $\Rightarrow$ probabilistically complete, but not viceversa.

Probabilistic RoadMaps (PRM)

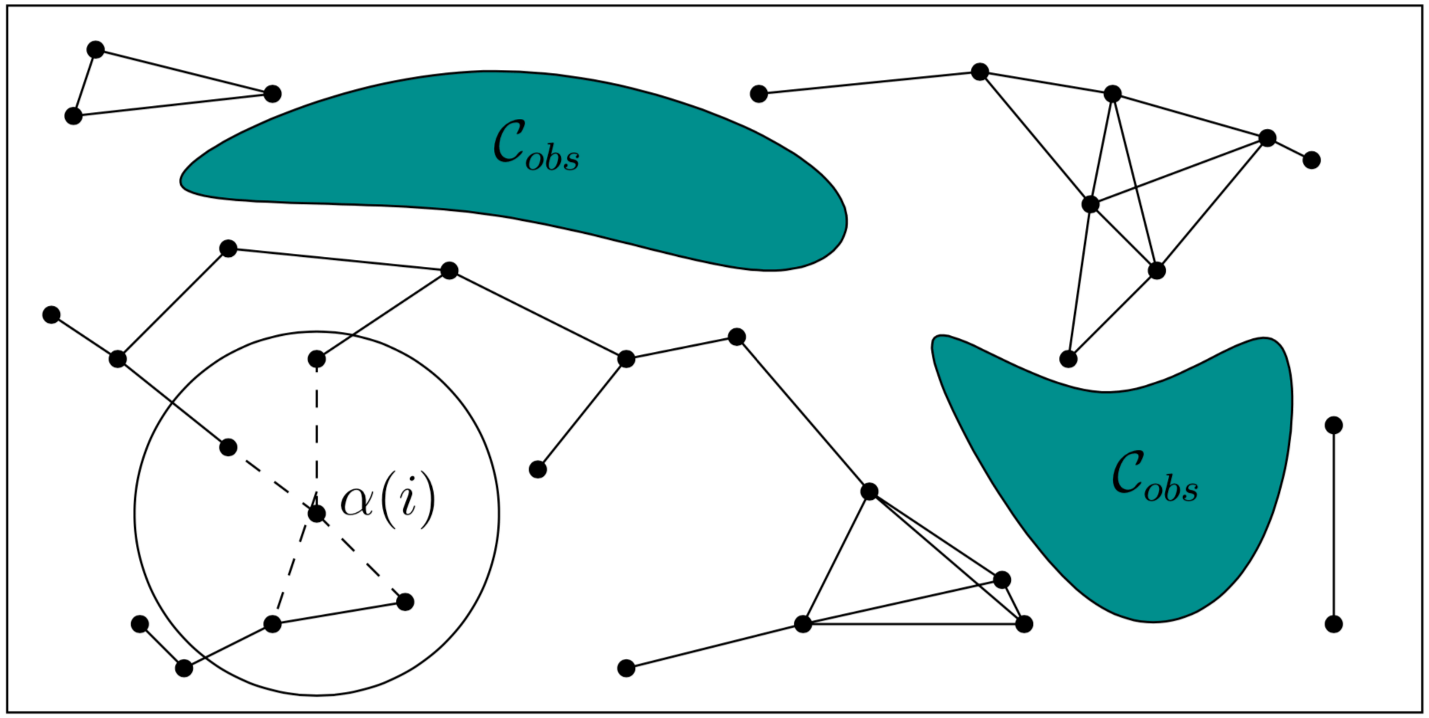
- Kavraki, Latombe 1996
- **Pre-processing stage:**
  - Sample *n* points from the sequence α.
  - Try to connect each point to the other points in a radius R.
    - Steering function + collision checking
  - Only allow up to k incoming connections.
- **Query stage:**
  - Connect start and end point to the closest points on the roadmap.
  - Find a path on the roadmap.



α: sampling sequence

Probabilistic RoadMaps (PRM)

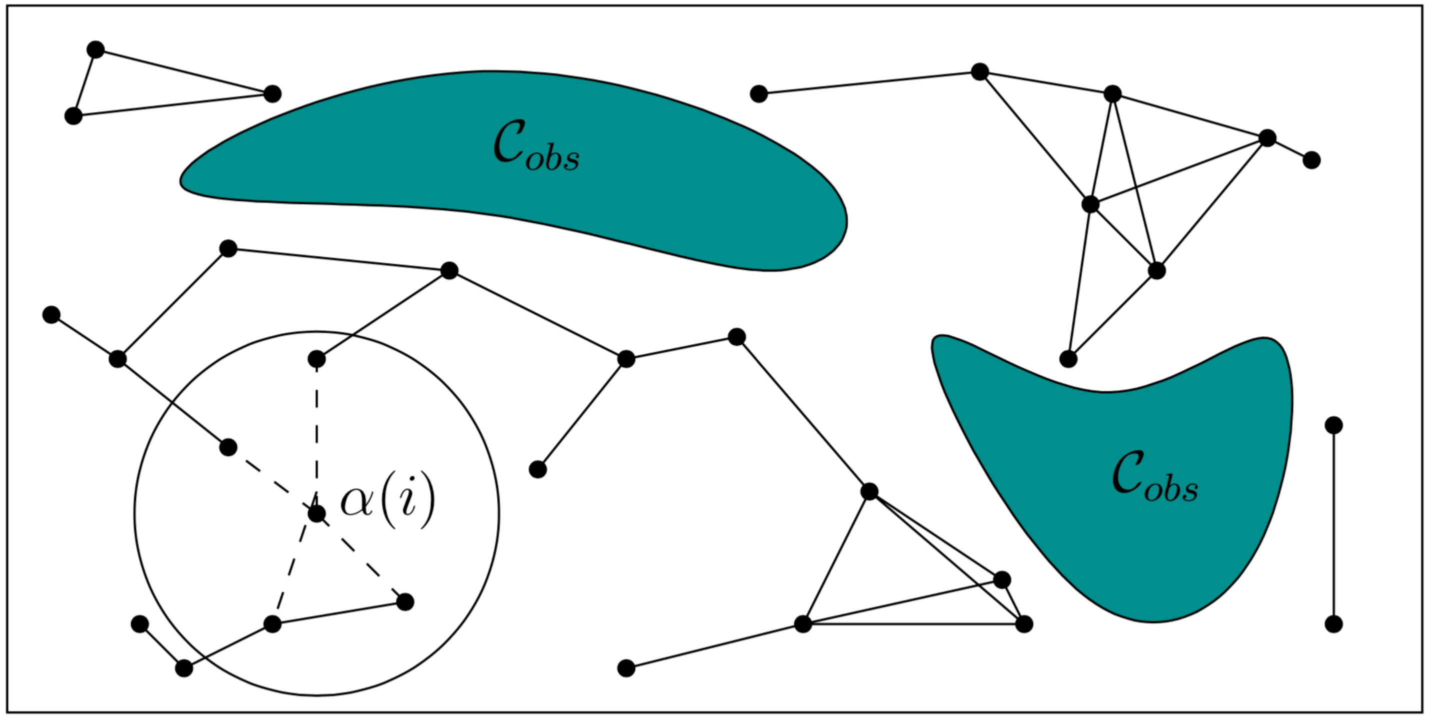
- **Useful for multiple queries** - you can reuse the graph.
- **Inefficient for single query** - the graph is independent of start and end points.
- How to choose the radius *R*?
- We can prove the following:
  - PRM is **probabilistically complete**
  - PRM is ***not*** **asymptotically optimal**.



