ASEN 5519

Homework 4

Problem 1:

a) Path: Start -> C -> K -> Goal
   Path Length: 7
   Number of Iterations: 9

b) To turn the algorithm into Dijkstra's, I made the heuristic values for each node equal to zero so that the priority is only dependent on the path length from start.

c) Path: Start -> C -> K -> Goal
   Path Length: 7
   Number of Iterations: 11

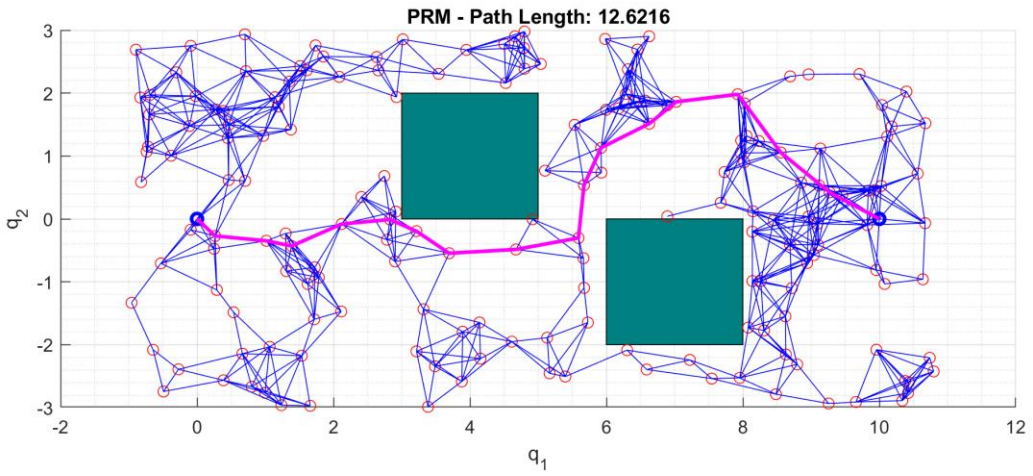
d) A* search implementation performs better since it found the shortest path to goal in less iterations than Dijkstra's. I would choose A* to search for a path in a *large graph* from start to goal assuming a good heuristic is used. I would choose Dijkstra's to search every node in the graph since Dijkstra's encounters every possible path.

e) I would allow the search algorithm to search all neighbors instead of only the children of a given node. I would also have to modify the backpointer for an already visited node if the path cost to that node is less than the path from before.

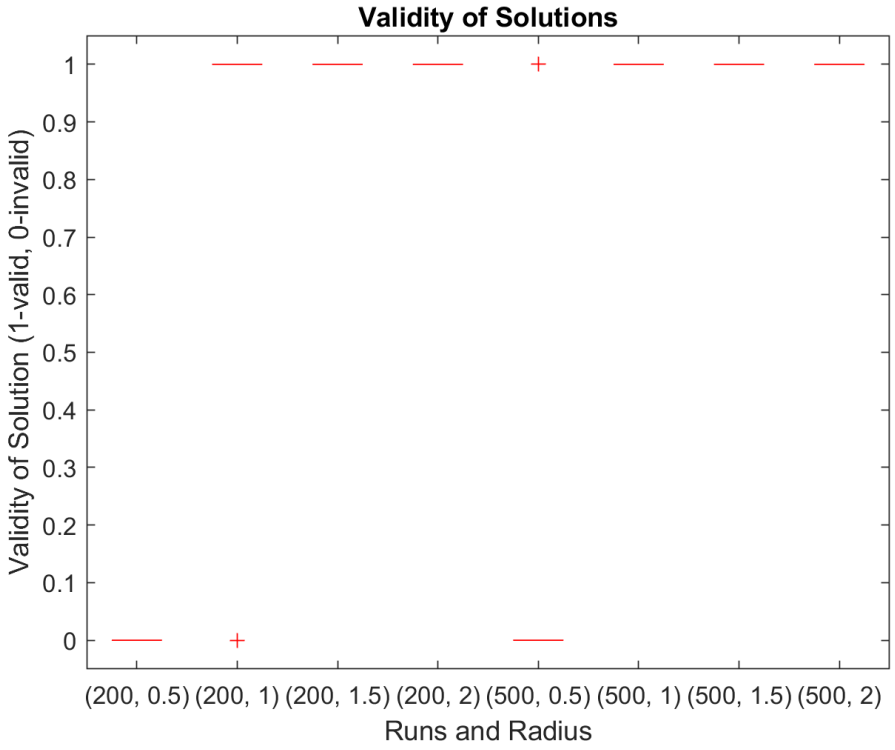
Problem 2:

a)

