Report for Assignment 3

April 8, 2020

Implementation

Based on the code given, we implement the path planning simulation domain (2D map navigation) in **createDomain.py**. We treat the domain as an object which consists of its dimension, obstacles, goal and start. When creating a domain instance, we specify the number of obstacles, which helps facilitate our experiment in part 4. We also write some codes to help visualize our result by using matplotlib.pyplot. This function, whose name is **drawDomain**, receives a path (a list of tuples) and draws the path within the domain.

Quadtree vs FBSP Decomposition

write your result here plz

Rapidly Exploring Random Tree (RRT)

We first implement some additional data structures (Node and Tree) in **Structure.py**. Node is a subclass of obstacle but it has one more instance: parent. Each tree consists of a list of nodes and its root. Each tree has a function called **findNearestNode**, it receives a node in the domain and outputs the nearest node in the tree.

We then implement the algorithm in **RRTsolver.py**. A solver receives a problem instance (a domain) and outputs the solution and stats. At each iteration, **solve** (the algorithm) generates a random node with some probability of choosing the goal(**randomNode**). Then it attempts to find a new node on the line connecting the random node and the nearest node, determined by the step size. Finally, it connect this new node to the nearest node in the tree (**extend**). When the new node is extremely close to the goal, the algorithm connects the new node and the goal directly and the problem is solved and **traceBack** is triggered. It takes advantage of the data structure and outputs a whole path from the goal to the start by calling a node's parent repeatedly. Here are some screenshots for different numbers of obstacles.

When the number of obstacles is 10:

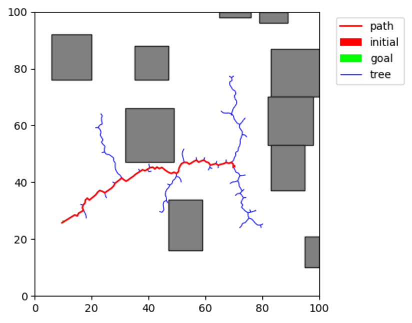

Figure 1: 10 obstacles, length,width$\sim$ uniform$(10, 20)$

When the number of obstacles is 10:

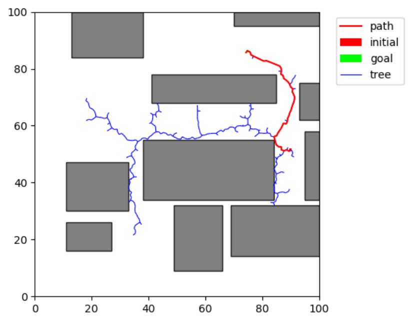

Figure 2: 10 obstacles, length,width$\sim$ uniform$(10, 50)$

When the number of obstacles is 20:

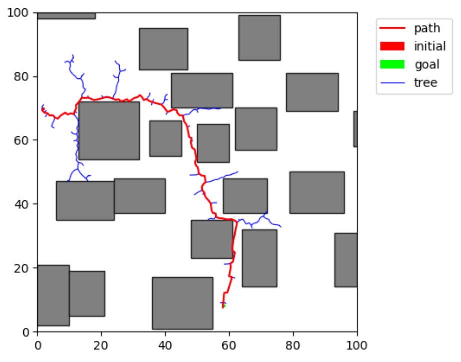

Figure 3: 20 obstacles, length,width$\sim$ uniform$(10, 20)$

When the number of obstacles is 20:

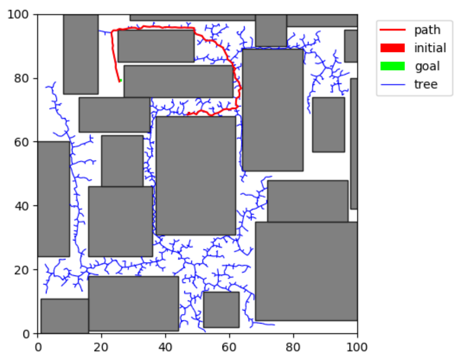

Figure 4: 20 obstacles, length,width$\sim$ uniform$(10, 50)$

When the number of obstacles is 30:

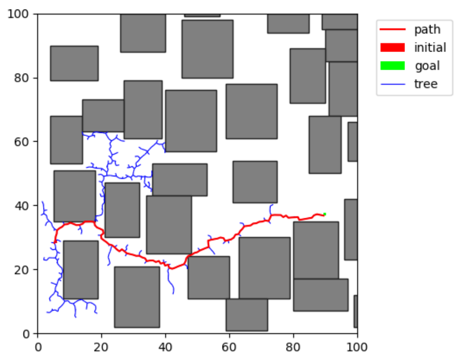

Figure 5: 30 obstacles

When the number of obstacles is 40:

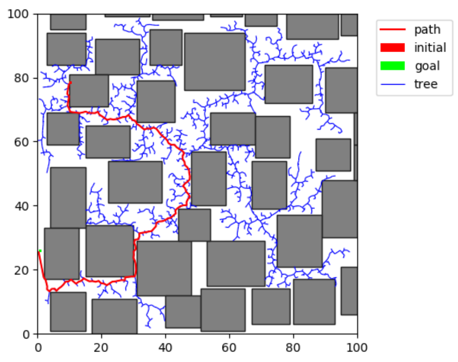

Figure 6: 40 obstacles

When the number of obstacles is 45:

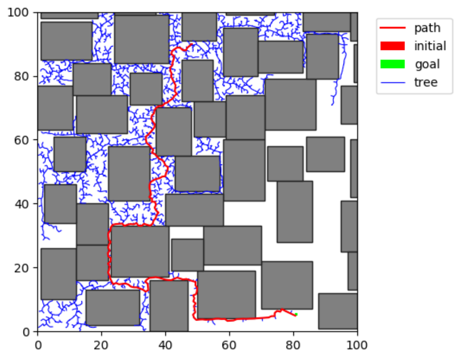

Figure 7: 45 obstacles

When the number of obstacles is 45:

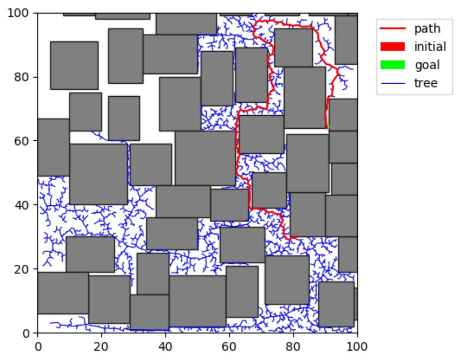

Figure 8: 45 obstacles

After repeating the experiment several times, we are able to obtain a table, which indicates the behavior of the algorithm:

Experiment			
number of obstacles/maximum size	run time(avg)	num of nodes in the path(avg)	num of nodes in the tree(avg)
10/20	0.17	45.25	169.75
10/50	0.55425	87.25	252.25
20/20	0.8525	91.5	333
20/50	5.979	98.25	795
30/20	1.789	115.75	414
40/20	20.715	136.75	1459
45/20	57.59	167.67	2511.33

In summary, when the maximum size of obstacle is fixed, the run time and the number of nodes in the tree increase exponentially as the number of obstacles increases. The number of nodes in the path increases as the number of obstacles increases, but it is impacted less than the run time. On the other hand, when the number of obstacles is fixed, if we change the maximum size of obstacles, the rum time and the number of nodes in the tree increase in general. Furthermore, if the number of obstacles is already large, then changing the maximum size will bring less impact to the behavior of the algorithm.