Our first motion planner- The sPRM

Algorithms and Data Structures 2 – Motion Planning and its applications

University of Applied Sciences Stuttgart

Dr. Daniel Schneider

Some history about PRM

Kavraki, Lydia E., Petr Svestka, Jean-Claude Latombe, and Mark H. Overmars (1996).

"Probabilistic roadmaps for path planning in high-dimensional configuration spaces".

https://www.cs.cmu.edu/~./motionplanning/papers/sbp_papers/PRM/prmbasic_01.pdf

Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces

Lydia E. Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H. Overmars

Abstract— **A new motion planning method for robots in static workspaces is presented. This method proceeds in two phases: a learning phase and a query phase. In the learning phase, a probabilistic roadmap is constructed and stored as a graph whose nodes correspond to collision-free configurations and whose edges correspond to feasible paths between these configurations. These paths are computed using a simple and fast local planner. In the query phase, any given start and goal configurations of the robot are connected to two nodes of the roadmap; the roadmap is then searched for a path joining these two nodes. The method is general and easy to implement. It can be applied to virtually any type of holonomic robot. It requires selecting certain parameters (e.g., the duration of the learning phase) whose values depend on the scene, that is the robot and its workspace. But these values turn out to be relatively easy to choose. Increased efficiency can also be achieved by tailoring some components of the method (e.g., the local planner) to the considered robots. In this paper the method is applied to planar articulated robots with many degrees of freedom. Experimental results show that path planning can be done in a fraction of a second on a contemporary workstation (≈ 150 MIPS), after learning for relatively short periods of time (a few dozen seconds).**

I. Introduction

We present a new planning method which computes collision-free paths for robots of virtually any type moving among stationary obstacles (static workspaces). However, our method is particularly interesting for robots with many degrees of freedom (dof), say five or more. Indeed, an increasing number of practical problems involve such robots, while very few effective motion planning methods, if any, are available to solve them. The method proceeds in two phases: a *learning phase* and a *query phase*.

In the learning phase a probabilistic *roadmap* is constructed by repeatedly generating random free configurations of the robot and connecting these configurations using some simple, but very fast motion planner. We call this planner the *local planner*. The roadmap thus formed in the free configuration space (C-space [37]) of the robot is stored as an undirected graph R. The configurations are the nodes of R and the paths computed by the local planner are the edges of R. The learning phase is concluded by some postprocessing of R to improve its connectivity.

Following the learning phase, multiple queries can be answered. A *query* asks for a path between two free configurations of the robot. To process a query the method first attempts to find a path from the start and goal configurations to two nodes of the roadmap. Next, a graph search is done to find a sequence of edges connecting these nodes in the roadmap. Concatenation of the successive path segments transforms the sequence found into a feasible path for the robot.

Notice that the learning and the query phases do not have to be executed sequentially. Instead, they can be interwoven to adapt the size of the roadmap to difficulties encountered during the query phase, thus increasing the learning flavor of our method. For instance, a small roadmap could be first constructed; this roadmap could then be augmented (or reduced) using intermediate data generated while queries are being processed. This interesting possibility will not be explored in the paper, though it is particularly useful to conduct trial-and-error experiments in order to decide how much computation time should be spent in the learning phase.

To run our planning method the values of several parameters must first be selected, e.g., the time to be spent in the learning phase. While these values depend on the *scene*, i.e., the robot and the workspace, it has been our experience that good results are obtained with values spanning rather large intervals. Thus, it is not difficult to choose one set of satisfactory values for a given scene or family of scenes, through some preliminary experiments. Moreover, increased efficiency can be achieved by tailoring several components of our method, in particular the local planner, to the considered robots. Overall, we found the method quite easy to implement and run. Many details can be engineered in one way or another to fit better the characteristics of an application domain.

We have demonstrated the power of our method by applying it to a number of difficult motion planning problems involving a variety of robots. In this paper we report in detail on experiments with planar articulated robots (or linkages) with many dofs moving in constrained workspaces. However, the method is directly applicable to other kinds of holonomic robots, such as spatial articulated robots in 3-D workspaces [29]. Additionally, a version of the method described here has been successfully applied to nonholonomic car-like robots [48]. In all cases, experimental results show that the learning

Manuscript received August 18, 1994; revised May 1, 1995. This work was supported in part ARPA grant N00014-92-J-1809, ONR grant N00014-94-1-0721, the Rockwell Foundation, ESPRIT III BRA Project 6546 (PROMotion), and by the Dutch Organization for Scientific Research (NWO). This paper was recommended by publication by Associate Editor M. Erdmann upon evaluation of reviewers' comments.

L. E. Kavraki and J.-C. Latombe are with the Robotics Laboratory, Department of Computer Science, Stanford University, Stanford, CA 94305 USA (e-mails: {kavraki, latombe}@cs.stanford.edu).

P. Švestka and M. H. Overmars are with the Department of Computer Science, Utrecht University, 3508 TB Utrecht, The Netherlands (e-mails: {petr, markov}@cs.ruu.nl).

Publisher Item Identifier S 1042-296X(96)03830-X.

1042-296X/96$05.00 © 1996 IEEE

Some history about PRM

Manocha, Dinesh, Christian Lauterbach, and Jia Pan (2010).

"G-Planner: Real-time motion planning and global navigation using GPUs"

https://www.researchgate.net/publication/221607145_g-Planner_Real-time_Motion_Planning_and_Global_Navigation_using_GPUs

g-Planner: Real-time Motion Planning and Global Navigation using GPUs

Jia Pan and **Christian Lauterbach** and **Dinesh Manocha**
Department of Computer Science, University of North Carolina at Chapel Hill
{panj, cl, dm}@cs.unc.edu
http://gamma.cs.unc.edu/gplanner/

Abstract

We present novel randomized algorithms for solving global motion planning problems that exploit the computational capabilities of many-core GPUs. Our approach uses thread and data parallelism to achieve high performance for all components of sample-based algorithms, including random sampling, nearest neighbor computation, local planning, collision queries and graph search. This approach can efficiently solve both the multi-query and single-query versions of the problem and obtain considerable speedups over prior CPU-based algorithms. We demonstrate the efficiency of our algorithms by applying them to a number of 6DOF planning benchmarks in 3D environments. Overall, this is the first algorithm that can perform real-time motion planning and global navigation using commodity hardware.

Introduction

Motion planning is one of the fundamental problems in algorithmic robotics. The classical formulation of the problem is: given an arbitrary robot, R, and an environment composed of obstacles, compute a continuous collision-free path for R from an initial configuration to the final configuration. It is also known as the *navigation problem* or *piano mover's problem*. Besides robotics, motion planning algorithms are also used in CAD/CAM, computer animation, computer gaming, computational drug-design, manufacturing, medical simulations etc.

There is extensive literature on motion planning and global navigation. At a broad level, they can be classified into local and global approaches. The local approaches, such as those based on artificial potential field methods (Khatib 1986), are quite fast but not guaranteed to find a path. On the other hand, global methods based on criticality analysis or roadmap computation (Schwartz and Sharir 1983; Canny 1988) are guaranteed to find a path. However, the complexity of these exact or complete algorithms increases as an exponential function of the number of degrees-of-freedom (DOF) of the robot and their implementations have been restricted to only low DOF.

Practical methods for global motion planning for high-DOF robots are based on randomized sampling (Kavraki et al. 1996; LaValle and Kuffner 2000). These methods attempt to capture the topology of the free space of the robot by generating random configurations and connect nearby configurations using local planning methods. The resulting algorithms are probabilistically complete and have been successfully used to solve many high-DOF motion planning and navigation problems in different applications. However, they are too slow for interactive applications or dynamic environments.

Main Results: We present a novel parallel algorithm for real-time motion planning of high DOF robots that exploits the computational capability of a \$400 commodity graphics processing unit (GPU). Current GPUs are programmable many-core processors that can support thousands of concurrent threads and we use them for real-time computation of a probabilistic roadmap (PRM) and a lazy planner. We describe efficient parallel strategies for the construction phase that include sample generation, collision detection, connecting nearby samples and local planning. The query phase is also performed in parallel based on graph search. In order to design an efficient single query planner, we use a lazy strategy that defers collision checking and local planning. In order to accelerate the overall performance, we also describe new hierarchy-based collision detection algorithms.

The performance of the algorithm is governed by the topology of the underlying free space as well as the methods used for sample generation and nearest neighbor computation. In practice, our algorithm can generate thousands of samples for robots with 3 or 6 DOFs and compute the roadmap for these samples at close to interactive rates including construction of all hierarchies. It performs no precomputation and is applicable to dynamic scenes, articulated models or non-rigid robots. We highlight its performance on multiple benchmarks on a commodity PC with a NVIDIA GTX 285 GPU and observe a 10-80 times performance improvement over CPU-based implementations.

The rest of the paper is organized as follows. We survey related work on motion planning and GPU-based algorithms in Section 2. Section 3 gives an overview of our approach and we present parallel algorithms for the construction and query phase in Section 4. We highlight our performance on different motion planning benchmarks in Section 5 and compare with prior methods.

Presentation of the sPRM

Algorithm 3: sPRM($\{(c^i_{init}, c^i_{goal})\}, r, n$)

$E \leftarrow \emptyset,\ V \leftarrow \emptyset$

foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**

  $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$

for $j \leftarrow 0$ **to** n **do**

  $V \leftarrow V \cup CFreeSample()$

foreach $v \in V$ **do**

  $U \leftarrow Neighbors(v, V, r)$

  foreach $u \in U$ **do**

    if $(edgeIsValid(u, v))$ **then**

      $E \leftarrow E \cup (u, v)$

foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**

  if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**

    $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$

  else

    $\sigma_i \leftarrow \emptyset$

return $\{\sigma_i\}$

- This is simplified version of the classical PRM algorithm.
- We will see that this version is very easy to implement.
- It is also easy to parallelize.
- Therefore it has gained much popularity.

Presentation of the sPRM

Algorithm 3: sPRM($\{(c^i_{init}, c^i_{goal})\}, r, n$)

$E \leftarrow \emptyset,\ V \leftarrow \emptyset$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
    $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$
for $j \leftarrow 0$ **to** n **do**
    $V \leftarrow V \cup CFreeSample()$
foreach $v \in V$ **do**
    $U \leftarrow Neighbors(v, V, r)$
    foreach $u \in U$ **do**
        if $(edgeIsValid(u, v))$ **then**
            $E \leftarrow E \cup (u, v)$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
    if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**
        $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$
    else
        $\sigma_i \leftarrow \emptyset$
return $\{\sigma_i\}$

- It also can be used to plan multiple queries but also for single query.

Presentation of the sPRM

Algorithm 3: sPRM($\{(c^i_{init}, c^i_{goal})\}, r, n$)

$E \leftarrow \emptyset$, $V \leftarrow \emptyset$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
    $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$
for $j \leftarrow 0$ **to** n **do**
    $V \leftarrow V \cup CFreeSample()$
foreach $v \in V$ **do**
    $U \leftarrow Neighbors(v, V, r)$
    foreach $u \in U$ **do**
        if $(edgeIsValid(u, v))$ **then**
            $E \leftarrow E \cup (u, v)$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
    if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**
        $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$
    else
        $\sigma_i \leftarrow \emptyset$
return $\{\sigma_i\}$

- Beside the start and goal points it has 2 major parameters.
- r is the search radius in the algorithm. More later.
- n is the amount of samples that are created.

