Emanuele Costantino

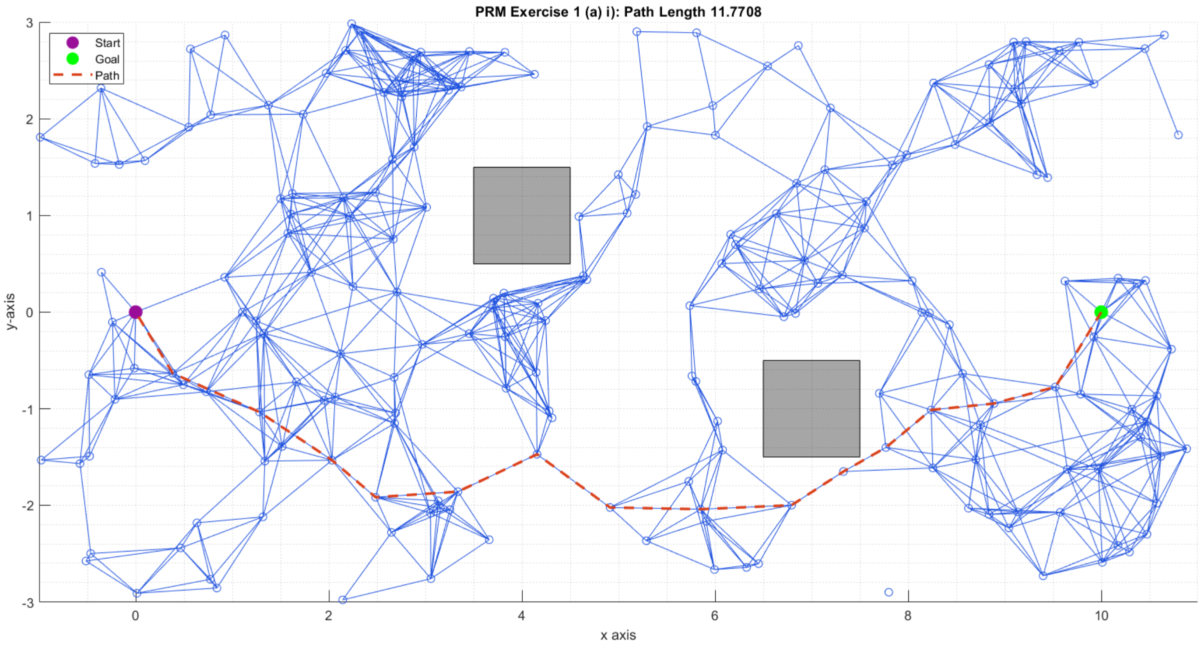
10/12/2021

Homework #7

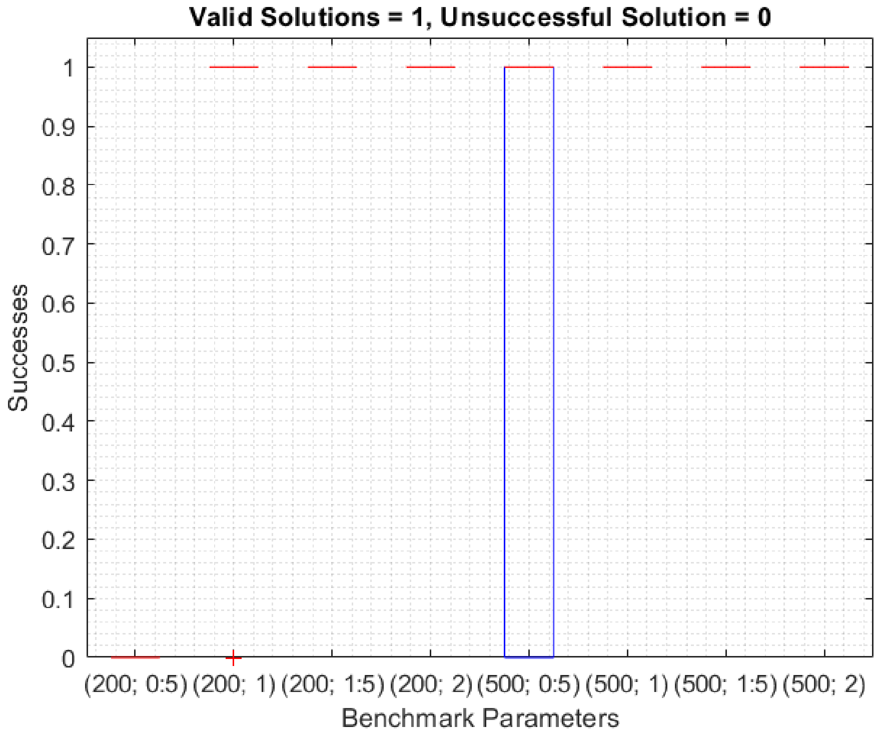
Exercise 1

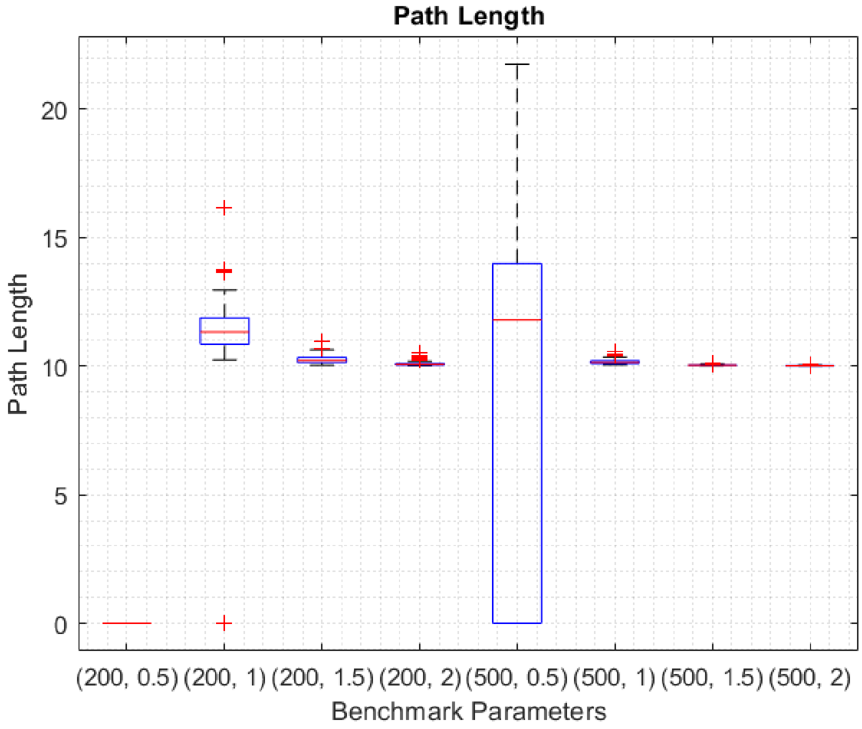
a) Exercise 2(a) Homework 5

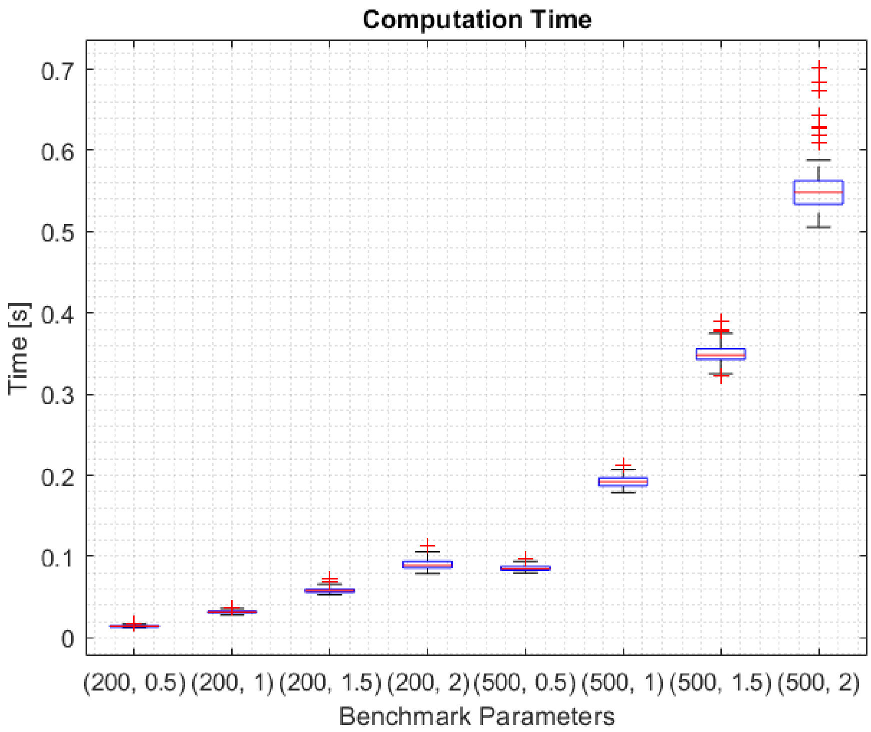
i) Roadmap Plot



ii) Benchmark Box Plots


Valid Solutions = 1, Unsuccessful Solution = 0
Successes
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
(200; 0:5)
(200; 1)
(200; 1:5)
(200; 2)
(500; 0:5)
(500; 1)
(500; 1:5)
(500; 2)
Benchmark Parameters