α: sampling sequence

Probabilistic RoadMaps (PRM)

- Complexity for N nodes is N^2.
- **How to improve efficiency:**
    - Connect only to the ***k* nearest neighbours**
      $\Rightarrow$ *N log N*
    - **Variable radius**: decrease the radius R as a function of N. How?



α: sampling sequence

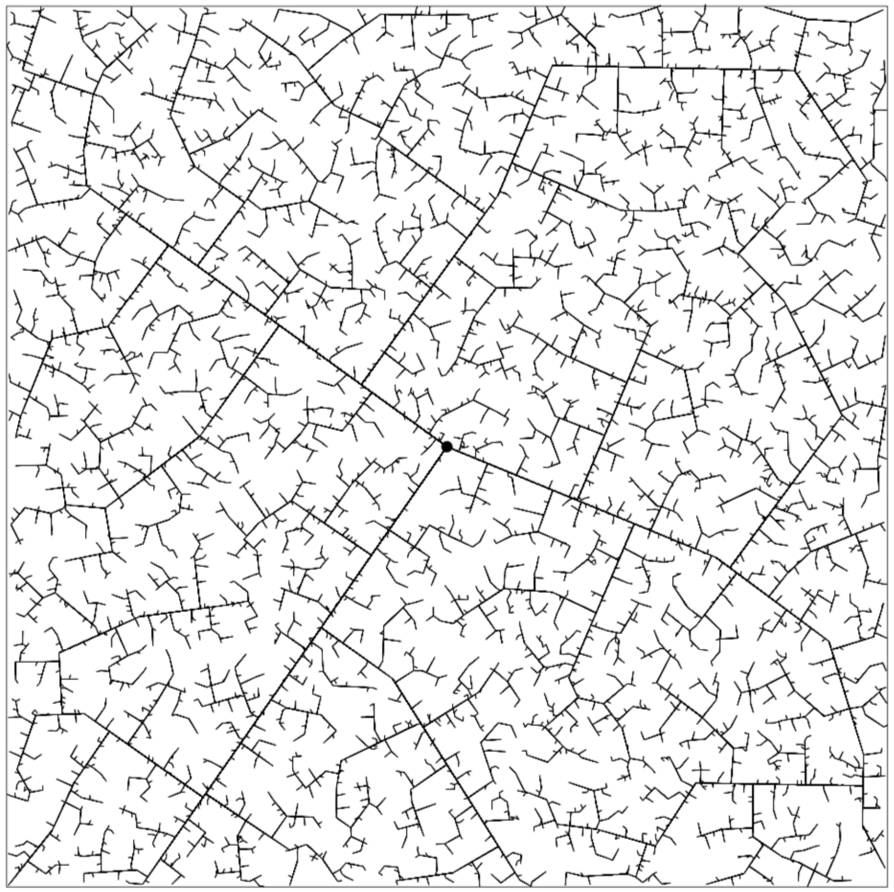
Tree-based search

- **Idea:** to make the search more efficient,
  we build a tree anchored at the starting node.

- Stop when you find a path to the goal.

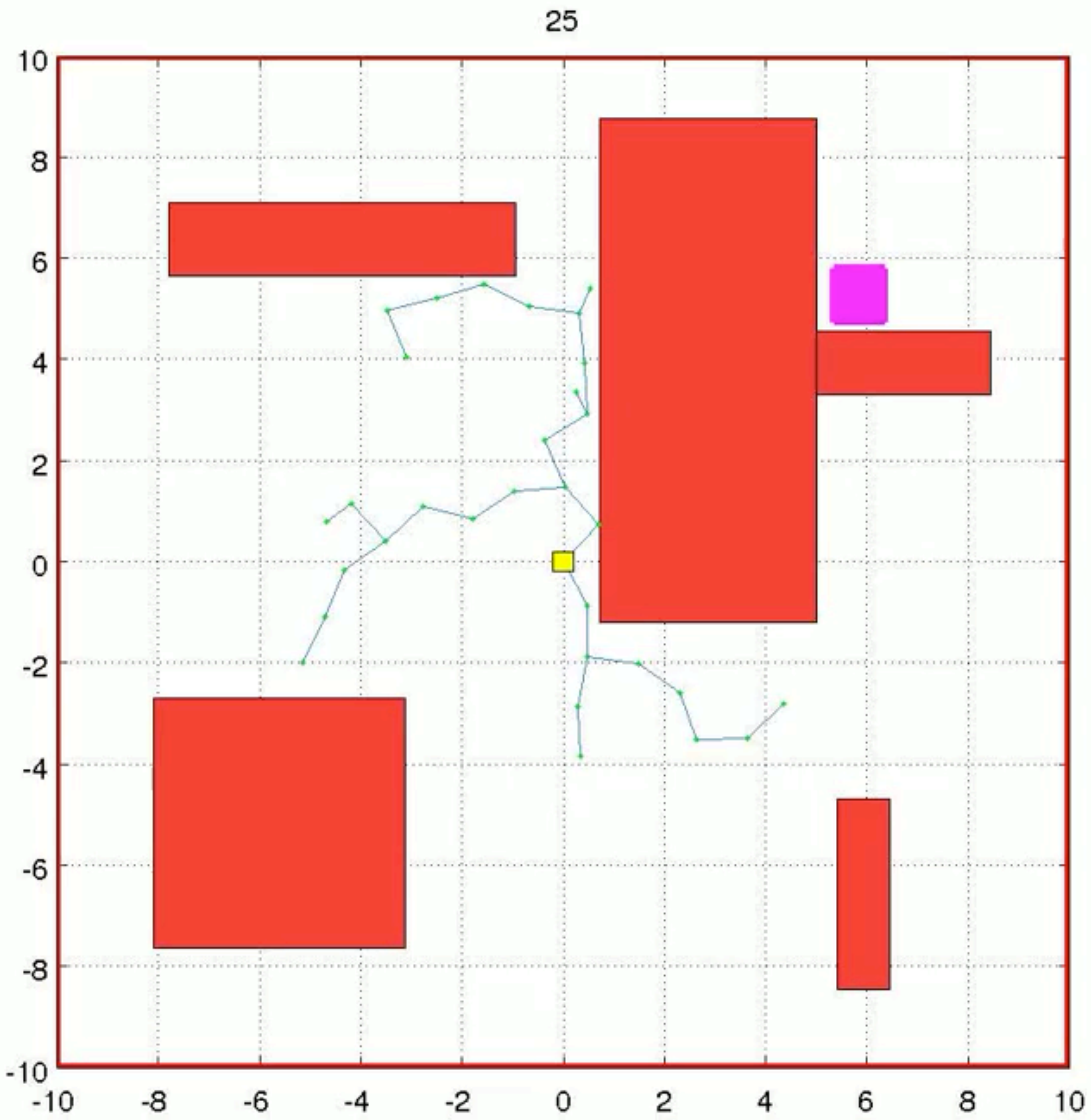
- Need to **explore rapidly** but also be **dense**.

