Book Ch 6.2 and 6.3.2-6.3.3, 4.3.2

ED5215

Sampling-Based Planners

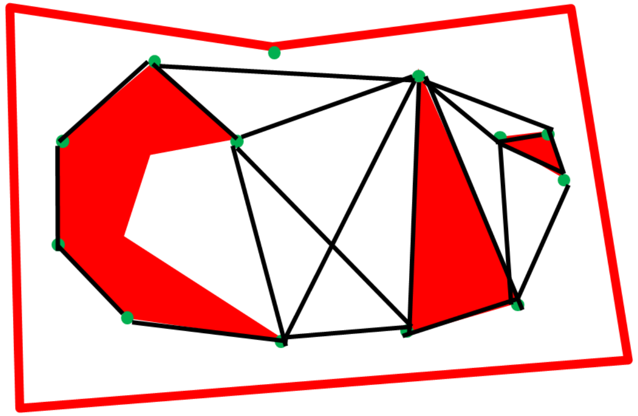
Nirav Patel

Assistant Professor, Department of Engineering Design

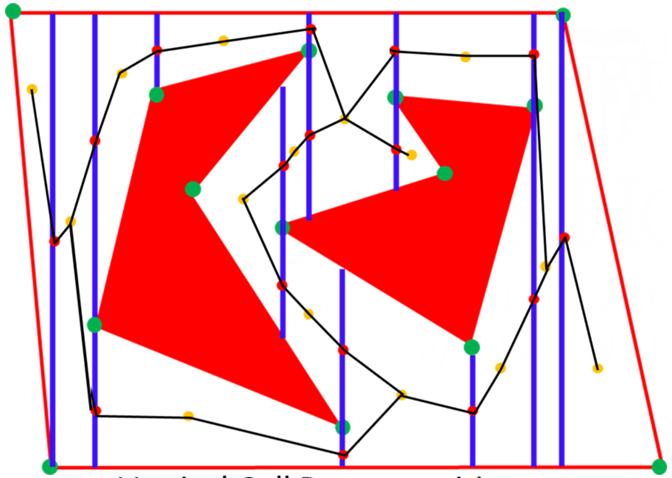
IIT Madras

Moodle page available at: https://coursesnew.iitm.ac.in/

Visibility Roadmaps



Reduced visibility map



Vertical Cell Decomposition

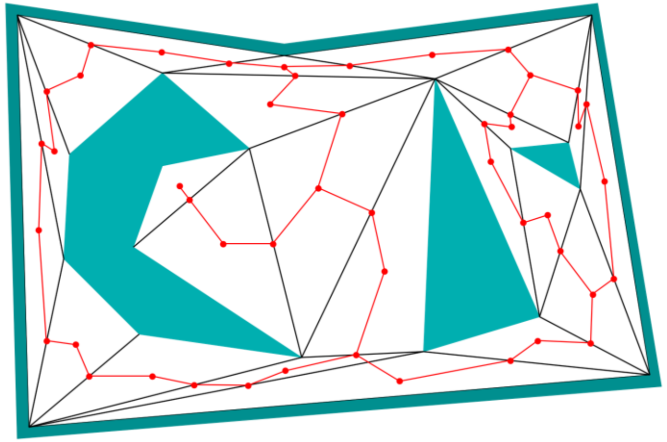
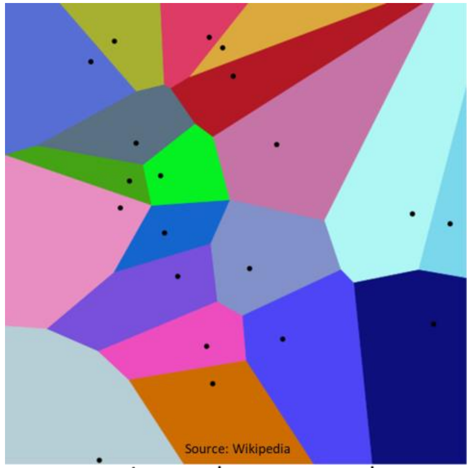
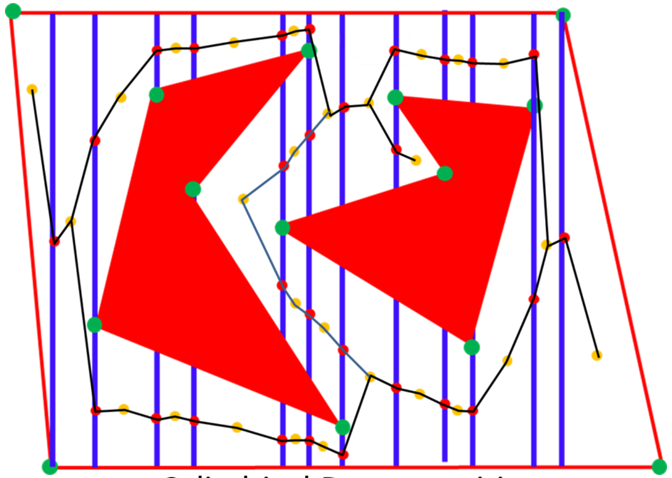


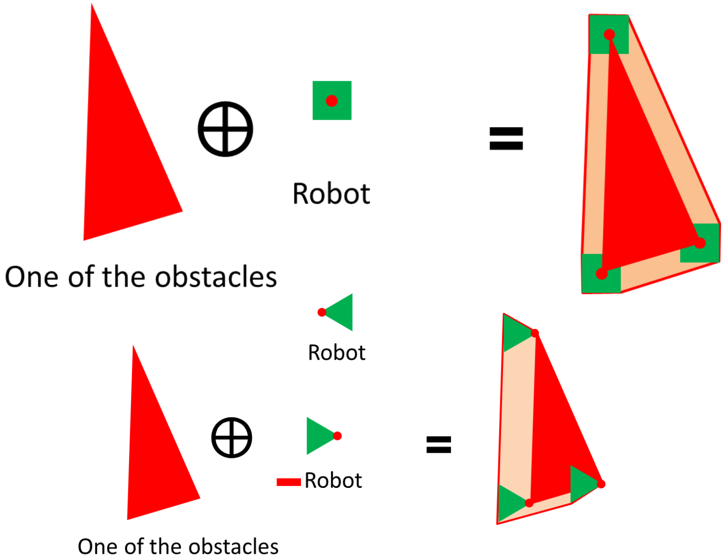
Figure 6.17: A roadmap obtained from the triangulation.



maximum clearance roadmap



Cylindrical Decomposition



Minkowski Sum

World isn't that simple/nice !

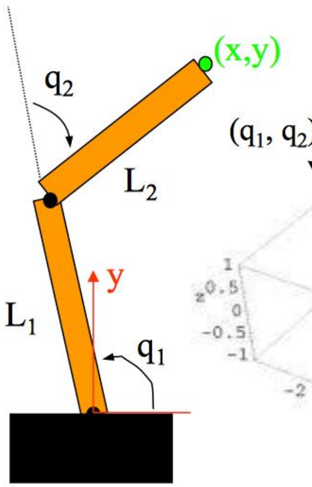
- Relax most assumptions
  - Complex robot model and environments
  - Robot may not be polygonal
  - C-space obstacles may not be polygonal
  - Robot can rotate and translate
  - Robot with chain of rigid bodies
- Grid-based and visibility graph based methods don't generalize
- Finding a feasible path might become challenging
- Solution
  - Lets look at them!!

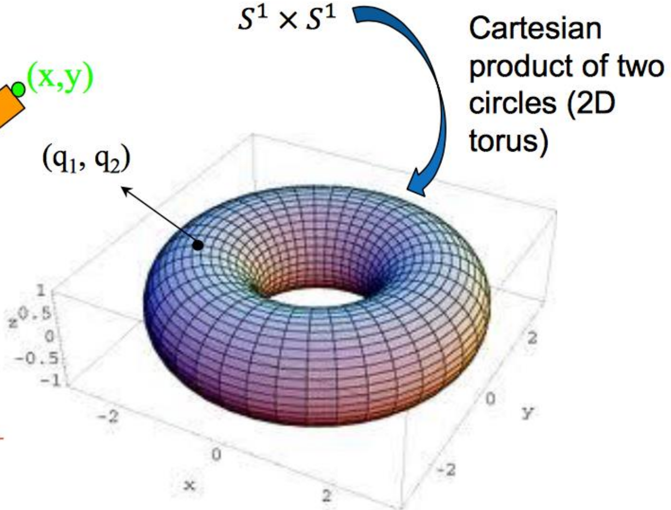
What is a configuration

- A complete specification of the robot
- Mobile robot that can move only left-right, back-forth
  - Will have configuration specified by two variables $(x, y) \in R^2$
- Mobile robot that can move only left-right, back-forth and rotate
  - has configuration specified by three variables x,y, theta $\in R^2 X [0,2\pi)$
- A serial manipulator with 2 joints
  - Has configuration specified by two variables $(q_1 , q_2) \in [0,2\pi) X [0,2\pi)$
- A serial manipulator with ***n*** joints
  - has configuration specified by two variables $(q_1, q_2, \ldots, q_n) \in [0,2\pi) X [0,2\pi) X \ldots X [0,2\pi)$

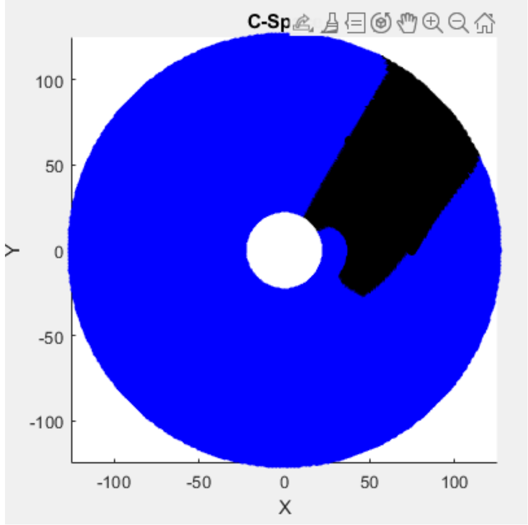
Configuration space: C-space

- All possible configurations
  - Typically a large number
- A two DOF robot
- How about if we create a grid in the C-space?
- C-space wont have any vertices/edges in C-space obstacles



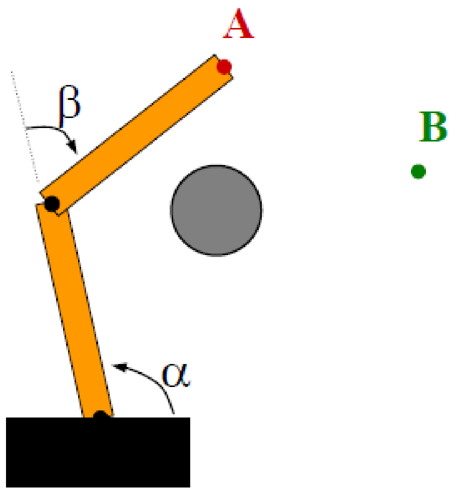


Topology of Configuration Space



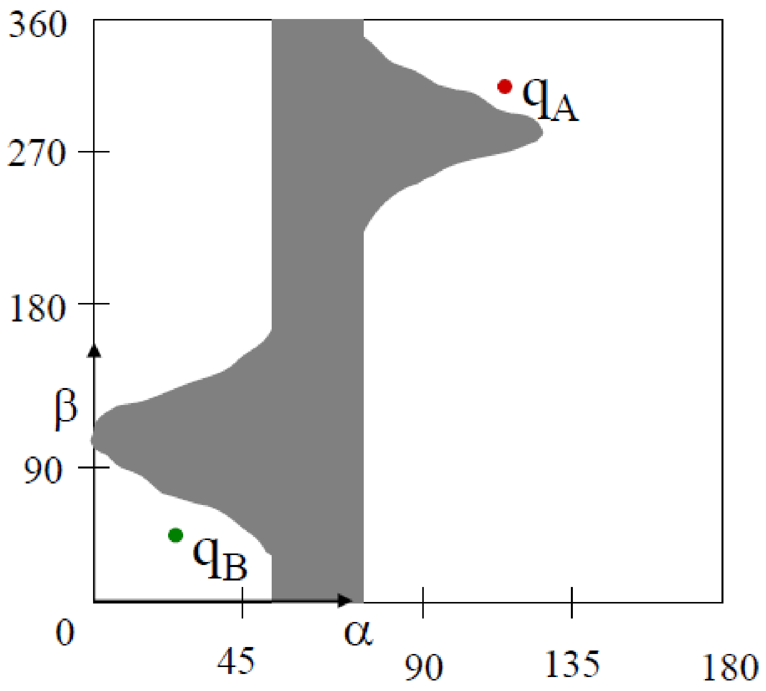
C-Space with obstacle

Reference *configuration*



An obstacle in the robot's workspace

How do we get from A to B ?

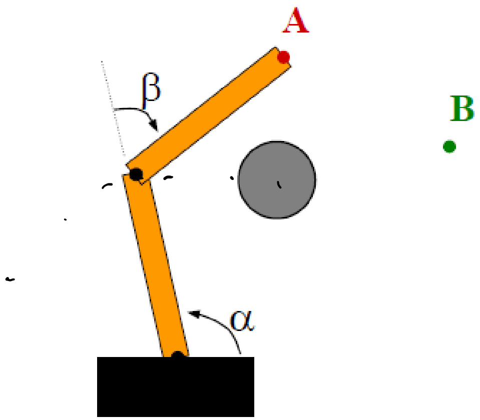


The C-space representation
of this obstacle…

16-735, Howie Choset with slides from G.D. Hager, Z. Dodds, and Dinesh Mocha

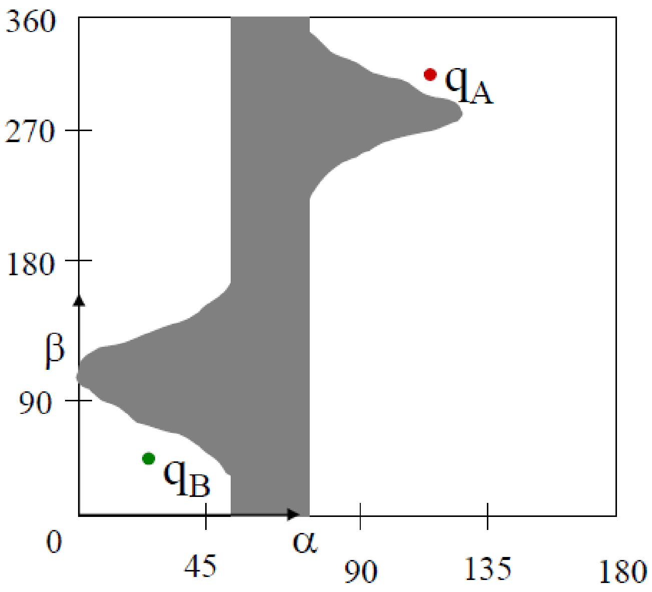
C-Space with obstacle

Reference *configuration*



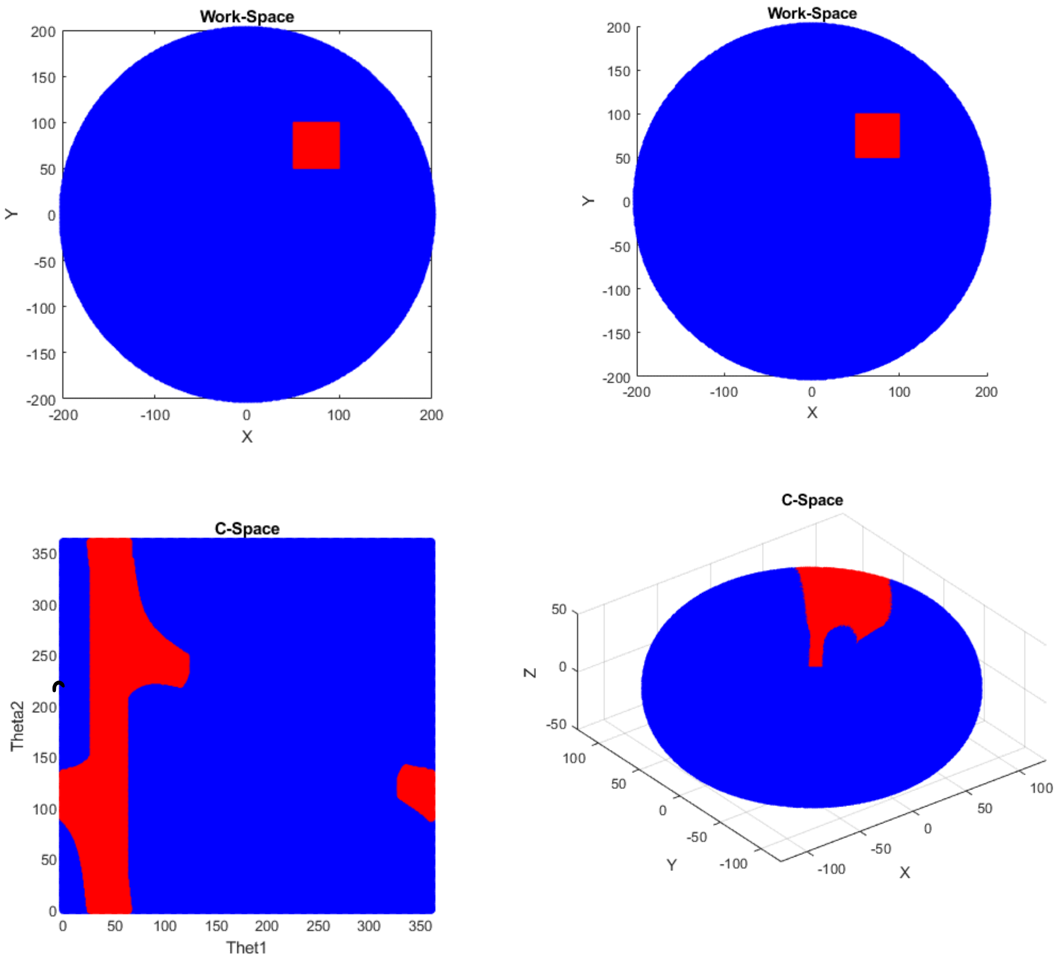
An obstacle in the robot's workspace

How do we get from **A** to **B** ?