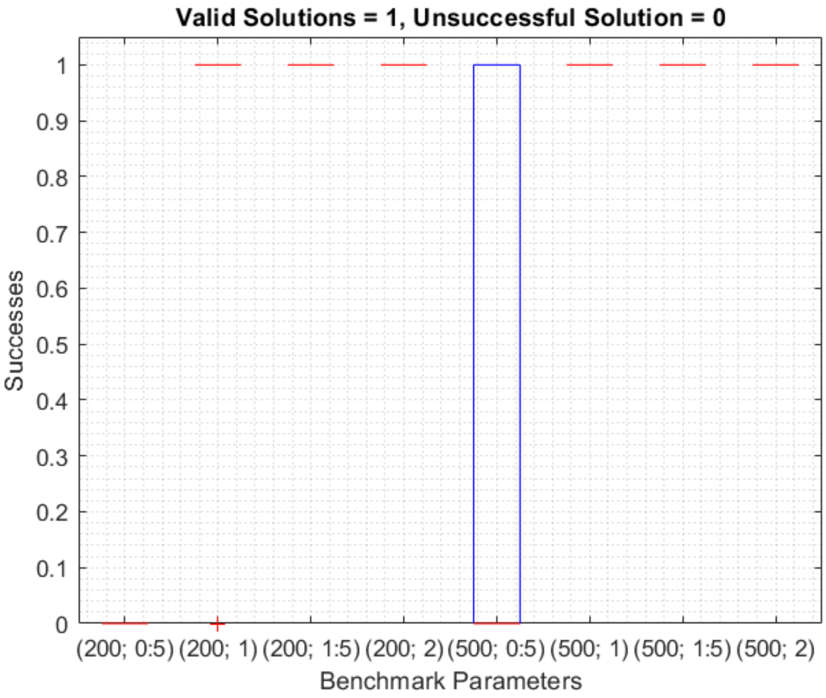

Path Length
Path Length
20
15
10
5
0
(200, 0.5)
(200, 1)
(200, 1.5)
(200, 2)
(500, 0.5)
(500, 1)
(500, 1.5)
(500, 2)
Benchmark Parameters

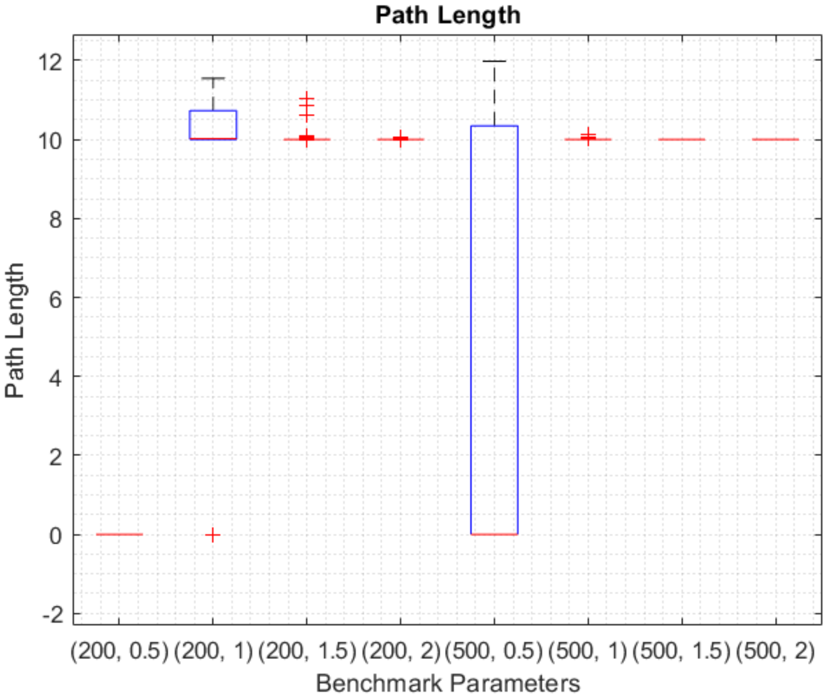


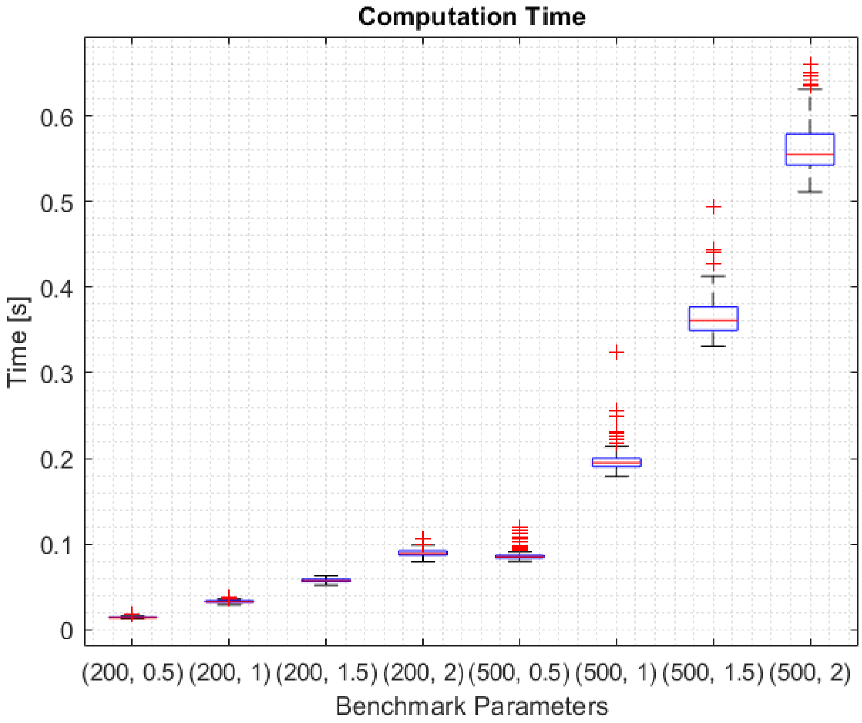
iii) To maximize number of successful paths found while minimizing the path length and computation time, I found that n = 200 and r = 2 is the optimal configuration. With this configuration, the PRM algorithm always found a path with relatively (with respect to the other path lengths) short path lengths and few outliers. This configuration does have a slightly higher average runtime, but I believe finding a short path is much more important than computation time within reason. For n = 500 and r = 2 , PRM was always successful in finding a short/low variance path, but the average run time was roughly 850% longer than the n = 200, r = 2 configuration with only a 0.69% decrease in path length.

NOTE: if a path was not found, the path length for that run was set to 0. I could have also gone the other way and made the path length for unsuccessful paths a very large number.

iv) Once again, n = 200 and r = 2 is the optimal configuration. This configuration was always successful (within the 100 runs) in finding a valid path, mean path length was close to that of the n = 500, r = 2 configuration with few outliers, and an average run time below 0.1 seconds.