45 iterations



2345 iterations

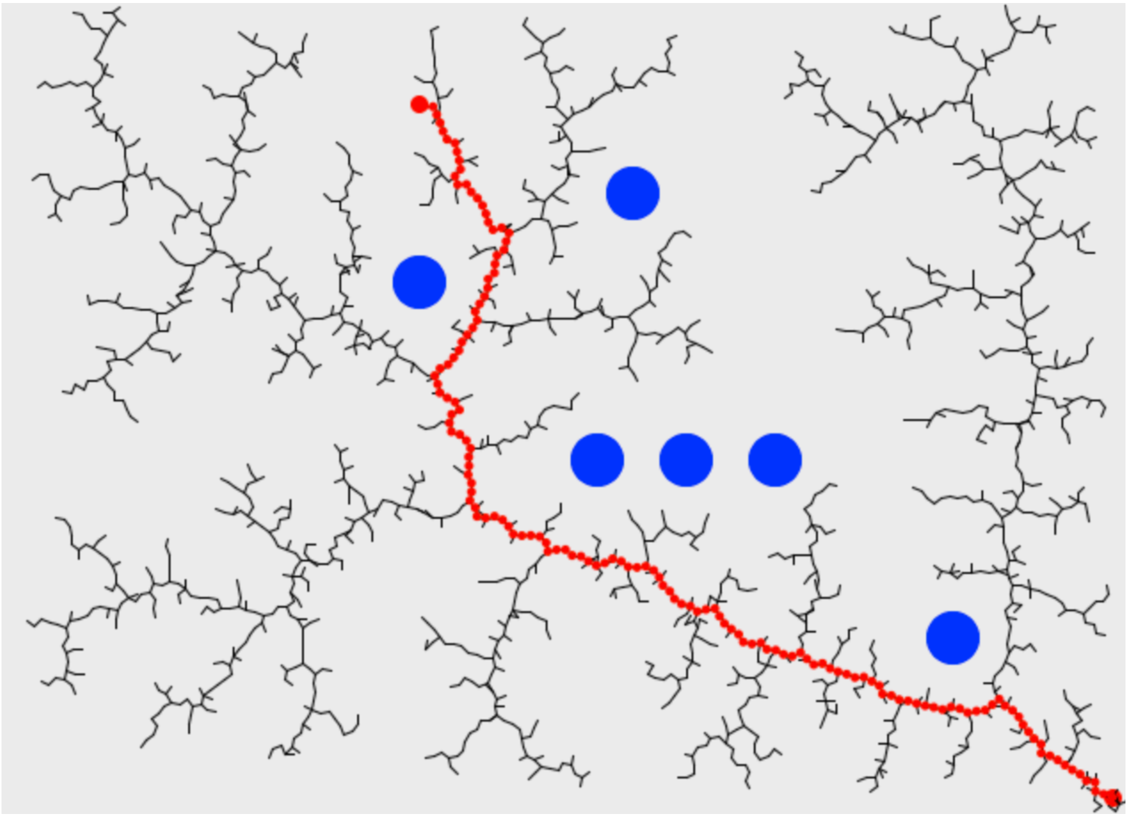
Rapidly-exploring random trees (RRT)

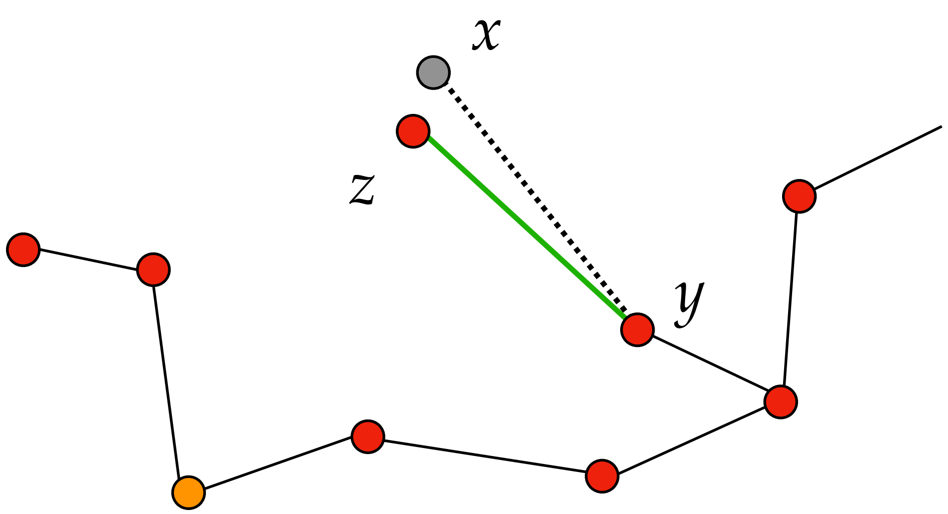


Picture credit: Karaman

Rapidly-exploring random trees (RRT)

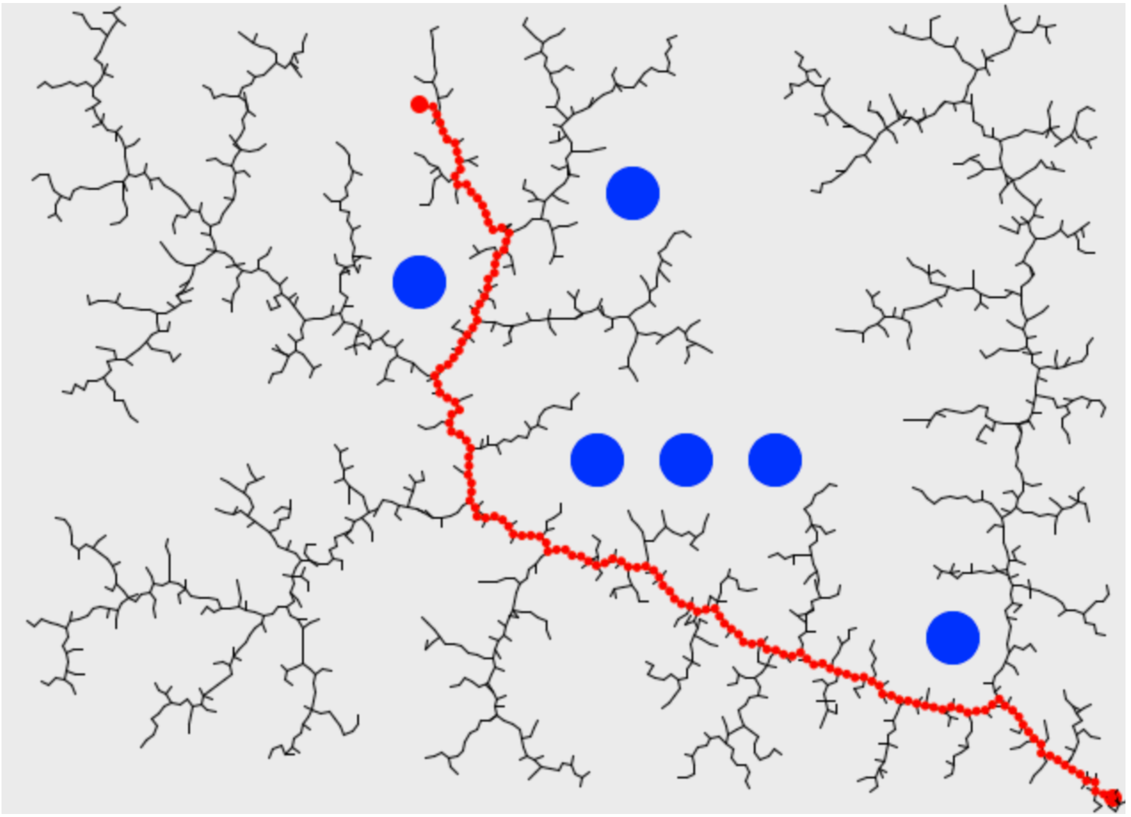
- Lavalle, Kuffner 2001
- Start with a node at
  the start configuration.
- Iterate N times:
  - Sample either a random point x
    or the goal with probability p~10%.
  - Find the closest node y.
  - Find a point z that is close to y
    that you can connect from x.
    - No "perfect" steering needed.
  - Consider adding the edge z-y.
    - Check for collisions.
- Stop when you find
  a path to the goal region.