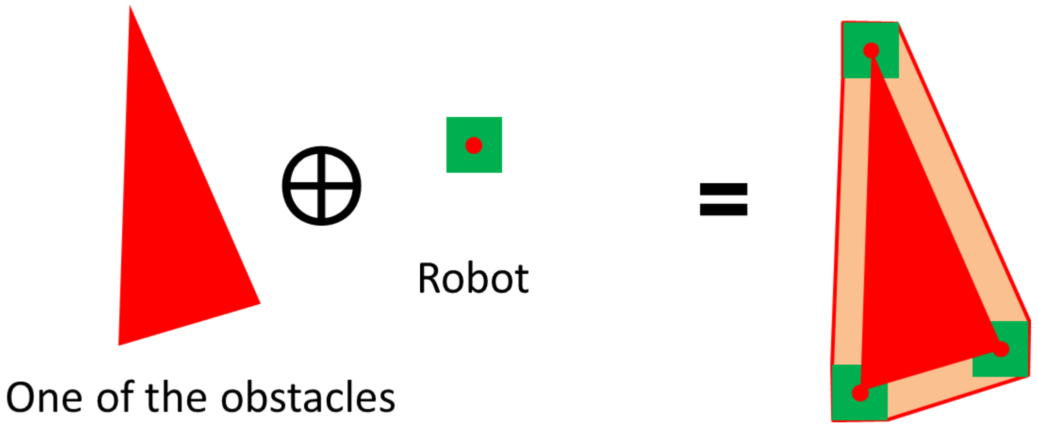
The C-space representation of this obstacle…

16-735, Howie Choset with slides from G.D. Hager, Z. Dodds, and Dinesh Mocha

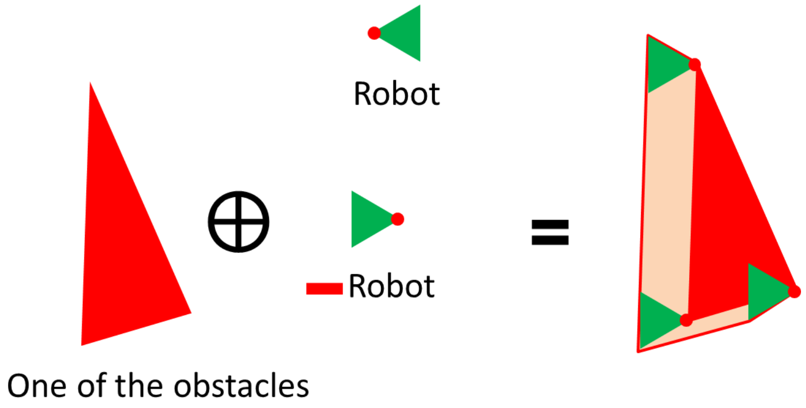


How do we compute C-space

- If we keep the assumption that robot and obstacles are polygonal
  - Minkowski sum/difference



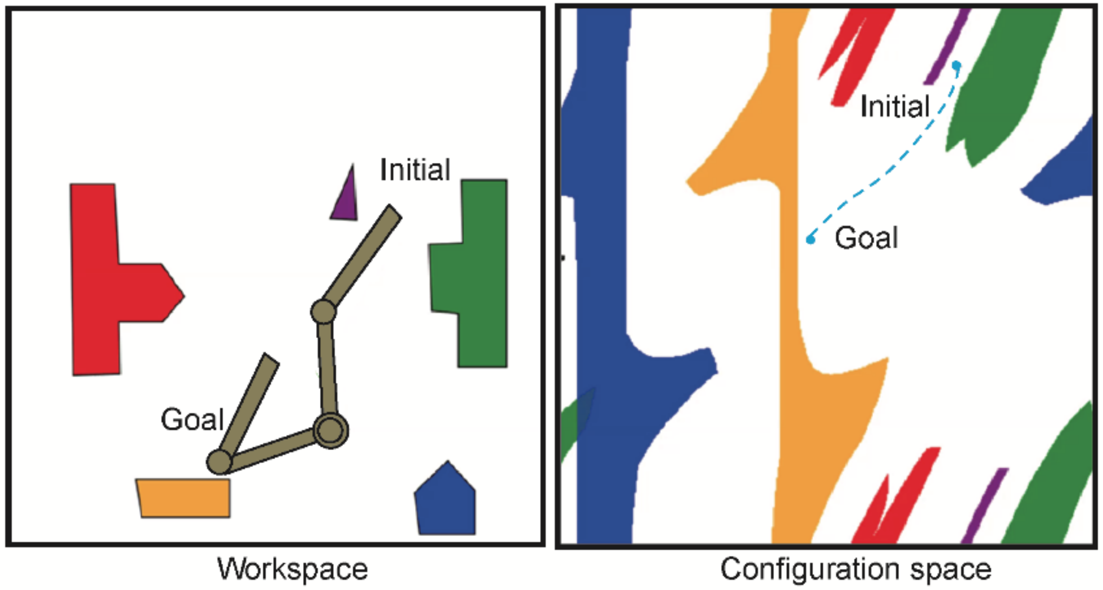
Minkowski Sum



Minkowski Difference

How do we compute C-space

- If they are not polygonal then??



- C-Space can be decomposed into
  - C-Space$_{free}$
  - C-Space$_{obs}$
- How to decompose the C-Space
  - Iterate through all possible configurations and check if its in ?
    - Free-space -> add to C-Space$_{free}$
    - Obstacle/collision -> add to C-Space$_{obs}$

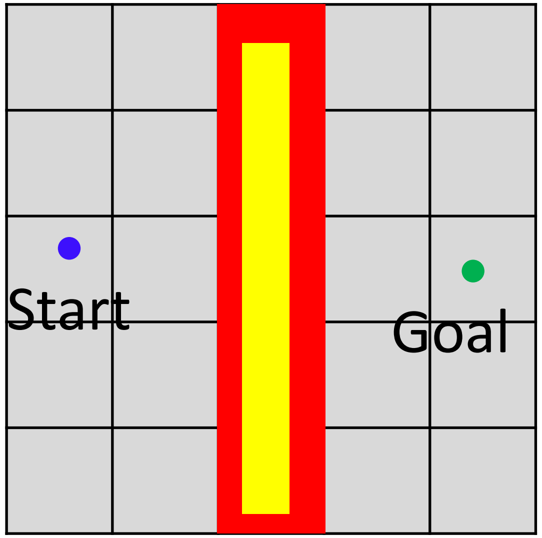
Image source: Pratap Tokekar, ECE 4984/5984: (Advanced) Robot Motion Planning, Virginia tech

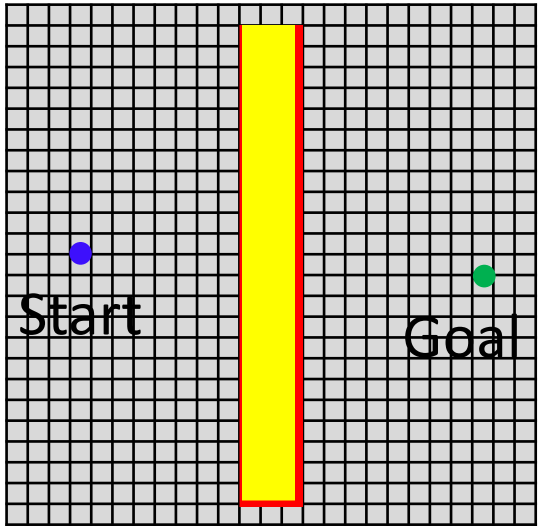
We can't compute C-Space$_{obs}$: Solution 1

- Choose a configuration (remember what is a configuration)
- Check if this configuration is in collision
  – Self-collision
  – Collision with an obstacle
- To check collision we need a collision checker
  – Lets assume we have a really fast collision checker (most physical engines do this for us)
- Lets discretize the C-Space (in principle nothing is continuous when we use computers !!)
  – Any problems here???

We can't compute C-Space$_{obs}$: Solution 1

- Discretize the C-Space (in principle nothing is continuous when we use computers !!)
  – Grid resolution/grid size???
    - Low-res -> may not find a solution !!
    - High-res -> too large!!





- For each collision free configuration add a vertex to a graph
  – Remember in C-Space robot is a point/vertex

We can't compute C-Space$_{obs}$: Solution 1

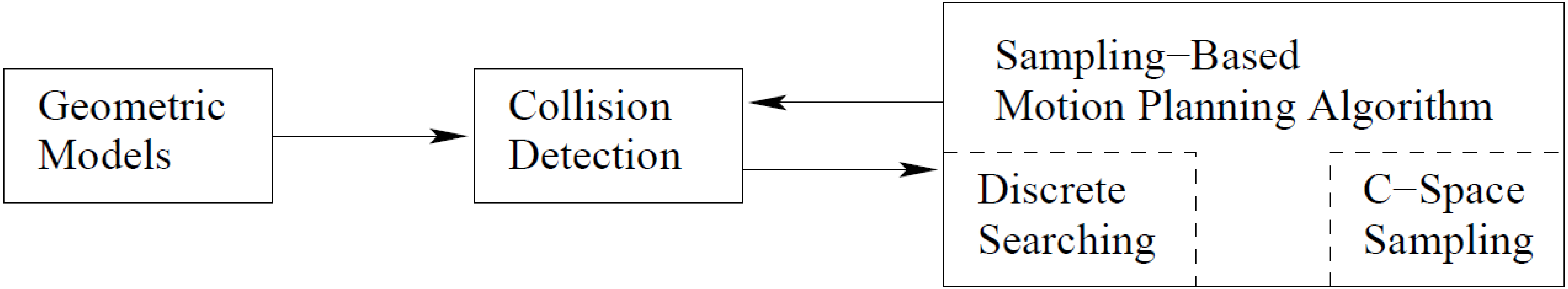
- Remember we are in C-Space not in the work/taskspace
- Robot is a point/vertex
- C-Space can be high-dimensional manifold
  - Complex and obstacles could present themselves as narrow openings !!
- Finding a feasible path itself might be challenging
- Finding an optimal path: well that's a good dream!!!

We can’t compute C-Space$_{obs}$: Solution 2

- Some form of algorithm to choose/produce a sequence of configurations in C-Space
  - We get samples $\boldsymbol{x_i}$ in C-Space
- Check if sample $\boldsymbol{x_i}$ is in C-Space$_{free}$ or in C-Space$_{obs}$
- If $\boldsymbol{x_i}$ is in C_{free}
  - Add $\mathbf{x}_i$ to a graph
  - We have to create new edge/s : we talk about this in coming lectures
- If $\boldsymbol{x_i}$ is in C_{obs}
  - Discard it and get a new sample
- Build the graph and then we know what to do with that graph!!
  - Another kind of a roadmap!!

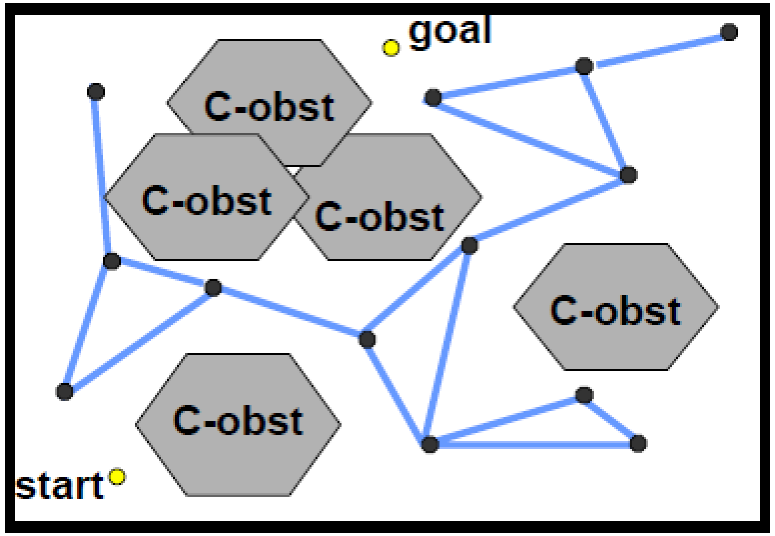
We can't compute C-Space$_{obs}$: Solution 2

- **Environment**
  - **Obstacles**
  - **Free space**
- **Collision checking**



Probabilistic Roadmaps

- What's good?
  - Sampling-based
  - Probabilistically complete : finds a solution given enough time
  - No need to construct the C-Space
  - Work well for higher dimensional spaces
  - Multiple queries possible once we have build the map/graph



- What's bad?
  - Doesn't work well for some problems
  - Might not sample in narrow passages
  - Might not be able to connect sub-graphs on constraint surfaces
  - Neither optimal nor complete!

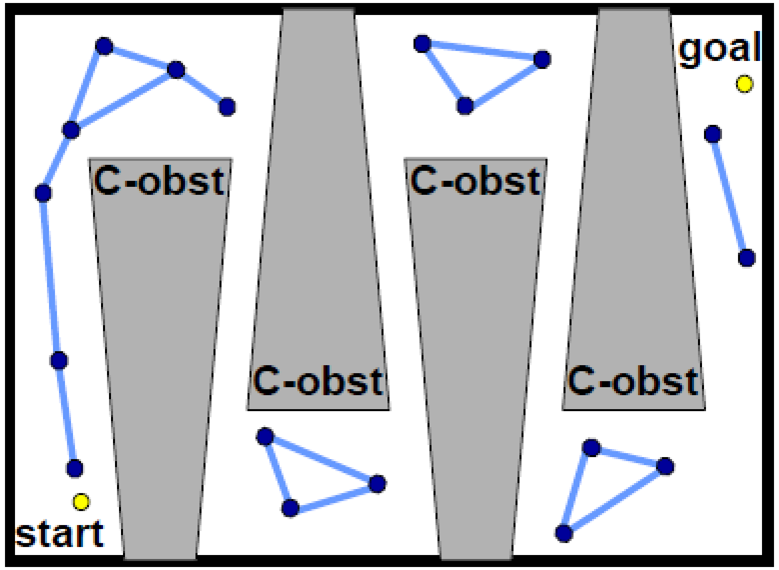


Image source: RI 16-735, Howie Choset with slides from Nancy Amato, Sujay Bhattacharjee, G.D. Hager, S. LaValle, and a lot from James Kuffner

Probabilistic Roadmaps

- Two phases
  - Learning phase
    - Random free configurations are connected using a fast local motion planner
  - Query phase
    - Add start and goal to the graph!!
- **DO WE NEED TO HAVE ONLY POLYGONAL OBSTACLES??**