Presentation of the sPRM

Algorithm 3: sPRM($\{(c^i_{init}, c^i_{goal})\}, r, n$)

$E \leftarrow \emptyset$, $V \leftarrow \emptyset$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$
for $j \leftarrow 0$ **to** n **do**
  $V \leftarrow V \cup CFreeSample()$
foreach $v \in V$ **do**
  $U \leftarrow Neighbors(v, V, r)$
  foreach $u \in U$ **do**
    if $(edgeIsValid(u, v))$ **then**
      $E \leftarrow E \cup (u, v)$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**
    $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$
  else
    $\sigma_i \leftarrow \emptyset$
return $\{\sigma_i\}$

- This E is the edge data structure.
- At the beginning this data structure empty.
- In this data structure all edges between two configurations are stored.
- E is later used for a Djikstra algorithm.
- Therefore the edge data structure E needs to be a linked list.
- Note:
    - In the practical study work you do not implement the Djikstra. Use a library.
    - The Djikstra is only used at the end of the algorithm.

Presentation of the sPRM

Algorithm 3: sPRM$(\{(c^i_{init}, c^i_{goal})\}, r, n)$

$E \leftarrow \emptyset$, $V \leftarrow \emptyset$

foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**

  $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$

for $j \leftarrow 0$ **to** n **do**

  $V \leftarrow V \cup CFreeSample()$

foreach $v \in V$ **do**

  $U \leftarrow Neighbors(v, V, r)$

  foreach $u \in U$ **do**

    if $(edgeIsValid(u, v))$ **then**

      $E \leftarrow E \cup (u, v)$

foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**

  if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**

    $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$

  else

    $\sigma_i \leftarrow \emptyset$

return $\{\sigma_i\}$

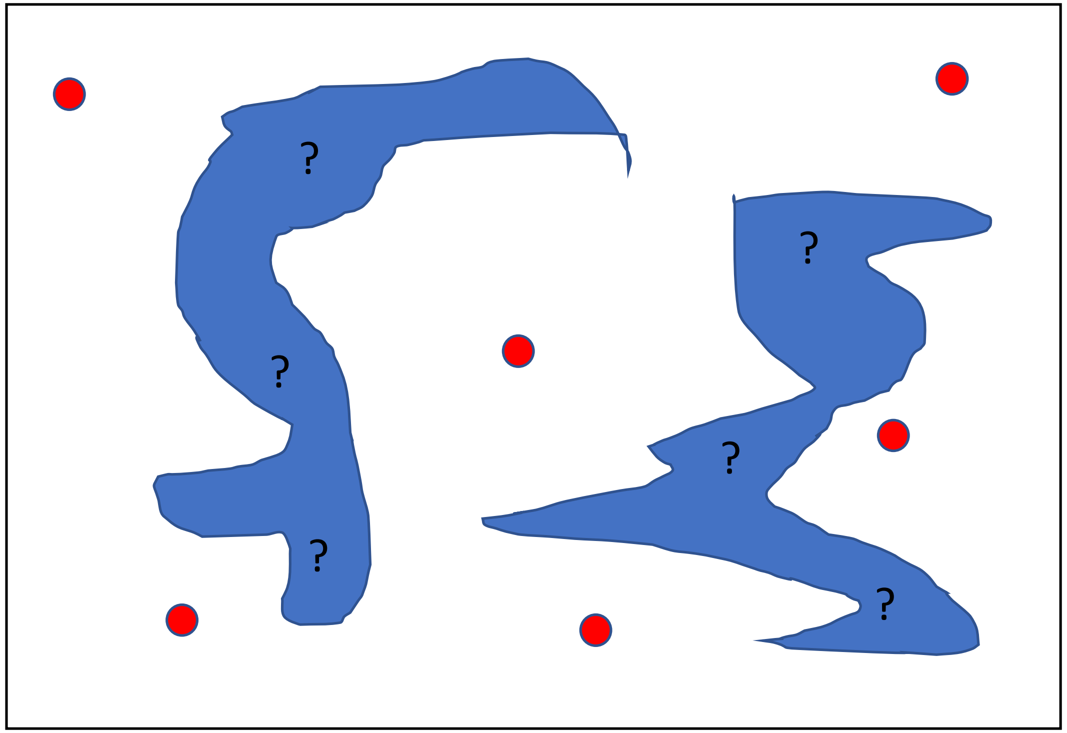
- This V is the vertex data structure.
- This data structure is also empty at the beginning.
- In this data structure all configurations are stored.
- This data structure must be iterable.
- This data structure can be implemented using a simple array.

Presentation of the sPRM

Algorithm 3: sPRM($\{(c^i_{init}, c^i_{goal})\}, r, n$)

$E \leftarrow \emptyset, V \leftarrow \emptyset$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$
for $j \leftarrow 0$ **to** n **do**
  $V \leftarrow V \cup CFreeSample()$
foreach $v \in V$ **do**
  $U \leftarrow Neighbors(v, V, r)$
  foreach $u \in U$ **do**
    if $(edgeIsValid(u, v))$ **then**
      $E \leftarrow E \cup (u, v)$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**
    $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$
  else
    $\sigma_i \leftarrow \emptyset$
return $\{\sigma_i\}$

- In this part of the algorithm all start and goal configurations are added to the vertex data structure.

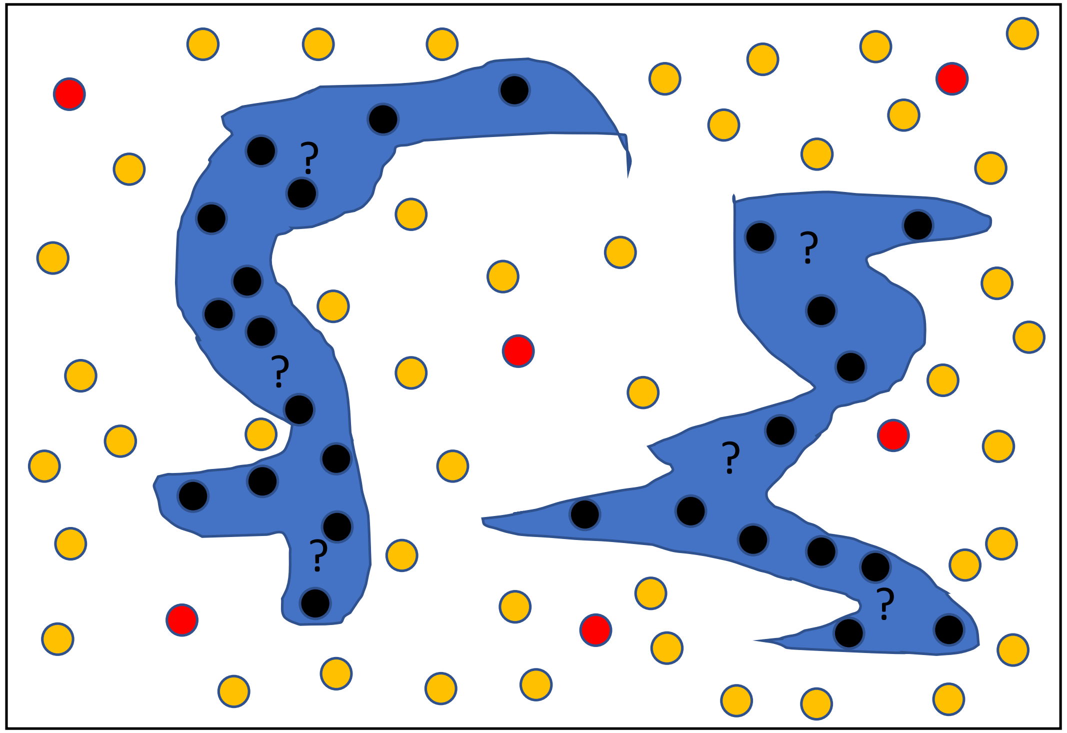


Presentation of the sPRM

Algorithm 3: sPRM$(\{(c^i_{init}, c^i_{goal})\}, r, n)$

$E \leftarrow \emptyset,\ V \leftarrow \emptyset$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$
for $j \leftarrow 0$ **to** n **do**
  $V \leftarrow V \cup CFreeSample()$
foreach $v \in V$ **do**
  $U \leftarrow Neighbors(v, V, r)$
  foreach $u \in U$ **do**
    if $(edgeIsValid(u, v))$ **then**
      $E \leftarrow E \cup (u, v)$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**
    $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$
  else
    $\sigma_i \leftarrow \emptyset$
return $\{\sigma_i\}$

- In this part of the algorithm n samples are computed.
- Samples that are in $\mathcal{C}_{obs}$ are ignored.
- The ignored samples characterize $\mathcal{C}_{obs}$ but in the simple sPRM these samples are not used.



Presentation of the sPRM

Algorithm 3: sPRM$(\{(c^i_{init}, c^i_{goal})\}, r, n)$

$E \leftarrow \emptyset,\ V \leftarrow \emptyset$

foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**

  $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$

for $j \leftarrow 0$ **to** n **do**

  $V \leftarrow V \cup CFreeSample()$

foreach $v \in V$ **do**

  $U \leftarrow Neighbors(v, V, r)$

  foreach $u \in U$ **do**

    if $(edgeIsValid(u, v))$ **then**

      $E \leftarrow E \cup (u, v)$

foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**

  if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**

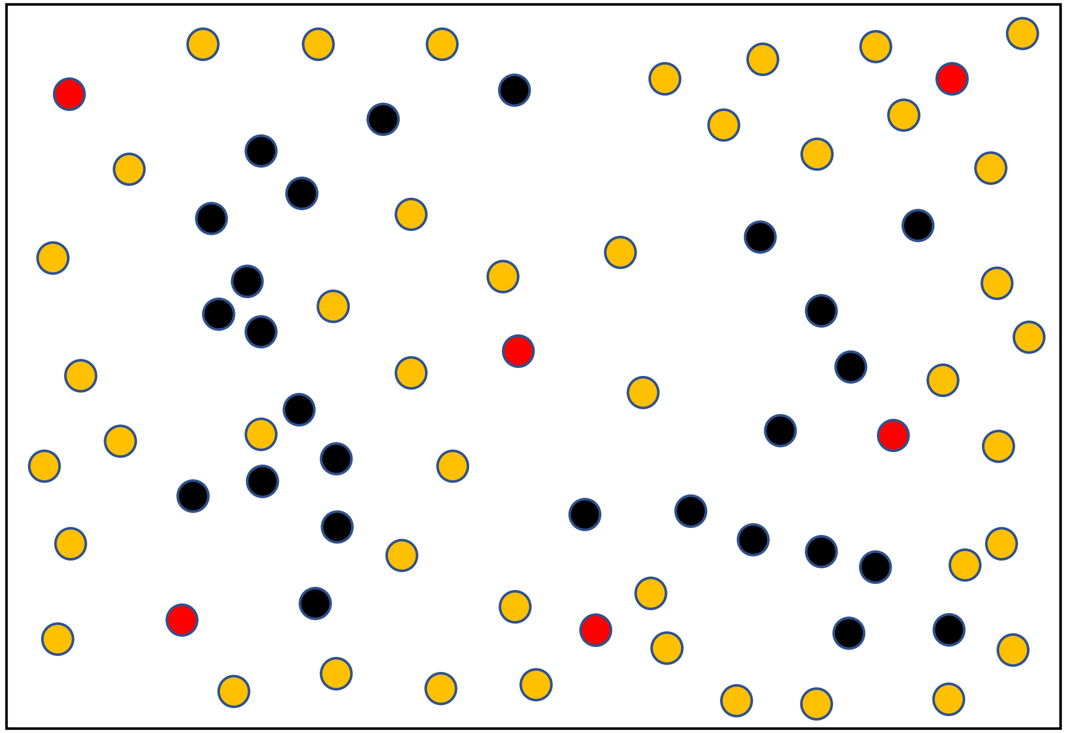
    $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$

  else

    $\sigma_i \leftarrow \emptyset$

return $\{\sigma_i\}$

- We do not know $\mathcal{C}_{obs}$ but we keep it in the picture for visualization.