Valid Solutions = 1, Unsuccessful Solution = 0
Successes
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
(200; 0:5)
(200; 1)
(200; 1:5)
(200; 2)
(500; 0:5)
(500; 1)
(500; 1:5)
(500; 2)
Benchmark Parameters

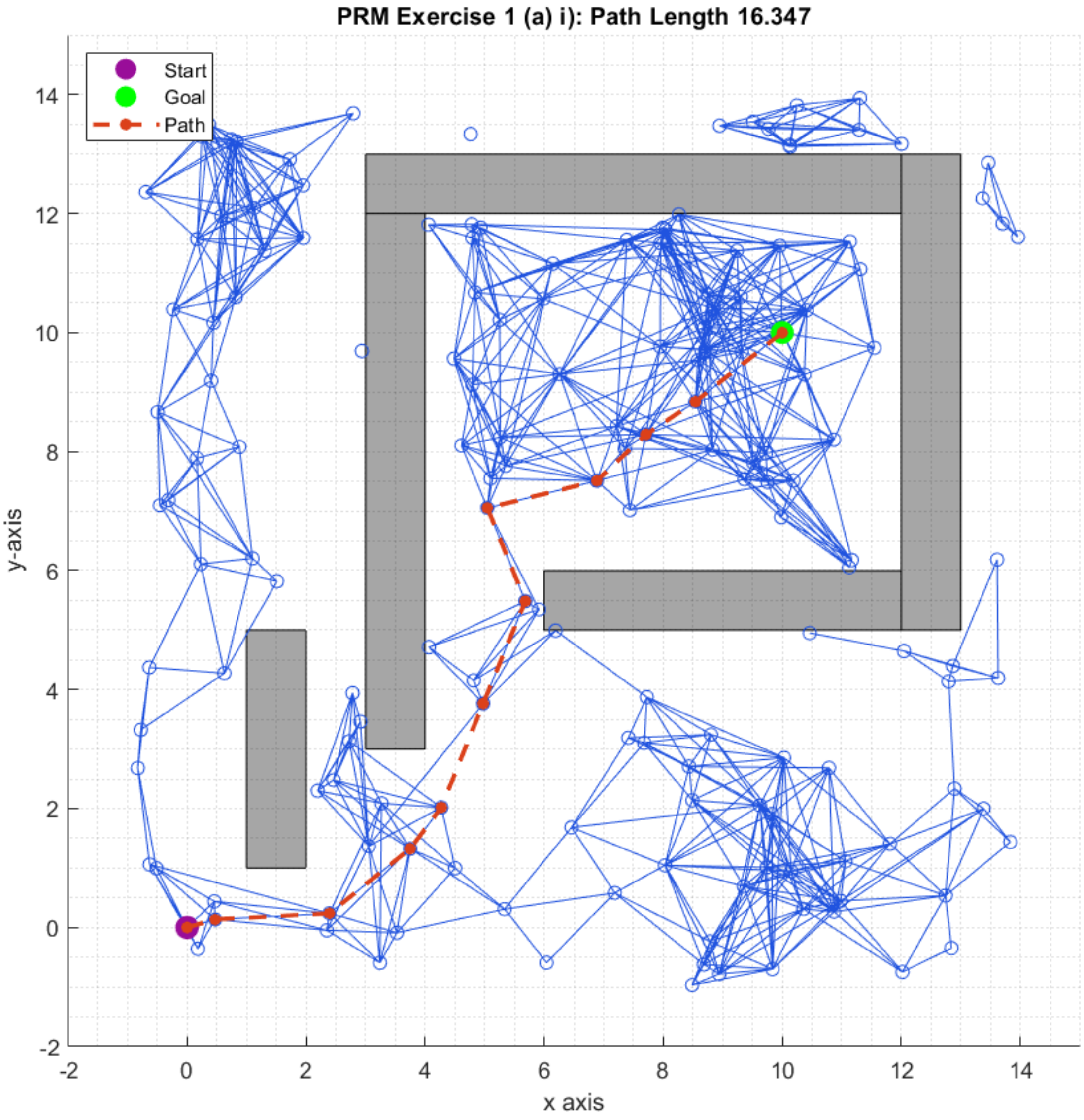

Path Length
Path Length
12
10
8
6
4
2
0
-2
(200, 0.5)
(200, 1)
(200, 1.5)
(200, 2)
(500, 0.5)
(500, 1)
(500, 1.5)
(500, 2)
Benchmark Parameters


Computation Time
Time [s]
0
0.1
0.2
0.3
0.4
0.5
0.6
(200, 0.5)
(200, 1)
(200, 1.5)
(200, 2)
(500, 0.5)
(500, 1)
(500, 1.5)
(500, 2)
Benchmark Parameters

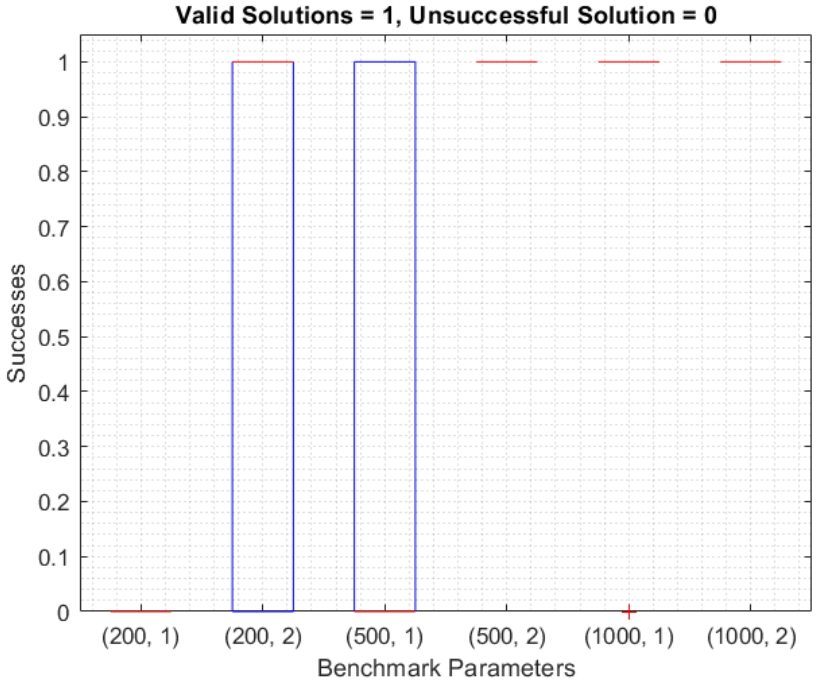
b)