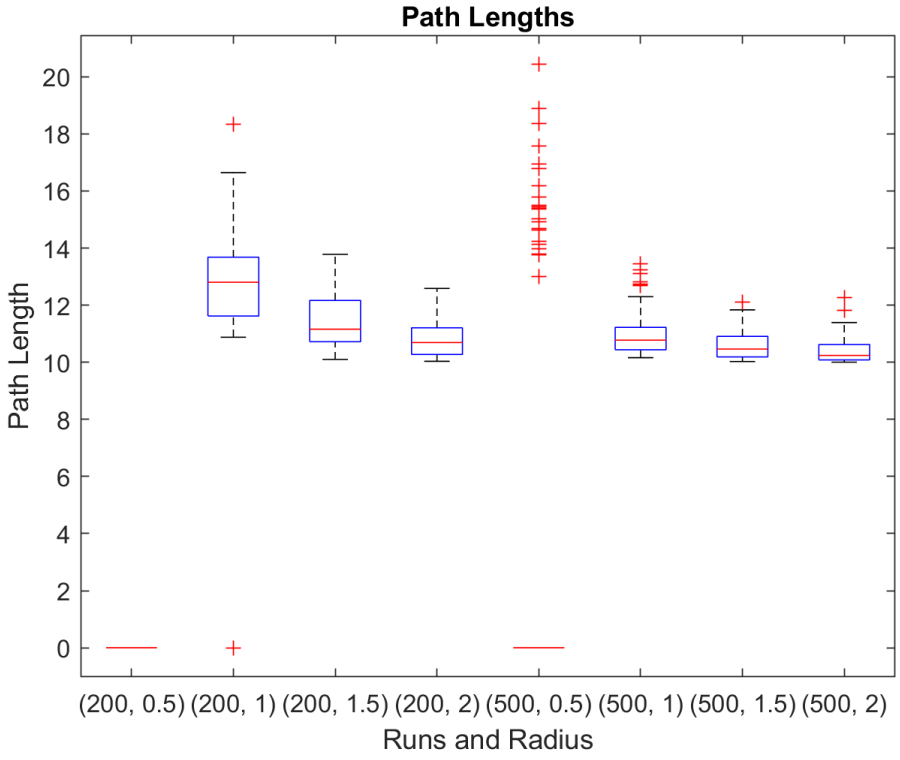
i)

