Random sample-based path finding

Václav Hlaváč

Czech Technical University in Prague

Czech Institute of Informatics, Robotics and Cybernetics

166 36 Prague 6, Jugoslávských partyzánů 1580/3, Czech Republic

http://people.ciirc.cvut.cz/hlavac, vaclav.hlavac@cvut.cz

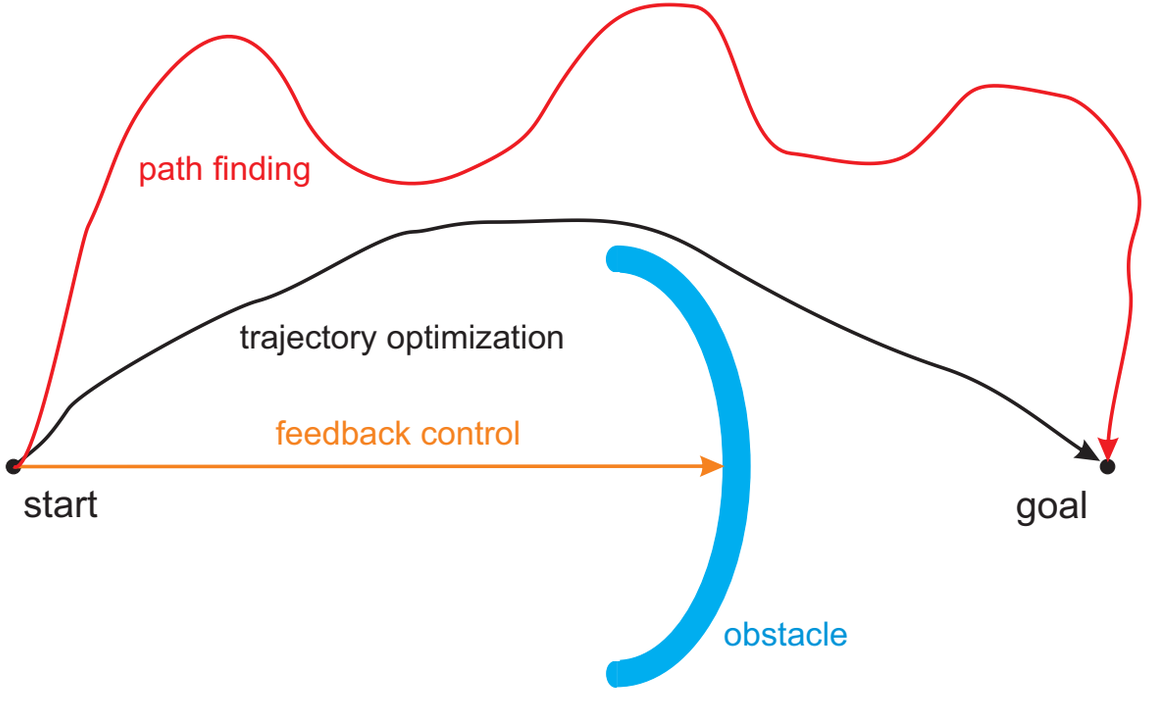
also Center for Machine Perception, http://cmp.felk.cvut.cz

Courtesy: Peter Abbeel, Hovie Choset, Jan Faigl, Daniela Rus, Marc Toussaint.

Outline of the talk:

- Random sample-based path finding.
- Probabilistic roadmap method (PRM).
- PRM and narrow passages.
- Rapidly exploring random trees (RRT).
-
-

Reminder: feedback control, (local) trajectory generation, path planning



- Feedback control solves many simpler navigation tasks, though not optimally in general. It does not cope with more complicated situations as, e.g. the obstacle avoidance.
- Trajectory optimization provides the locally optimal solution for more complicated tasks.
- Path finding, a global search, is needed for most complicated tasks. Here, we are satisfied with a valid solution. The reason is that the path finding algorithms have often exponential computational complexity (both in time and memory). Finding the optimal path would be even more complex.

Lecture plan

Pravious lecture: Methods based on $\mathcal{C}$-space discretization, graph traversal and some of them improved by heuristics.

- Path finding as a graph search.
- Bug algorithm.
- Adding potentials to guide the feedback control while avoiding obstacles.

Today: Random sample-based path finding.

- Probabilistic roadmaps (PRMs).
- Rapidly exploring random trees (RRTs).

Sample-based algorithm, core ideas

- Core ideas are shared with more general Monte Carlo algorithms. They are randomized, heuristic algorithms whose output may be incorrect with a certain (typically small) probability.

- Key idea: Rather than exhaustively explore all possibilities, explore a smaller subset of possibilities randomly while keeping track of the progress.
- Facilities "probing" deeper in a search graph much earlier than any exhaustive algorithm can.
- The caveat: We must sacrifice typically both completeness and optimality Classic tradeoff between solution quality and runtime performance.

We will remind Dijkstra algorithm finding the shortest path in the graph first.

Next, we will explain the probabilistic road map method.

Sample-based path finding algorithms; pros, cons

Pros

- Probabilistically complete.
- Do not construct the $\mathcal{C}$-space.
- Apply easily to high-dimensional $\mathcal{C}$-spaces.
- Support fast queries if enough preprocessing was done.
- Many success stories solving previously unsolved tasks.

Cons

- Requires solving 2-point boundary value problem.
- Do not work well for some tasks.
  - Unlikely to place sample nodes in narrow passages.
  - It is hard to sample/connect nodes on constraint surfaces.
- Neither optimality nor completeness.

Single-query vs. multiple-query motion planning

Single-query motion planning (batch) methods compute only a single path between a start configuration and a goal configuration.

They gain efficiency by developing an understanding of the configuration space connectivity that pertains to this particular query.

Example: Probabilistic Roadmap [L. E. Kavraki 1996].

Multiple-query motion planning (incremental) methods approximate the connectivity of the entire configuration space. While this is computationally more expensive, the resulting roadmap can be used to quickly compute a path between any two configurations.

Multi-query methods are well suited to static environments where the expensive precomputation is worthwhile.

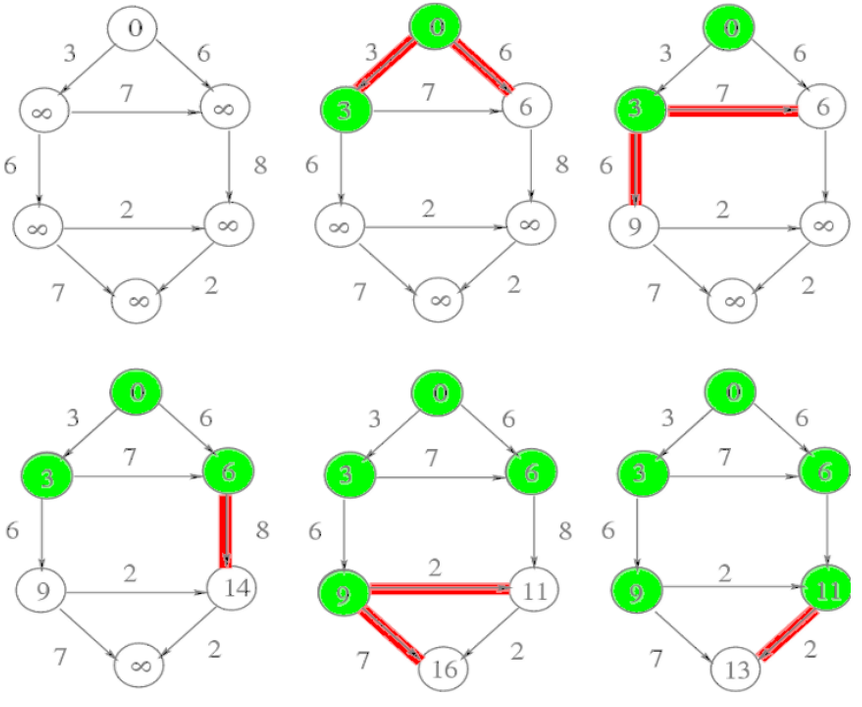
Example: Rapidly-exploring random tree [S. M. LaValle 1998].

Dijkstra algorithm for the shortest path in general graphs

- s – source node
- $d(j)$ – the minimal distance from node s to node j
- $pred(j)$ – predecessor of the node j

Dijkstra algorithm (1956) finds the shortest node from the source node s to all nodes.

% N – set of all graph nodes
$V := s$ % visited nodes;
$U := N \setminus s$ % unvisited nodes;
$d(s) := 0$, $i := s$;
while $|V|<|N|$ **do**
  $choose(i, j)$:
    $d(j) := \min_{k,m}\{d(k) + c_{km} \mid k \in V, m \in U\}$;
  $U = U \setminus \{j\}$;
  $V = V + \{j\}$;
  $pred(j) := i$;
end



- Incrementally labels nodes with their distance-from-start.
- Produces optimal (shortest) paths.
- Performance $\mathcal{O}(|N|^2)$ with the heap reshuffling $\mathcal{O}(|N| \log |N|)$.

Probabilistic roadmap (PRM) planner

- The probabilistic roadmap planner is a motion planning algorithm determining a path between a starting configuration of the robot and a goal configuration while avoiding collisions.
- The basic idea behind PRM is to take random samples from the configuration space of the robot, testing them for whether they are in the free space, and use a local planner to attempt to connect these configurations to other nearby configurations.
- The probabilistic roadmap planner consists of two phases: a construction and a query phase.
  - In the construction phase, a roadmap (graph) is built. A random configuration is created. It is connected to neighbors, e.g. k-nearest ones.
  - In the query phase, the start and goal configurations are connected to the graph, and the path is obtained by a Dijkstra's shortest path query.

PRM references

- Kavraki, L. E.; Švestka, P.; Latombe, J.-C.; Overmars, M. H. (1996), *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*, IEEE Transactions on Robotics and Automation, 12 (4): 566–580, doi:10.1109/70.508439.
- Geraerts, R.; Overmars, M. H. (2002), *A comparative study of probabilistic roadmap planners*, Proceedings of the Workshop on the Algorithmic Foundations of Robotics, pp. 43–57.