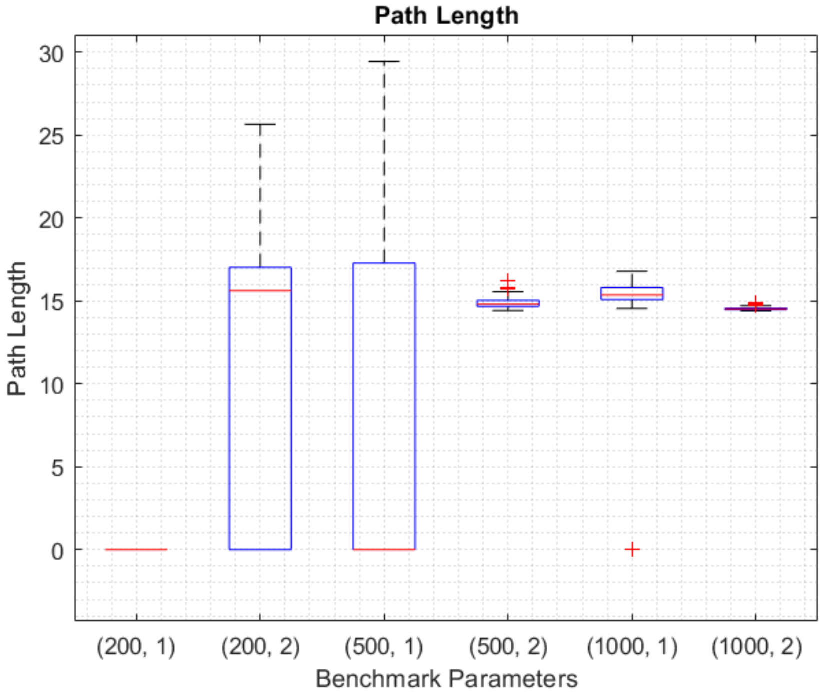
WORKSPACE 1

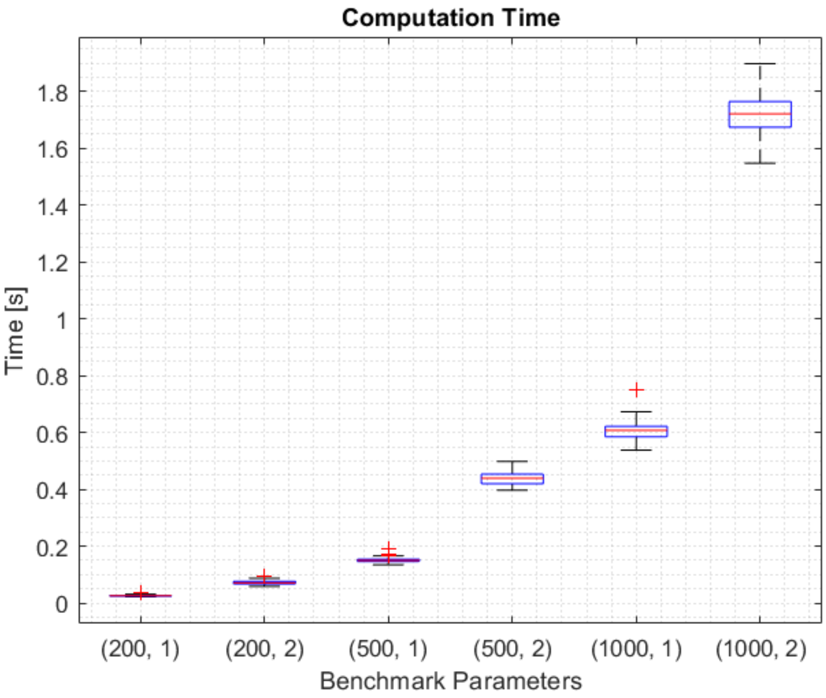
i) n = 200 and r = 2 PRM and path



ii) Benchmark Box Plots

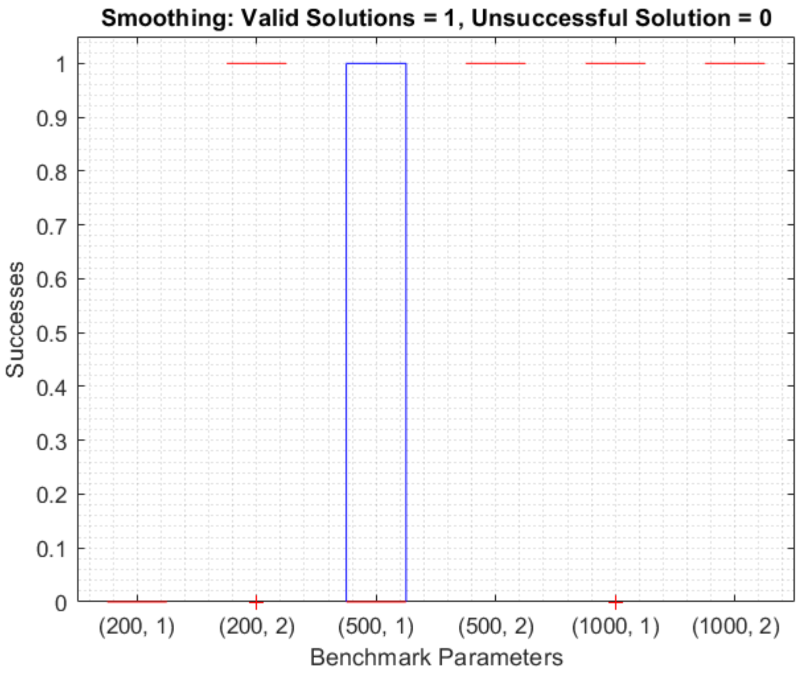

Valid Solutions = 1, Unsuccessful Solution = 0
Successes
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
(200, 1)
(200, 2)
(500, 1)
(500, 2)
(1000, 1)
(1000, 2)
Benchmark Parameters

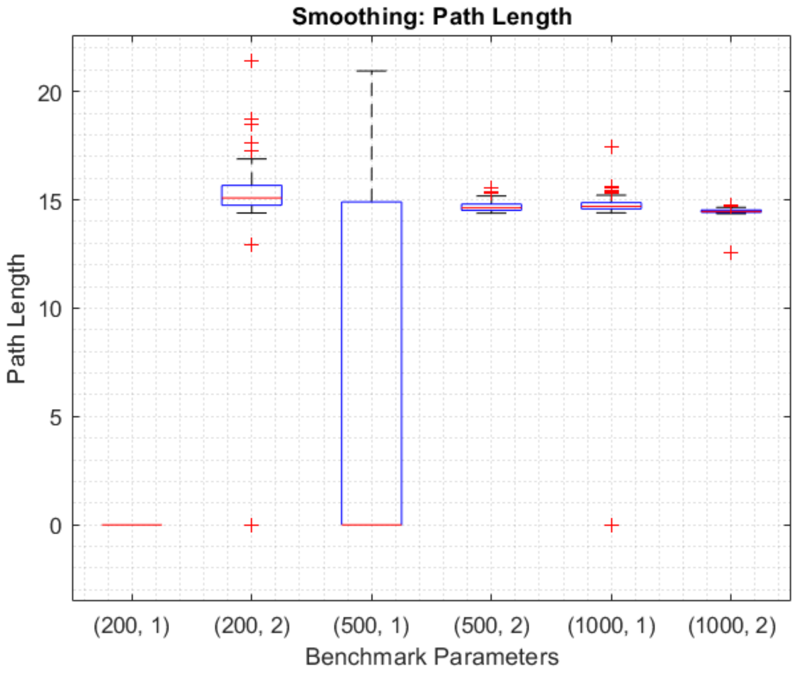

Path Length
Path Length
0
5
10
15
20
25
30
(200, 1)
(200, 2)
(500, 1)
(500, 2)
(1000, 1)
(1000, 2)
Benchmark Parameters

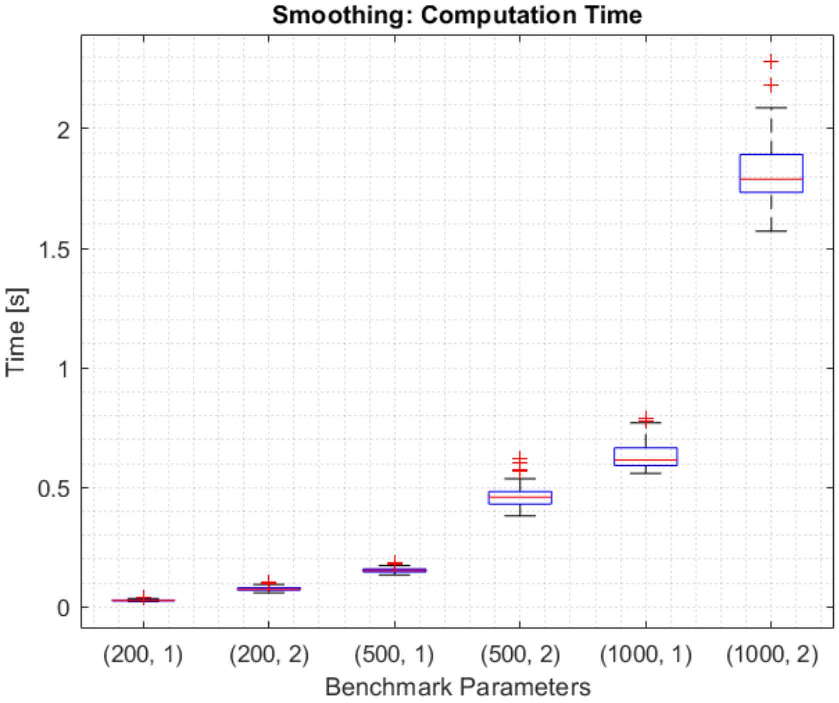


iii) For W1, n = 500 r = 2 is the best configuration because it was able to always find a valid solution for each of the 100 runs. Additionally, the path length boxplot had few outliers and a median path length roughly 2% larger than the n = 1000 r = 2 configuration while being approximately 3 times faster.

iv) For W1, n = 500 r = 2 is the best configuration after smoothing because it was able to always find a valid solution for each of the 100 runs. Additionally, the path length boxplot had few outliers and a median path length roughly 1% larger than the n = 1000 r = 2 configuration while being approximately 4 times faster.