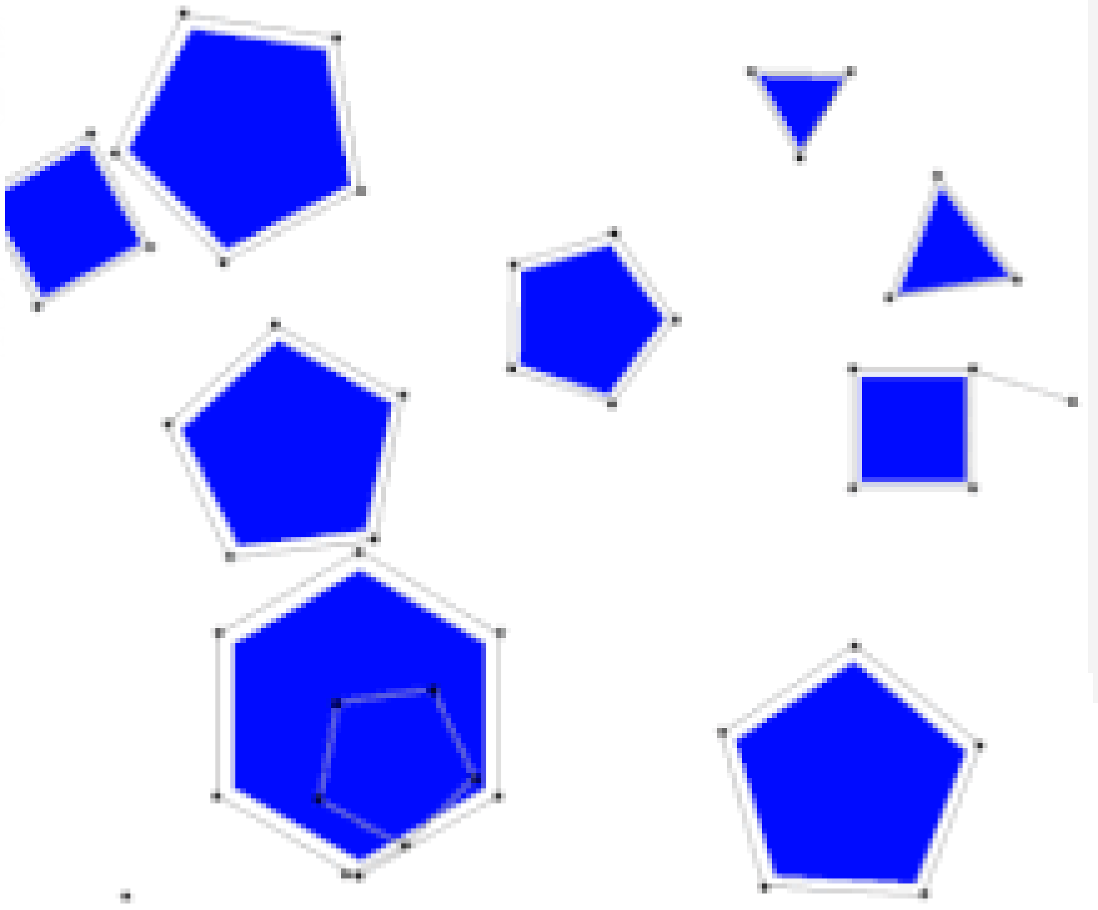
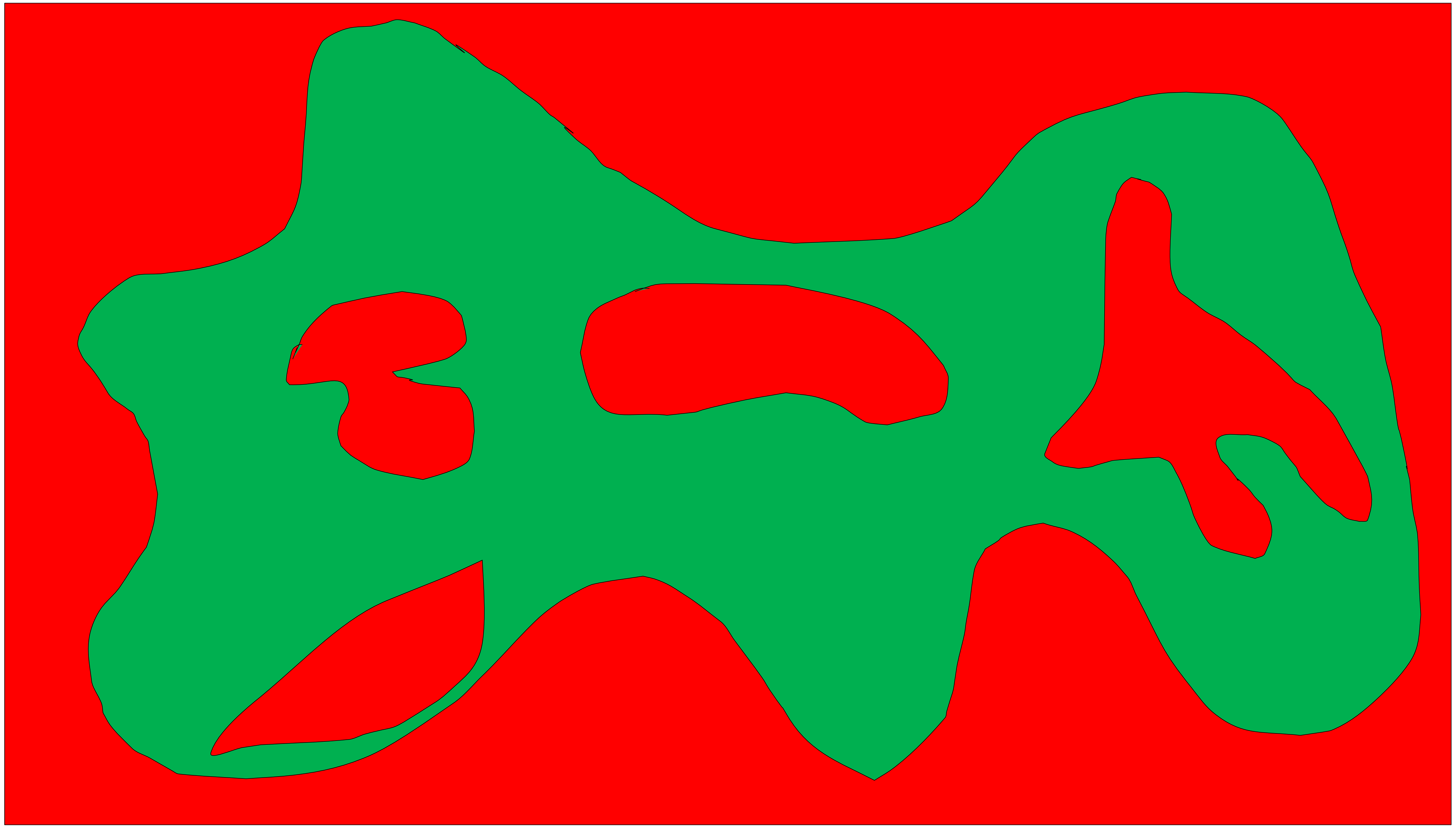


Image Source: Wikipedia

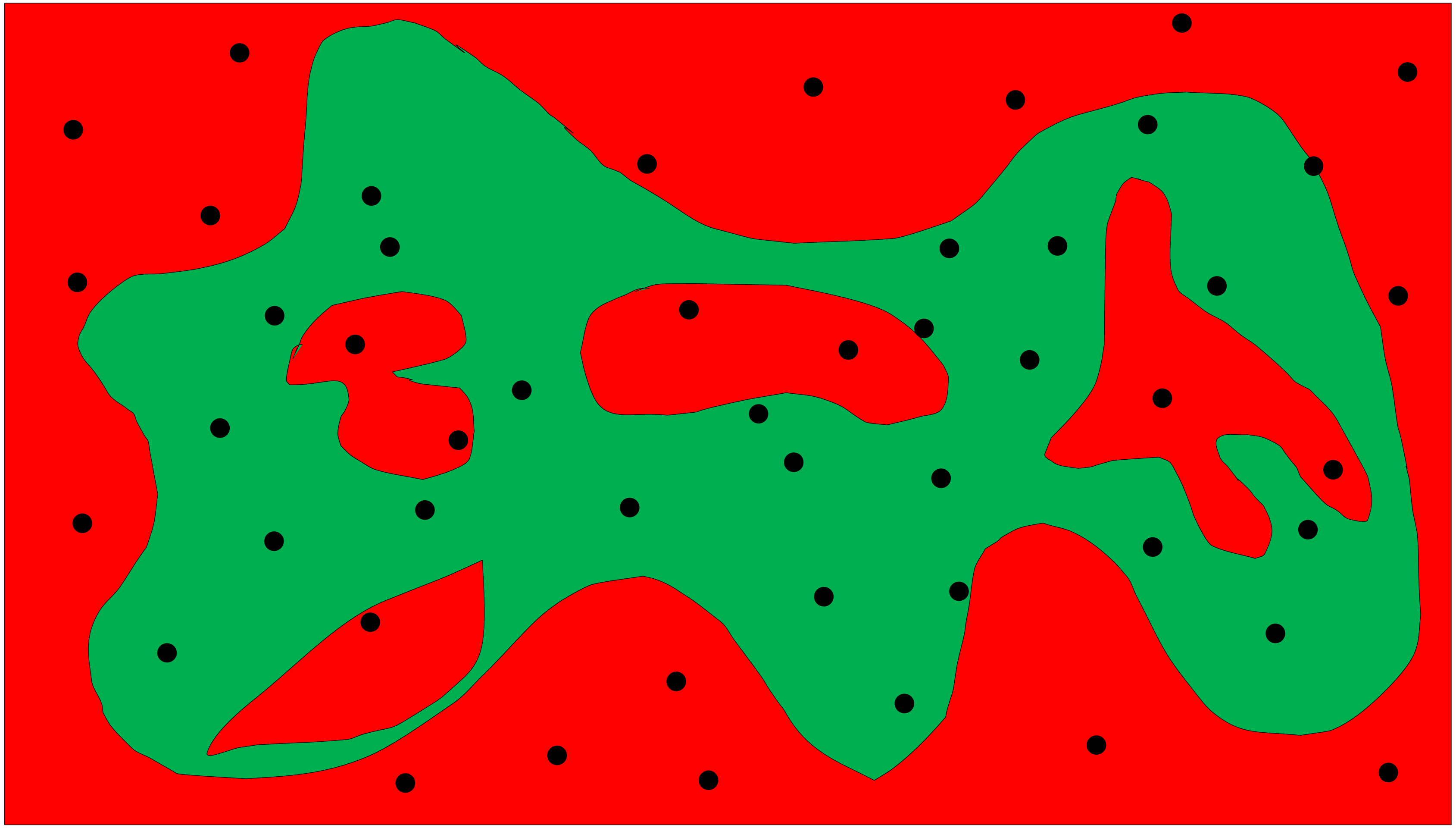
Probabilistic Roadmaps

- Lets assume some C-Space in $\mathbf{R}^n$ like below
- **Red**: obstacles or $\mathbf{C}_{\mathbf{obs}}$
- **Green**: free space or $\mathbf{C}_{\mathbf{free}}$