Probabilistic roadmap, concepts and notation

Notation

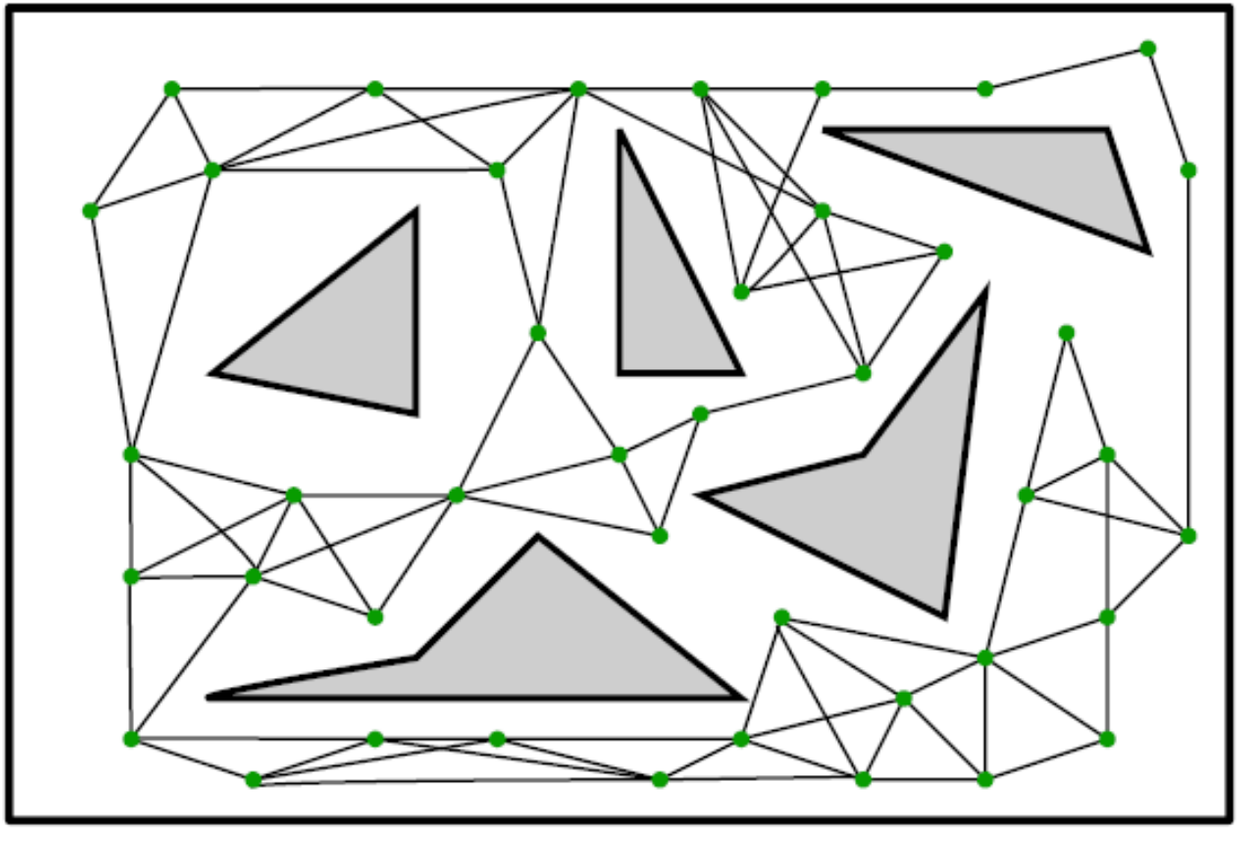
- $q \in \mathbb{R}$ is a configuration in $\mathcal{C}$-space.
- Q_{free} is set of free configurations, i.e. without collisions.

Required modules

- Method for Sampling points in $\mathcal{C}$-space.
- Method for validating points in $\mathcal{C}$-space, e.g. those points belonging to obstacles.

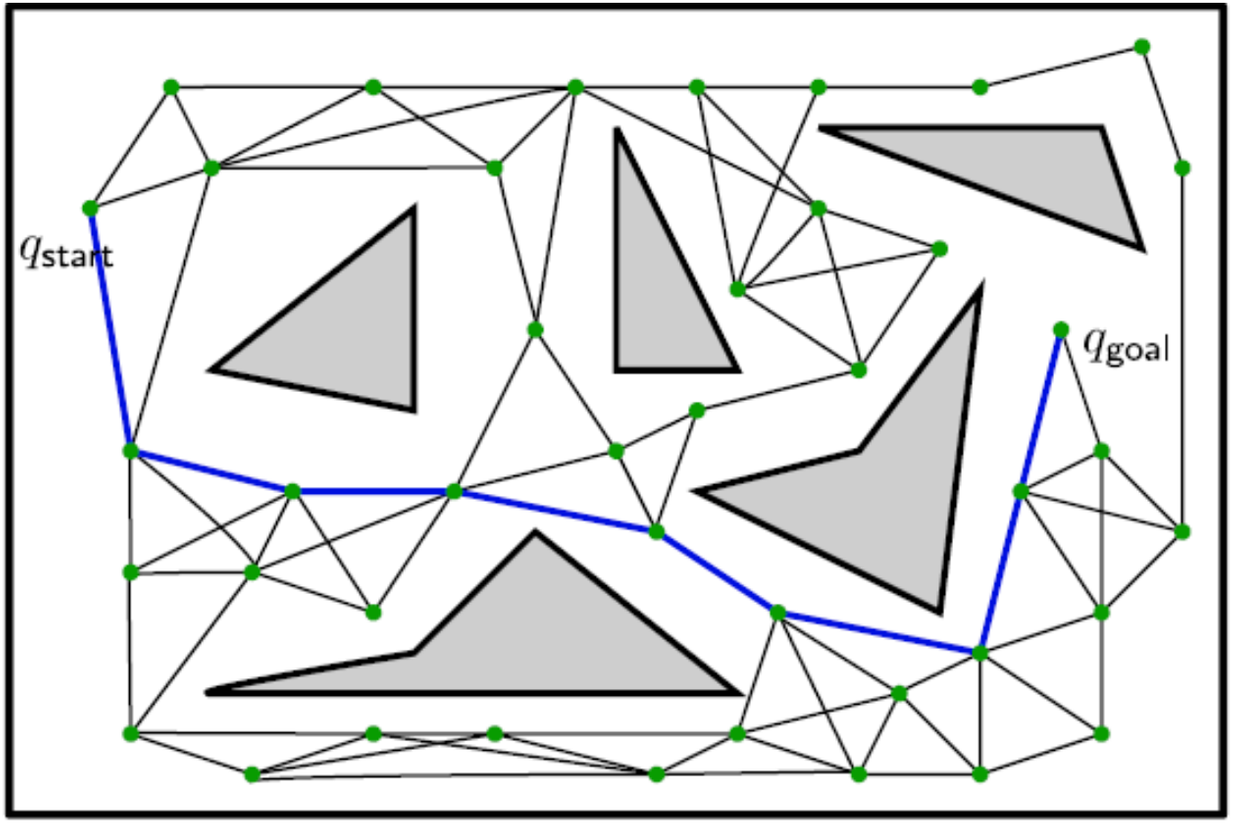
Probabilistic roadmap, graph generation

- Probabilistic road map generates a graph $G = (V, E)$ of configurations such that configurations along edges are $\in Q_{\text{free}}$.



Probabilistic roadmap, path finding

Given the graph, use (e.g.) Dijkstra algorithm to find the path from q_{start} to q_{goal}.



Probabilistic roadmap, the algorithm

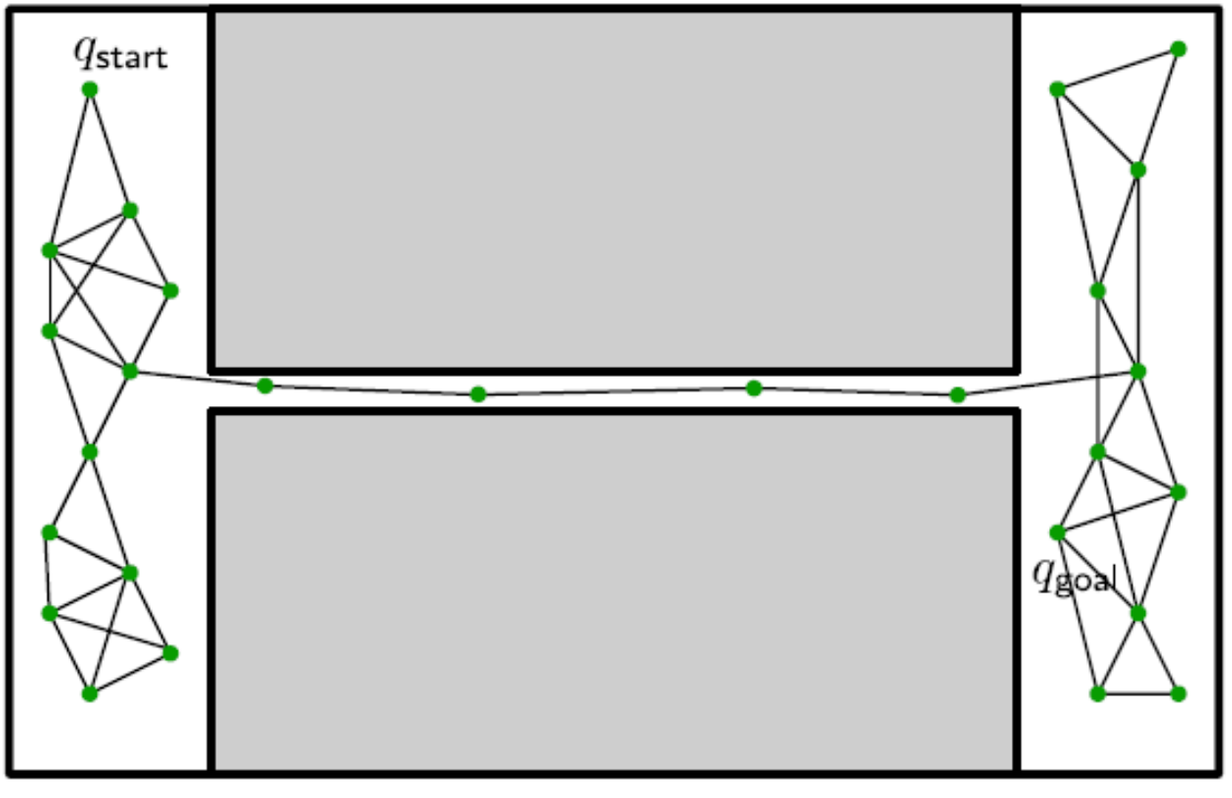
Input: number n of samples, number k number of nearest neighbors
Output: PRM $G = (V, E)$

1: initialize $V = \emptyset$, $E = \emptyset$
2: **while** $|V| < n$ **do** *// find n collision free points q_i*
3: $q \leftarrow$ random sample from Q
4: **if** $q \in Q_{\text{free}}$ **then** $V \leftarrow V \cup \{q\}$
5: **end while**
6: **for all** $q \in V$ **do** *// check if near points can be connected*
7: $N_q \leftarrow k$ nearest neighbors of q in V
8: **for all** $q' \in N_q$ **do**
9: **if** path$(q, q') \in Q_{\text{free}}$ **then** $E \leftarrow E \cup \{(q, q')\}$
10: **end for**
11: **end for**

where path(q, q') is a local planner (easiest: straight line)

Probabilistic roadmap, problem: narrow passages

The smaller the gap (clearance δ) the more unlikely to sample such points.



Probabilistic roadmap; pros and cons

Pros

- Very simple algorithmically.
- Highly explorative.
- Allows probabilistic performance guarantees.
- Good to answer many queries in an unchanged environment

Cons

- Precomputation of an exhaustive roadmap takes a long time (but not necessary for "Lazy PRMs").
- Hard to sample/connect nodes on constraint surfaces.

