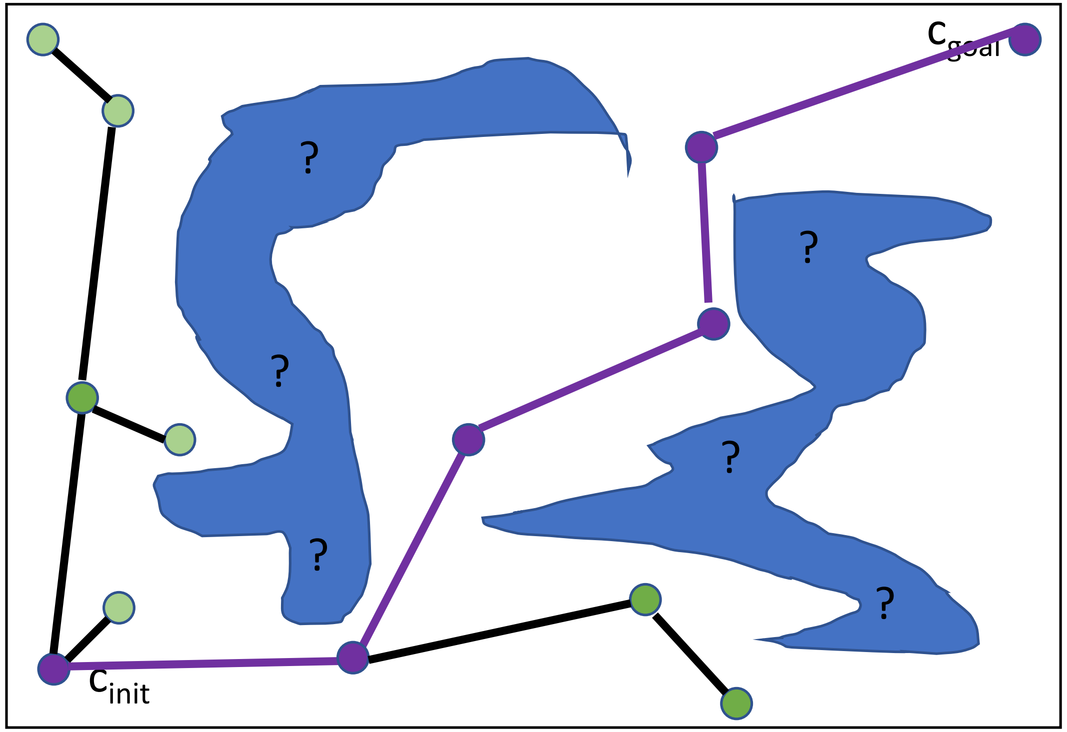
Path Optimization

Algorithms and Data Structures 2 – Motion Planning and its applications

University of Applied Sciences Stuttgart

Dr. Daniel Schneider

Motion Planning in practice



- Motion planning will return any path.
- One tries to use as little sample points as possible (performance).
- There is not criteria for optimal path given.

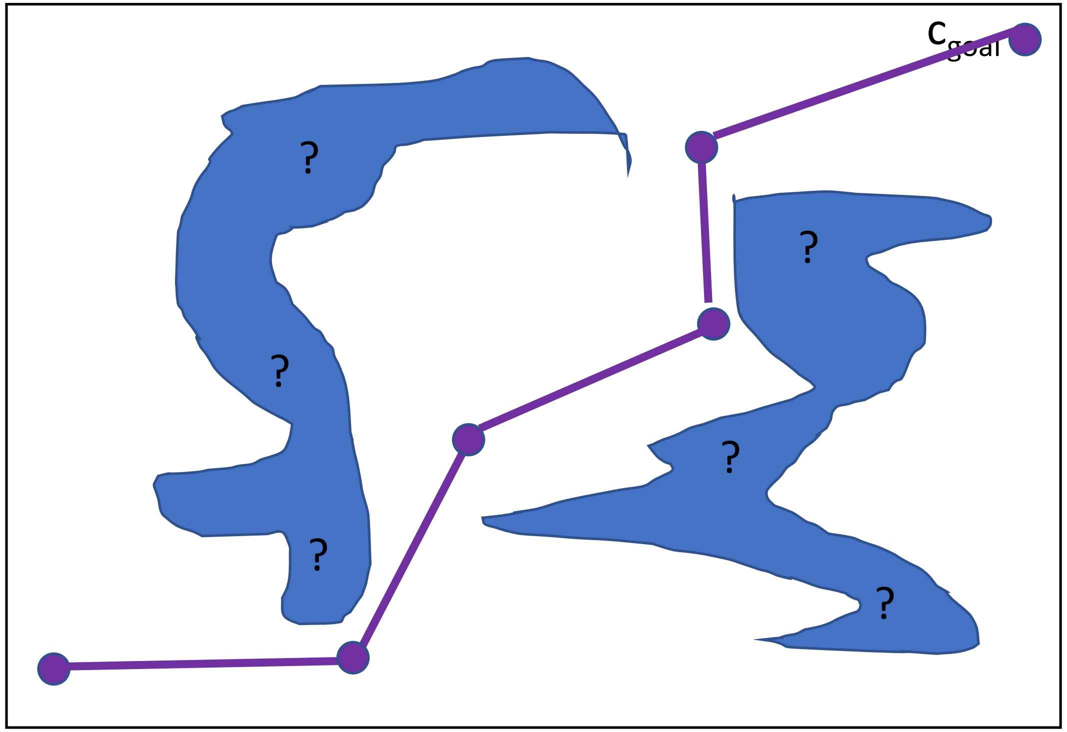
→ How to get more optimal paths?

What is optimal?

Some common cost functions:

- Find the shortest path (a short) for the robot from start to goal.
- Find a path that has the maximum clearance to its environment.
- Find a path that only uses little rotations.
- Find a path that limited the energy (e.g. the energy a “real” robot needs to execute a movement).
- …

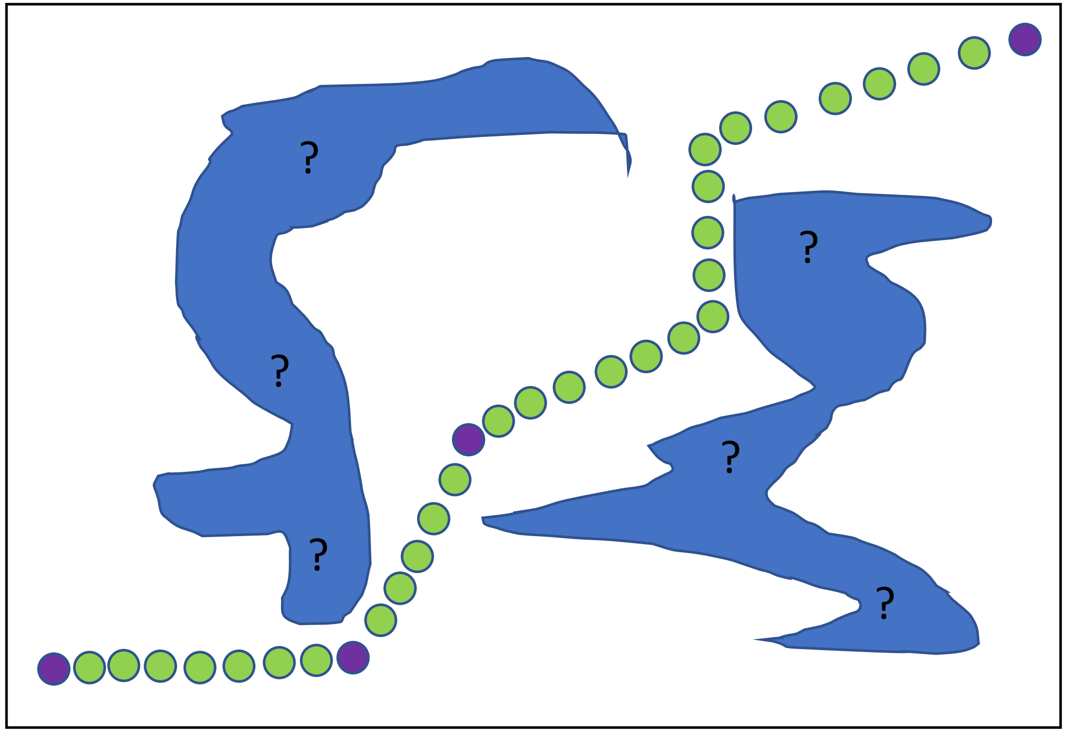
Some easy brute force approach – Path length



Goal:

Compute a shorter path.

Some easy brute force approach – Path length



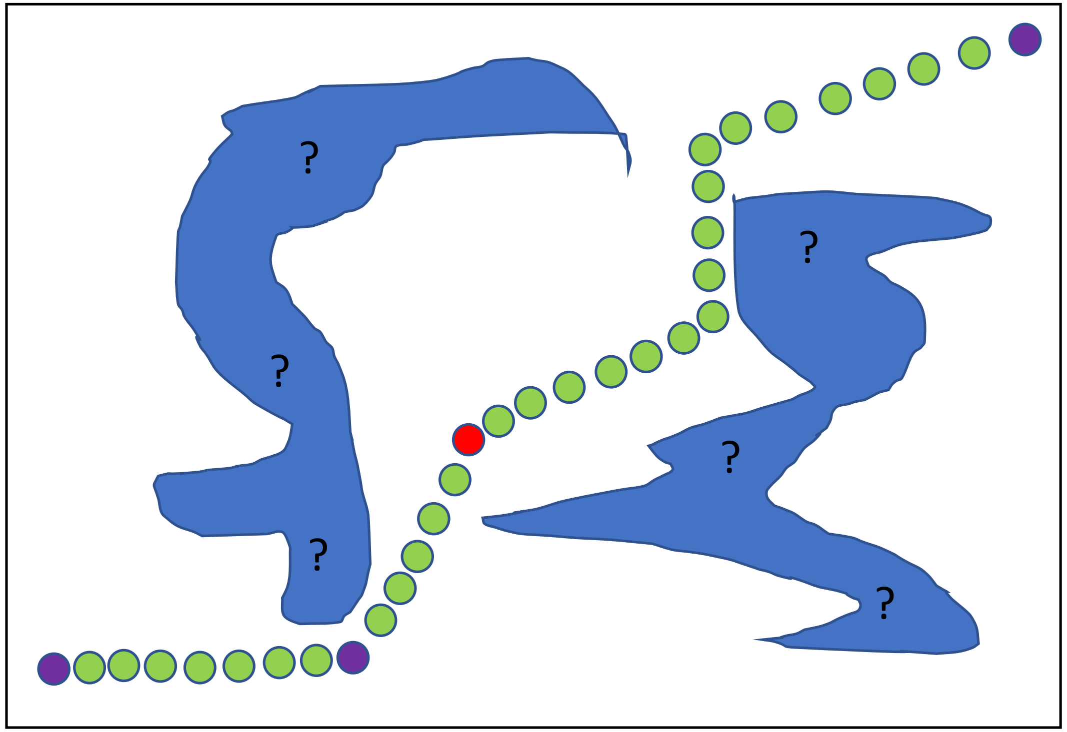
Goal:

Compute a shorter path.

Approach:

1. Sample the computed path.

Some easy brute force approach – Path length



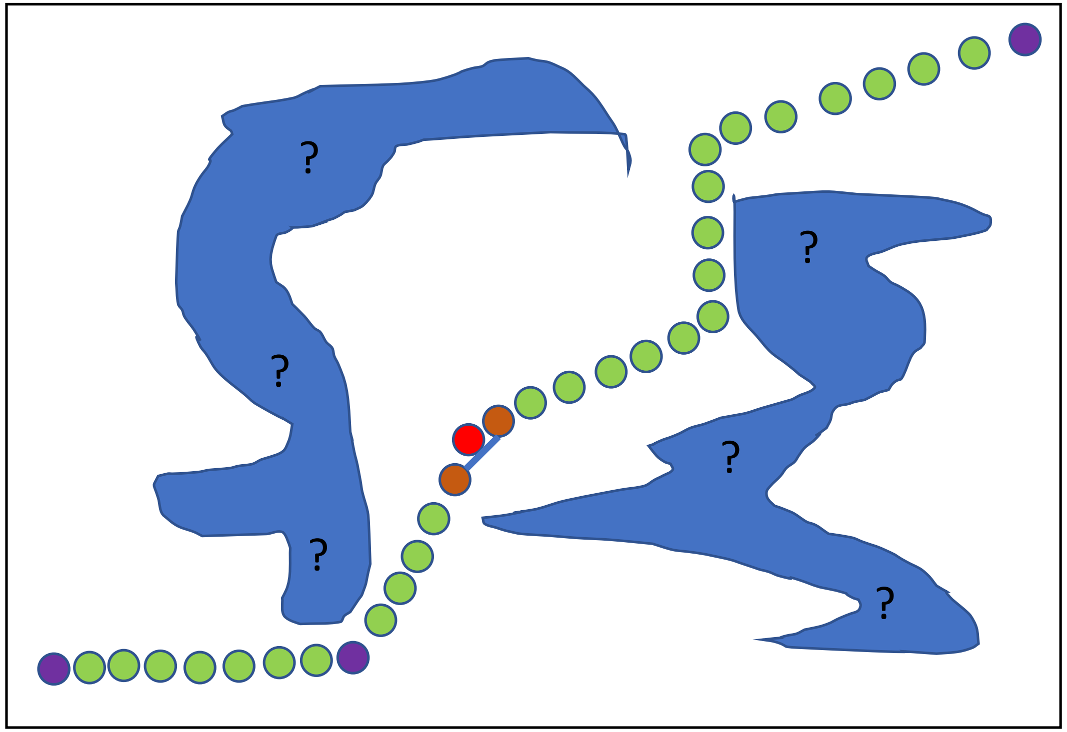
Goal:

Compute a shorter path.

Approach:

1. Sample the computed path.
2. Take a random point.

Some easy brute force approach – Path length



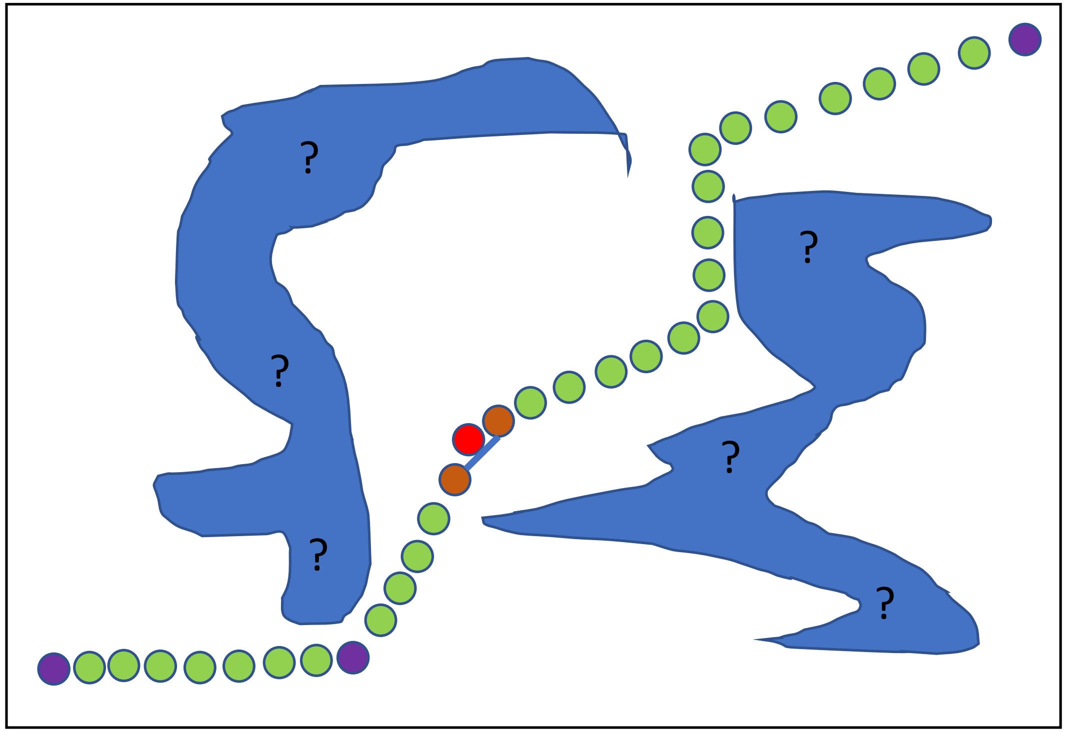
Goal:

Compute a shorter path.

Approach:

1. Sample the computed path.
2. Take a random point.
3. Take neighbours and try to connect them directly.

Some easy brute force approach – Path length



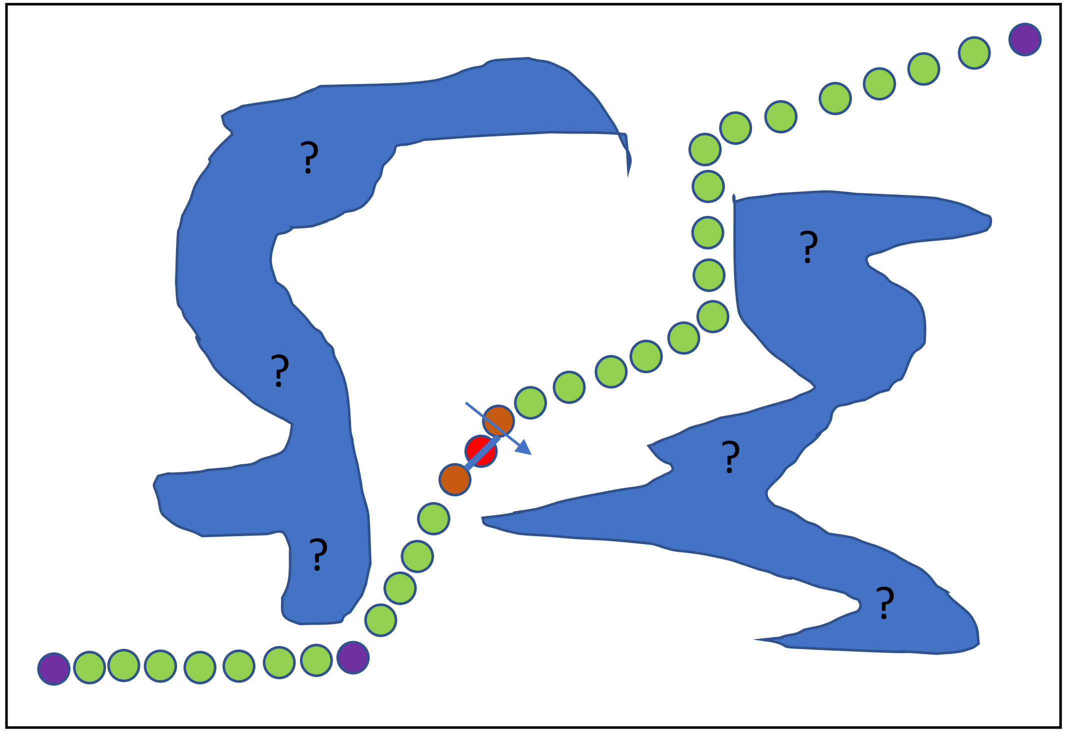
Goal:

Compute a shorter path.

Approach:

1. Sample the computed path.
2. Take a random point.
3. Take neighbours and try to connect them directly.
4. If works, move the random point to the mid.

Some easy brute force approach – Path length



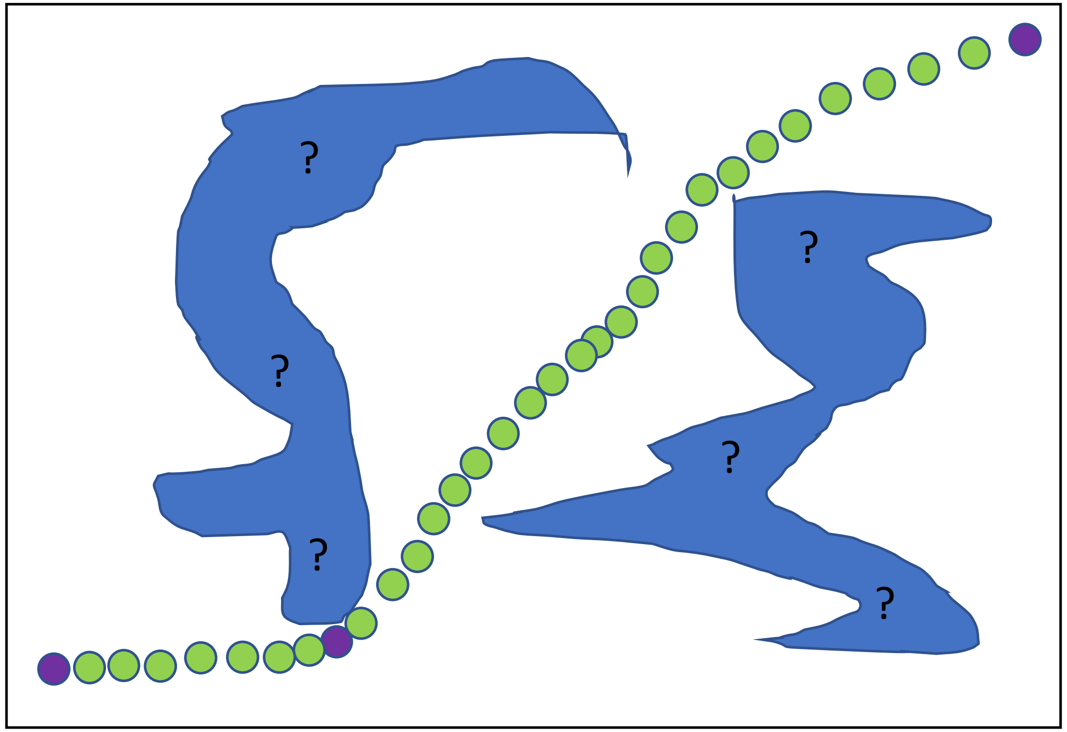
Goal:

Compute a shorter path.

Approach:

1. Sample the computed path.
2. Take a random point.
3. Take neighbours and try to connect them directly.
4. If works, move the random point to the mid.

Some easy brute force approach – Path length



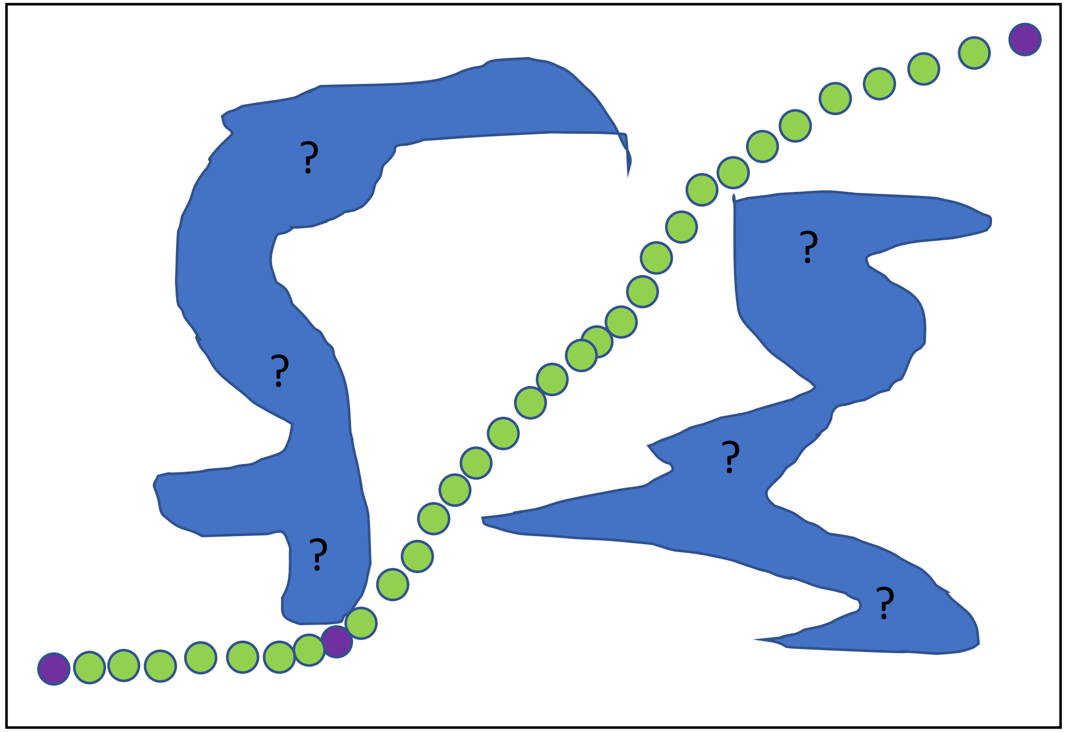
Goal:

Compute a shorter path.

Approach:

1. Sample the computed path.
2. Take a random point.
3. Take neighbours and try to connect them directly.
4. If works, move the random point to the mid.
5. Repeat!