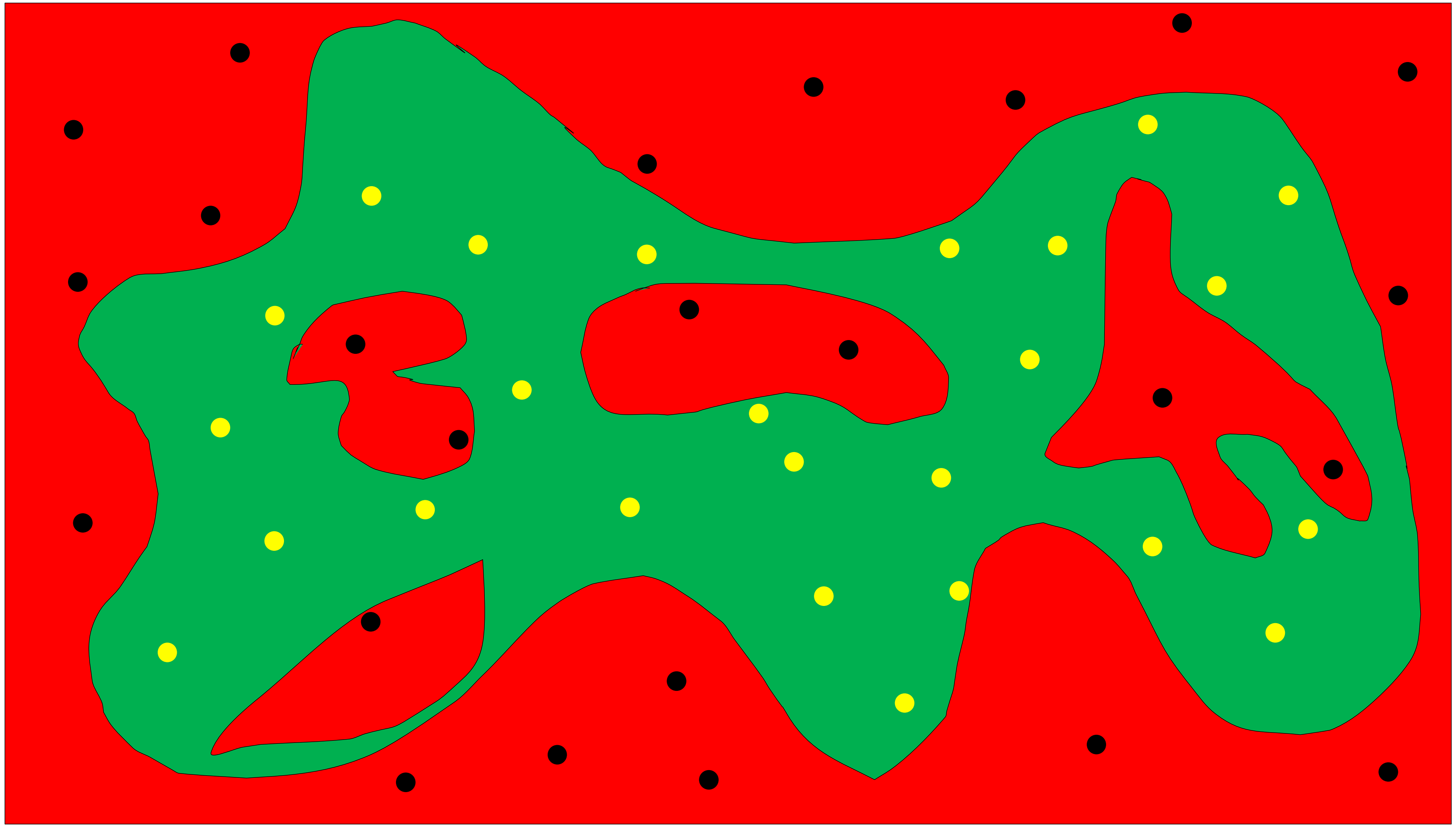
Probabilistic Roadmaps

- Pick a random configuration



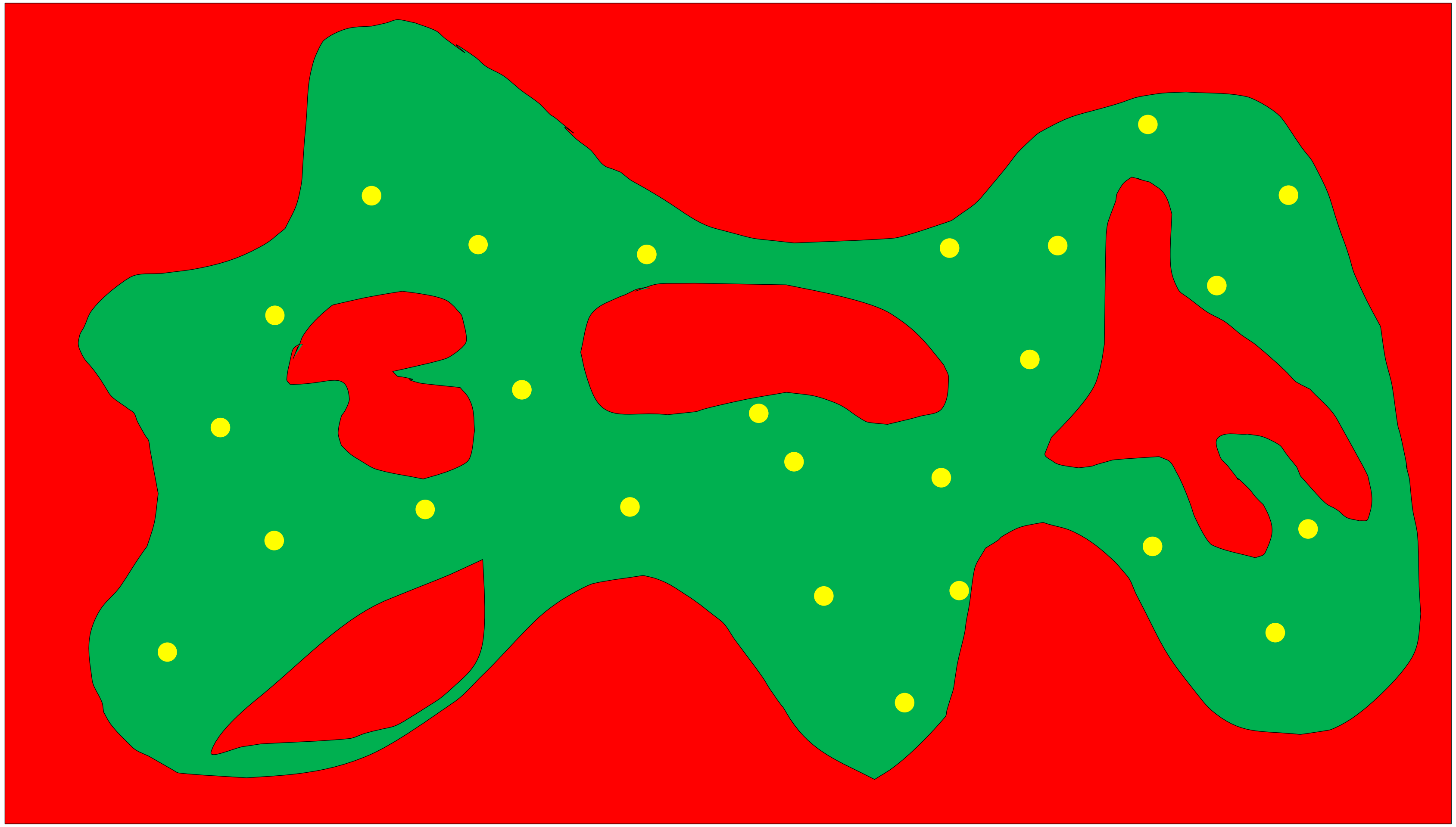
Probabilistic Roadmaps

- Check if random samples are in $\mathbf{C_{free}}$ or $\mathbf{C_{obs}}$



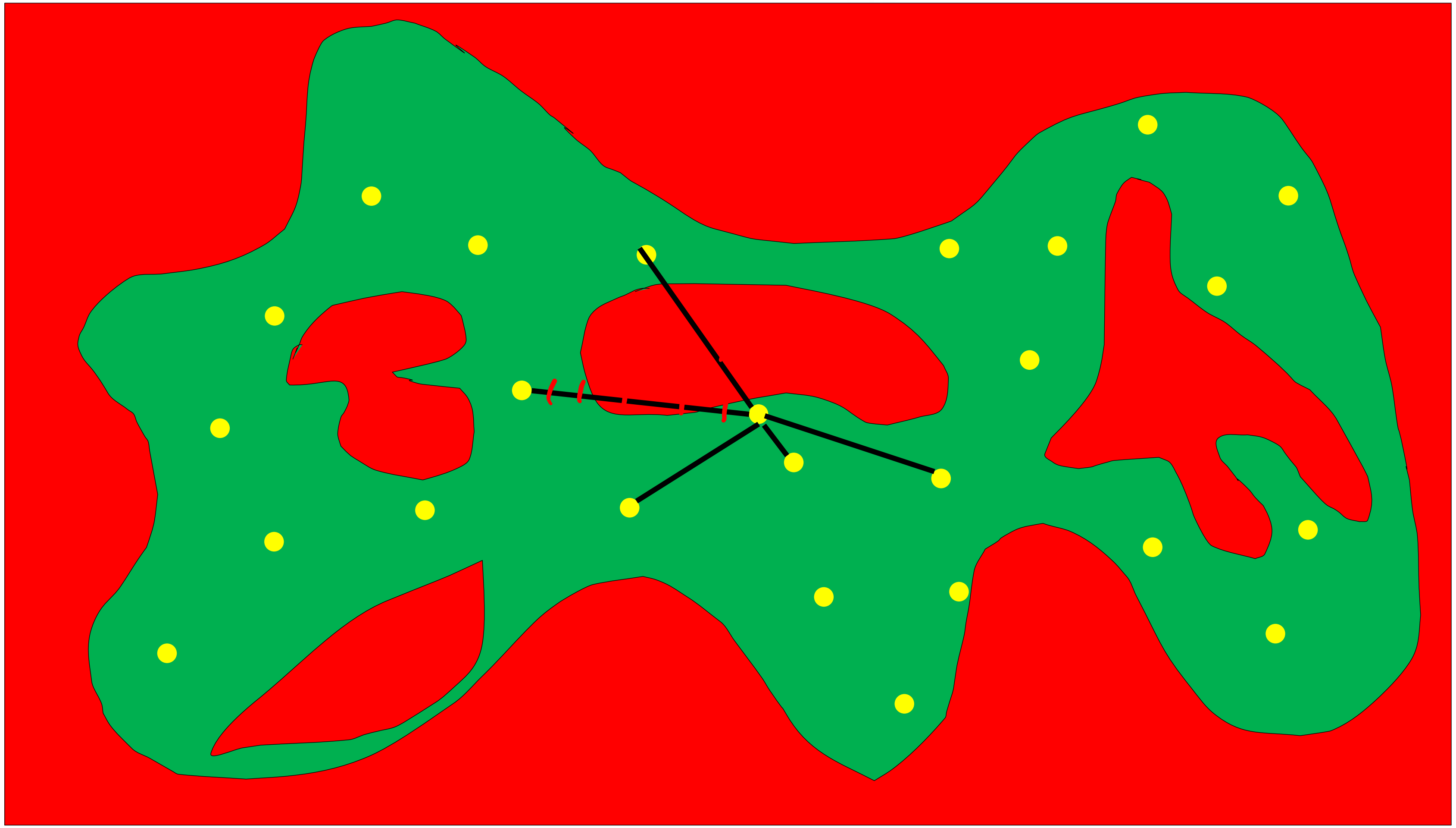
Probabilistic Roadmaps

- Discard the ones in $\mathbf{C}_{\mathbf{obs}}$



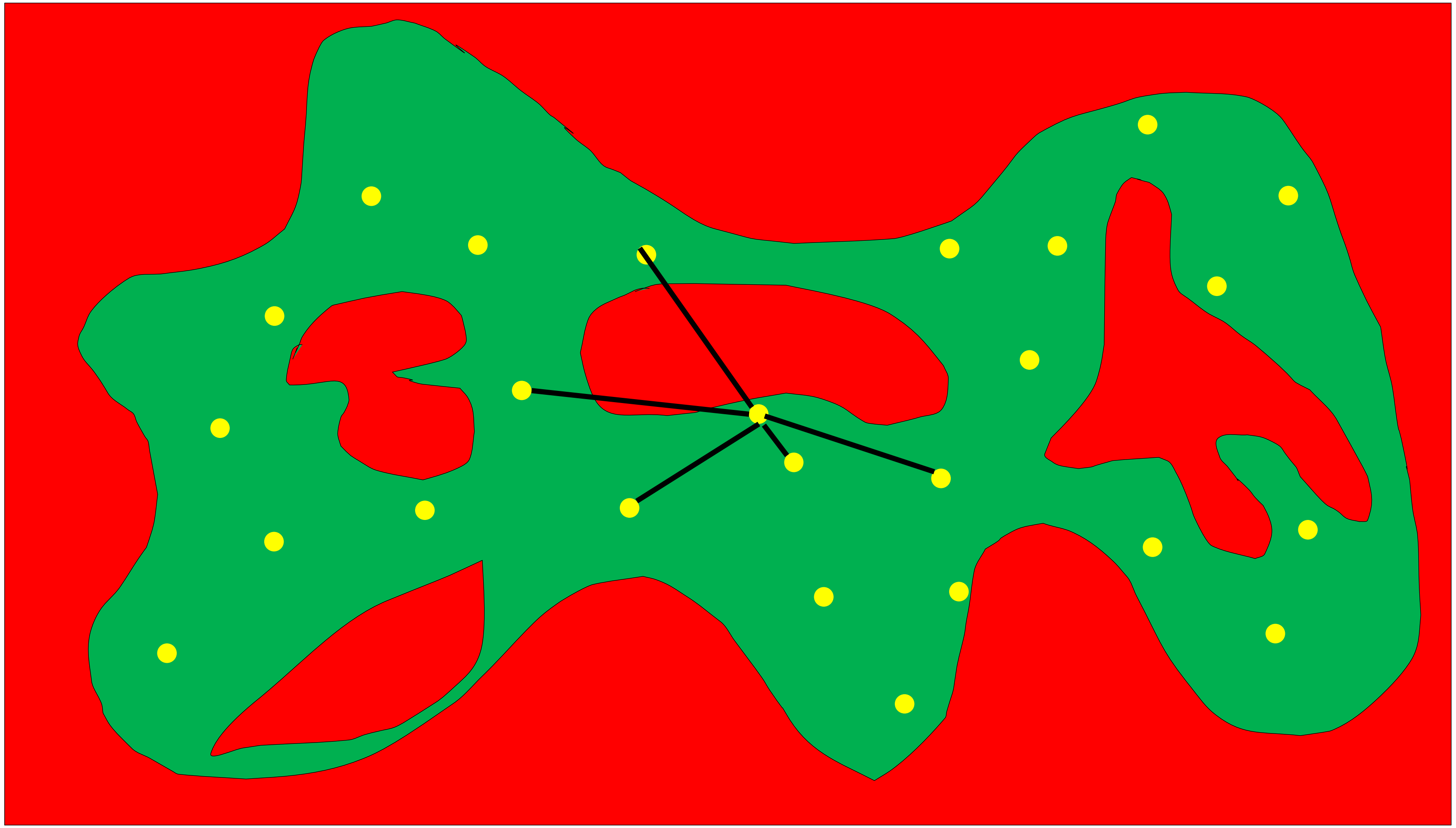
Probabilistic Roadmaps

- Link each collision free configuration to its nearest neighbors



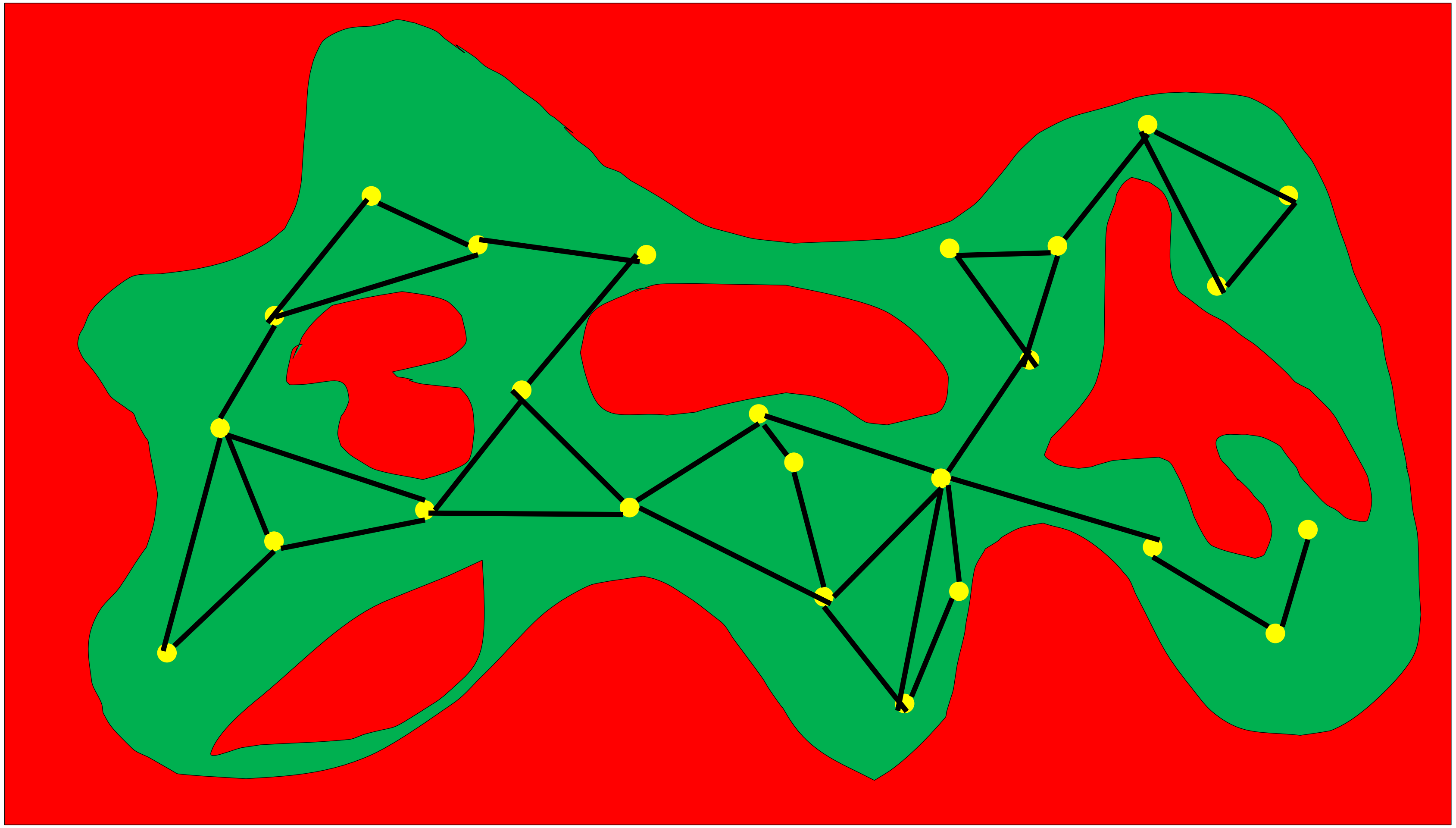
Probabilistic Roadmaps

- Link each collision free configuration to its nearest neighbours



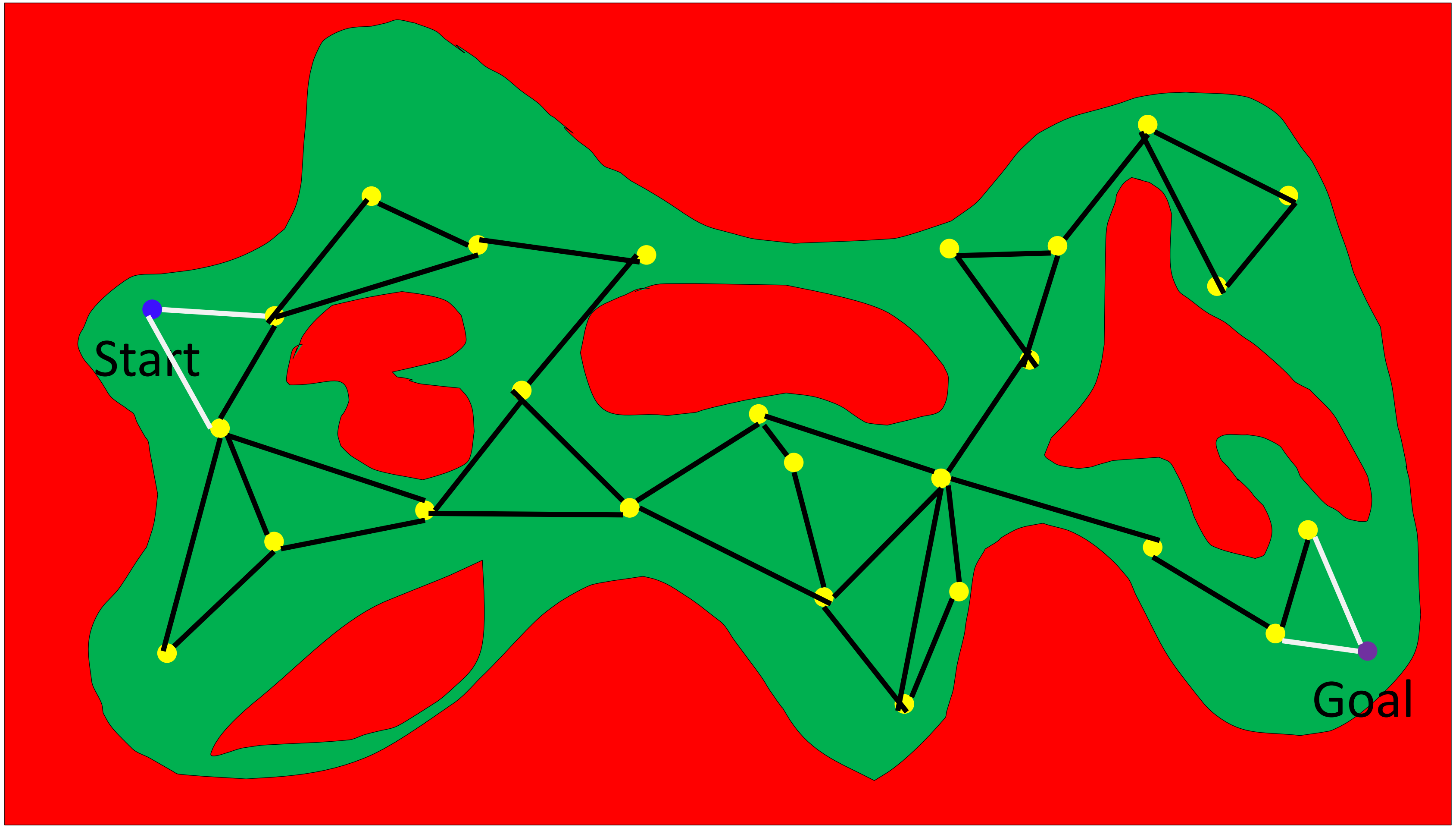
Probabilistic Roadmaps

- The collision free links are retained to form the PRM



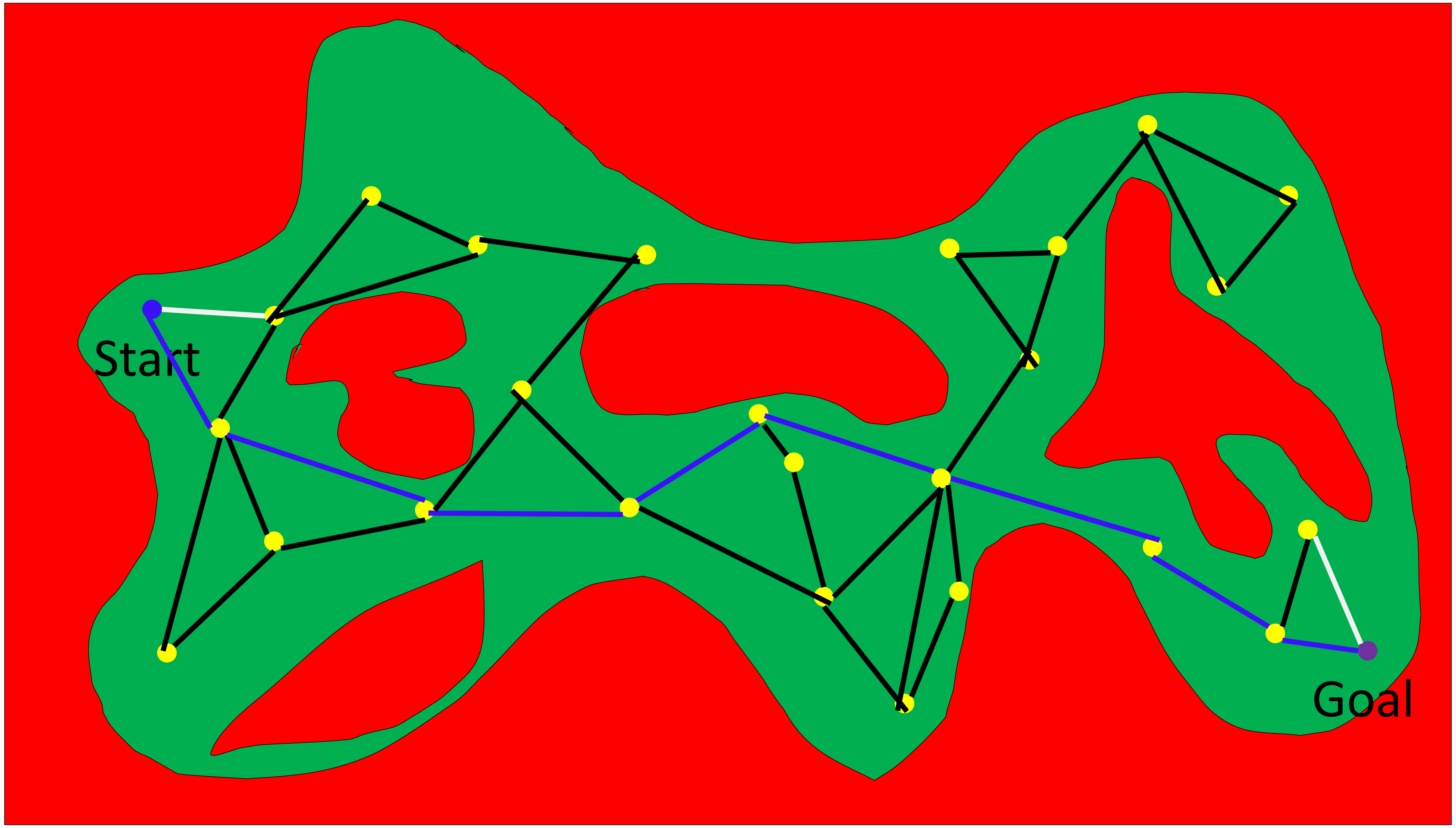
Probabilistic Roadmaps

- The collision free links are retained to form the PRM



Probabilistic Roadmaps

- The collision free links are retained to form the PRM



Probabilistic Roadmaps

- Initialize a set of points including X_s and X_g
- Randomly sample the points in C-space
  - How to sample in C-space uniformly or with some sort of bias based on some prior information
- Connect nearby points if there exists a collision free path between them
  - This is challenging and easy only for holonomic systems
  - Requires collision checking: computationally expensive
- Find a path from X_s to X_g in the graph

Probabilistic Roadmaps: Sampling

- Uniform sampling
  - sample uniformly at random from $[0,1]^n$
  - sample randomly from each parameter/DOF [0,1]

- Gaussian sampling
  - Generate sample X_1 uniformly at random
  - Generate another sample X_2 according to Gaussian distribution with mean X_1
  - If one of the X_1 or X_2 lies in C_{free} and other lies in C_{obs}, then the one in C_{free} is kept as a vertex in the roadmap

- Bridge-test and medial-axis sampling

Probabilistic Roadmaps: Selecting neighbours

- Nearest K
  - **K** closest points to $\mathbf{X}_i$, we need some distance metrics (how about Euclidean distance for a starter!)
- Component K
  - K nearest samples from each connected component of G
- Radius
  - Take all pints within radius ***r*** centred at X_i with an upper limit of K
  - Can change the radius adaptively as number samples increase
- Visibility
  - May be connect $\mathbf{X}_i$ to all visible vertices
  - Might be a bit expensive process
  - Basically visibility roadmap!!