Probabilistic roadmap; a little theory

for uniform sampling in d-dimensional space.

- Let $q_1, q_2 \in Q_{\text{free}}$ and γ be a path in Q_{free} connecting q_1 and q_2.
- δ is the clearance of the path γ, i.e. the distance to obstacles.
- $\sigma = \frac{|B_1|}{2^d |Q_{\text{free}}|}$, where B_1 is a hyperball of radius δ centered at q_1.
- The probability that the probabilistic roadmap (PRM) method finds the path after n samples is

$$P(\text{PRM-success}|n) \geq 1 - \frac{2|\gamma|}{\delta} e^{-\sigma \delta^d n}$$

- The result is probabilistic complete because one can achieve certainty with high enough n.
- For a given success probability, n needs to be exponential in d.

Probabilistic roadmap, challenges

1. Connecting neighboring points:

   It is easy for holonomic systems (i.e., for which you can move each degree of freedom at will at any time) only. It requires solving a Boundary Value Problem in general.

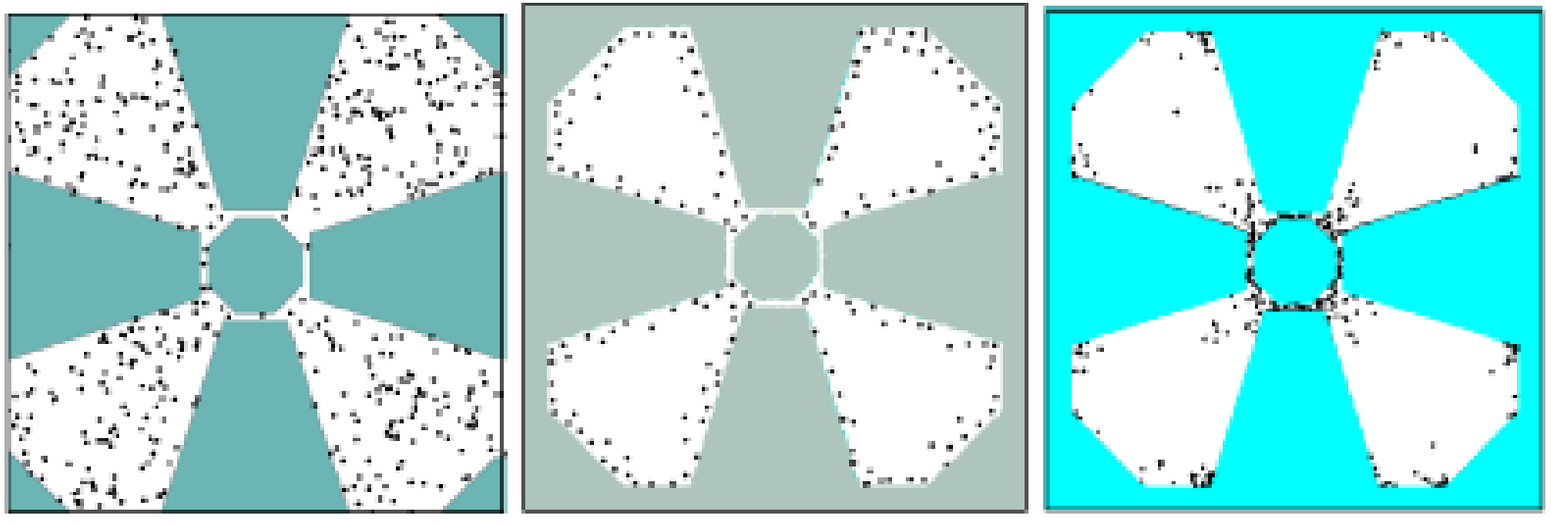
2. Collision checking:

   It often takes majority of time in applications.

3. Sampling:

   How to sample uniformly (or biased according to prior information) over configuration space?

Narrow passages; improved sampling strategies



Uniform sampling Gaussian sampling Bridge sampling

- Gaussian sampling:

  Generate q_1 randomly; $q_2 \sim (\mathcal{N}, \sigma)$; if $q_1 \in Q_{\text{free}}$ and $q_2 \notin Q_{\text{free}}$ then add q_1 (or vice versa).

- Bridge sampling:

  Generate q_1 randomly; $q_2 \sim (\mathcal{N}, \sigma)$; $q_3 = \frac{q_1+q_2}{2}$; if $q_1, q_2 \notin Q_{\text{free}}$ and $q_3 \in Q_{\text{free}}$ then add q_3.

Figure courtesy: O. Brock

Rapidly-exploring random trees (RRTs)

- A rapidly exploring random tree (RRT) is an algorithm designed to efficiently search nonconvex, high-dimensional spaces by randomly building a space-filling tree.
- The tree is constructed incrementally from samples drawn randomly from the search space and is inherently biased to grow towards large unsearched areas of the problem.
- Original publication: LaValle, Steven M. (October 1998). *Rapidly-exploring random trees: A new tool for path planning*. Technical Report. Computer Science Department, Iowa State University (TR 98-11).
- RRTs can be viewed as a Monte Carlo technique to generate open-loop trajectories for nonlinear systems with state constraints.
- A single-query algorithm. $\Rightarrow$ The trajectory is not optimal usually.

Courtesy: Wikipedia, Rapidly-exploring random tree

RRT method; informal description

- RRT grows a tree rooted at the starting configuration by using random samples from the search space.
- As each sample is drawn, a connection is attempted between it and the nearest state in the tree. If the connection is feasible (passes entirely through free space and obeys any constraints), this results in the addition of the new state to the tree.
- With uniform sampling of the search space, the probability of expanding an existing state is proportional to the size of its Voronoi region. As the largest Voronoi regions belong to the states on the frontier of the search, this means that the tree preferentially expands towards large unsearched areas.

Courtesy: Wikipedia, Rapidly-exploring random tree

RRT algorithm; the simplest

Simplest RRT with a straight line local planner and the step size α.

Input : q_{start}, number of nodes n, stepsize α
Output: Tree $T = (V, E)$

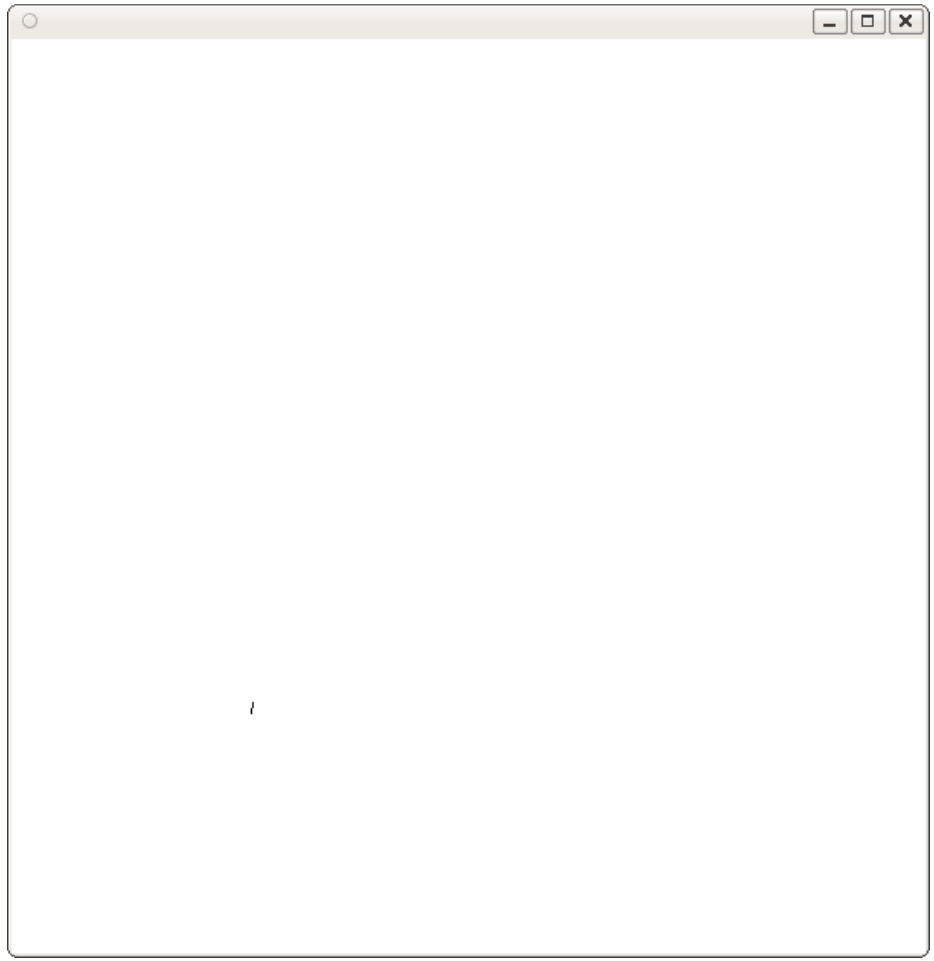
Initialize $V = q_{\text{start}}$, $E = \emptyset$. ;
for $i = 0$ **to** n **do**
  $q_{\text{taget}} \leftarrow$ random sample from Q ;
  $q_{\text{near}} \leftarrow$ nearest neighbor of q_{taget} in V ;
  $q_{\text{new}} \leftarrow q_{\text{near}} + \frac{\alpha}{|q_{\text{near}} - q_{\text{taget}}|}(q_{\text{near}} - q_{\text{taget}})$;
  if $q_{\text{new}} \in Q_{\text{free}}$ **then**
    $V \leftarrow V \cup \{q_{\text{new}}\}$;
    $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
  end
end