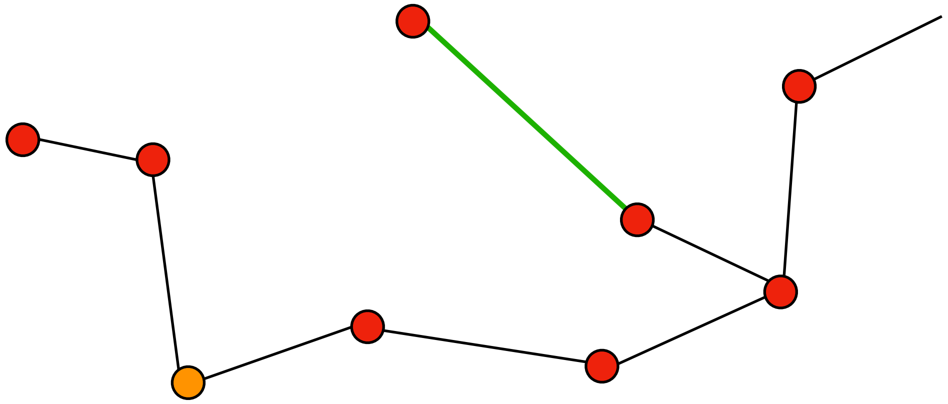




Rapidly-exploring random trees (RRT)

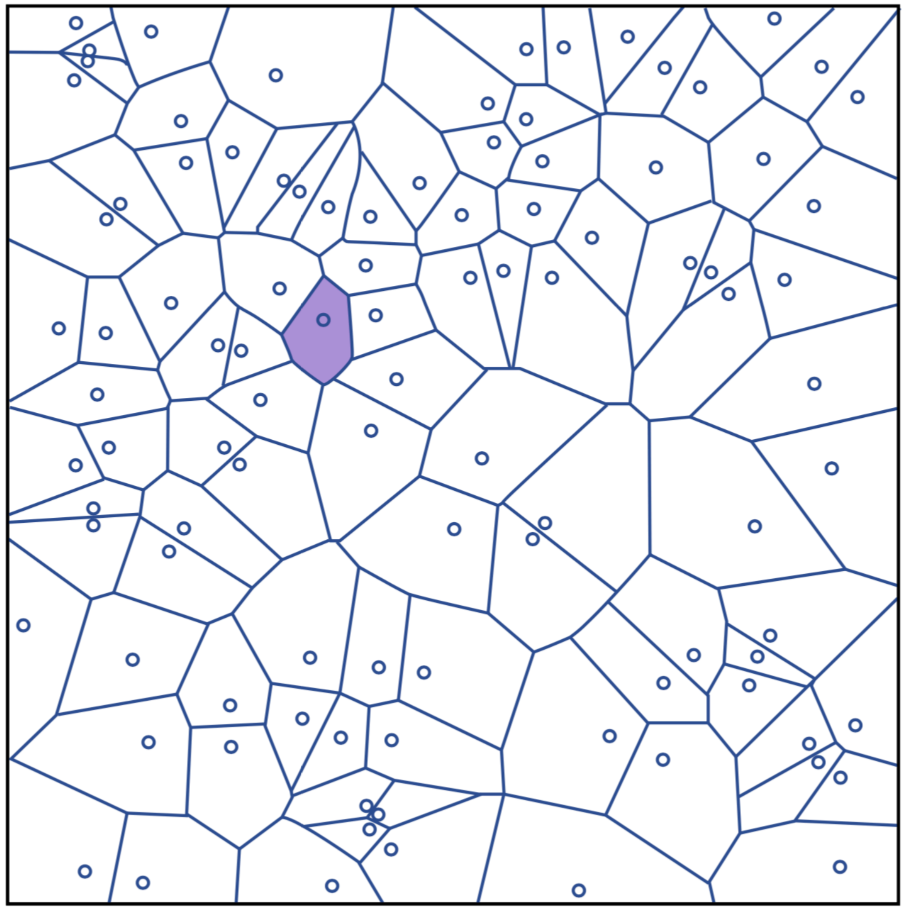
- Good only for **single query.**
- **Probabilistically complete.**
- Very fast compared to PRMs.
- **<u>Not</u> asymptotically optimal.**