Probabilistic Roadmaps

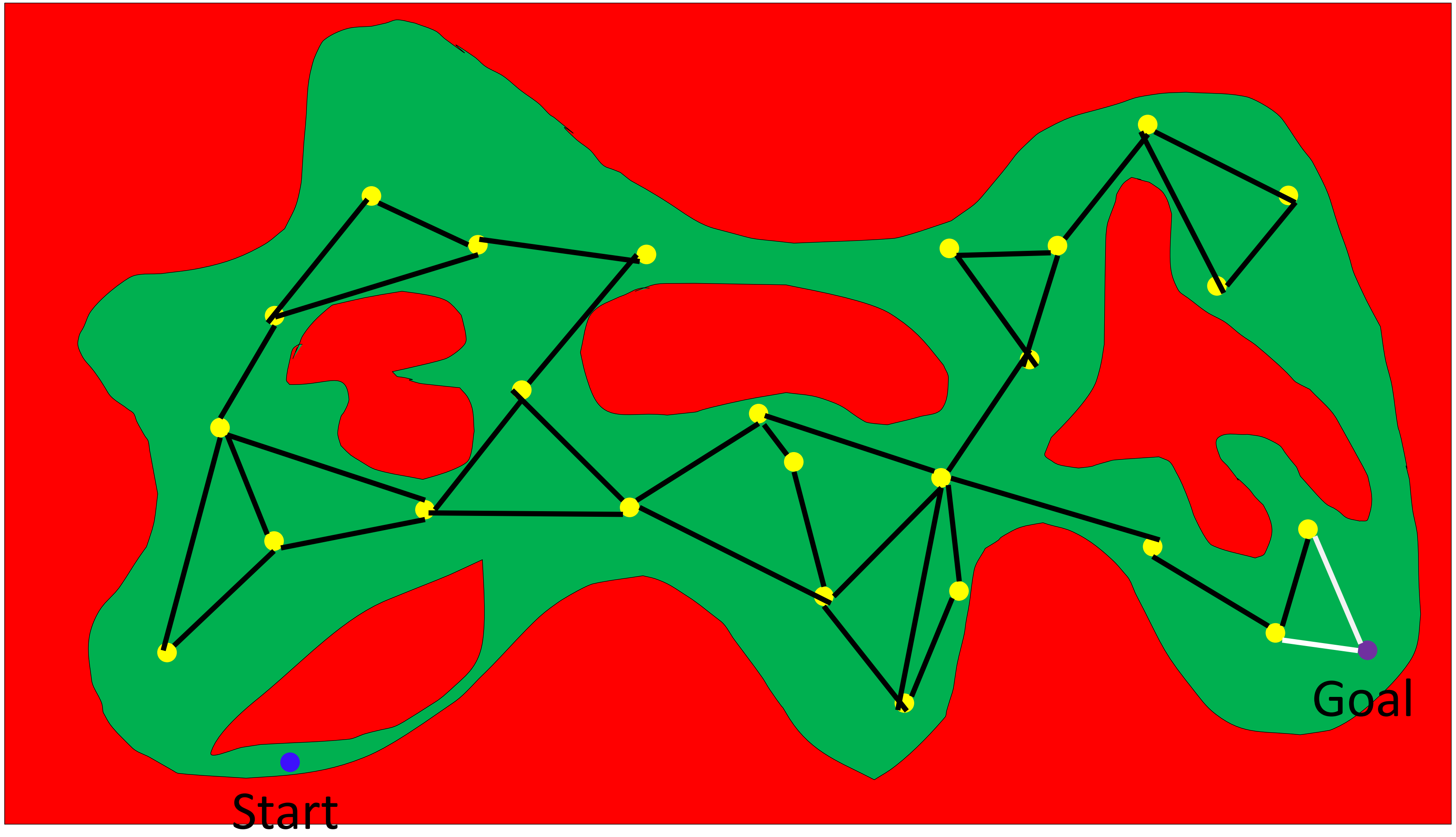
BUILD_ROADMAP

$\mathcal{G}$.init(); $i \leftarrow 0$;
while $i < N$
  if $\alpha(i) \in \mathcal{C}_{free}$ **then**
    $\mathcal{G}$.add_vertex($\alpha(i)$); $i \leftarrow i + 1$;
    for each $q \in$ NEIGHBORHOOD($\alpha(i)$,$\mathcal{G}$)
      if ((**not** $\mathcal{G}$.same_component($\alpha(i), q$)) **and** CONNECT($\alpha(i), q$)) **then**
        $\mathcal{G}$.add_edge($\alpha(i), q$);

Figure 5.25: The basic construction algorithm for sampling-based roadmaps. Note that i is not incremented if $\alpha(i)$ is in collision. This forces i to correctly count the number of vertices in the roadmap.

Probabilistic Roadmaps

- Check this start point!!

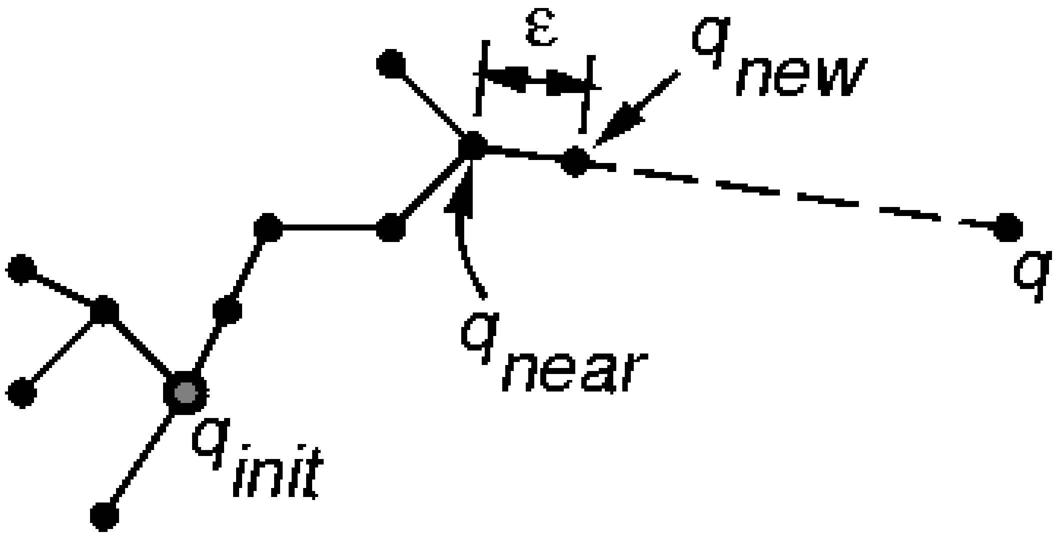


Probabilistic Roadmaps

- Pros
  - Probabistically complete
  - If we run it long enough the graph will contain a solution if one exists
  - Multiple queries possible after the graph is generated
- Cons
  - Build graph over C-space but there is no strategy for generating a path
  - Might not find a solution in narrow passages in a short period of time

Rapidly exploring Random Trees

- Build a tree through generating "next states" by executing some random controls



GENERATE_RRT(x_{init}, K, Δt)

$\mathcal{T}$.init(x_{init});
for $k = 1$ **to** K **do**
    $x_{rand} \leftarrow$ RANDOM_STATE();
    $x_{near} \leftarrow$ NEAREST_NEIGHBOR(x_{rand}, $\mathcal{T}$);
    $u \leftarrow$ SELECT_INPUT(x_{rand}, x_{near});
    $x_{new} \leftarrow$ NEW_STATE(x_{near}, u, Δt);
    $\mathcal{T}$.add_vertex(x_{new});
    $\mathcal{T}$.add_edge(x_{near}, x_{new}, u);
Return $\mathcal{T}$

Rapidly exploring Random Trees

- Its one of the most poplar algorithms since it was introduced by LaValle in 98
- Since then many researchers have come up with many many variants and extensions

Rapidly exploring Dense Trees (RDTs)

- Its one of the most poplar algorithms since it was introduced by LaVale in 98
- Since then many researchers have come up with many many variants and extensions

SIMPLE_RDT(q_0)

$\mathcal{G}$.init(q_0);
for $i = 1$ **to** k **do**
    $\mathcal{G}$.add_vertex($\alpha(i)$);
    $q_n \leftarrow$ NEAREST($S(\mathcal{G}), \alpha(i)$);
    $\mathcal{G}$.add_edge($q_n, \alpha(i)$);

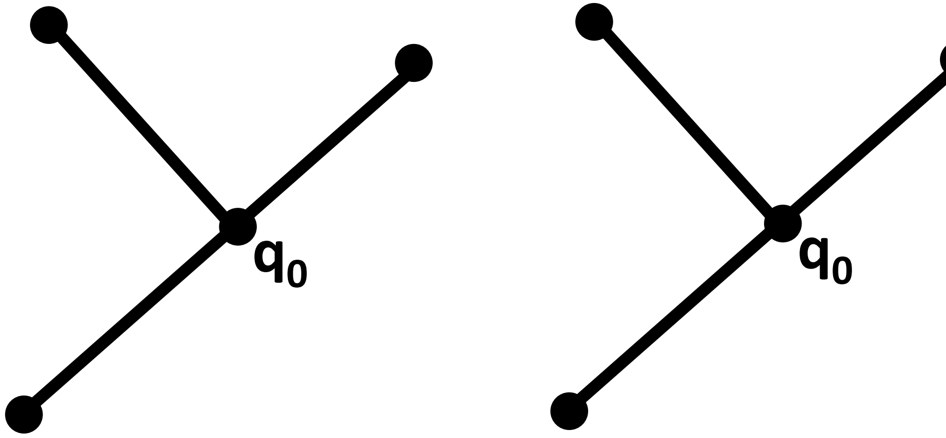
q_0: starting configuration
α(*i*): sampled configuration

Rapidly exploring Dense Trees (RDTs)

SIMPLE_RDT(q_0)

$\mathcal{G}$.init(q_0);
for $i = 1$ **to** k **do**
    $\mathcal{G}$.add_vertex($\alpha(i)$);
    $q_n \leftarrow$ NEAREST($S(\mathcal{G}), \alpha(i)$);
    $\mathcal{G}$.add_edge($q_n, \alpha(i)$);

q_0: starting configuration
$\alpha(i)$: sampled configuration



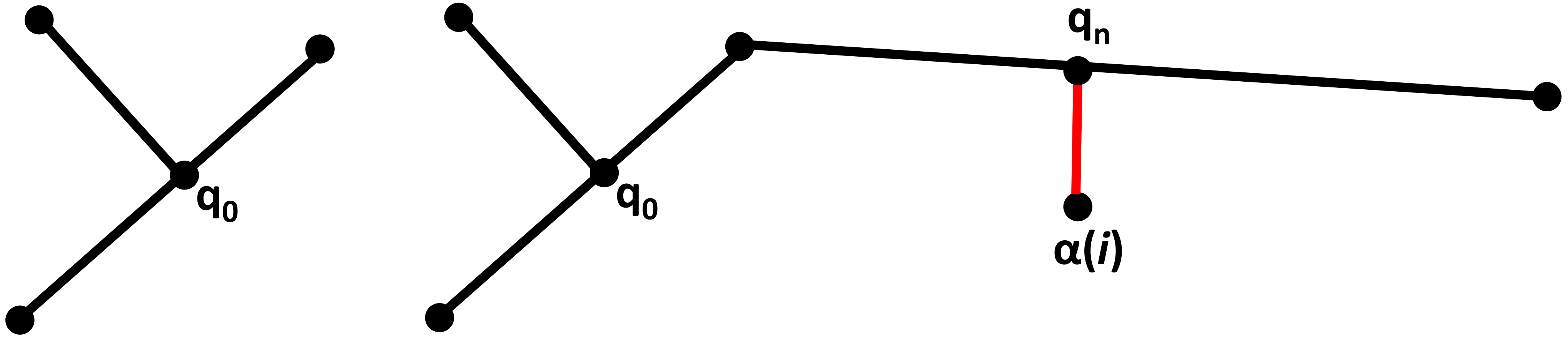
q_n: nearest configuration to the newly sampled configuration **$\alpha(i)$**
Connect them !!!

Rapidly exploring Dense Trees (RDTs)

$\text{SIMPLE\_RDT}(q_0)$

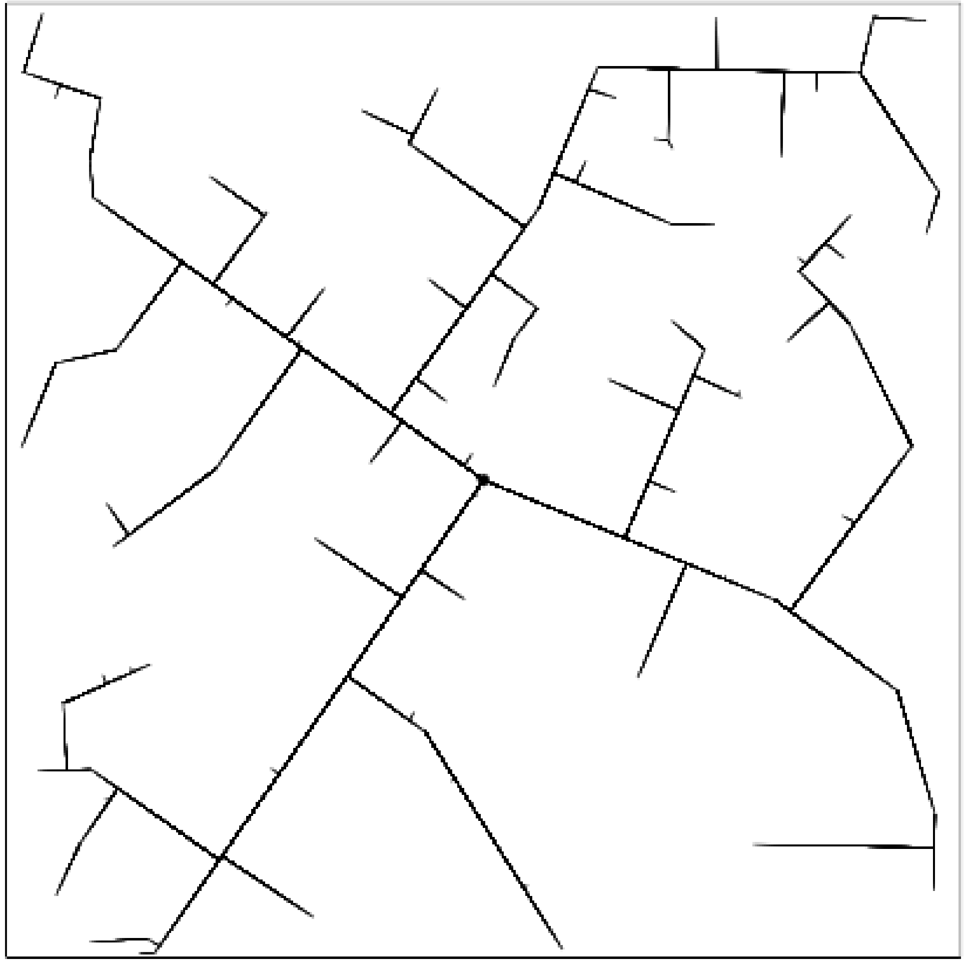
$\mathcal{G}.\text{init}(q_0);$
for $i = 1$ **to** k **do**
  $\mathcal{G}.\text{add\_vertex}(\alpha(i));$
  $q_n \leftarrow \text{NEAREST}(S(\mathcal{G}), \alpha(i));$
  $\mathcal{G}.\text{add\_edge}(q_n, \alpha(i));$

q_0: starting configuration
$\alpha(i)$: sampled configuration

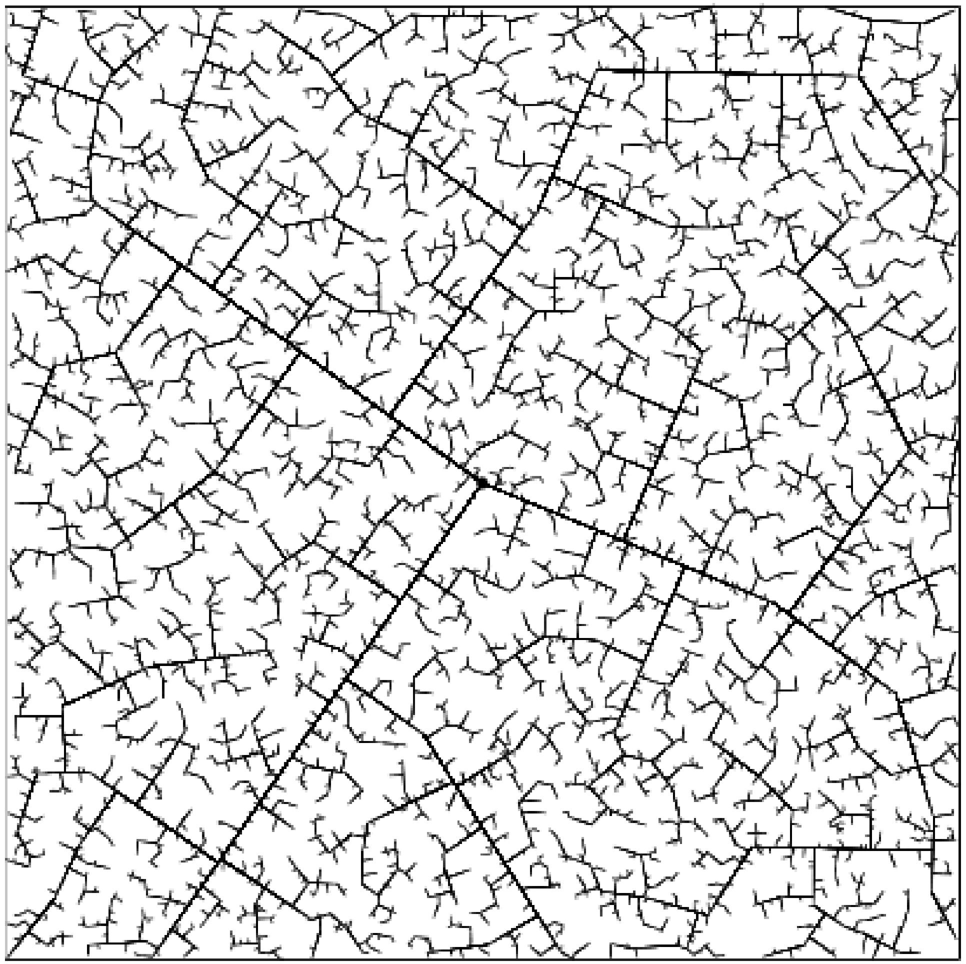


If nearest point/configuration lies on an existing edge then split the edge and add new vertex q_n