Courtesy: Marc Toussaint

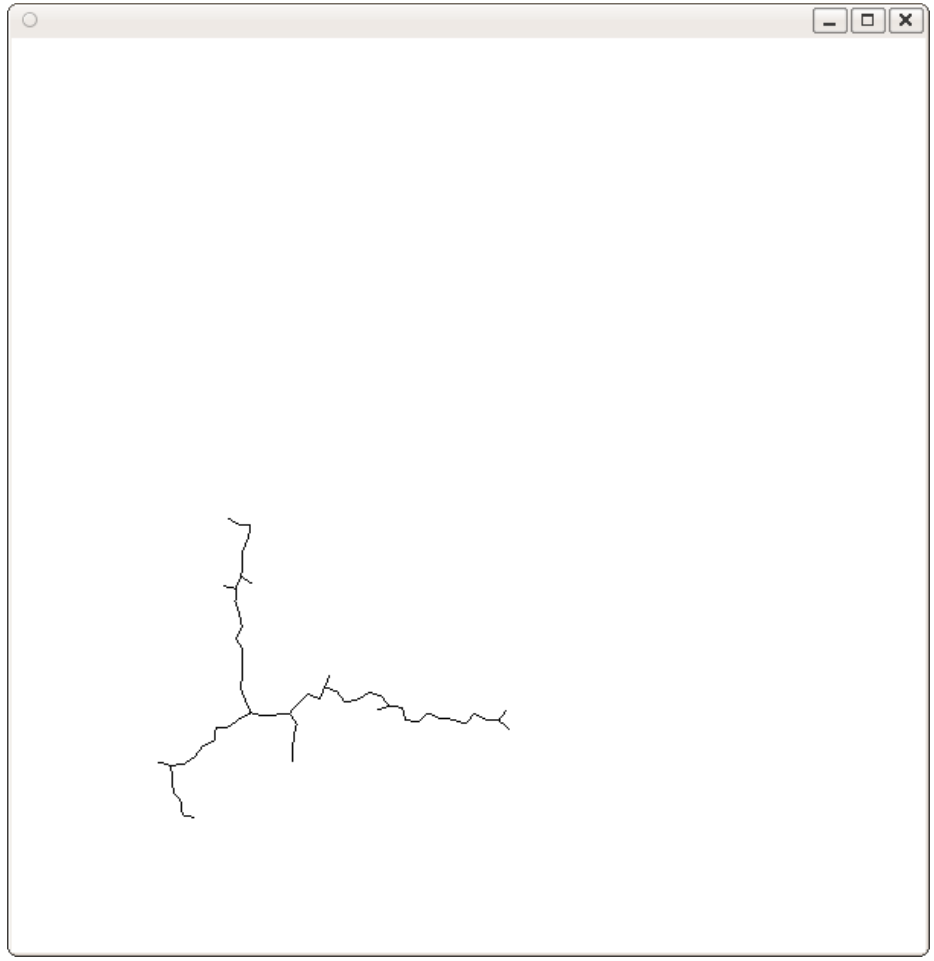
RRT algorithm; growing directly towards the goal

Input : q_{start}, q_{goal}, number of nodes n, stepsize α, β
Output: Tree $T = (V, E)$

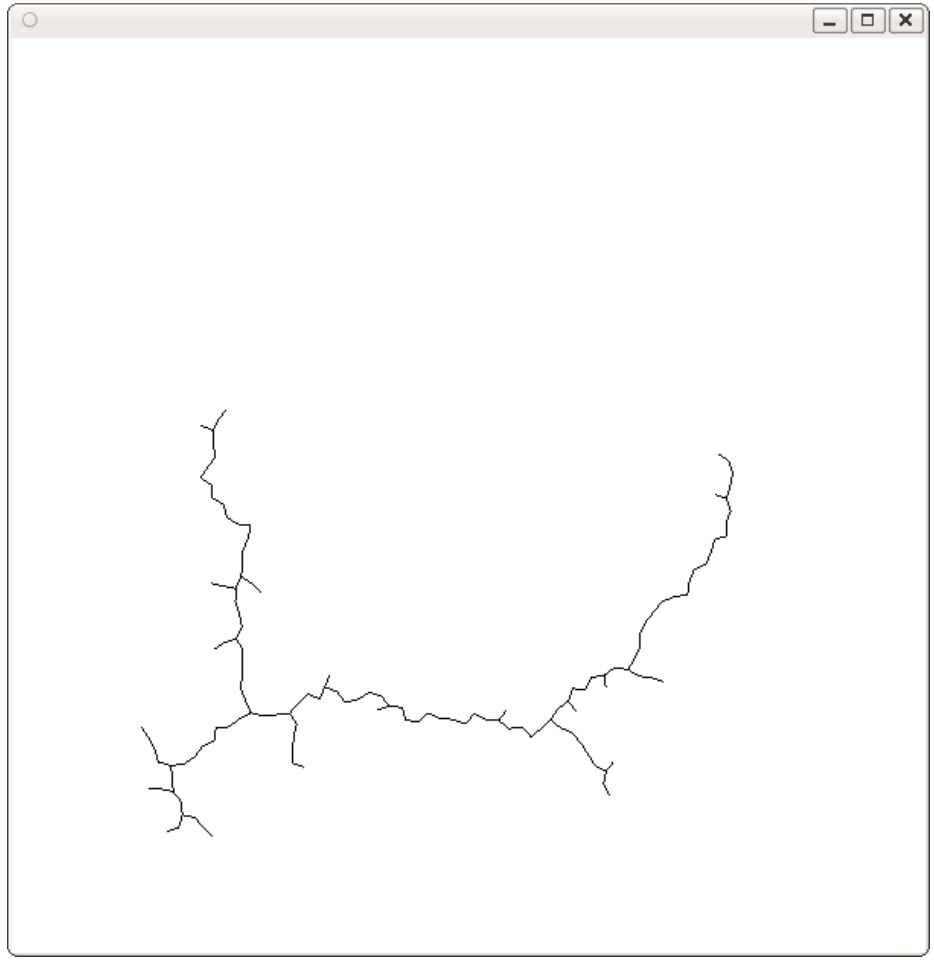
Initialize $V = q_{\text{start}}$, $E = \emptyset$. ;
for $i = 0$ **to** n **do**
  if *rand(0,1)* $< \beta$ **then** $q_{\text{taget}} \leftarrow q_{\text{goal}}$;
  else $q_{\text{taget}} \leftarrow$ random sample from Q;
  $q_{\text{taget}} \leftarrow$ random sample from Q ;
  $q_{\text{near}} \leftarrow$ nearest neighbor of q_{taget} in V ;
  $q_{\text{new}} \leftarrow q_{\text{near}} + \frac{\alpha}{|q_{\text{near}} - q_{\text{taget}}|}(q_{\text{near}} - q_{\text{taget}})$;
  if $q_{\text{new}} \in Q_{\text{free}}$ **then**
    $V \leftarrow V \cup \{q_{\text{new}}\}$;
    $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
  end
end