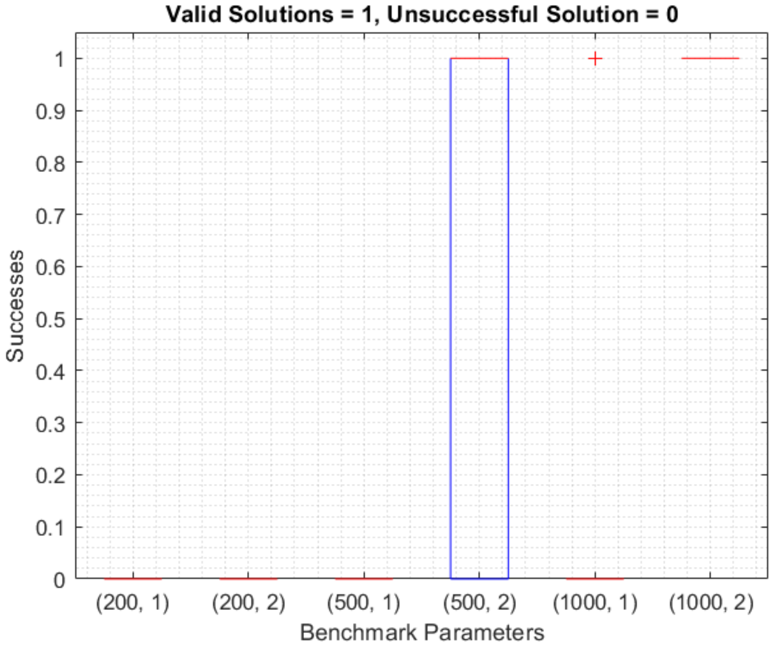


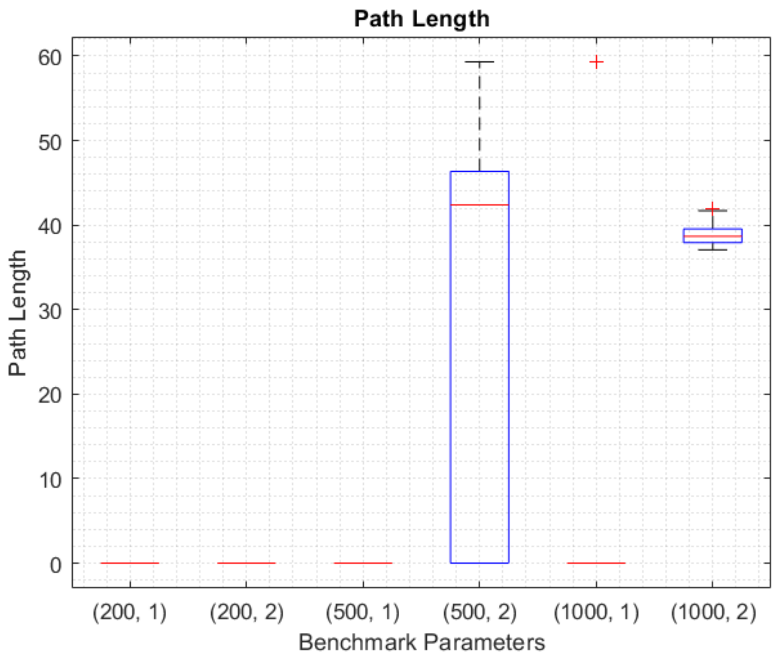


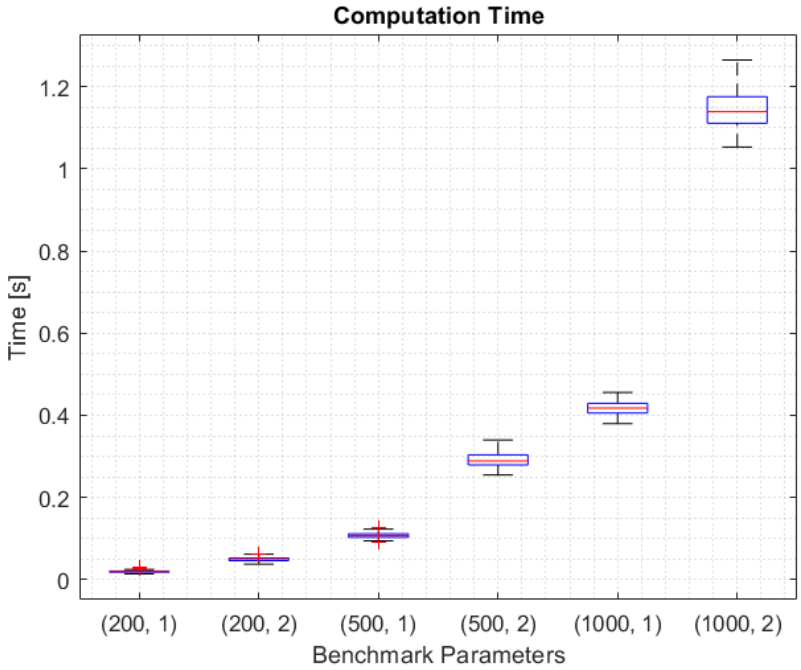
WORKSPACE 2

i) For n = 200 and r = 2 PRM was not able to find a valid path for W2.

ii)