Presentation of the sPRM

Algorithm 3: sPRM$(\{(c^i_{init}, c^i_{goal})\}, r, n)$

$E \leftarrow \emptyset,\ V \leftarrow \emptyset$

foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**

  $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$

for $j \leftarrow 0$ **to** n **do**

  $V \leftarrow V \cup CFreeSample()$

foreach $v \in V$ **do**

  $U \leftarrow Neighbors(v, V, r)$

  foreach $u \in U$ **do**

    if $(edgeIsValid(u, v))$ **then**

      $E \leftarrow E \cup (u, v)$

foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**

  if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**

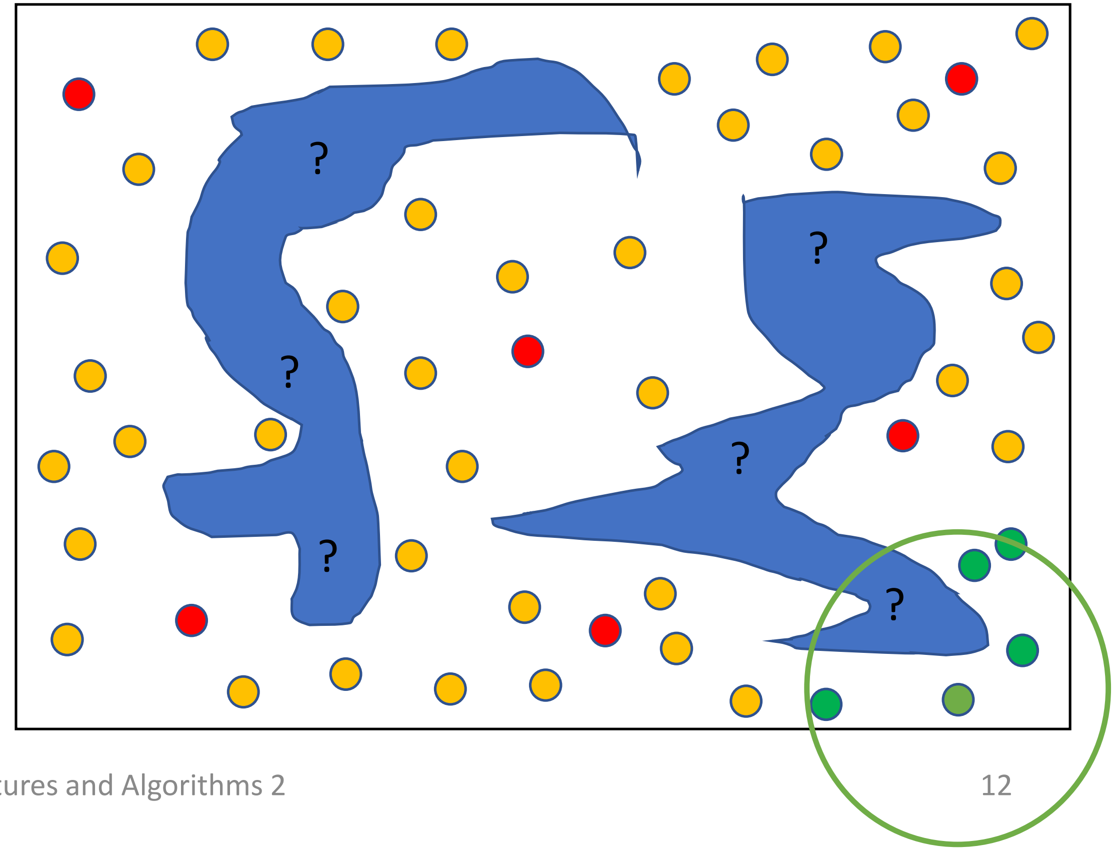
    $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$

  else

    $\sigma_i \leftarrow \emptyset$

return $\{\sigma_i\}$

- We iterate over all samples.
- We search all neigbours of the sample in a radius r.

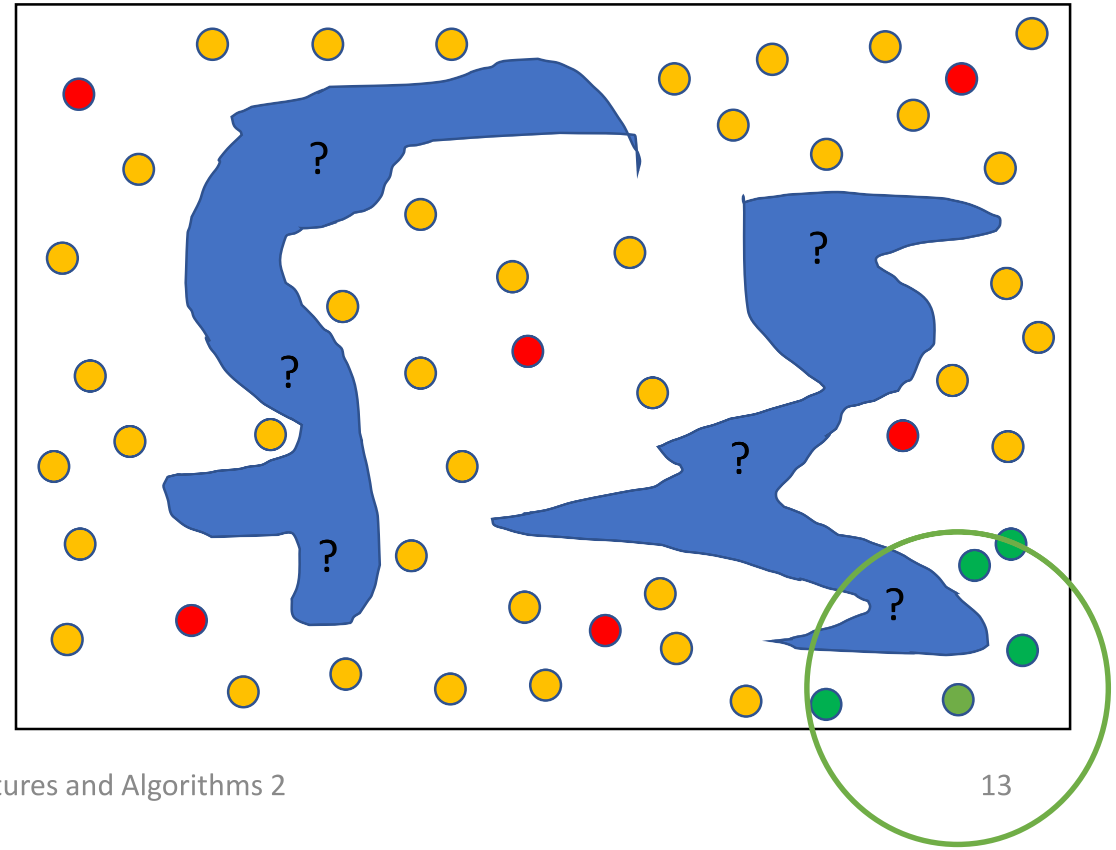


Presentation of the sPRM

Algorithm 3: sPRM$(\{(c^i_{init}, c^i_{goal})\}, r, n)$

$E \leftarrow \emptyset$, $V \leftarrow \emptyset$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
    $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$
for $j \leftarrow 0$ **to** n **do**
    $V \leftarrow V \cup CFreeSample()$
foreach $v \in V$ **do**
    $U \leftarrow Neighbors(v, V, r)$
    foreach $u \in U$ **do**
        if $(edgeIsValid(u, v))$ **then**
            $E \leftarrow E \cup (u, v)$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
    if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**
        $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$
    else
        $\sigma_i \leftarrow \emptyset$
return $\{\sigma_i\}$

- All these neighbours are stored in a simple array U.



Presentation of the sPRM

Algorithm 3: sPRM($\{(c^i_{init}, c^i_{goal})\}, r, n$)

$E \leftarrow \emptyset$, $V \leftarrow \emptyset$

foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**

  $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$

for $j \leftarrow 0$ **to** n **do**

  $V \leftarrow V \cup CFreeSample()$

foreach $v \in V$ **do**

  $U \leftarrow Neighbors(v, V, r)$

  foreach $u \in U$ **do**

    if $(edgeIsValid(u, v))$ **then**

      $E \leftarrow E \cup (u, v)$

foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**

  if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**

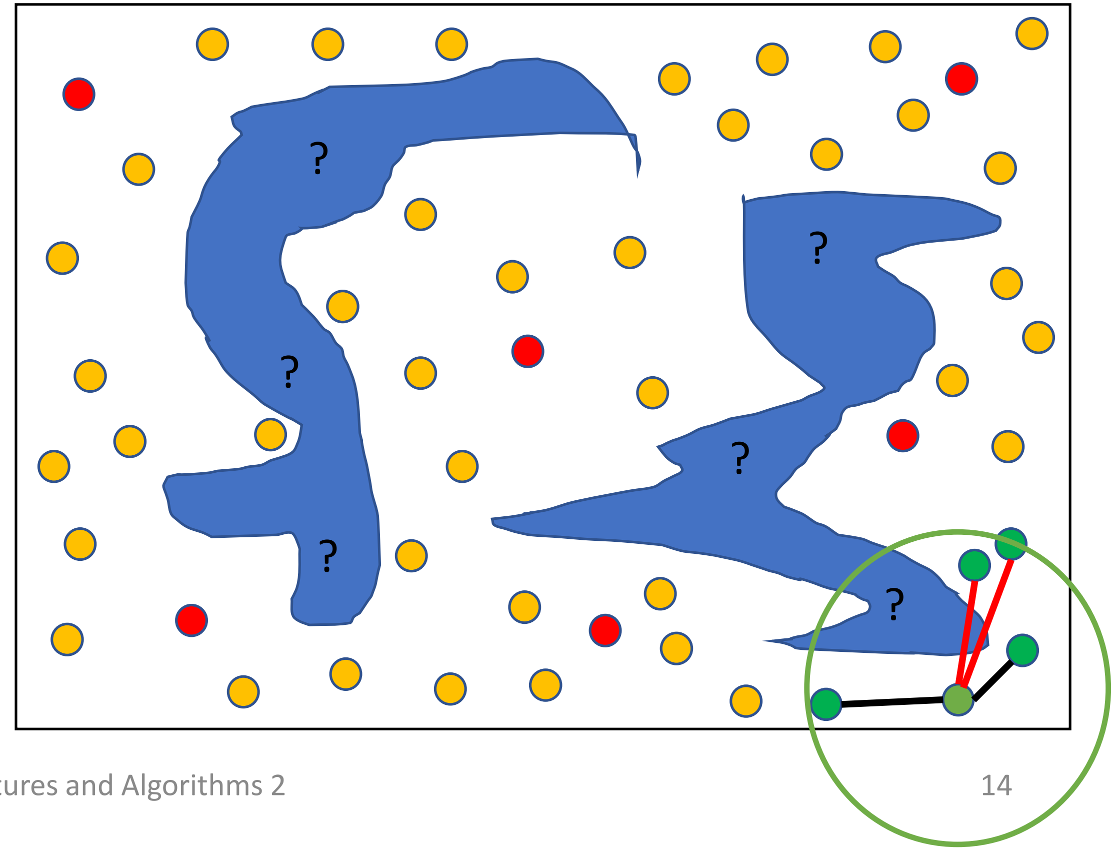
    $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$

  else

    $\sigma_i \leftarrow \emptyset$

return $\{\sigma_i\}$

- Afterwards we iterate over all configuration in U.
- Then we check whether the edge between the two configurations are valid.
- All valid edges are added to E.