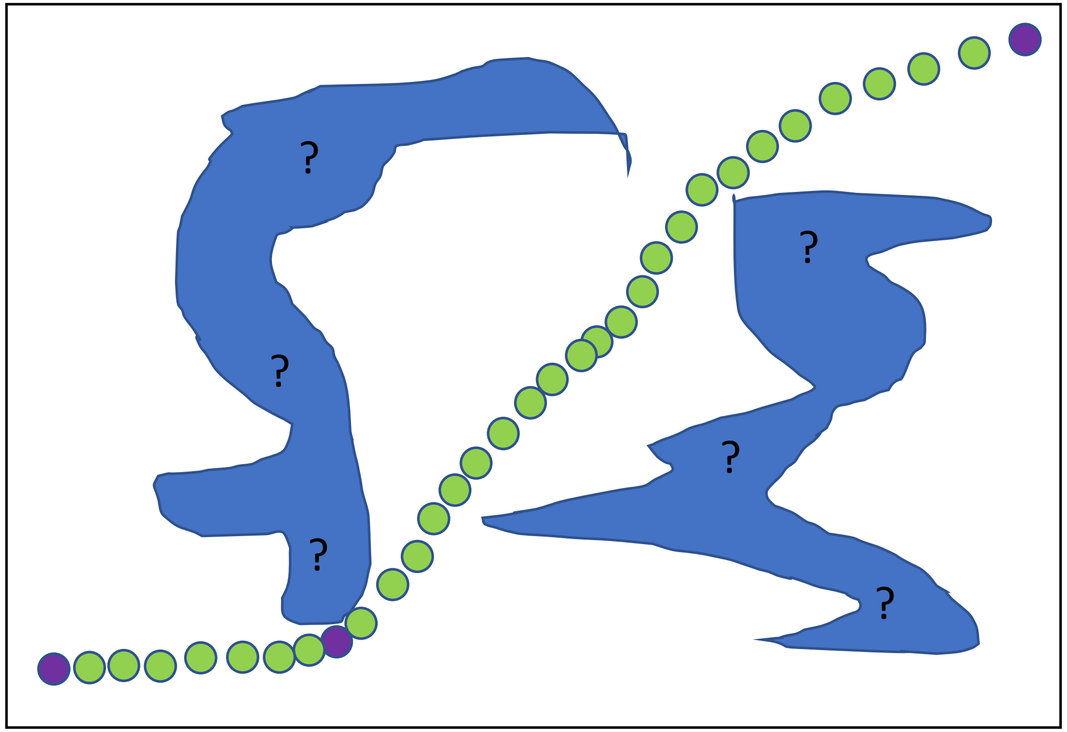
Some easy brute force approach



Some Notes for practice:

If you just optimize the path length you will end up with a path that is short but also the robot moves close to its environment.

Some easy brute force approach

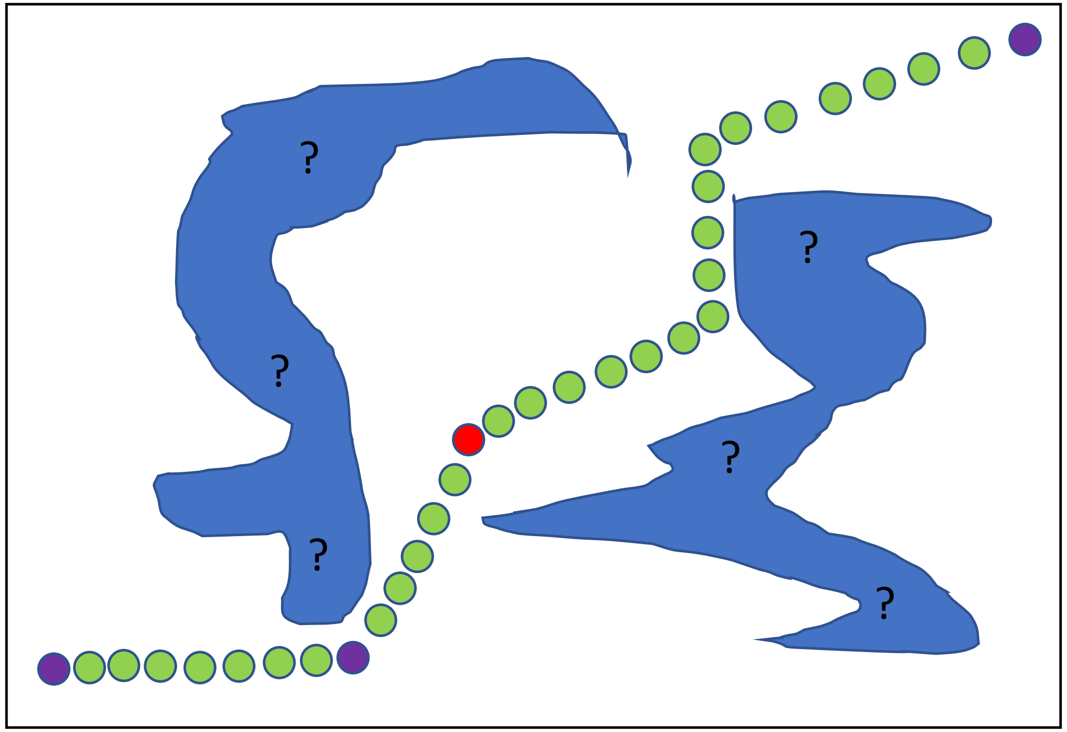


Some Notes for practice:

If you just optimize the path length you will end up with a path that is short but also the robot moves close to its environment.

This is often unwanted and a path with maximal clearance is needed.

Some easy brute force approach – Clearance



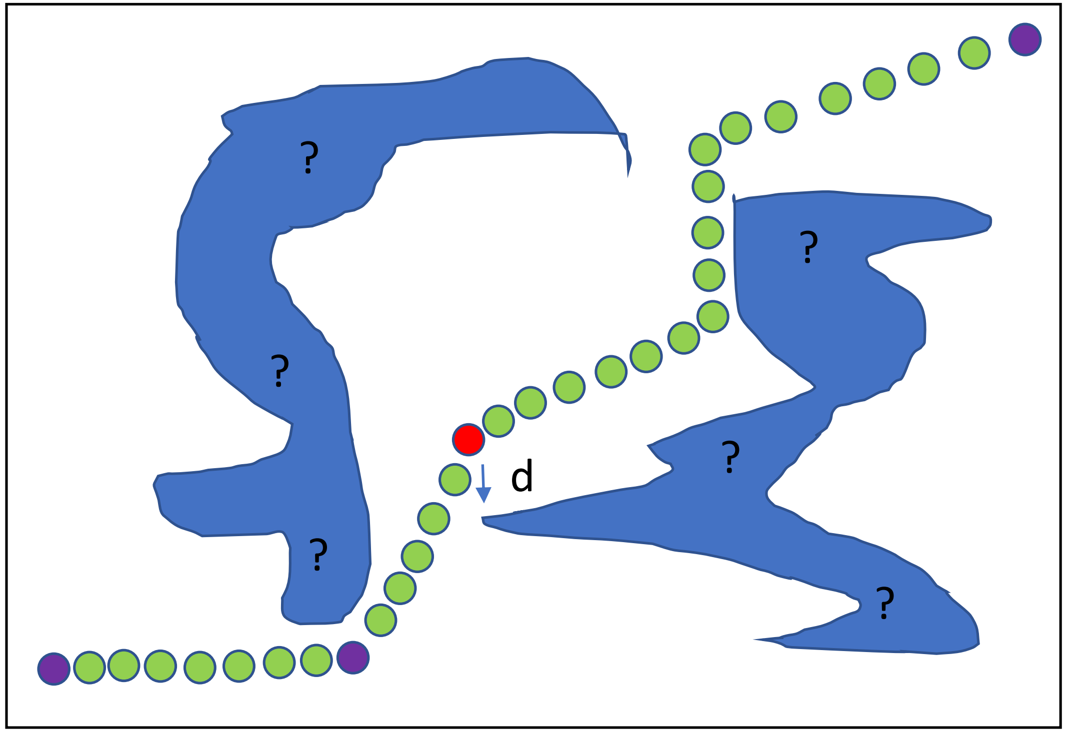
Goal:

Compute a path with max. clearance.

Approach:

1. Sample the computed path.
2. Take a random point.

Some easy brute force approach – Clearance



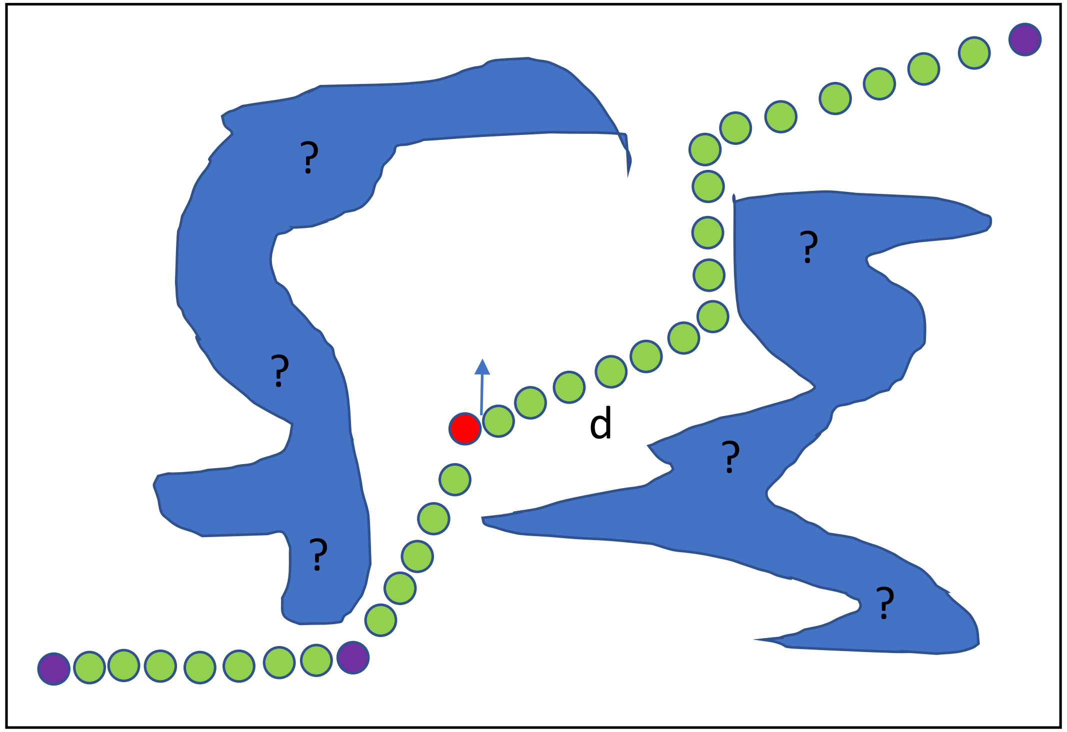
Goal:

Compute a path with max. clearance.

Approach:

1. Sample the computed path.
2. Take a random point.
3. Compute minimal distance

Some easy brute force approach – Clearance



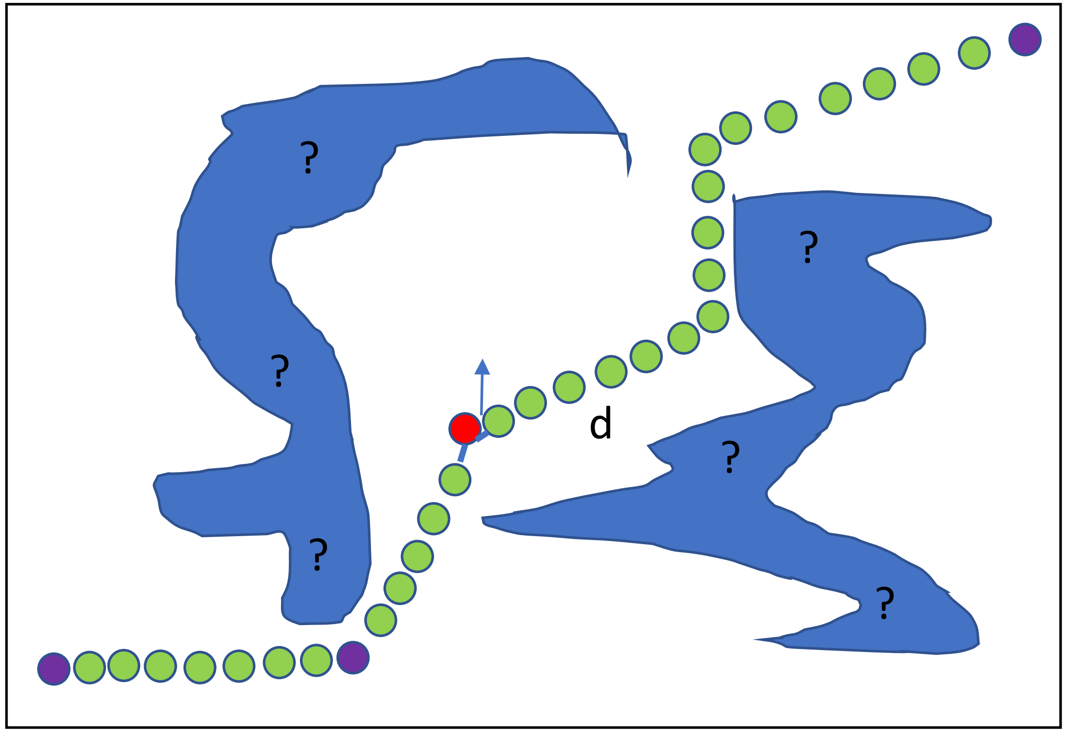
Goal:

Compute a path with max. clearance.

Approach:

1. Sample the computed path.
2. Take a random point.
3. Compute minimal distance
4. Push the point away from it.

Some easy brute force approach – Clearance



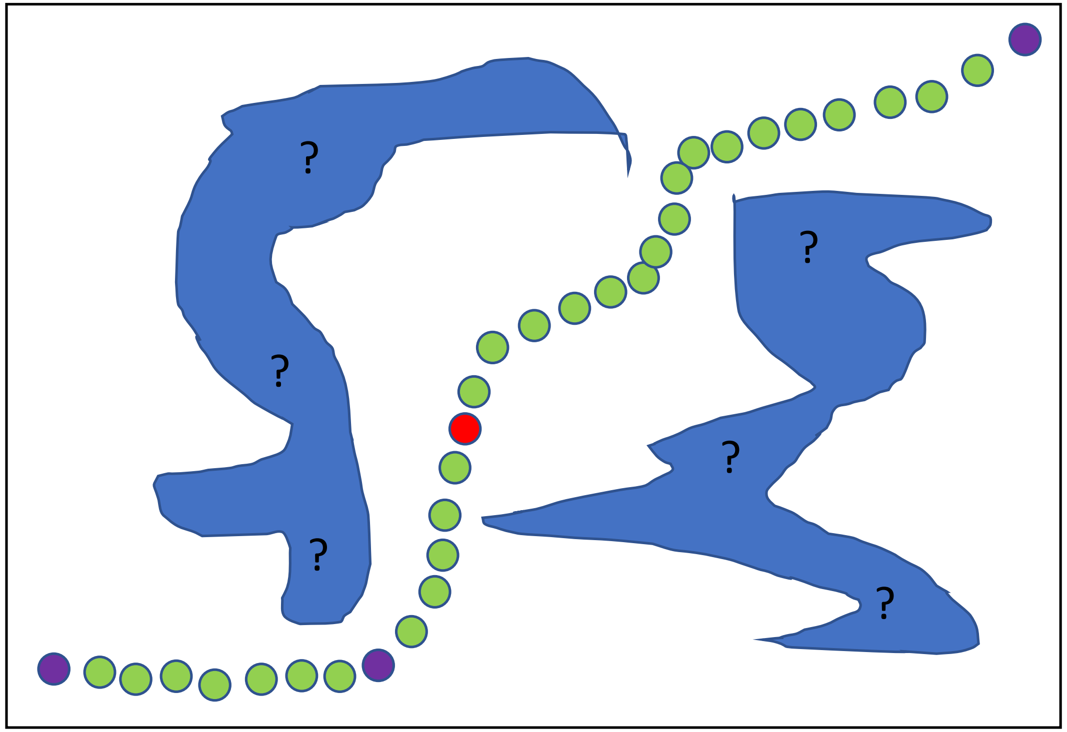
Goal:

Compute a path with max. clearance.

Approach:

1. Sample the computed path.
2. Take a random point.
3. Compute minimal distance
4. Push the point away from it.
5. Check if you can still connect neighbours.

Some easy brute force approach – Clearance



Goal:

Compute a path with max. clearance.

Approach:

1. Sample the computed path.
2. Take a random point.
3. Compute minimal distance
4. Push the point away from it.
5. Check if you can still connect neighbours.
6. Repeat

Is this postprocessing the only way?

- After the rise of motion planning algorithms, the community was researching algorithms that can also find optimal paths.
- In 2011, Sertac Karaman and Emilio Frazzoli presented the paper "**Sampling-based Algorithms for Optimal Motion Planning**"
- They presented sampling-based motion planners that find the optimal path and not only any random path.
- In the years after that presentation, many papers have been published that extend this work to various motion planner and also improvements on this first idea were presented.

Sampling-based Algorithms for Optimal Motion Planning

Sertac Karaman Emilio Frazzoli*

Abstract

During the last decade, sampling-based path planning algorithms, such as Probabilistic RoadMaps (PRM) and Rapidly-exploring Random Trees (RRT), have been shown to work well in practice and possess theoretical guarantees such as probabilistic completeness. However, little effort has been devoted to the formal analysis of the quality of the solution returned by such algorithms, e.g., as a function of the number of samples. The purpose of this paper is to fill this gap, by rigorously analyzing the asymptotic behavior of the cost of the solution returned by stochastic sampling-based algorithms as the number of samples increases. A number of negative results are provided, characterizing existing algorithms, e.g., showing that, under mild technical conditions, the cost of the solution returned by broadly used sampling-based algorithms converges almost surely to a non-optimal value. The main contribution of the paper is the introduction of new algorithms, namely, PRM* and RRT*, which are provably asymptotically optimal, i.e., such that the cost of the returned solution converges almost surely to the optimum. Moreover, it is shown that the computational complexity of the new algorithms is within a constant factor of that of their probabilistically complete (but not asymptotically optimal) counterparts. The analysis in this paper hinges on novel connections between stochastic sampling-based path planning algorithms and the theory of random geometric graphs.

Keywords: Motion planning, optimal path planning, sampling-based algorithms, random geometric graphs.

1 Introduction

The robotic motion planning problem has received a considerable amount of attention, especially over the last decade, as robots started becoming a vital part of modern industry as well as our daily life (Latombe, 1991; LaValle, 2006; Choset et al., 2005). Even though modern robots may possess significant differences in sensing, actuation, size, workspace, application, etc., the problem of navigating through a complex environment is embedded and essential in almost all robotics applications. Moreover, this problem is relevant to other disciplines such as verification, computational biology, and computer animation (Latombe, 1999; Bhatia and Frazzoli, 2004; Branicky et al., 2006; Cortes et al., 2007; Liu and Badler, 2003; Finn and Kavraki, 1999).

Informally speaking, given a robot with a description of its dynamics, a description of the environment, an initial state, and a set of goal states, the motion planning problem is to find a sequence of control inputs so as the drive the robot from its initial state to one of the goal states while obeying the rules of the environment, e.g., not colliding with the surrounding obstacles. An algorithm to address this problem is said to be *complete* if it terminates in finite time, returning a valid solution if one exists, and failure otherwise.

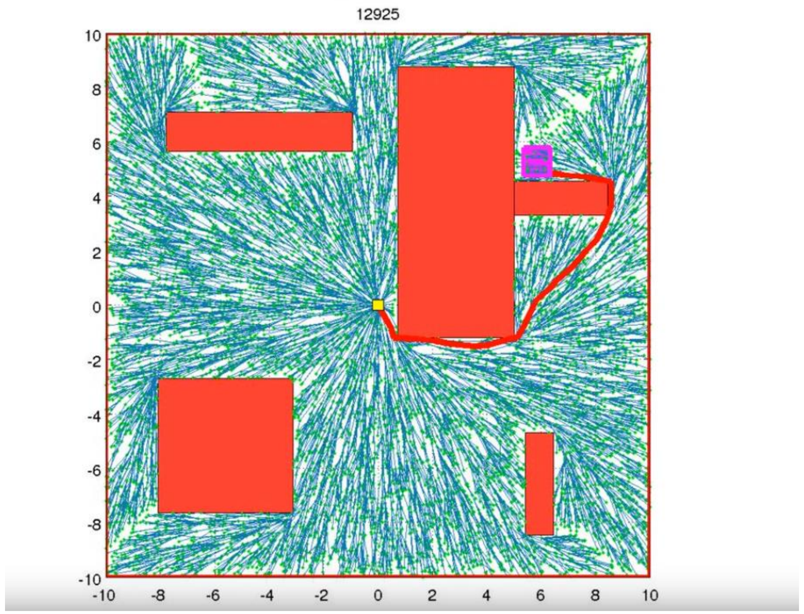
Unfortunately, the problem is known to be very hard from the computational point of view. For example, a basic version of the motion planning problem, called the generalized piano movers

*The authors are with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA.

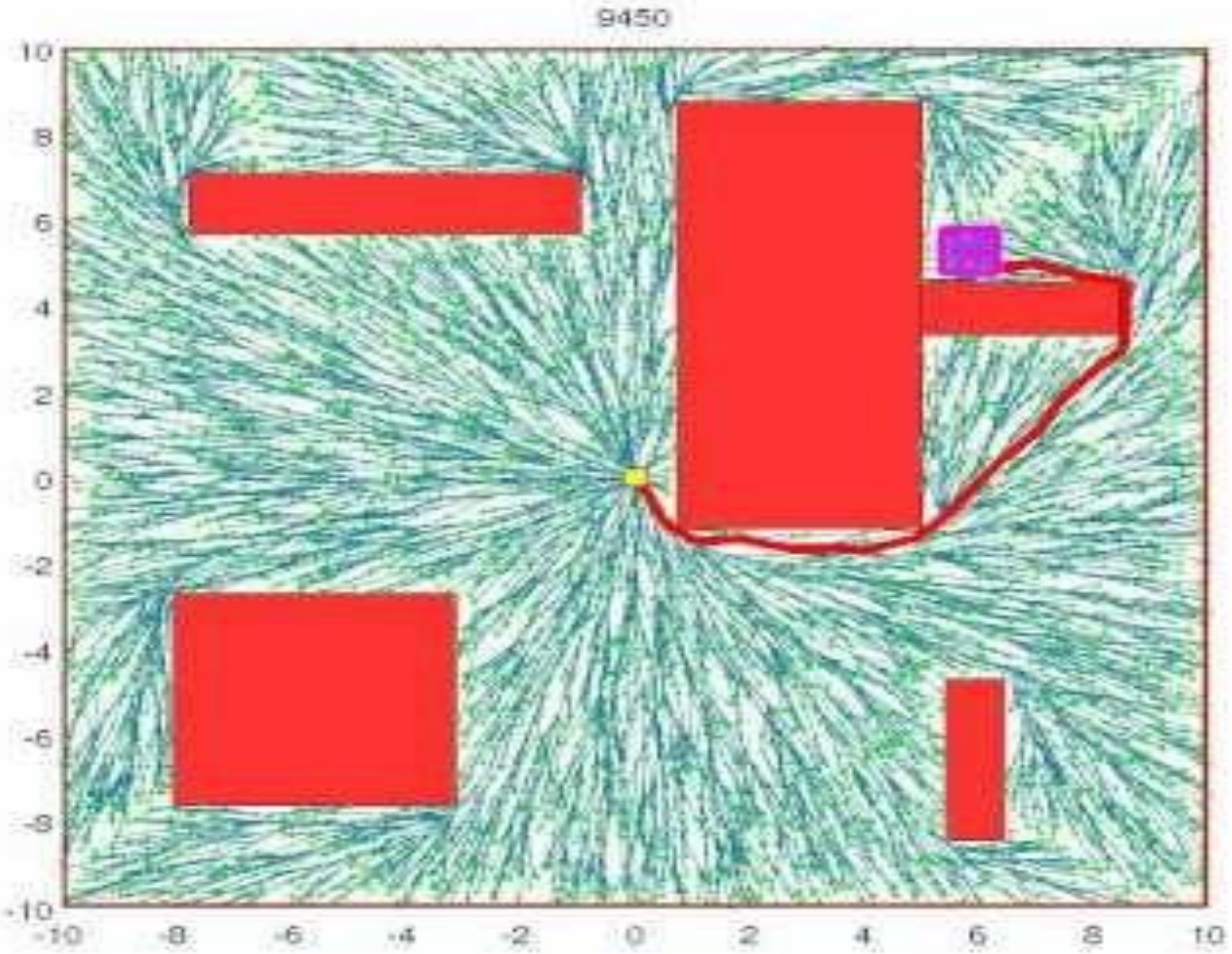
1

How does the RRT* works

- https://www.youtube.com/watch?v=YKiQTJpPFkA



RRT* algorithm illustrative example


9450
10
8
6
4
2
0
-2
-4
-6
-8
-10
-10
-8
-6
-4
-2
0
2
4
6
8
10

RRT vs RRT*

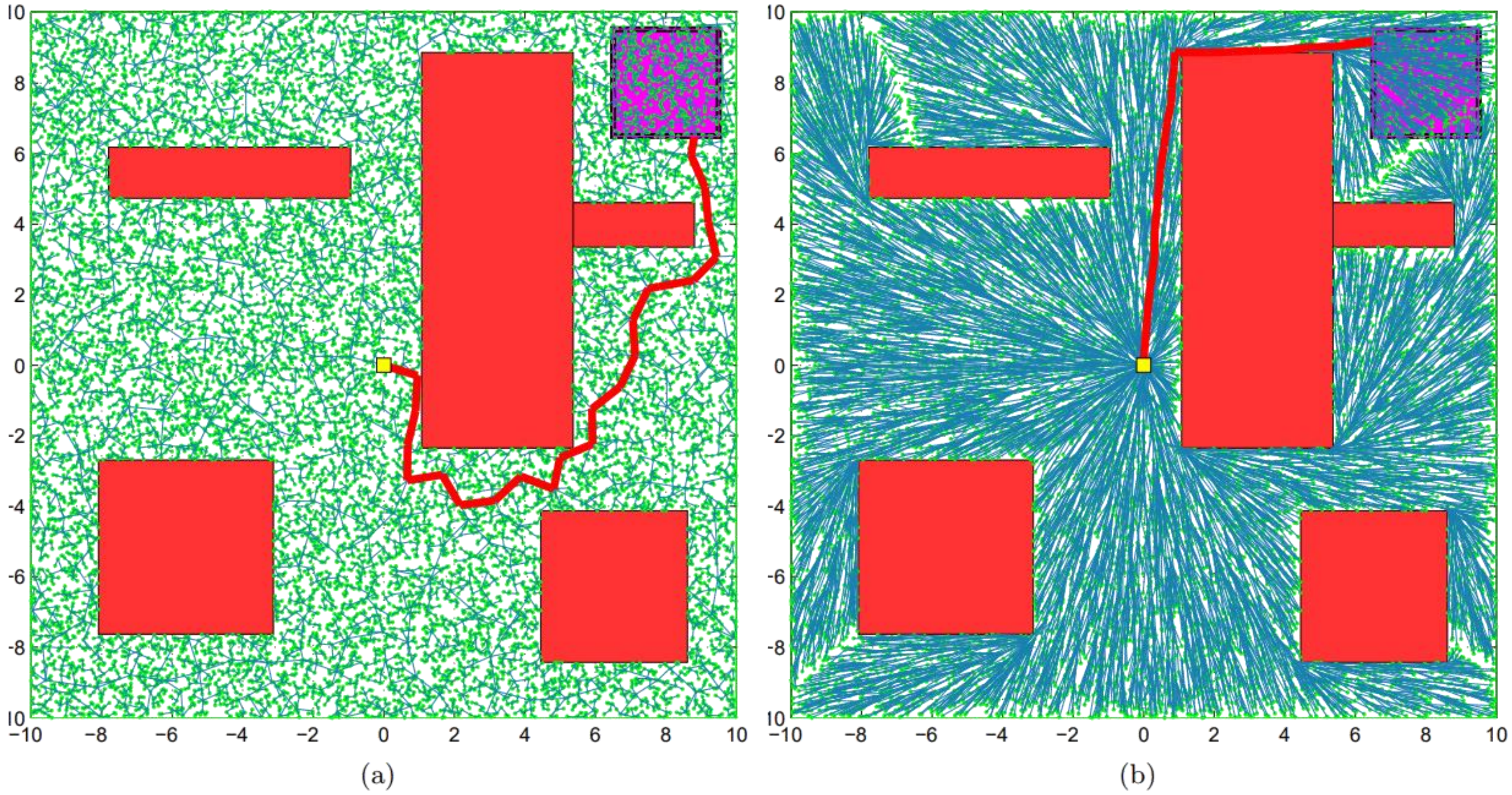


Figure 14: A Comparison of the RRT (shown in (a)) and RRT* (shown in (b)) algorithms on a simulation example with obstacles. Both algorithms were run with the same sample sequence for 20,000 samples. The cost of best path in the RRT and the RRG were 21.02 and 14.51, respectively.

