Introduction to motion planning, RRT Connect and PRM

Robotics and Computer vision (RoVi)

Aljaz Kramberger

alk@mmmi.sdu.dk

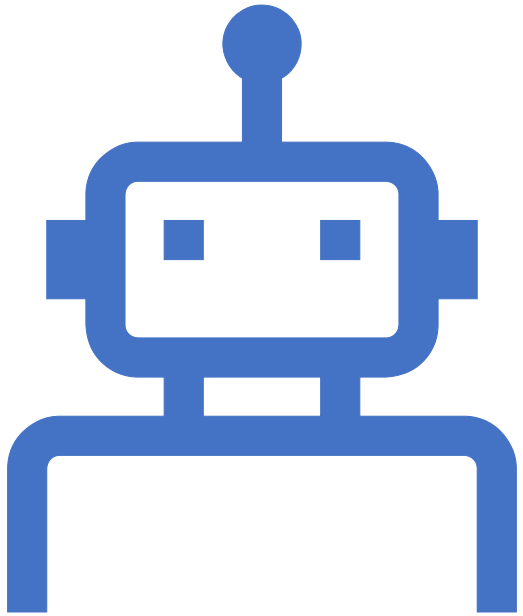
SDU Robotics

The Maersk Mc-Kinney Moller Institute

University of Southern Denmark

Based on slides of Christian Schlette and Christoffer Sloth

Siciliano, Bruno, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. Robotics: modelling, planning and control. Springer Science & Business Media, 2010.



Agenda

- Introduction to motion planning
- RRT
- RRT-Connect
- A*
- RRT*
- Probabilistic roadmaps

Introduction to motion planning

Terminology - Basics

- **Cartesian coordinates** – an abstract description of how something is located in space. The Cartesian coordinate system (aka Cartesian space or workspace) uses 6 dimensions (or less, if describing **subspaces**, e.g. planes or lines).
- **Configuration** – an abstract description of how a kinematic system is laid out in space, e.g. by joint vectors for manipulators or position/orientation vectors for mobile platforms.
- **Configuration space** – the space $\mathcal{C}$ (C-space) defined by all possible configurations of a kinematic system, with N dimensions for kinematics with N independently controllable joints.
- **Free space** – the subspace $\mathcal{C}_{free}$ of configuration space that encompasses all configurations of a kinematic system which do not yield collisions with **obstacles** (typically objects in Cartesian space).
- **Goal** – a desired, collision-free target configuration for a kinematic system.

Motion Planning Types

- **Path planning versus motion planning:** The path planning problem is a subproblem of the general motion planning problem. Path planning is a purely geometric problem of finding a collision-free path $q(s);\ s\ \in\ [0;\ 1]$, from a start configuration $q(0)\ =\ q_{start}$ to a goal configuration $q(1)\ =\ q_{goal}$ without concern for the dynamics, the duration of motion, or constraints on the motion or on the control inputs. It is assumed that the path returned by the path planner can be time scaled to create a feasible trajectory.
- **Control inputs:** m = n versus $m\ <\ n$. If there are fewer control inputs m than degrees of freedom n, then the robot is incapable of following many paths, even if they are collision-free. *For example, a car has n = 3 (the position and orientation of the chassis in the plane) but m = 2 (forward - backward motion and steering); it cannot slide directly sideways into a parking space.*

Motion Planning Types

- **Online versus off-online:** A motion planning problem requiring an immediate result, perhaps because dynamic obstacles, desires for a fast, online, planner. If the environment is static then a slower off-online planner may suffice.
- **Optimal vs. satisfying:** Time optimality can be an issue.
- **Exact vs. approximate:** an approximate solution in the vicinity of the goals might be good enough.
- **With or without obstacles:** The motion planning problem can be challenging even in the absence of obstacles, particularly if m < n or optimality is desired.

Properties of motion planners

- **Multiple-query versus single-query planning:** If the robot is being asked to solve several motion planning problems in an unchanging environment, it may be worth spending the time building a data structure that accurately represents C_{free}. This data structure can then be searched to solve multiple planning queries efficiently. Single-query planners solve each new problem from scratch.
- **Anytime planning:** An anytime planner is one that continues to look for better solutions after a first solution is found. The planner can be stopped at any time, for example when a specified time limit has passed, and the best solution returned.
- **Completeness:** A motion planner is said to be *complete* if it is guaranteed to find a solution in finite time if one exists, and to report failure if there is no feasible motion plan. A planner is *probabilistically complete* if the probability of finding a solution, if one exists, tends to 1 as the planning time goes to infinity.

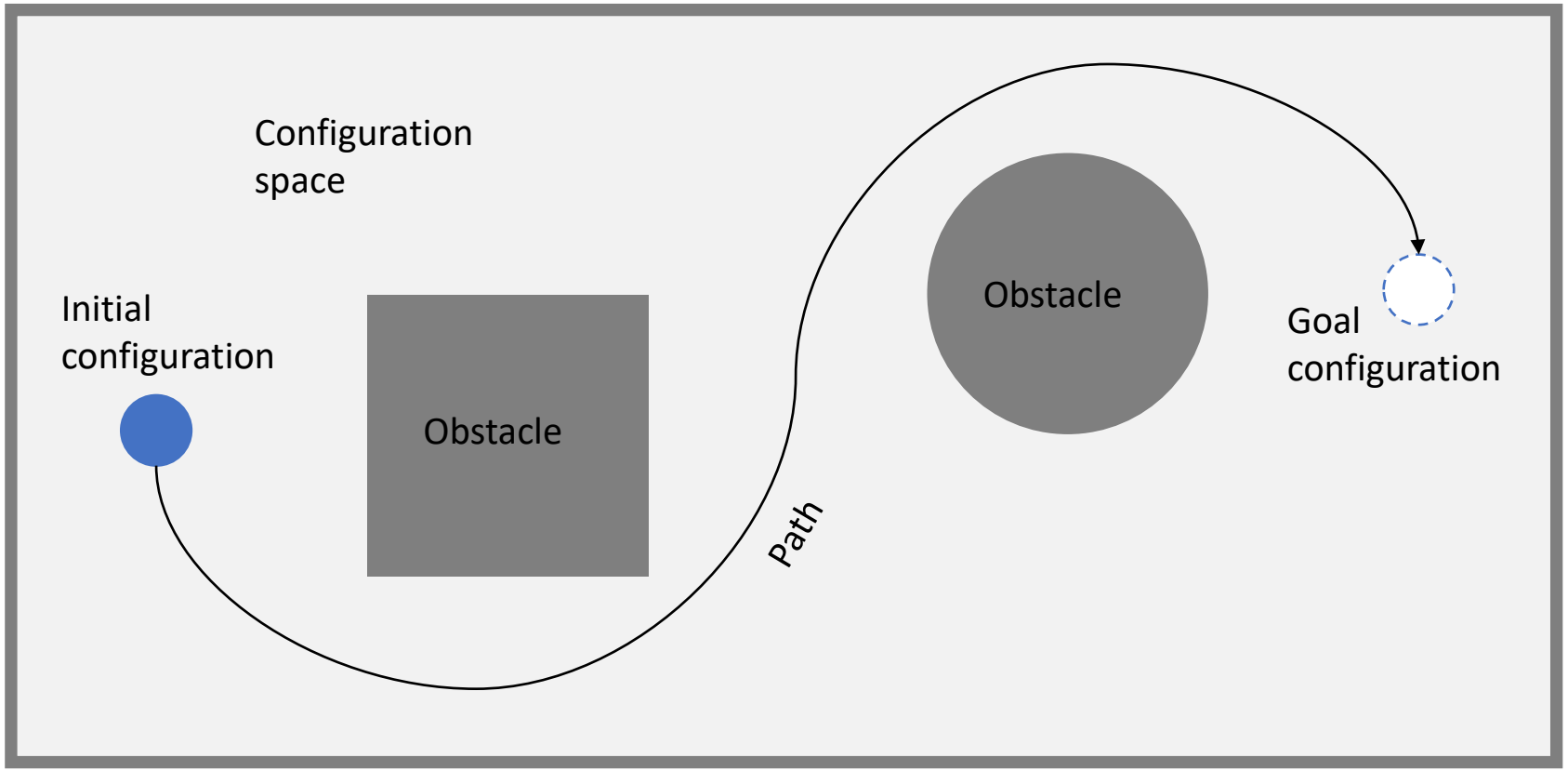
Simple problem 1

Configuration:

$$\boldsymbol{q} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Configuration space $\mathcal{C}$:

$x \in \mathbb{R}, x_{min} < x < x_{max},$
$y \in \mathbb{R}, y_{min} < y < y_{max}$



Problem definition
Motion planning

Given:

- Kinematic system which motions are described by its possible configurations $\boldsymbol{q}$ in configuration space $\mathcal{C}$.
- Initial configuration $\boldsymbol{q}_{init}$
- Goal configuration $\boldsymbol{q}_{goal}$

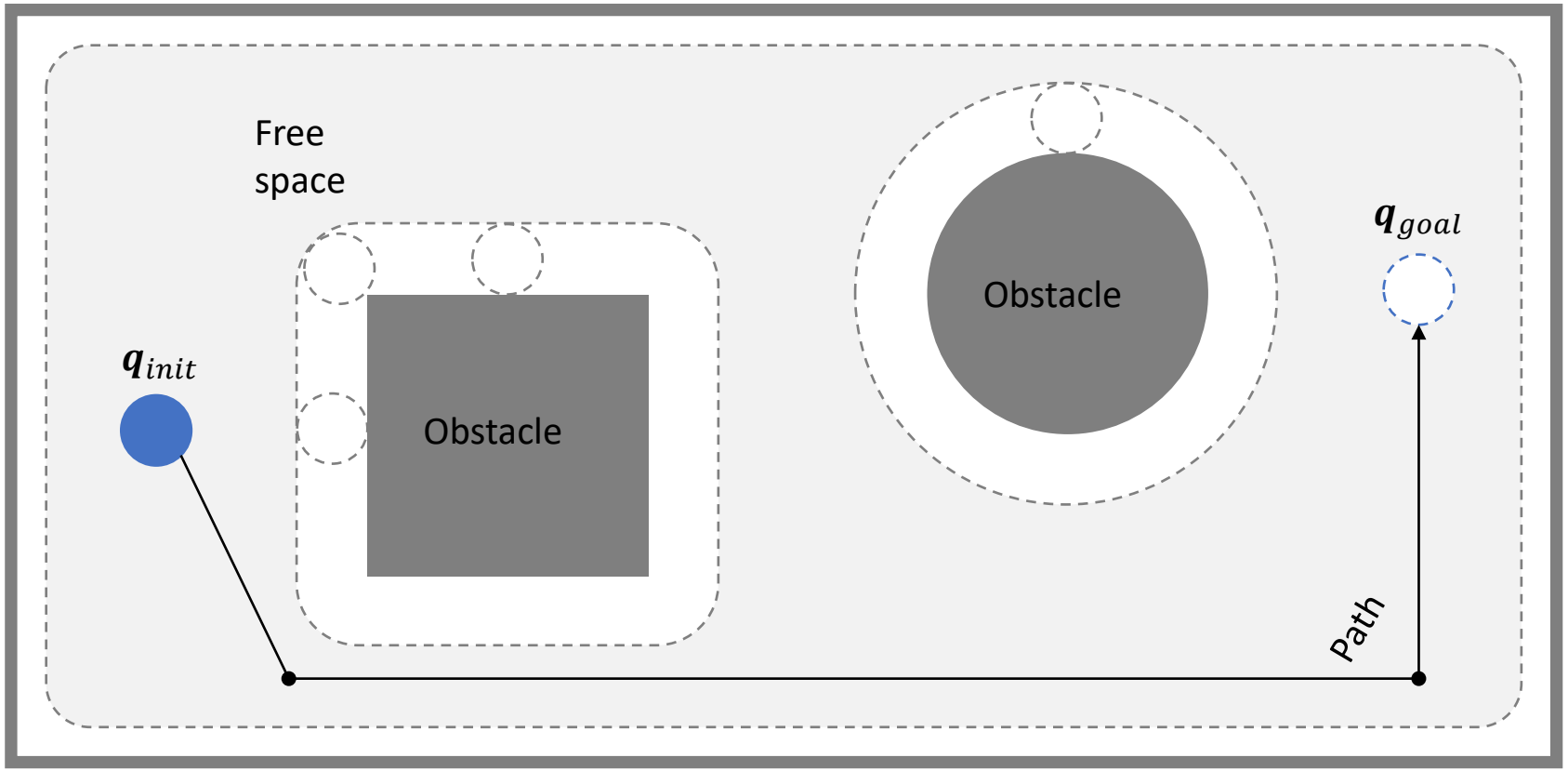
Problem:

Find a collision-free, continuous path in configuration space that connects start and goal configuration.

Basic approach
Search in free space

Enlarge obstacles by robot geometry to restrict all possible configurations to collision-free subspace $\mathcal{C}_{free}$.

Find a sequence of valid configurations in $\mathcal{C}_{free}$ and construct a path.



Sample problem 2

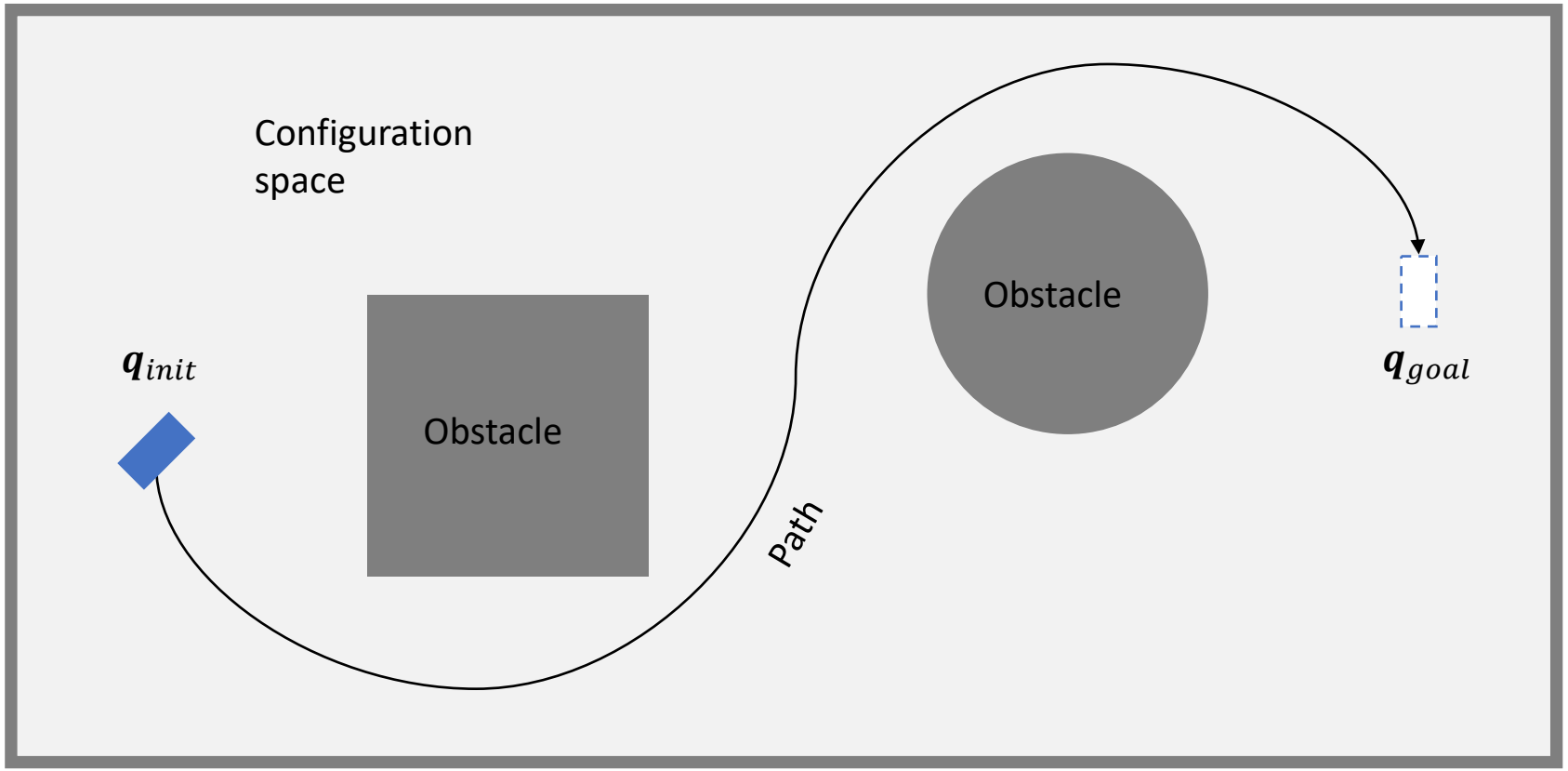
$\boldsymbol{q} \in \mathbb{R}^3$

Configuration:

$$\boldsymbol{q} = \begin{bmatrix} x \\ y \\ \varphi \end{bmatrix}$$

Configuration space $\mathcal{C}$:

$x \in \mathbb{R}, x_{min} < x < x_{max},$
$y \in \mathbb{R}, y_{min} < y < y_{max},$
$\varphi \in \mathbb{R}, \varphi_{min} < \varphi < \varphi_{max}$

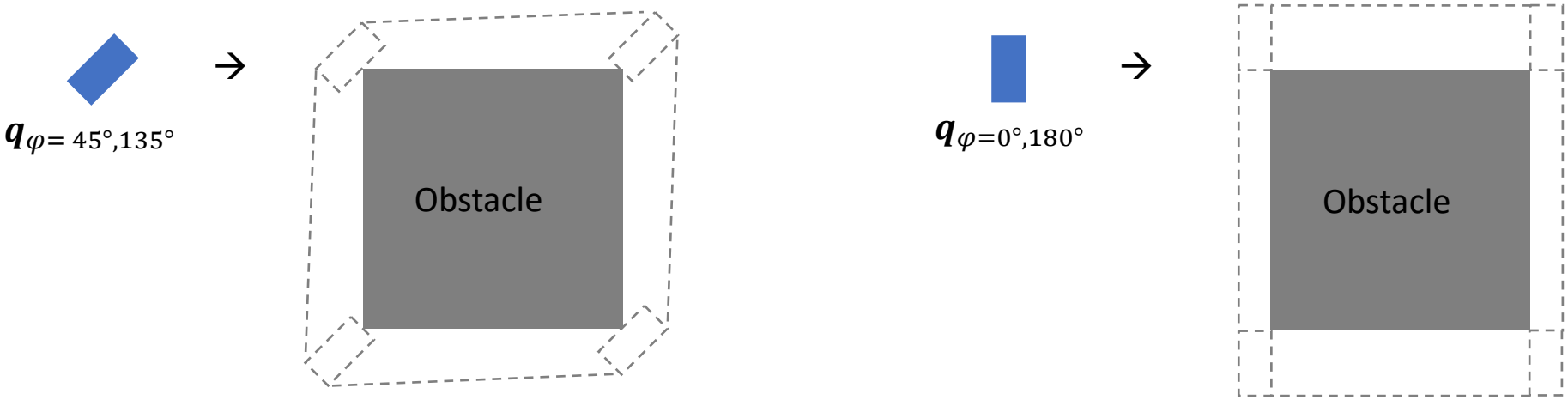


Free space construction

$\mathcal{C}_{free}(\boldsymbol{q})$

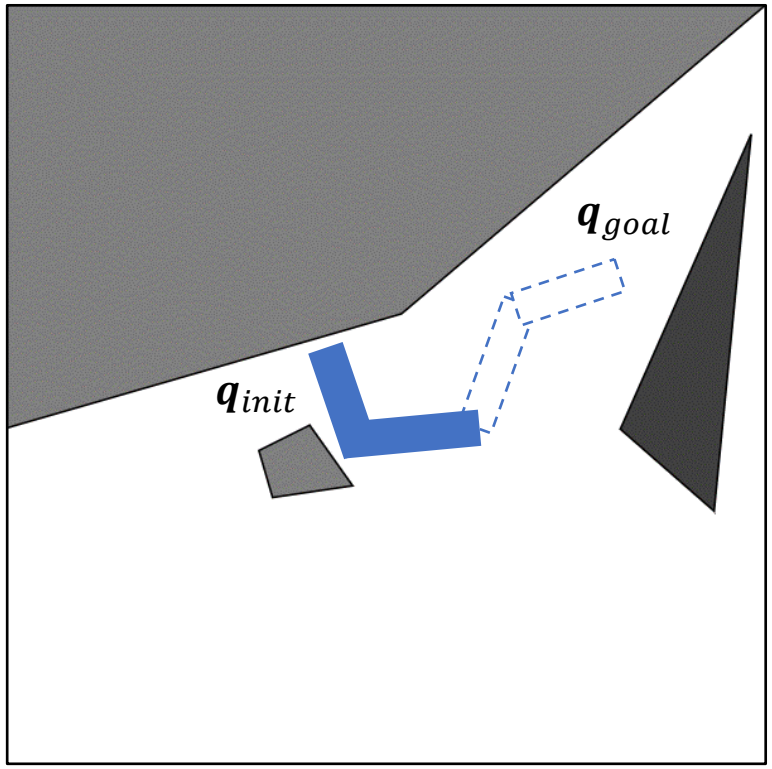
In general, free space is a function of the current configuration, $\mathcal{C}_{free}(\boldsymbol{q})$!

Thus it's typically $\mathcal{C}_{free}(\boldsymbol{q}(t))$, if the kinematic systems moves with $\boldsymbol{q}(t)$.



C-space for manipulators

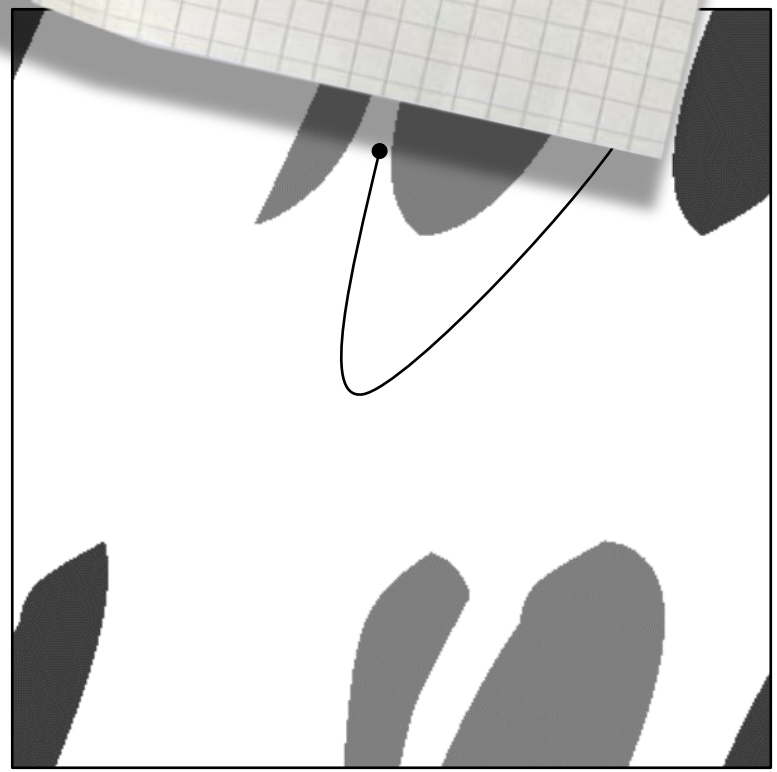
Two planar joints



Workspace

Exercise / 10 min.

https://www.cs.unc.edu/~jeffi/c-space/robot.xhtml.