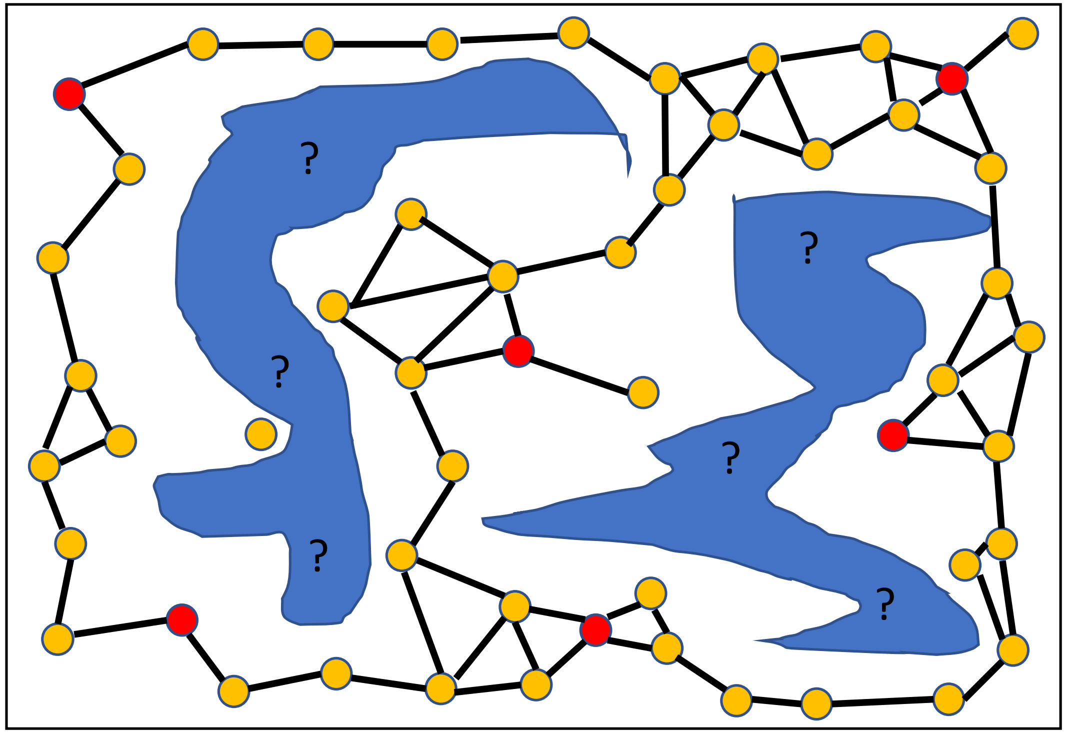


Presentation of the sPRM

Algorithm 3: sPRM$(\{(c^i_{init}, c^i_{goal})\}, r, n)$

$E \leftarrow \emptyset$, $V \leftarrow \emptyset$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
    $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$
for $j \leftarrow 0$ **to** n **do**
    $V \leftarrow V \cup CFreeSample()$
foreach $v \in V$ **do**
    $U \leftarrow Neighbors(v, V, r)$
    foreach $u \in U$ **do**
        if $(edgeIsValid(u, v))$ **then**
            $E \leftarrow E \cup (u, v)$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
    if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**
        $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$
    else
        $\sigma_i \leftarrow \emptyset$
return $\{\sigma_i\}$

- Now we do this for all configurations in V.
- We get a graph that characterizes $\mathcal{C}_{free}$

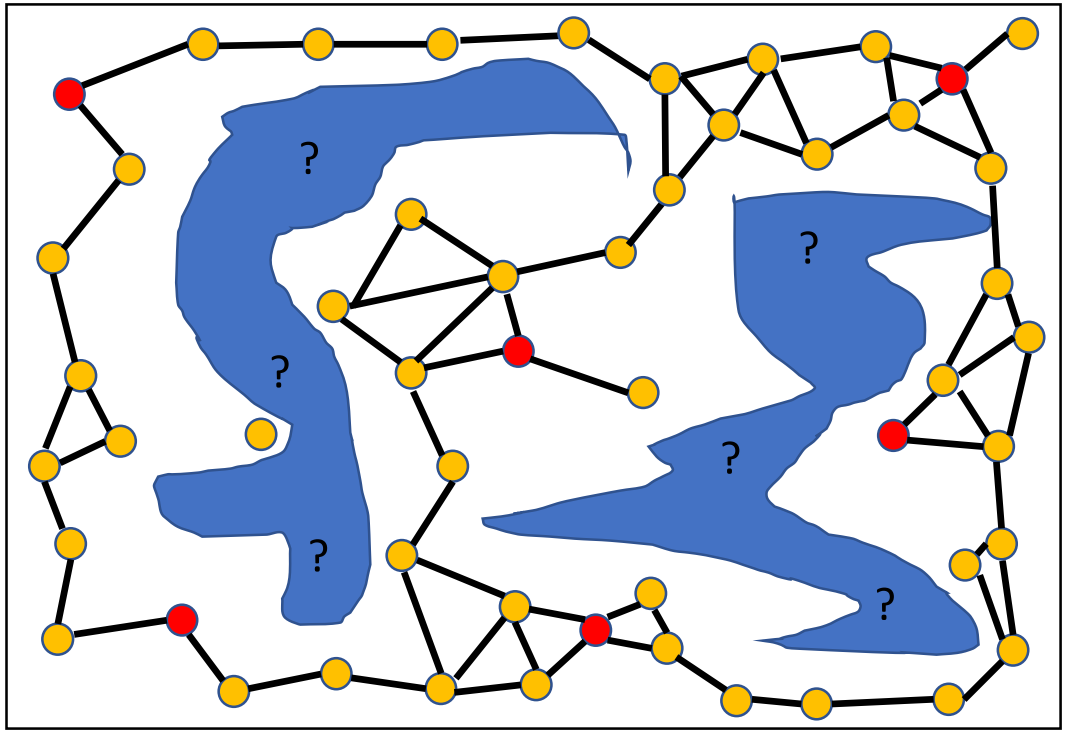


Presentation of the sPRM

Algorithm 3: sPRM$(\{(c^i_{init}, c^i_{goal})\}, r, n)$

$E \leftarrow \emptyset,\ V \leftarrow \emptyset$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$
for $j \leftarrow 0$ **to** n **do**
  $V \leftarrow V \cup CFreeSample()$
foreach $v \in V$ **do**
  $U \leftarrow Neighbors(v, V, r)$
  foreach $u \in U$ **do**
    if $(edgeIsValid(u, v))$ **then**
      $E \leftarrow E \cup (u, v)$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**
    $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$
  else
    $\sigma_i \leftarrow \emptyset$
return $\{\sigma_i\}$

- We iterate over all queries.
- If the start and goal points are part of our graph and we check if there exists a connection between start and goal.



Presentation of the sPRM

Algorithm 3: sPRM($\{(c^i_{init}, c^i_{goal})\}, r, n$)

$E \leftarrow \emptyset,\ V \leftarrow \emptyset$

foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**

  $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$

for $j \leftarrow 0$ **to** n **do**

  $V \leftarrow V \cup CFreeSample()$

foreach $v \in V$ **do**

  $U \leftarrow Neighbors(v, V, r)$

  foreach $u \in U$ **do**

    if $(edgeIsValid(u, v))$ **then**

      $E \leftarrow E \cup (u, v)$

foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**

  if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**

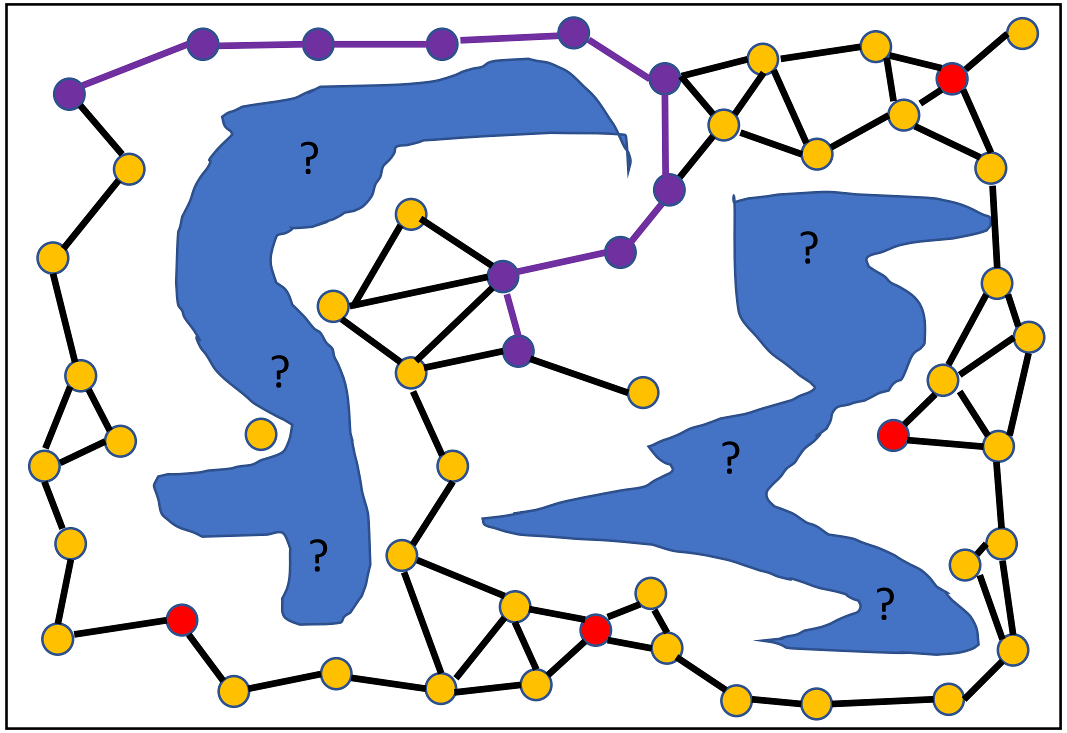
    $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$

  else

    $\sigma_i \leftarrow \emptyset$

return $\{\sigma_i\}$

- Then we call the shortestPath algorithm and get our solution to the motion planning problem.