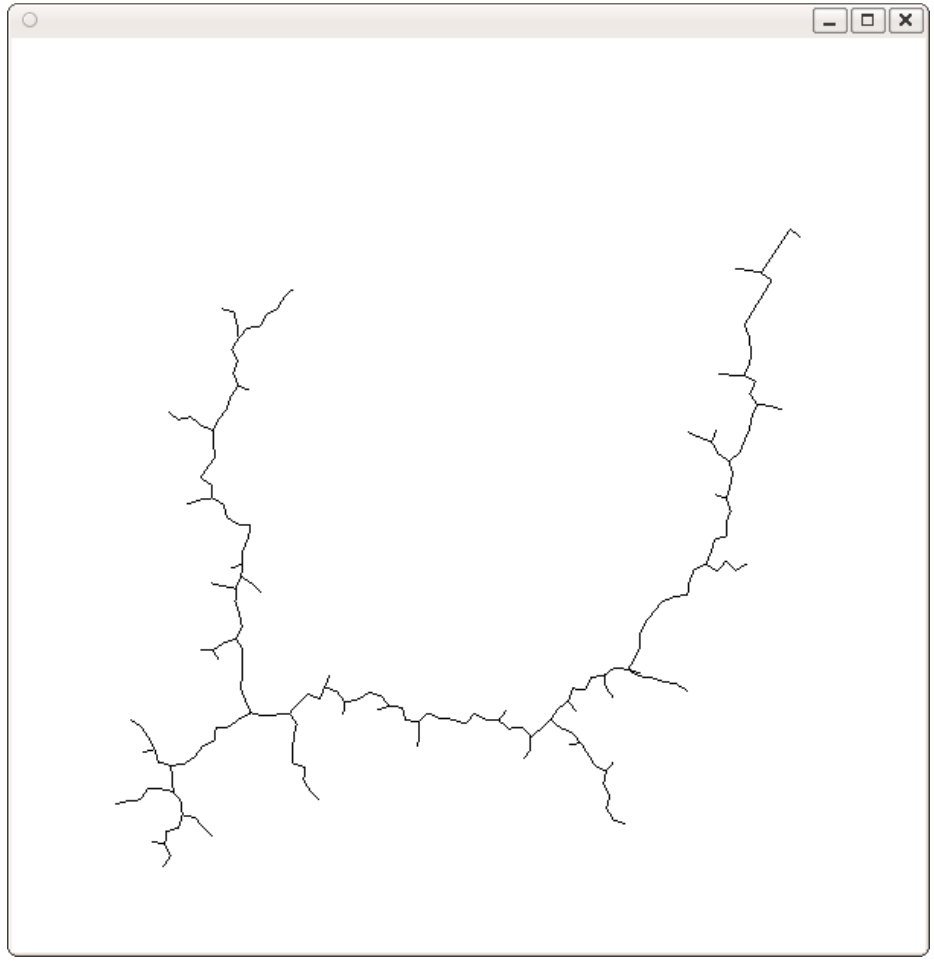
$n = 1$



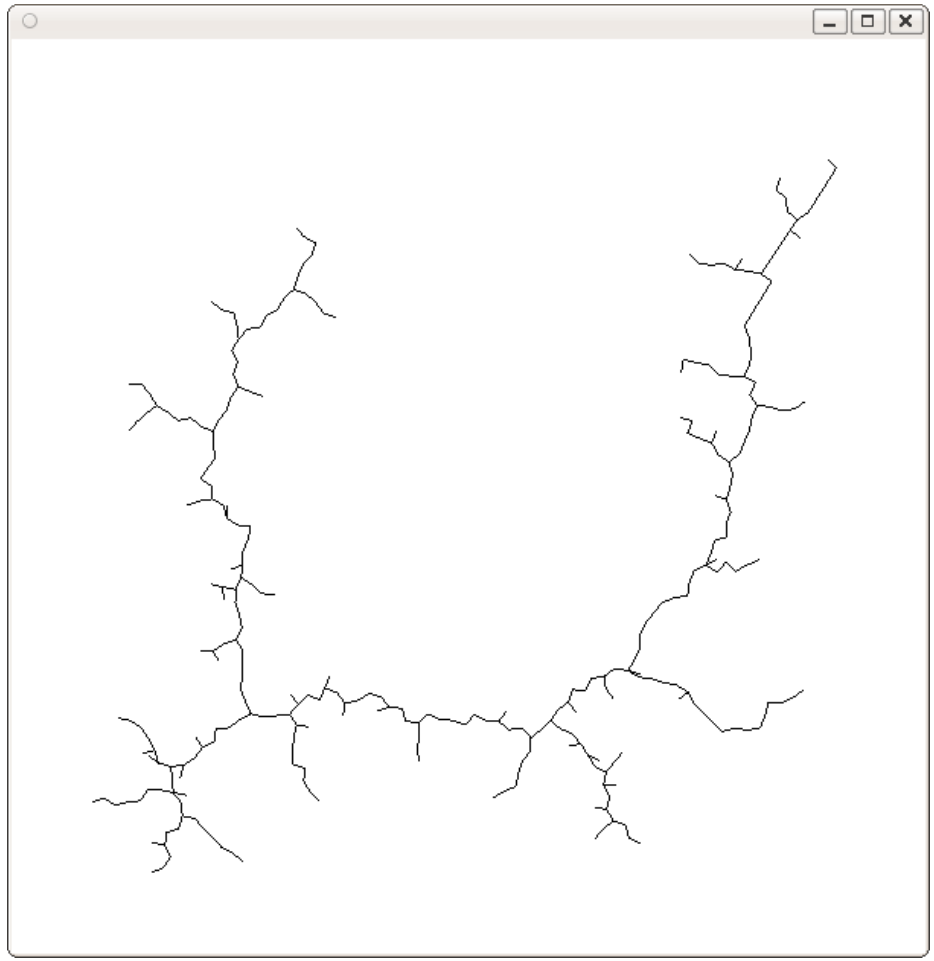
$n = 100$



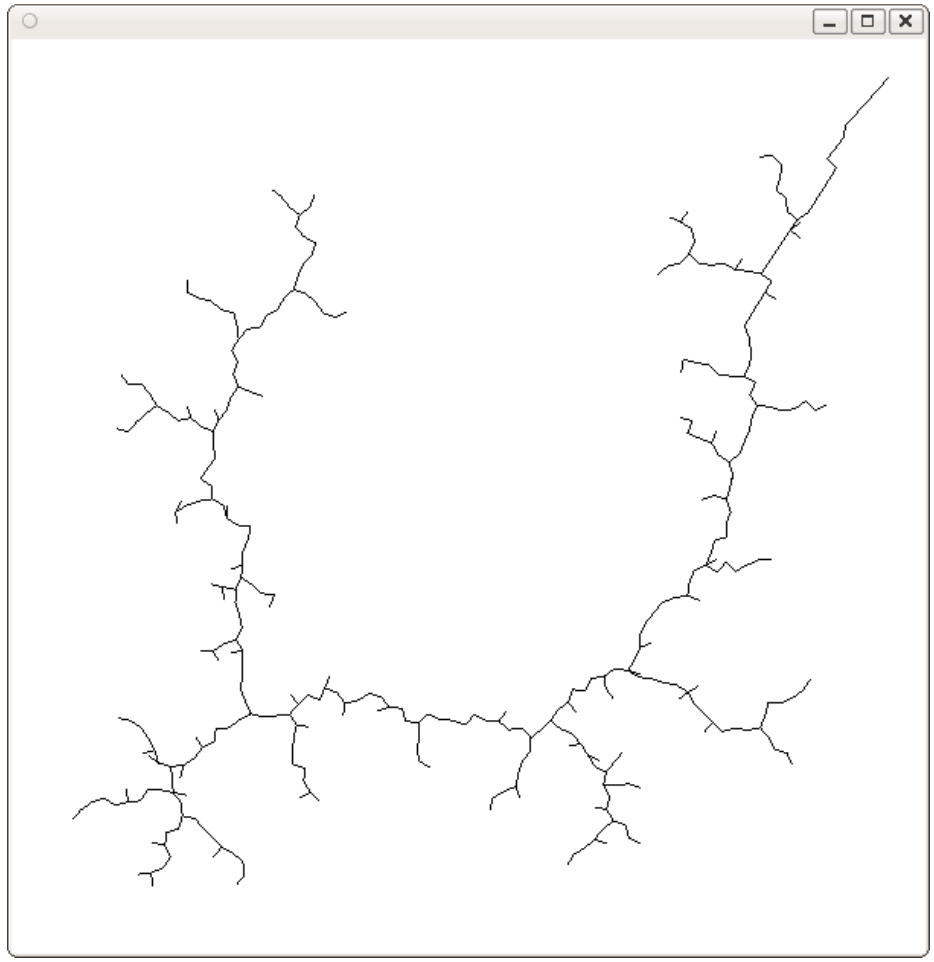
$n = 200$



$n = 300$

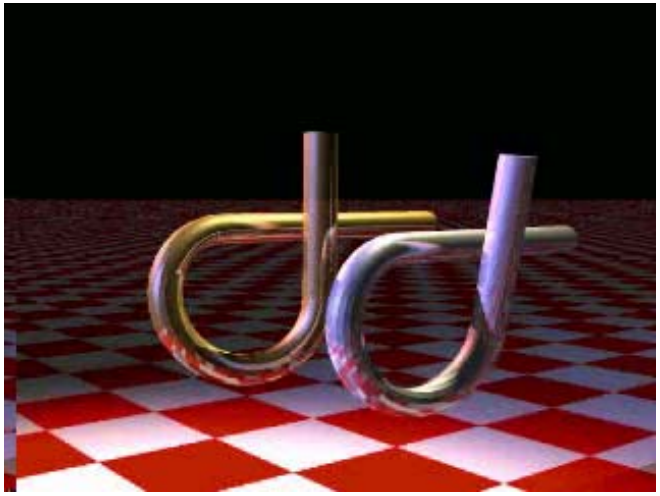


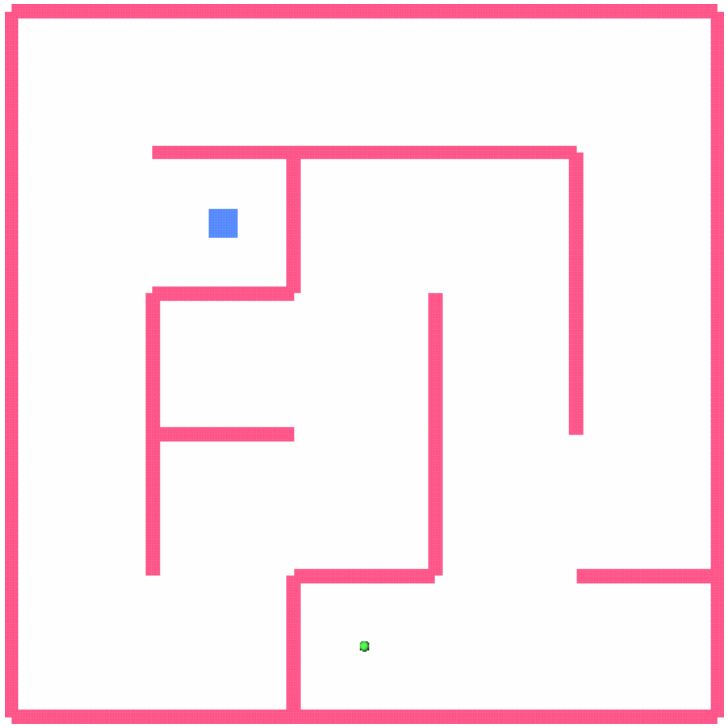
$n = 400$



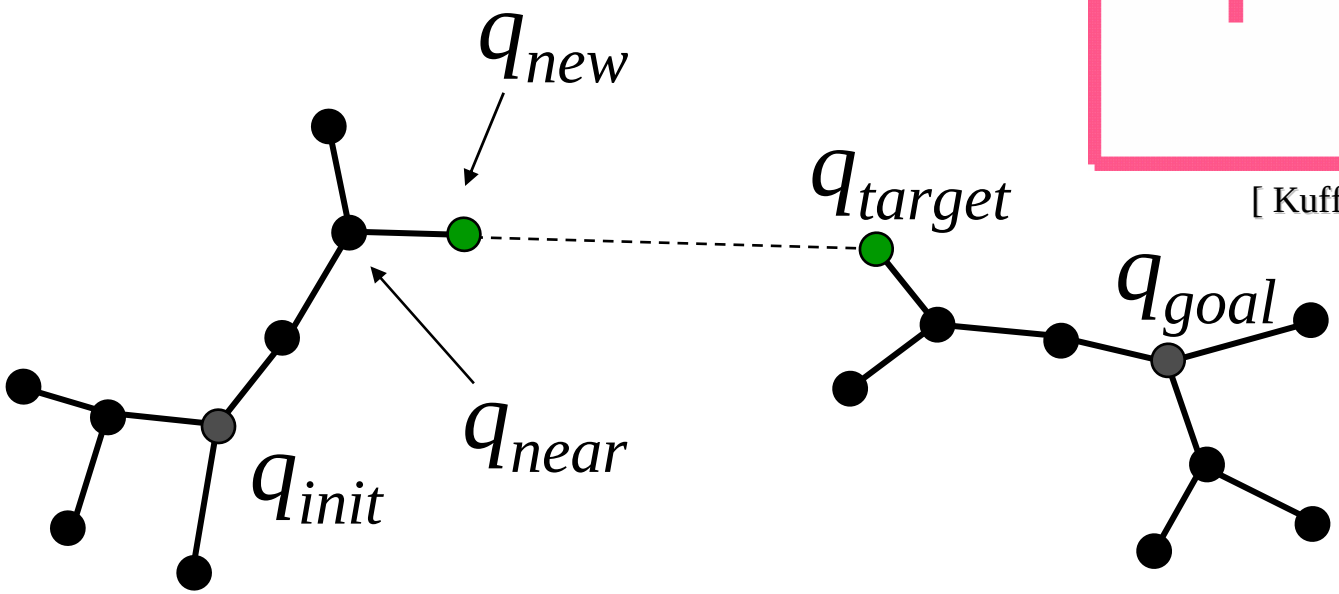
$n = 500$

Grow two RRTs towards each other