Rapidly exploring Dense Trees (RDTs)



45 iterations



2345 iterations

RDTs : with obstacles

RDT(q_0)

$\mathcal{G}$.init(q_0);
for $i = 1$ **to** k **do**
    $q_n \leftarrow$ NEAREST($S, \alpha(i)$);
    $q_s \leftarrow$ STOPPING-CONFIGURATION(q_n,$\alpha(i)$);
    if $q_s \neq q_n$ **then**
        $\mathcal{G}$.add_vertex(q_s);
        $\mathcal{G}$.add_edge(q_n, q_s);

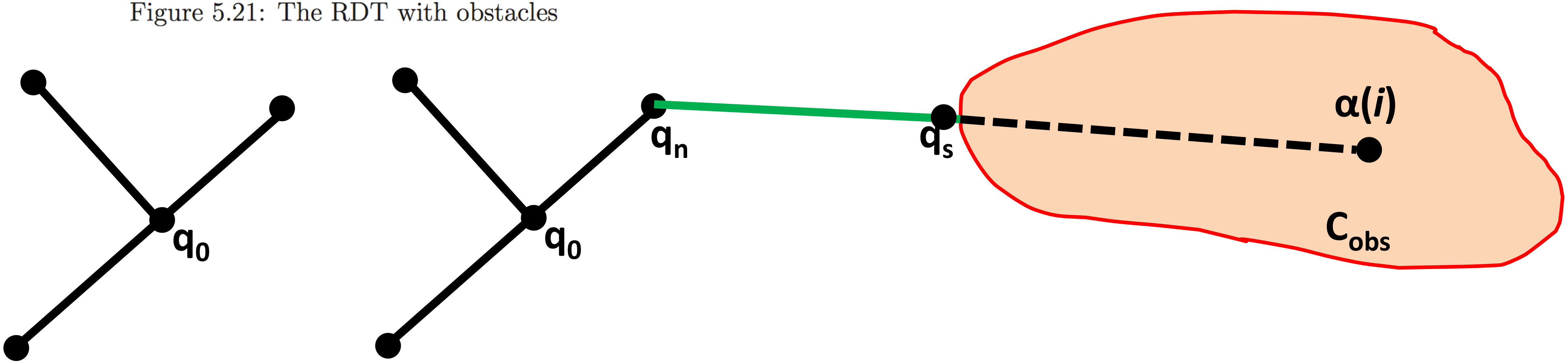
Figure 5.21: The RDT with obstacles

RDTs : with obstacles

RDT(q_0)
$\mathcal{G}$.init(q_0);
for $i = 1$ **to** k **do**
  $q_n \leftarrow$ NEAREST($S, \alpha(i)$);
  $q_s \leftarrow$ STOPPING-CONFIGURATION(q_n,$\alpha(i)$);
  if $q_s \neq q_n$ **then**
    $\mathcal{G}$.add_vertex(q_s);
    $\mathcal{G}$.add_edge(q_n, q_s);

Figure 5.21: The RDT with obstacles

q_0: starting configuration
$\alpha(i)$: sampled configuration
q_s: stopping configuration at the boundary of an obstacle



q_n: nearest configuration to the newly sampled configuration **$\alpha(i)$**

->If here is an obstacle, the edge travels up to the obstacle boundary, as far as allowed by the collision detection algorithm.

How to reach to a Goal??

- So far we have been building a tree from $\mathbf{q_0}$ **(could be the start point)!!**
- With some probability **p** sample goal configuration G and see if there is a nearest vertex in the existing tree that could connect to it
  - E.g every 200th sample would be Goal configuration

```
RDT(q0)
G.init(q0);
for i = 1 to k do
    qn ← NEAREST(S, α(i));
    qs ← STOPPING-CONFIGURATION(qn, α(i));
    if qs ≠ qn then
        G.add_vertex(qs);
        G.add_edge(qn, qs);
```

Figure 5.21: The RDT with obstacles

```
RDT(q0)
G.init(q0);
for i = 1 to k do
    If modulo(k, 200) == 0
        qn <- NEAREST(S, Goal)
    else
    qn ← NEAREST(S, α(i));
    qs ← STOPPING-CONFIGURATION(qn, α(i));
    if qs ≠ qn then
        G.add_vertex(qs);
        G.add_edge(qn, qs);
```

How to find nearest configuration?

- We are in C-Space : it may not be simple Euclidian space !!
- Every vertex is a specific configuration
- How to calculate the distance between two configurations?
  - We need some distance function
    - How about the absolute difference between each of the configuration parameters?
    - For articulated arms, project the configuration to the workspace and calculate cartesian distance?

Distance metrics

- Consider two configurations
  - $c_1(q_1, q_2, \ldots, q_n)$ and $c_2(q_1, q_2, \ldots, q_n)$
- What are the possible distance metrics?
  - Metrics in configuration space
    - Manhattan metric or difference or elementwise distance: $c_1 - c_2$
    - Euclidean metric or distance: $\mathrm{sqrt}(c_1^2 - c_2^2)$
  - Metrics in task space (for mobile robots and articulated arms/chain of rigid bodies)
    - p_1 = taskspace pose of c_1 and p_1 = taskance pose of c_2
    - Manhattan metric or difference or element-wise distance: $p_1 - p_2$
    - Euclidean metric or distance: $\mathrm{sqrt}(p_1^2 - p_2^2)$
- We will look at some common metric spaces for mobile robots: remember it is not easy to calculate the distance between rotations!!

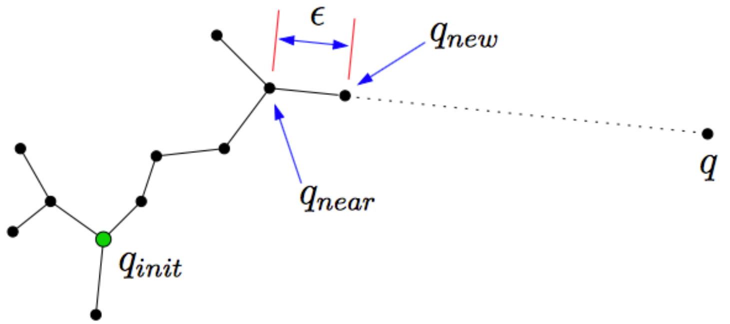
RRT: A particular case of RDTs!

- Don’t extend all the way to the newly sampled configuration !
- Rather walk/move just a small step towards that randomly sampled configuration
- How to take that small step?
  - Depends on how a robot can take that small step !!!
  - We will discuss this for a few special cases

```
BUILD_RRT(q_init)
  T.init(q_init);
  for k = 1 to K do
      q_rand ← RANDOM_CONFIG();
      EXTEND(T, q_rand);
  Return T
```

```
EXTEND(T, q)
  q_near ← NEAREST_NEIGHBOR(q, T);
  if NEW_CONFIG(q, q_near, q_new) then
      T.add_vertex(q_new);
      T.add_edge(q_near, q_new);
      if q_new = q then
          Return Reached;
      else
          Return Advanced;
  Return Trapped;
```

Figure 2: The basic RRT construction algorithm.



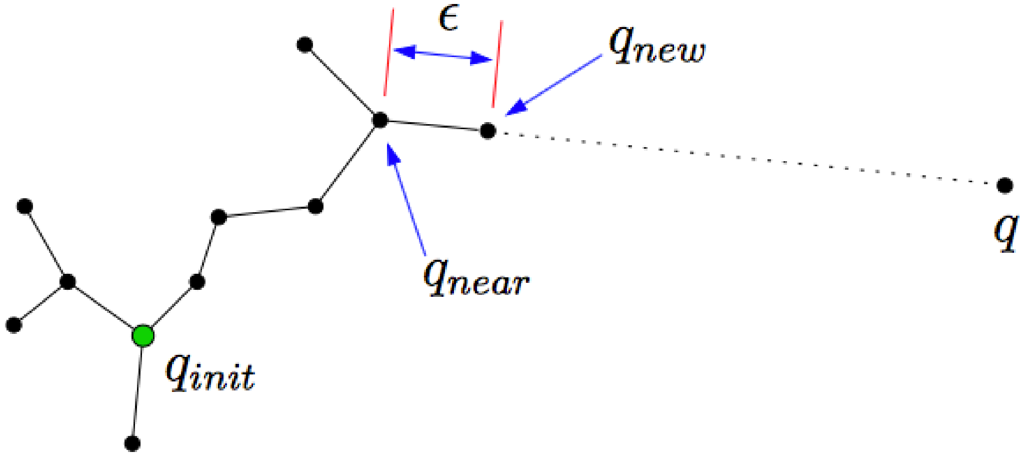
RRT: A particular case of RDTs!

BUILD_RRT(q_{init})

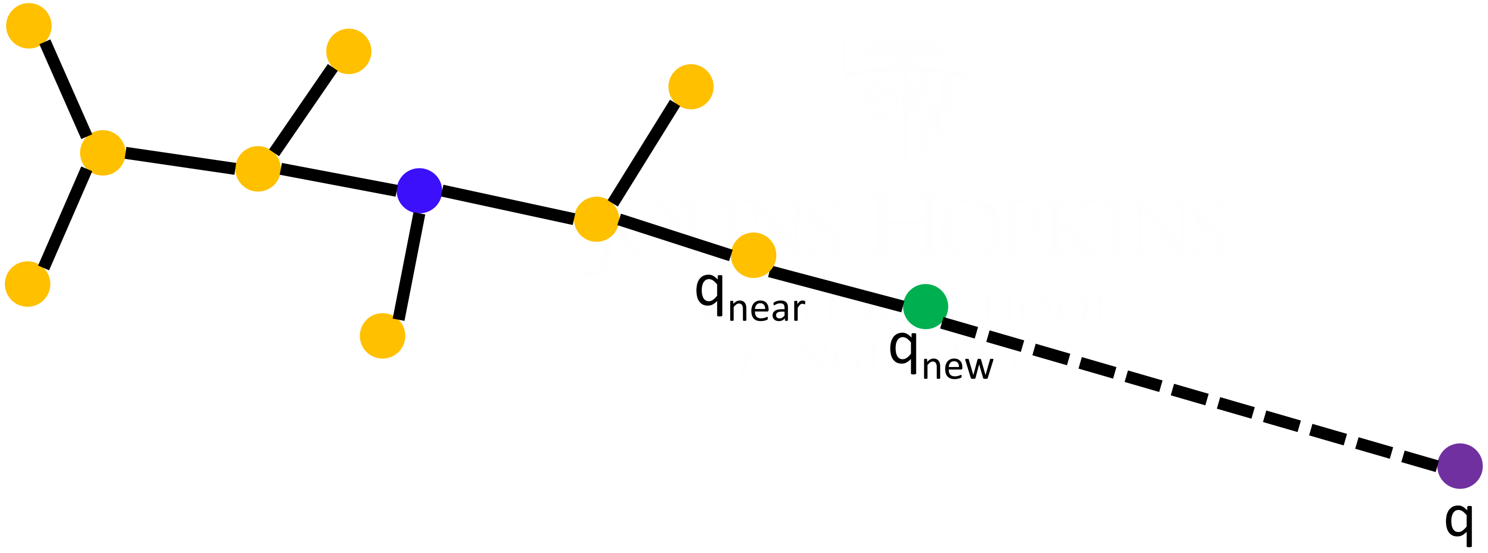
$\mathcal{T}$.init(q_{init});
for $k = 1$ **to** K **do**
    $q_{rand} \leftarrow$ RANDOM_CONFIG();
    EXTEND($\mathcal{T}, q_{rand}$);
Return $\mathcal{T}$

EXTEND($\mathcal{T}, q$)

$q_{near} \leftarrow$ NEAREST_NEIGHBOR($q, \mathcal{T}$);
if NEW_CONFIG(q, q_{near}, q_{new}) **then**
    $\mathcal{T}$.add_vertex(q_{new});
    $\mathcal{T}$.add_edge(q_{near}, q_{new});
    if $q_{new} = q$ **then**
        Return *Reached*;
    else
        Return *Advanced*;
Return *Trapped*;



RRT: EXTEND()

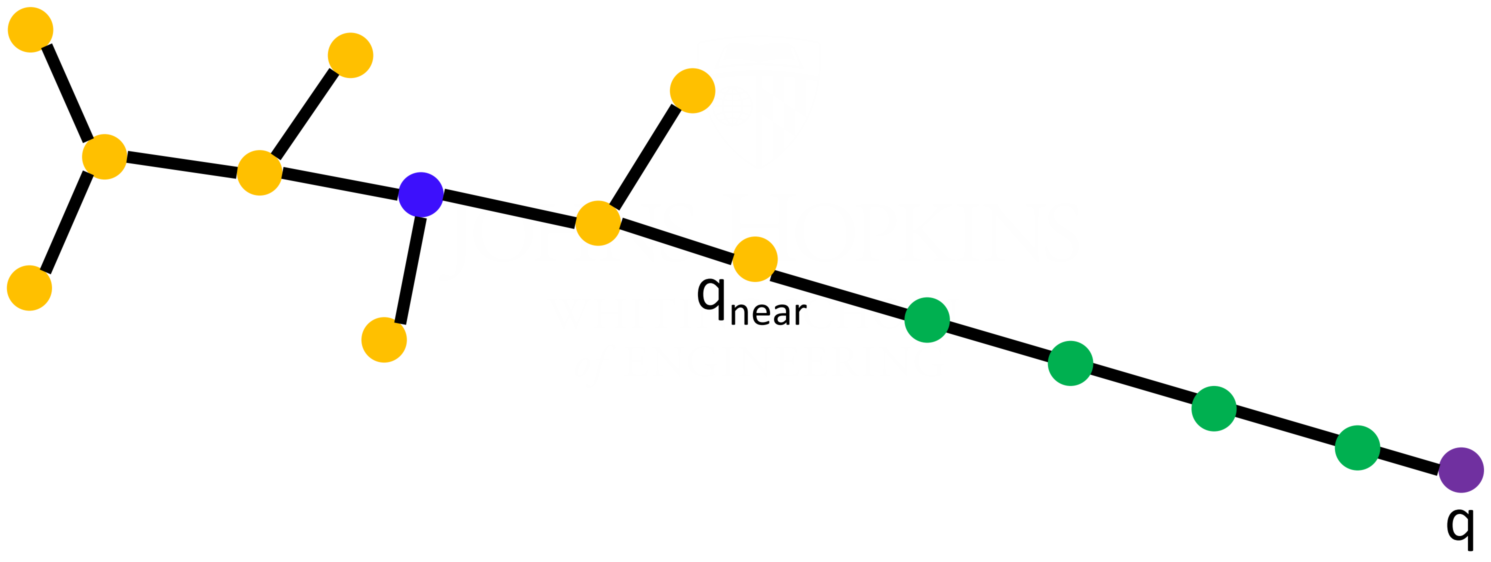


- Take only 1 step (step size is a parameter for this subroutine)

EXTEND($\mathcal{T}, q$)

$q_{near} \leftarrow$ NEAREST_NEIGHBOR($q, \mathcal{T}$);
if NEW_CONFIG(q, q_{near}, q_{new}) **then**
    $\mathcal{T}$.add_vertex(q_{new});
    $\mathcal{T}$.add_edge(q_{near}, q_{new});
    if $q_{new} = q$ **then**
        Return *Reached*;
    else
        Return *Advanced*;
Return *Trapped*;

RRT: CONNECT()



- Step towards q_{random} until
  - It reaches the q_{random}
  - Hit and obstacle
- How is this different from RDT???