Sources:
Sampling-based Algorithms for Optimal Motion Planning– Karaman and Frazolli
 - https://arxiv.org/pdf/1105.1186.pdf

What about classical motion planners?

- Why should you ever use „normal“ motion planners again?
- Looking at the time complexity of the algorithms shows that the Star version is the same as the classical version.
- But the Star Version is able to provide the optimal path.

Table 1: Summary of results. Time and space complexity are expressed as a function of the number of samples n, for a fixed environment.

	Algorithm	Probabilistic Completeness	Asymptotic Optimality	Monotone Convergence	Time Complexity: Processing	Time Complexity: Query	Space Complexity
Existing Algorithms	PRM	Yes	No	Yes	$O(n \log n)$	$O(n \log n)$	$O(n)$
	sPRM	Yes	Yes	Yes	$O(n^2)$	$O(n^2)$	$O(n^2)$
	k-sPRM	Conditional	No	No	$O(n \log n)$	$O(n \log n)$	$O(n)$
	RRT	Yes	No	Yes	$O(n \log n)$	$O(n)$	$O(n)$
Proposed Algorithms	PRM*, k-PRM*	Yes	Yes	No	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
	RRG, k-RRG	Yes	Yes	Yes	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
	RRT*, k-RRT*	Yes	Yes	Yes	$O(n \log n)$	$O(n)$	$O(n)$

Sources:
Sampling-based Algorithms for Optimal Motion Planning– Karaman and Frazolli - https://arxiv.org/pdf/1105.1186.pdf

But…

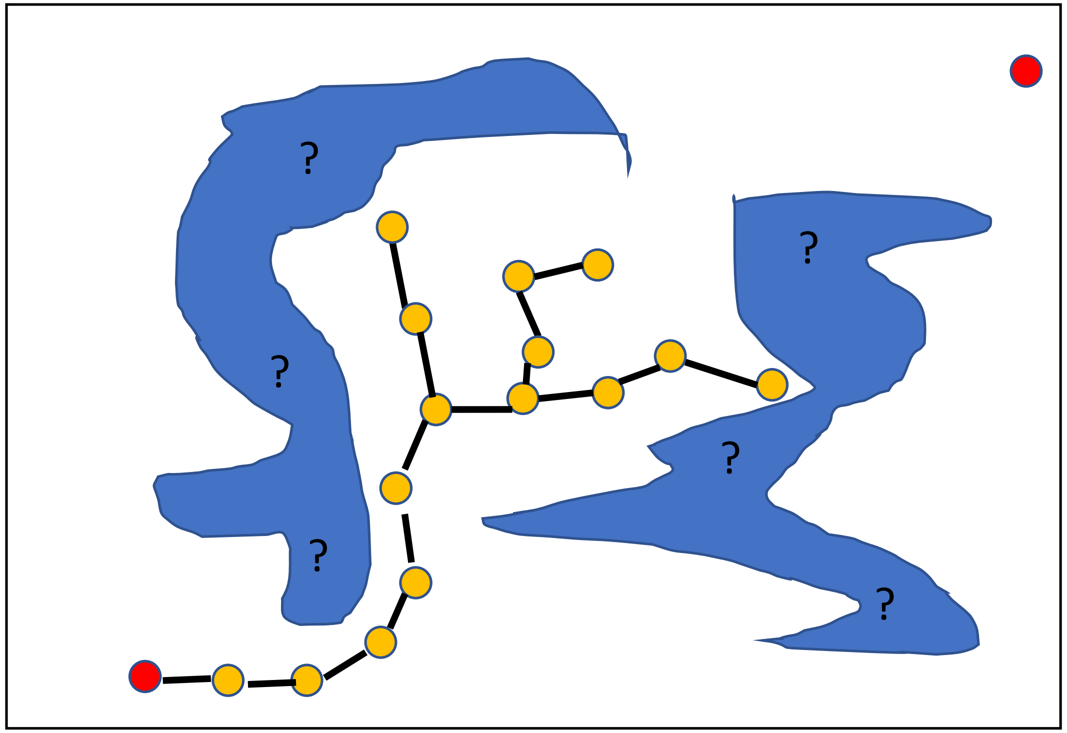
- The optimal algorithms explore the configuration space less effective.
- Moreover, the optimality search, favors short/optimal motions.
- Optimal motion are not necessary on the solution path.
- The solution path can also contain „high cost“ movements.

→ Solving motion problems with narrow passages with optimal planner is very ineffective.

- In higher dimensions the convergence to the optimal path is very slow.
- Performance decreases even more with higher number of DOFs

In Summary: By finding optimal path during the planning phase decreases performance.

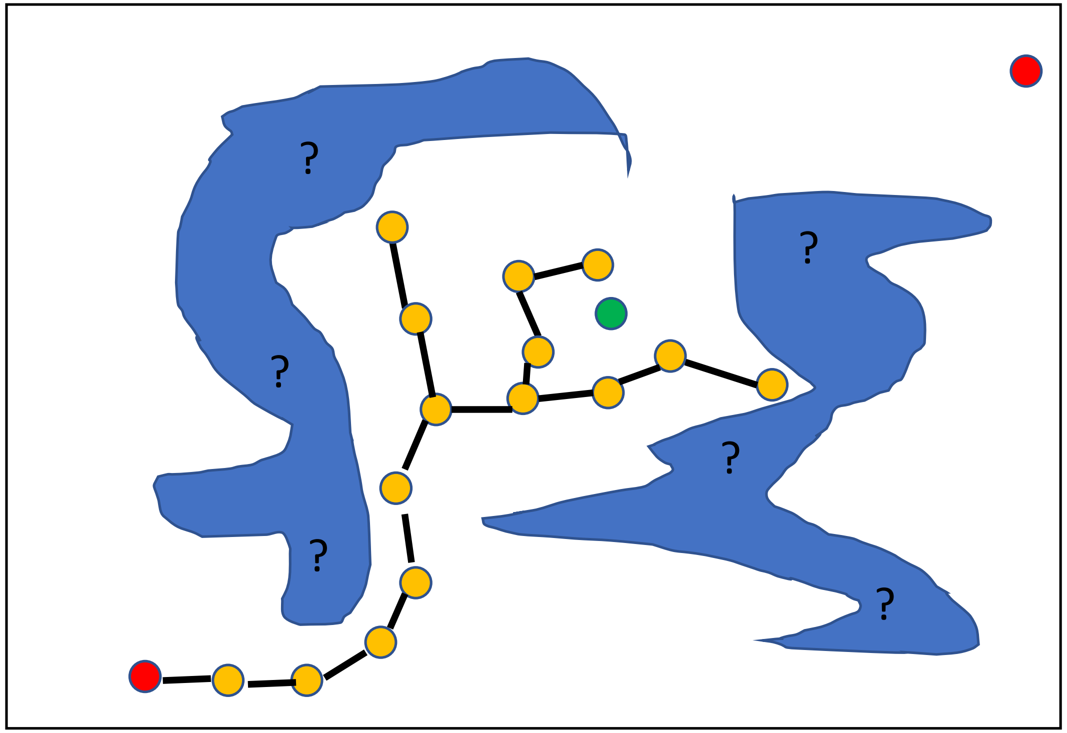
How does it work?



Algorithm 6: RRT*

$V \leftarrow \{x_{\mathrm{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\mathrm{rand}} \leftarrow \mathtt{SampleFree}_i$;
  $x_{\mathrm{nearest}} \leftarrow \mathtt{Nearest}(G = (V, E), x_{\mathrm{rand}})$;
  $x_{\mathrm{new}} \leftarrow \mathtt{Steer}(x_{\mathrm{nearest}}, x_{\mathrm{rand}})$;
  if $\mathtt{ObtacleFree}(x_{\mathrm{nearest}}, x_{\mathrm{new}})$ **then**
    $X_{\mathrm{near}} \leftarrow \mathtt{Near}(G = (V, E), x_{\mathrm{new}}, \min\{\gamma_{\mathrm{RRT^*}}(\log(\mathrm{card}\,(V))/\,\mathrm{card}\,(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\mathrm{new}}\}$;
    $x_{\mathrm{min}} \leftarrow x_{\mathrm{nearest}}$; $c_{\mathrm{min}} \leftarrow \mathtt{Cost}(x_{\mathrm{nearest}}) + c(\mathtt{Line}(x_{\mathrm{nearest}}, x_{\mathrm{new}}))$;
    foreach $x_{\mathrm{near}} \in X_{\mathrm{near}}$ **do** // Connect along a minimum-cost path
      if $\mathtt{CollisionFree}(x_{\mathrm{near}}, x_{\mathrm{new}}) \wedge \mathtt{Cost}(x_{\mathrm{near}}) + c(\mathtt{Line}(x_{\mathrm{near}}, x_{\mathrm{new}})) < c_{\mathrm{min}}$ **then**
        $x_{\mathrm{min}} \leftarrow x_{\mathrm{near}}$; $c_{\mathrm{min}} \leftarrow \mathtt{Cost}(x_{\mathrm{near}}) + c(\mathtt{Line}(x_{\mathrm{near}}, x_{\mathrm{new}}))$
    $E \leftarrow E \cup \{(x_{\mathrm{min}}, x_{\mathrm{new}})\}$;
    foreach $x_{\mathrm{near}} \in X_{\mathrm{near}}$ **do** // Rewire the tree
      if $\mathtt{CollisionFree}(x_{\mathrm{new}}, x_{\mathrm{near}}) \wedge \mathtt{Cost}(x_{\mathrm{new}}) + c(\mathtt{Line}(x_{\mathrm{new}}, x_{\mathrm{near}})) < \mathtt{Cost}(x_{\mathrm{near}})$
      then $x_{\mathrm{parent}} \leftarrow \mathtt{Parent}(x_{\mathrm{near}})$;
      $E \leftarrow (E \setminus \{(x_{\mathrm{parent}}, x_{\mathrm{near}})\}) \cup \{(x_{\mathrm{new}}, x_{\mathrm{near}})\}$
return $G = (V, E)$;

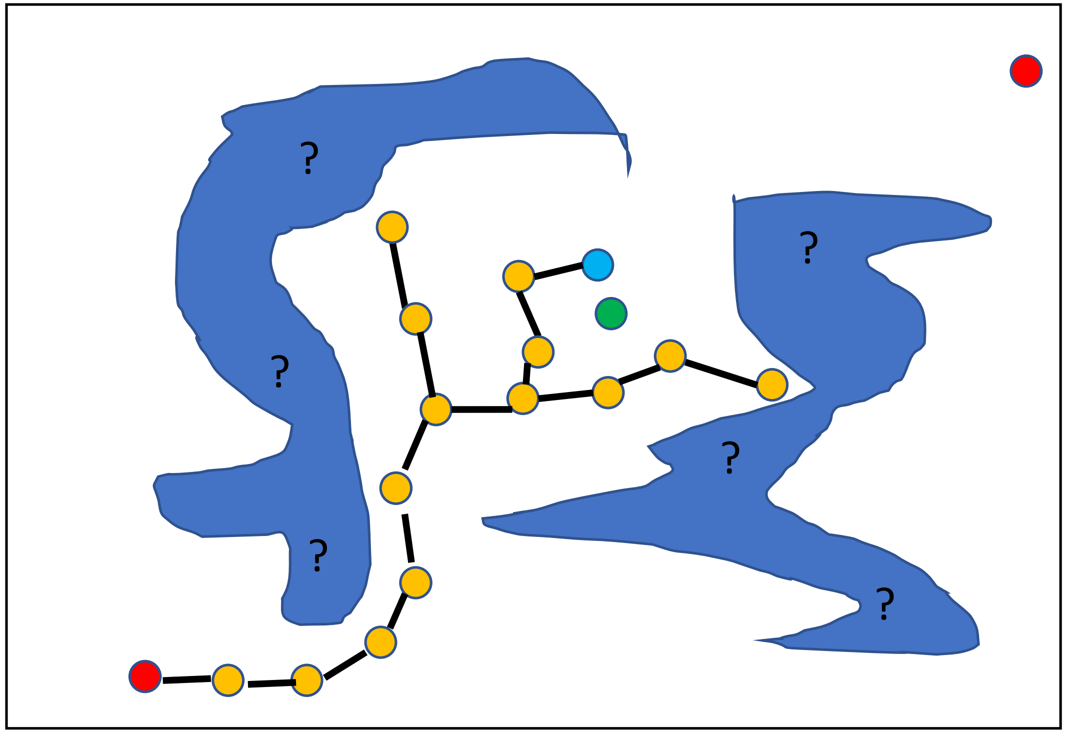
How does it work?



Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\text{min}} \leftarrow x_{\text{nearest}}$; $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$; $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
      then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
        $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

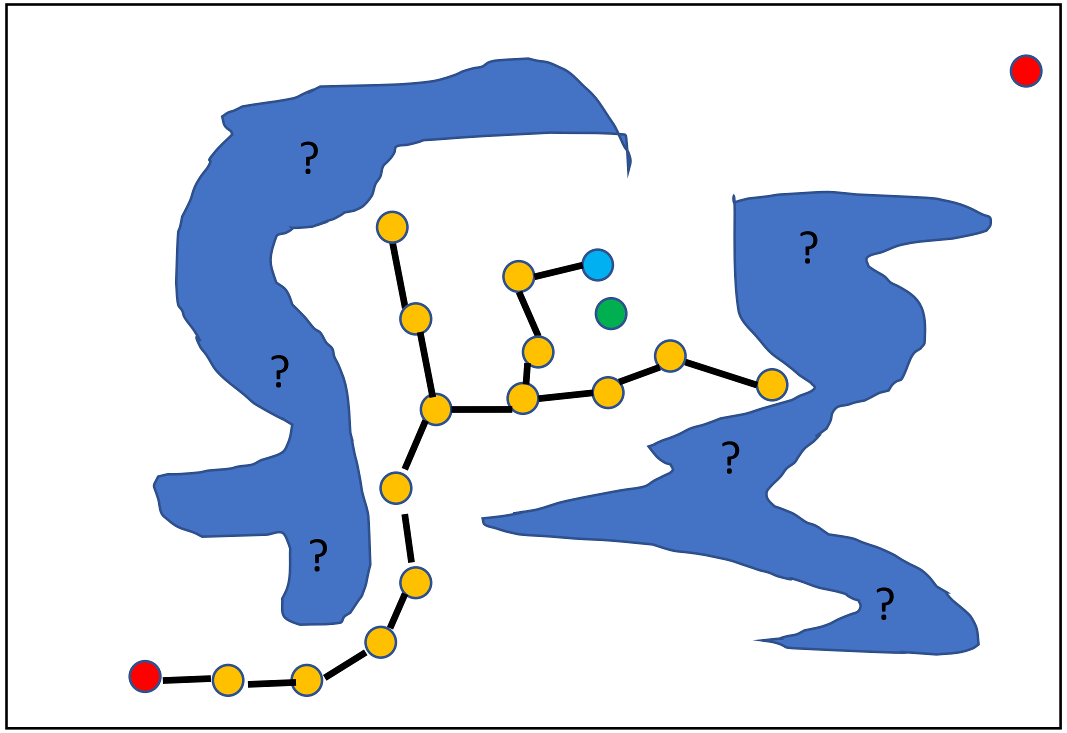
How does it work?



Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}(V))/\text{card}(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\min} \leftarrow x_{\text{nearest}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\min}$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
      then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
        $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

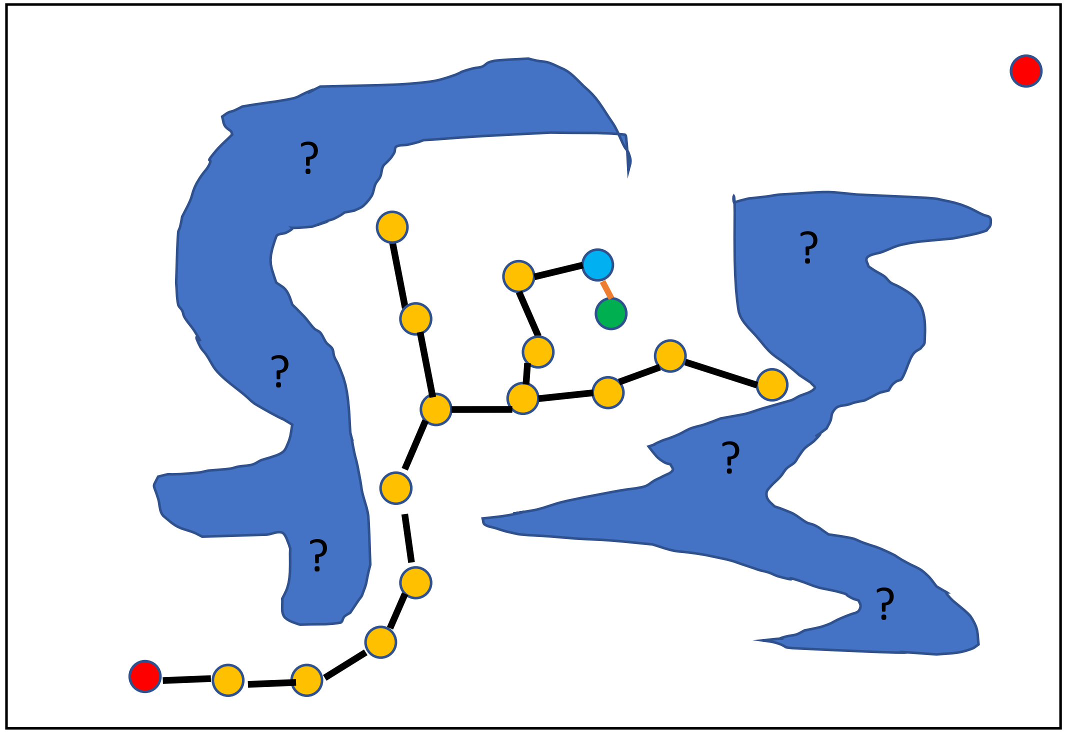
How does it work?



Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\};\ E \leftarrow \emptyset;$
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i;$
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}});$
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})\ ;$
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}(V))/\text{card}(V))^{1/d}, \eta\})\ ;$
    $V \leftarrow V \cup \{x_{\text{new}}\};$
    $x_{\text{min}} \leftarrow x_{\text{nearest}};\ c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}));$
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}};\ c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\};$
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
      then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}});$
        $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E);$

How does it work?

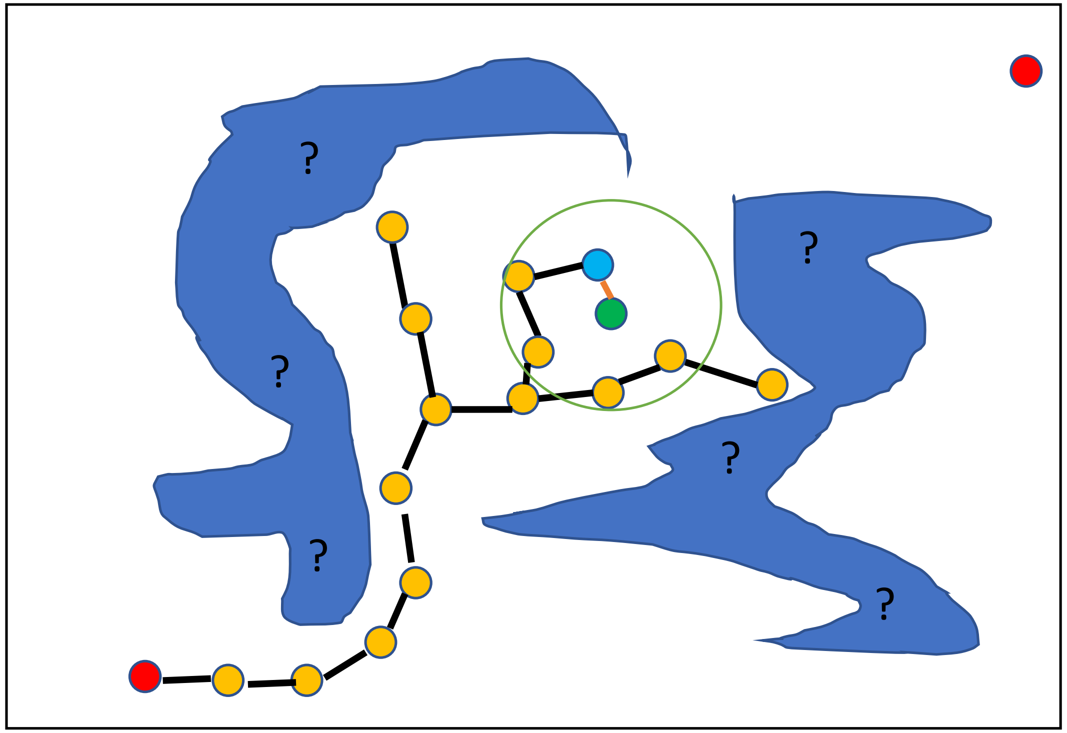


Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\min} \leftarrow x_{\text{nearest}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\min}$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
      then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
        $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

Note: The edge as well as the node are not yet added to the data structure.

How does it work?



Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\};\ E \leftarrow \emptyset;$

for $i = 1, \ldots, n$ **do**

  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i;$

  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}});$

  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})\ ;$

  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**

    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})\ ;$

    $V \leftarrow V \cup \{x_{\text{new}}\};$

    $x_{\text{min}} \leftarrow x_{\text{nearest}};\ c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}));$

    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path

      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**

        $x_{\text{min}} \leftarrow x_{\text{near}};\ c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$

    $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\};$

    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree

      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$

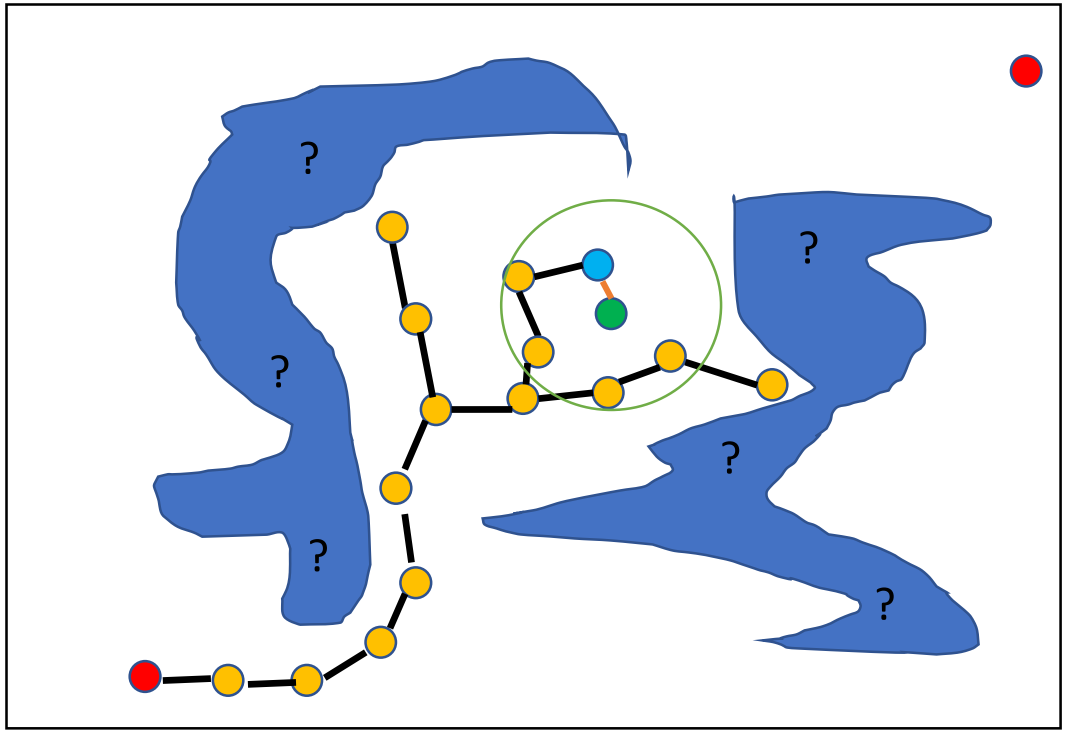
      then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}});$

        $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$

return $G = (V, E);$

Note: One radius of the circle has to be defined based on the dimension of the configuration space.