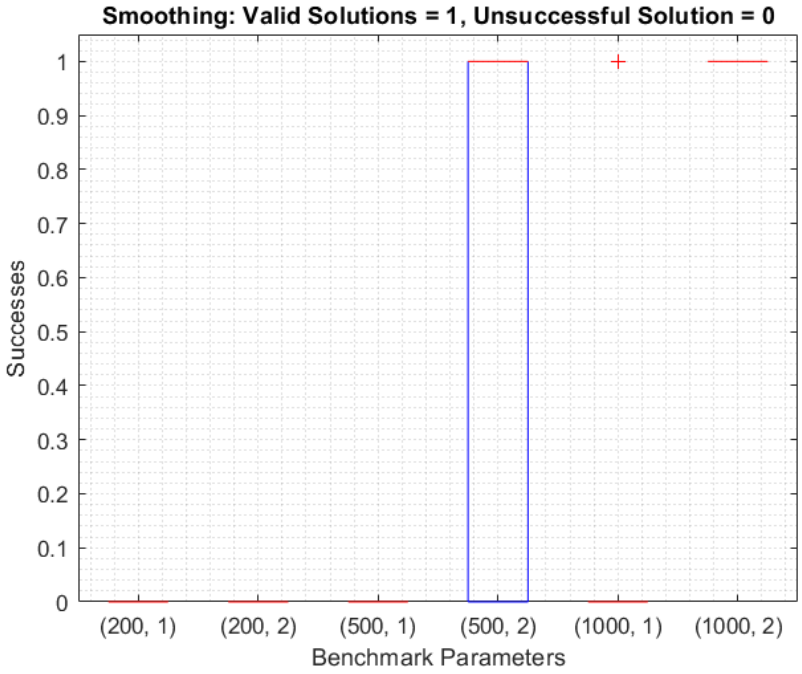


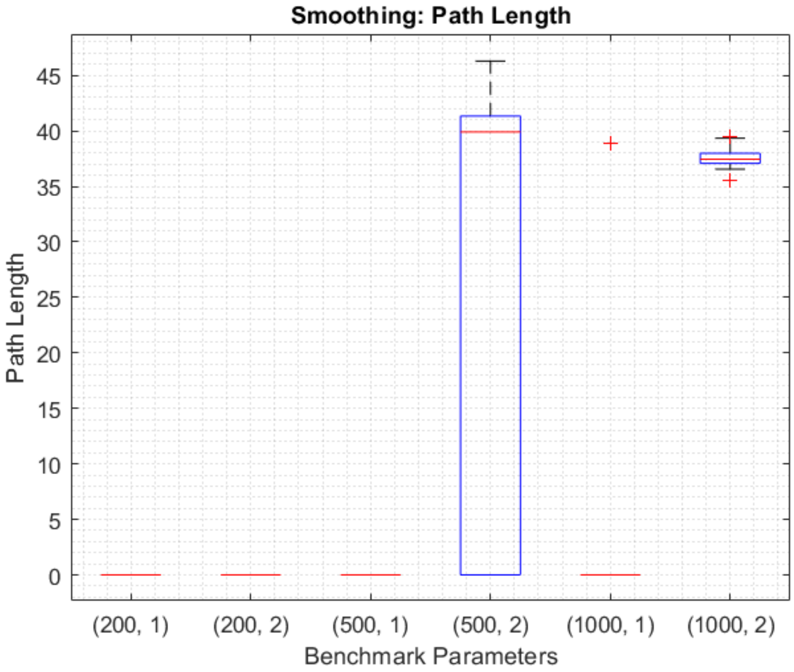


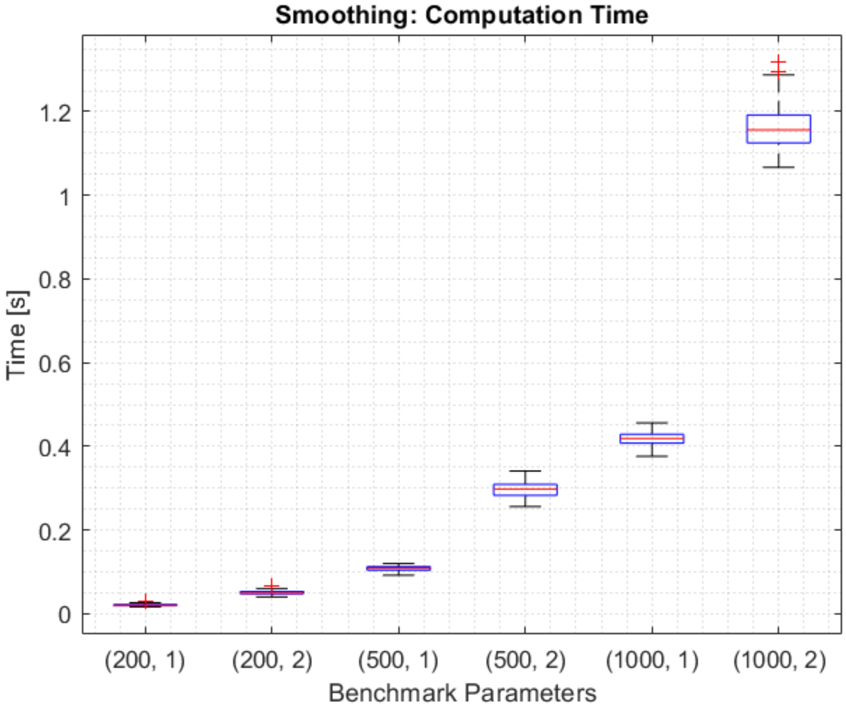


iii) For W2, n = 1000 r = 2 is the best configuration because it was the most reliable in finding valid solutions in the 100 runs. All other configurations had either very large box plots or low success rate for valid solutions. Although this configuration does take long to run, finding a path has a larger weight than the other benchmarks.

iv) Even after smoothing, for W2, n = 1000 r = 2 is the best configuration for the same reasons stated in art iii)





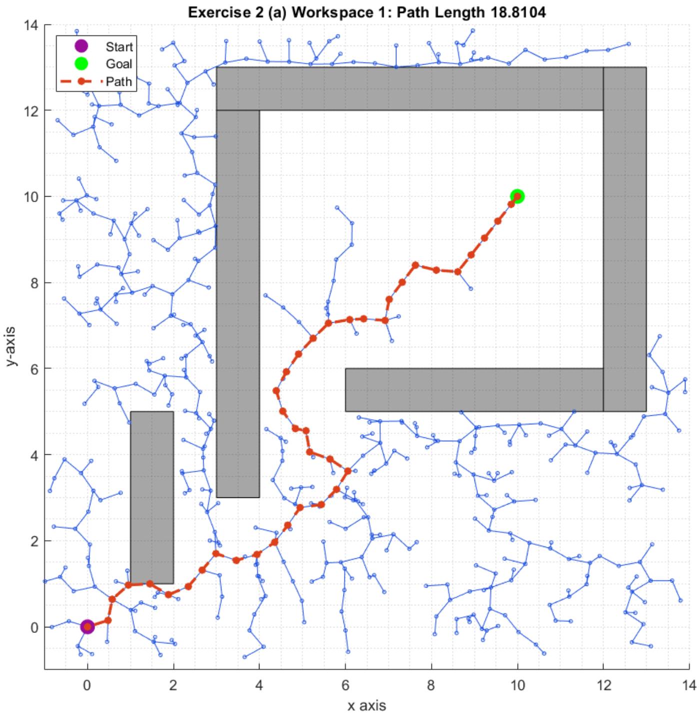


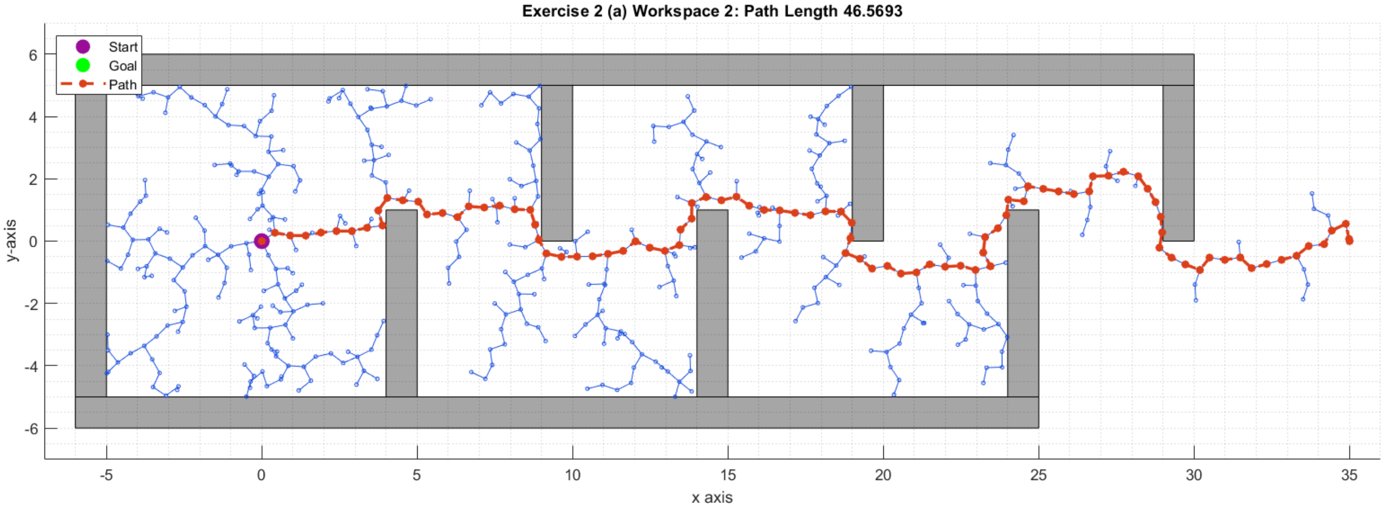
c) Yes. Small changes would have to be made in the sense that we now introduce a kinematic model of the 2-DOF manipulator. We will need to use this model to check for collisions when we randomly sample for θ_1 and θ_2. After this, I could freely sample and check for collisions in the *discrete* C-Space map to ensure that connections between samples do not collide C-Space obstacle. This would be relatively simple because I represent each grid as a square which I could easily add as an obstacle file in my code.

Additionally, how you define distance in this workspace is very important because our topology has changed from R^2 to T^2. This means that we can suddenly "appear" on the other side of our C-space because 0 = 2π to cosine and sine.