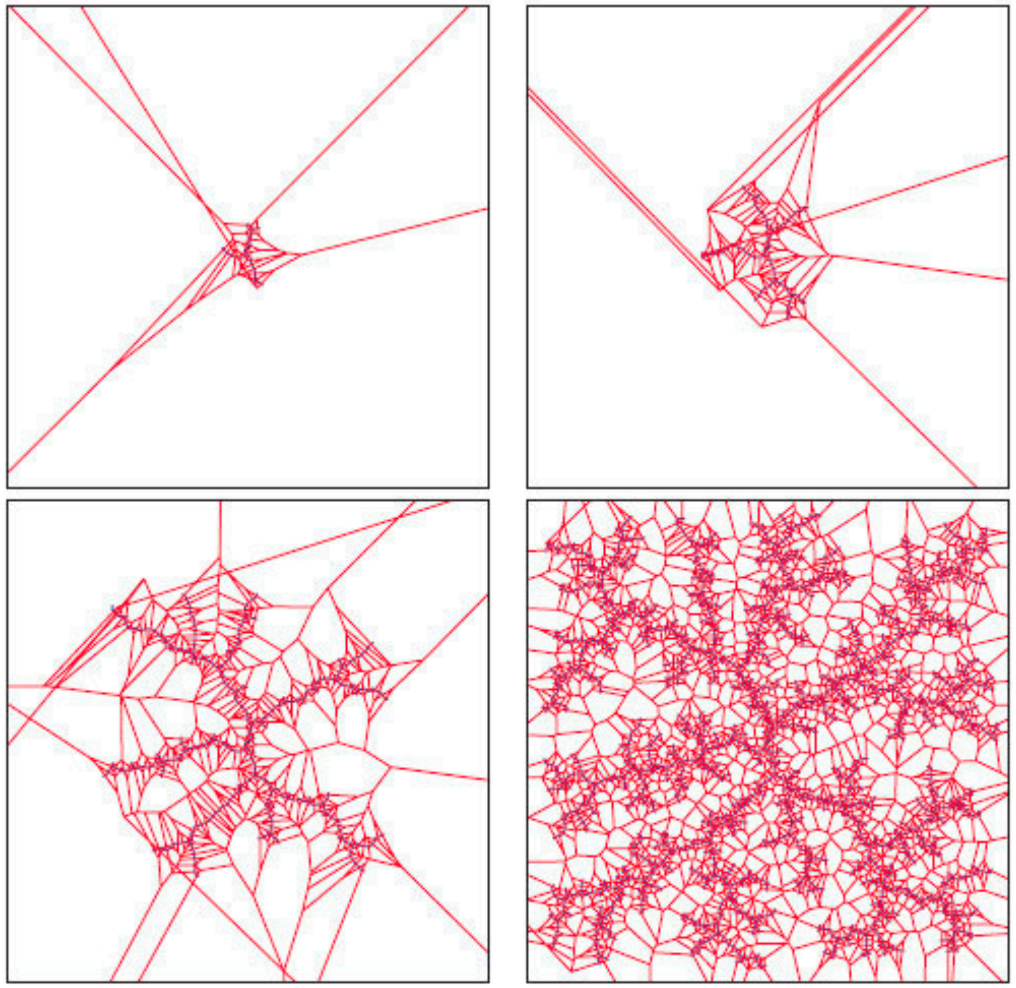




Why RRTs are fast: Voronoi Bias

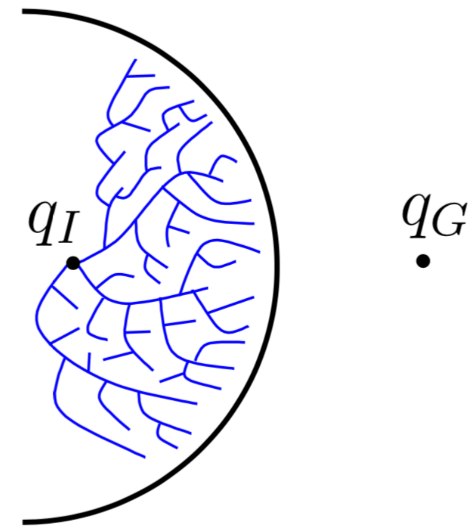
- **RRTs explore rapidly** because of the **"Voronoi bias"**.
- Nodes that are more "isolated" at the edges of unexplored areas have larger Voronoi regions and therefore more likely to be selected.

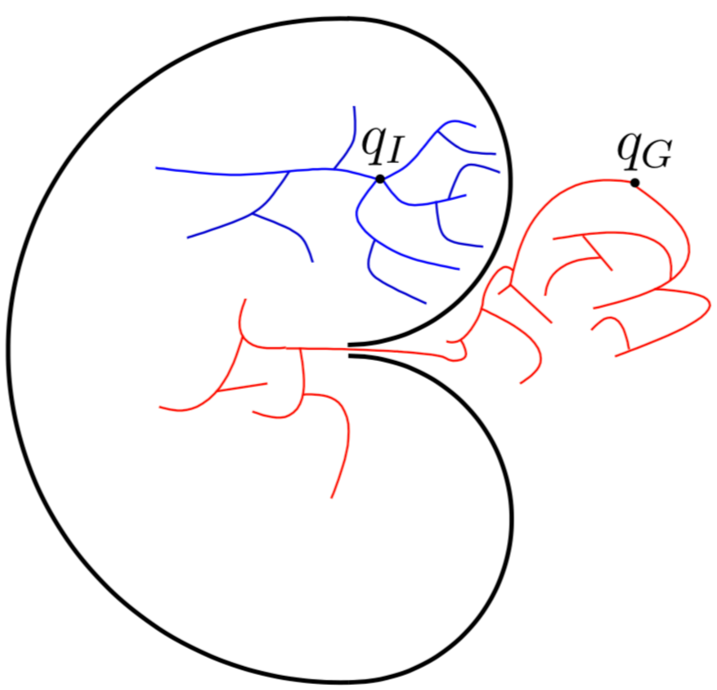




Two trees

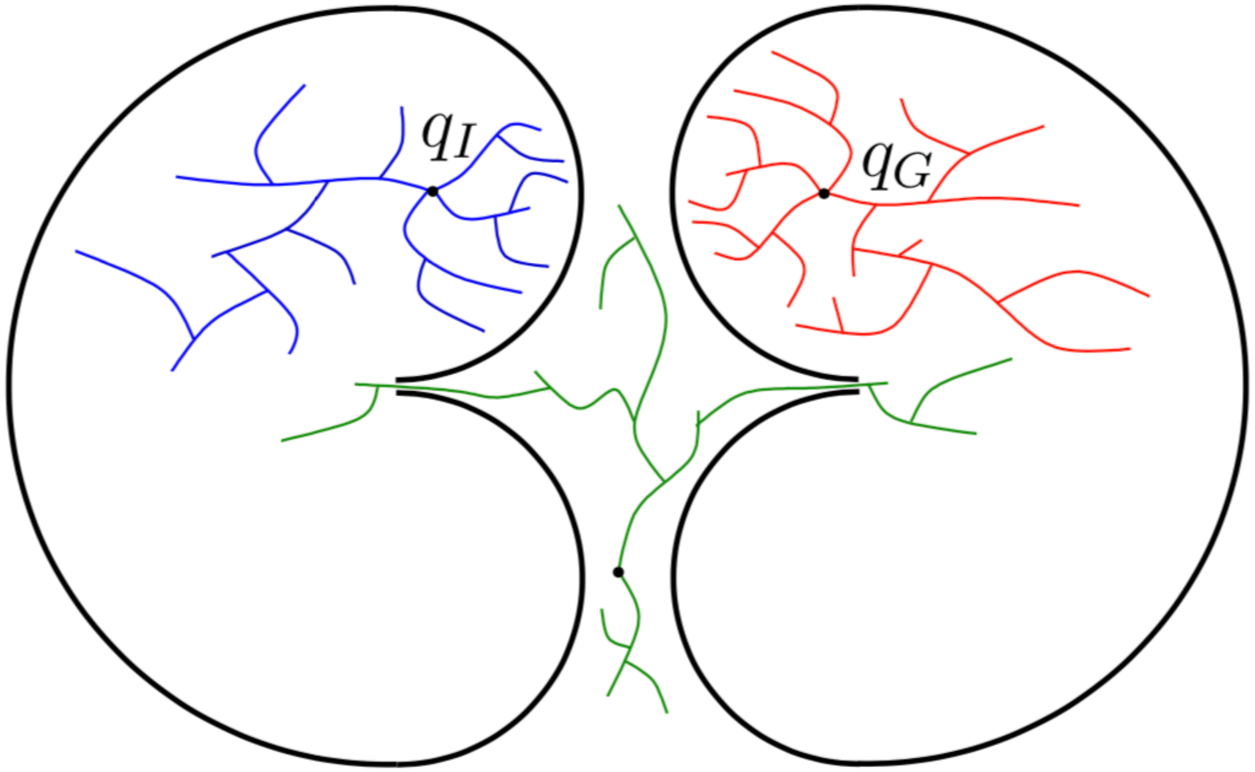
- Idea: **we grow 2 trees:**
  - **one from the start**
  - **one from the goal**
    with the inverse dynamics
- When the trees "touch"
  we have found a solution.