How does it work?

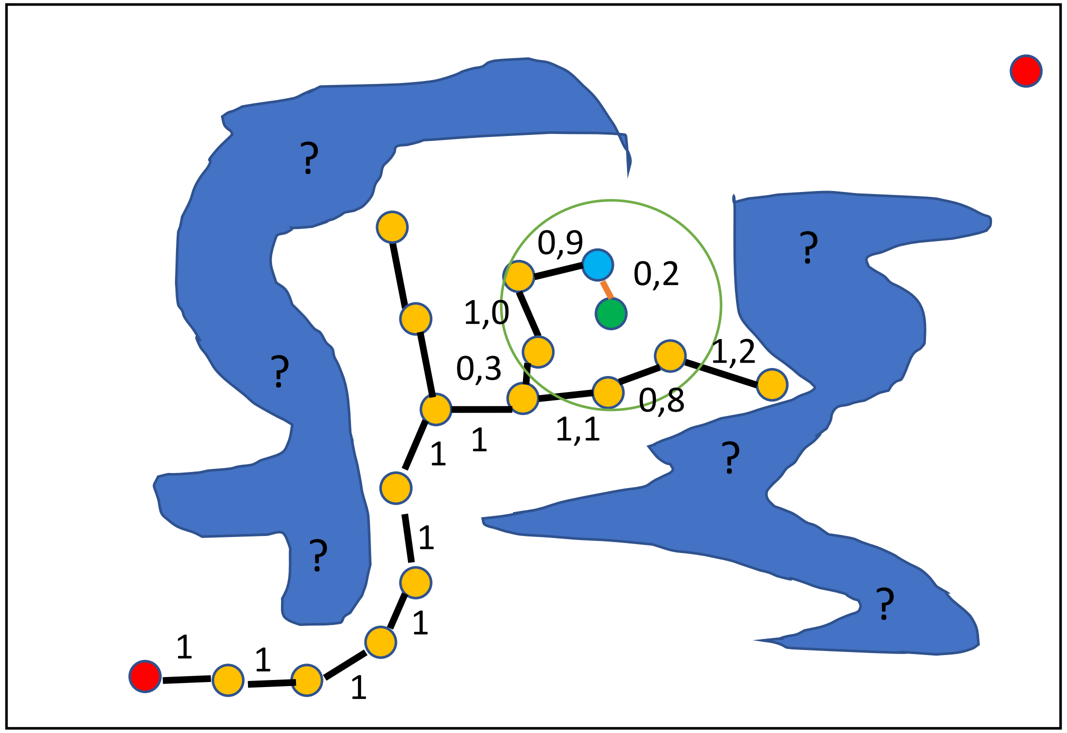


Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\text{min}} \leftarrow x_{\text{nearest}}$; $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$; $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
      then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
        $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

Note: Only if the path was obstacle free and there are neighboring nodes --> the node is added to V.

How does it work?

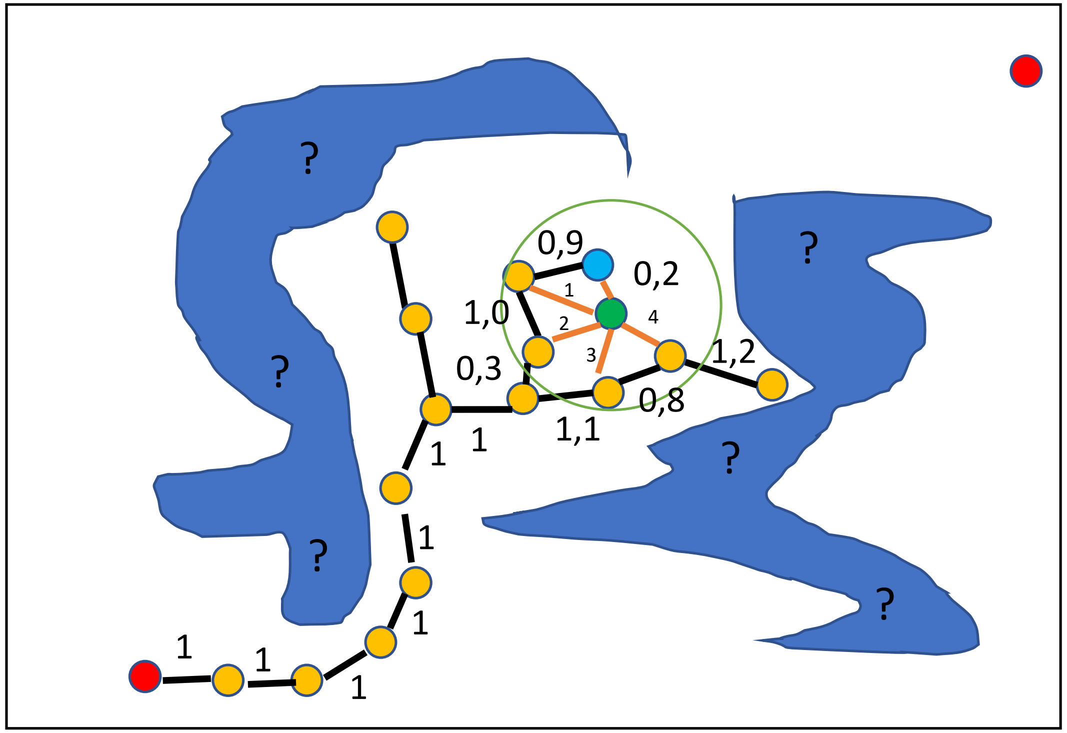


Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}(V))/\,\text{card}(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\text{min}} \leftarrow x_{\text{nearest}}$; $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$; $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
      then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
        $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

Note: c_{min} = 1+1+1+1+1+1+0,3+1+0,9+0,2 = 8,4

How does it work?

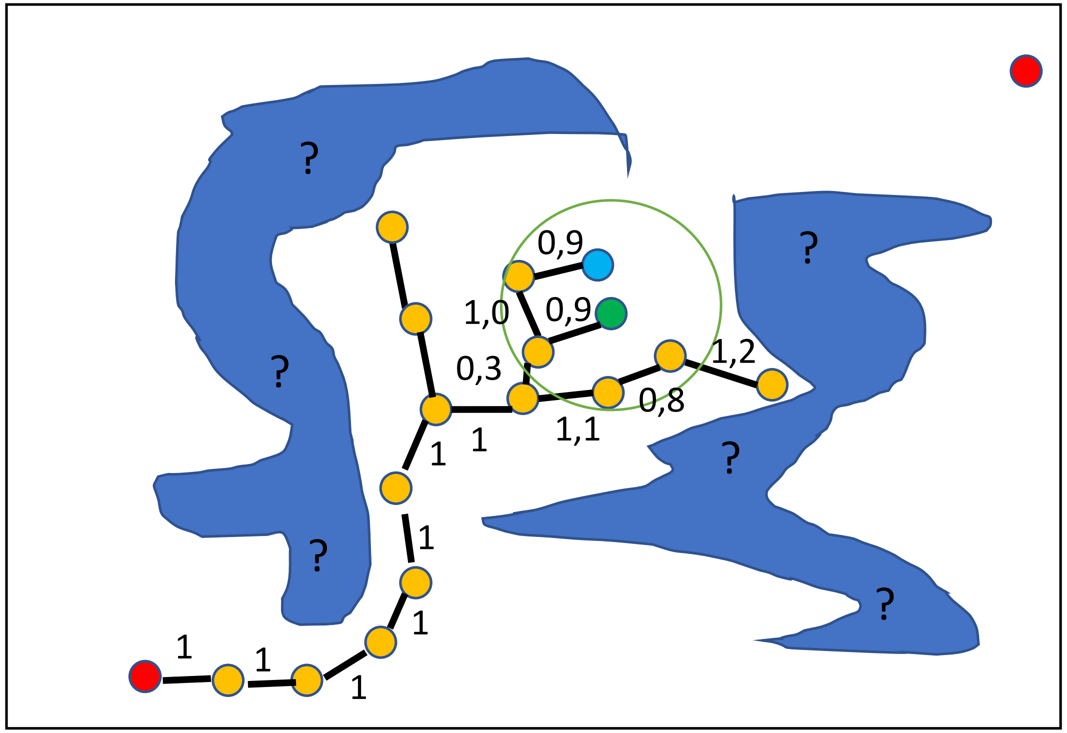


Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}(V))/\text{card}(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\text{min}} \leftarrow x_{\text{nearest}}$; $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$; $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
      then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
        $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

Note: c_{min} = 1+1+1+1+1+1+0,3+1+0,9+0,2 = 8,4
c_1= 1+1+1+1+1+1+0,3+1+1 = 8,3
c_2= 1+1+1+1+1+1+0,3+0,9 = 7,2
c_3= 1+1+1+1+1+1+1,1+0,7 = 7,9
c_4= 1+1+1+1+1+1+1,1+0,8+0,5 = 8,4

How does it work?

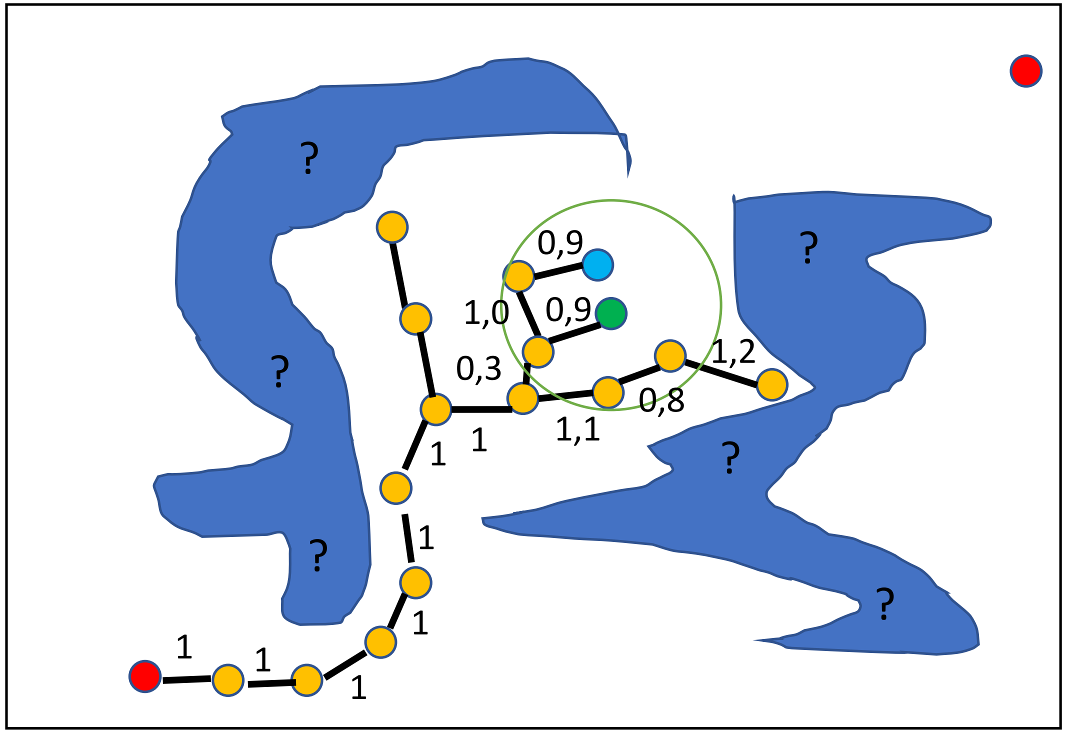


Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\min} \leftarrow x_{\text{nearest}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\min}$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
      then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
        $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

Note: c_{min}= 1+1+1+1+1+1+0,3+0,9 = 7,2

How does it work?

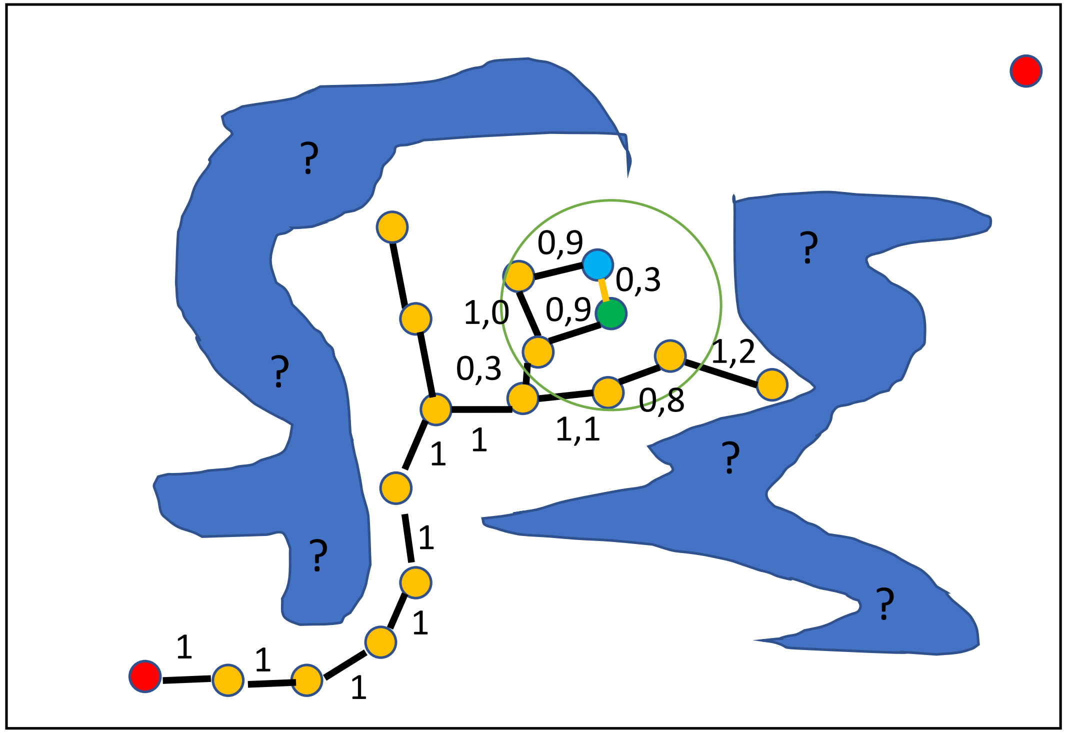


Algorithm 6: RRT*

$V \leftarrow \{x_{\mathrm{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\mathrm{rand}} \leftarrow \mathtt{SampleFree}_i$;
  $x_{\mathrm{nearest}} \leftarrow \mathtt{Nearest}(G = (V, E), x_{\mathrm{rand}})$;
  $x_{\mathrm{new}} \leftarrow \mathtt{Steer}(x_{\mathrm{nearest}}, x_{\mathrm{rand}})$;
  if $\mathtt{ObtacleFree}(x_{\mathrm{nearest}}, x_{\mathrm{new}})$ **then**
    $X_{\mathrm{near}} \leftarrow \mathtt{Near}(G = (V, E), x_{\mathrm{new}}, \min\{\gamma_{\mathrm{RRT^*}}(\log(\mathrm{card}\,(V))/\,\mathrm{card}\,(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\mathrm{new}}\}$;
    $x_{\mathrm{min}} \leftarrow x_{\mathrm{nearest}}$; $c_{\mathrm{min}} \leftarrow \mathtt{Cost}(x_{\mathrm{nearest}}) + c(\mathtt{Line}(x_{\mathrm{nearest}}, x_{\mathrm{new}}))$;
    foreach $x_{\mathrm{near}} \in X_{\mathrm{near}}$ **do** // Connect along a minimum-cost path
      if $\mathtt{CollisionFree}(x_{\mathrm{near}}, x_{\mathrm{new}}) \wedge \mathtt{Cost}(x_{\mathrm{near}}) + c(\mathtt{Line}(x_{\mathrm{near}}, x_{\mathrm{new}})) < c_{\mathrm{min}}$ **then**
        $x_{\mathrm{min}} \leftarrow x_{\mathrm{near}}$; $c_{\mathrm{min}} \leftarrow \mathtt{Cost}(x_{\mathrm{near}}) + c(\mathtt{Line}(x_{\mathrm{near}}, x_{\mathrm{new}}))$
    $E \leftarrow E \cup \{(x_{\mathrm{min}}, x_{\mathrm{new}})\}$;
    foreach $x_{\mathrm{near}} \in X_{\mathrm{near}}$ **do** // Rewire the tree
      if $\mathtt{CollisionFree}(x_{\mathrm{new}}, x_{\mathrm{near}}) \wedge \mathtt{Cost}(x_{\mathrm{new}}) + c(\mathtt{Line}(x_{\mathrm{new}}, x_{\mathrm{near}})) < \mathtt{Cost}(x_{\mathrm{near}})$
      then $x_{\mathrm{parent}} \leftarrow \mathtt{Parent}(x_{\mathrm{near}})$;
      $E \leftarrow (E \setminus \{(x_{\mathrm{parent}}, x_{\mathrm{near}})\}) \cup \{(x_{\mathrm{new}}, x_{\mathrm{near}})\}$
return $G = (V, E)$;

Note: Now the there might be the case, that there are new route that are more optimal → therefore we have to check if we need to rewire the tree.

How does it work?

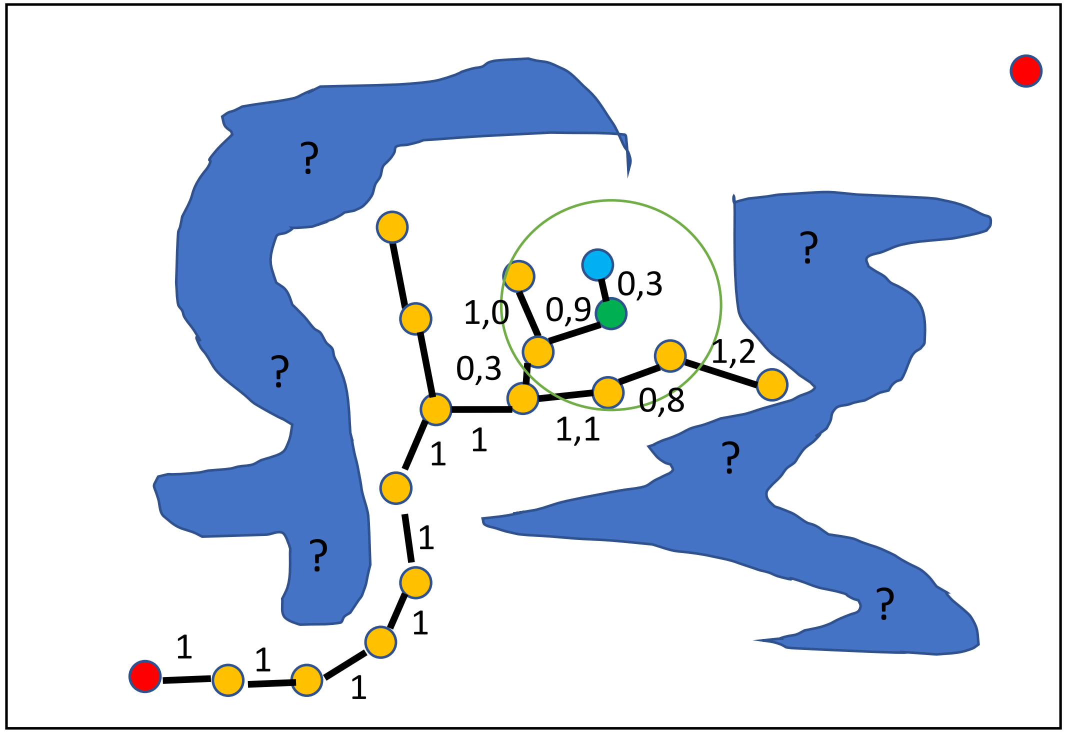


Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}(V))/\text{card}(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\text{min}} \leftarrow x_{\text{nearest}}$; $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \land \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$; $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \land \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
      then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
        $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

Note: This connection is better than the original one.
→ rewire

How does it work?

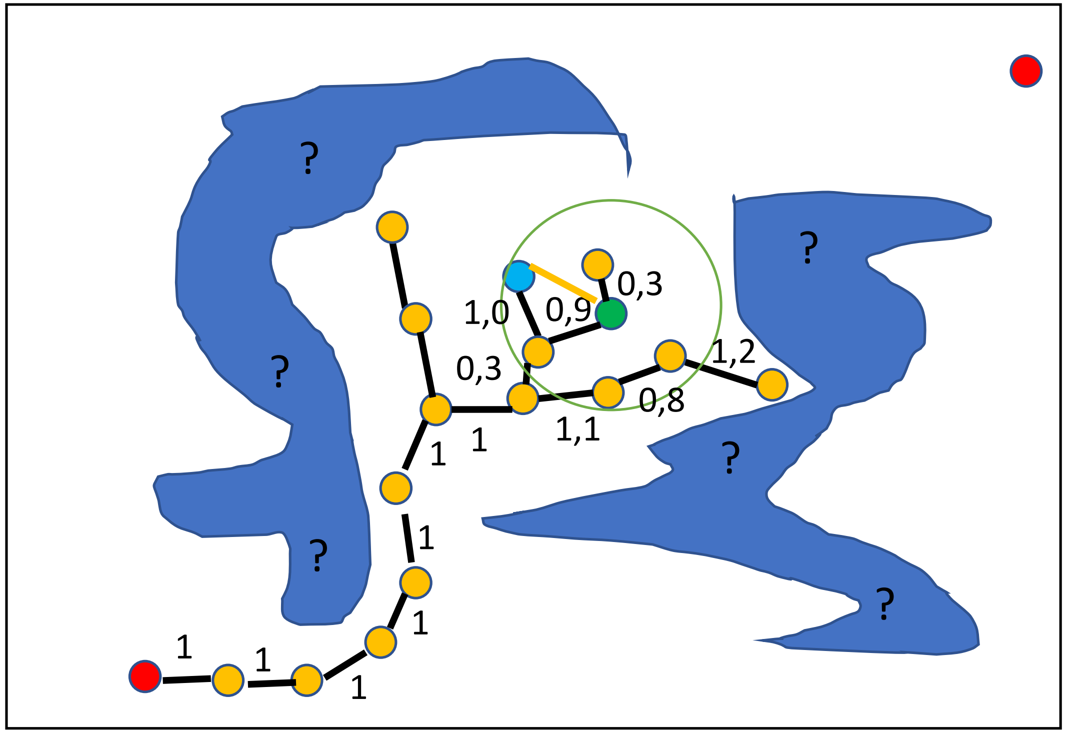


Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT*}}(\log(\text{card}(V))/\text{card}(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\min} \leftarrow x_{\text{nearest}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\min}$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
      then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
        $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

Note: This connection is better than the original one.
→ rewire

How does it work?

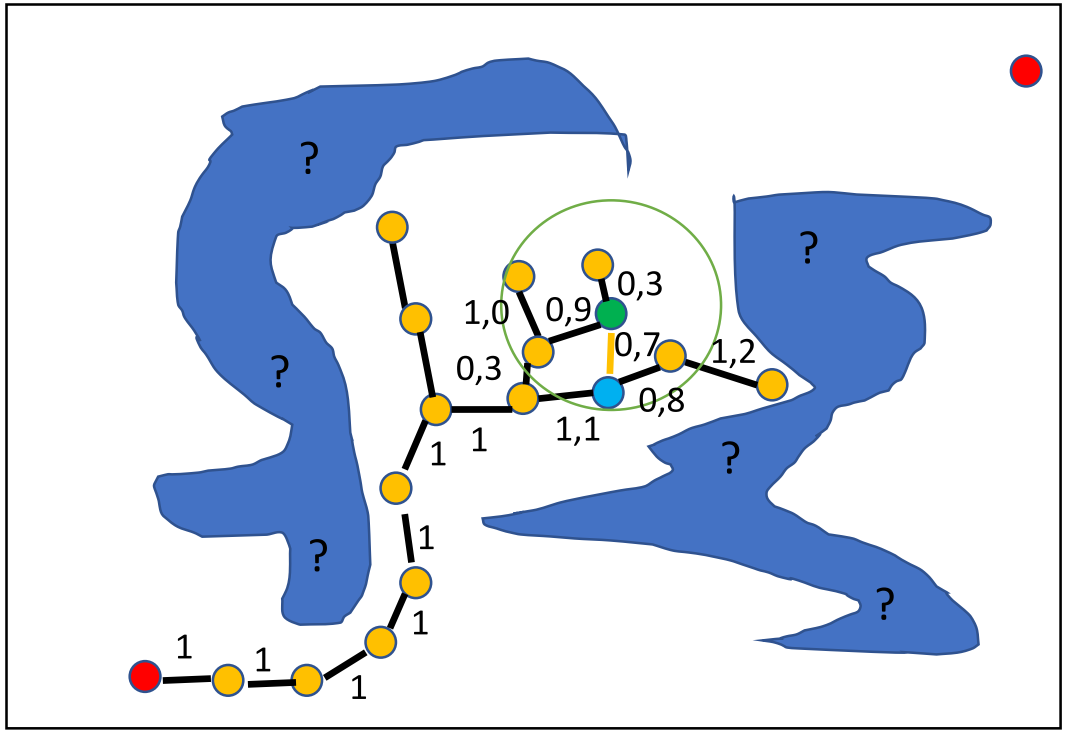


Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow$ `SampleFree`$_i$;
  $x_{\text{nearest}} \leftarrow$ `Nearest`$(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow$ `Steer`$(x_{\text{nearest}}, x_{\text{rand}})$;
  if `ObtacleFree`$(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow$ `Near`$(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}(V))/\text{card}(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\text{min}} \leftarrow x_{\text{nearest}}$; $c_{\text{min}} \leftarrow$ `Cost`$(x_{\text{nearest}}) + c($`Line`$(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if `CollisionFree`$(x_{\text{near}}, x_{\text{new}}) \wedge$ `Cost`$(x_{\text{near}}) + c($`Line`$(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$; $c_{\text{min}} \leftarrow$ `Cost`$(x_{\text{near}}) + c($`Line`$(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if `CollisionFree`$(x_{\text{new}}, x_{\text{near}}) \wedge$ `Cost`$(x_{\text{new}}) + c($`Line`$(x_{\text{new}}, x_{\text{near}})) <$ `Cost`(x_{near})
      then $x_{\text{parent}} \leftarrow$ `Parent`(x_{near});
        $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

Note: This connection is not → no rewire

How does it work?