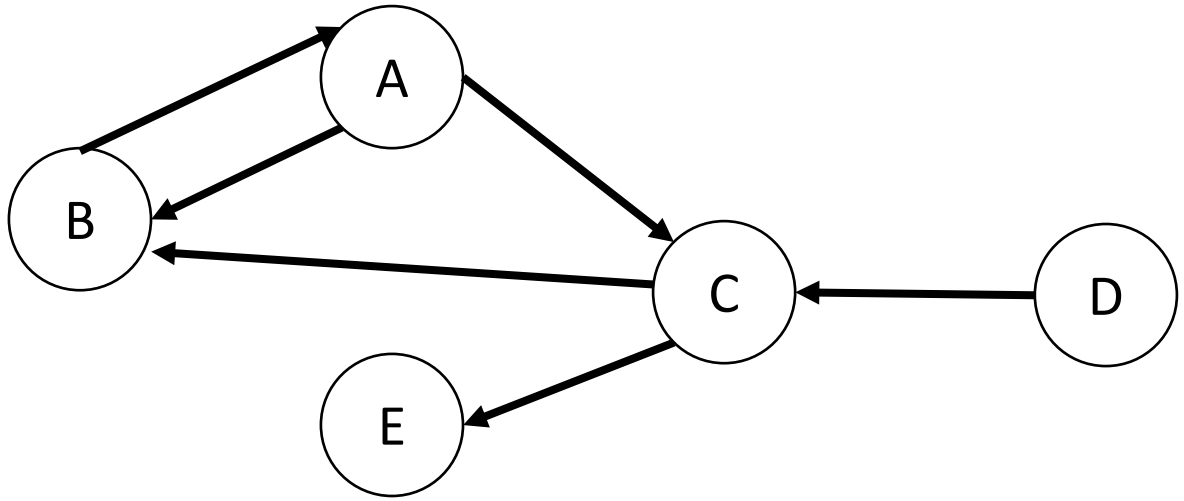
C-space

Graph and Trees

- Many motion planners explicitly or implicitly represent the C-space or state space as a graph.
- A graph consists of a collection of nodes V and a collection of edges E, where each edge E connects two nodes.
- In motion planning, a node typically represents a configuration or state while an edge between nodes V_1 and V_2 indicates the ability to move from V_1 to V_2 without penetrating an obstacle or violating other constraints.

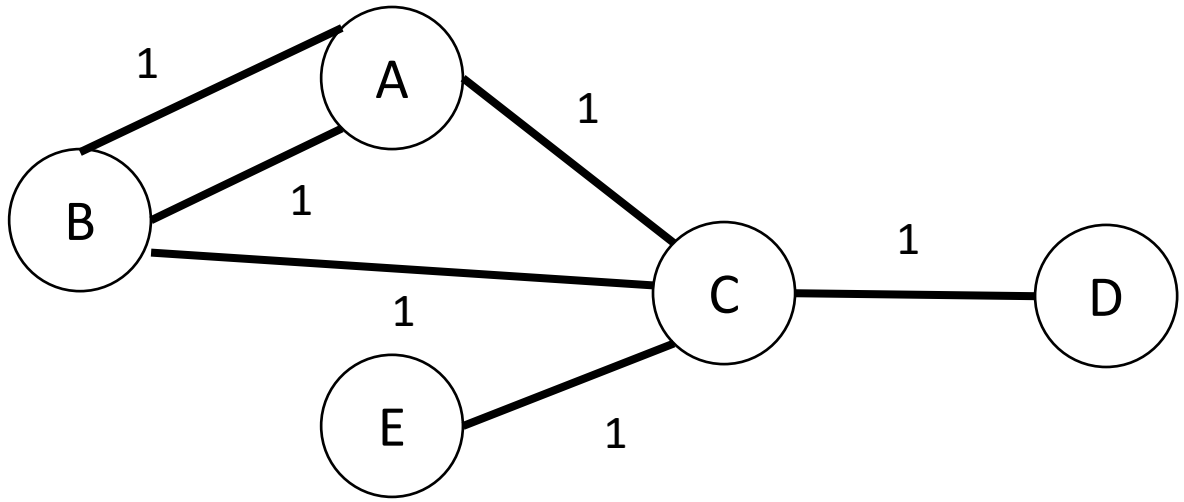
Graph and Trees

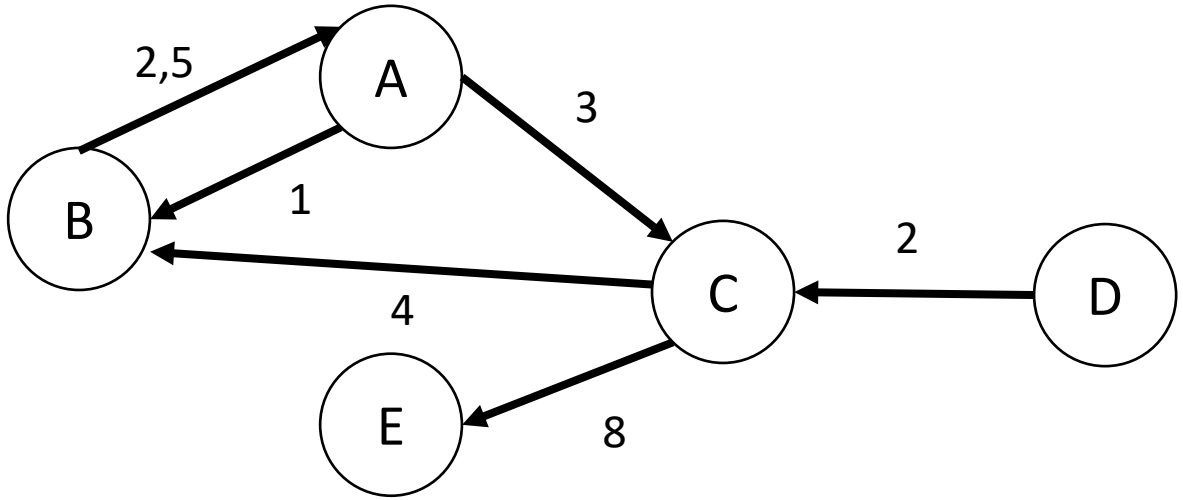
- A graph can be either **directed** or **undirected**.
- In an undirected graph, each edge is bidirectional: if the robot can travel from V_1 to V_2 then it can also travel from V_2 to V_1. In a directed graph, each edge allows travel in only one direction.
- The same two nodes can have two edges between them, allowing travel in opposite directions.



Graph and Trees

- Graphs can also be **weighted** or **unweighted**.
- In a weighted graph, each edge has a positive cost associated with traversing it.
- In an unweighted graph each edge has the same cost (e.g., 1). Thus the most general type of graph we consider is a weighted digraph.
- Direction and weight can also be considered





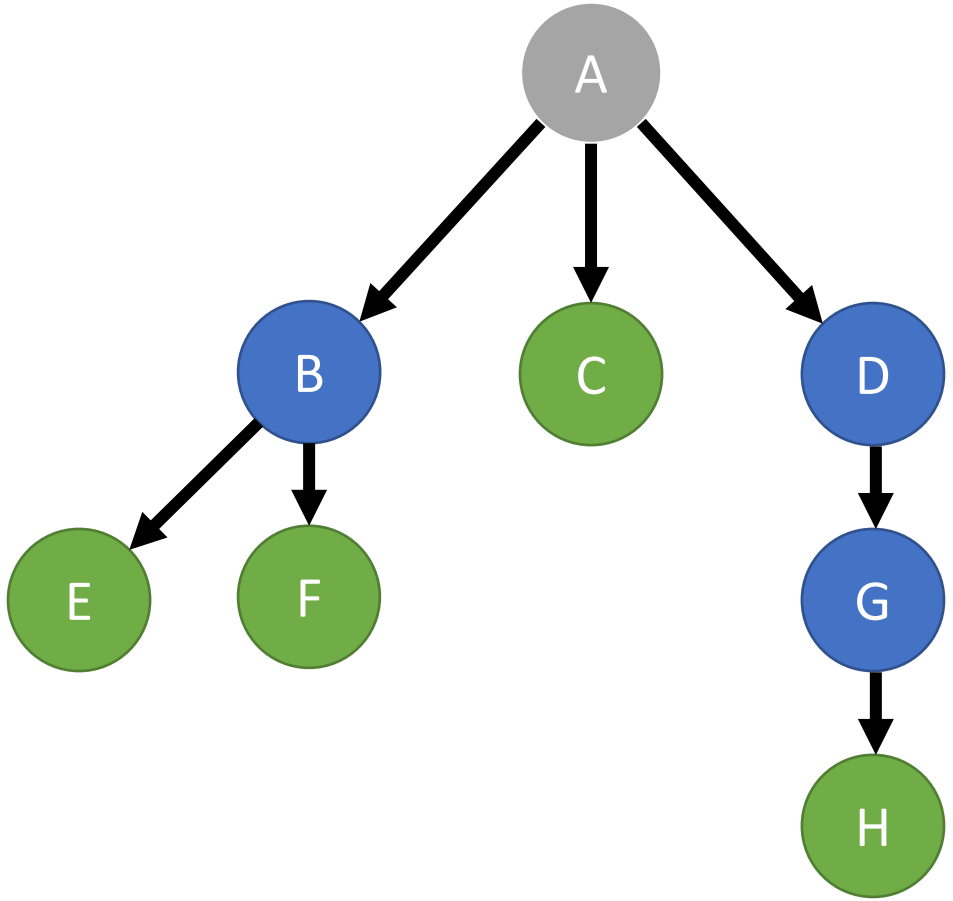
Graph and Trees

- A **tree** is a **directional graph** in which

  (1) there are no cycles and

  (2) each node has at most one parent node (i.e., at most one edge leading to the node).

- A tree has one root node (gray) with no parents, several children nodes (blue) and a number of leaf (green) nodes with no child.



Rapidly-exploring random trees (RRT)

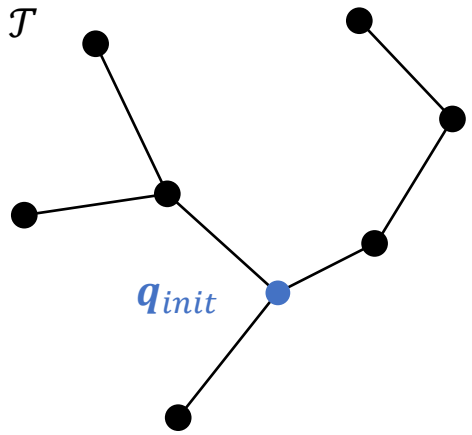
Rapidly-exploring random trees (RRT)
Steven M. LaValle, 1998

Symbols	
$\mathcal{T}$	Graph datastructure
$\boldsymbol{q}_x$	Configurations
Results of EXTEND	
REACHED	Goal reached
ADVANCED	Graph successfully advanced
TRAPPED	Graph can't be advanced here

BUILD_RRT($\boldsymbol{q}_{init}$)	
1	$\mathcal{T}$.init($\boldsymbol{q}_{init}$);
2	**for** k = 1 **to** K **do {**
3	$\boldsymbol{q}_{rand} \leftarrow$ RANDOM_CONFIG();
4	EXTEND($\mathcal{T}$, $\boldsymbol{q}_{rand}$);
5	**}** **return** $\mathcal{T}$;

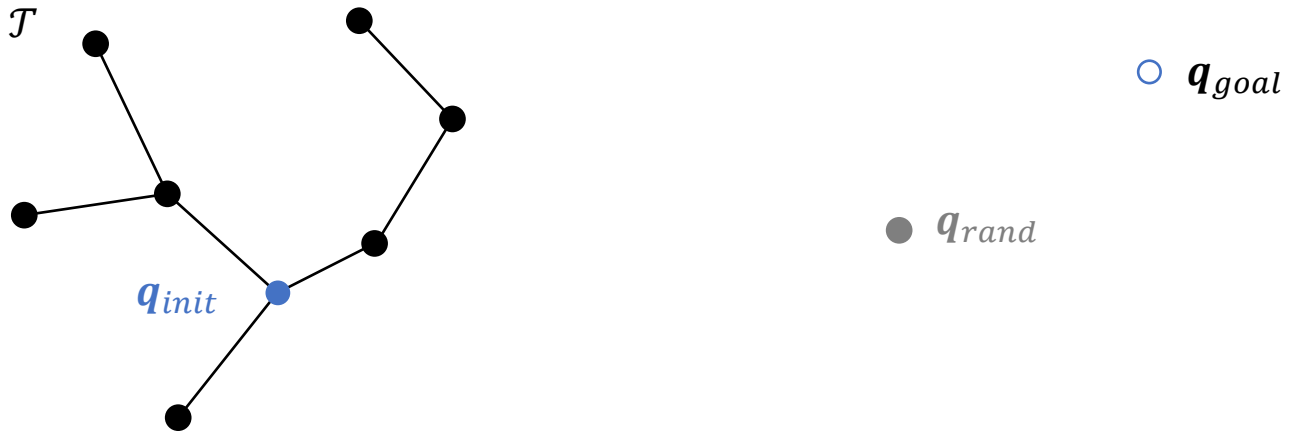
EXTEND($\mathcal{T}$, $\mathbf{q}$)	
1	$\boldsymbol{q}_{near} \leftarrow$ NEAREST_NEIGHBOR($\boldsymbol{q}$, $\mathcal{T}$);
2	**if** NEW_CONFIG($\boldsymbol{q}$, $\boldsymbol{q}_{near}$, $\boldsymbol{q}_{new}$) **then {**
3	$\mathcal{T}$.add_vertex($\boldsymbol{q}_{new}$);
4	$\mathcal{T}$.add_edge($\boldsymbol{q}_{near}$, $\boldsymbol{q}_{new}$);
5	**if** ($\boldsymbol{q}_{new} = \boldsymbol{q}$) **then { return** REACHED; **}**
6	**else { return** *ADVANCED*; }
9	**}** **return** *TRAPPED*;

RRT
Setup

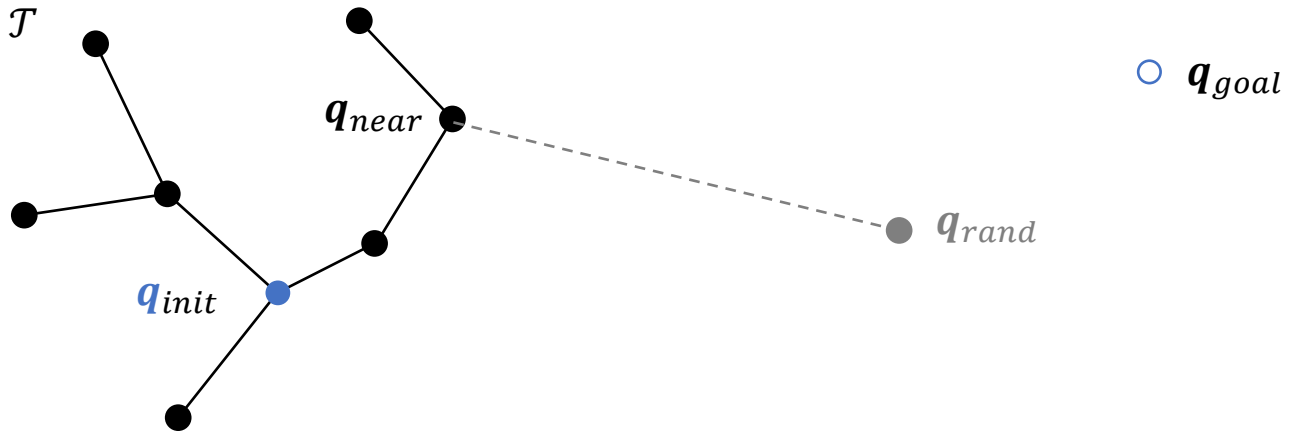


q_{goal}

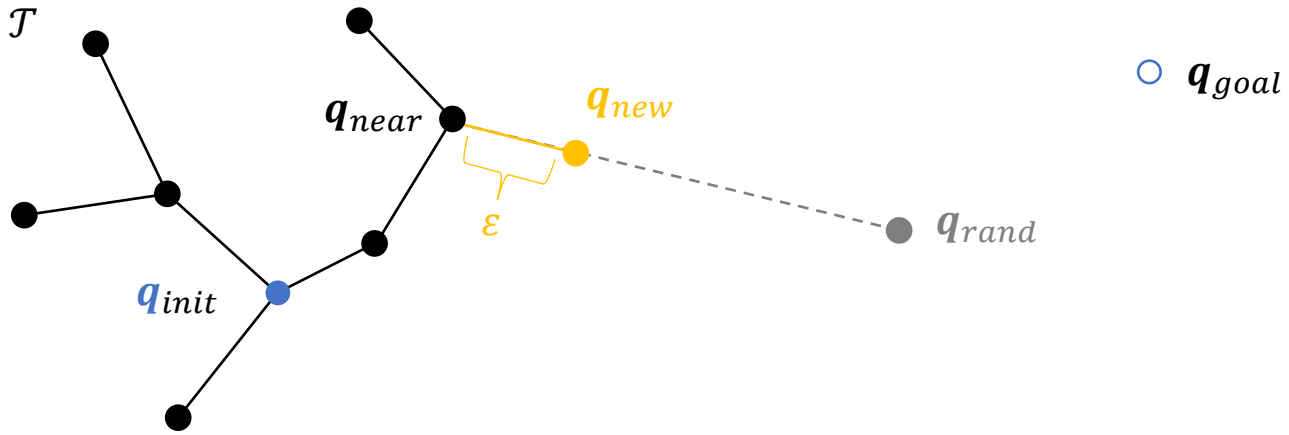
RRT
RANDOM_CONFIG



RRT
NEAREST_NEIGHBOR

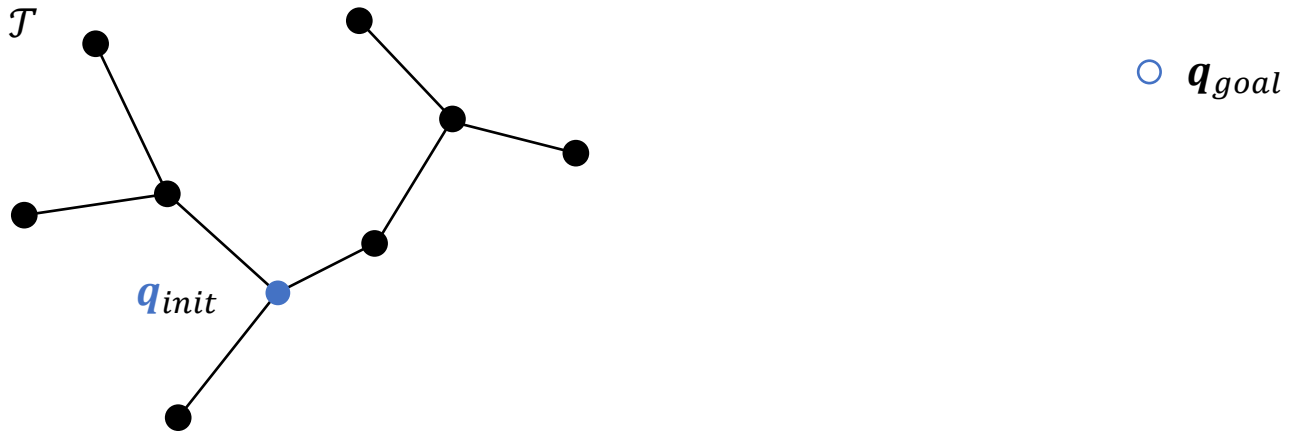


RRT
NEW_CONFIG



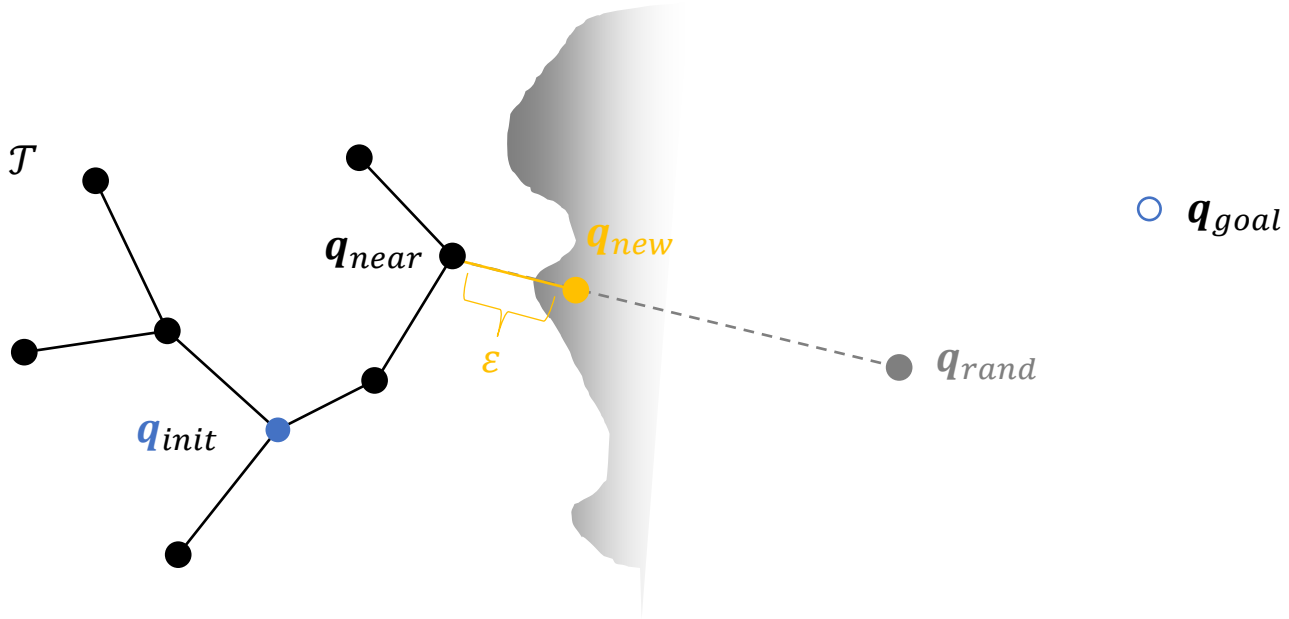
RRT
EXTEND returns ADVANCED

If $\boldsymbol{q}_{new} \in \mathcal{C}_{free}$, add it to $\mathcal{T}$ and proceed with k = k+1.