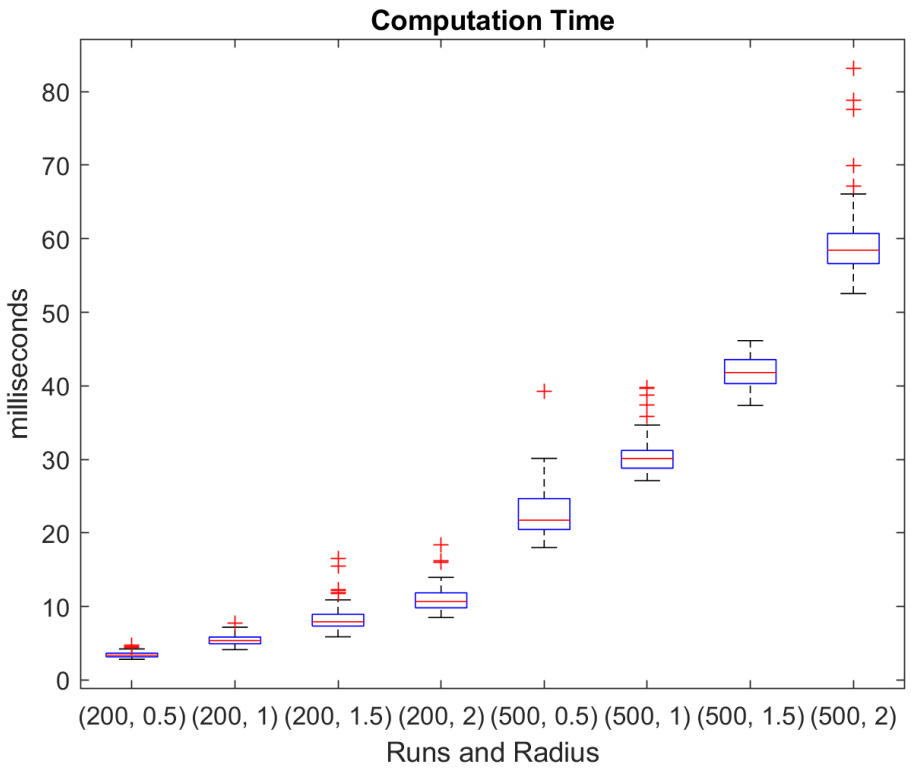


ii)



Note: Path lengths of zero are associated with invalid solutions





iii)

The optimal values for n and r are 200 and 2 respectively. This is true since for all simulations, a valid path was found, the path lengths do not differ too much from other configurations with 100% valid solutions, and the computation times are not the shortest but it's a necessary sacrifice for optimality w.r.t path length.

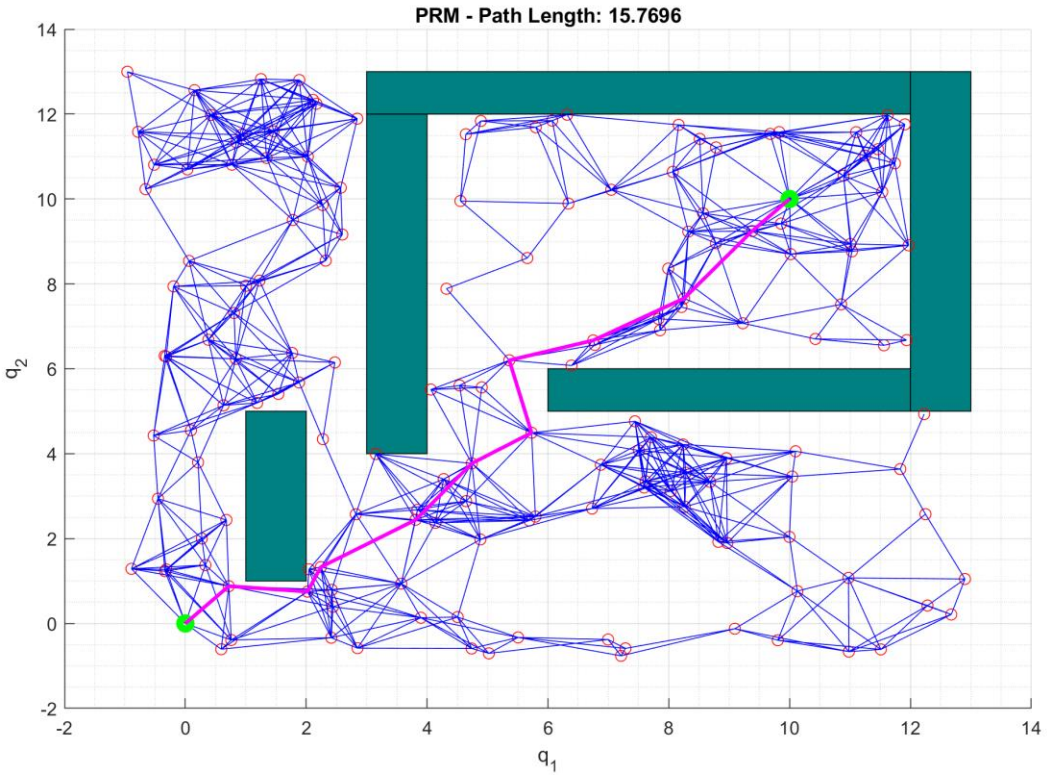
iv)

After path smoothing, the optimal values for n and r are 200 and 1.5 respectively. The validity of the solutions was the same as before smoothing. Now, the path lengths are all shorter and closer to optimal. The computation times did not chance much since path smoothing is fairly inexpensive computationally.