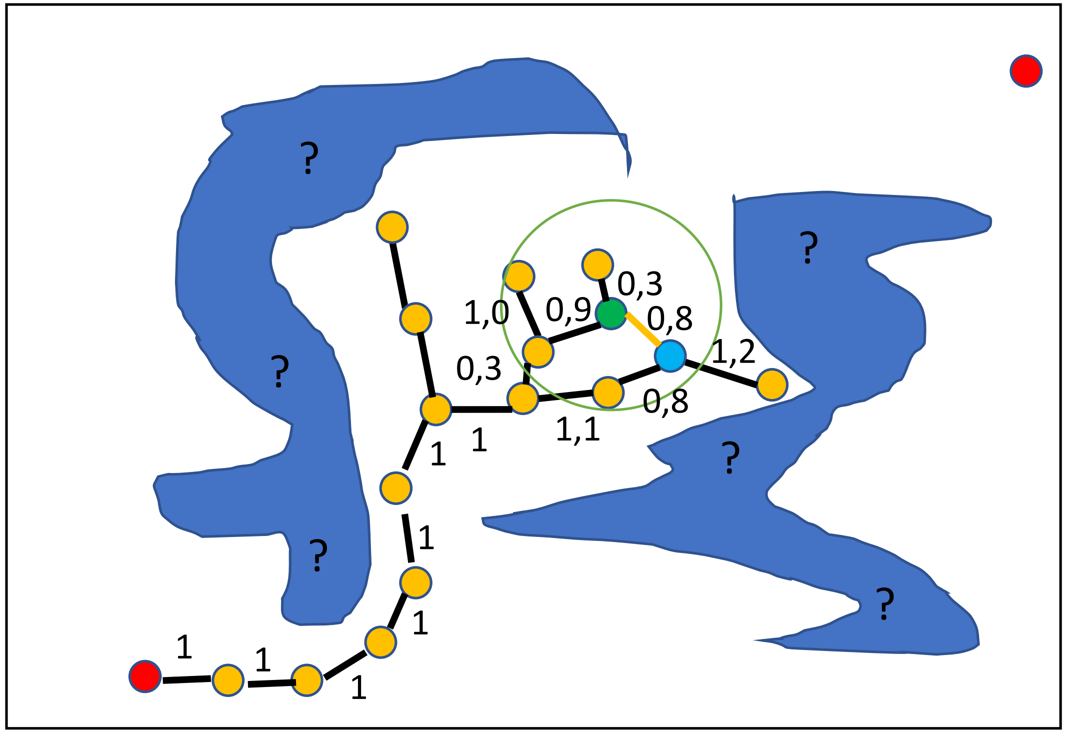


Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\};\ E \leftarrow \emptyset;$
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow$ SampleFree$_i$;
  $x_{\text{nearest}} \leftarrow$ Nearest$(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow$ Steer$(x_{\text{nearest}}, x_{\text{rand}})$;
  if ObtacleFree$(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow$ Near$(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT*}}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\text{min}} \leftarrow x_{\text{nearest}};\ c_{\text{min}} \leftarrow$ Cost$(x_{\text{nearest}}) + c($Line$(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if CollisionFree$(x_{\text{near}}, x_{\text{new}}) \wedge$ Cost$(x_{\text{near}}) + c($Line$(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}};\ c_{\text{min}} \leftarrow$ Cost$(x_{\text{near}}) + c($Line$(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if CollisionFree$(x_{\text{new}}, x_{\text{near}}) \wedge$ Cost$(x_{\text{new}}) + c($Line$(x_{\text{new}}, x_{\text{near}})) <$ Cost(x_{near})
      then $x_{\text{parent}} \leftarrow$ Parent(x_{near});
        $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

Note: This connection is not → no rewire

How does it work?

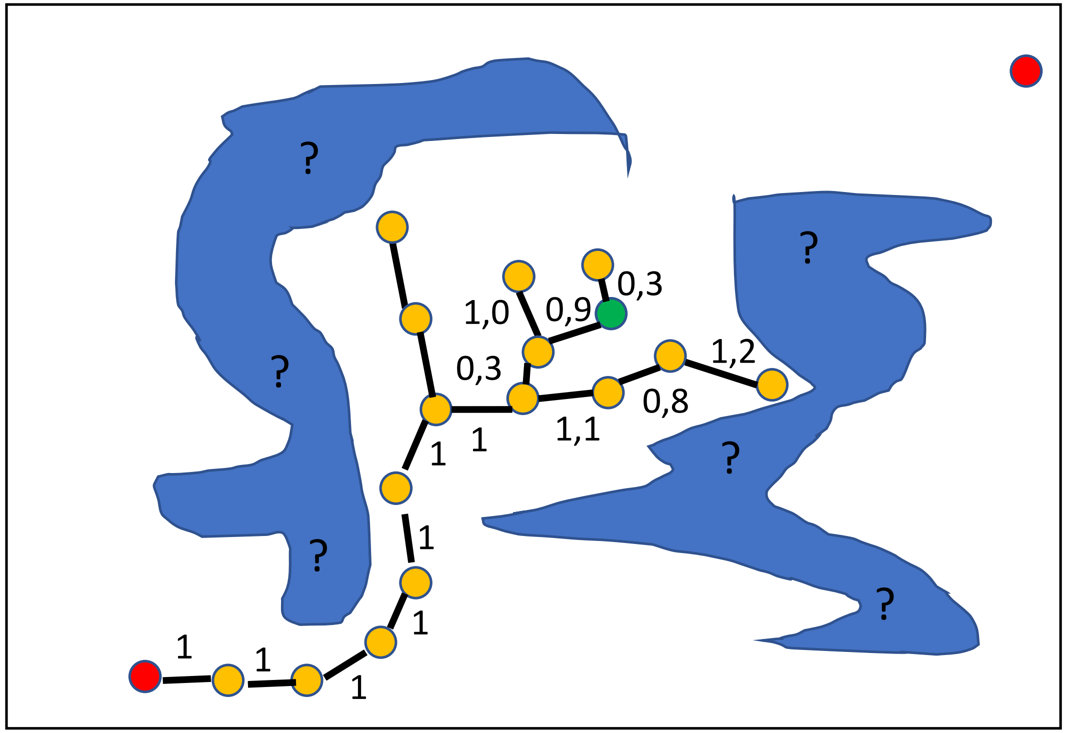


Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow$ `SampleFree`$_i$;
  $x_{\text{nearest}} \leftarrow$ `Nearest`$(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow$ `Steer`$(x_{\text{nearest}}, x_{\text{rand}})$;
  if `ObtacleFree`$(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow$ `Near`$(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}(V))/\text{card}(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\text{min}} \leftarrow x_{\text{nearest}}$; $c_{\text{min}} \leftarrow$ `Cost`$(x_{\text{nearest}}) + c($`Line`$(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if `CollisionFree`$(x_{\text{near}}, x_{\text{new}}) \wedge$ `Cost`$(x_{\text{near}}) + c($`Line`$(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$; $c_{\text{min}} \leftarrow$ `Cost`$(x_{\text{near}}) + c($`Line`$(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if `CollisionFree`$(x_{\text{new}}, x_{\text{near}}) \wedge$ `Cost`$(x_{\text{new}}) + c($`Line`$(x_{\text{new}}, x_{\text{near}})) <$ `Cost`(x_{near})
      then $x_{\text{parent}} \leftarrow$ `Parent`(x_{near});
        $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

Note: This connection is not → no rewire

How does it work?



Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT*}}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\text{min}} \leftarrow x_{\text{nearest}}$; $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$; $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
      then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

Note: Repeat!

Exercise

- Write down an example and make a few iterations with the algorithm.
  - You can ignore C_{obs} and only consider the tree.
  - Iterate until you did some rewires
  - Write down each iteration.

Summary

- We have seen that is possible to use sampling-based motion planners to find optimal path.
- The iterations take longer and are a bit more complex to implement.
- These algorithms are struggling to find path in narrow passage and in high dimension.
- In practice the Star algorithms are combined with classic approaches:
  - Find one feasible path with the RRT
  - Start the iterations of the RRT* in order to improve the solution path or find a new path that is more optimal.
- For most of the "classical" motion planner there a "Star version" has now been published.

Optimisation for Motion Planning

- A well-known algorithm for optimisation is the so-called ACO (Ant colony algorithm)
- Invented in 1991 it is a well-known **biological motivated** algorithm.
- It is **the** example aglorithm for swarm intelligence.
- The colony is more intelligent and capable of doing things than the individula ant.
- How does it work?

→ *We will try to understand how the ACO works and how it can be applied to motion planning.*

The idea of ACO

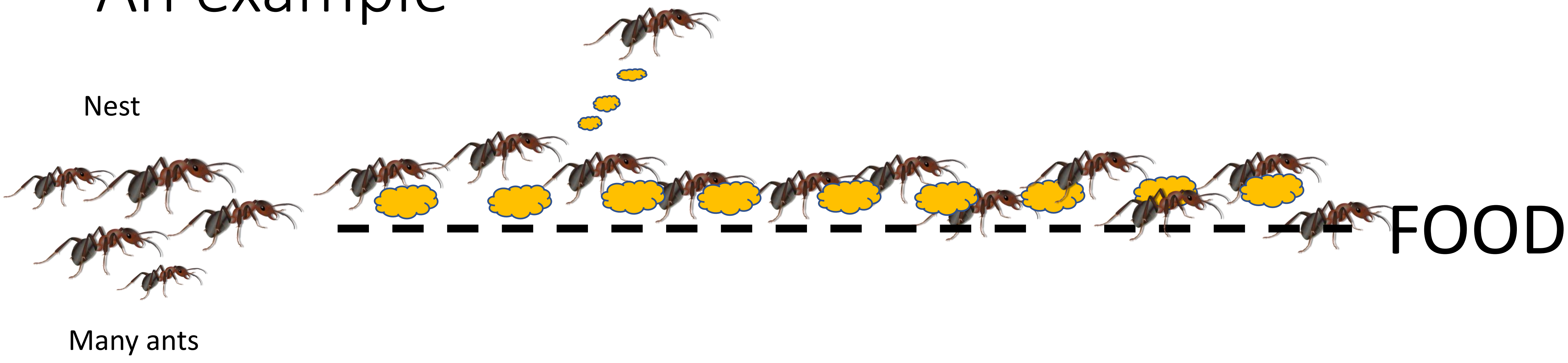
- A huge number of ants is able to find a shortest path from one location to the other.
- One single ant does not know the shortest path.
- The ant use pheromones that they drop on the path they are walking
- Other ants are attracted by the pheromones.

An example

FOOD

Many ants

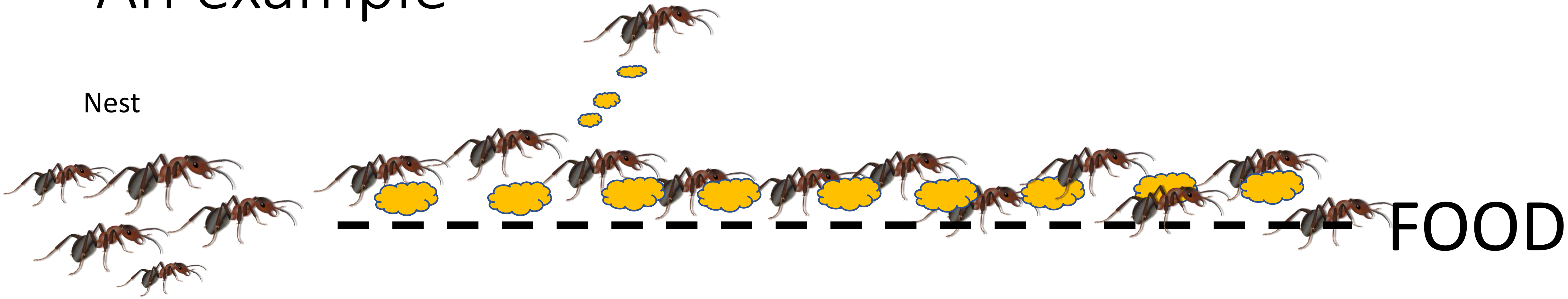
An example



- Assume they have already found the path in the past.
- Many pheromons are dropped on the floor.
- Each ant can only drop a certain amount of pheromones.
- The ant follow the pheromons of other ants with a high probabilty.

An example

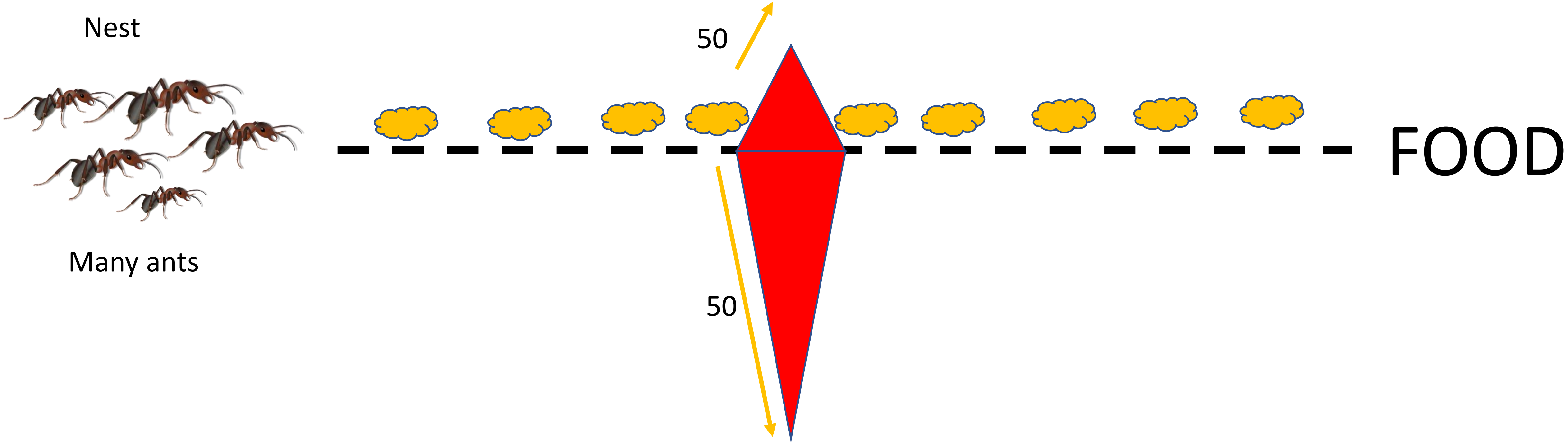
Nest



Many ants

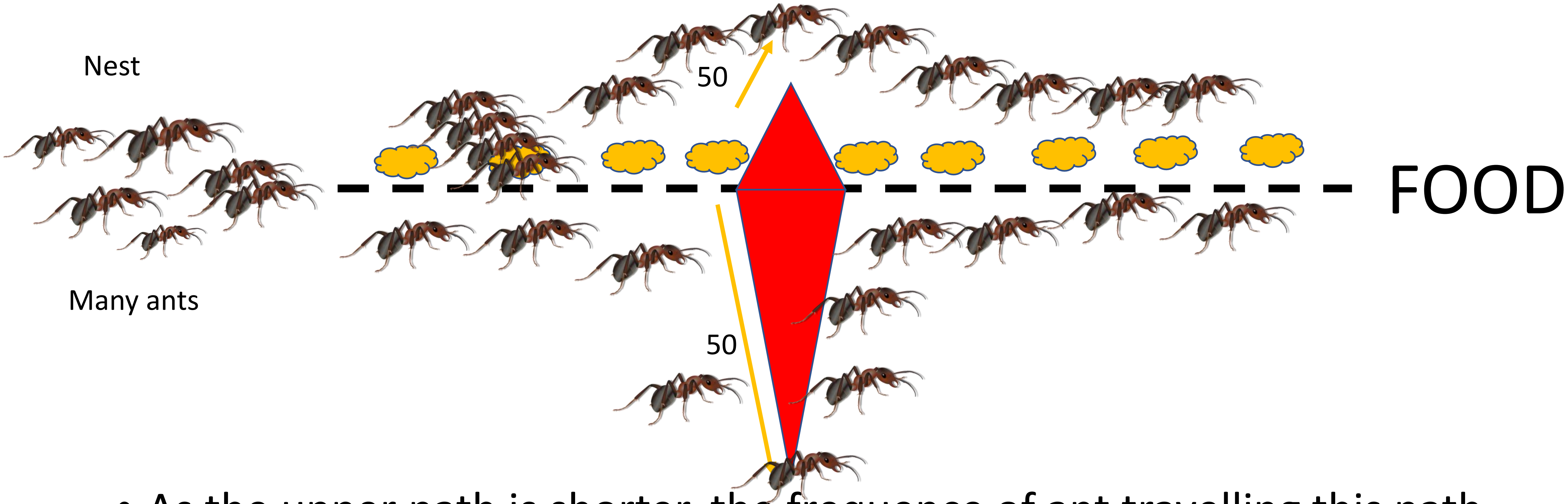
- The probability is proportional to the amount of pheromones.
- If many ant walk the path, the pheromones are accumulating.
- There is the chance that sometimes an ant gets „lost“. But the trail of pheromones it is leaving behind is small.
- On the shortest path to the food, the pheromones will be accumulating the most.

An example



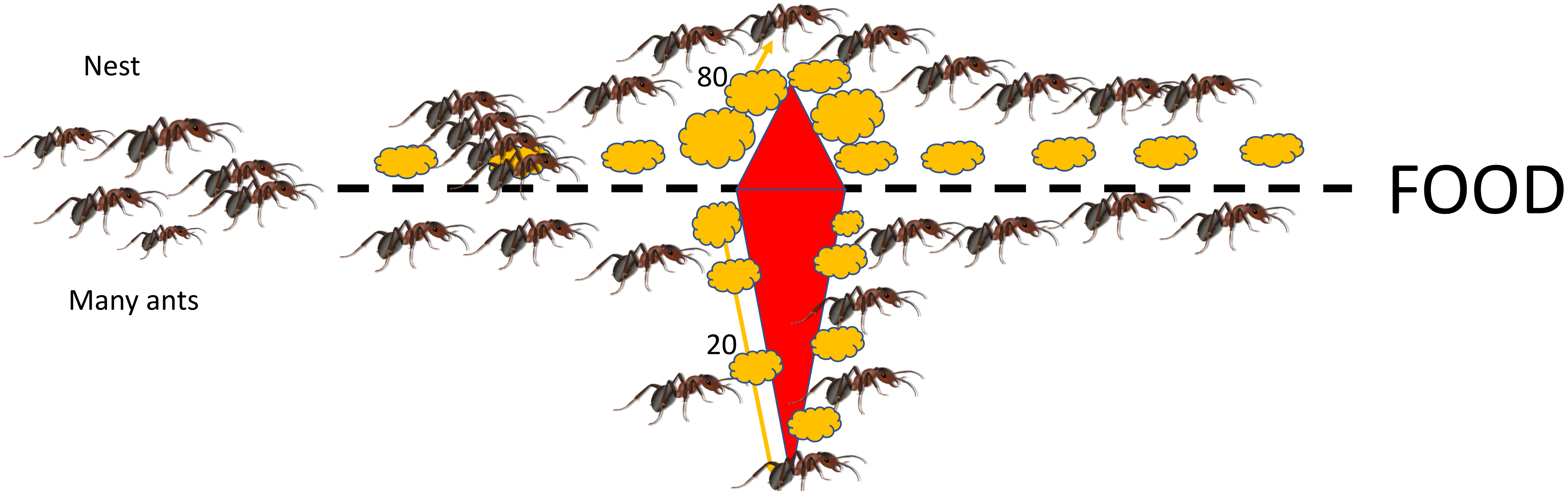
- If an obstacle suddenly appears, the chance of the ants are 50/50 to take the one path or the other.

An example



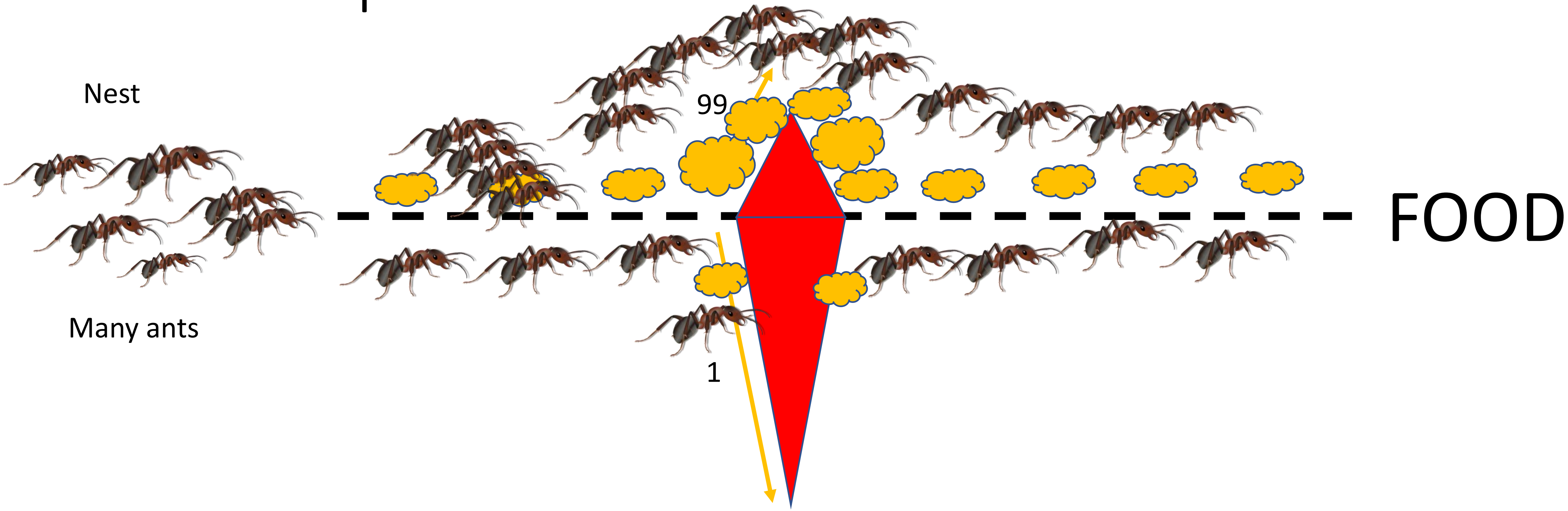
- As the upper path is shorter, the frequence of ant travelling this path is higher.
- Note: ants are travelling to the Food and back!

An example



- As more ant are travelling the upper path, more pheromens are dropped on this path.
- This changes the probablity of ants taking one way or the other.

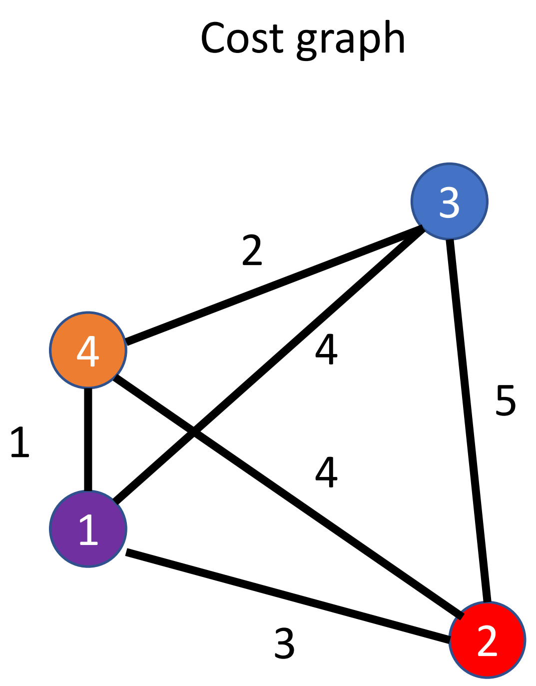
An example

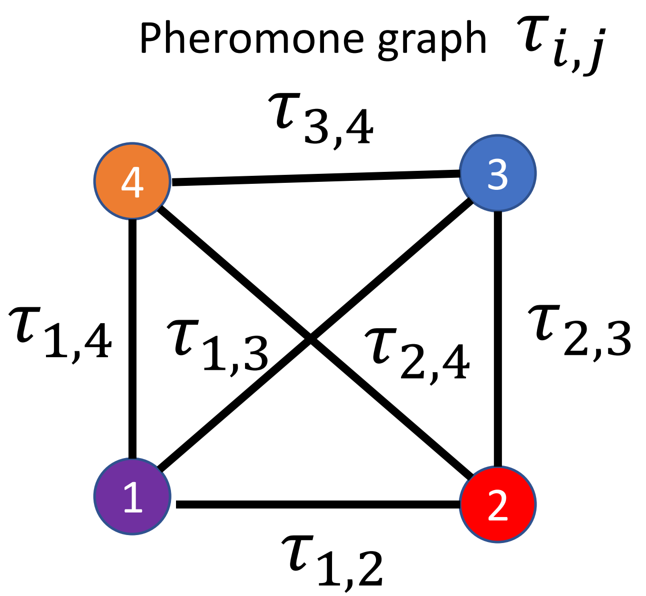


- From time to time more ants are taking the upper path and the probablity for the lower path get less.
- Even more so, the algorithm make the pheromens evaporate.