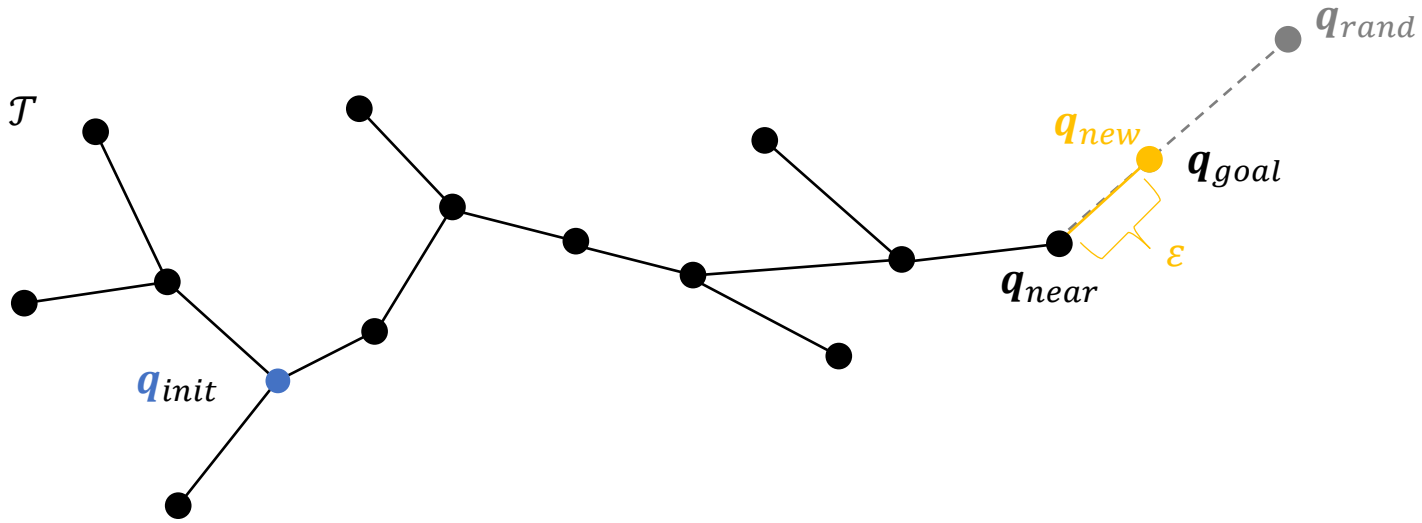
RRT
EXTEND returns TRAPPED

If $\boldsymbol{q}_{new} \in \mathcal{C}_{obs}$, drop it and proceed with k = k+1.



RRT
EXTEND returns REACHED

If $\boldsymbol{q}_{new} = \boldsymbol{q}_{goal}$, $\mathcal{T}$ includes a path between $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$ in $\mathcal{C}_{free}$!



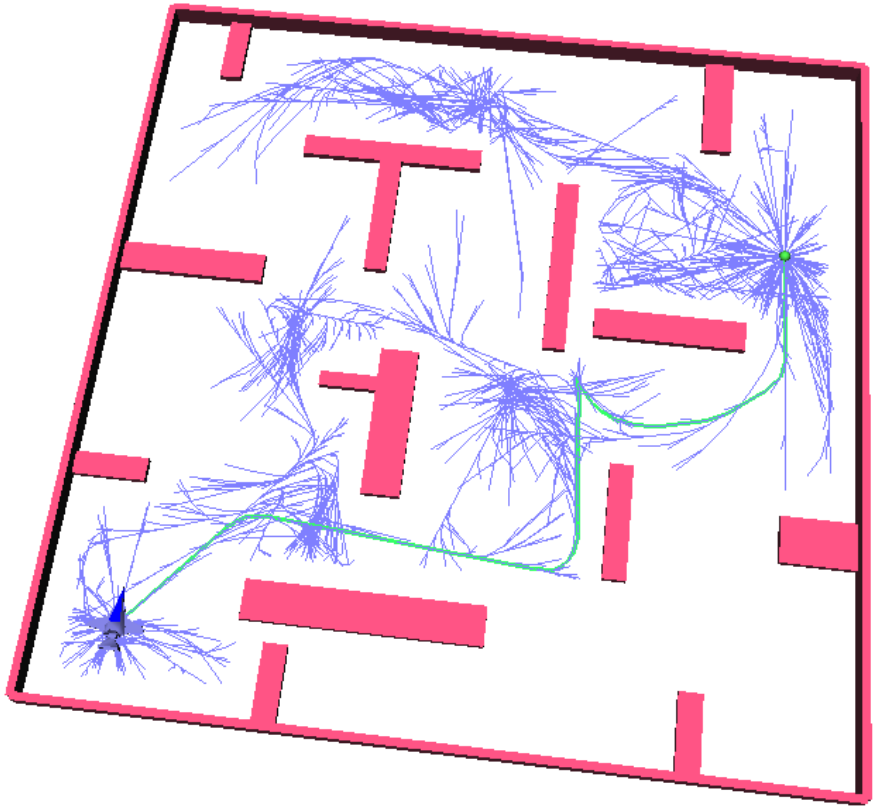
Exploration vs. Exploitation
Step size and biased sampling

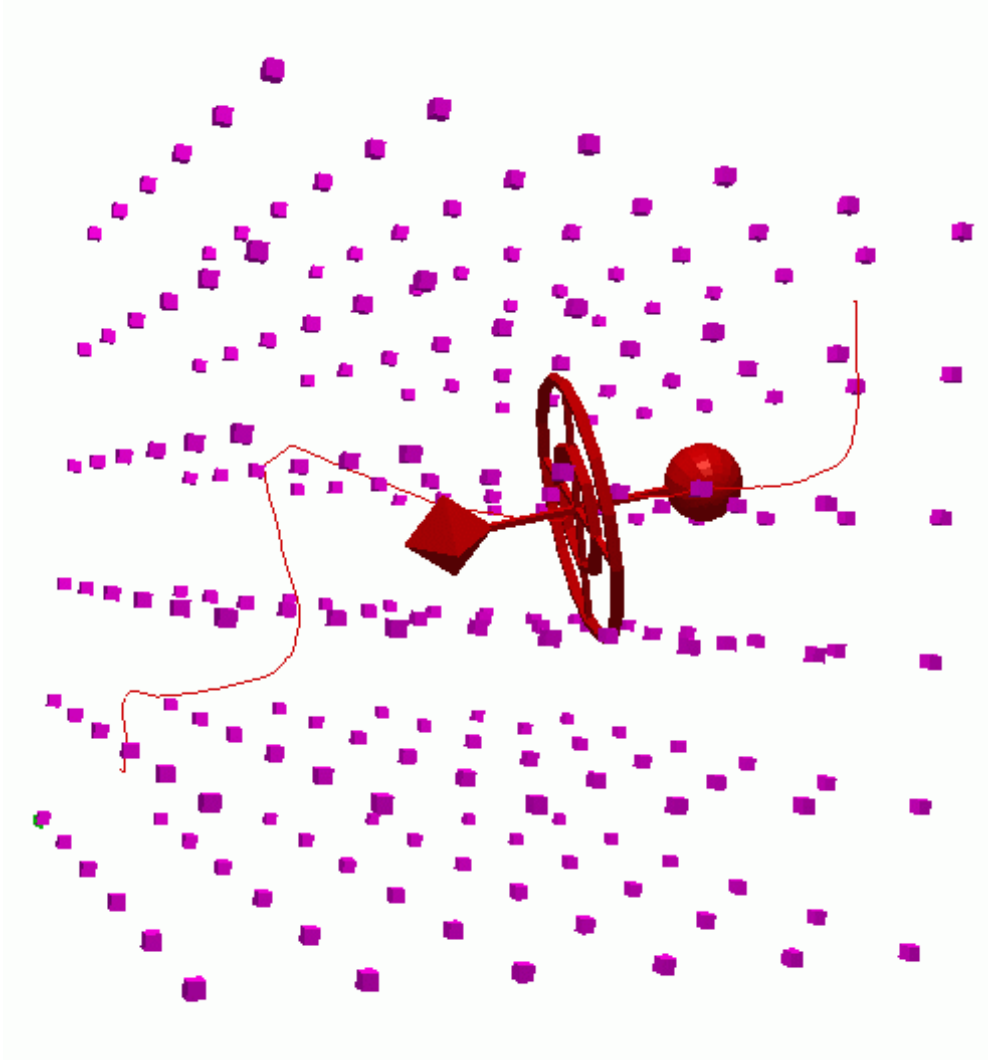
- **Step size**
- Parameter ϵ controls the distance covered by RRT in one step. Smaller ϵ will detect obstacles more accurately, but will also take longer to compute a valid path reaching the goal. Larger ϵ will progress faster towards in C-space, but may also miss details of obstacles.

- **Biased sampling**
- Instead of using truly random sample configurations, RRT can be tuned to explore the C-space more rapidly by favoring unexplored regions, e.g. by selecting samples from larger **Voronoi regions** with higher probability.

RRT
Example

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





RRT -Discussion

- It can be shown, that RRT yields **probabilistic completeness**, as it will eventually reach and cover all of C-space.
- RRT generally shows a better performance and needs less memory than conventional approaches.

- However, to find the NEAREST-NEIGHBOR the tree typically is computational expensive!
- Plus, RRT misses a drive towards the goal…

→ Both issues are addressed in RRT-Connect.

RRT- Connect

RRT-Connect

James J. Kuffner, Steven M. LaValle, 2000

Symbols	
S	Result of EXTEND
$\mathcal{T}_i$	Graph datastructures
$\boldsymbol{q}_x$	Configurations
Results of EXTEND and CONNECT	
REACHED	Goal reached
ADVANCED	Graph successfully advanced
TRAPPED	Graph can't be advanced here
Results of RRT_CONNECT_PLANNER	
PATH($\mathcal{T}_\alpha$, $\mathcal{T}_\beta$)	Valid path between $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$
FAILURE	Unable to find a valid path in K steps

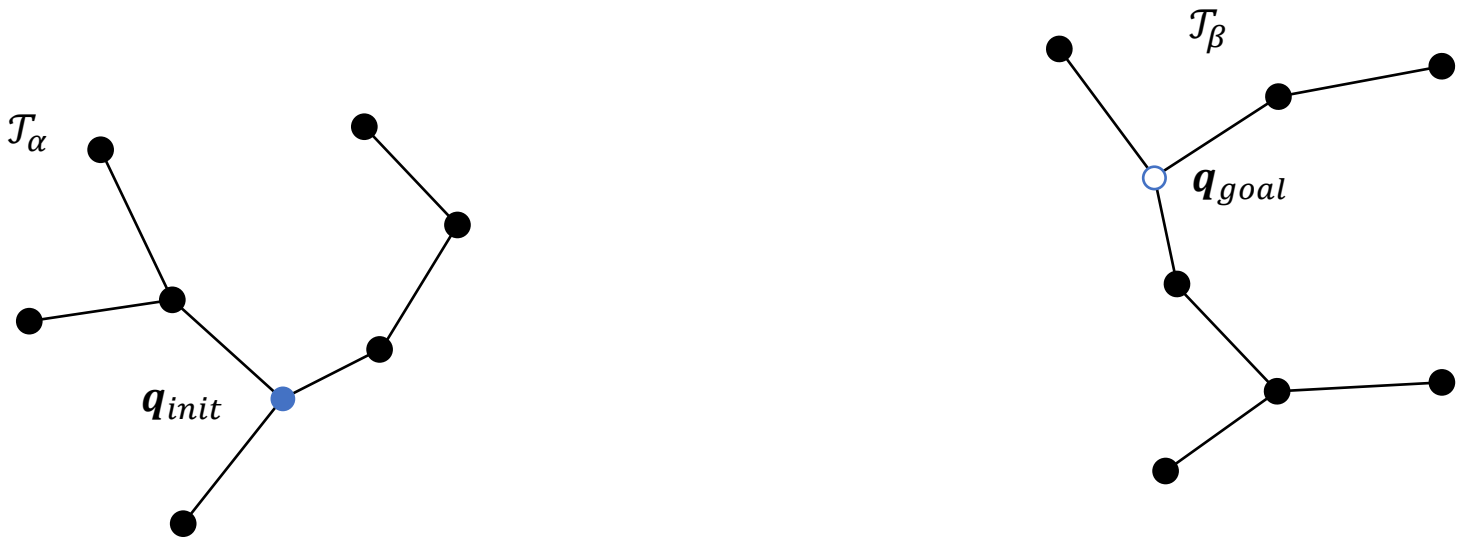
CONNECT($\mathcal{T}$, $\mathbf{q}$)

```
1  repeat {
2      S ← EXTEND(T, q);
3  } until not (S = ADVANCED);
4  return S;
```

RRT_CONNECT_PLANNER($\boldsymbol{q}_{init}$, $\boldsymbol{q}_{goal}$)

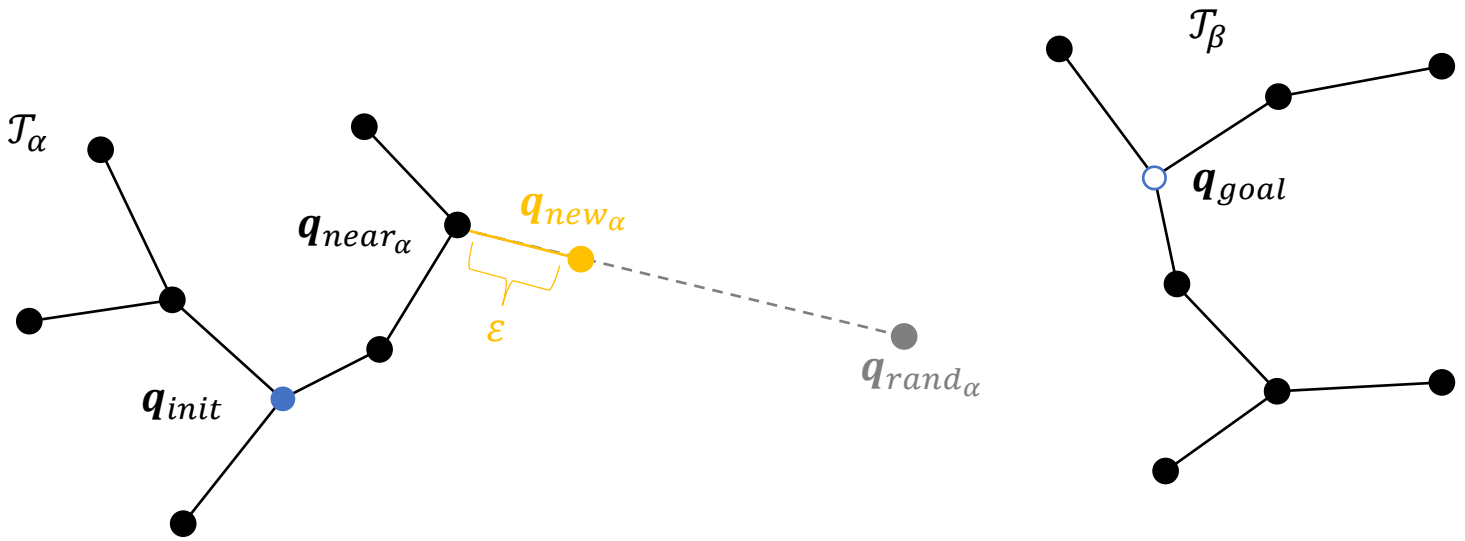
```
1  Tα.init(q_init); Tβ.init(q_goal);
2  for k = 1 to K do {
3      q_rand ← RANDOM_CONFIG();
4      if not (EXTEND(Tα, q_rand) = TRAPPED) then {
5          if (CONNECT(Tβ, q_new) = REACHED) then { return PATH(Tα, Tβ); }
6      } SWAP(Tα, Tβ);
7  } return FAILURE;
```

RRT
Setup


$\mathcal{T}_\alpha$
$\boldsymbol{q}_{init}$
$\mathcal{T}_\beta$
$\boldsymbol{q}_{goal}$

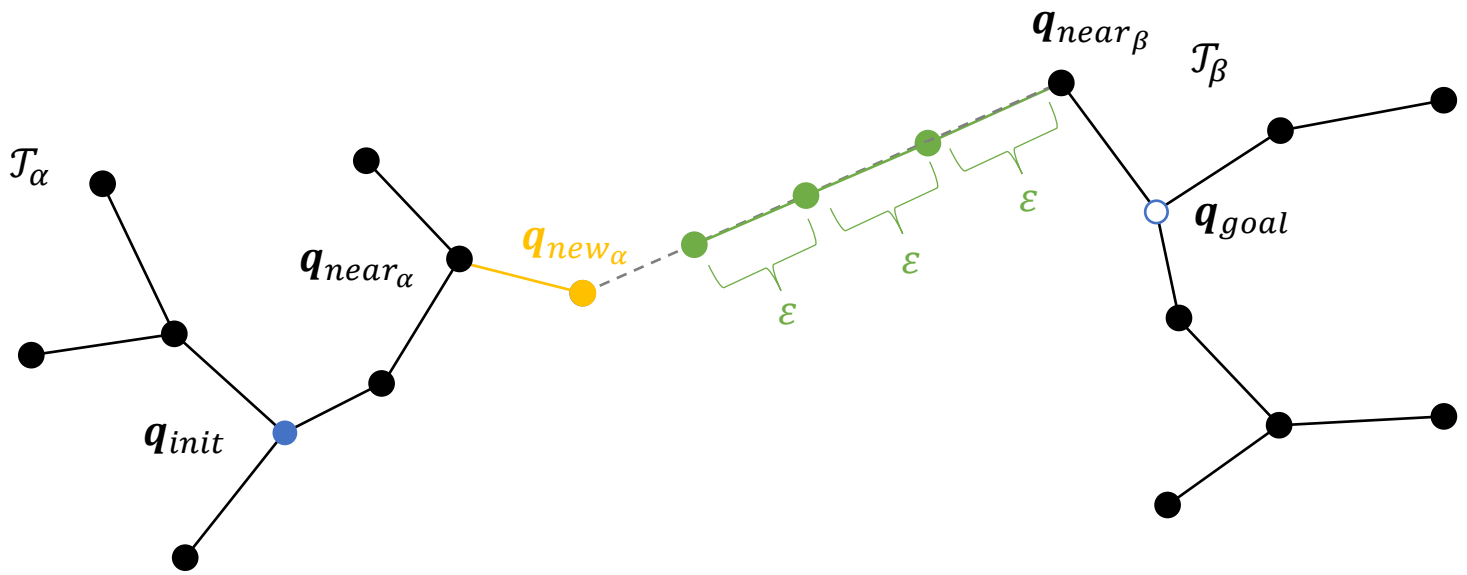
RRT
EXTEND $\mathcal{T}_\beta$

If $\boldsymbol{q}_{new_\alpha} \in \mathcal{C}_{free}$, add it to $\mathcal{T}_\alpha$ and proceed with CONNECT, otherwise SWAP.



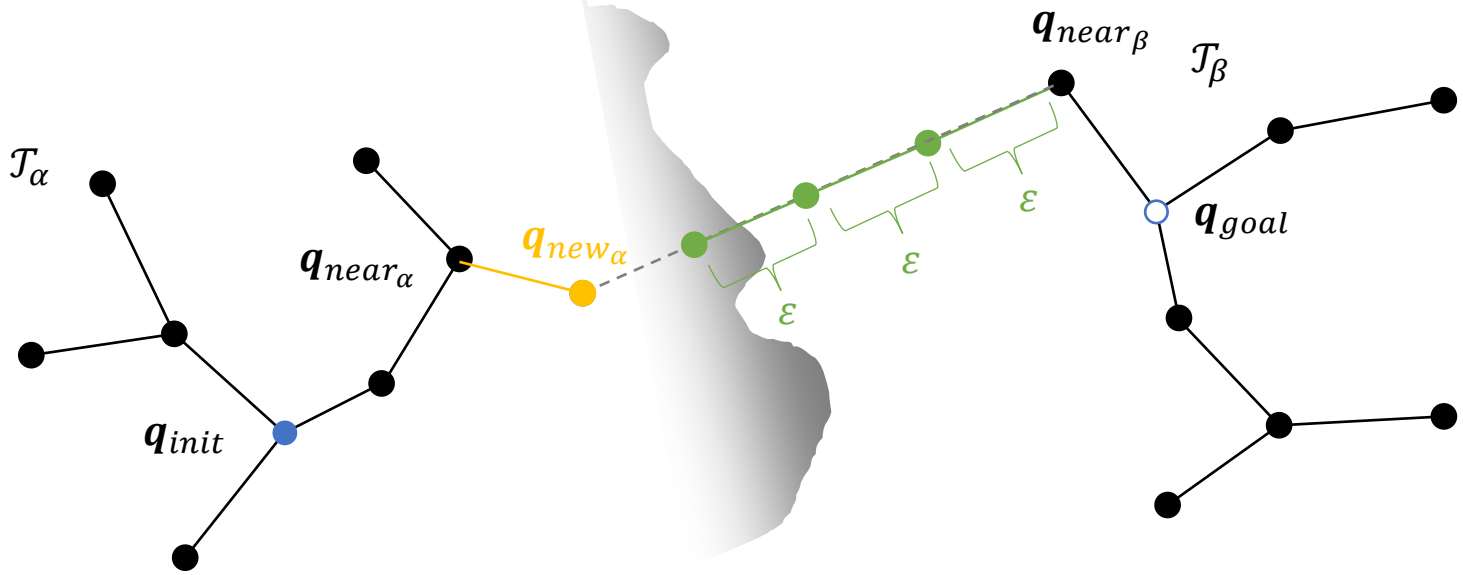
RRT
CONNECT $\mathcal{T}_\beta$

In a series of calls to EXTEND, try to connect $\boldsymbol{q}_{near_\beta}$ to $\boldsymbol{q}_{new_\alpha}$, otherwise SWAP.



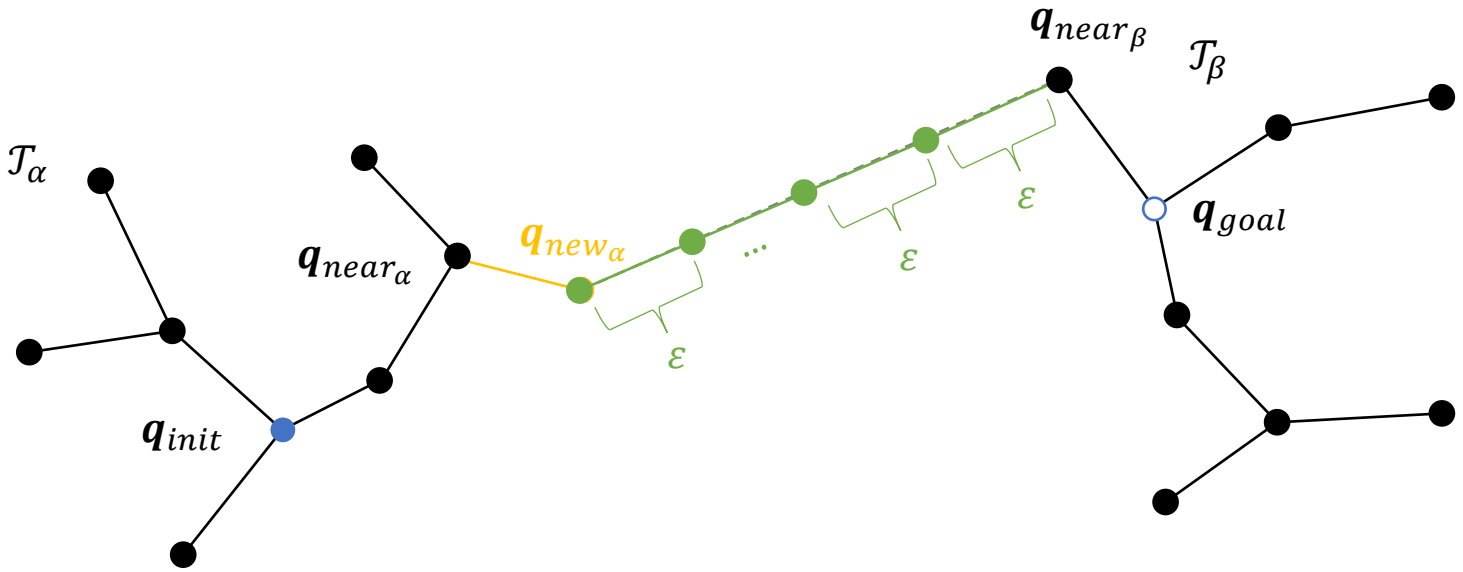
RRT
CONNECT $\mathcal{T}_\beta$ returns TRAPPED

If CONNECT stops due to obstacles, RRT-Connect continues with SWAP.