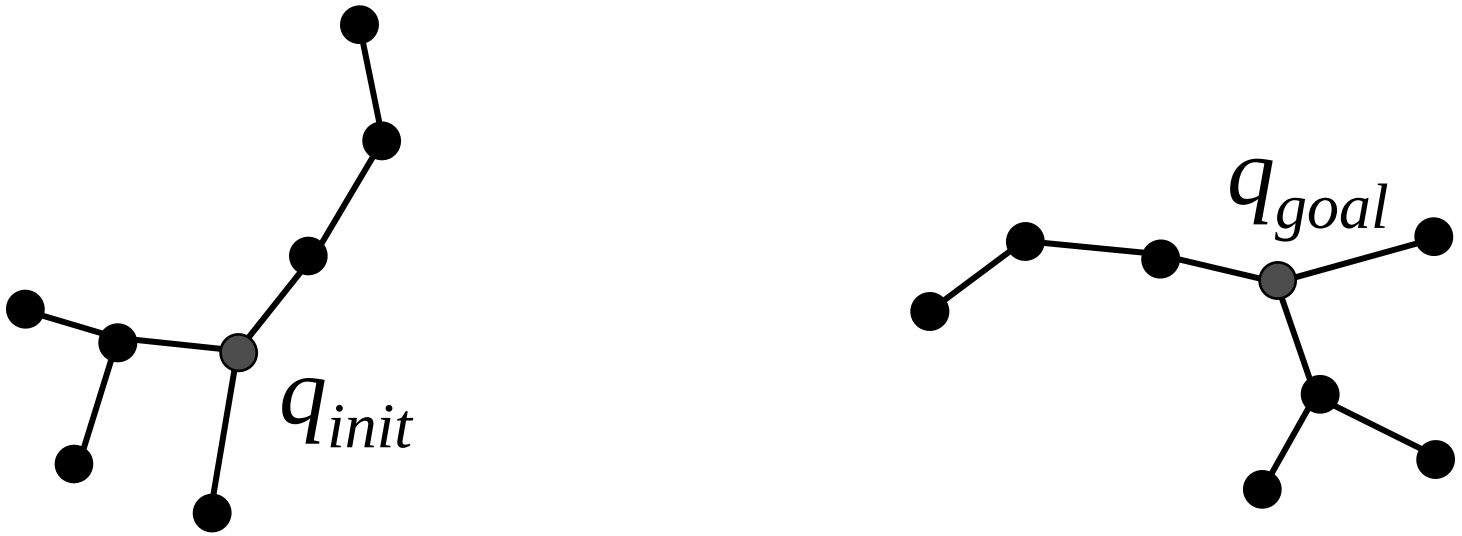




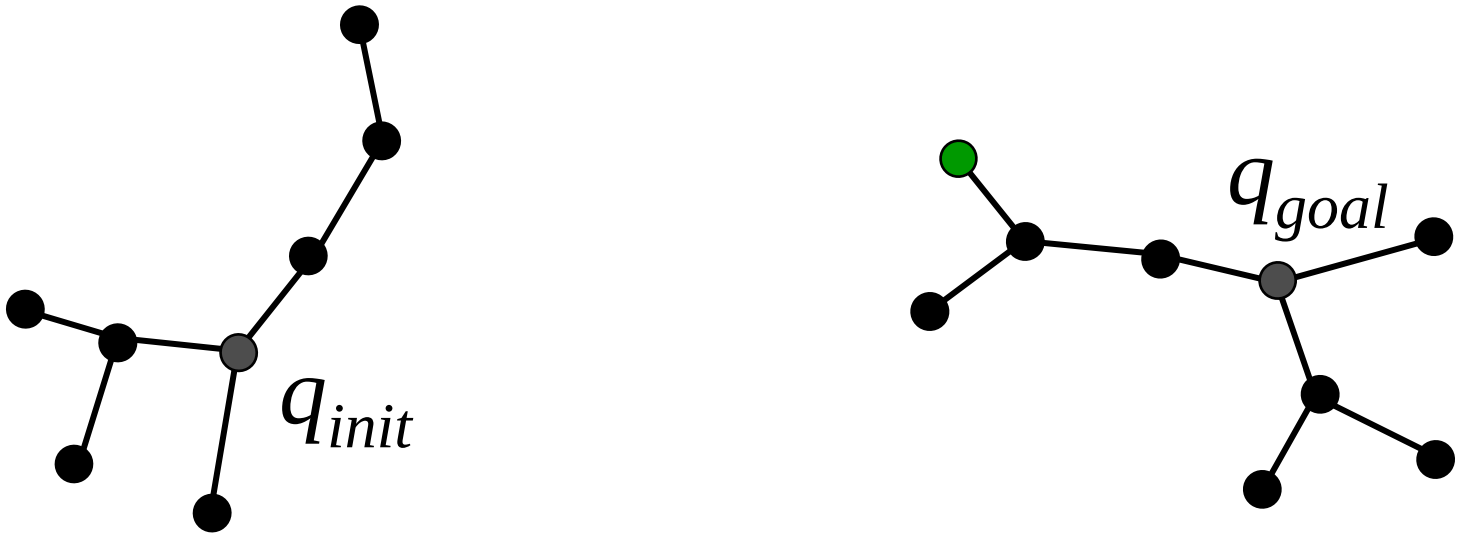
[Kuffner, LaValle ICRA '00]



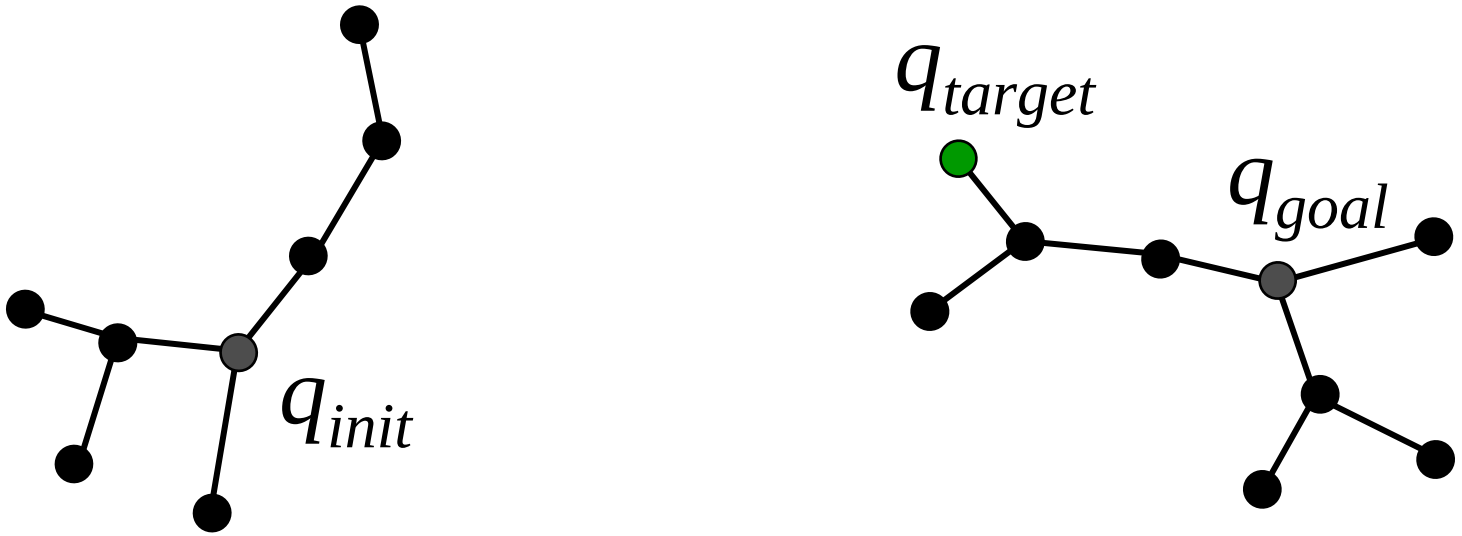
A single RRT-Connect iteration...



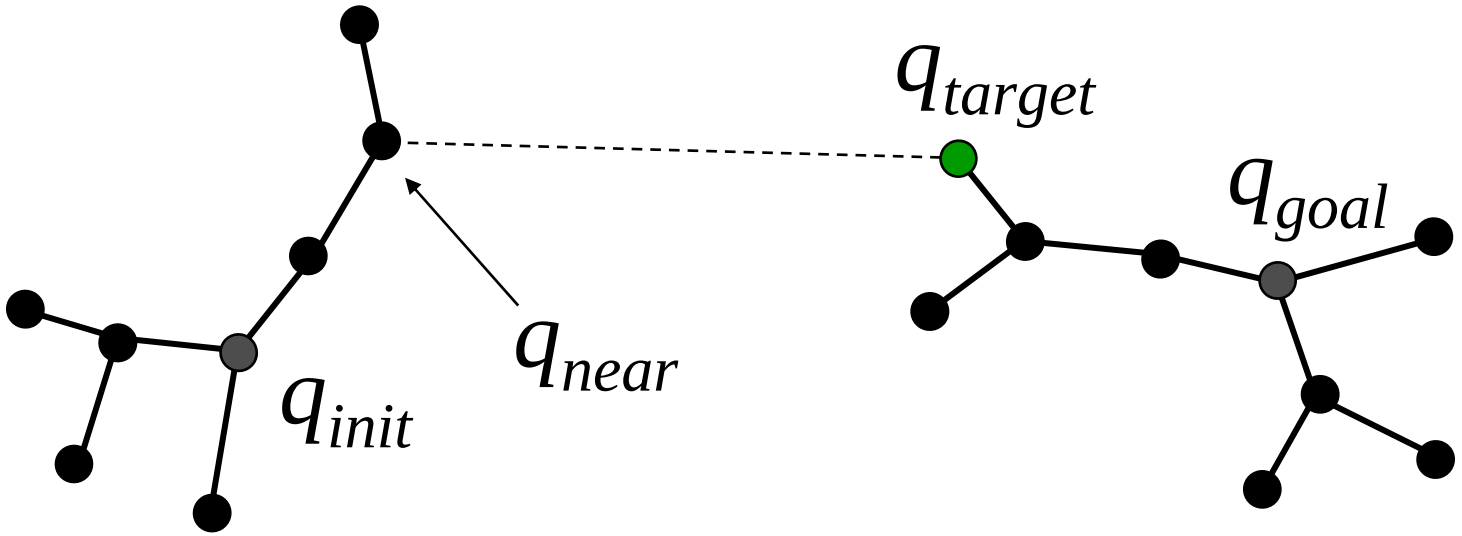
1) One tree grown using random target



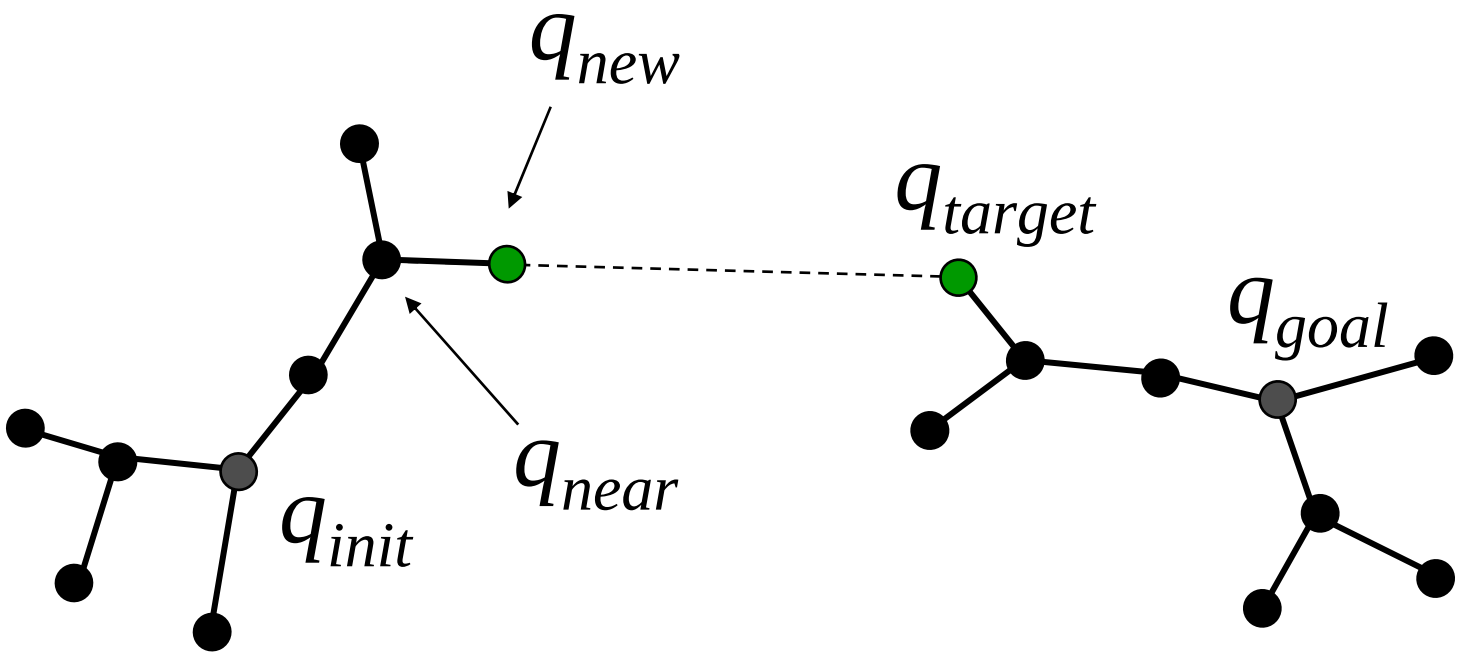
2) New node becomes target for other tree