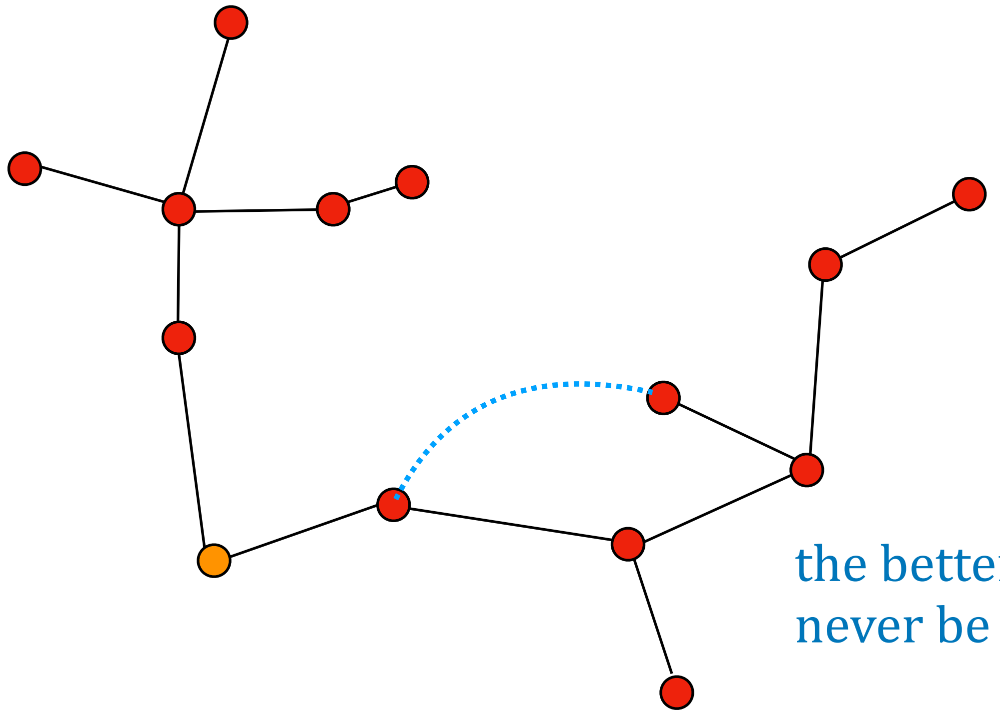
Three trees?

- In some cases it might be helpful to have more trees.
- Problem: the “fly traps”



Why RRTs are not optimal

- This goes **against intuition**: if we keep growing the graph shouldn't we sample all trajectories in the end?
- No: the previous samples bias the next samples.
- Note: **Once a path between two nodes has been found there will be no other path considered.**
- Hence: to achieve optimality you **need to rewire the graph**.



the better blue path will
never be considered

RRT* "RRT star"

- Karaman, Frazzoli 2010
- There are three improvements over RRT that together make the algorithm optimal:

  1. **Shrinking radius** for finding neighbours in a principled way.

     $$r = \gamma \sqrt[d]{(\log n)/n}$$

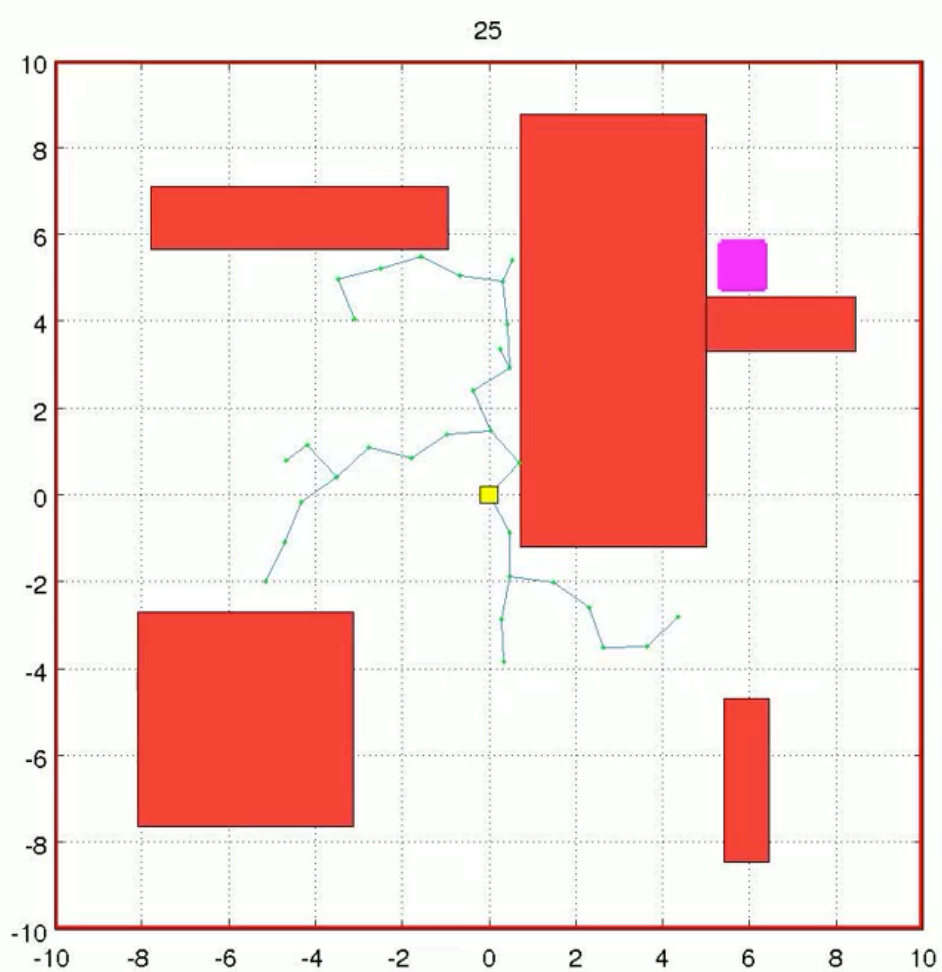
     n: iteration
     d: dimensionality
     γ: environment

  2. Connect to the point that has the best overall path cost, not the closest.
  3. After adding a point, **the tree is "rewired"** so that all paths are optimal.