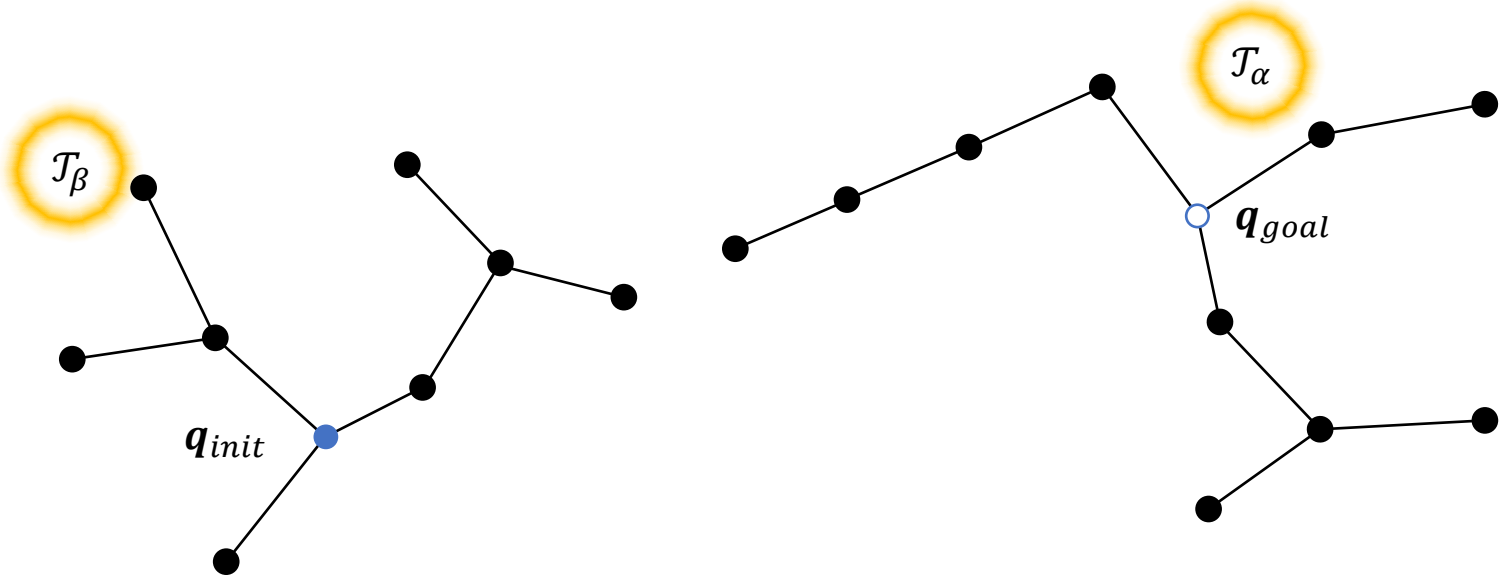
RRT
CONNECT $\mathcal{T}_\beta$ returns REACHED

If CONNECT reaches $\boldsymbol{q}_{new_\alpha}$, RRT-Connect can return a path between $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$!



RRT
SWAP

SWAP ensures balanced growth of both graphs.



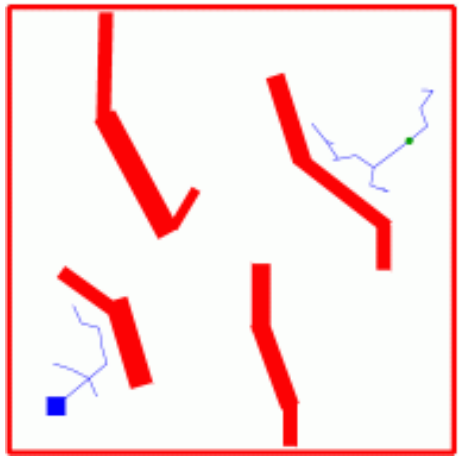
RRT-Connect
Discussion

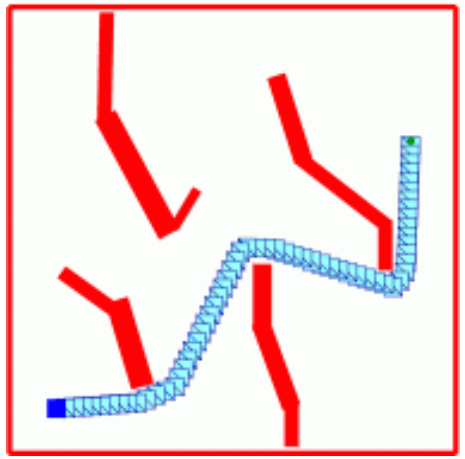
- In comparison to RRT, RRT-Connect is more directed towards reaching the goal and thus often faster.
- Plus, RRT-Connect calls NEAREST-NEIGHBOR more efficiently, as CONNECT can assume the latest graph extension to be the nearest neighbor.

→ Today, we prefer generally RRT-Connect over basic RRT.

RRT-Connect
Example 1 – 2DOF square robot

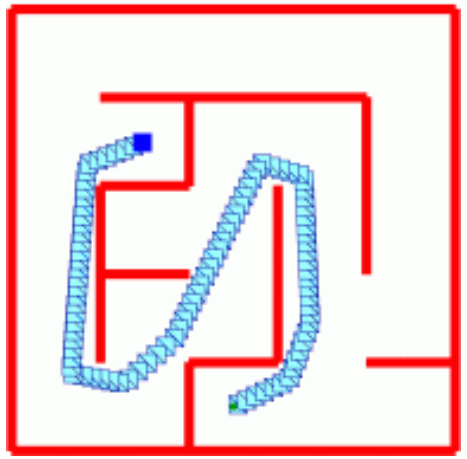
http://www.kuffner.org



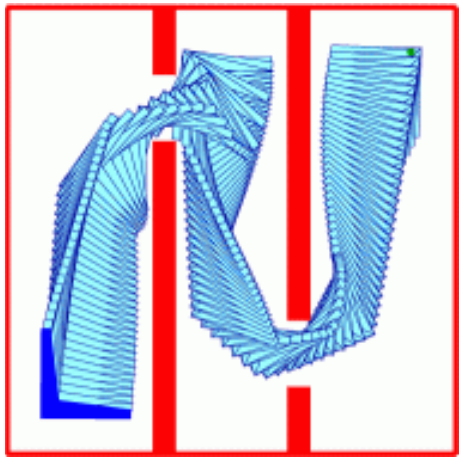
RRT-Connect
Example 2 – 2DOF square robot

http://www.kuffner.org



RRT-Connect
Example 3 – 3DOF L-shaped robot

http://www.kuffner.org



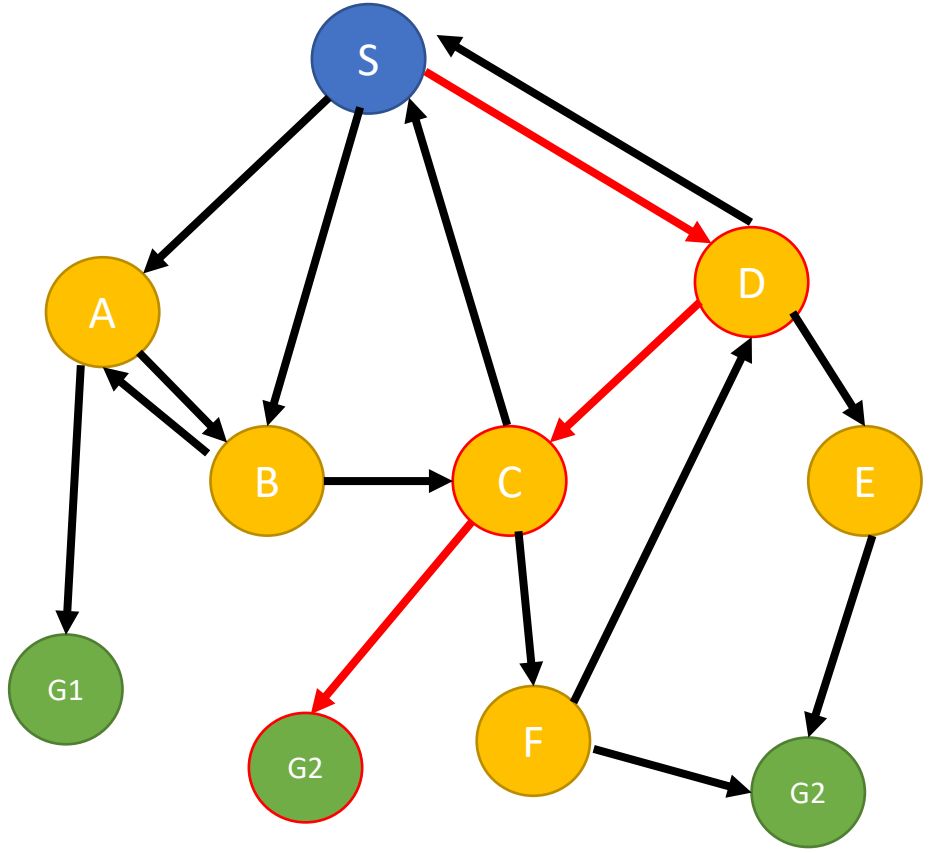
Optimization based motion planning

Optimization based motion planning

- The methods presented previously in the course are sample-based, probabilistic, and are based on rules (heuristics). The presented planners are probabilistically complete.
- If one can formulate an optimization problem for solving the motion planning problem, then under certain conditions such planner is complete.

A algorithm*

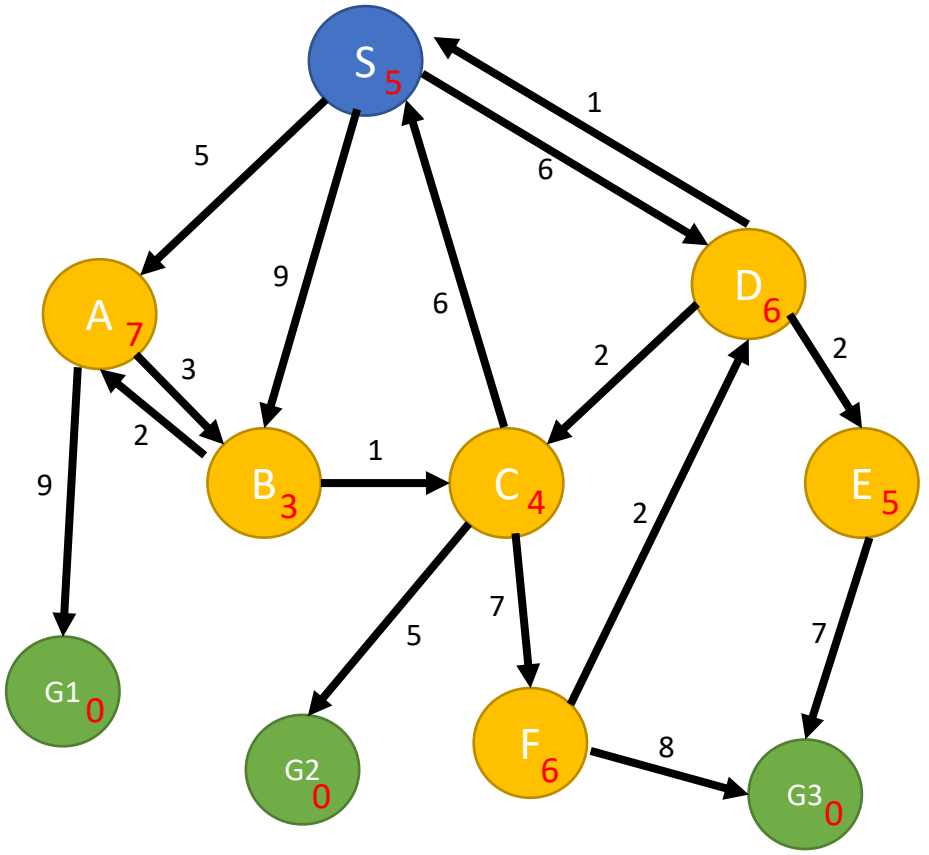
- The A* search algorithm efficiently finds a minimum-cost path on a graph when the cost of the path is simply the sum of the positive edge costs along the path.
- The algorithm uses heuristic rules to determine the cost of the edges and nodes, which are used for finding the optimal path.
- The algorithm is probabilistically complete.
- The planner will always find the shortest path between the nodes.



A algorithm*

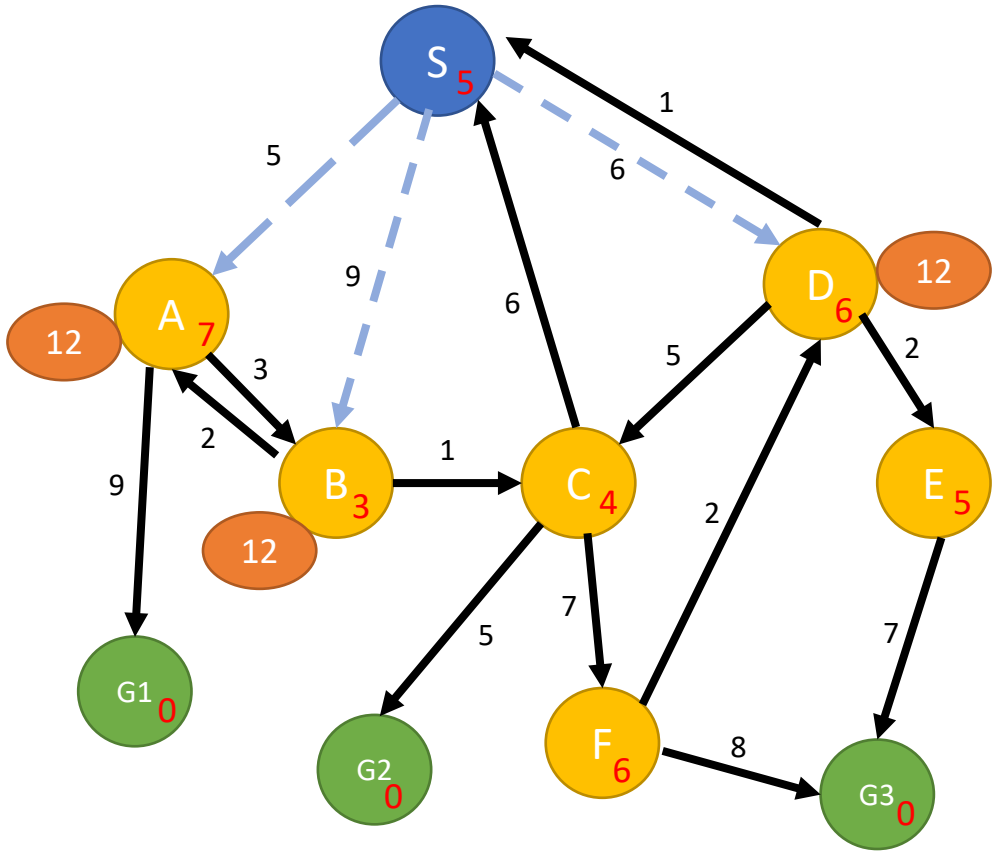
- The challenge is to find the shortest path between S and G1,G2 and G3
- Add cost to the edges and nodes,
- The edge cost can be the distance between the nodes,
- The node cost is heuristically determined and can be the distance to the goal state

Important to use a heuristic measure that never overestimates the cost



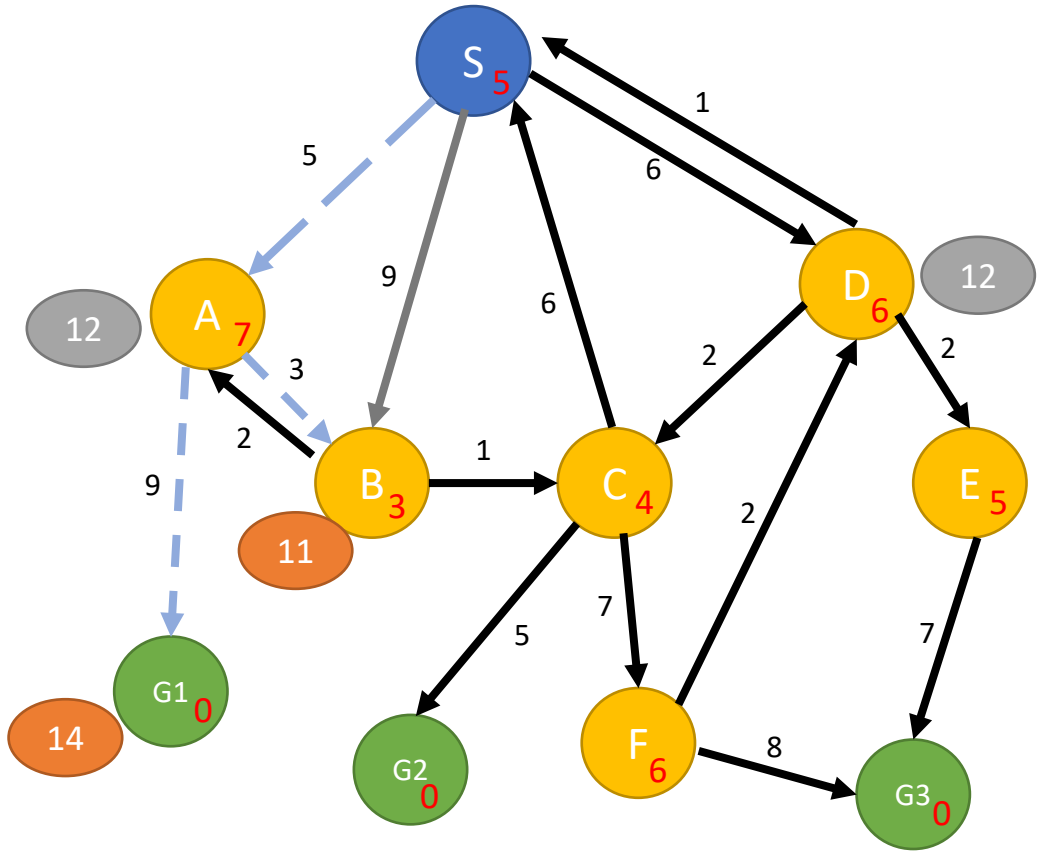
A algorithm*

- We start computing the A* score
  - Score = cost of the path + heuristics
- Candidate node: S(5)
- Frontier: A(7), B(3), D(6)
- Visited: S(5)



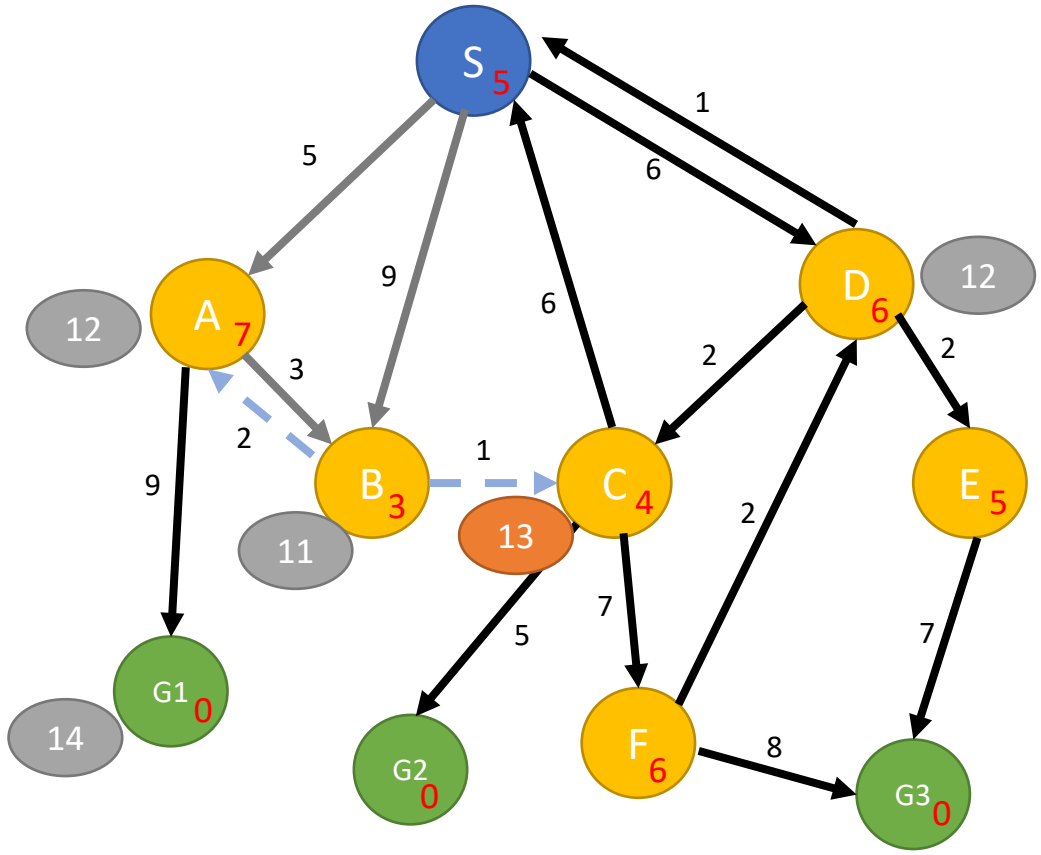
A algorithm*

- We start computing the A* score
  - Score = cost of the path + heuristics
- Candidate node: A(7)
- Frontier: B(3), G1(0)
- Visited: S(5), A(12)