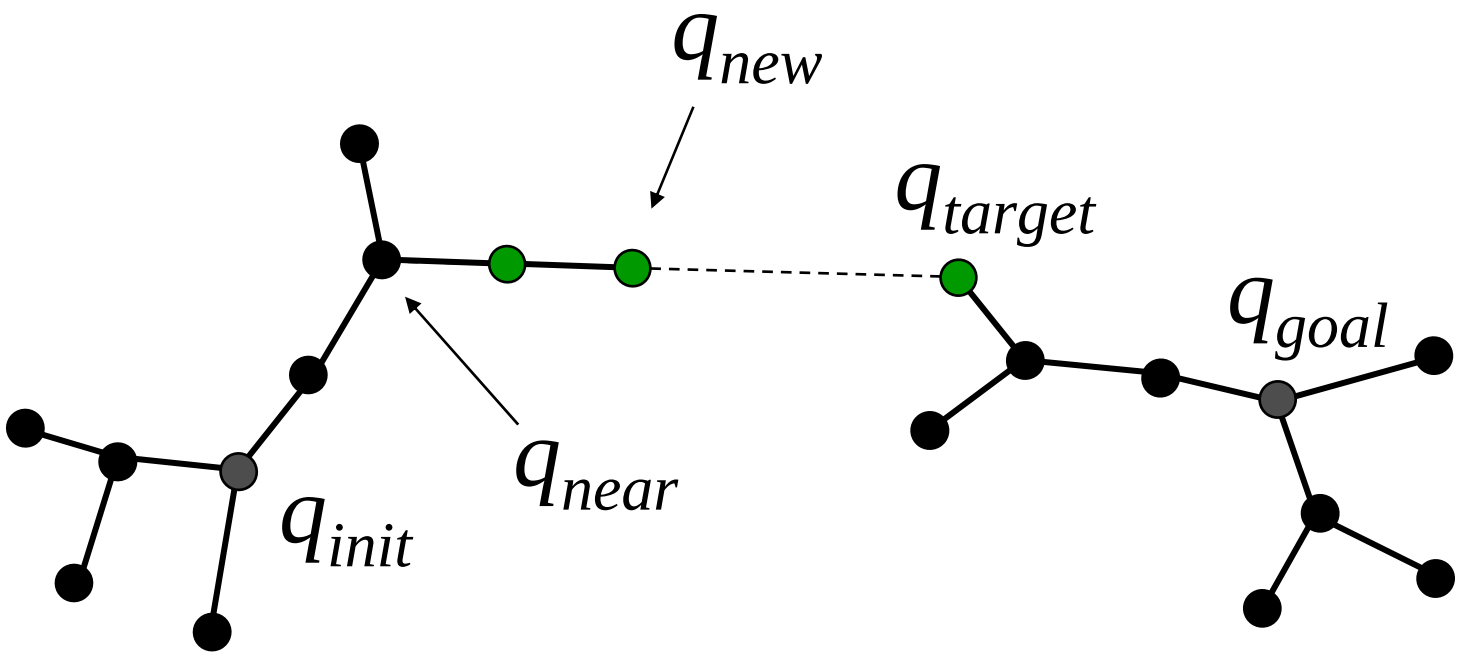
3) Calculate node “nearest” to target



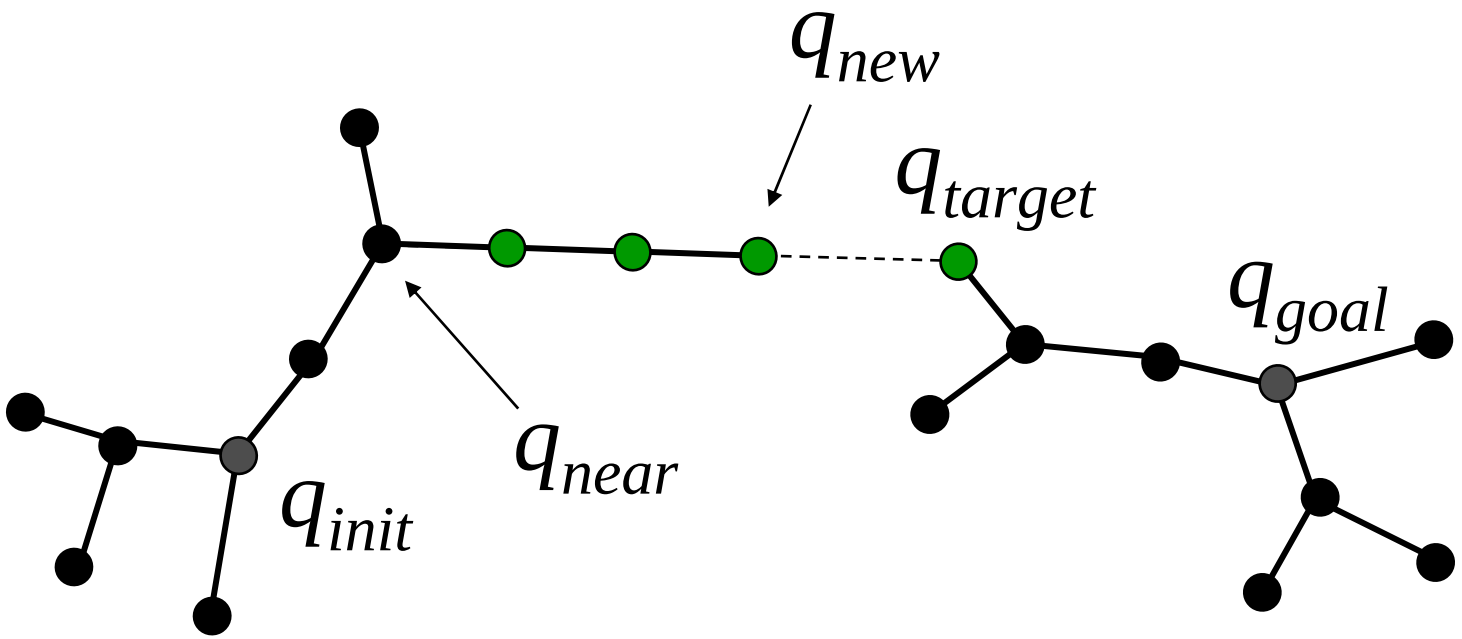
4) Try to add new collision-free branch



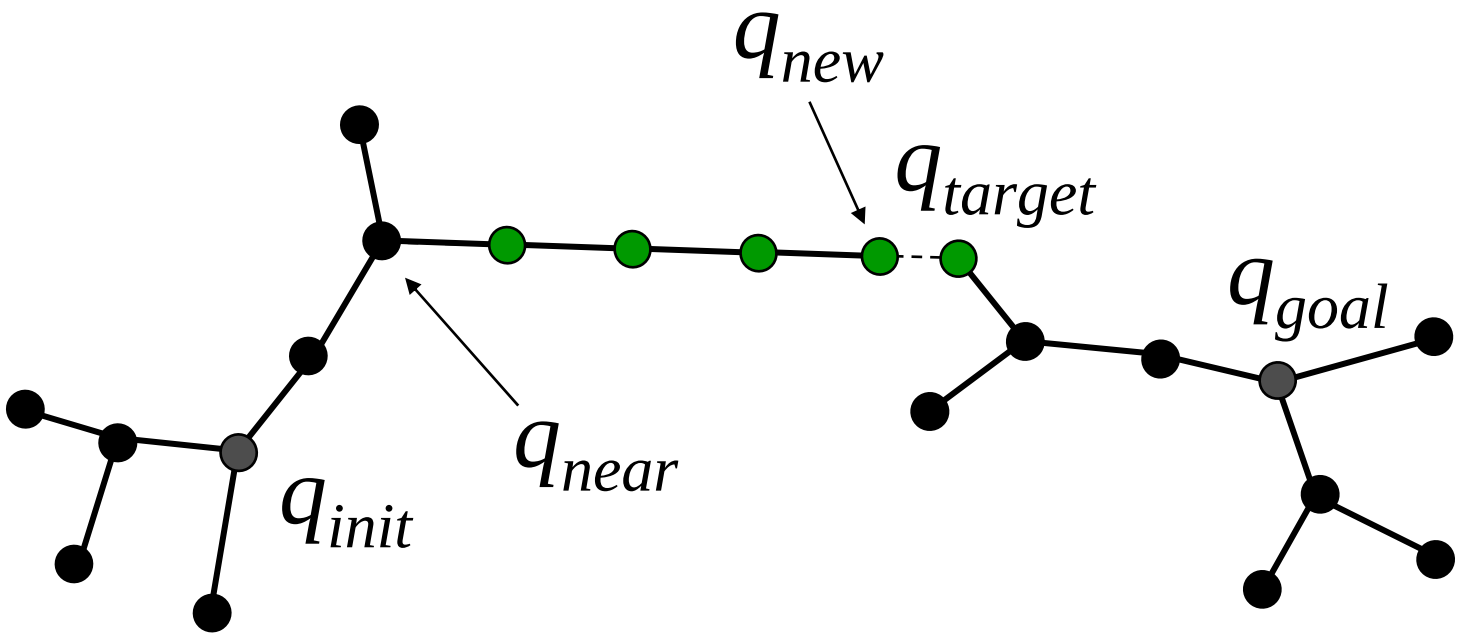
5) If successful, keep extending branch



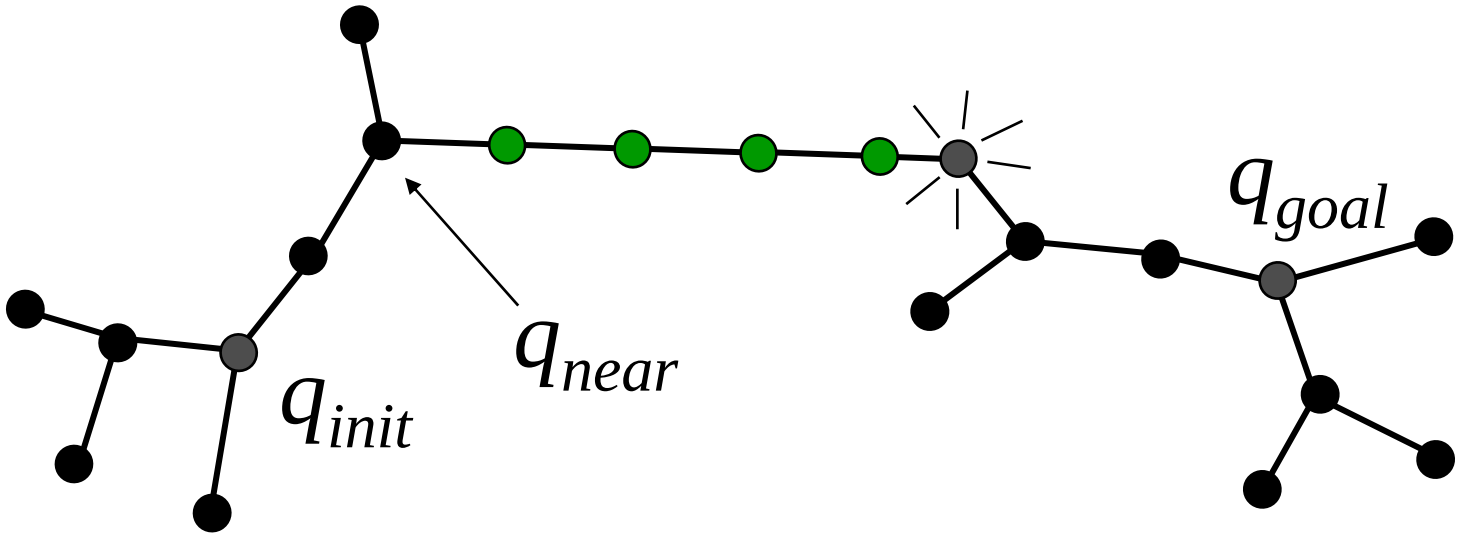
5) If successful, keep extending branch



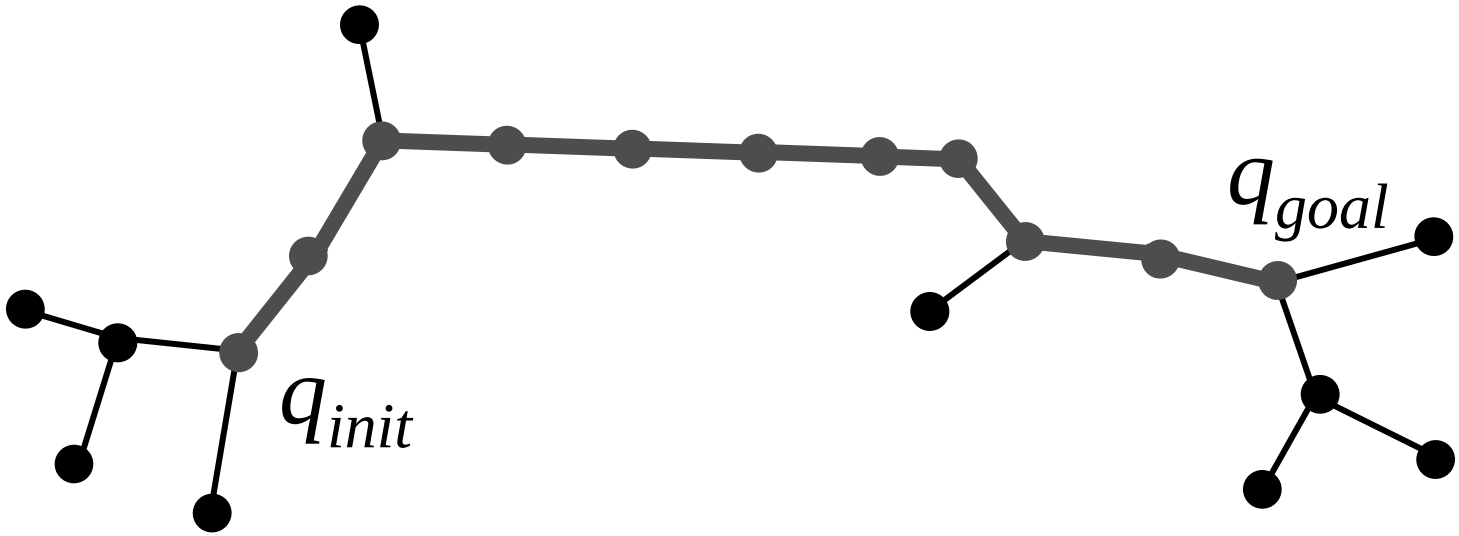
5) If successful, keep extending branch



6) Path found if branch reaches target

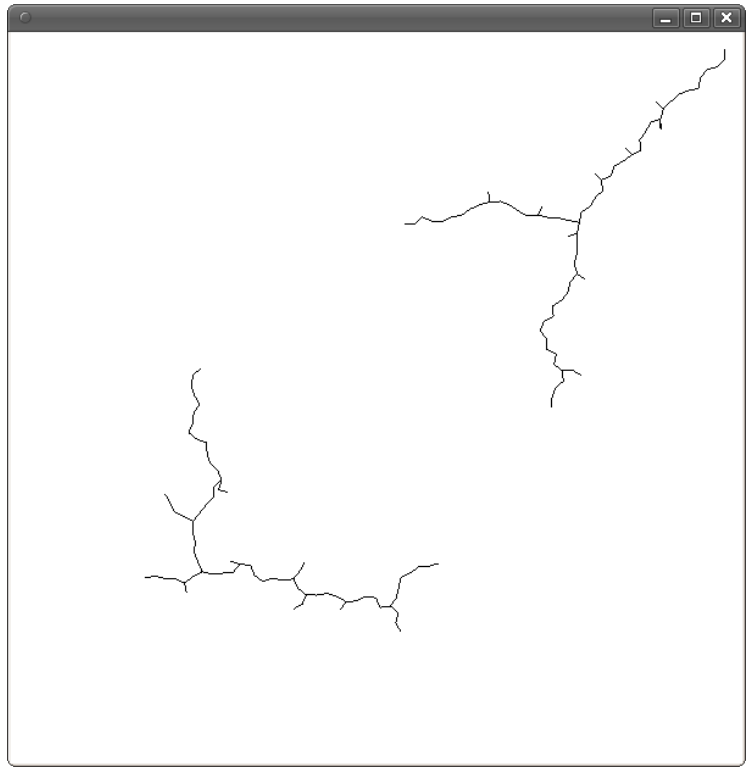


7) Return path connecting start and goal



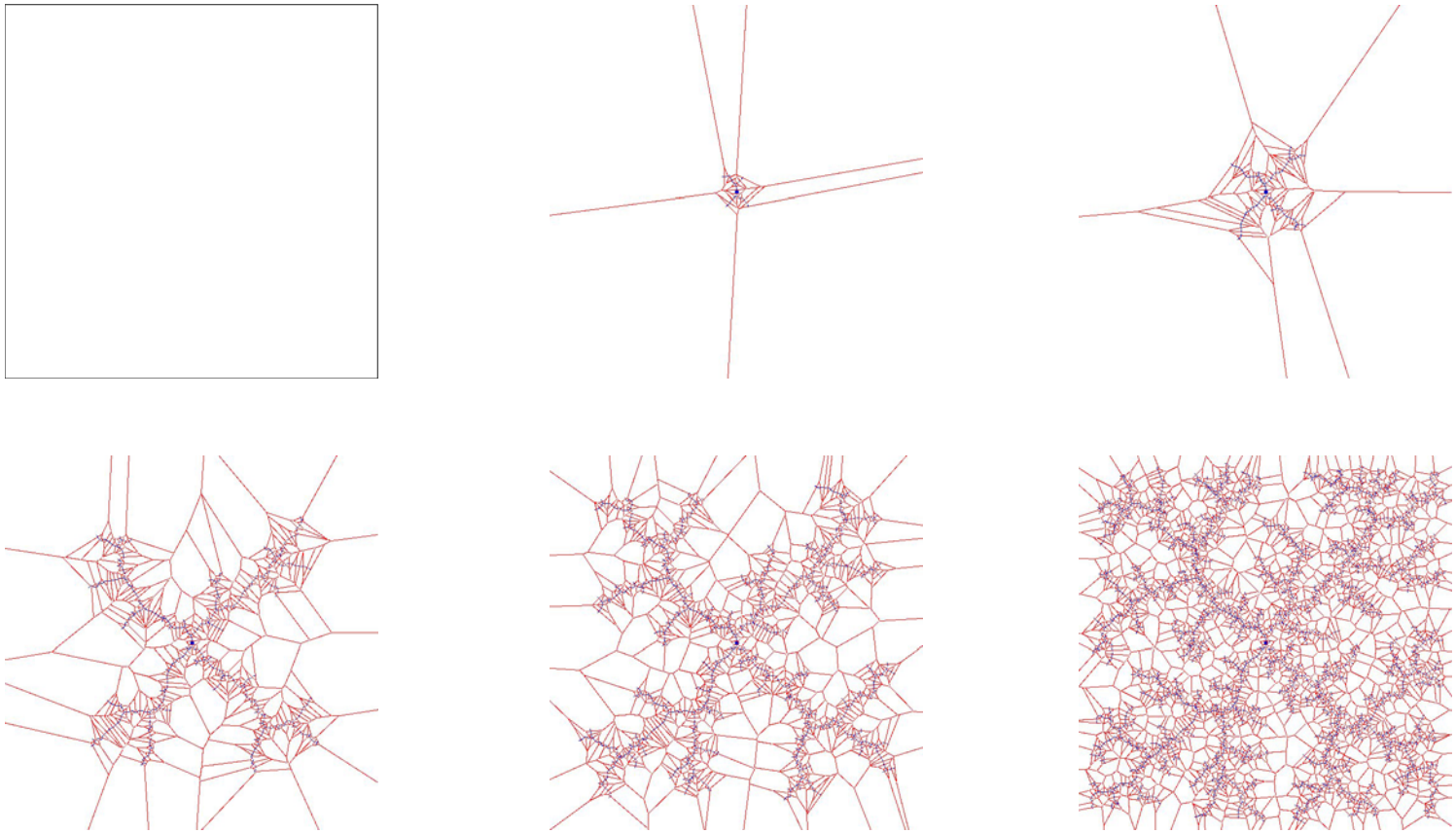
Bi-directional search

- grow two trees starting from q_{start} and q_{goal}



let one tree grow towards the other
(e.g., “choose q_{new} of T_1 as q_{target} of T_2”)

RRTs and
Bias toward large Voronoi regions



http://msl.cs.uiuc.edu/rrt/gallery.html

Summary: RRTs

- Pros (shared with PRMs):
  – Algorithmically very simple
  – Highly explorative
  – Allows probabilistic performance guarantees

- Pros (beyond PRMs):
  – Focus computation on single query $(q_{\text{start}}, q_{\text{goal}})$ problem
  – Trees from multiple queries can be merged to a roadmap
  – Can be extended to differential constraints (nonholonomic systems)

- To keep in mind (shared with PRMs):
  – The metric (for nearest neighbor selection) is sometimes critical
  – The local planner may be non-trivial