Video

How to model?





Each edge describes the distance in the graph

Each edge describes the level of pheromones

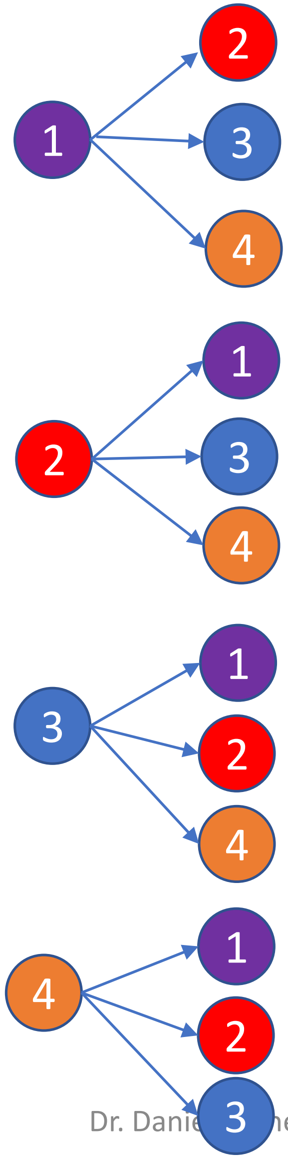
Exercise

- What kind of data structure would you use for the cost graph?
- What kind of data structure would you use for the pheromone graph?

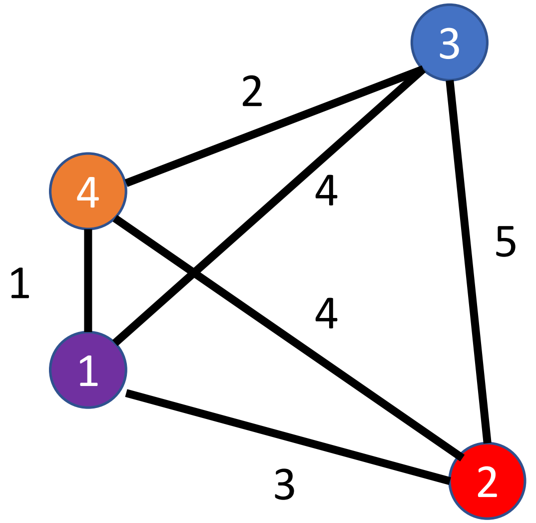
Note: Reason why you would you that sort of data structure.

How to model?

Linked List

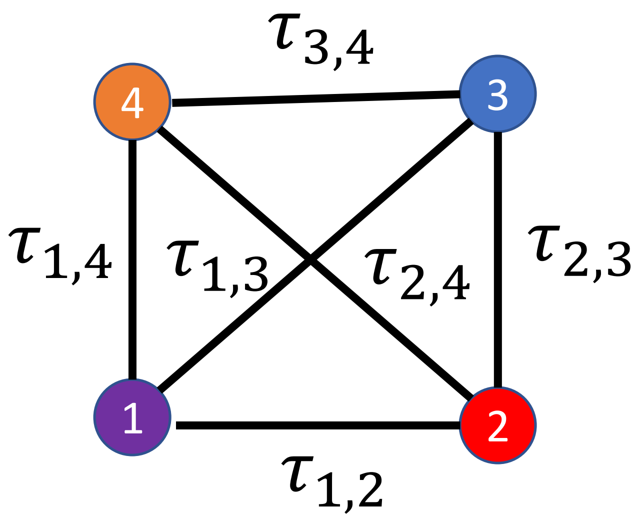


Cost graph



- To simulate the ant walk, you have to travel through the graph.
- A linked list enable you to easily travel through the graph.

Pheromone graph



Matrix

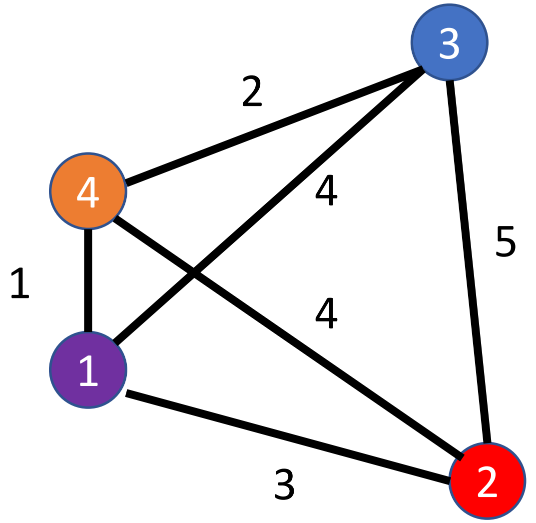
	1	2	3	4
1	0	$\tau_{1,2}$	$\tau_{1,3}$	$\tau_{1,4}$
2	0	0	$\tau_{2,3}$	$\tau_{2,4}$
3	0	0	0	$\tau_{3,4}$
4	0	0	0	0

- You only need to access the pheromones level on each edge.
- Access a value in a matrix is fast.
- You do not travel through this graph.
- Upper triangle matrices are easy and simple to store in modern programming languages.

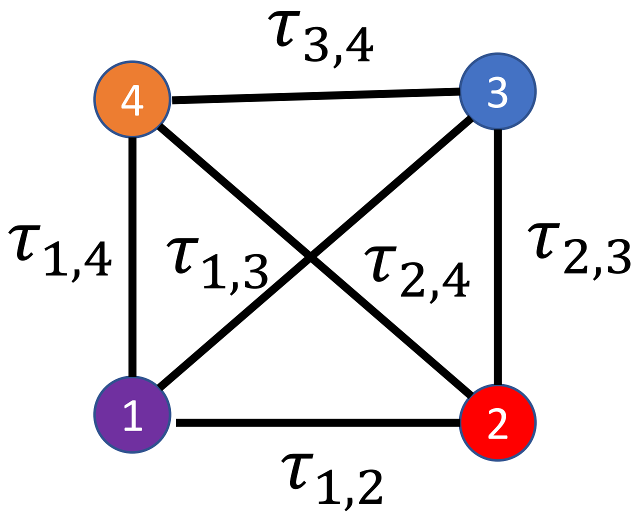
How to model?

How to compute this pheromone level?

Cost graph



Pheromone graph $\tau_{i,j}$



Each edge describes the distance in the graph

Each edge describes the level of pheromones

How to model?

This is an ant

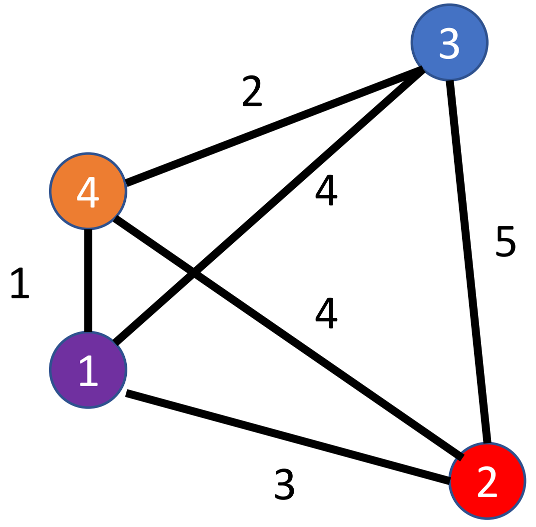
This is the length of the ants travel from nest to food

$$\Delta\tau_{i,j}^{k} = \begin{cases} 1/L_k \\ 0 \end{cases}$$

This is the first edge index

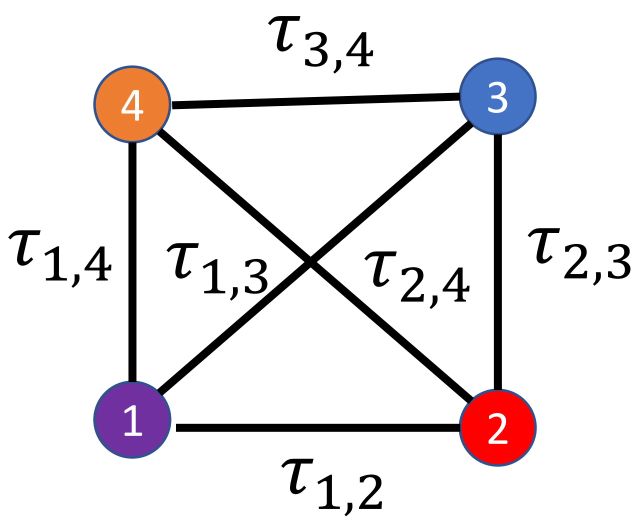
This is the second edge index

Cost graph



Each edge describes the distance in the graph

Pheromone graph $\tau_{i,j}$



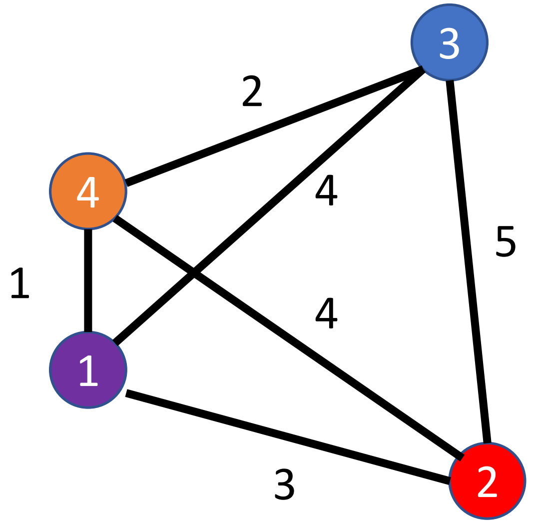
Each edge describes the level of pheromones

How to model?

m is the amount of total ants

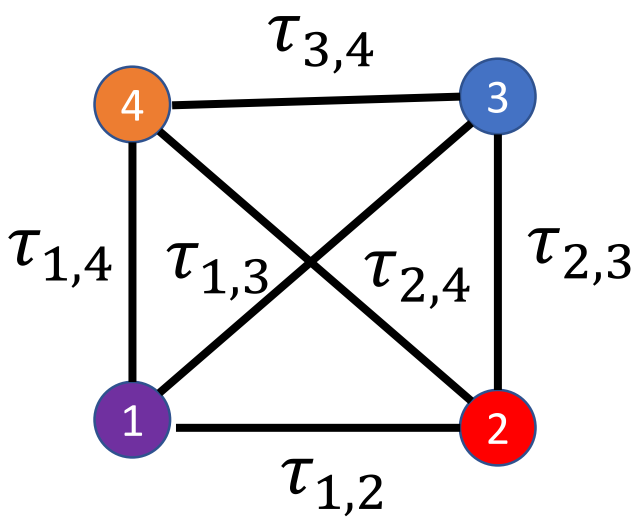
$$\Delta\tau_{i,j}^{k} = \begin{cases} 1/L_k \\ 0 \end{cases}$$

Cost graph



Each edge describes the distance in the graph

Pheromone graph $\tau_{i,j}$



Each edge describes the level of pheromones

$$\tau_{i,j} = (1-\rho)\tau_{i,j}^{*} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}$$

This is the pheromone level on edge i,j

See above

This is the pheromone level on the edge in the last iteration

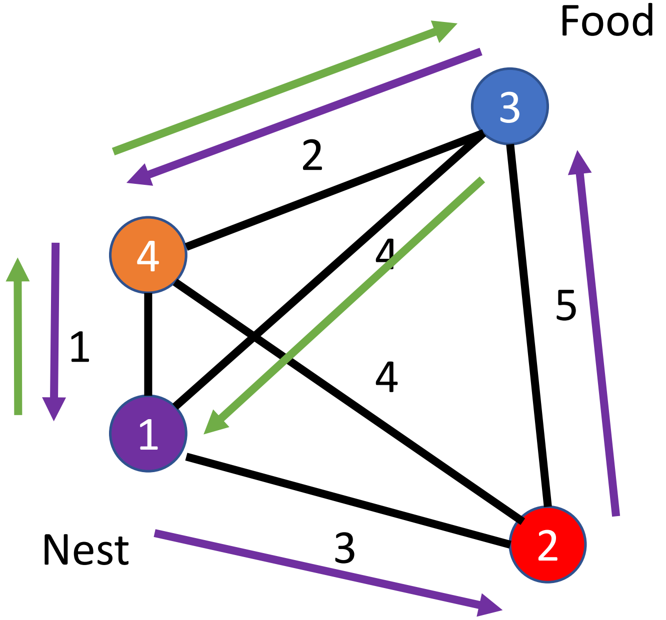
This is a constant that simulates evaporation

Exercise

$$\Delta\tau_{i,j}^{k} = \begin{cases} 1/L_k \\ 0 \end{cases}$$

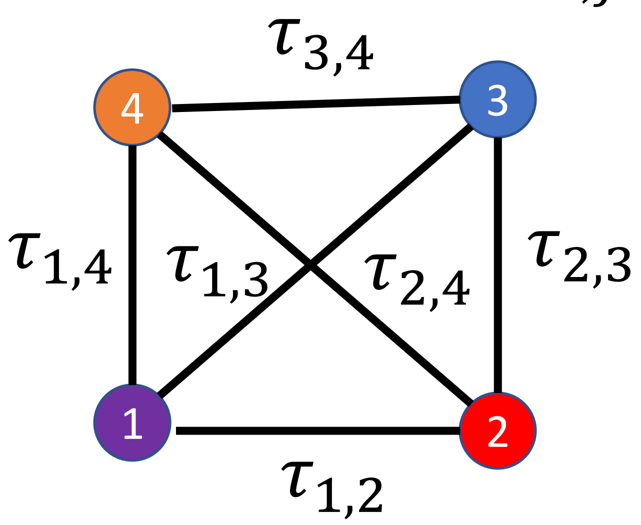
$$\tau_{i,j} = (1-\rho)\tau_{i,j}^{*} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}$$

Cost graph



Each edge describes the distance in the graph

Pheromone graph $\tau_{i,j}$



Each edge describes the level of pheromones

- Compute the new pheromone level $\tau_{i,j}$ After two ant travel, as shown on the left.
- Give the result in a matrix form
- The initial values of $\tau_{i,j}$ is 0,5
- Also give the initial matrix.
- $\rho = 0$ (no evaporation)
- Ants start on node 1

Solution

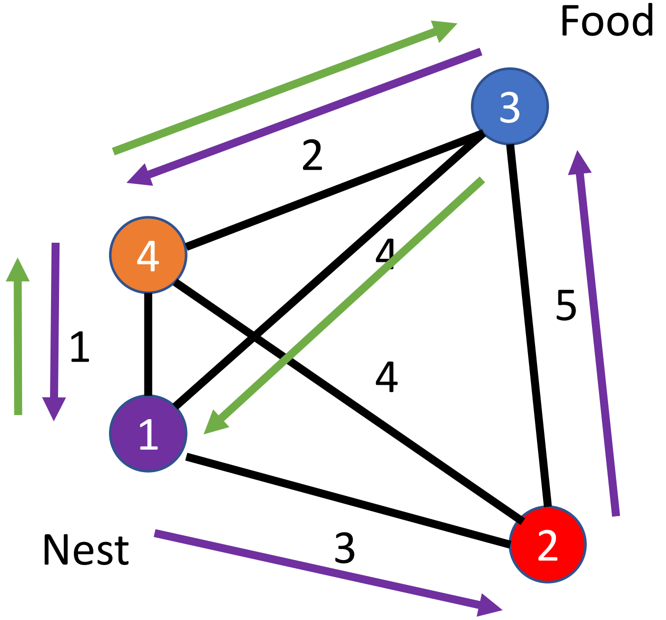
Initial Matrix

0	0,5	0,5	0,5
0	0	0,5	0,5
0	0	0	0,5
0	0	0	0

$$\Delta\tau_{i,j}^{k} = \begin{cases} 1/L_k \\ 0 \end{cases}$$

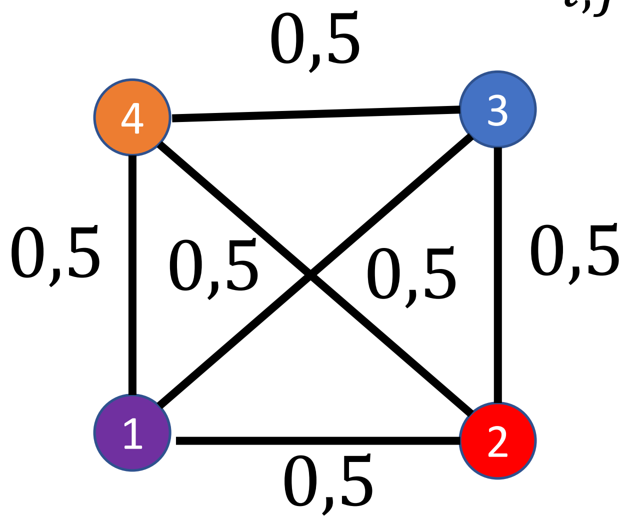
$$\tau_{i,j} = (1-\rho)\tau_{i,j}^{*} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}$$

Cost graph



Each edge describes the distance in the graph

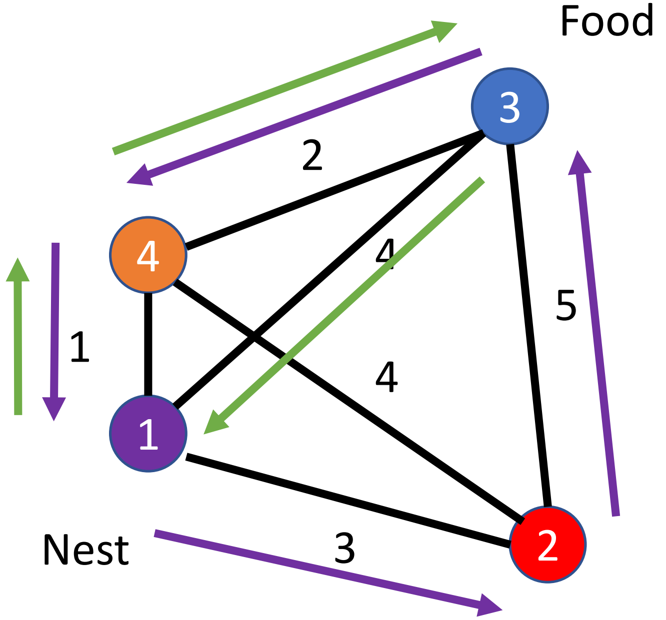
Pheromone graph $\tau_{i,j}$



Each edge describes the level of pheromones

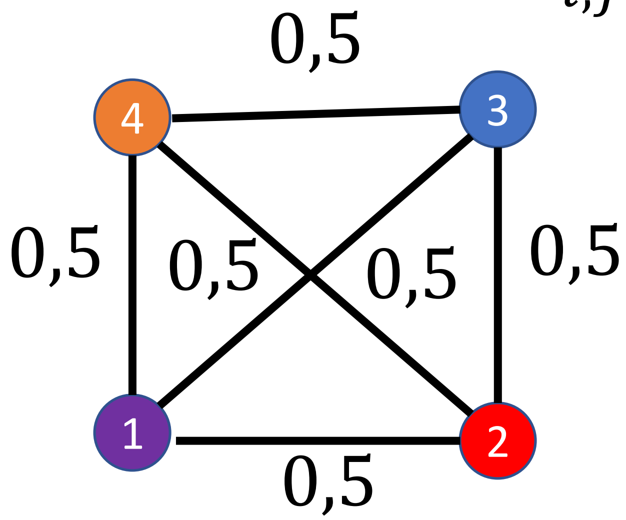
Solution

Cost graph



Each edge describes the distance in the graph

Pheromone graph $\tau_{i,j}$



Each edge describes the level of pheromones

Initial Matrix

0	0,5	0,5	0,5
0	0	0,5	0,5
0	0	0	0,5
0	0	0	0

$$\Delta\tau_{i,j}^{k} = \begin{cases} 1/L_k \\ 0 \end{cases}$$

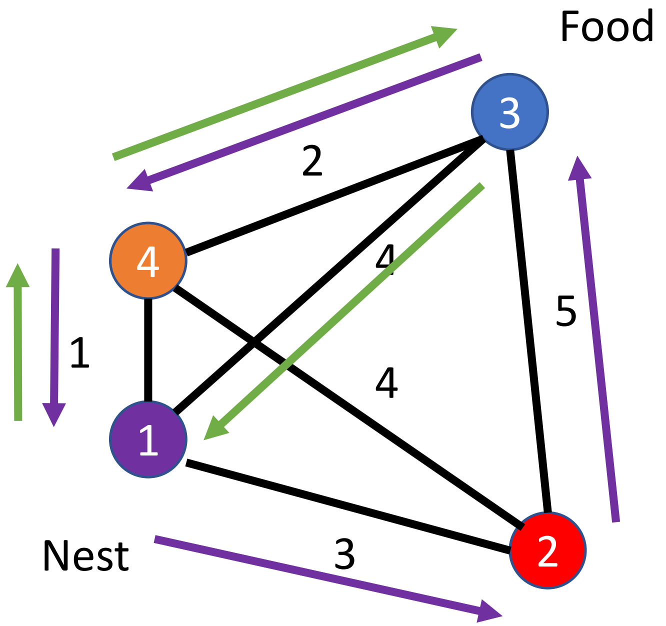
$$\tau_{i,j} = (1-\rho)\tau_{i,j}^{*} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}$$

$$L_1 = 3 + 5 + 2 + 1 = 11$$

$$\Delta\tau_{i,j}^{1} = \frac{1}{11} = 0{,}09$$

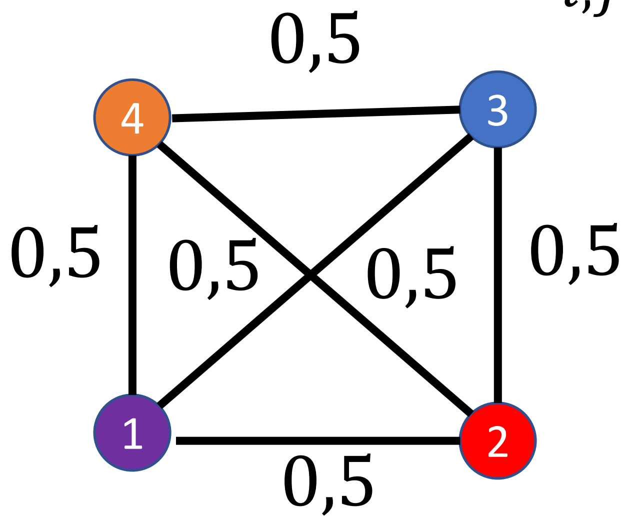
Solution

Cost graph



Each edge describes the distance in the graph

Pheromone graph $\tau_{i,j}$



Each edge describes the level of pheromones

Initial Matrix

$$\begin{matrix} 0 & 0{,}5 & 0{,}5 & 0{,}5 \\ 0 & 0 & 0{,}5 & 0{,}5 \\ 0 & 0 & 0 & 0{,}5 \\ 0 & 0 & 0 & 0 \end{matrix}$$

$$\Delta\tau_{i,j}^{k} = \begin{cases} 1/L_k \\ 0 \end{cases}$$

$$\tau_{i,j} = (1-\rho)\tau_{i,j}^{*} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}$$

$$L_1 = 3 + 5 + 2 + 1 = 11$$

$$\Delta\tau_{i,j}^{1} = \frac{1}{11} = 0{,}09$$

$$L_2 = 1 + 2 + 4 = 7$$

$$\Delta\tau_{i,j}^{2} = \frac{1}{7} = 0{,}14$$

Solution

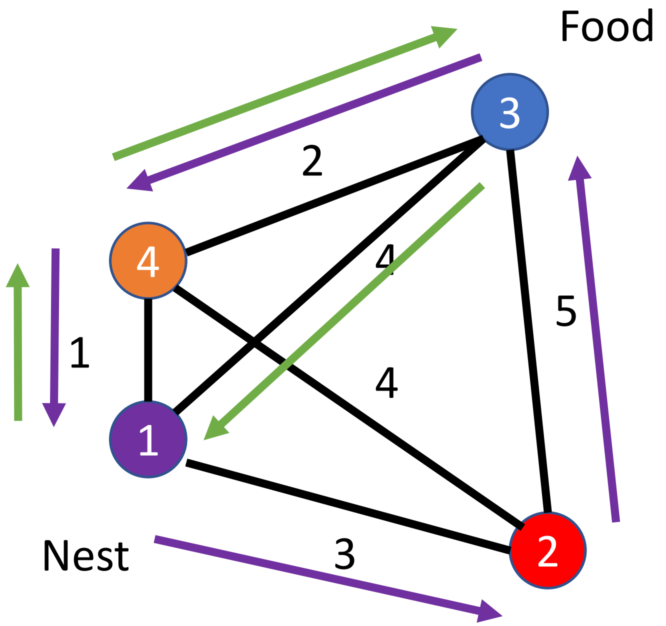
Initial Matrix

$$\begin{matrix} 0 & 0{,}5 & 0{,}5 & 0{,}5 \\ 0 & 0 & 0{,}5 & 0{,}5 \\ 0 & 0 & 0 & 0{,}5 \\ 0 & 0 & 0 & 0 \end{matrix}$$

$$\Delta\tau_{i,j}^{k} = \begin{cases} 1/L_k \\ 0 \end{cases}$$

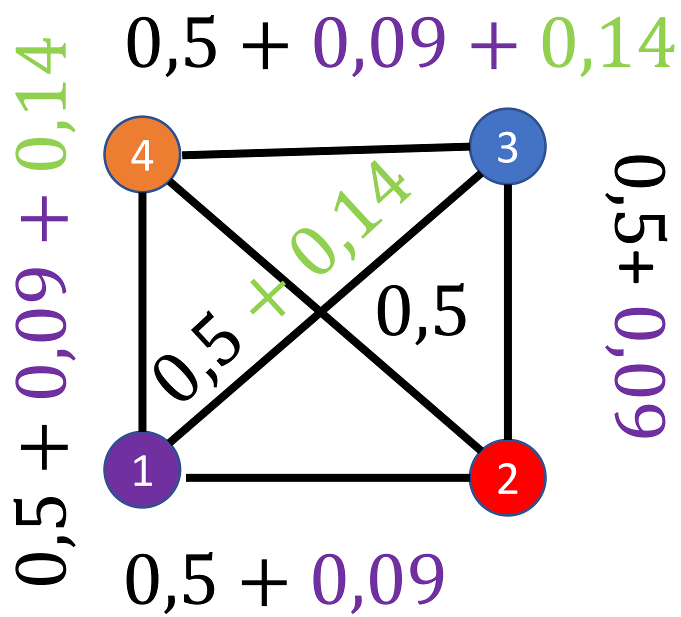
$$\tau_{i,j} = (1-\rho)\tau_{i,j}^{*} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}$$