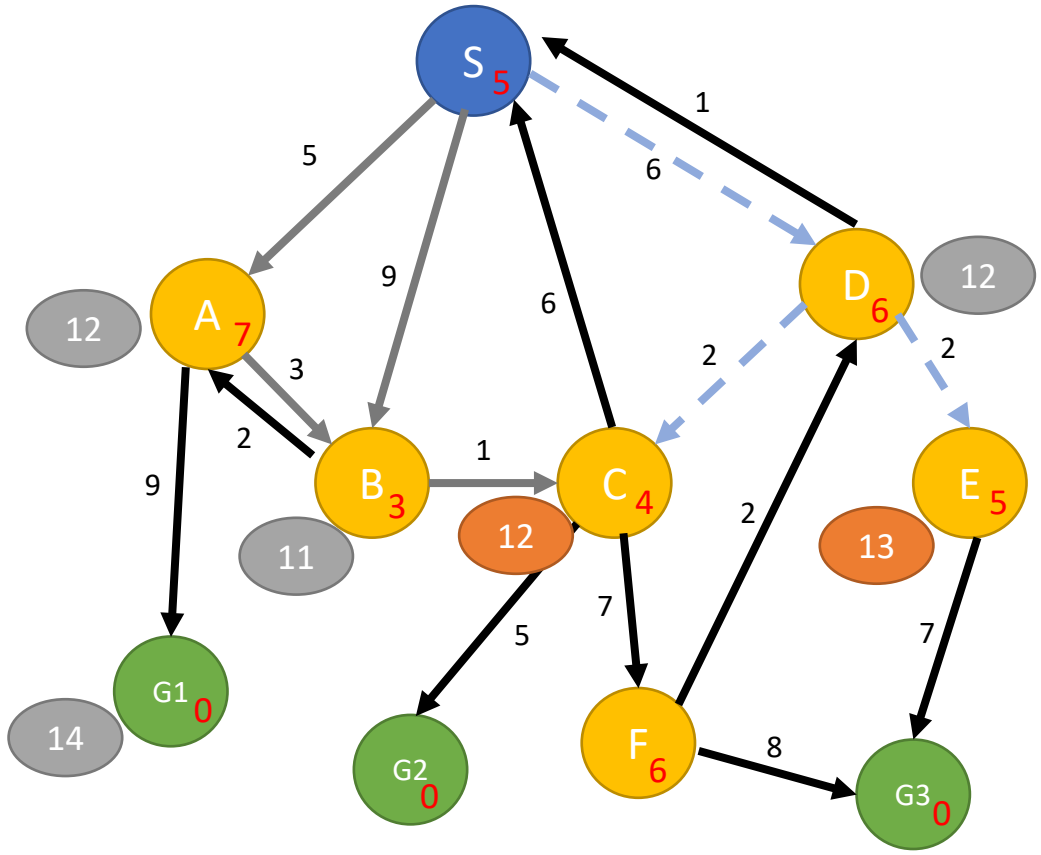
A algorithm*

- We start computing the A* score
  - Score = cost of the path + heuristics
- Candidate node: B(3)
- Frontier: A(7), C(4)
- Visited: S(5), A(12),B(11)



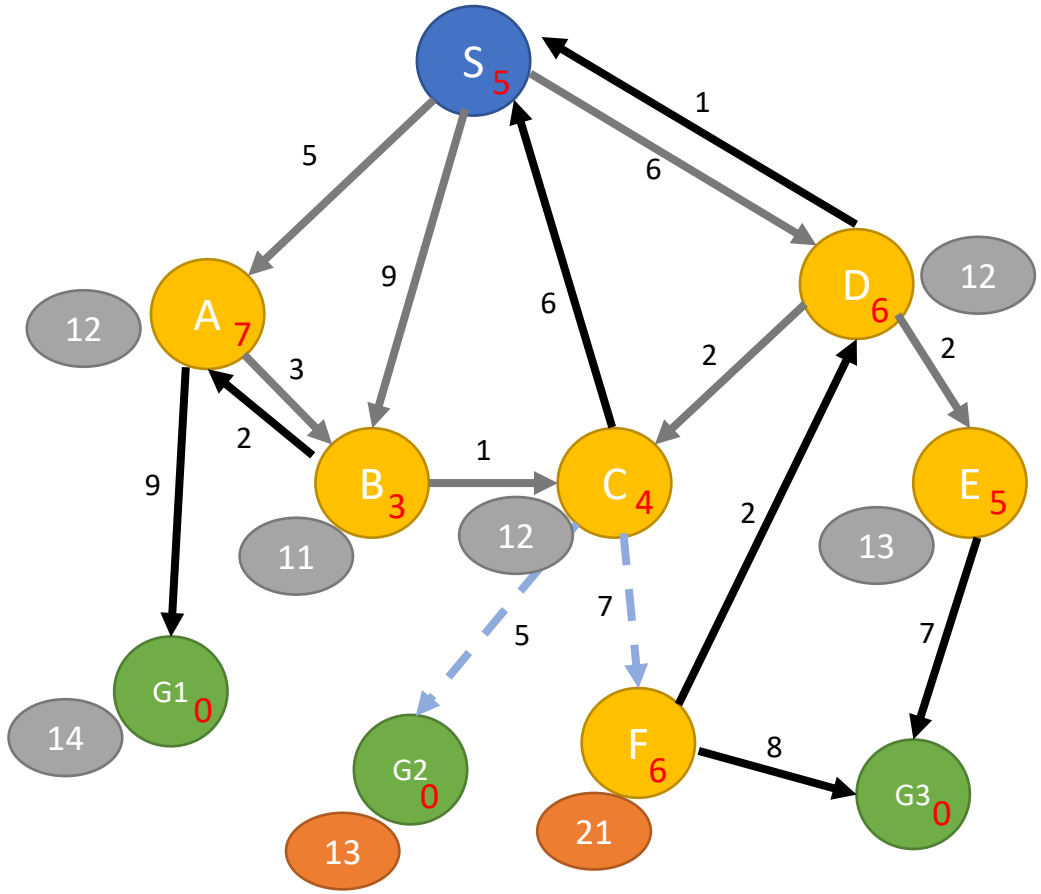
A algorithm*

- We start computing the A* score
  - Score = cost of the path + heuristics
- Candidate node: D(6)
- Frontier: C(4), E(5)
- Visited: S(5), A(12),B(11), D(12)



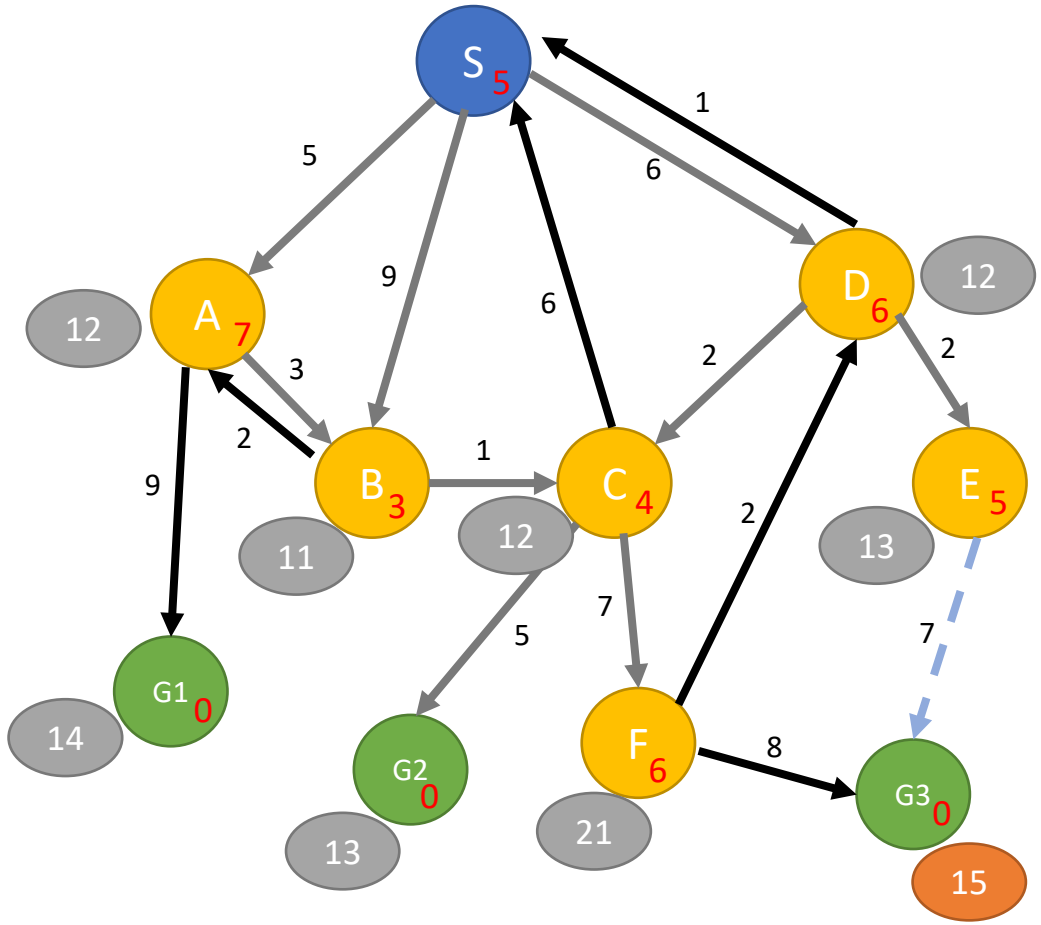
A algorithm*

- We start computing the A* score
  - Score = cost of the path + heuristics
- Candidate node: C(4)
- Frontier: G2(0), F(6)
- Visited: S(5), A(12),B(11), D(12), C(12)



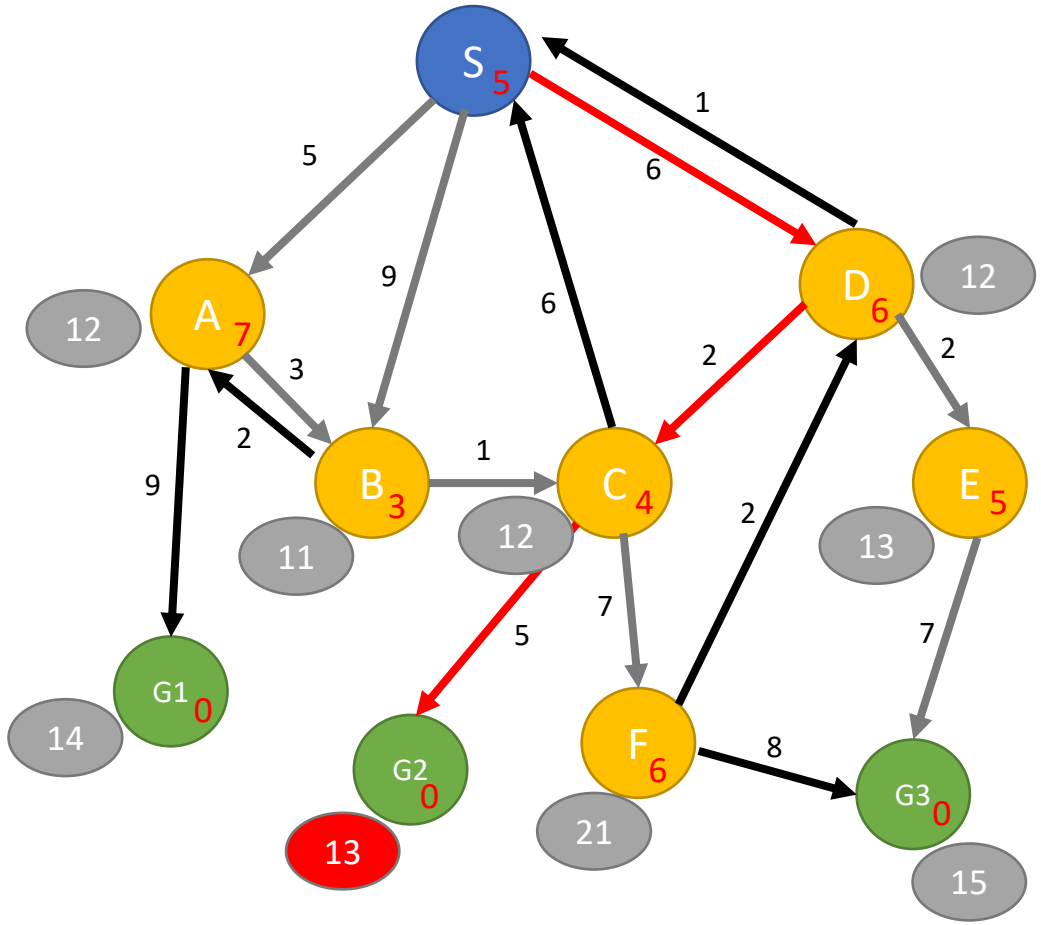
A algorithm*

- We start computing the A* score
  - Score = cost of the path + heuristics
- Candidate node: E(5)
- Frontier: G3(0)
- Visited: S(5), A(12),B(11), D(12), C(12), E(13)



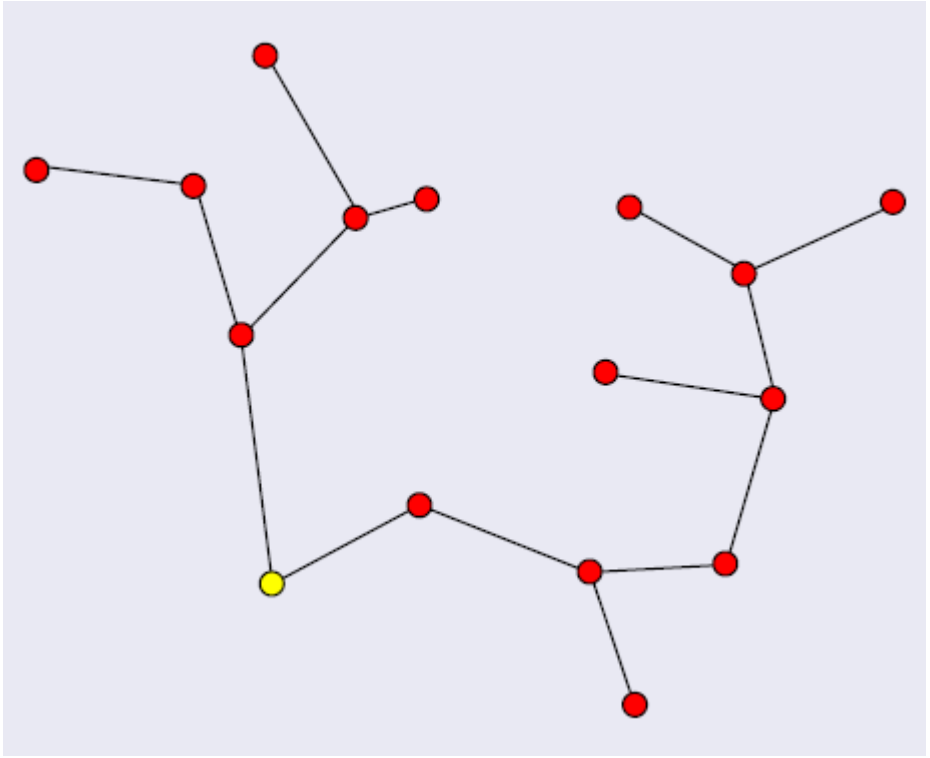
A algorithm*

- We start computing the A* score
  - Score = cost of the path + heuristics of the node
- Candidate node: E(5)
- Frontier: G3(0)
- Visited: S(5), A(12),B(11), D(12), C(12), E(13)
- The optimal path is the lowest score of the goal states.



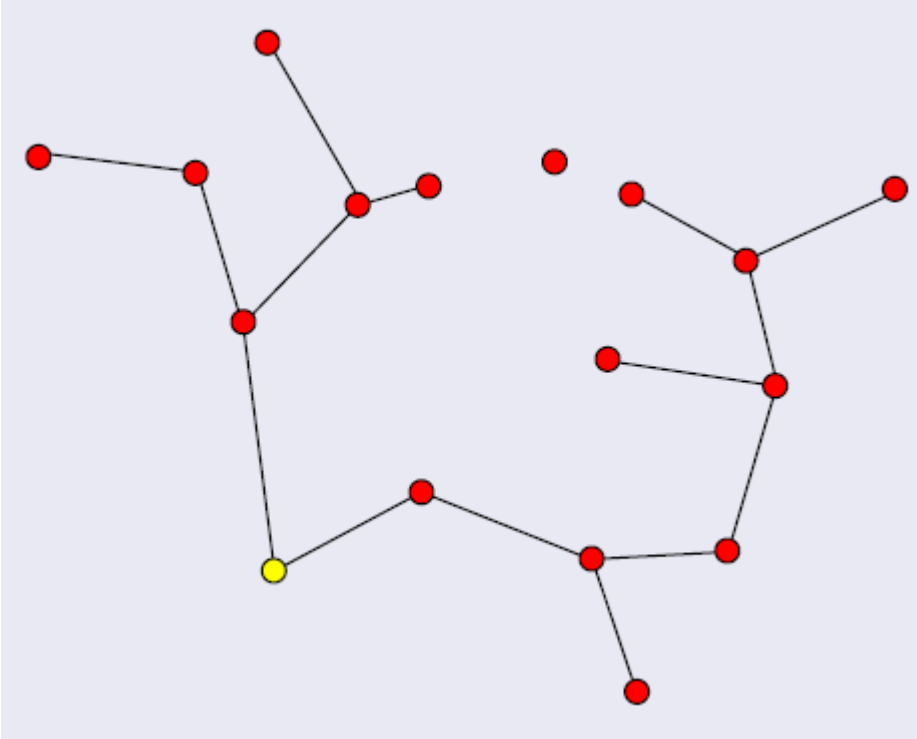
RRT*

- RRT* essentially “rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRT.



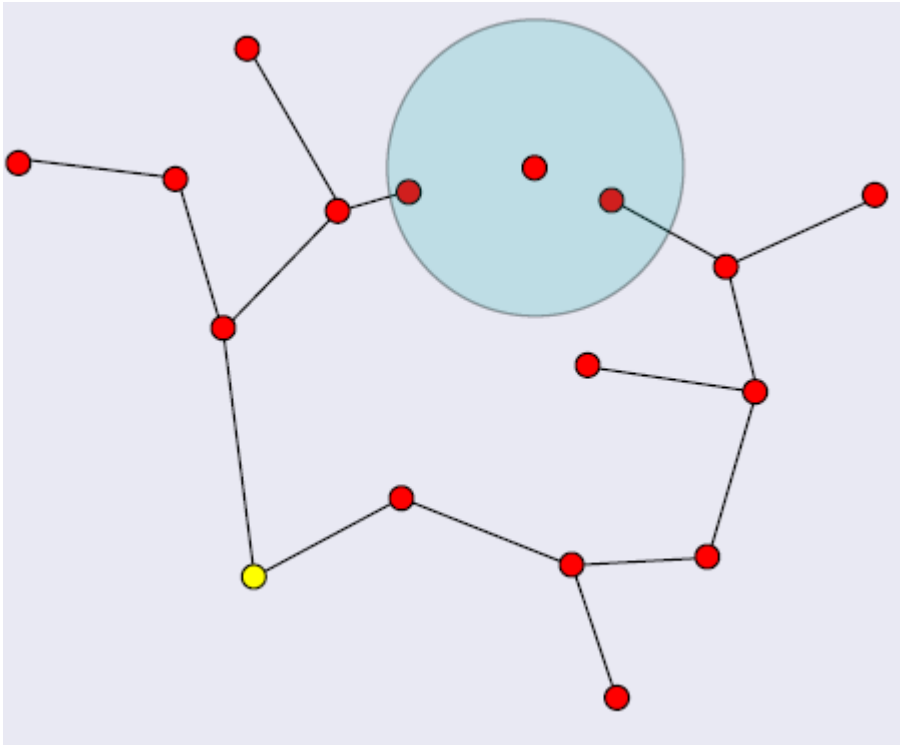
RRT*

- RRT* essentially “rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRT.



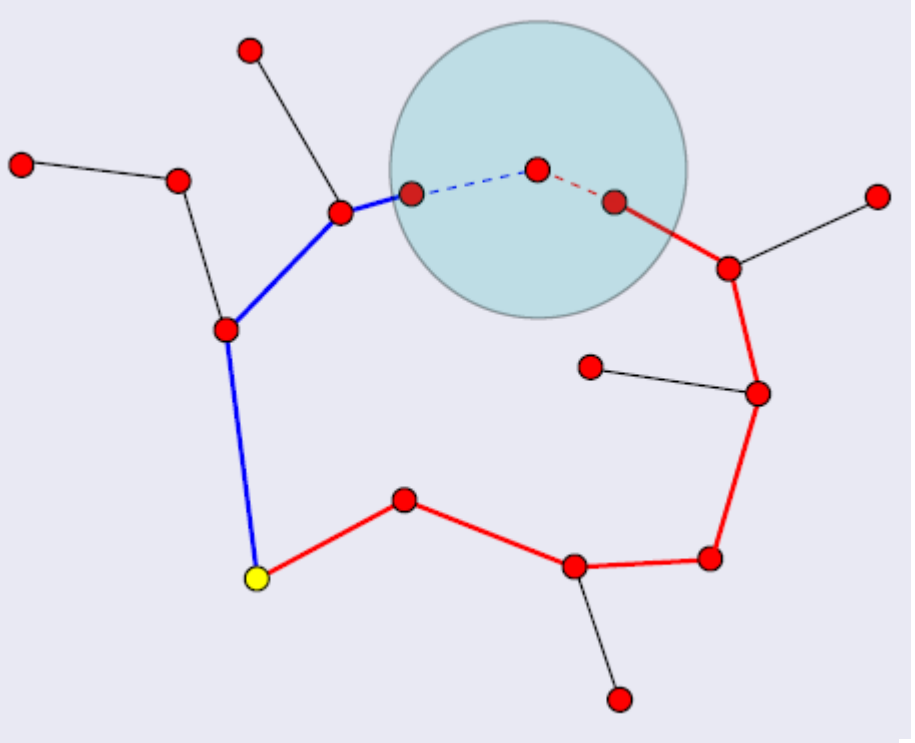
RRT*

- RRT* essentially “rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRT.



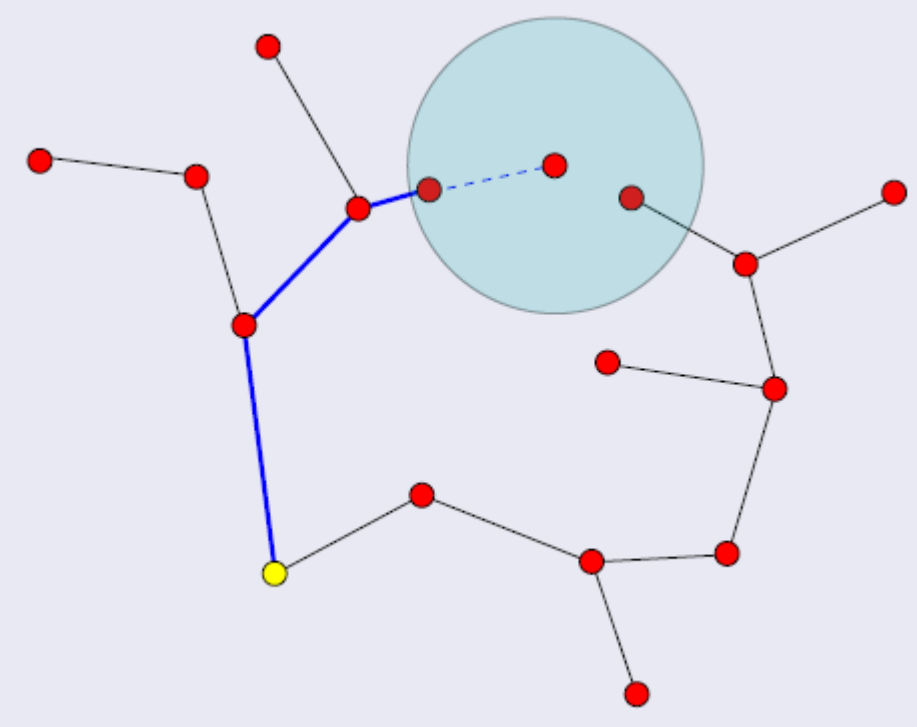
RRT*

- RRT* essentially “rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRT.