Exercise 2

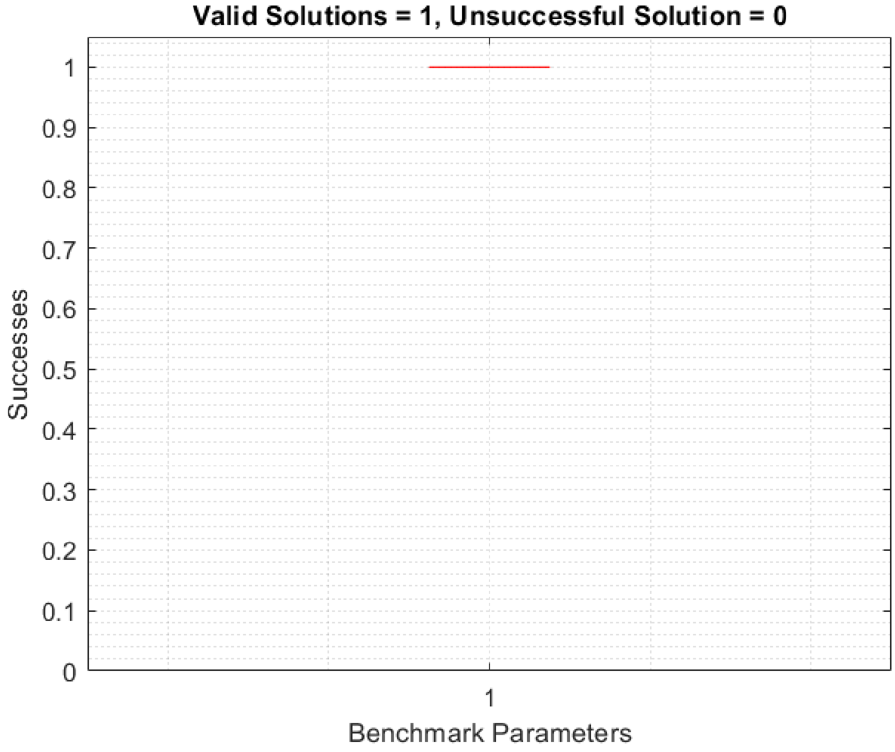
a) GoalBiasRRT planner path

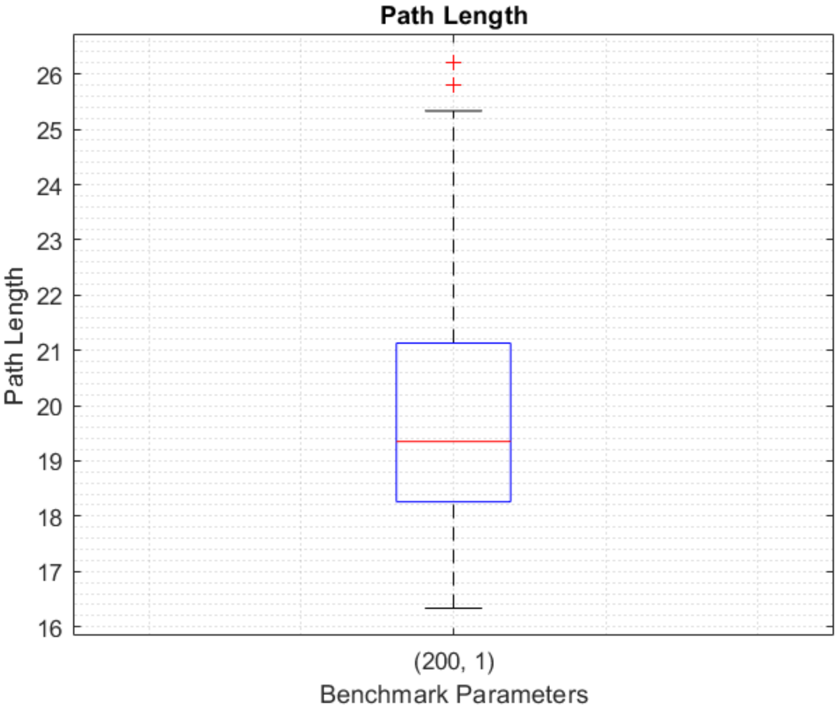


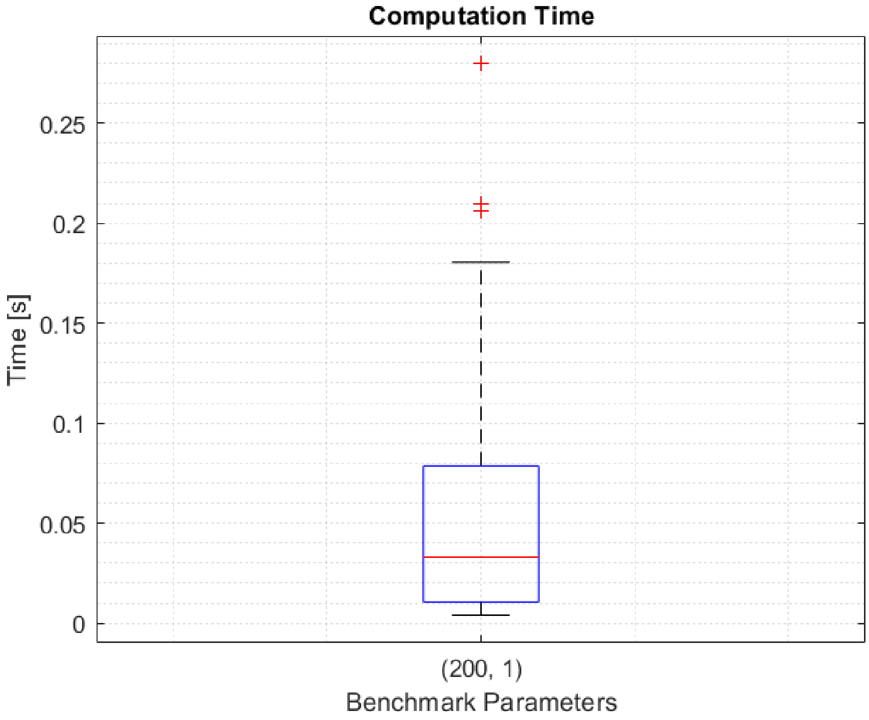


b)