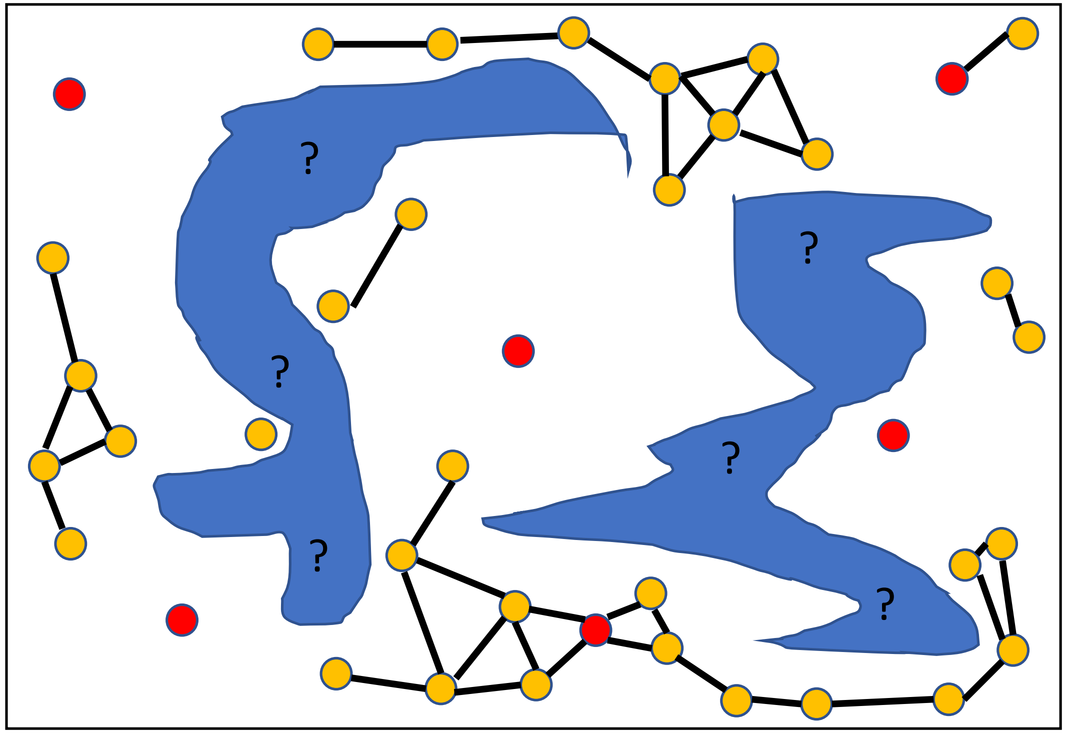
Presentation of the sPRM

Algorithm 3: sPRM($\{(c^i_{init}, c^i_{goal})\}, r, n$)

$E \leftarrow \emptyset$, $V \leftarrow \emptyset$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$
for $j \leftarrow 0$ **to** n **do**
  $V \leftarrow V \cup CFreeSample()$
foreach $v \in V$ **do**
  $U \leftarrow Neighbors(v, V, r)$
  foreach $u \in U$ **do**
    if $(edgeIsValid(u, v))$ **then**
      $E \leftarrow E \cup (u, v)$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**
    $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$
  else
    $\sigma_i \leftarrow \emptyset$
return $\{\sigma_i\}$

Notes:

- If there are **too little** samples n there might be the case that the solution can not be found.



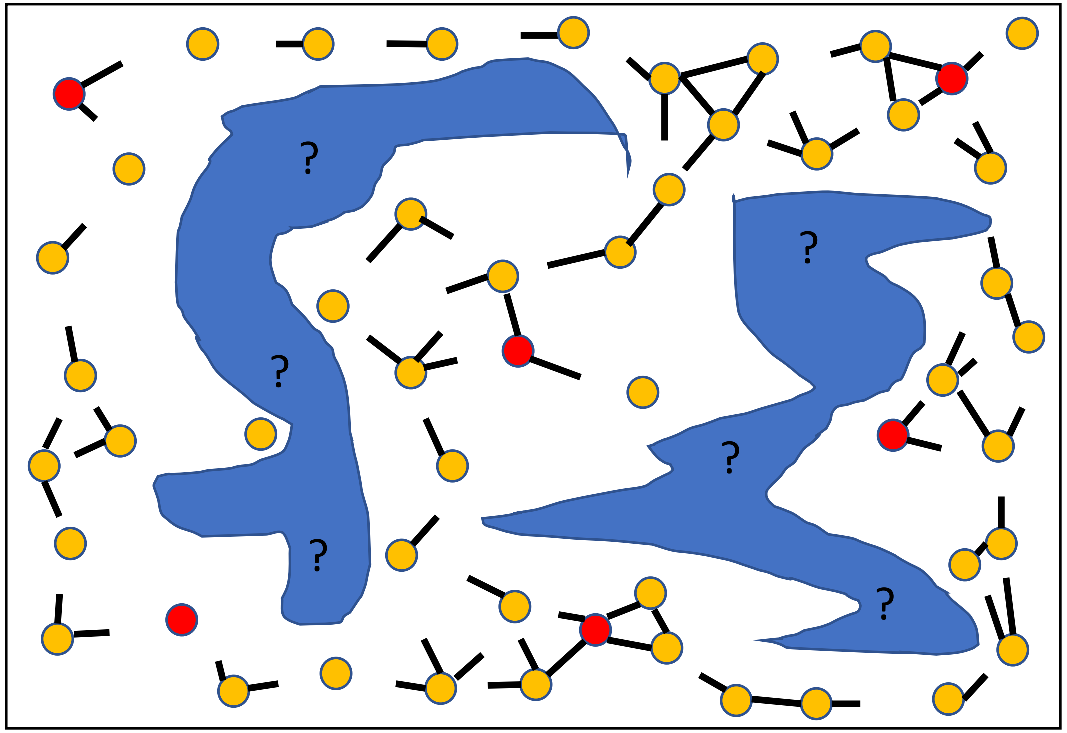
Presentation of the sPRM

Algorithm 3: sPRM($\{(c^i_{init}, c^i_{goal})\}, r, n$)

$E \leftarrow \emptyset,\ V \leftarrow \emptyset$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$
for $j \leftarrow 0$ **to** n **do**
  $V \leftarrow V \cup CFreeSample()$
foreach $v \in V$ **do**
  $U \leftarrow Neighbors(v, V, r)$
  foreach $u \in U$ **do**
    if $(edgeIsValid(u, v))$ **then**
      $E \leftarrow E \cup (u, v)$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**
    $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$
  else
    $\sigma_i \leftarrow \emptyset$
return $\{\sigma_i\}$

Notes:

- If the radius r is **too small** → more samples are needed or the a soluition is not found



Presentation of the sPRM

Algorithm 3: sPRM($\{(c^i_{init}, c^i_{goal})\}, r, n$)