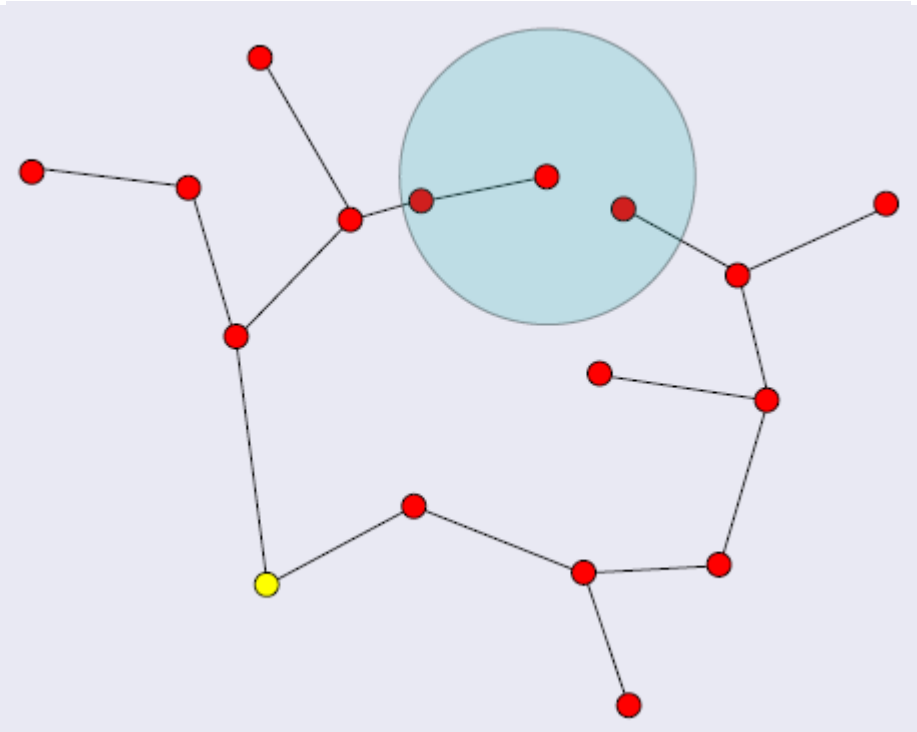
RRT*

- RRT* essentially “rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRT.



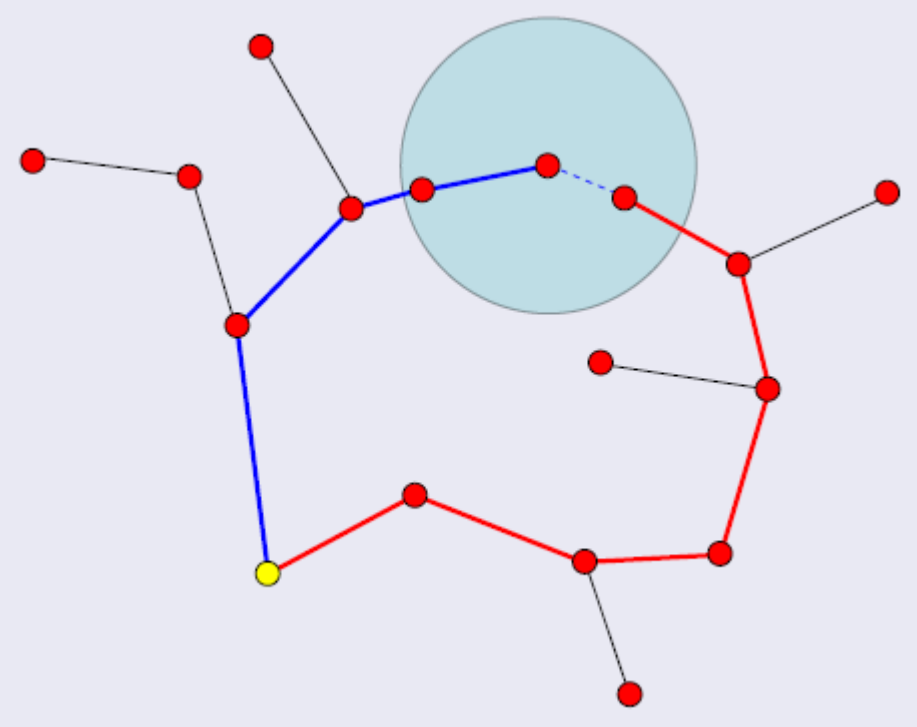
RRT*

- RRT* essentially “rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRT.



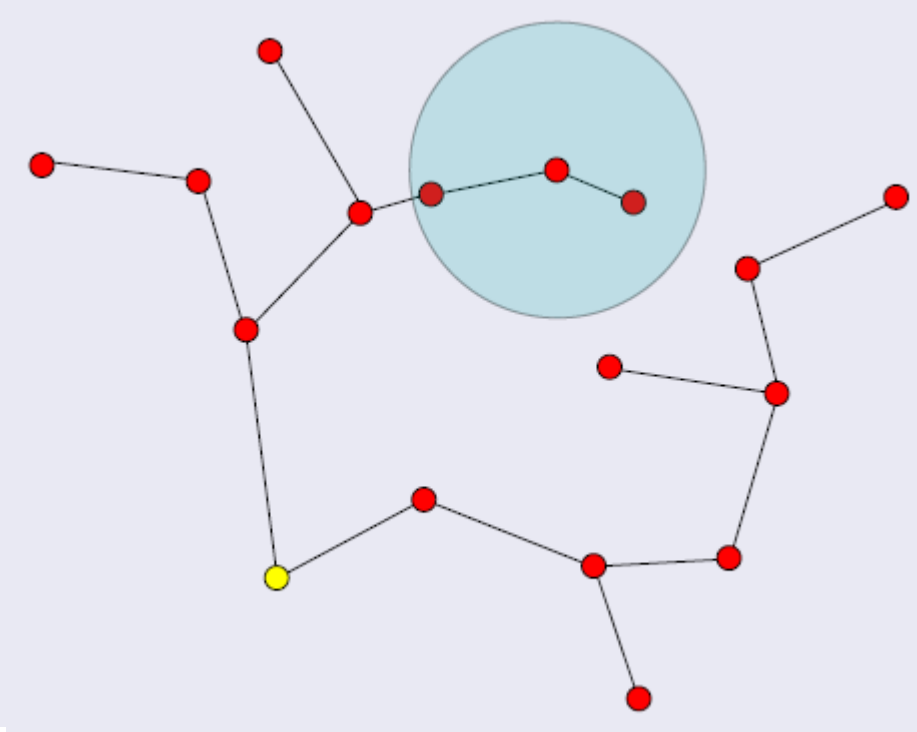
RRT*

- RRT* essentially “rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRT.



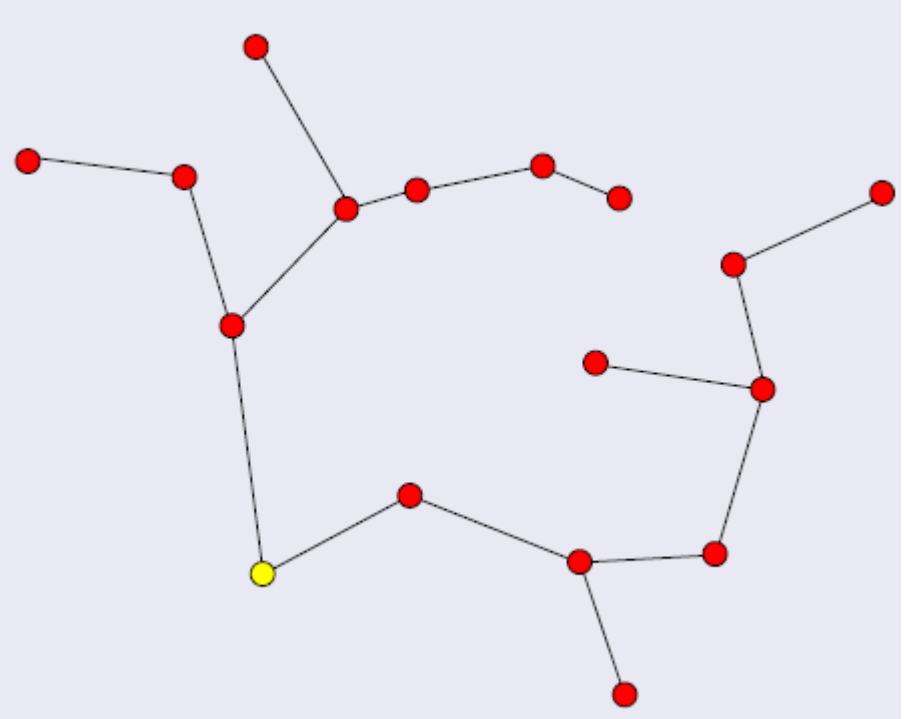
RRT*

- RRT* essentially “rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRT.



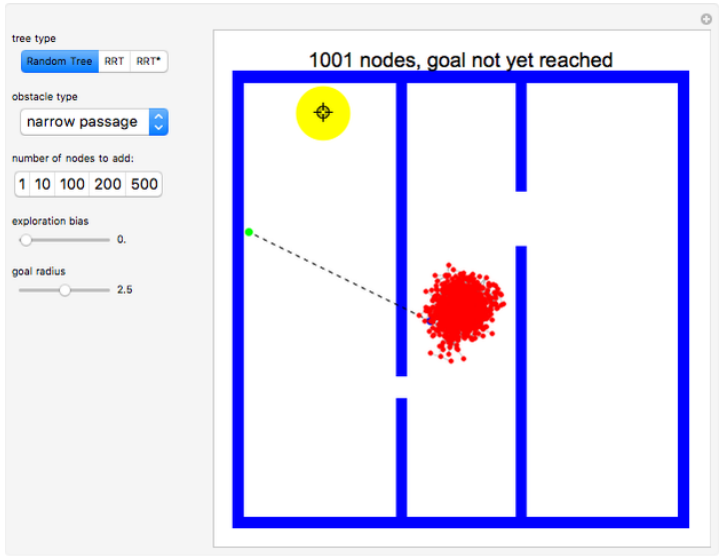
RRT*

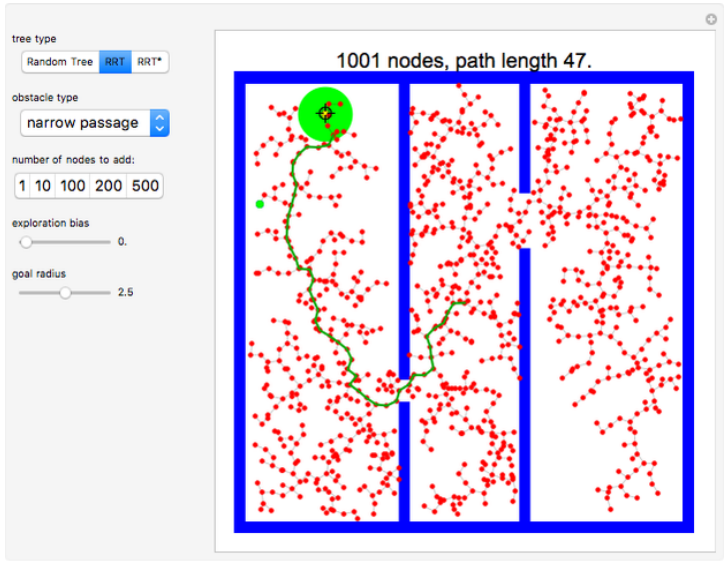
- RRT* essentially “rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRT.

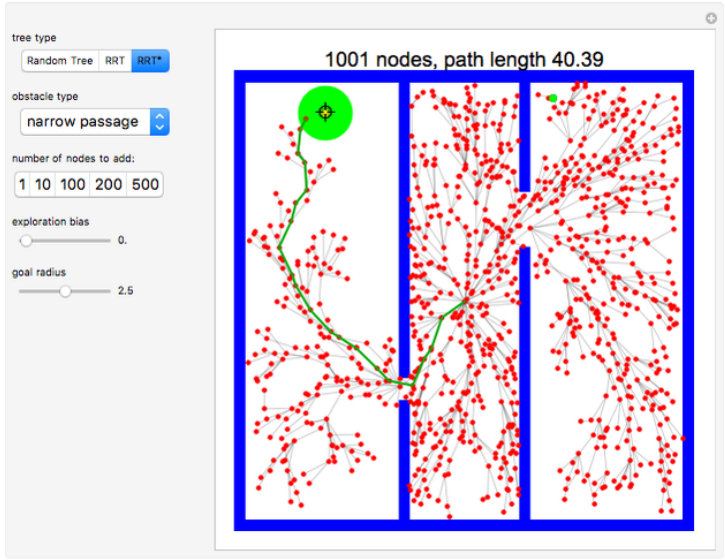


Example

- let's have a look at the the following example:
- https://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/



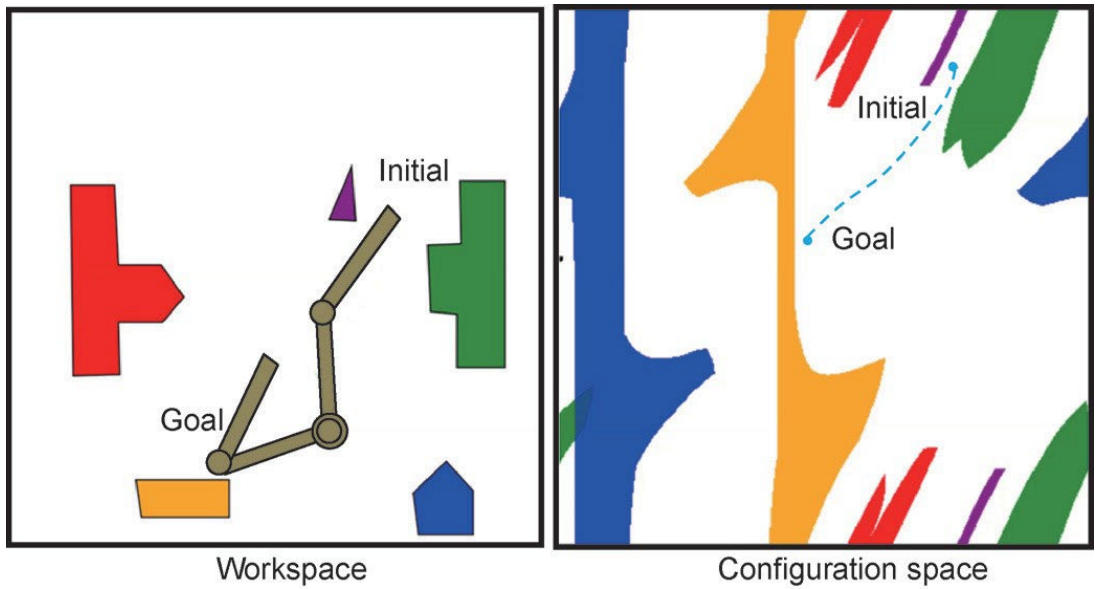




Roadmaps

Introduction

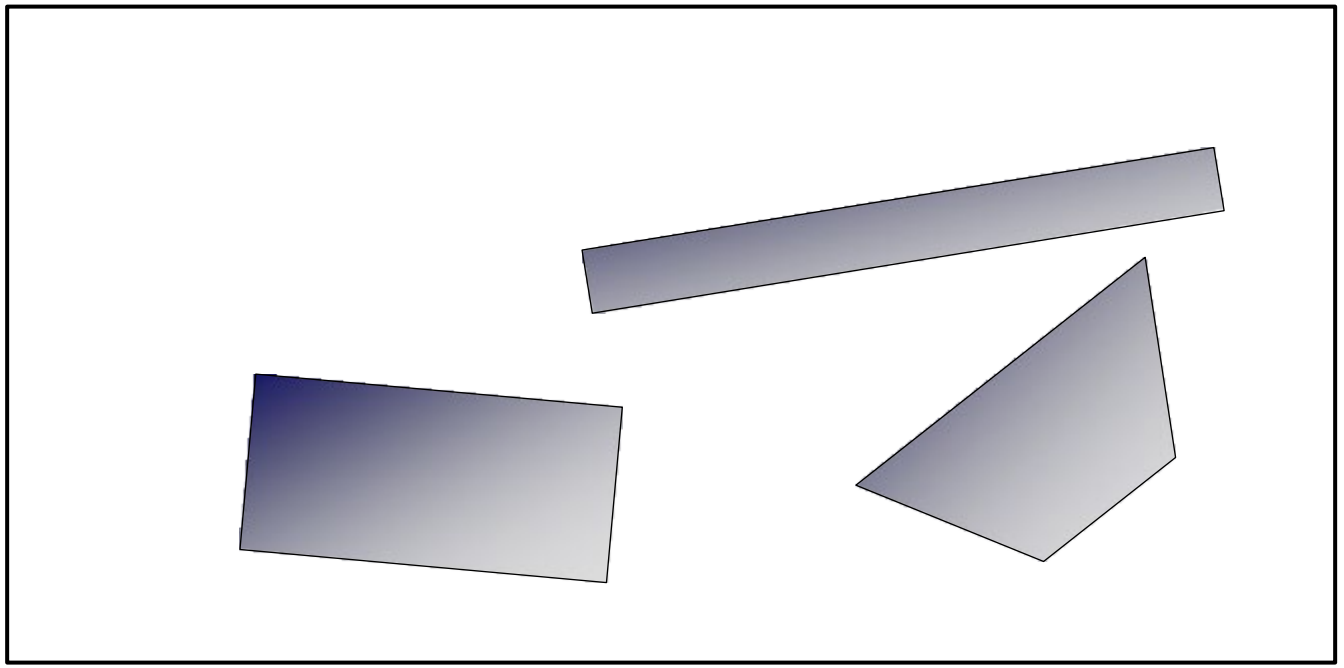
- Motion planning in high-dimensional configuration spaces is complex, and hence slow. Thus, for high-dimensional problems it is not appropriate to generate a description of the free configuration space (C_{free})



Alternative: Approximate the free configuration space by random sampling. This is accomplished with ***Probabilistic Roadmaps*** (PRM).

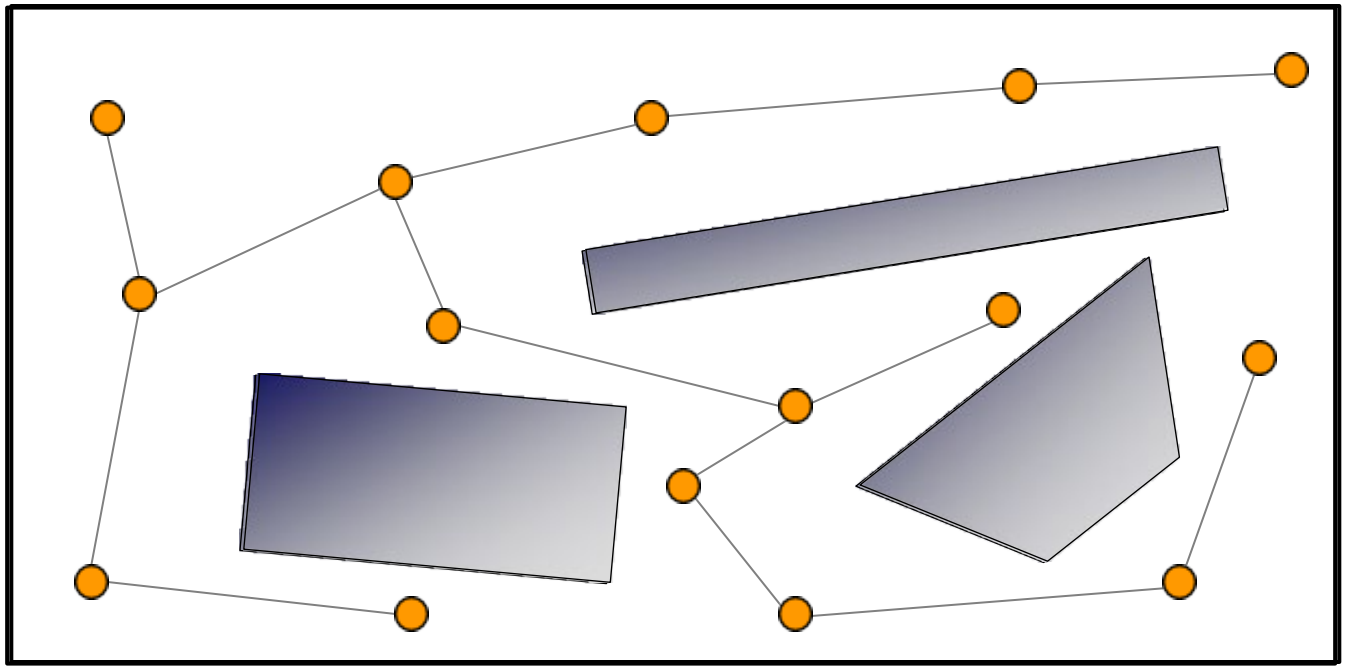
Introduction – idea behind the sampling based motion planning

Rather than describing the free configuration space, a roadmap is provided that will allow motion planning.



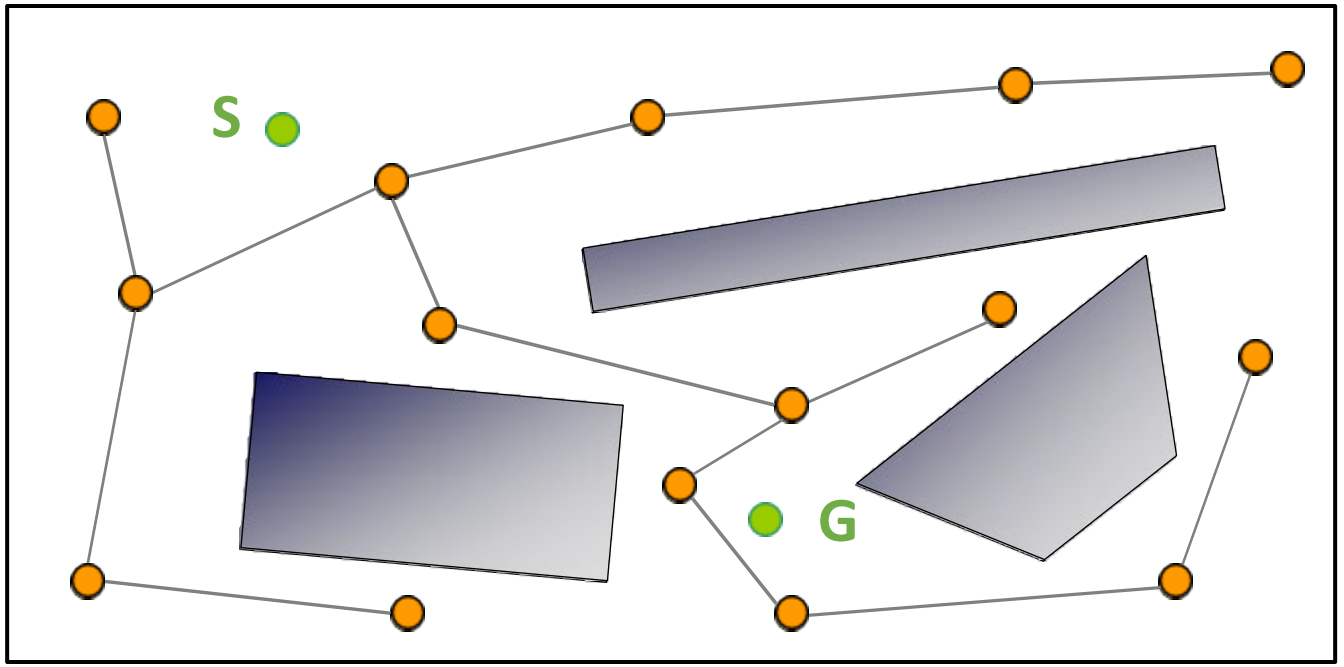
Introduction – idea behind the sampling based motion planning

Rather than describing the free configuration space, a roadmap is provided that will allow motion planning.