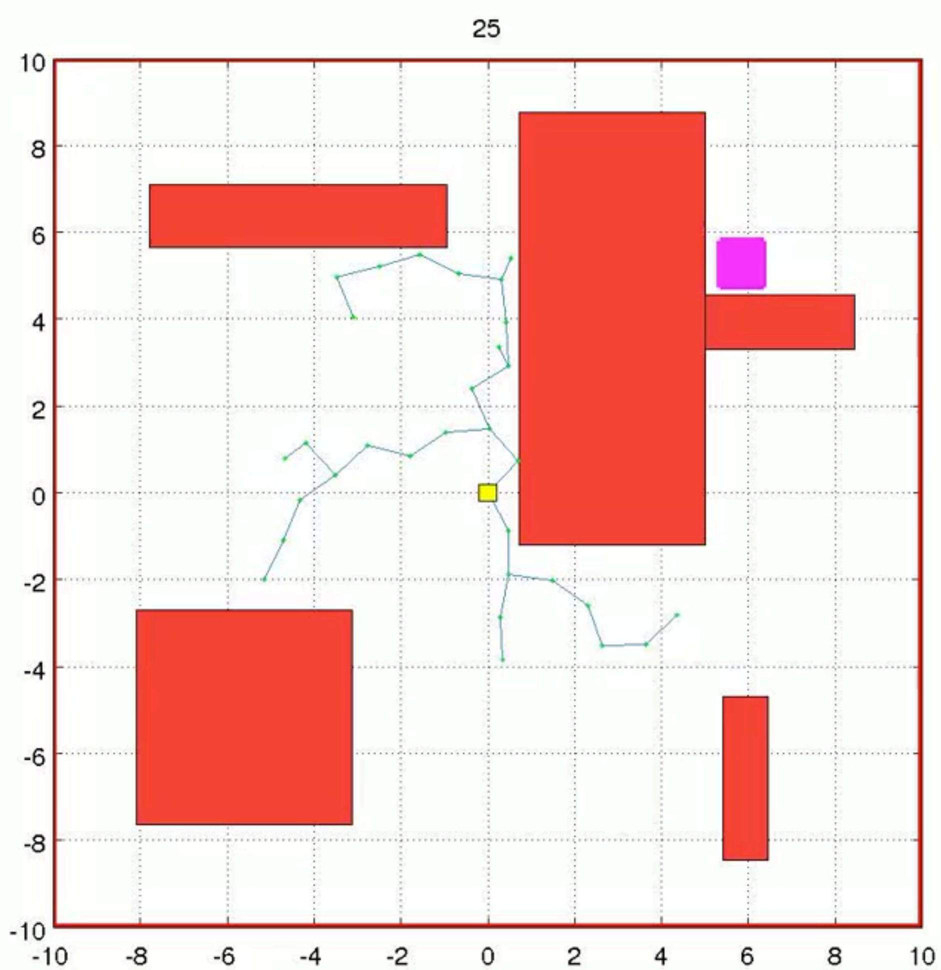
- Very technical proof for (1) hard to understand. The effect of (2)-(3) is more intuitive to see.

The re-wiring process in RRT*

RRT

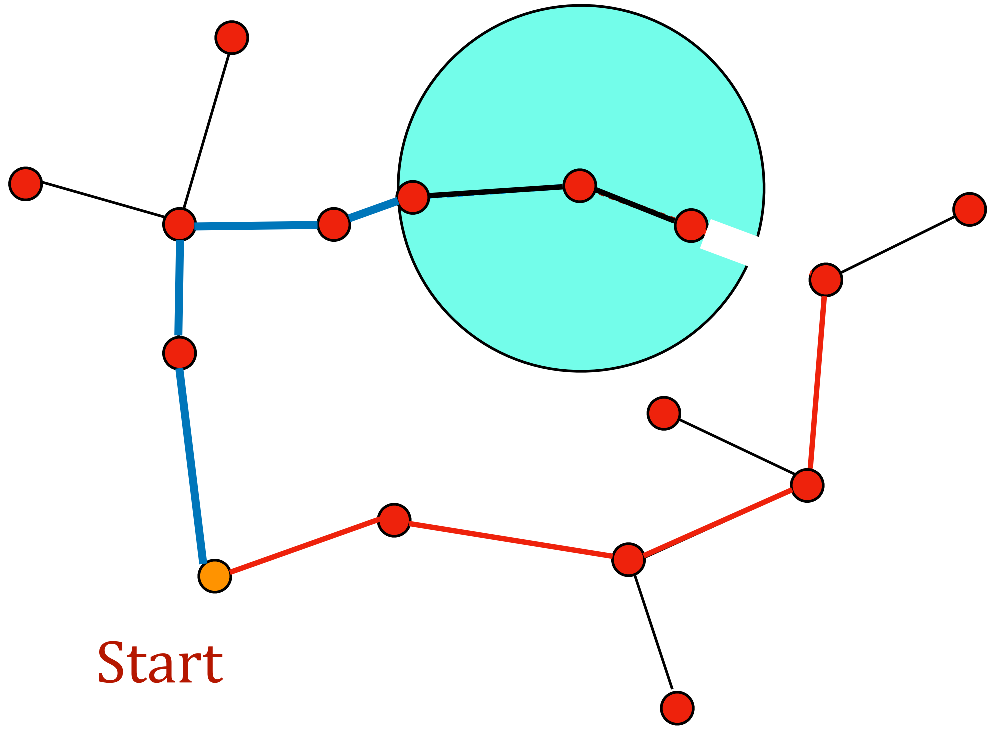


RRT*



The re-wiring process in RRT*

(Assuming for simplicity perfect steering.)



1. Sample new point

2. Look for neighbors in a radius *R* (adaptively changed with *N*)

3. Consider the paths to that point

4. Connect only to the point with the best overall path. (not the closest point)

5. For the other candidates, consider if it would be better for them to connect to the new point instead of the previous parent.

- Note how the rewiring improves the cost-to-come for the other vertices.