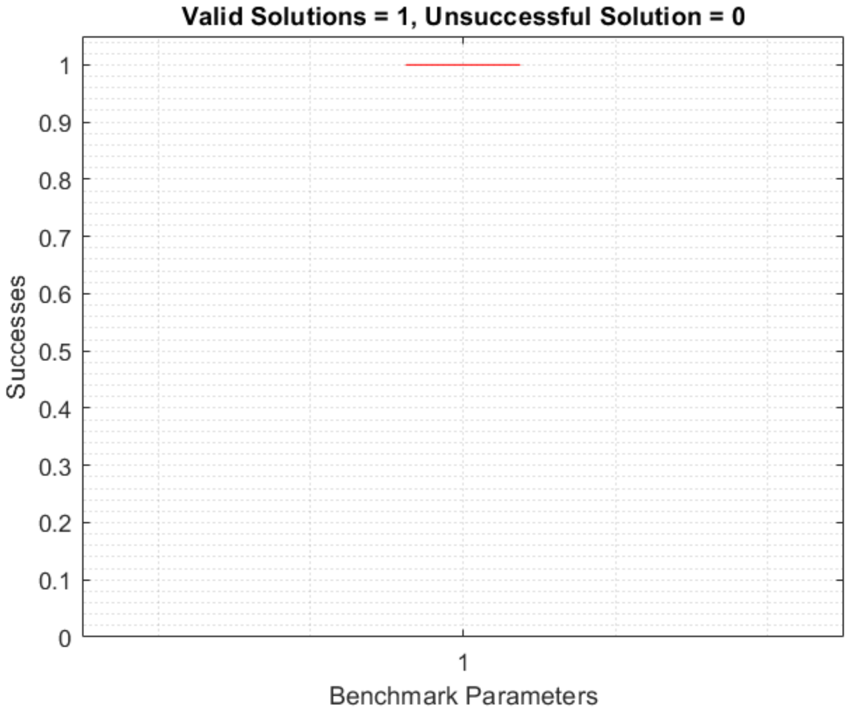
Workspace 1

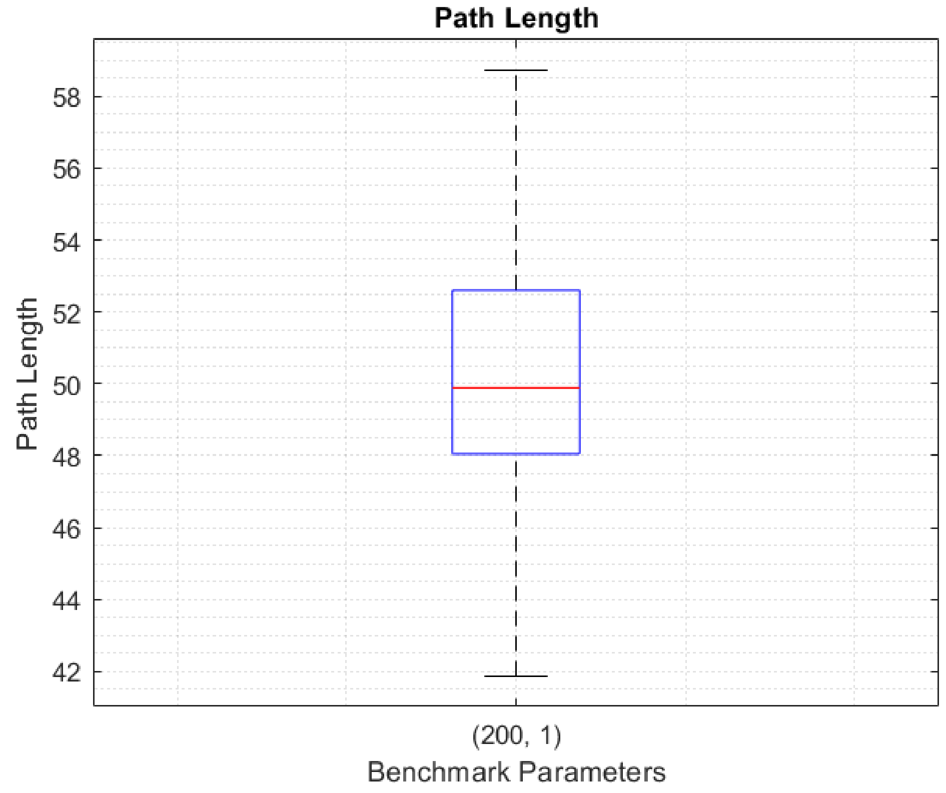


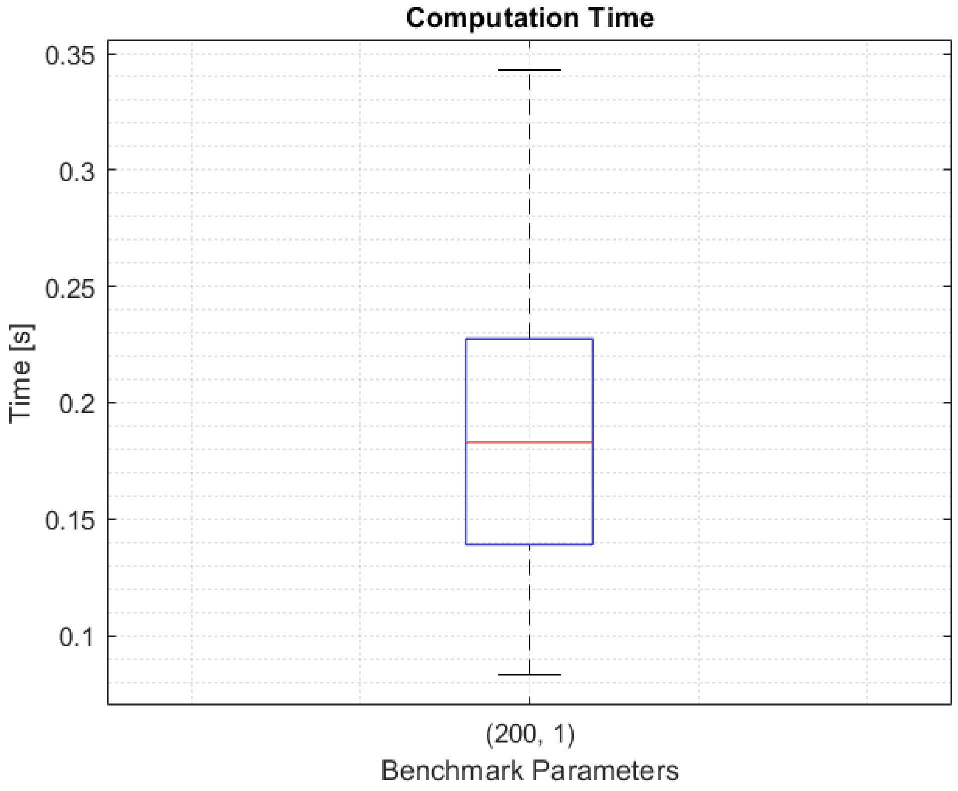



Computation Time
Time [s]
0
0.05
0.1
0.15
0.2
0.25
(200, 1)
Benchmark Parameters

Workspace 2


Valid Solutions = 1, Unsuccessful Solution = 0
Successes
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1
Benchmark Parameters


Path Length
Path Length
58
56
54
52
50
48
46
44
42
(200, 1)
Benchmark Parameters


Computation Time
Time [s]
0.35
0.3
0.25
0.2
0.15
0.1
(200, 1)
Benchmark Parameters

c) Yes. Just like PRM, small changes would need to be made to add the kinematic model of the 2-DOF manipulator. We will need to use this model to check for collisions when we randomly sample for θ_1 and θ_2. After this, I could freely sample and check for collisions in the *discrete* C-Space map to ensure that connections between samples do not collide C-Space obstacle. This would be relatively simple because I represent each grid as a square which I could easily add as an obstacle file in my code. How we define distance in this workspace will also change because our topology has changed from R^2 to T^2. This means that we can suddenly "appear" on the other side of our C-space because 0 = 2π to cosine and sine.

Exercise 3

Gradient Descent: the gradient descent method was the most difficult to use across various maps. The local minima problem is prevalent even in simple maps. This required clever algorithms to find Q* and d^*_{goal} or prior knowledge of the map to be able to find a path. Additionally, results were highly variable depending on how you chose minimum distance to obstacle. The path lengths generated by the gradient descent method were "reasonable" and allowed for the robot to keep a safe distance from the obstacles. In general, I believe that the gradient descent method is unreliable and should be avoided for real world applications. I could see how this method may be useful in higher dimensional applications.

Wavefront: since the wavefront planner is complete with respect to the grid, it was important to have some idea of what the maps you were planning in looked like. This would allow you to determine the grid size necessary to be able to move through close obstacles. The wavefront planner has to major flaws: computation time and proximity to obstacles. The wavefront planner, as I implemented, to several minutes to generate a discretized version of the maps. Additionally, the planner traveled very close to the obstacles. This would not be desirable in a real-world application. The wavefront planner was convenient because once the map was discretized/initialized, the planner would be able to find the shortest path with respect to the grid to goal. Overall, this planner is great for spaces that are or can be easily discretized (mazes). The wavefront planner would not be efficient in higher dimensions as the number of generated cells is exponential with the number of dimensions.

PRM: As we saw in this assignment, PRM was successful in finding a path to goal when the number of nodes (n) and neighbor radius (r) allowed for connectivity between sub graphs of the roadmap. The great thing about PRMs is that if a path is not found for a certain (n,r) pair, it is easy to iterate through different combinations to find one that works. Because PRMs are probabilistically complete, the user need only increase n (obviously not to infinity) iteratively to see if they may find a path to goal in a reasonable amount of time. The user is then limited only by the required computation time of their PRM algorithm. I would imagine that in real life the user would set an upper bound on the number of nodes before they may claim that a path doesn't exist.

RRT: RRT performed very well for the parameters chosen in this assignment. I believe this is because we allowed for 5000 nodes in our tree. RRT is primarily concerned with reaching, and unlike PRM, RRT stops running once a new node is within ϵ distance of goal. This cause RRT to terminate very quickly and return a path to goal. Compared to PRM, wavefront, and gradient descent, the paths returned by RRT were very inefficient and would require some smoothing. I believe RRT would be the most applicable in real world scenarios with very limited knowledge of the local environment.