$E \leftarrow \emptyset, V \leftarrow \emptyset$

foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**

  $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$

for $j \leftarrow 0$ **to** n **do**

  $V \leftarrow V \cup CFreeSample()$

foreach $v \in V$ **do**

  $U \leftarrow Neighbors(v, V, r)$

  foreach $u \in U$ **do**

    if $(edgeIsValid(u, v))$ **then**

      $E \leftarrow E \cup (u, v)$

foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**

  if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**

    $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$

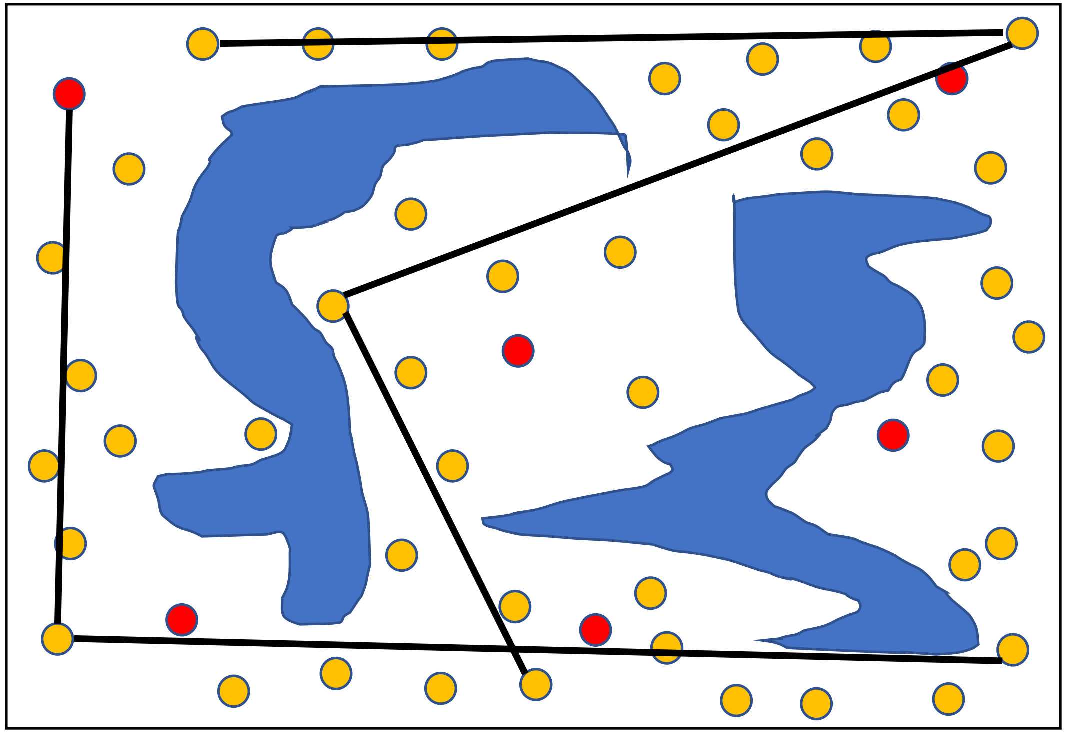
  else

    $\sigma_i \leftarrow \emptyset$

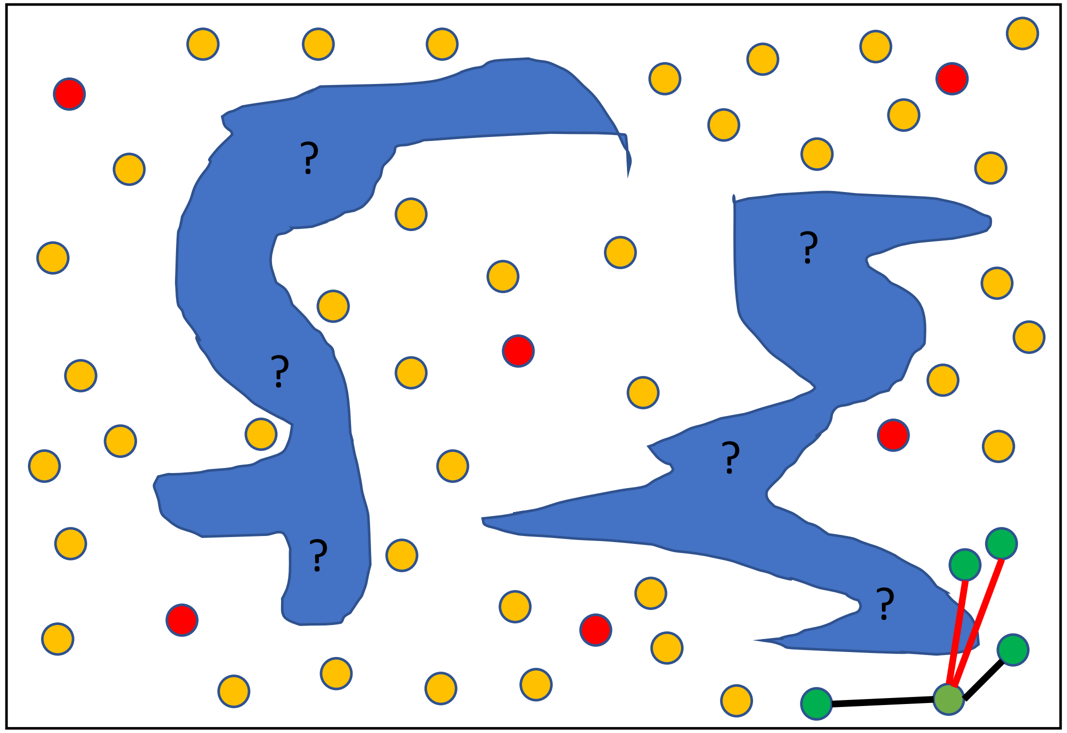
return $\{\sigma_i\}$

Notes:

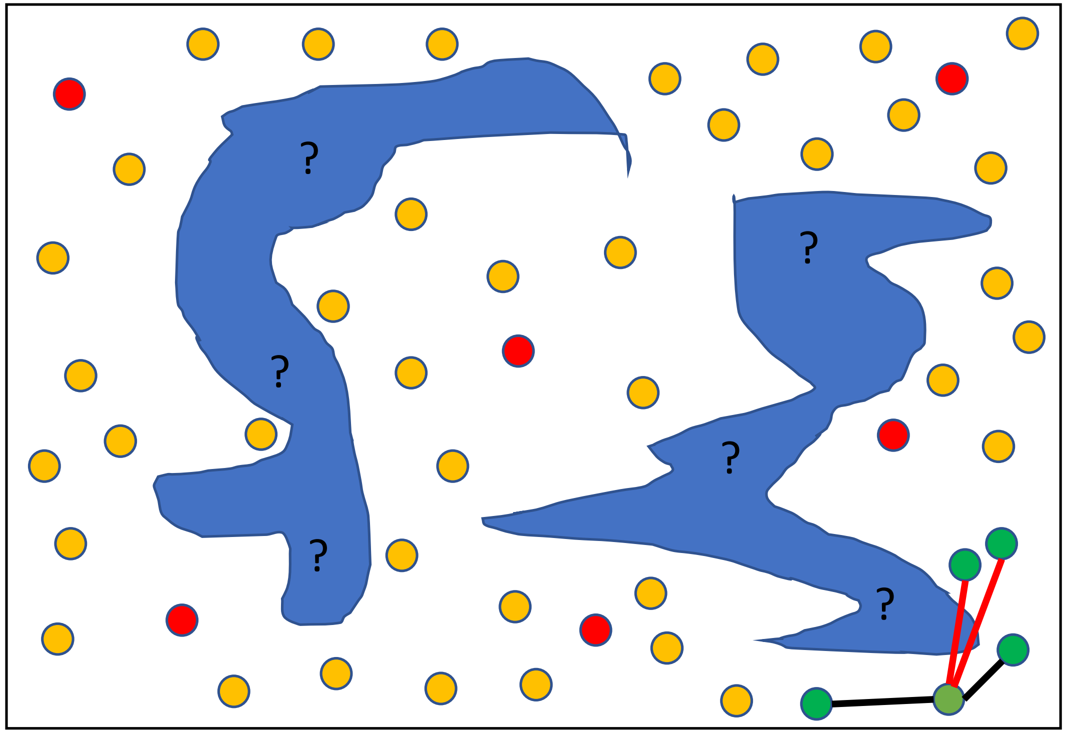
- If the radius r is **large** → $U \sim V$ → the edge connection is of quadratic complexity. → Edge connection takes the most time → bad performance.



How to check for a valid edge?



How to check for a valid edge?

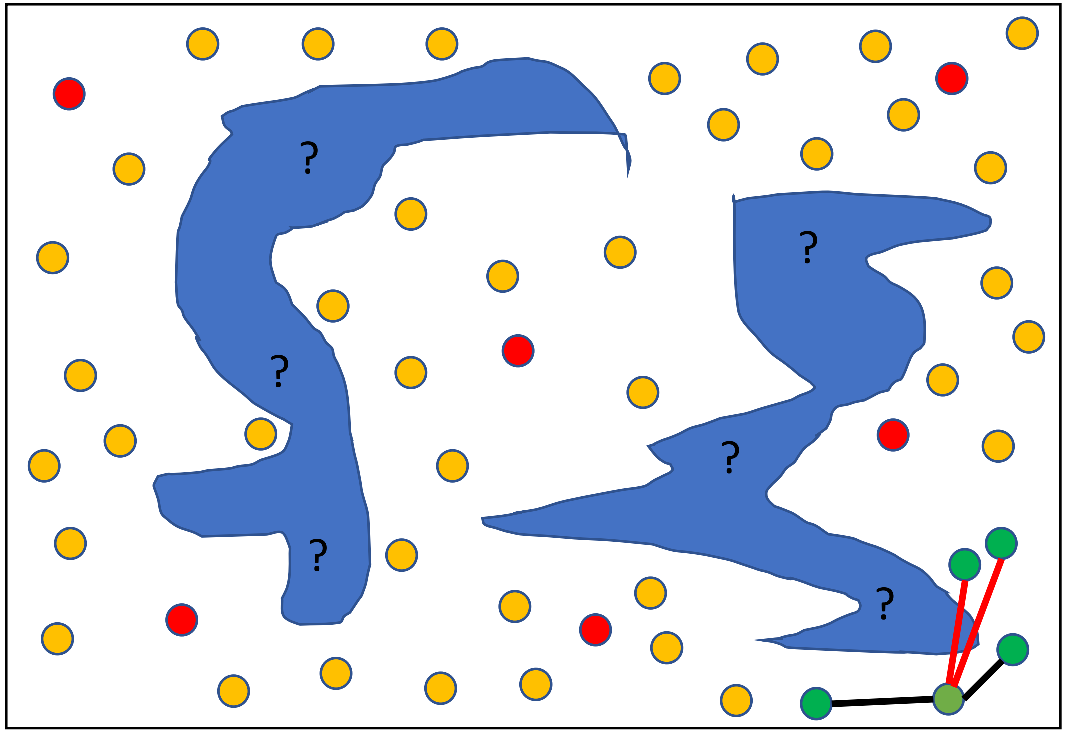


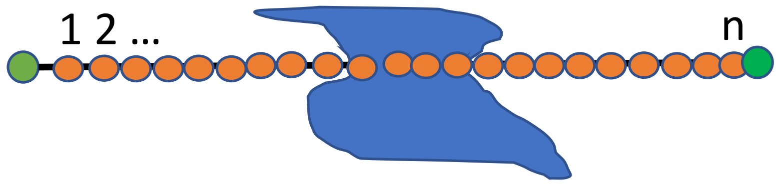
Variant 1:

1. Consider the problem in the workspace.
2. Do a continuous collision detection.
3. Describe the path of each vertex/geometry with a function. Compute intersection point of the geometry.

→ Very challenging. → We use more simple approach.

How to check for a valid edge?



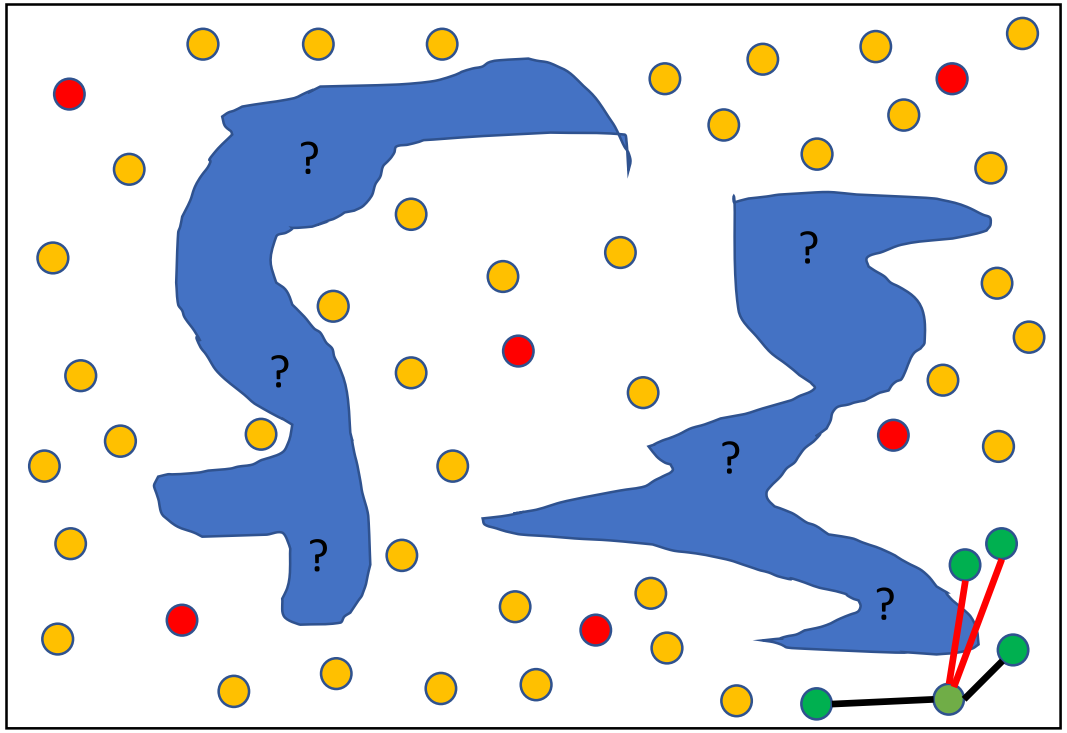


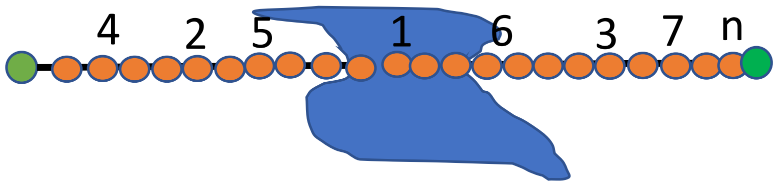
Variant 2:

1. Take a high and dense resolution.
2. Discretize the edge into many samples.
3. Start at the first configuration and go to goal.
4. If there is a invalid configuration

→ stop and return false, otherwise return true.