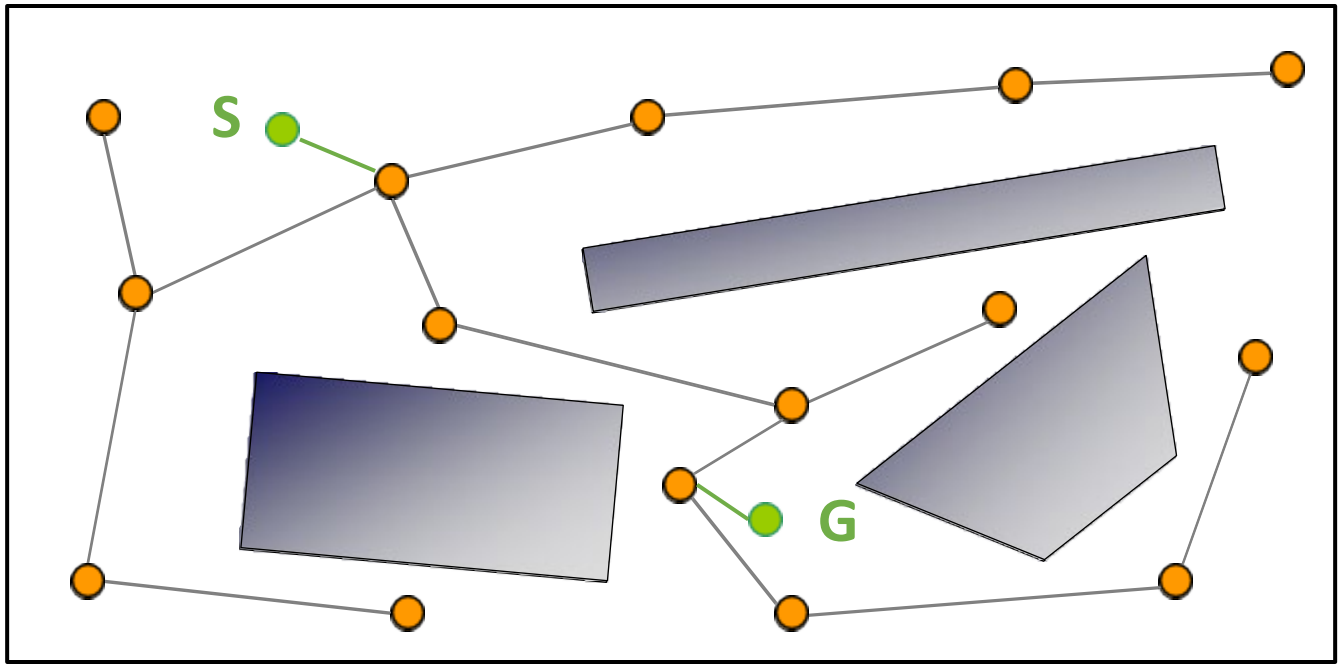
Introduction – idea behind the sampling based motion planning

Rather than describing the free configuration space, a roadmap is provided that will allow motion planning.



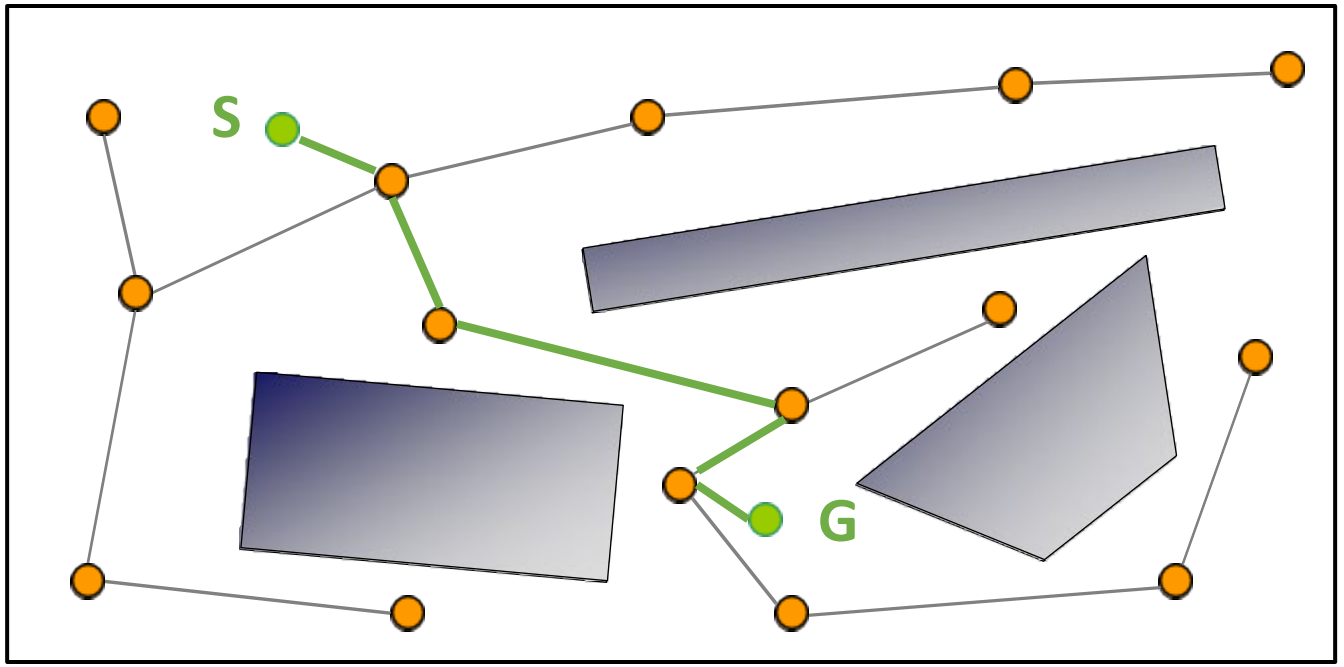
Introduction – idea behind the sampling based motion planning

Rather than describing the free configuration space, a roadmap is provided that will allow motion planning.



Introduction – idea behind the sampling based motion planning

Rather than describing the free configuration space, a roadmap is provided that will allow motion planning.



Introduction – problem formulation

Problem (Single Query Planner)

Given a robot with n degrees of freedom, a start configuration $q_{init} \in C_{free}$*, and a goal configuration* $q_{goal} \in C_{free}$*. Find a path P between* q_{init} *and* q_{goal} *such that* $P \subset C_{free}$.

Problem (Multi Query Planner)

Given a robot with n degrees of freedom and a configuration space $C = C_{free} \cup C_{obstacle}$*. Create a data structure for efficiently finding paths* $P_i \subset C_{free}$ *between* $q_{init,\,i} \in C_{free}$ *and* $q_{goal,\,i} \in C_{free}$ $for\ i = 1, 2, \ldots, N$.

Roadmaps

If we are moving a robot in an environment multiple times (Multi query), then it makes sense to create a ***map*** of the environment. Based on the map, one can generate paths more quickly.

Maps a categorized as

- **Topological**: Graph like structure, possibly augmented with metric information,
- **Geometric**: Geometric primitives such as line segments and triangle meshes.
- **Grids**: Occupancy grids.

We only address topological maps; in particular, ***roadmaps***.

Roadmaps - definition

A roadmap is a topological map, i.e., a graph where vertices corresponds to locations, and edges corresponds to paths between locations.

An ***undirected graph*** is a pair $G = (V, E)$, where $V = \{v_1, v_1, \ldots, v_n\}$ is a nonempty set of nodes (or vertices), and $E \subseteq V \times V$ is a set of edges such that

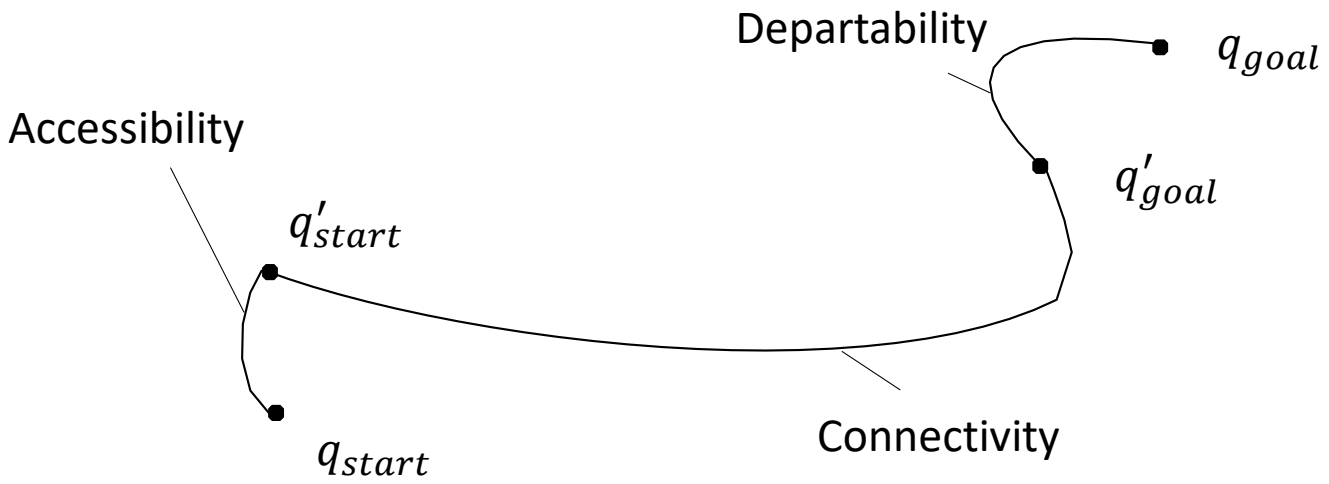
$$(v_i, v_j) \in E \;\Rightarrow\; (v_j, v_i) \in E$$

The edge (v_i, v_j) is denoted e_{ij}.

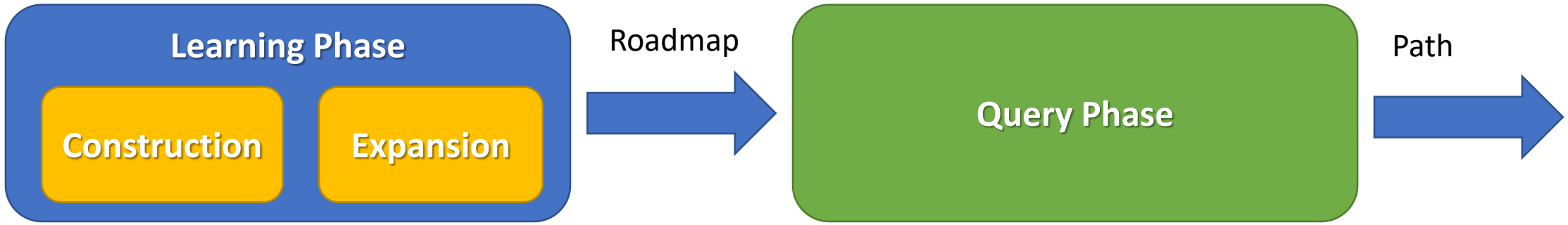
Roadmaps - definition

A union of one-dimensional curves is a ***roadmap*** (RM) if for all q_{start} and q_{goal} in C_{free} that can be connected by a path, the following properties hold

1. **Accessibility**: there exists a path from $q_{start} \in C_{free}$ to some $q'_{start} \in RM$
2. **Departability**: there exists a path from some $q'_{goal} \in RM$ to $q_{goal} \in C_{free}$
3. **Connectivity**: there exists a path in $RM_{q'_{start}}$ between q'_{start} and q'_{goal}



Probabilistic Roadmaps



- It is expensive to generate roadmap
- If multiple queries are made, can speed up the planning in the runtime phase.

Probabilistic roadmaps - representation

The roadmap is an *undirected graph* $R = (N, E)$, where

- N is a set of robot configurations belonging to the *free C-Space*
- E is a set of paths, where an edge (a, b) corresponds to a feasible path connecting the configurations a and b.

A ***deterministic local planner*** is applied to avoid storing the local path between pairs of configurations.

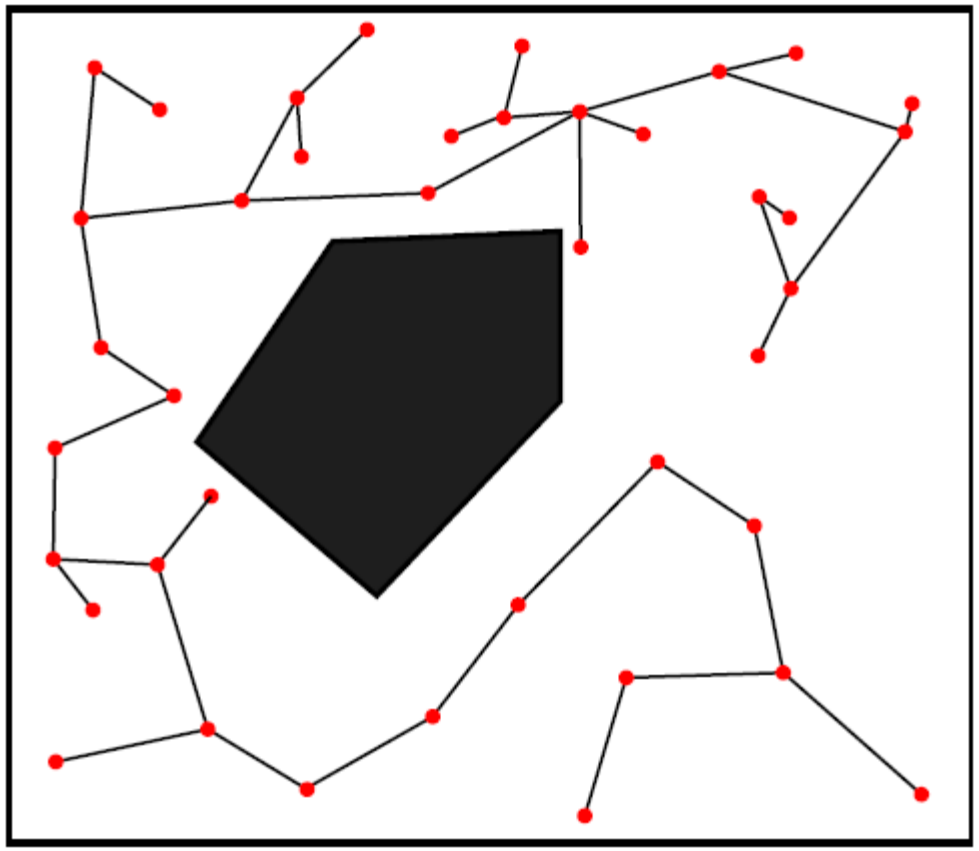
Only the *general local planner* (straight line in C-Space) is considered in this lecture.

Probabilistic roadmaps – Learning phase

Algorithm (Construction)

Input: *Pseudo-metric* $D : \mathcal{C} \times \mathcal{C} \to \mathbb{R}_+ \cup \{0\}$, *threshold* $\delta > 0$, $i_{\max} \in \mathbb{N}$
Output: *Roadmap* R
Procedure:

$N \leftarrow \emptyset$, $E \leftarrow \emptyset$, $i \leftarrow 0$
while $i < i_{\max}$ ***do***
  $c \leftarrow \mathit{Rand}(\mathcal{C}_{\text{free}})$
  $N_c \leftarrow \{n \in N |\ D(c,n) < \delta\}$
  $N = N \cup \{c\}$
  for $n \in N_c$ *(in order of inc.* $D(c,n)$*)* ***do***
    if $\nexists \mathit{Path}(c,n) \wedge \mathit{LP}[c,n] \in \mathcal{C}_{\text{free}}$ ***then***
      $E \leftarrow E \cup \{(c,n)\}$
    end if
  end for
  $i \leftarrow i + 1$
end while
return $R = (N, E)$



Probabilistic roadmaps – Learning phase

generate random configurations

Algorithm (Construction)

Input: *Pseudo-metric* $D : \mathcal{C} \times \mathcal{C} \to \mathbb{R}_+ \cup \{0\}$, *threshold* $\delta > 0$

Output: *Roadmap* R

Procedure:

1: $N \leftarrow \emptyset$, $E \leftarrow \emptyset$
2: **while** $i < i_{\max}$ **do**
3: $c \leftarrow Rand(\mathcal{C}_{free})$
4: $N_c \leftarrow \{n \in N | \ D(c,n) < \delta\}$
5: $N = N \cup \{c\}$
6: **for** $n \in N_c$ *(in order of inc.* $D(c,n)$*)* **do**
7: **if** $\nexists Path(c,n) \wedge LP[c,n] \in \mathcal{C}_{free}$ **then**
8: $E \leftarrow E \cup \{(c,n)\}$
9: **end if**
10: **end for**
11: $i \leftarrow i + 1$
12: **end while**
return $R = (N, E)$

Random collision-free configurations are generated as

1: **loop**
2: c← $Rand(C)$
3: **if** $c \in C_{free}$ **then**
return c
4: **end if**
5: **end loop**

The function $Rand(C)$ generates random numbers uniformly distributed on C.

Probabilistic roadmaps – Learning phase

local planner

Algorithm (Construction)

Input: *Pseudo-metric* $D : \mathcal{C} \times \mathcal{C} \to \mathbb{R}_+ \cup \{0\}$, *threshold* $\delta > 0$
Output: *Roadmap* R
Procedure:

$N \leftarrow \emptyset, E \leftarrow \emptyset$
while $i < i_{\max}$ ***do***
  $c \leftarrow \mathtt{Rand}(\mathcal{C}_{\text{free}})$
  $N_c \leftarrow \{n \in N |\ D(c,n) < \delta\}$
  $N = N \cup \{c\}$
  for $n \in N_c$ *(in order of inc.* $D(c,n)$*)* ***do***
    if $\nexists \mathtt{Path}(c,n) \wedge \mathtt{LP}[c,n] \in \mathcal{C}_{\text{free}}$ ***then***
      $E \leftarrow E \cup \{(c,n)\}$
    end if
  end for
  $i \leftarrow i + 1$
end while
return $R = (N, E)$

- The choice of local planner should consider
  - System class (holonomic (constraints on configuration) vs. nonholonomic (other constraints e.g., vel.))
  - Nondeterministic vs. deterministic
  - Speed
  - Memory space
- The *general local planner* is considered (simple e.g., straight path).

Probabilistic roadmaps – Learning phase

Neighbors

Algorithm (Construction)

Input: *Pseudo-metric* $D : \mathcal{C} \times \mathcal{C} \to \mathbb{R}_+ \cup \{0\}$, *threshold* $\delta > 0$
Output: *Roadmap* R
Procedure:

1: $N \leftarrow \emptyset, E \leftarrow \emptyset$
2: **while** $i < i_{\max}$ **do**
3: $c \leftarrow \mathtt{Rand}(\mathcal{C}_{\text{free}})$
4: $N_c \leftarrow \{n \in N \mid D(c,n) < \delta\}$
5: $N = N \cup \{c\}$
6: **for** $n \in N_c$ *(in order of inc.* $D(c,n)$*)* **do**
7: **if** $\nexists \mathtt{Path}(c,n) \wedge \mathtt{LP}[c,n] \in \mathcal{C}_{\text{free}}$ **then**
8: $E \leftarrow E \cup \{(c,n)\}$
9: **end if**
10: **end for**
11: $i \leftarrow i + 1$
12: **end while**
return $R = (N, E)$

Assumption: The likelihood of a local path planner finding a collision-free path between two nodes is reverse proportional to the distance between the nodes.

Remark: The number of nodes in Nc may also be restricted to have an upper bound on the execution time of each iteration of the algorithm. This also makes the algorithm independent of the size of the roadmap.

Probabilistic roadmaps – Learning phase

Neighbors

Algorithm (Construction)

Input: *Pseudo-metric* $D : \mathcal{C} \times \mathcal{C} \to \mathbb{R}_+ \cup \{0\}$, *threshold* $\delta > 0$
Output: *Roadmap* R
Procedure:

$N \leftarrow \emptyset, E \leftarrow \emptyset$
while $i < i_{\max}$ **do**
  $c \leftarrow \mathtt{Rand}(\mathcal{C}_{\text{free}})$
  $N_c \leftarrow \{n \in N | \ D(c,n) < \delta\}$
  $N = N \cup \{c\}$
  for $n \in N_c$ *(in order of inc.* $D(c,n)$*)* **do**
    if $\nexists \mathtt{Path}(c,n) \wedge \mathtt{LP}[c,n] \in \mathcal{C}_{\text{free}}$ **then**
      $E \leftarrow E \cup \{(c,n)\}$
    end if
  end for
  $i \leftarrow i + 1$
end while
return $R = (N, E)$

The distance function D is used to construct and sort the set N_c can be defined in many different ways. The distance should indicate the chance for the local planner to fail to compute a collision-free path. The distance function is defined as

$$D(c,n) = \max_{x \in X_{robot}} \|x(n) - x(c)\|$$

where x denotes a point on the robot, $x(c)$ is the position of x in the workspace when the robot is at configuration c, and $||\ \cdot\ ||$ is the Euclidian distance.