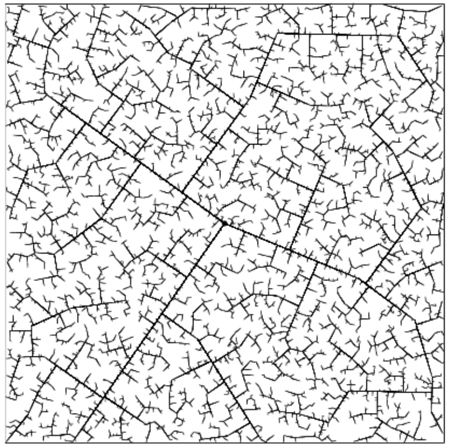
EXTEND($\mathcal{T}, q$)

$q_{near} \leftarrow$ NEAREST_NEIGHBOR($q, \mathcal{T}$);
if NEW_CONFIG(q, q_{near}, q_{new}) **then**
    $\mathcal{T}$.add_vertex(q_{new});
    $\mathcal{T}$.add_edge(q_{near}, q_{new});
    if $q_{new} = q$ **then**
        Return *Reached*;
    else
        Return *Advanced*;
Return *Trapped*;

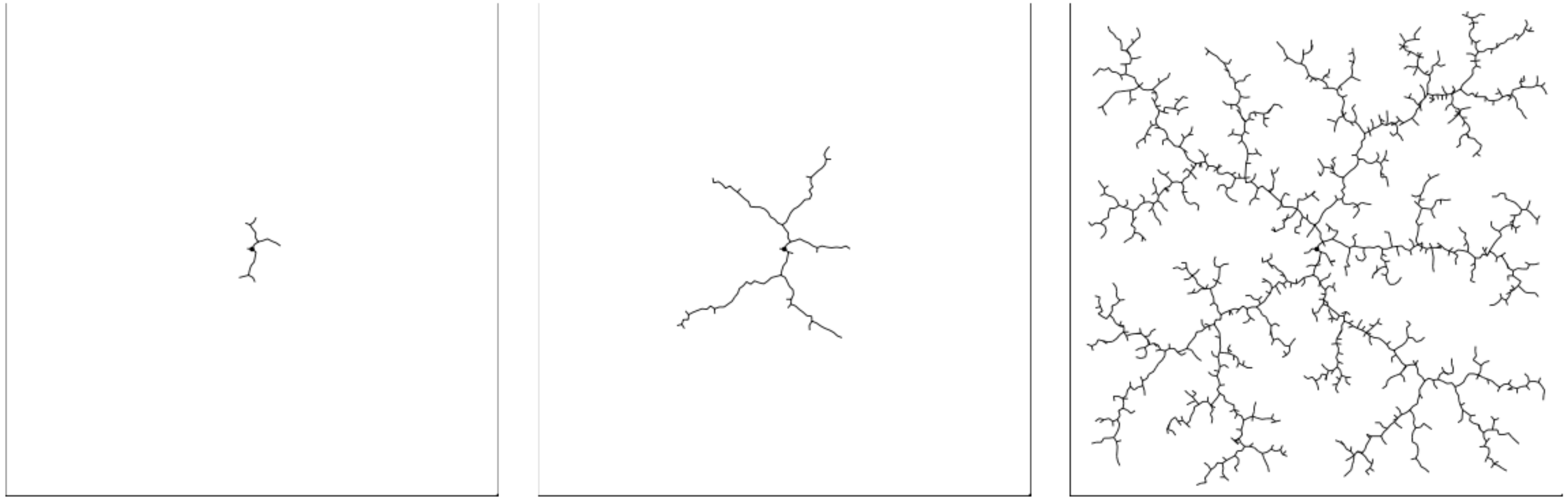
Tree would grow in a different manner!

- Observations !!!

45 iterations

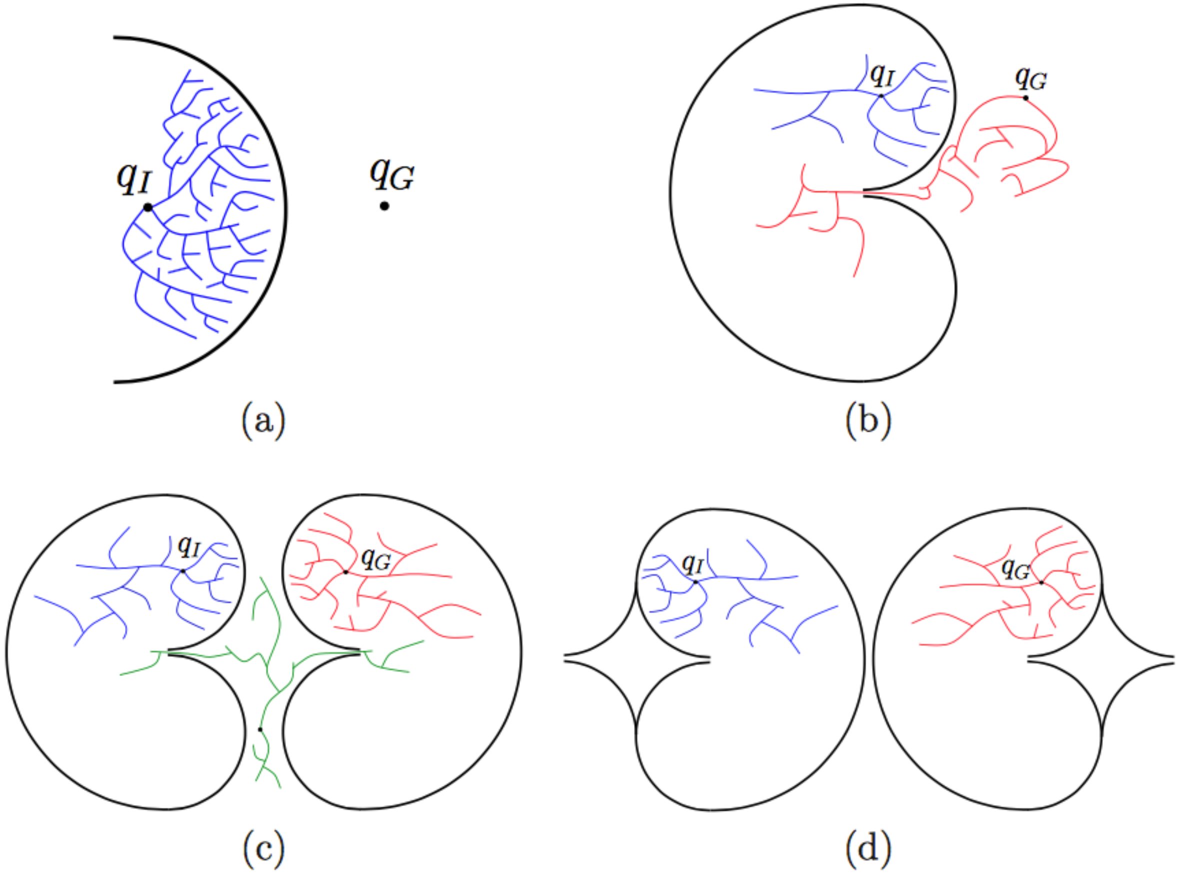


2345 iterations

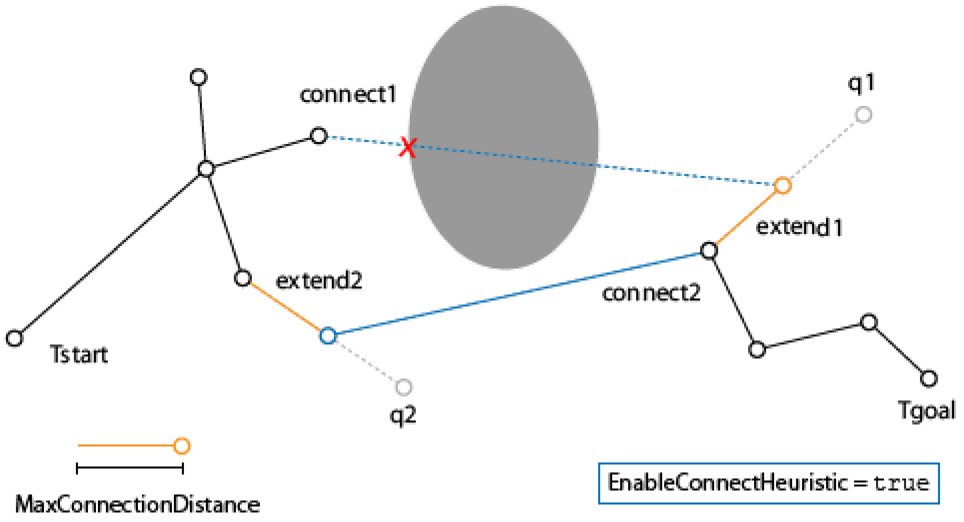


What could go wrong !!!

- Algorithm could get trapped !!



Solution: Bi-directional RRT



RDT_BALANCED_BIDIRECTIONAL(q_I, q_G)

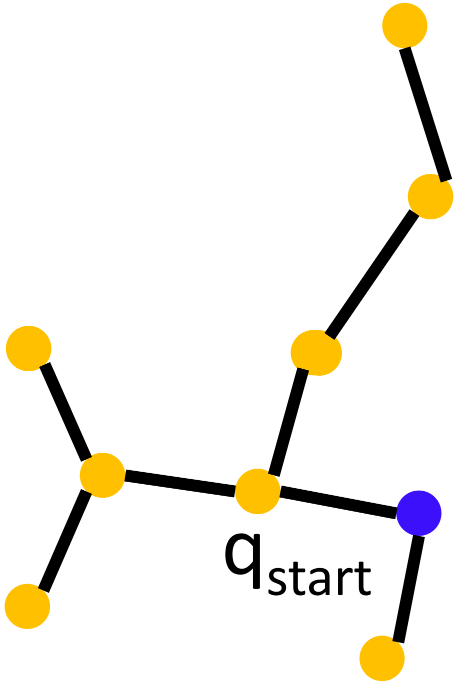
T_a.init(q_I); T_b.init(q_G);
for $i = 1$ **to** K **do**
    $q_n \leftarrow$ NEAREST($S_a, \alpha(i)$);
    $q_s \leftarrow$ STOPPING-CONFIGURATION(q_n,$\alpha(i)$);
    if $q_s \neq q_n$ **then**
        T_a.add_vertex(q_s);
        T_a.add_edge(q_n, q_s);
        $q'_n \leftarrow$ NEAREST(S_b, q_s);
        $q'_s \leftarrow$ STOPPING-CONFIGURATION(q'_n,q_s);
        if $q'_s \neq q'_n$ **then**
            T_b.add_vertex(q'_s);
            T_b.add_edge(q'_n, q'_s);
        if $q'_s = q_s$ **then return** SOLUTION;
    if $|T_b| > |T_a|$ **then** SWAP(T_a, T_b);
return FAILURE

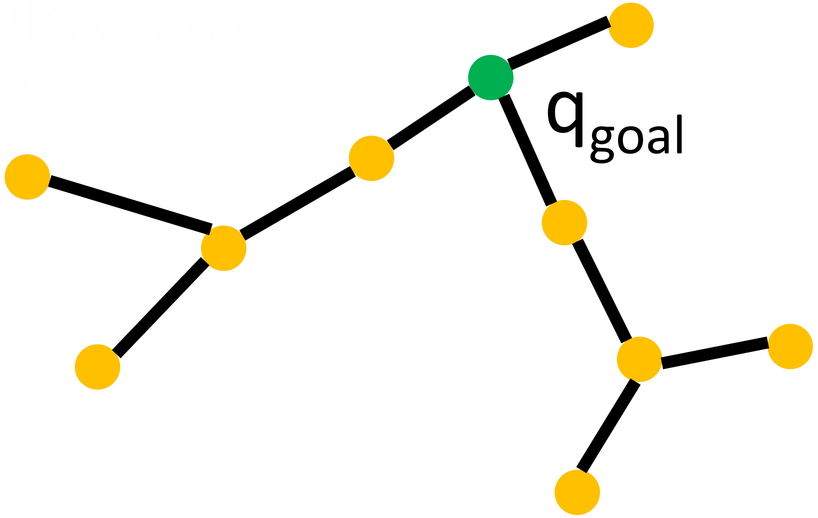
Solution: Bi-directional RRT

RDT_BALANCED_BIDIRECTIONAL(q_I, q_G)

T_a.init(q_I); T_b.init(q_G);
for $i = 1$ **to** K **do**
    $q_n \leftarrow$ NEAREST$(S_a, \alpha(i))$;
    $q_s \leftarrow$ STOPPING-CONFIGURATION$(q_n, \alpha(i))$;
    if $q_s \neq q_n$ **then**
        T_a.add_vertex(q_s);
        T_a.add_edge(q_n, q_s);
        $q'_n \leftarrow$ NEAREST(S_b, q_s);
        $q'_s \leftarrow$ STOPPING-CONFIGURATION(q'_n, q_s);
        if $q'_s \neq q'_n$ **then**
            T_b.add_vertex(q'_s);
            T_b.add_edge(q'_n, q'_s);
        if $q'_s = q_s$ **then return** SOLUTION;
    if $|T_b| > |T_a|$ **then** SWAP(T_a, T_b);
return FAILURE