How to check for a valid edge?





Variant 3:

1. Take a high and dense resolution.
2. Discretize the edge into many samples.
3. Use interval halving.
4. If there is a invalid configuration

→ stop and return false, otherwise return true.

What can be parallelized with the sPRM

Algorithm 3: sPRM($\{(c^i_{init}, c^i_{goal})\}, r, n$)

$E \leftarrow \emptyset$, $V \leftarrow \emptyset$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$
for $j \leftarrow 0$ **to** n **do**
  $V \leftarrow V \cup CFreeSample()$
foreach $v \in V$ **do**
  $U \leftarrow Neighbors(v, V, r)$
  foreach $u \in U$ **do**
    if $(edgeIsValid(u, v))$ **then**
      $E \leftarrow E \cup (u, v)$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**
    $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$
  else
    $\sigma_i \leftarrow \emptyset$
return $\{\sigma_i\}$

Notes:
- Adding a configuration to an array can be parallelized.

Benefit:
Low.
The size of queries is small.

GPU brings benefits? No

What can be parallelized with the sPRM

Algorithm 3: sPRM($\{(c^i_{init}, c^i_{goal})\}, r, n$)

$E \leftarrow \emptyset, V \leftarrow \emptyset$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$
for $j \leftarrow 0$ **to** n **do**
  $V \leftarrow V \cup CFreeSample()$
foreach $v \in V$ **do**
  $U \leftarrow Neighbors(v, V, r)$
  foreach $u \in U$ **do**
    if $(edgeIsValid(u, v))$ **then**
      $E \leftarrow E \cup (u, v)$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**
    $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$
  else
    $\sigma_i \leftarrow \emptyset$
return $\{\sigma_i\}$

Notes:
- The collision detection of one configuration is independent of an other.
- Collision detection itsself can be parallelized.

Benefit:
High.
Collision detection is expensive and the amount of samples is large. Especially for more complex problems and higher dimensions.

GPU brings benefits? Yes

What can be parallelized with the sPRM

Algorithm 3: sPRM($\{(c^i_{init}, c^i_{goal})\}, r, n$)

- $E \leftarrow \emptyset,\ V \leftarrow \emptyset$
- **foreach** $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  - $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$
- **for** $j \leftarrow 0$ **to** n **do**
  - $V \leftarrow V \cup CFreeSample()$
- **foreach** $v \in V$ **do**
  - $U \leftarrow Neighbors(v, V, r)$
  - **foreach** $u \in U$ **do**
    - **if** $(edgeIsValid(u, v))$ **then**
      - $E \leftarrow E \cup (u, v)$
- **foreach** $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  - **if** $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**
    - $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$
  - **else**
    - $\sigma_i \leftarrow \emptyset$
- **return** $\{\sigma_i\}$

Notes:

- The collision detection of the edges are independent from each other.
- Collision detection itsself can be parallelized.

Benefit:

High.

Checking for an edge needs collision detection.

GPU brings benefits? Yes

What can be parallelized with the sPRM

Algorithm 3: sPRM$(\{(c^i_{init}, c^i_{goal})\}, r, n)$

$E \leftarrow \emptyset$, $V \leftarrow \emptyset$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  $V \leftarrow V \cup c^i_{init} \cup c^i_{goal}$
for $j \leftarrow 0$ **to** n **do**
  $V \leftarrow V \cup CFreeSample()$
foreach $v \in V$ **do**
  $U \leftarrow Neighbors(v, V, r)$
  foreach $u \in U$ **do**
    if $(edgeIsValid(u, v))$ **then**
      $E \leftarrow E \cup (u, v)$
foreach $((c^i_{init}, c^i_{goal}) \in \{(c^i_{init}, c^i_{goal})\})$ **do**
  if $connected(c^i_{init}, c^i_{goal}, V, E)$ **then**
    $\sigma_i = shortestPath(c^i_{init}, c^i_{goal}, V, E)$
  else
    $\sigma_i \leftarrow \emptyset$
return $\{\sigma_i\}$

Notes:

- The collision detection of the edges are independent from each other.
- Collision detection itsself can be parallelized.

Benefit:
Medium
Parallelizing of the Djkstra calls can bring benefits if the Amount of queries is high.

GPU brings benefits? No

Some advantages/disadvantages

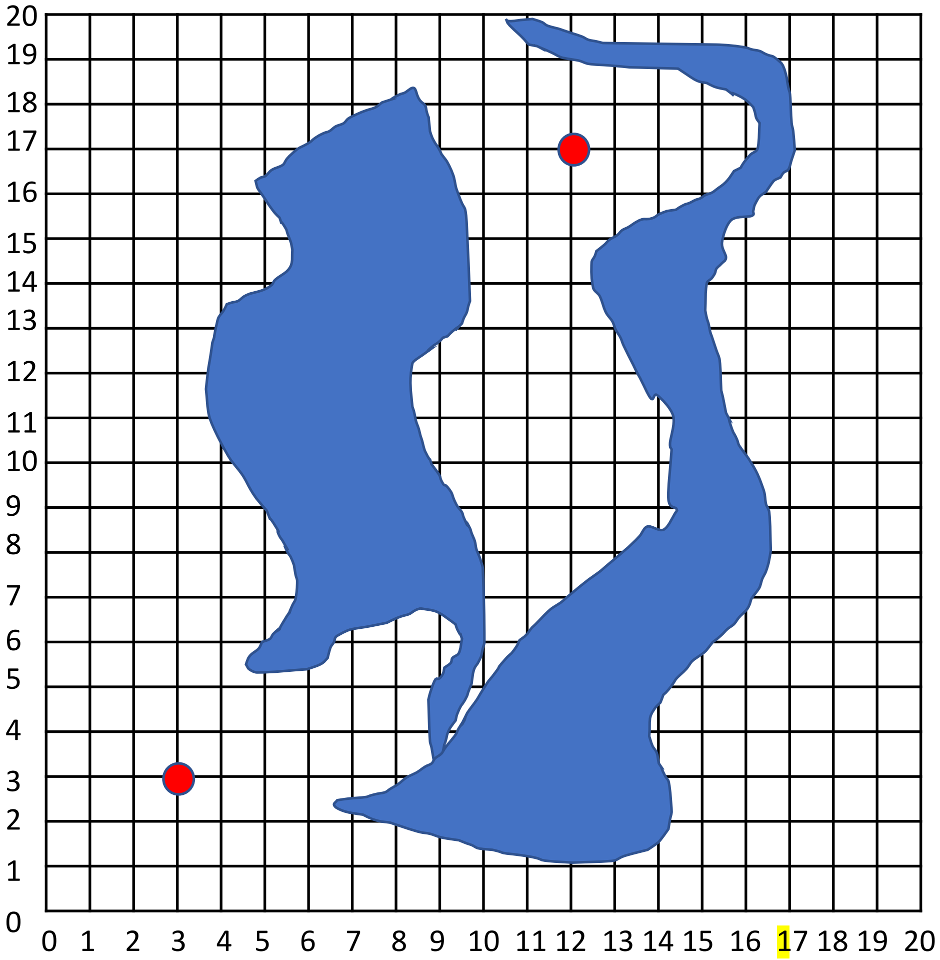
- Is usable to for multi-query motion planning.
- Is easy to implement, parallelize and GPU relevant.
- You have to define the amount of samples beforehand.
    - If the motion planning problem is easy → little samples are needed.
    - Unnecessary time is spend to solve the problem.
    - If the motion planning problem is hard → many samples are needed.
    - Algorithm has to be called multiple times with increasing samples.
- Needs some extensions to solve hard problems.
- Without parallelizing it is one of the slower motion planning algorithms.

Sampling-based motion planner

The sPRM falls into the category of "sampling-based motion planners":

- It uses samples of the configuration space to solve the problem.
- It builds up some kind of data structure in the configuration space.
- In the literature, there are many kind of data structures (sPRM =Map) in the configuration space.
- We will address the two major data structures in this lecture.
- Although the algorithms work in the configuration space → Many approaches include/transform information of the workspace into the configuration space.

Exercise



Compute the sPRM and give

- the solution sequence
- The size of V and E.

Input:

The start (3,3) and (12/17) are already added to V.

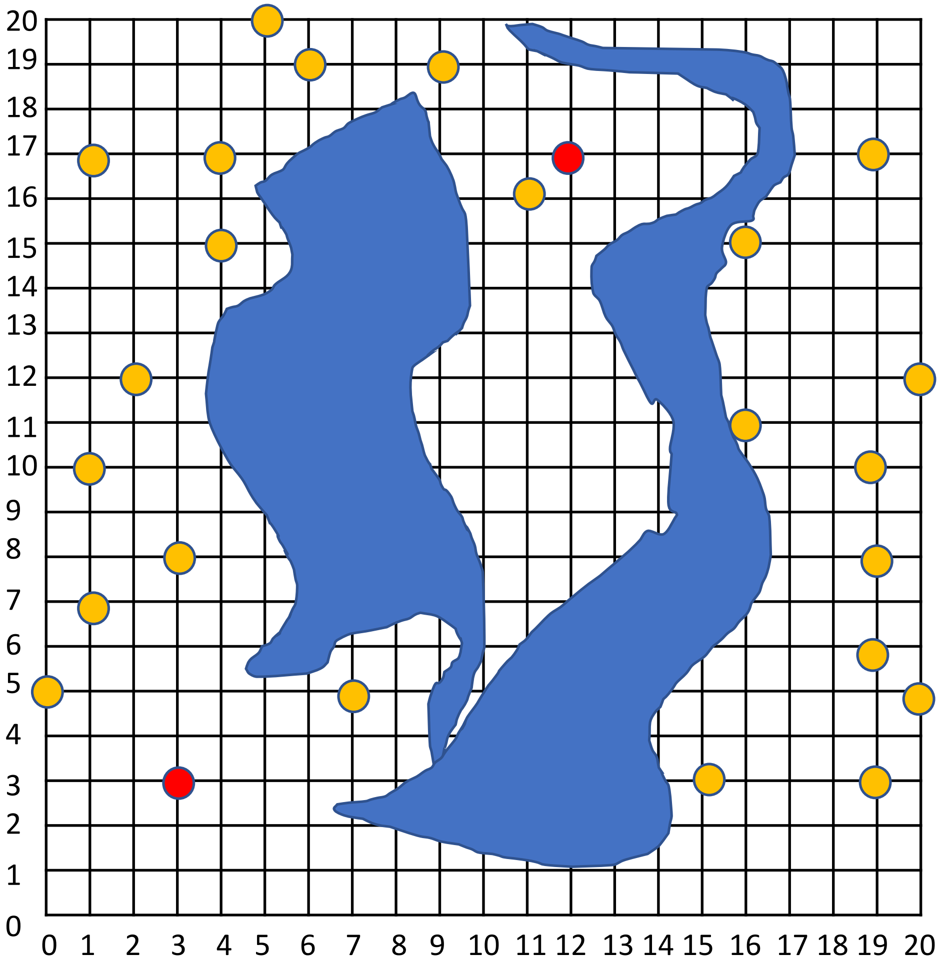
Use this random sequence:
1,10,2,12,19,10,19,6,19,17,16,11,20,12,19,
8,16,15,3,8,1,17,4,15,15,3,4,17,11,16,
19,3,20,5,0,5,5,20,9,9,7,5,
5,12,9,19,6,19,1,7

And use the radius $r = 3{,}99$. with Euclidean metric.