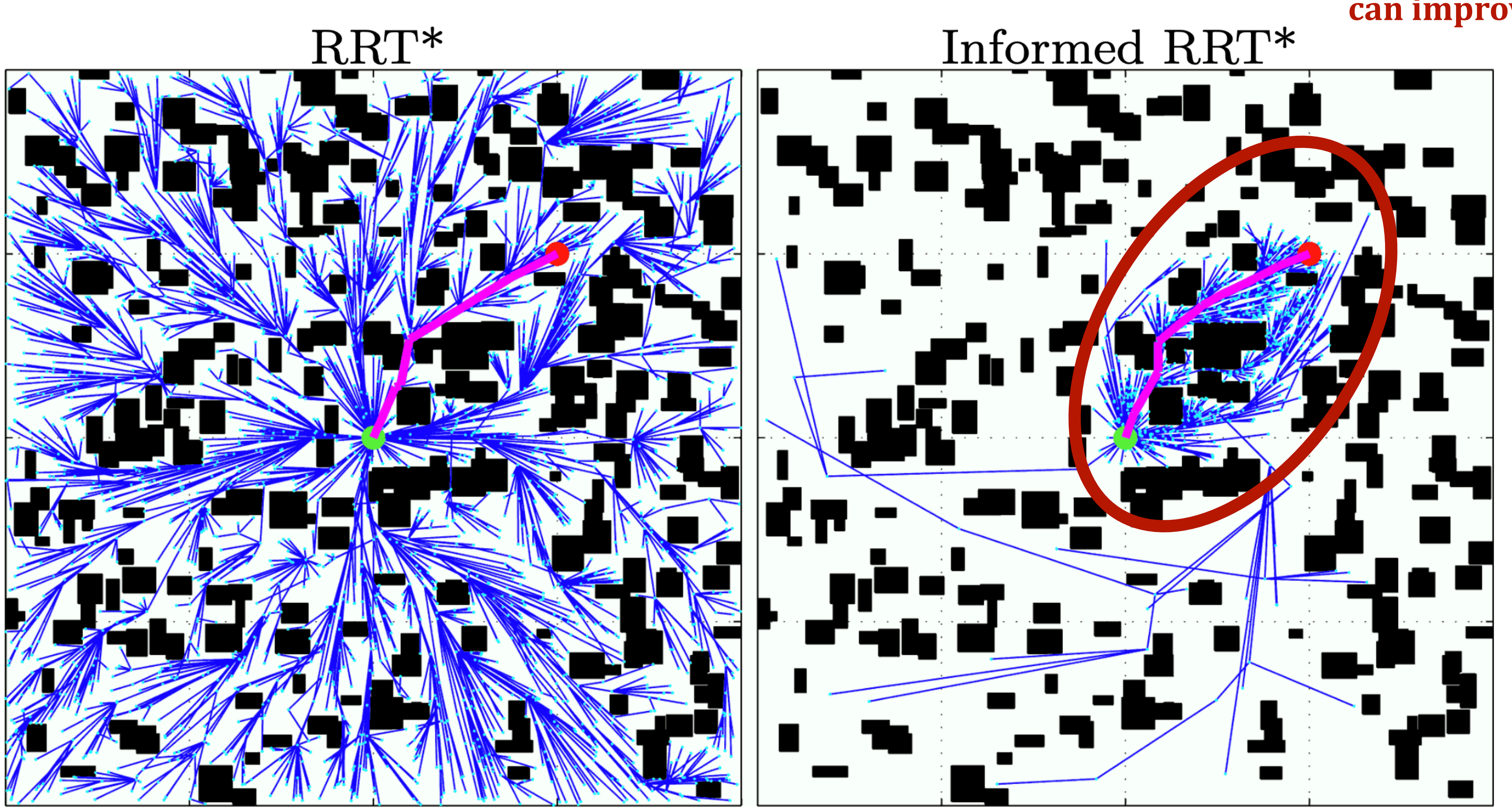
Extensions to RRT*

- There are **many more variations** one can formulate of RRT.
- Search "RRT* algorithm" on Youtube for many pretty videos!
- Extensions:
    - **biasing the sampling according to environment geoemtry**
    - **dynamic environments** (repair paths that become unfeasible)
    - better use of **additional heuristics**
        - Example: Informed RRT* (CMU) - next slides

Informed RRT*

- Example: Informed RRT* (CMU)
  https://www.youtube.com/watch?v=nsl-5MZfwu4

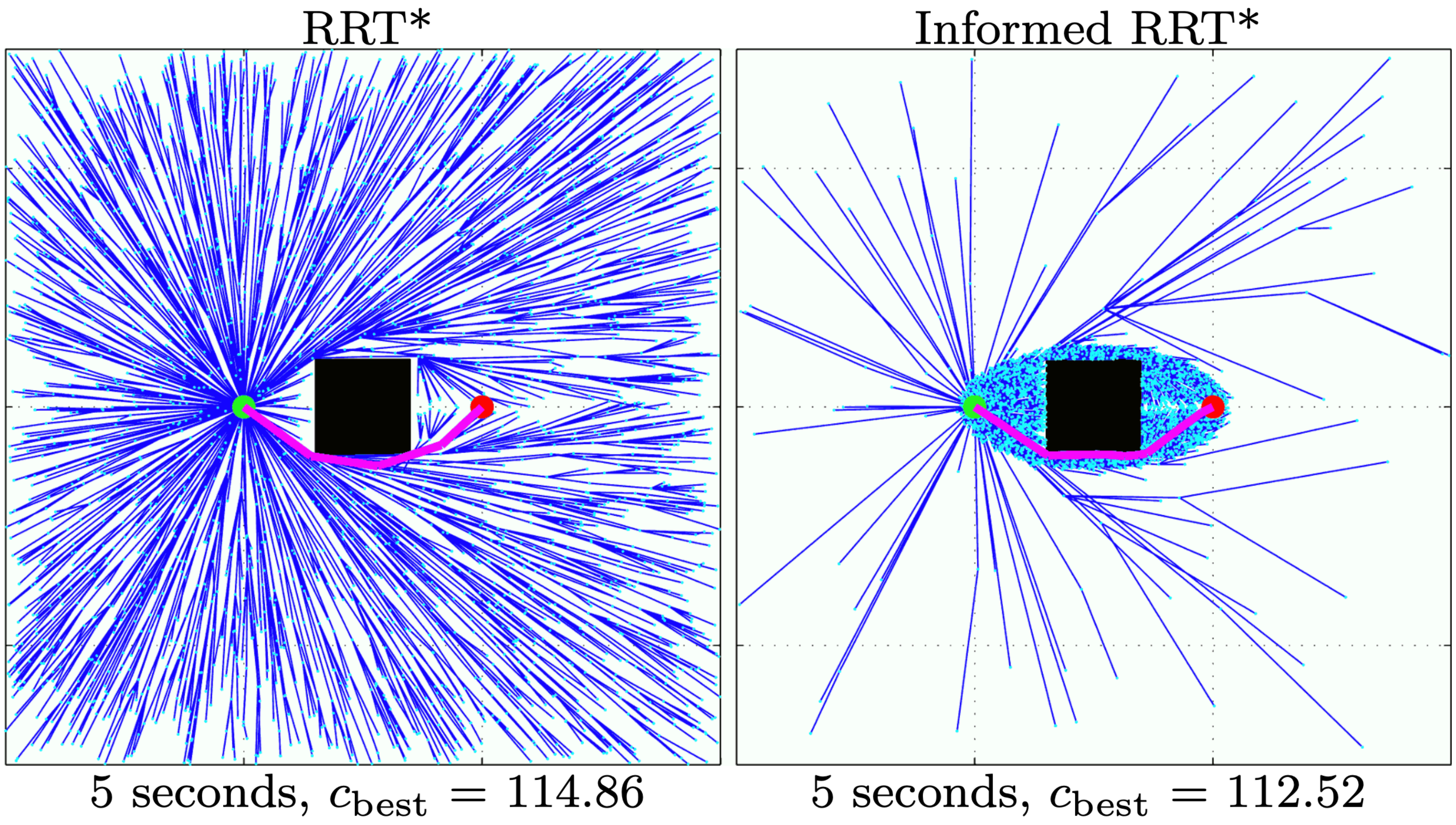
Focus on points that can improve the solution



8.26 seconds, $c_{\text{best}} = 0.76$ 1 second, $c_{\text{best}} = 0.76$

Informed RRT*

- Example: Informed RRT* (CMU)
  https://www.youtube.com/watch?v=nsl-5MZfwu4



Conclusions for PRM, RRT, RRT*

- Sampling-based search methods are attractive because:
  - **Easy to implement:** new collision checking and steering functions can be plugged in for new environments and dynamics.
  - **Robust:** can work with conservative collision checking and steering
  - **Scale well with dimensions**
  - **Very cute animations**
- **Cons:**
  - **It is very hard** to make it really fast (random memory access)
    - Not obvious how to parallelize

Conclusions on motion planning

- General guidance:
  - For **long-horizon, complex** geometric planning:
    - **Single query:** use Informed RRT*
    - **Multiple queries:** PRMs or their * variant (not covered in slides)
  - For **short-horizon, low-latency** decisions: try motion primitives
- Note: You still need to do graph planning as part of these methods.
- Also: You still want to refine the path using a local optimization method.

Are we done with planning?

- No, **motion planning is only the simple part** of the overall planning problem.
- In fact, we did not consider:
  - Optimization criteria other than minimal time
  - Uncertainty in state evolution
  - Uncertainty in sensing
  - Modeling errors
  - Other agents that might be adversarial (game theory)
  - Anytime planning: what to do if you have only limited computation.

	motion planning	**in general**
solution is	**a nominal path**	**a feedback strategy from the information state of the agent playing a game with other agents**

- There is no general approach for the complete problem that is computationally tractable.