Cost graph



Each edge describes the distance in the graph

Pheromone graph $\tau_{i,j}$



Each edge describes the level of pheromones

$$L_1 = 3 + 5 + 2 + 1 = 11$$

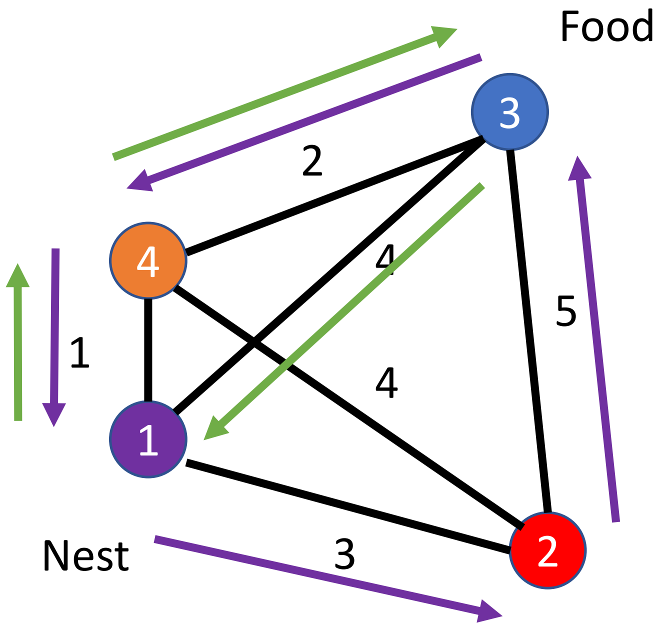
$$\Delta\tau_{i,j}^{1} = \frac{1}{11} = 0{,}09$$

$$L_2 = 1 + 2 + 4 = 7$$

$$\Delta\tau_{i,j}^{2} = \frac{1}{7} = 0{,}14$$

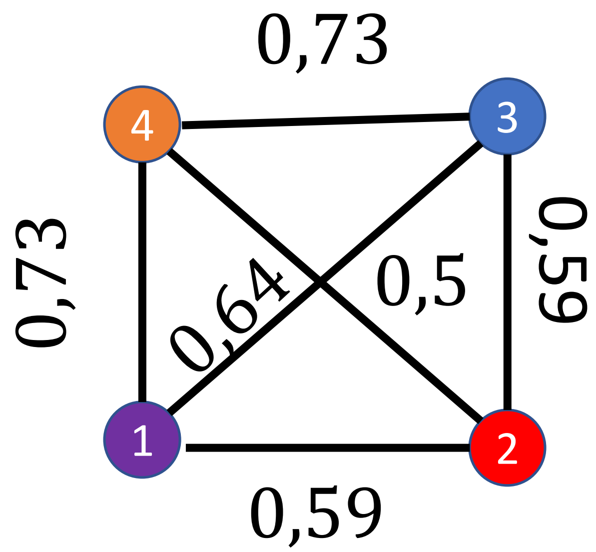
Solution

Cost graph



Each edge describes the distance in the graph

Pheromone graph $\tau_{i,j}$



Each edge describes the level of pheromones

Initial Matrix

0	0,5	0,5	0,5
0	0	0,5	0,5
0	0	0	0,5
0	0	0	0

$$\Delta\tau_{i,j}^{k} = \begin{cases} 1/L_k \\ 0 \end{cases}$$

$$\tau_{i,j} = (1-\rho)\tau_{i,j}^{*} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}$$

$$L_1 = 3 + 5 + 2 + 1 = 11$$

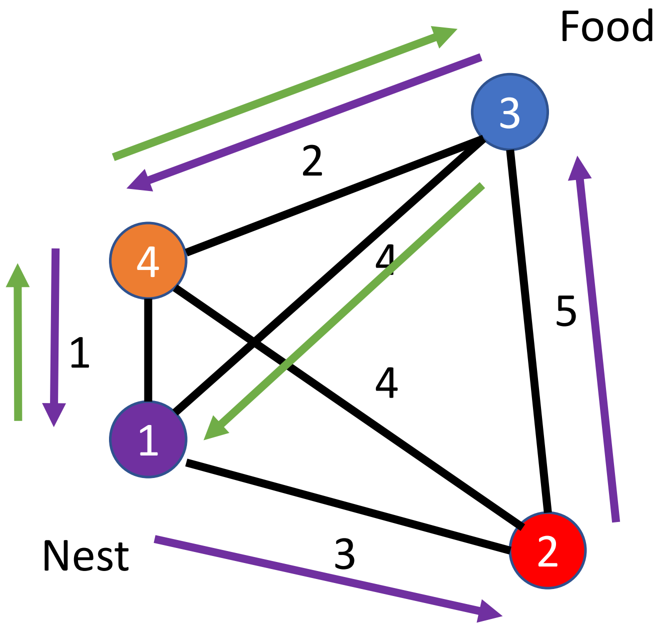
$$\Delta\tau_{i,j}^{1} = \frac{1}{11} = 0{,}09$$

$$L_2 = 1 + 2 + 4 = 7$$

$$\Delta\tau_{i,j}^{2} = \frac{1}{7} = 0{,}14$$

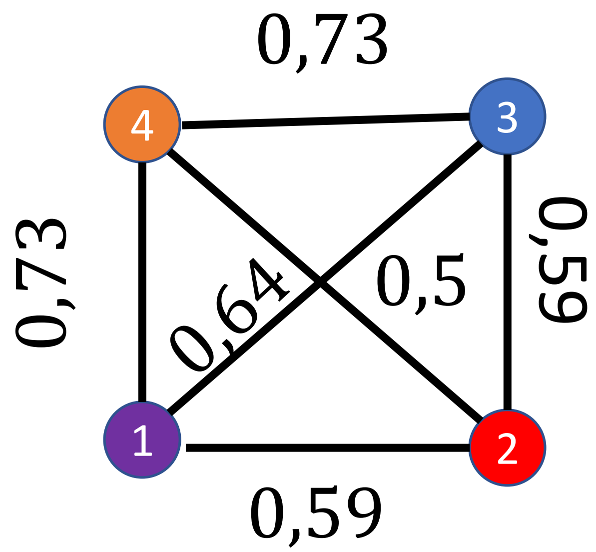
Solution

Cost graph



Each edge describes the distance in the graph

Pheromone graph $\tau_{i,j}$



Each edge describes the level of pheromones

Initial Matrix

$$\begin{matrix} 0 & 0,5 & 0,5 & 0,5 \\ 0 & 0 & 0,5 & 0,5 \\ 0 & 0 & 0 & 0,5 \\ 0 & 0 & 0 & 0 \end{matrix}$$

$$\Delta\tau_{i,j}^{k} = \begin{cases} 1/L_k \\ 0 \end{cases}$$

$$\tau_{i,j} = (1-\rho)\tau_{i,j}^{*} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}$$

$$L_1 = 3 + 5 + 2 + 1 = 11$$

$$\Delta\tau_{i,j}^{1} = \frac{1}{11} = 0,09$$

$$L_2 = 1 + 2 + 4 = 7$$

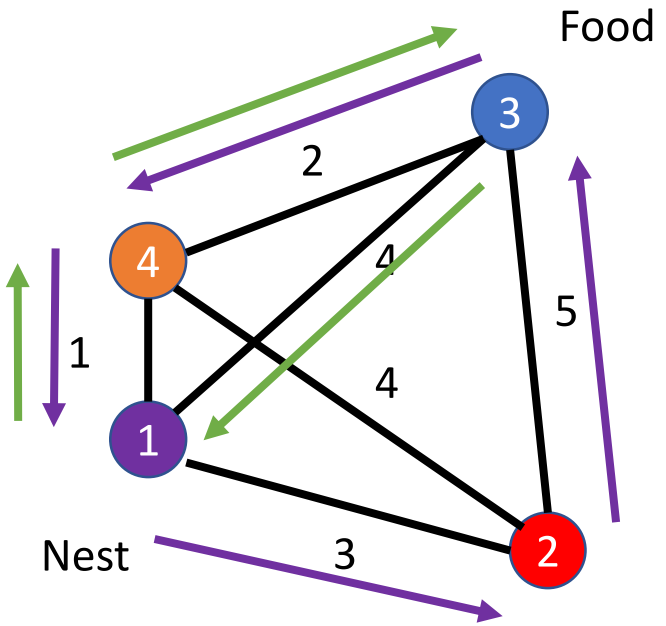
$$\Delta\tau_{i,j}^{2} = \frac{1}{7} = 0,14$$

Result Matrix

$$\begin{matrix} 0 & 0,59 & 0,64 & 0,73 \\ 0 & 0 & 0,59 & 0,5 \\ 0 & 0 & 0 & 0,73 \\ 0 & 0 & 0 & 0 \end{matrix}$$

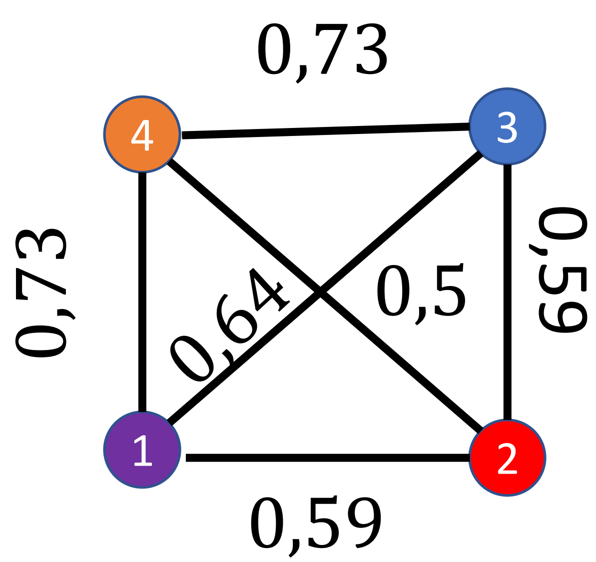
How to decide on a path for the ant?

Cost graph



Each edge describes the distance in the graph

Pheromone graph $\tau_{i,j}$



Each edge describes the level of pheromones

This is the pheromone level

$$P_{i,j} = \frac{\tau_{i,j} * \eta_{ij}}{\sum_{all\ neighbours\ of\ i} \tau_{i,j} * \eta_{ij}}$$

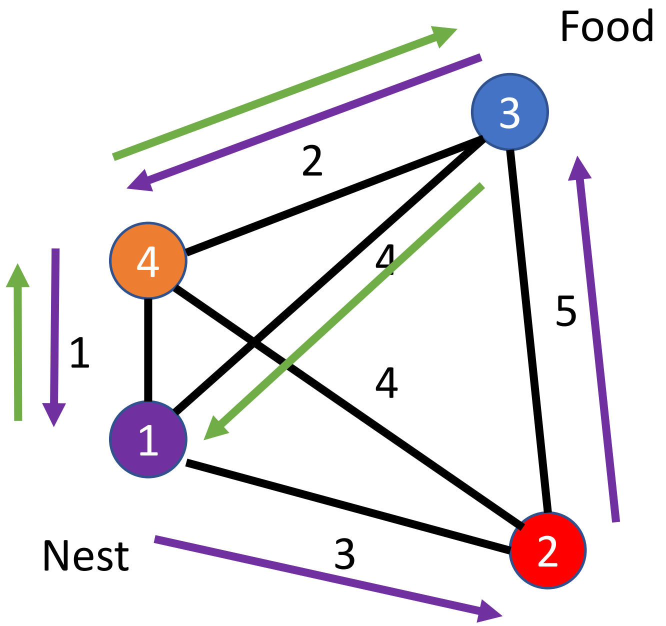
$$\eta_{ij} = \frac{1}{L_{i,j}}$$

Sum all product of pheromone multiplied by inverse edge distance that are originated in i

Inverse of the edge distance

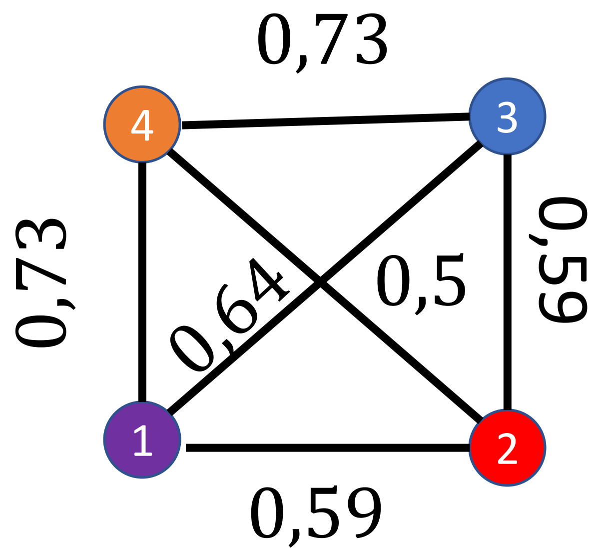
Exercise

Cost graph



Each edge describes the distance in the graph

Pheromone graph $\tau_{i,j}$



Each edge describes the level of pheromones

Compute the probabilities for an ant, starting in node 1.

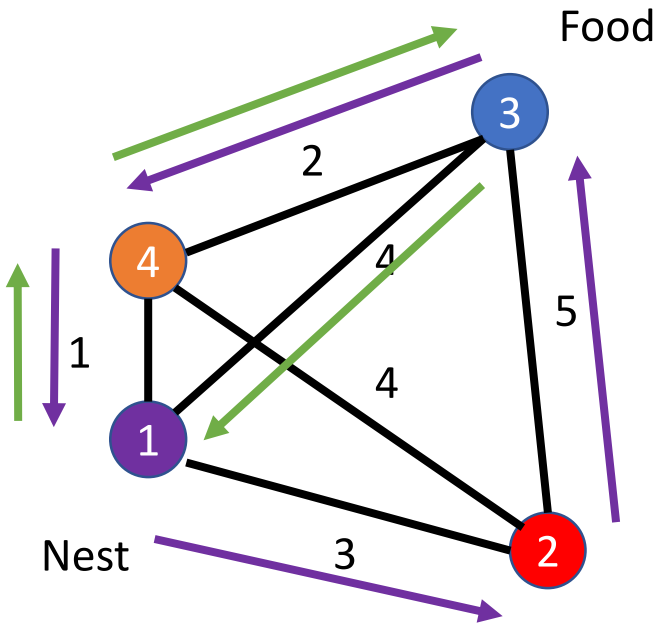
Note: Compute the probability for each edge that starts in 1.

$$P_{i,j} = \frac{\tau_{i,j} * \eta_{ij}}{\sum_{all\ neighbours\ of\ i} \tau_{i,j} * \eta_{ij}}$$

$$\eta_{ij} = \frac{1}{L_{i,j}}$$

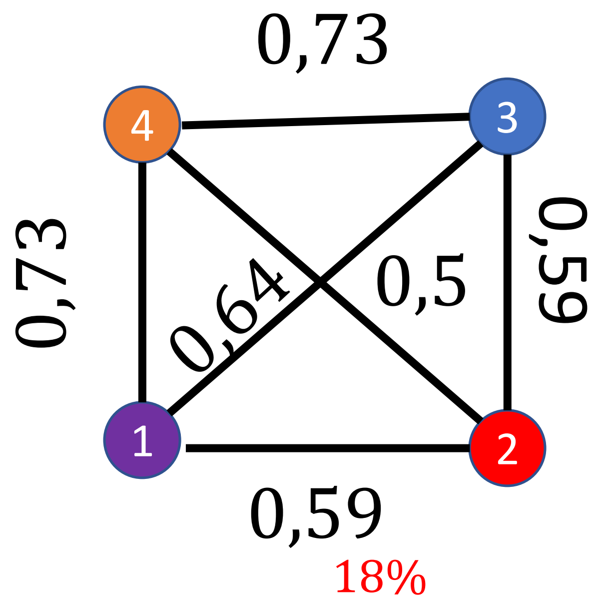
Solution

Cost graph



Each edge describes the distance in the graph

Pheromone graph $\tau_{i,j}$



Each edge describes the level of pheromones

Compute the probabilities for an ant, starting in node 1.

$$P_{i,j} = \frac{\tau_{i,j} * \eta_{ij}}{\sum_{all\ neighbours\ of\ i} \tau_{i,j} * \eta_{ij}}$$

$$\eta_{ij} = \frac{1}{L_{i,j}}$$

$$P_{1,2} = \frac{\tau_{i,j} * \eta_{ij}}{\sum_{all\ neighbours\ of\ i} \tau_{i,j} * \eta_{ij}}$$

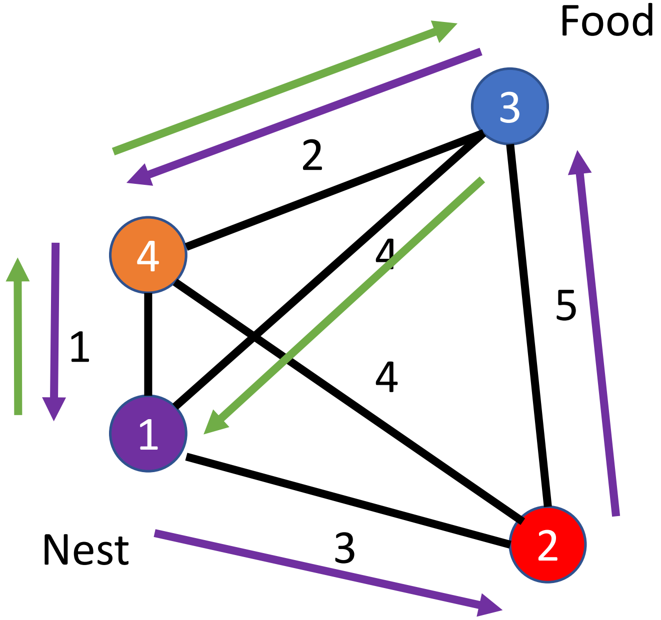
$$= \frac{\tau_{1,2} * \eta_{1,2}}{\tau_{1,2} * \eta_{1,2} + \tau_{1,3} * \eta_{1,3} + \tau_{1,4} * \eta_{1,4}}$$

$$= \frac{0{,}59 * \frac{1}{3}}{0{,}59 * \frac{1}{3} + 0{,}64 * \frac{1}{4} + 0{,}73 * \frac{1}{1}}$$

$$= \frac{0{,}19}{0{,}19 + 0{,}16 + 0{,}73} = 0{,}18 = 18\%$$

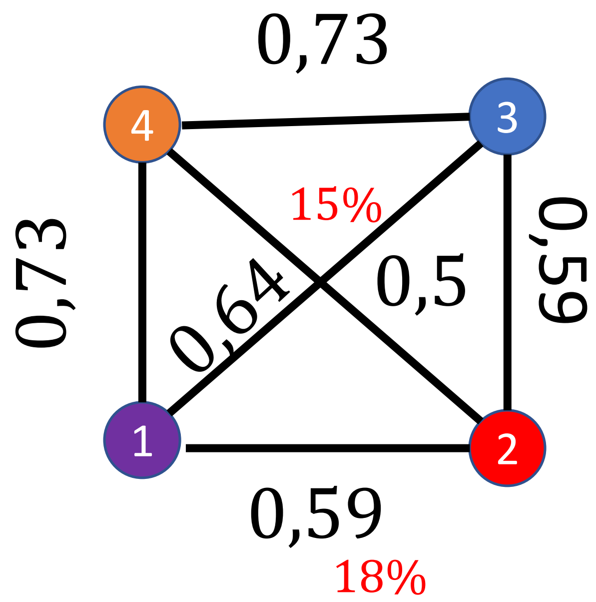
Solution

Cost graph



Each edge describes the distance in the graph

Pheromone graph $\tau_{i,j}$



Each edge describes the level of pheromones

Compute the probabilities for an ant, starting in node 1.

$$P_{i,j} = \frac{\tau_{i,j} * \eta_{ij}}{\sum_{all\ neighbours\ of\ i} \tau_{i,j} * \eta_{ij}}$$

$$\eta_{ij} = \frac{1}{L_{i,j}}$$

$$P_{1,2} = \frac{\tau_{i,j} * \eta_{ij}}{\sum_{all\ neighbours\ of\ i} \tau_{i,j} * \eta_{ij}}$$

$$= \frac{\tau_{1,2} * \eta_{1,2}}{\tau_{1,2} * \eta_{1,2} + \tau_{1,3} * \eta_{1,3} + \tau_{1,4} * \eta_{1,4}}$$

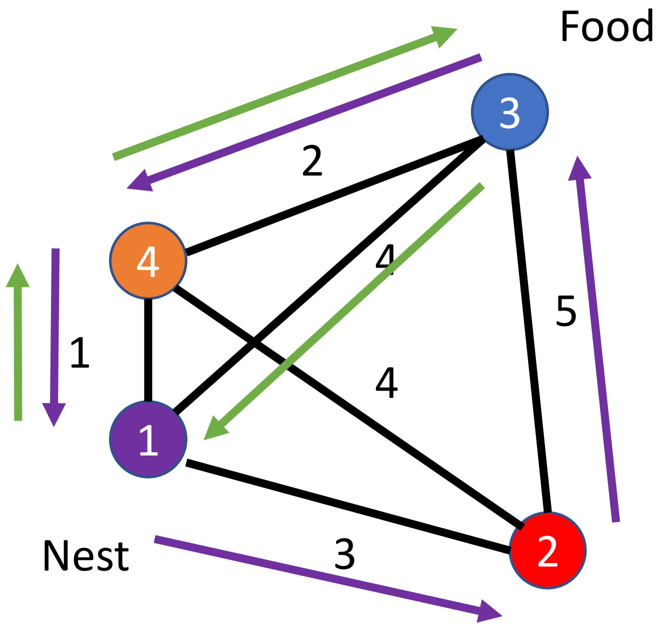
$$= \frac{0,59 * \frac{1}{3}}{0,59 * \frac{1}{3} + 0,64 * \frac{1}{4} + 0,73 * \frac{1}{1}}$$

$$= \frac{0,19}{0,19 + 0,16 + 0,73} = 0,18 = 18\%$$

$$P_{1,3} = \frac{0,16}{0,19 + 0,16 + 0,73} = 0,15 = 15\%$$

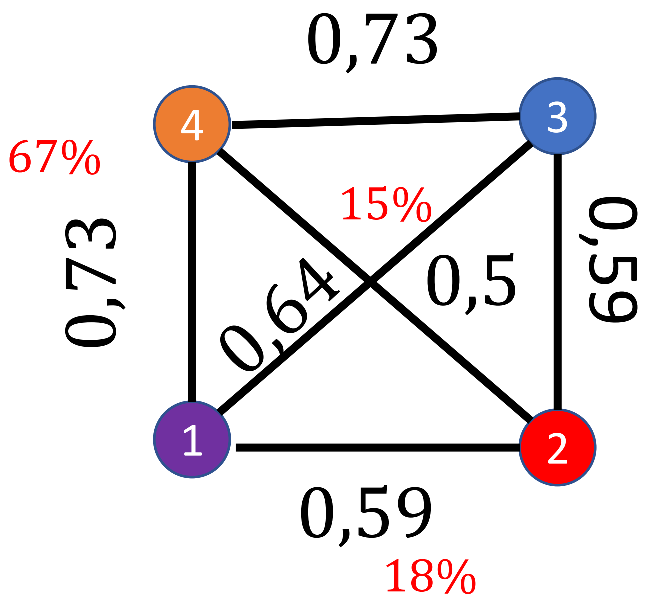
Solution

Cost graph



Each edge describes the distance in the graph

Pheromone graph $\tau_{i,j}$



Each edge describes the level of pheromones

Compute the probabilities for an ant, starting in node 1.

$$P_{i,j} = \frac{\tau_{i,j} * \eta_{ij}}{\sum_{all\ neighbours\ of\ i} \tau_{i,j} * \eta_{ij}}$$

$$\eta_{ij} = \frac{1}{L_{i,j}}$$

$$P_{1,2} = \frac{\tau_{i,j} * \eta_{ij}}{\sum_{all\ neighbours\ of\ i} \tau_{i,j} * \eta_{ij}}$$

$$= \frac{\tau_{1,2} * \eta_{1,2}}{\tau_{1,2} * \eta_{1,2} + \tau_{1,3} * \eta_{1,3} + \tau_{1,4} * \eta_{1,4}}$$

$$= \frac{0{,}59 * \frac{1}{3}}{0{,}59 * \frac{1}{3} + 0{,}64 * \frac{1}{4} + 0{,}73 * \frac{1}{1}}$$

$$= \frac{0{,}19}{0{,}19 + 0{,}16 + 0{,}73} = 0{,}18 = 18\%$$

$$P_{1,3} = \frac{0{,}16}{0{,}19 + 0{,}16 + 0{,}73} = 0{,}15 = 15\%$$

$$P_{1,4} = \frac{0{,}73}{0{,}19 + 0{,}16 + 0{,}73} = 0{,}67 = 67\%$$

Summary

- So we know how the ACO algorithm works.
- We now can compute pheromones, guide the ants and know what data structures need to be used.
- We have seen that simulating ants can compute shortest paths.

But how can this be used in Motion Planning?