Note:

- Do not include samples of $\mathcal{C}_{obs}$ to the roadmap
- Connect only configuration with edges where allowed based on the sPRM algorithm.
- Edges are only included once.

Exercise



Compute the sPRM and give

- the solution sequence
- The size of V and E.

Input:

The start (3,3) and (12/17) are already added to V.

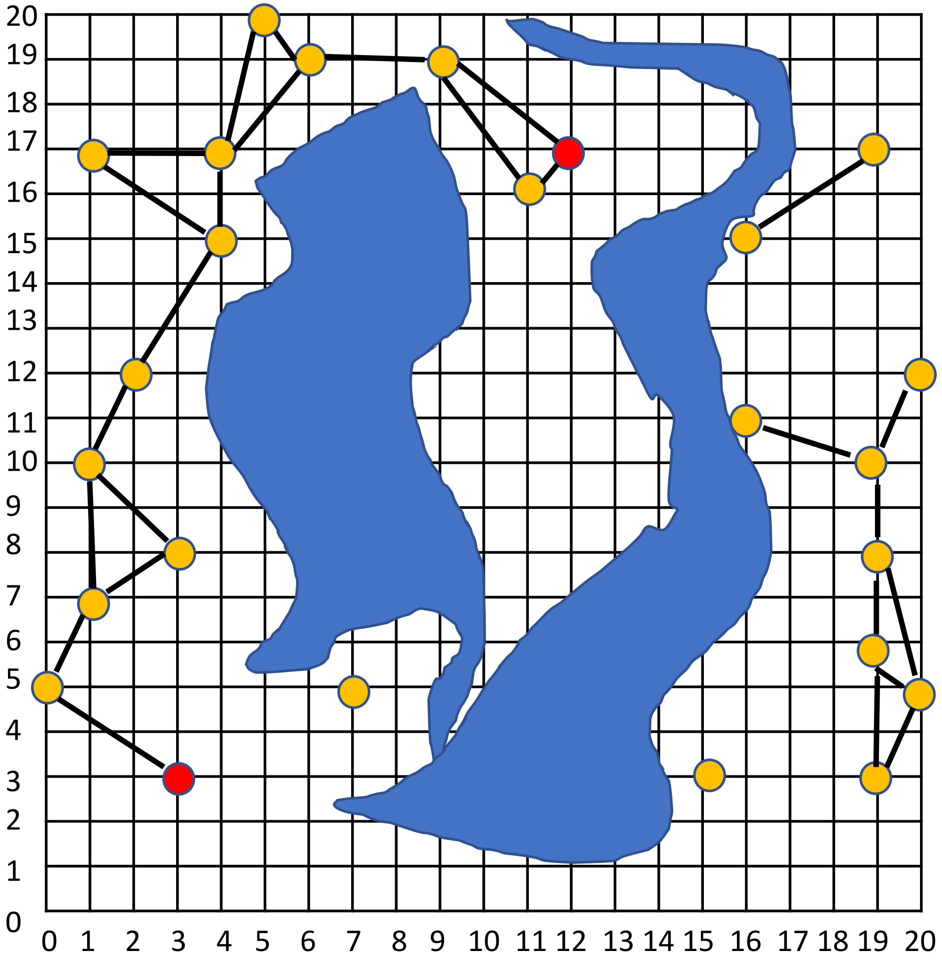
Use this random sequence:
1,10,2,12,19,10,19,6,19,17,16,11,20,12,19,
8,16,15,3,8,1,17,4,15,15,3,4,17,11,16,
19,3,20,5,0,5,5,20,9,9,7,5,
5,12,9,19,6,19,1,7

And use the radius $r = 3{,}99.$ with Euclidean metric.

Note:

- Do not include samples of $\mathcal{C}_{obs}$ to the roadmap
- Connect only configuration with edges where allowed based on the sPRM algorithm.
- Edges are only included once.

Exercise



Compute the sPRM and give

- the solution sequence
- The size of V and E.

Input:

The start (3,3) and (12/17) are already added to V.

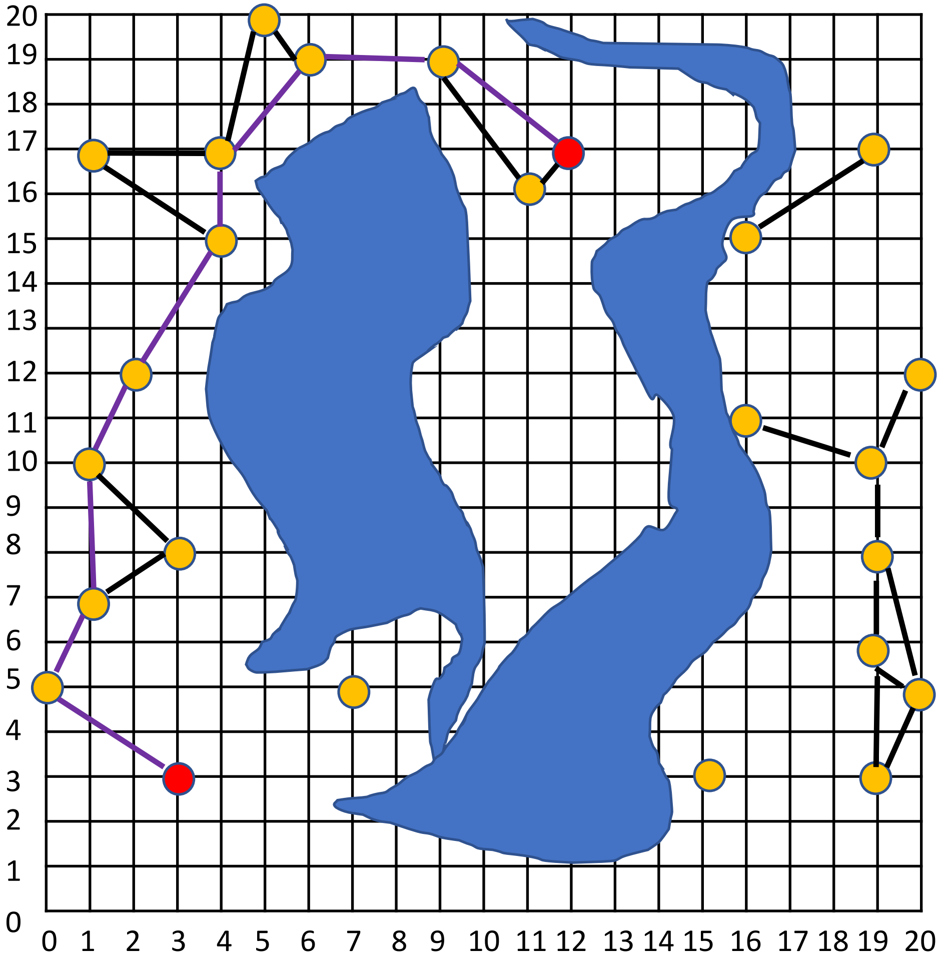
Use this random sequence:
1,10,2,12,19,10,19,6,19,17,16,11,20,12,19,
8,16,15,3,8,1,17,4,15,15,3,4,17,11,16,
19,3,20,5,0,5,5,20,9,9,7,5,
5,12,9,19,6,19,1,7

And use the radius $r = 3{,}99$. with Euclidean metric.

Note:

- Do not include samples of $\mathcal{C}_{obs}$ to the roadmap
- Connect only configuration with edges where allowed based on the sPRM algorithm.
- Edges are only included once.

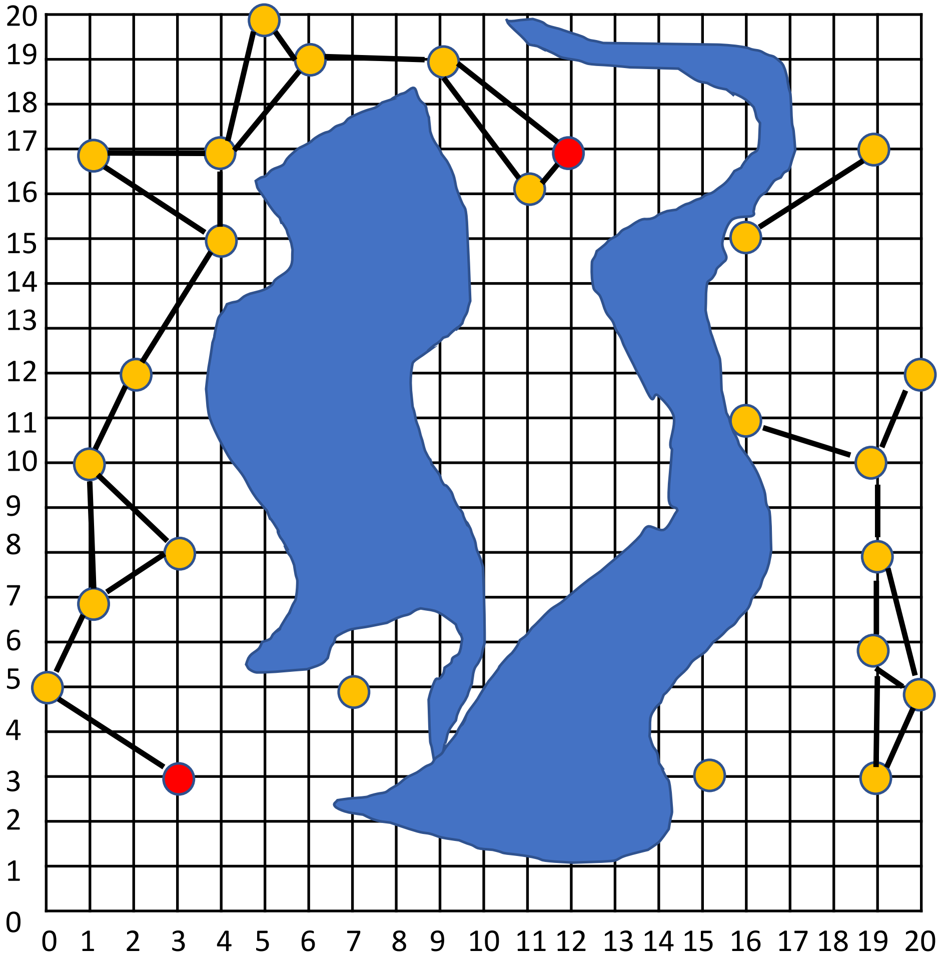
Exercise



Solution:

V has a size of 25
E has a size of 26
Solution path is:
3/3
0/5
1/7
1/10
2/12
4/15
4/17
6/19
9/19
12/17

Exercise



20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20