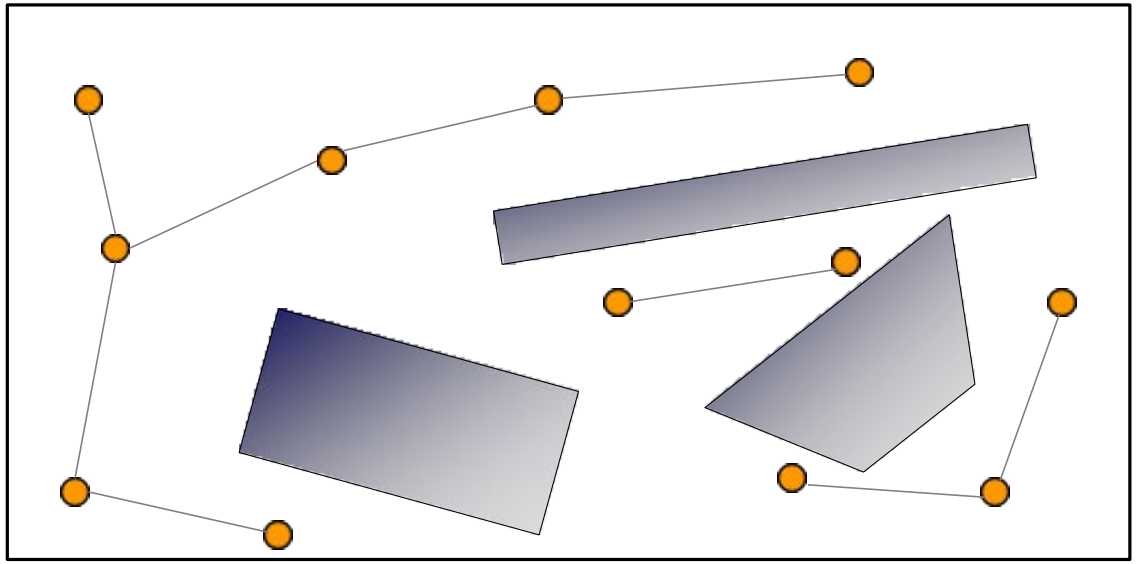
Probabilistic Roadmaps - Learning Phase

Expansion Algorithm - Overview *(From "Random Networks in Configuration Space for Fast Path Planning" by Lydia E. Kavraki, 1994)*

Algorithm (Expansion)

Input: *Roadmap* $R = (N, E)$, $j_{\max} \in \mathbb{N}$
Output: *Roadmap* $R = (N, E)$
Procedure:

1: *Compute the function* $w(x)$, $x \in N$
2: ***while*** $j < j_{\max}$ ***do***
3: $x \leftarrow \mathtt{Rand}_w(N)$
4: $c = \mathtt{Expand}(x)$
5: $N = N \cup \{c\}$
6: $E \leftarrow E \cup \{(c, x)\}$
7: $N_c \leftarrow \{n \in N |\ D(c, n) < \delta\}$
8: ***for*** $n \in N_c$ *(in order of inc.* $D(c, n)$*)* ***do***
9: ***if*** $\nexists \mathtt{Path}(c, n) \wedge \mathtt{LP}[c, n] \in \mathcal{C}_{\text{free}}$ ***then***
10: $E \leftarrow E \cup \{(c, n)\}$
11: ***end if***
12: ***end for***
13: $j \leftarrow j + 1$
14: ***end while***
return $R = (N, E)$



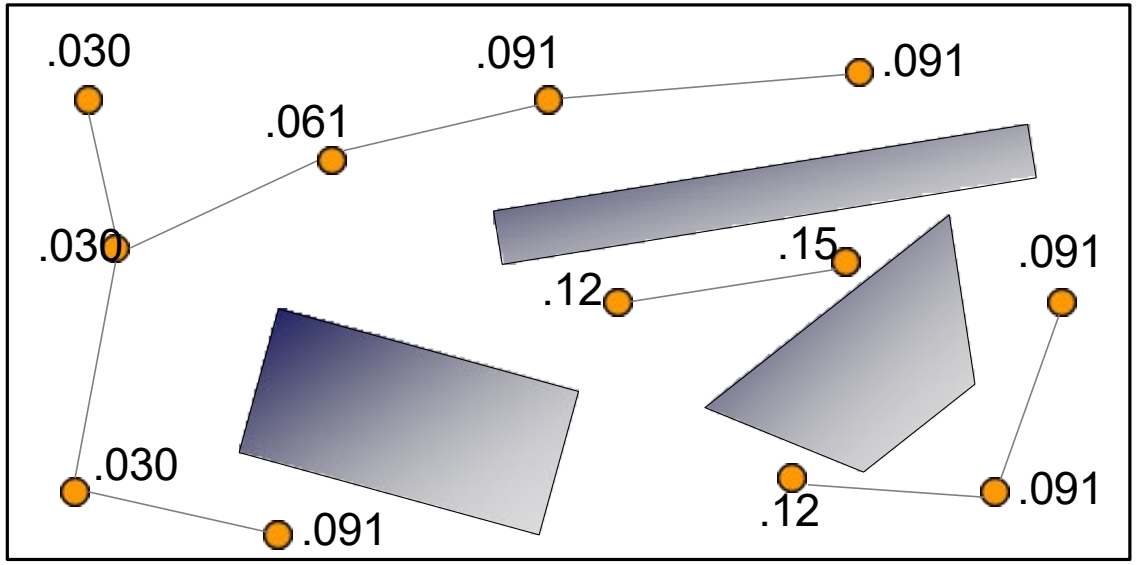
Probabilistic Roadmaps - Learning Phase

Expansion Algorithm - Overview *(From "Random Networks in Configuration Space for Fast Path Planning" by Lydia E. Kavraki, 1994)*

Algorithm (Expansion)

Input: Roadmap $R = (N, E)$, $j_{\max} \in \mathbb{N}$
Output: Roadmap $R = (N, E)$
Procedure:

1: Compute the function $w(x)$, $x \in N$
2: **while** $j < j_{\max}$ **do**
3: $x \leftarrow \mathit{Rand}_w(N)$
4: $c = \mathit{Expand}(x)$
5: $N = N \cup \{c\}$
6: $E \leftarrow E \cup \{(c, x)\}$
7: $N_c \leftarrow \{n \in N |\ D(c, n) < \delta\}$
8: **for** $n \in N_c$ (in order of inc. $D(c, n)$) **do**
9: **if** $\nexists \mathit{Path}(c, n) \wedge \mathit{LP}[c, n] \in \mathcal{C}_{\text{free}}$ **then**
10: $E \leftarrow E \cup \{(c, n)\}$
11: **end if**
12: **end for**
13: $j \leftarrow j + 1$
14: **end while**
return $R = (N, E)$



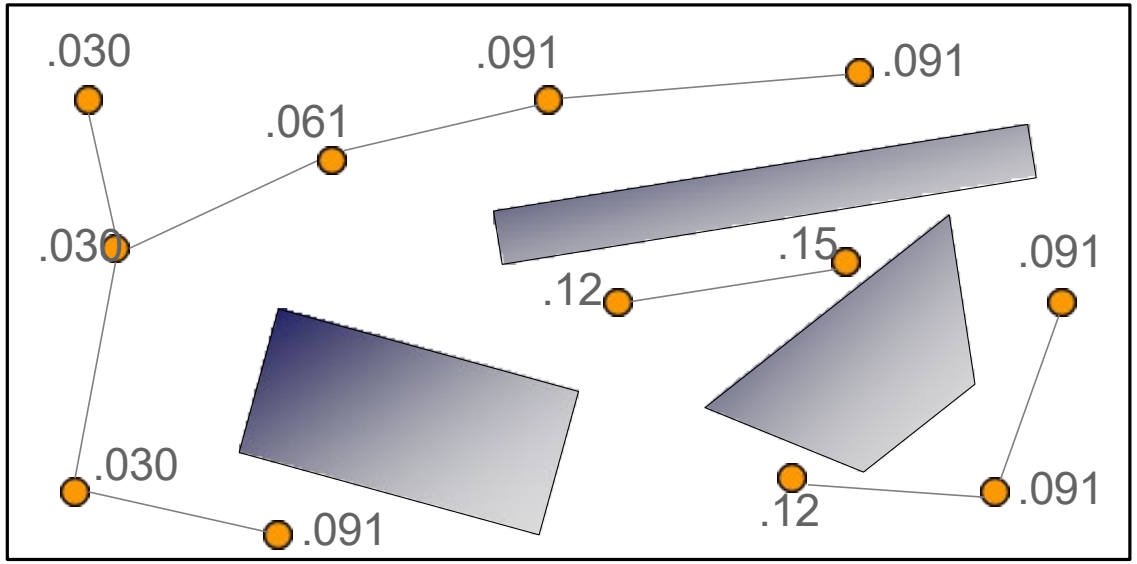
Probabilistic Roadmaps - Learning Phase

Expansion Algorithm - Overview *(From "Random Networks in Configuration Space for Fast Path Planning" by Lydia E. Kavraki, 1994)*

Algorithm (Expansion)

Input: *Roadmap* $R = (N, E)$, $j_{\max} \in \mathbb{N}$
Output: *Roadmap* $R = (N, E)$
Procedure:

1: *Compute the function* $w(x)$, $x \in N$
2: ***while*** $j < j_{\max}$ ***do***
3: $x \leftarrow \mathrm{Rand}_w(N)$
4: $c = \mathrm{Empand}(x)$
5: $N = N \cup \{c\}$
6: $E \leftarrow E \cup \{(c, x)\}$
7: $N_c \leftarrow \{n \in N \mid D(c, n) < \delta\}$
8: ***for*** $n \in N_c$ *(in order of inc.* $D(c, n)$*)* ***do***
9: ***if*** $\exists$ a $(c, n) \wedge [c, n] \in C_{free}$ ***then***
10: $E \leftarrow E \cup \{(c, n)\}$
11: ***end if***
12: ***end for***
13: $j \leftarrow j + 1$
14: ***end while***
return $R = (N, E)$



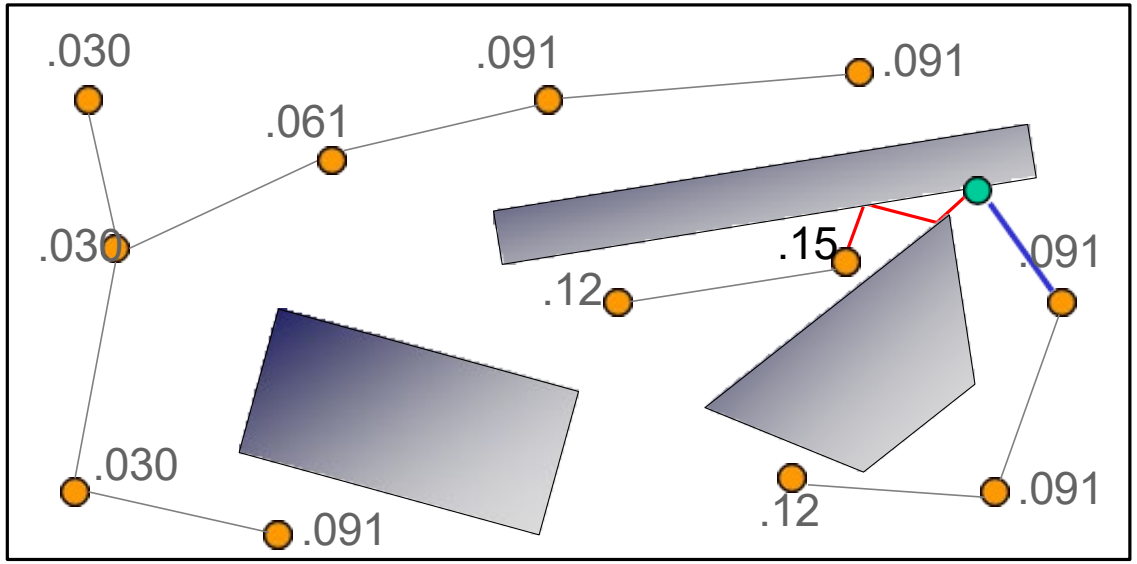
Probabilistic Roadmaps - Learning Phase

Expansion Algorithm - Overview *(From "Random Networks in Configuration Space for Fast Path Planning" by Lydia E. Kavraki, 1994)*

Algorithm (Expansion)

Input: Roadmap $R = (N, E)$, $j_{\max} \in \mathbb{N}$
Output: Roadmap $R = (N, E)$
Procedure:

1: Compute the function $w(x)$, $x \in N$
2: **while** $j < j_{\max}$ **do**
3: $x \leftarrow \mathtt{Rand}_w(N)$
4: $c = \mathtt{Expand}(x)$
5: $N = N \cup \{c\}$
6: $E \leftarrow E \cup \{(c, x)\}$
7: $N_c \leftarrow \{n \in N |\ D(c, n) < \delta\}$
8: **for** $n \in N_c$ (in order of inc. $D(c, n)$) **do**
9: **if** $\nexists \mathtt{Path}(c, n) \wedge \mathtt{LP}[c, n] \in \mathcal{C}_{\text{free}}$ **then**
10: $E \leftarrow E \cup \{(c, n)\}$
11: **end if**
12: **end for**
13: $j \leftarrow j + 1$
14: **end while**
return $R = (N, E)$



Probabilistic Roadmaps - Learning Phase

Expansion Algorithm – Random Node

Algorithm (Expansion)

Input: *Roadmap* $R = (N, E)$, $j_{\max} \in \mathbb{N}$
Output: *Roadmap* $R = (N, E)$
Procedure:

1: *Compute the function* $w(x)$, $x \in N$
2: ***while*** $j < j_{\max}$ ***do***
3: $x \leftarrow \mathit{Rand}_w(N)$
4: $c = \mathit{Expand}(x)$
5: $N = N \cup \{c\}$
6: $E \leftarrow E \cup \{(c, x)\}$
7: $N_c \leftarrow \{n \in N |\ D(c, n) < \delta\}$
8: ***for*** $n \in N_c$ *(in order of inc.* $D(c, n)$*)* ***do***
9: ***if*** $\nexists \mathit{Path}(c, n) \wedge \mathit{LP}[c, n] \in \mathcal{C}_{\text{free}}$ ***then***
10: $E \leftarrow E \cup \{(c, n)\}$
11: ***end if***
12: ***end for***
13: $j \leftarrow j + 1$
14: ***end while***
return $R = (N, E)$

The function $\mathrm{Rand}_w(N)$ will select a node $x \in N$ such that

$$Pr(x \text{ is selected}) = w(x)$$

Probabilistic Roadmaps - Learning Phase

Expansion Algorithm – Weight Function

Algorithm (Expansion)

Input: *Roadmap* $R = (N, E)$, $j_{\max} \in \mathbb{N}$
Output: *Roadmap* $R = (N, E)$
Procedure:

1: *Compute the function* $w(x)$, $x \in N$
2: **while** $j < j_{\max}$ **do**
3: $x \leftarrow \mathtt{Rand}_w(N)$
4: $c = \mathtt{Expand}(x)$ *(store path)*
5: $N = N \cup \{c\}$
6: $E \leftarrow E \cup \{(c, x)\}$
7: $N_c \leftarrow \{n \in N |\ D(c, n) < \delta\}$
8: **for** $n \in N_c$ *(in order of inc.* $D(c, n)$*)* **do**
9: **if** $\nexists \mathtt{Path}(c, n) \wedge \mathtt{LP}[c, n] \in \mathcal{C}_{\text{free}}$ **then**
10: $E \leftarrow E \cup \{(c, n)\}$
11: **end if**
12: **end for**
13: $j \leftarrow j + 1$
14: **end while**
return $R = (N, E)$

One way of defining w is by

$$w(c) = \frac{r_f(c)}{\sum_{a \in N} r_f(a)}$$

where r_f is the failure ratio defined as

$$r_f(c) = \frac{f(c)}{n(c) + 1}$$

where $n(c)$ is the total number of times the local planner tried to connect c to another node and $f(c)$ is the number of times it failed.

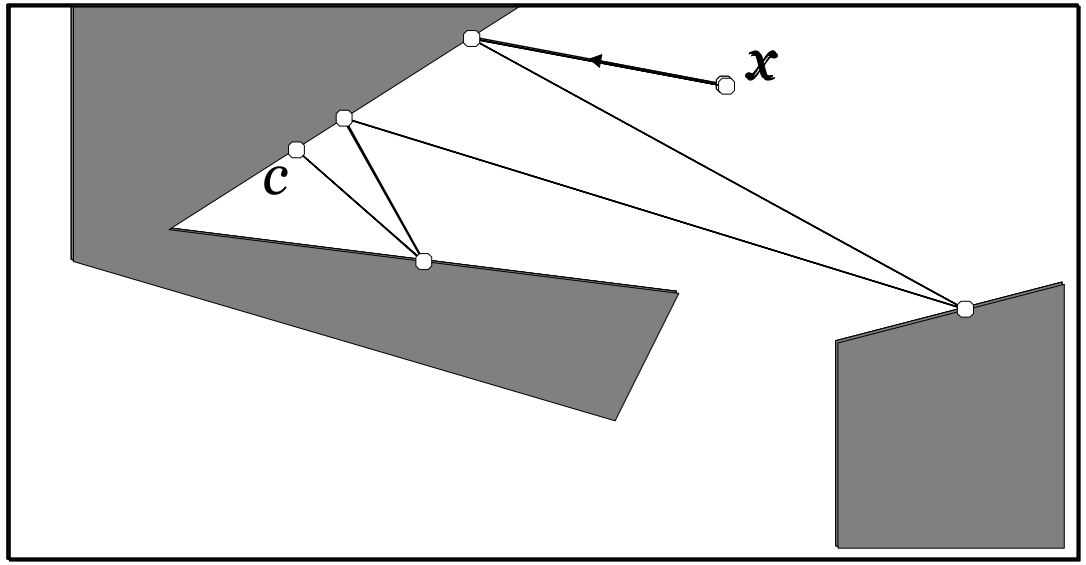
Probabilistic Roadmaps - Learning Phase

Expansion Algorithm – Expand

Algorithm (Expansion)

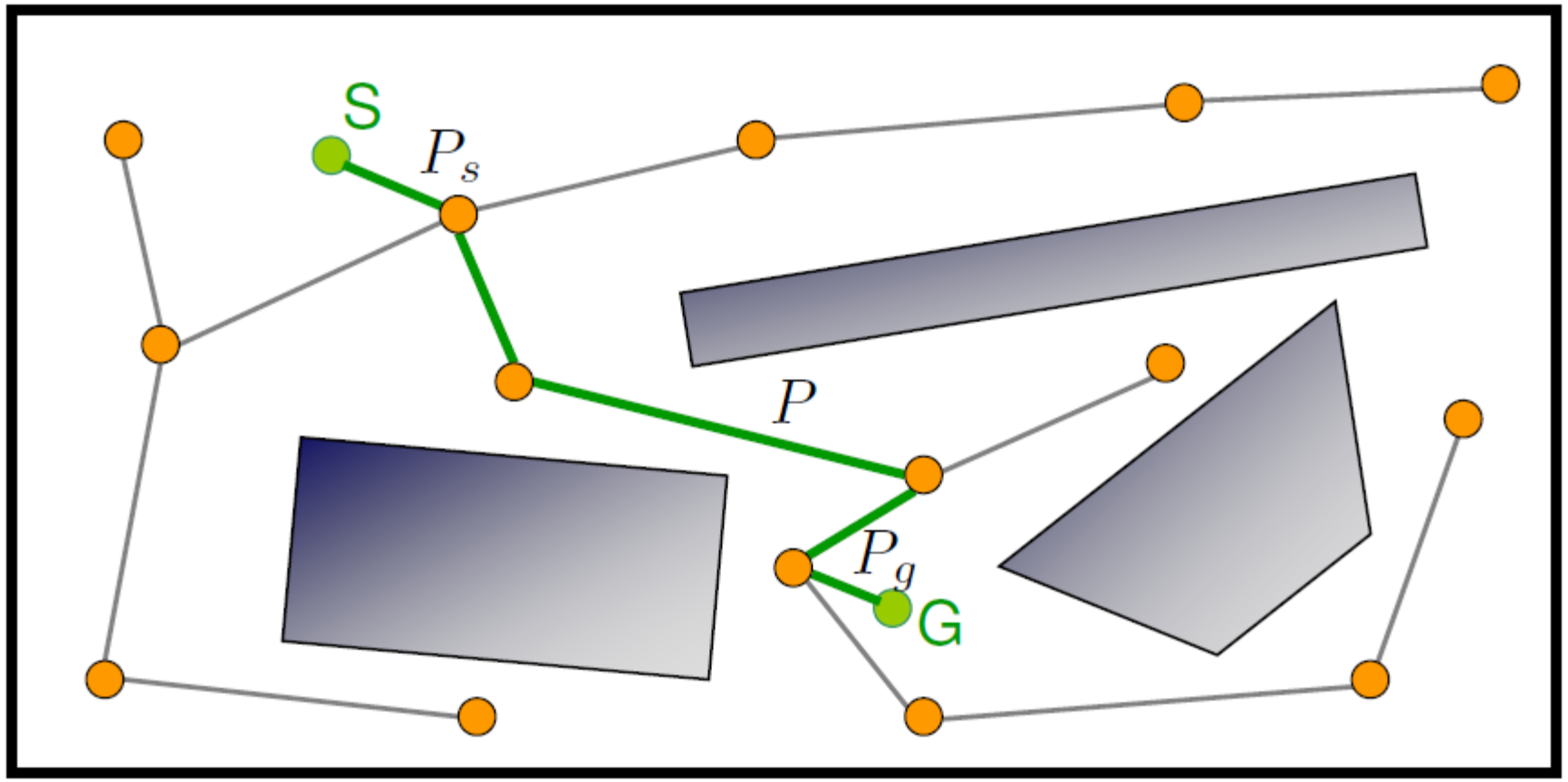
Input: *Roadmap* R
Output: *Roadmap* R
Procedure:

Compute the function $w(x)$ *for all* $x \in N$
while $j < j_{\max}$ **do**
  $x \leftarrow \mathtt{Rand}_w(N)$
  $c = \mathtt{Expand}(x)$
  $N = N \cup \{c\}$
  $E \leftarrow E \cup \{(c, x)\}$
  $N_c \leftarrow \{n \in N |\ D(c, n) < \delta\}$
  for $n \in N_c$ *(in order of inc.* $D(c, n)$*)* **do**
    if $\nexists \mathtt{Path}(c, n) \wedge \mathtt{LP}[c, n] \in \mathcal{C}_{\text{free}}$ **then**
      $E \leftarrow E \cup \{(c, n)\}$
    end if
  end for
  $j \leftarrow j + 1$
end while
return $R = (N, E)$



Probabilistic Roadmaps – Query Phase

Overview



Finding paths P_s, P, and P_g is accomplished in the query phase.

Probabilistic Roadmaps – Query Phase

Algorithm – Find Path from Query Point to Roadmap

Algorithm (Accessibility)

Input: *Roadmap* $R = (N, E)$, *configuration* c, *threshold* $\delta > 0$
Output: *Path* P_c
Procedure:

1: $i = 0$, $c_i = c$
2: **loop**
3: $N_{c_i} \leftarrow \{n \in N |\ D(c_i, n) < \delta\}$
4: **for** $n \in N_{c_i}$ *(in order of inc.* $D(c_i, n)$*)* **do**
5: **if** $\mathtt{LP}[c_i, n] \in \mathcal{C}_{\text{free}}$ **then**
6: **return** $P_c = \mathtt{LP}[c_i, n] \bigcup_{j=0}^{i-1} \mathtt{LP}[c_j, c_{j+1}]$
7: **else**
8: $c_{i+1} = \mathtt{RBW}(c_i)$
9: $i = i + 1$
10: **end if**
11: **end for**
12: **end loop**

Probabilistic Roadmaps – Query Phase

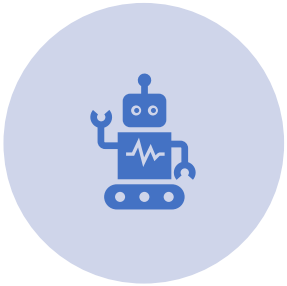
Completeness

- **Definition** (Probabilistic Completeness)
- A path planner is called probabilistically complete if, given a problem that is solvable, the probability that the planner solves the problem goes to 1 as the running time goes to infinity.

- **Proposition**
- The Probabilistic Roadmap is a probabilistically complete path planner.

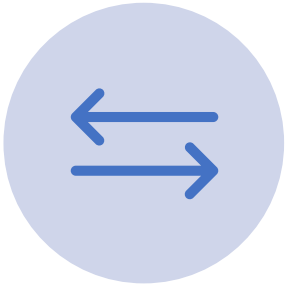
- And example can be found:
- https://demonstrations.wolfram.com/ProbabilisticRoadmapMethodForRobotArm/

Take home message



In configuration space obstacles are represented differently.

RRT connect outperforms the simple RRT algorithm.



Probabilistic roadmaps are probabilistic complete.

Probabilistic roadmaps work well as a multi query planner.