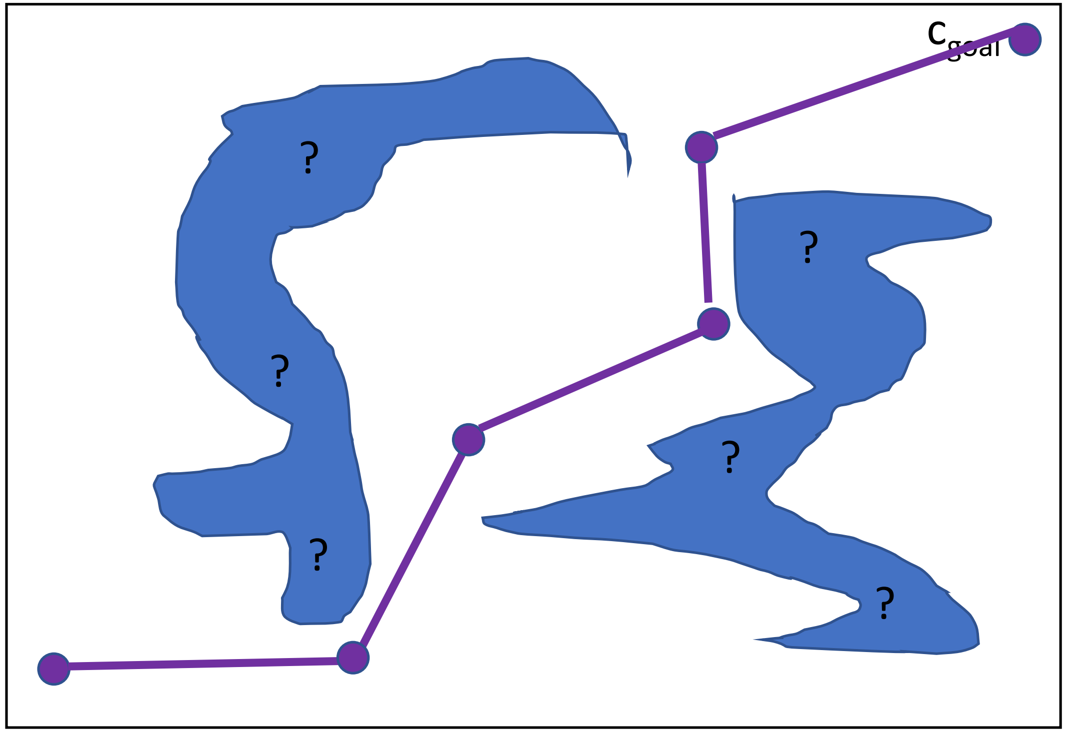
Exercise

Discuss in a group of two to three people, how to apply this idea to motion planning.

Think about these questions:

1. At what time of the algorithm would you apply this idea? Before/During/after the MP algo?
2. For what type of algorithms do you think this algorithm is most usefull? Roadmap-based algorithm? Tree-based algorithms? Single-query? Multi-query?...

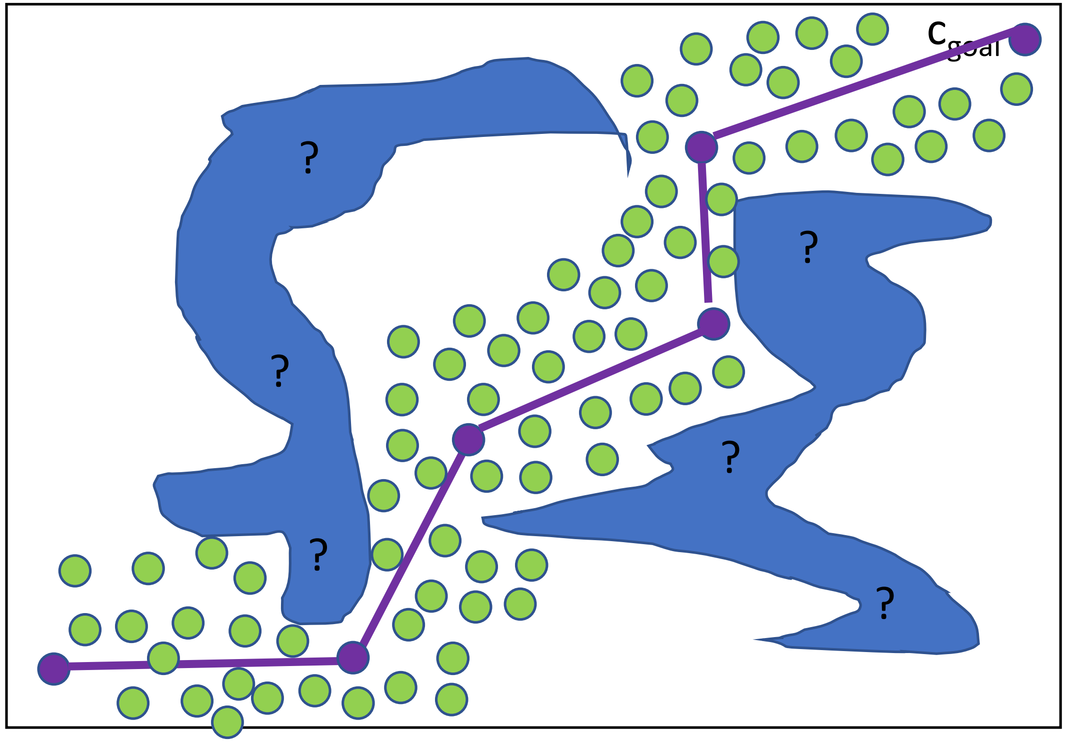
1.a Using ant to optimize the result path



Approach:

- Compute the path with a Tree-based planner.

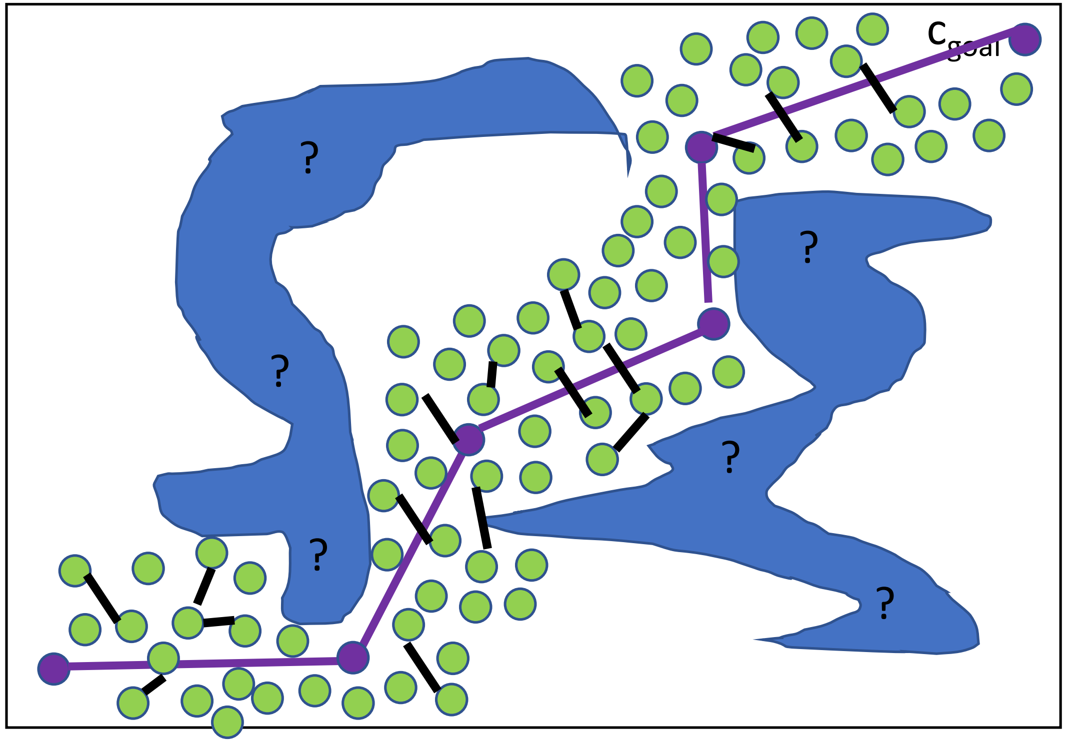
1.a Using ant to optimize the result path



Approach:

- Compute the path with a Tree-based planner.
- Sample the surrounding of the path.

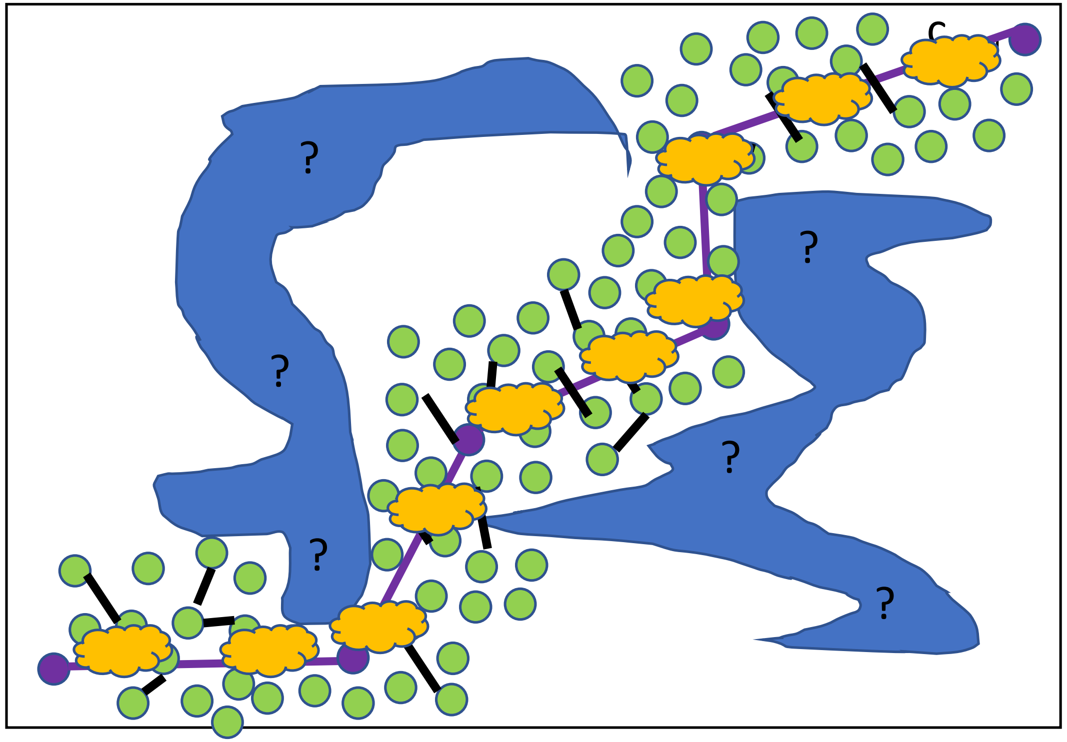
1.a Using ant to optimize the result path



Approach:

- Compute the path with a Tree-based planner.
- Sample the surrounding of the path.
- Connect to a graph, using NN.

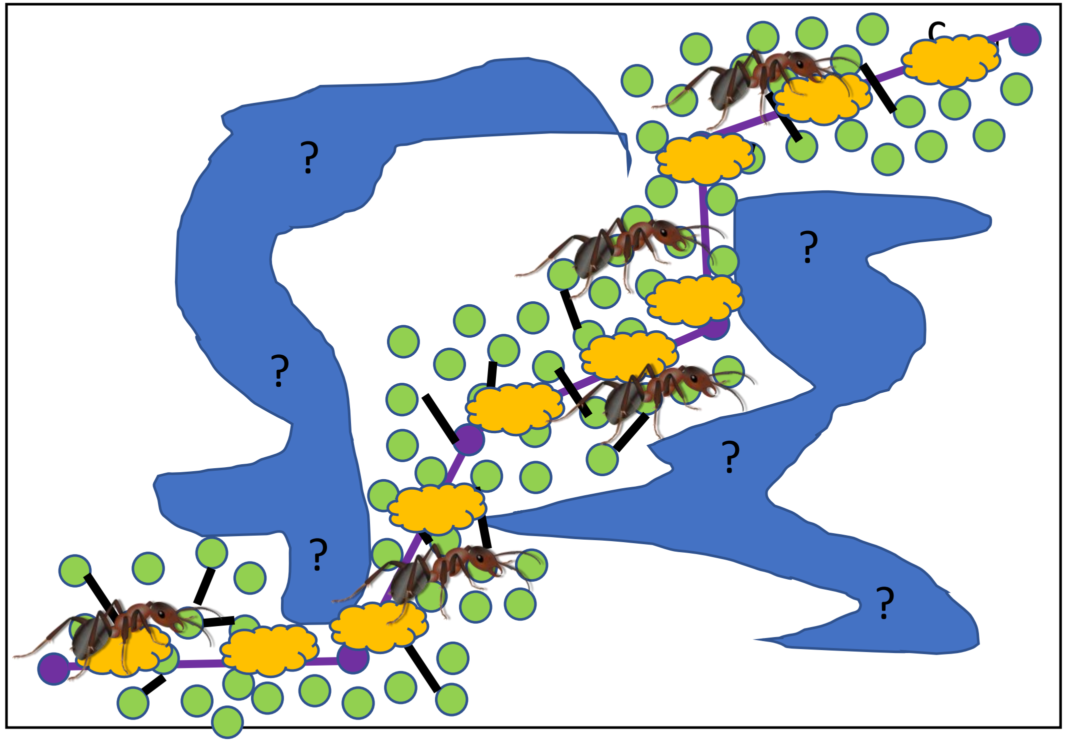
1.a Using ant to optimize the result path



Approach:

- Compute the path with a Tree-based planner.
- Sample the surrounding of the path.
- Connect to a graph, using NN.
- Put some initial pheromones on the solution path.

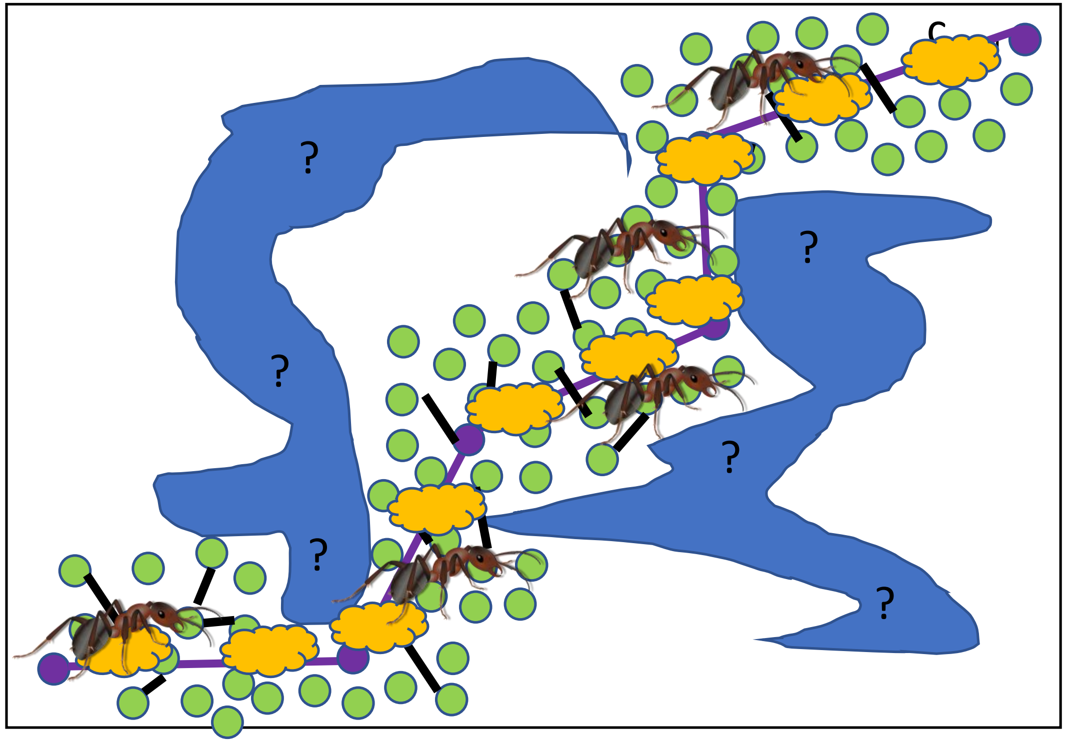
1.a Using ant to optimize the result path



Approach:

- Compute the path with a Tree-based planner.
- Sample the surrounding of the path.
- Connect to a graph, using NN.
- Put some initial pheromones on the solution path.
- Let ants optimize the path.

1.a Using ant to optimize the result path



→ Alternative to using Djkstra, but
→ Can only be used for scenarios with huge amount of samples.
→ Or if multiple goals or roundtrip is needed (TSP)
→ Otherwise go with Djkstra.

Approach:

- Compute the path with a Tree-based planner.
- Sample the surrounding of the path.
- Connect to a graph, using NN.
- Put some initial pheromones on the solution path.
- Let ants optimize the path.

Recap: What is the TSP?

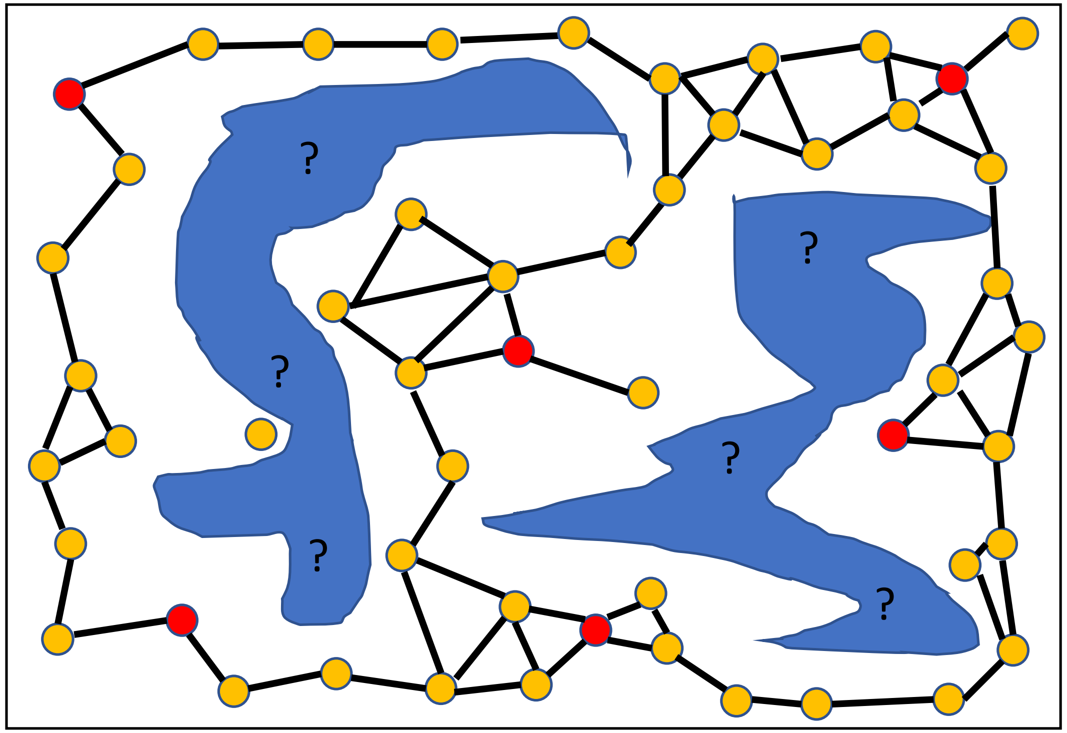
- Given x amount of cities.
- A salesman shall visit each city.
- Each city should only be visited once.
- Start city=end city (circle trip)

→Find the sequence of cities that result in the minimum travel distance for the salesman.

Note: This is NP-hard.

Image Source:https://www-m9.ma.tum.de/games/tsp-game/index_de.html

1.b Using ant to optimize the result path



Approach:

- Compute the path with a roadmap-based planner.
- Same as for tree-based. If only start goal given → go with Djkstra.

Conclusion

Approach:

- The ACO algorithm can be used to optimize motion planning problems that require multiple goals.
- In practice this is mainly needed for robots. There the TSP result is useful.
- Robots has to pick up things at x locations. Sequence irrelevant or only partial relevant.

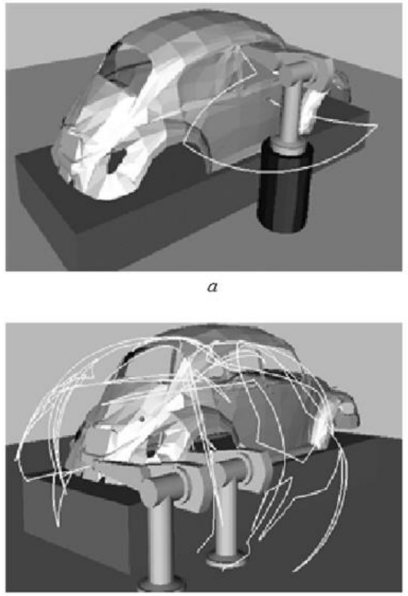

a
b

Fig. 1. *a*: A PUMA 560 on a mobile platform. *b*: Two PUMA 560s on the ground.

EUROCON 2005 Serbia & Montenegro, Belgrade, November 22-24, 2005

Ant Colony Robot Motion Planning

Mohd M. Mohamad, Matthew W. Dunnigan, and Nicholas K. Taylor

Abstract — **A new approach to robot motion planning is proposed by applying Ant Colony Optimization with the probabilistic roadmap planner (PRM). The PRM is a path planning method that consists of capturing the connectivity of the robot's free space in a network called the roadmap. An ant colony robot motion planning (ACRMP) method is proposed that takes the benefit of collective behaviour of ants foraging from a nest to a food source. Two groups of ants are placed at both the nest and food source respectively. A number of ants (agents) are released from the nest (start configuration) and begin to forage (search) towards the food (goal configuration). Each ant has a certain quantity of pheromone to be dropped along the path. The ants track down the pheromone trails previously dropped by the nest's ants to accomplish the path between the two points of nest and food respectively. Results from preliminary tests show that the ACRMP is capable of reducing the intermediate configuration between the initial and goal configuration in an acceptable running time.**

Keywords —**Planning, robotics and search**

I. INTRODUCTION

THIS paper describes the novel application of swarm intelligence to robotic arm manipulator motion planning. The probabilistic roadmap (PRM) is among the most efficient methods for planning robot motion. A PRM is a discrete representation of a continuous configuration space (C-space) generated by randomly sampling the free configurations of the C-space and connecting the points into a graph. Ant Colony Optimisation (ACO) is a swarm intelligence approach to solving optimisation problems. ACO has been successfully used to optimise the travelling salesman problem (TSP) and the quadratic assignment problem (QAP) [2]. To date, the applications of swarm intelligence in robotics have been primarily with collectives of robots or with sets of transfer functions in factory production lines to solve one major task. This paper introduces a new approach by applying ACO to robot motion planning, specifically manipulator arm type robots. It focuses on applying ant colony behaviour to search the robot's C-space. A preliminary experiment has been undertaken based on SBL-PRM multi-goal motion planning as a benchmark for this new algorithm. Our aim is to get a faster planner and reduce the number of intermediate configurations between the two query configurations.

M. M. Mohamad is a PhD. student of the Department of Electrical, Electronics, and Computer Engineering of School of Engineering and Physical Sciences, Heriot-Watt University (phone: 44-131-4513265; fax: 44-131-4514155; e-mail: M.Mohamad@hw.ac.uk).
M. W. Dunnigan, is a Senior Lecturer in the Department of Electrical, Electronics, and Computer Engineering in the School of Engineering and Physical Sciences, Heriot-Watt University (phone: 44-131-4513346; fax: 44-131-4514155; e-mail: M.W.Dunnigan@hw.ac.uk).
N. K. Taylor is a Senior Lecturer in the Department of Computer Science in the School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh (phone: 44-131-4513436; fax: 44-131-4513327; e-mail: N.K.Taylor@hw.ac.uk)

II. RELATION TO PREVIOUS WORK

There are two different fields that are involved in this research; PRM and ACO. A few previous publications in these fields will be described in this section.

A. Works on Probabilistic Roadmap

The problem of robot motion planning in known workspaces has been studied extensively over the last two decades [7]. PRM's have been proven to be an effective tool in solving motion-planning problems with many degrees of freedom [5], [10]. PRM constructs a roadmap of paths in configuration space [7]. A roadmap is a pre-computed undirectional graph covering the entire free space. Once the roadmap is constructed, it is used as a set of standardized paths. Path planning is then applied by connecting the initial and goal configurations to points in the roadmap and then searching the roadmap for the path between these points. The randomized techniques of PRM are presented in [6].

The work in [1] introduced PRM with lazy collision-checking (lazy-PRM). This planner assumes all nodes and paths are collision-free during the pre-computation phase. It then searches the roadmap for the shortest path. The nodes and path are checked for collision afterwards. If collisions have occurred, the corresponding nodes and edges are removed. The planner either finds the new shortest path, or first updates the roadmap with the new nodes and edges, then searches for the shortest path.

The approach in [10], [11] introduced single-query bi-directional path planning with lazy collision-checking (SBL-PRM). This single-query approach, instead of pre-computing a roadmap covering the entire free space, uses the two input query configurations to explore as little space as possible. The bi-directional approach explores the robot's free space by building a roadmap made of two trees rooted at the query configurations. Lazy collision-checking means that the collision tests are delayed along the edges of the roadmap until they are absolutely needed. The SBL-PRM is applied to multi-robot path planning in [11].

A continuation of SBL-PRM is performed in [9] for a problem of multi-goal motion planning which combines two problems: the shortest path and travelling-salesman problem. The planner uses a greedy algorithm that

1-4244-0049-X/05/$20.00 ©2005 IEEE 213

Mohd M. Mohamad, Matthew W. Dunnigan, and Nicholas K. Taylor (2014).
Ant Colony Robot Motion Planning
https://www.researchgate.net/publication/4241223_Ant_Colony_Robot_Motion_Planning