Solution: Bi-directional RRT

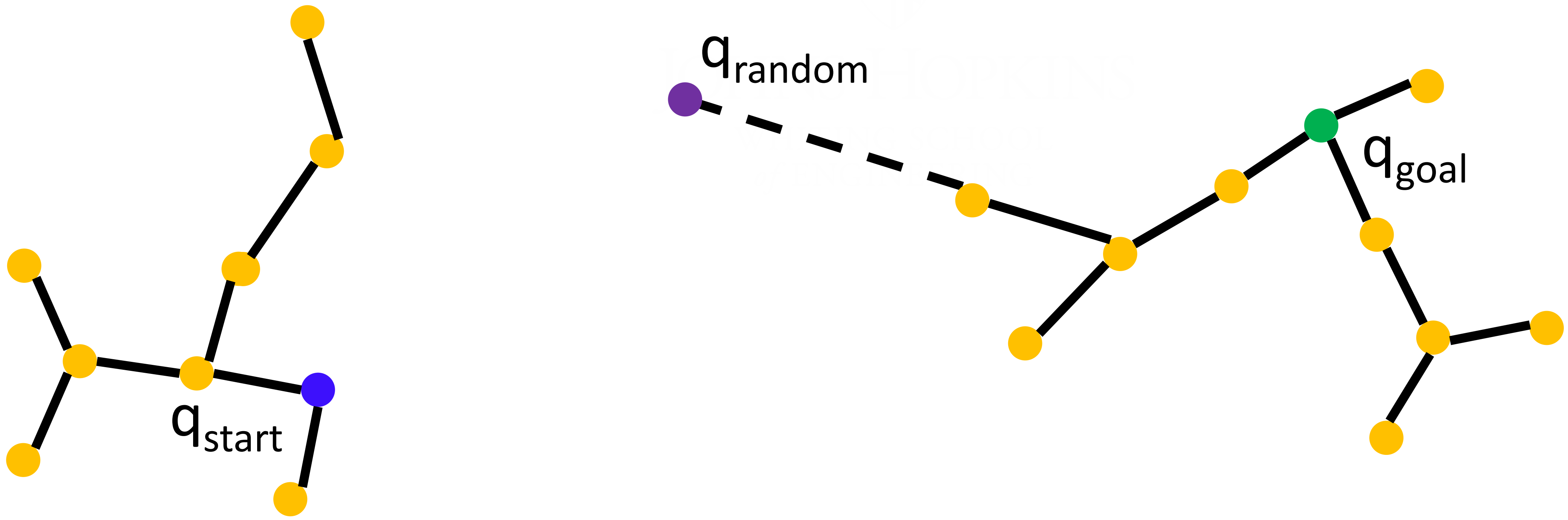




Connect

Extend

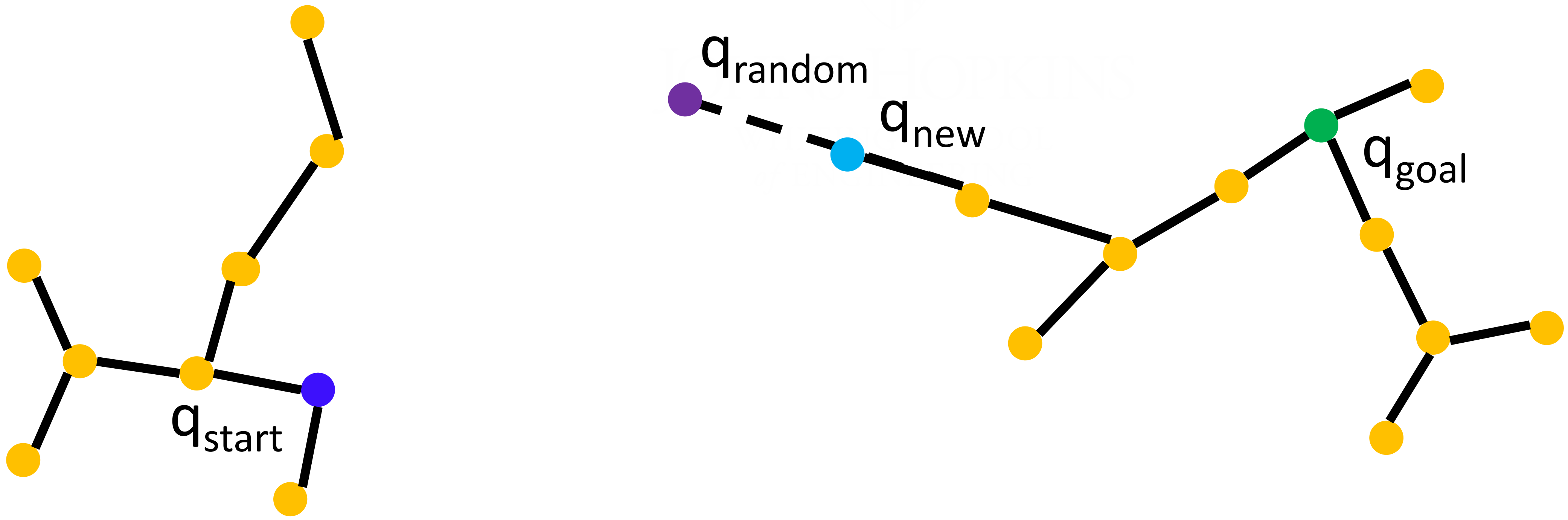
Solution: Bi-directional RRT



Connect

Extend

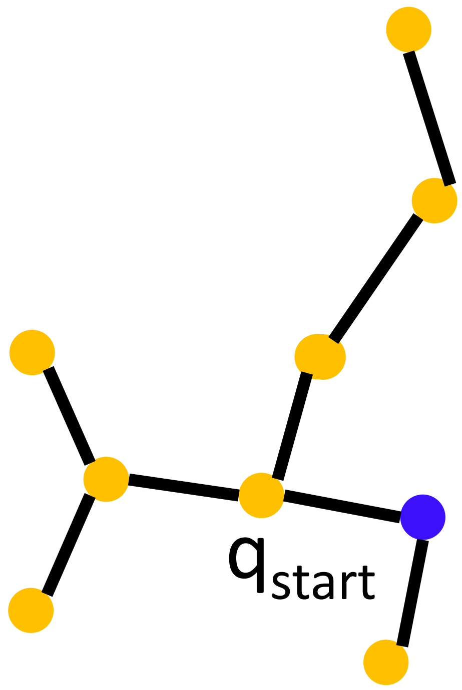
Solution: Bi-directional RRT

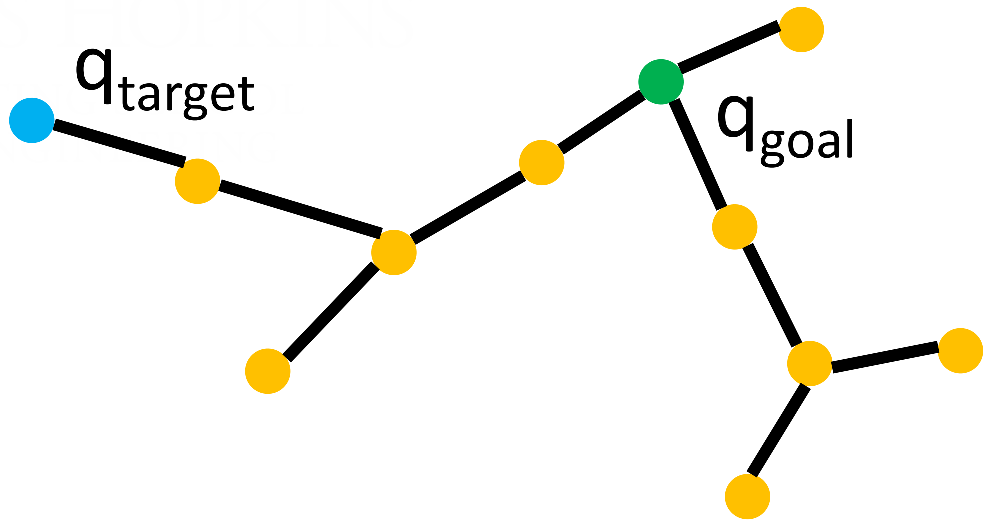


Connect

Extend

Solution: Bi-directional RRT

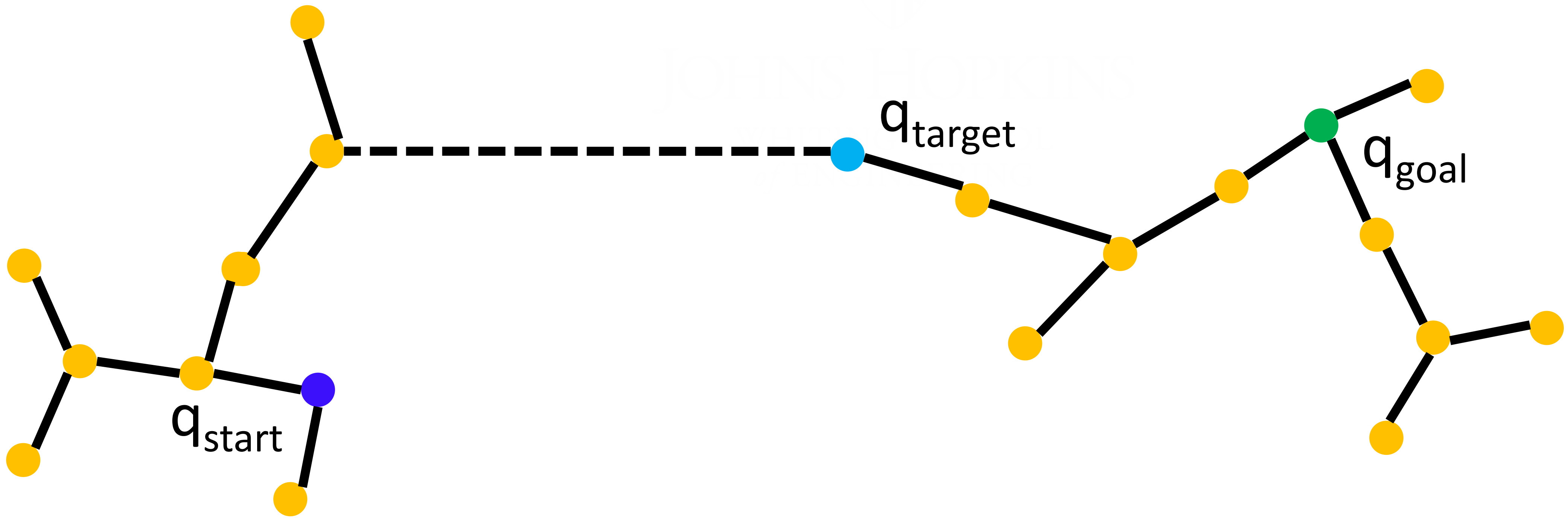




Connect

Extend

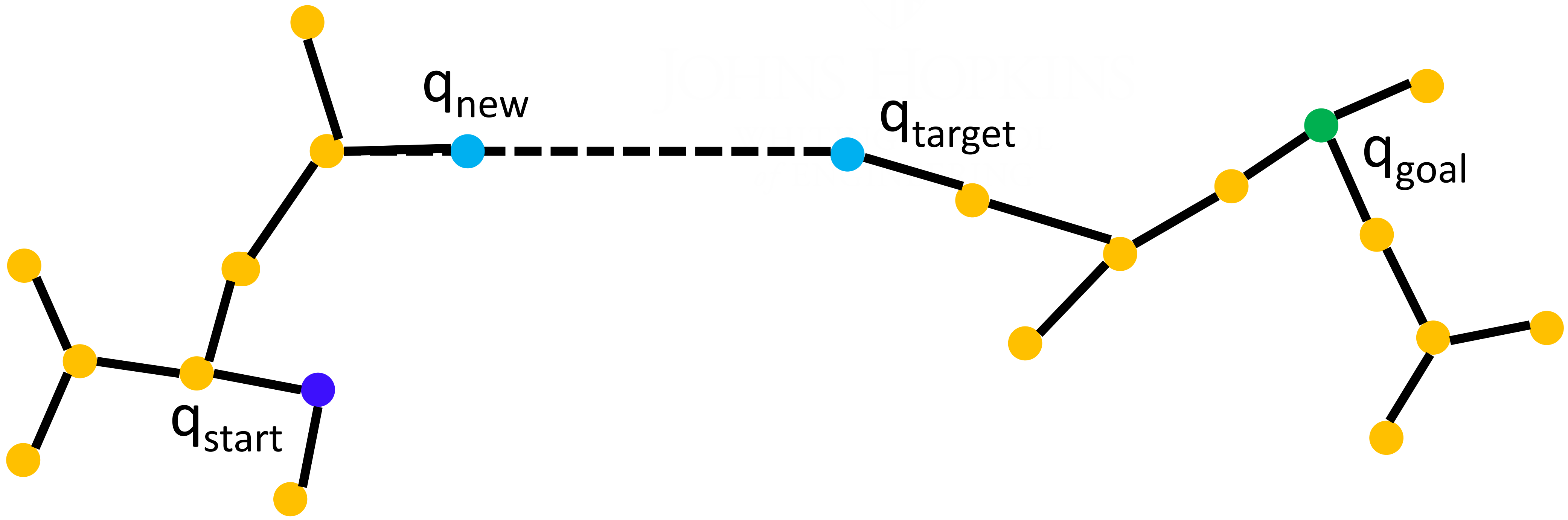
Solution: Bi-directional RRT



Connect

Extend

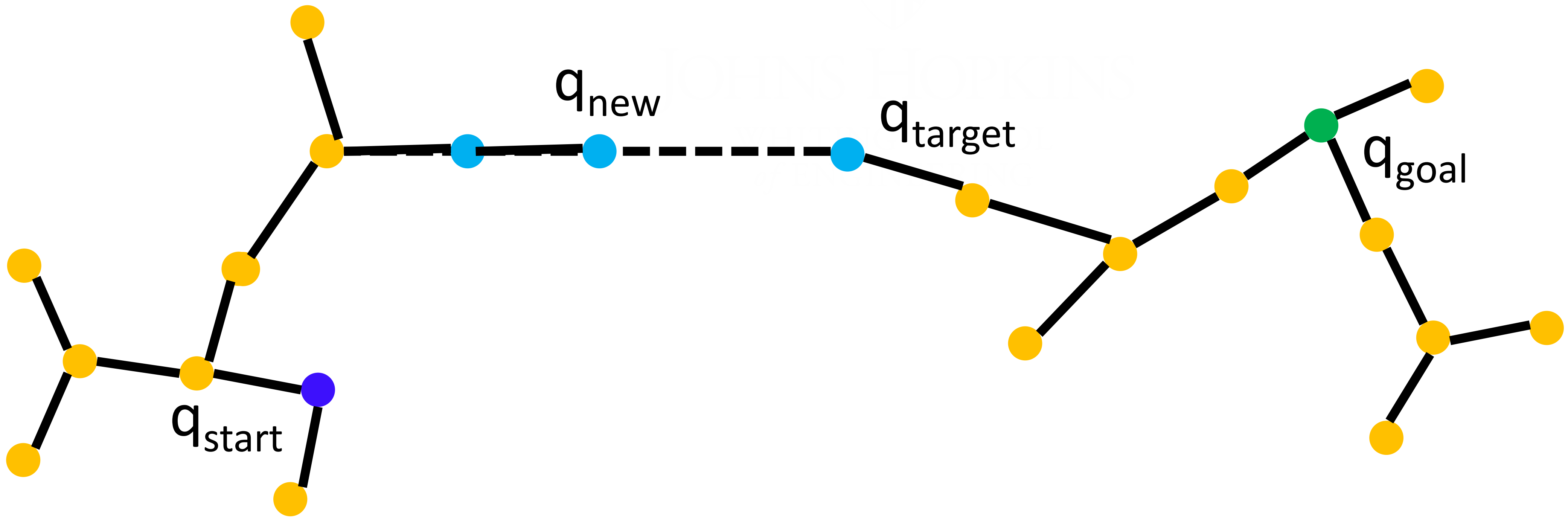
Solution: Bi-directional RRT



Connect

Extend

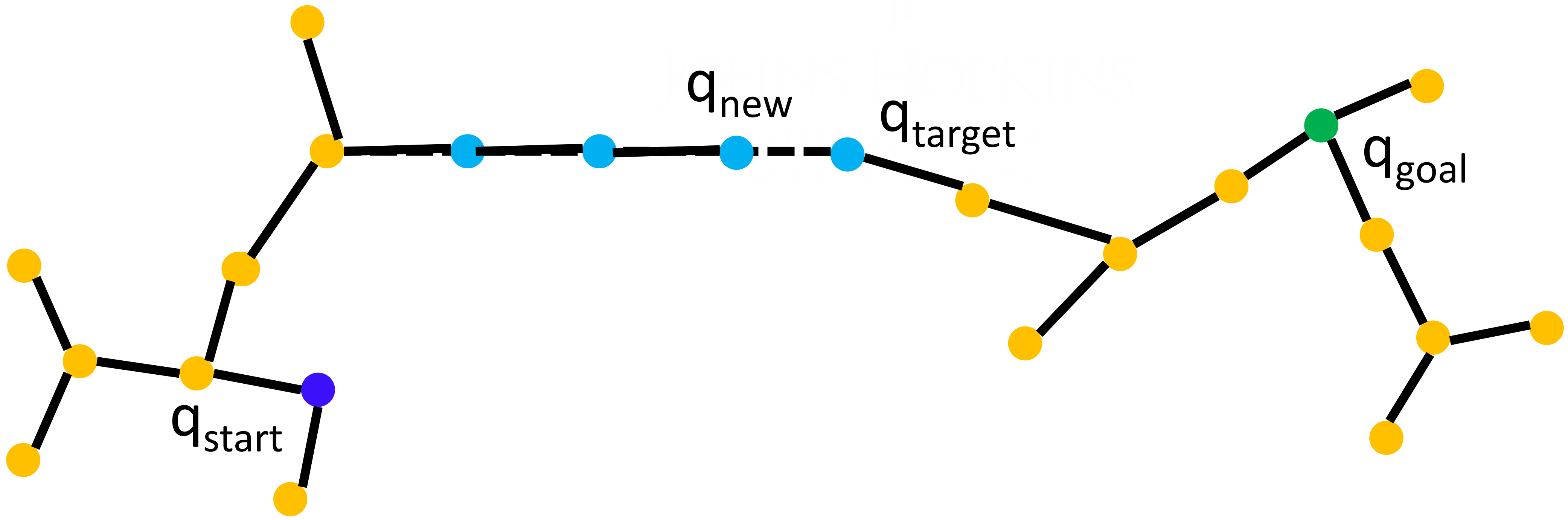
Solution: Bi-directional RRT



Connect

Extend

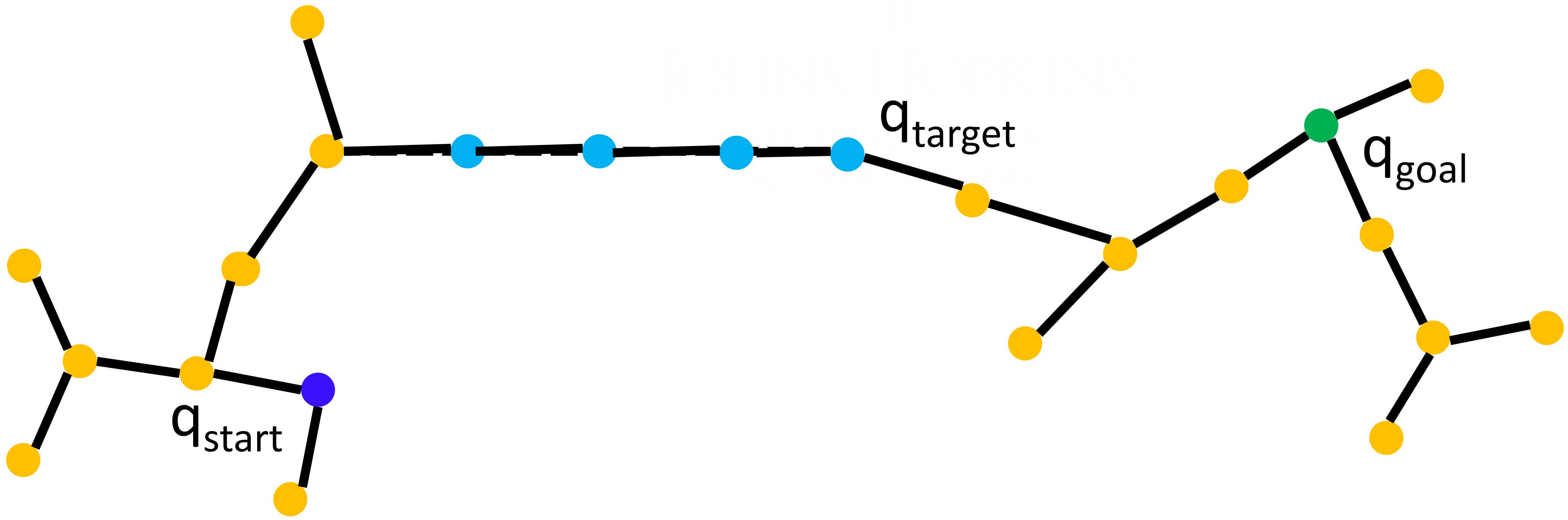
Solution: Bi-directional RRT



Connect

Extend

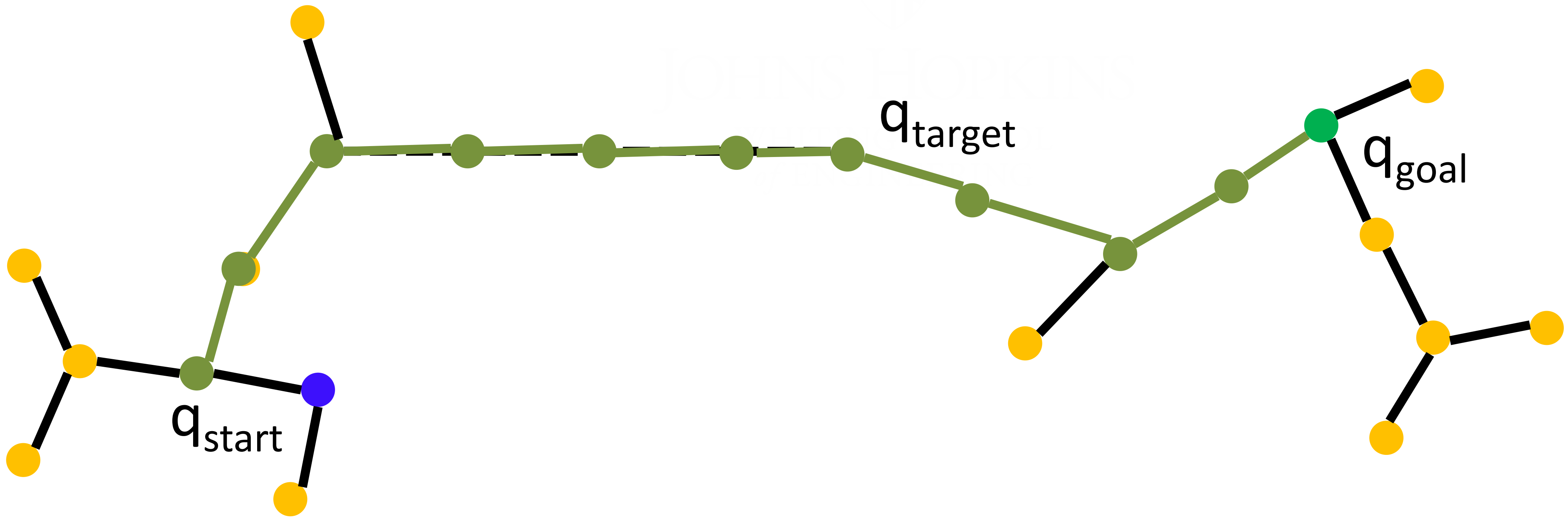
Solution: Bi-directional RRT



Connect

Extend

Solution: Bi-directional RRT



Connect

Extend

Credits

- Some of the figures are adapted from the textbook: Planning Algorithms by Steven M. LaValle
- Some of the slides are adapted from lecture notes by Pratap Tokekar, ECE 4984/5984: (Advanced) Robot Motion Planning, Virginia tech
- Some of the slides are adopted from Pieter Abbeel’s lectures