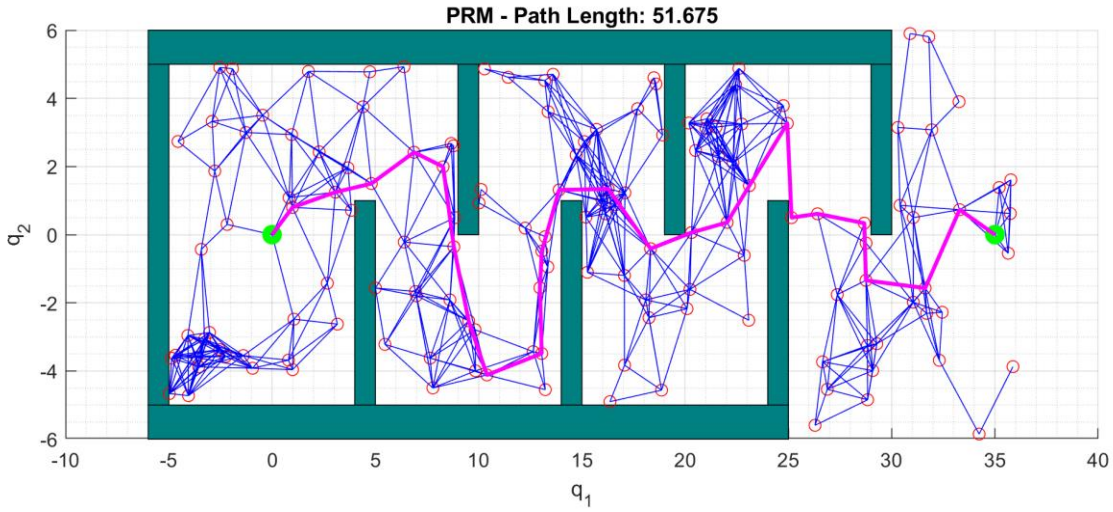
b)

i)

Workspace 1

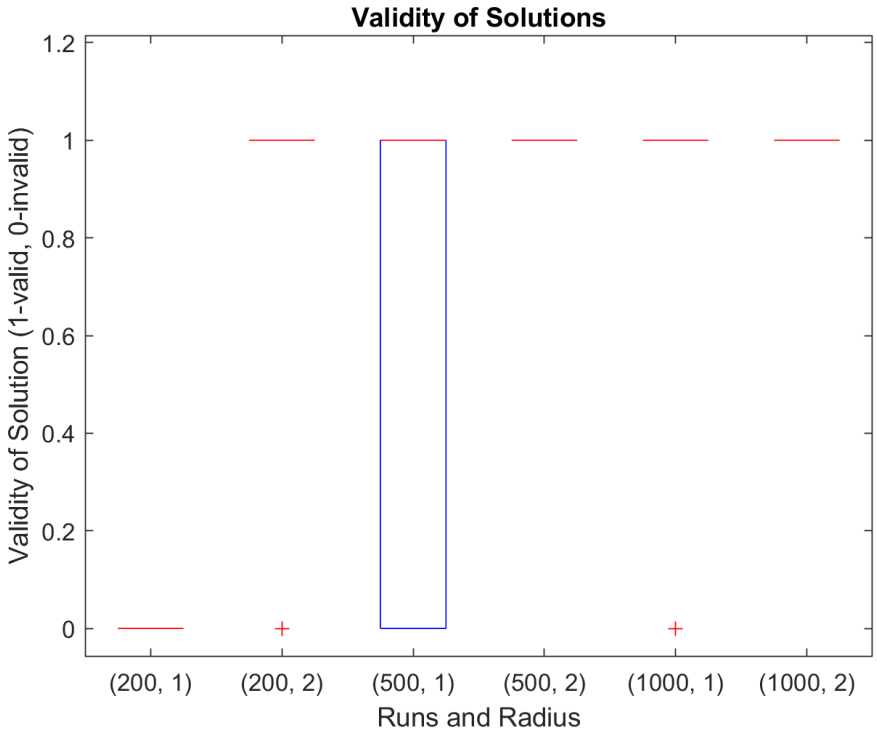


Workspace 2

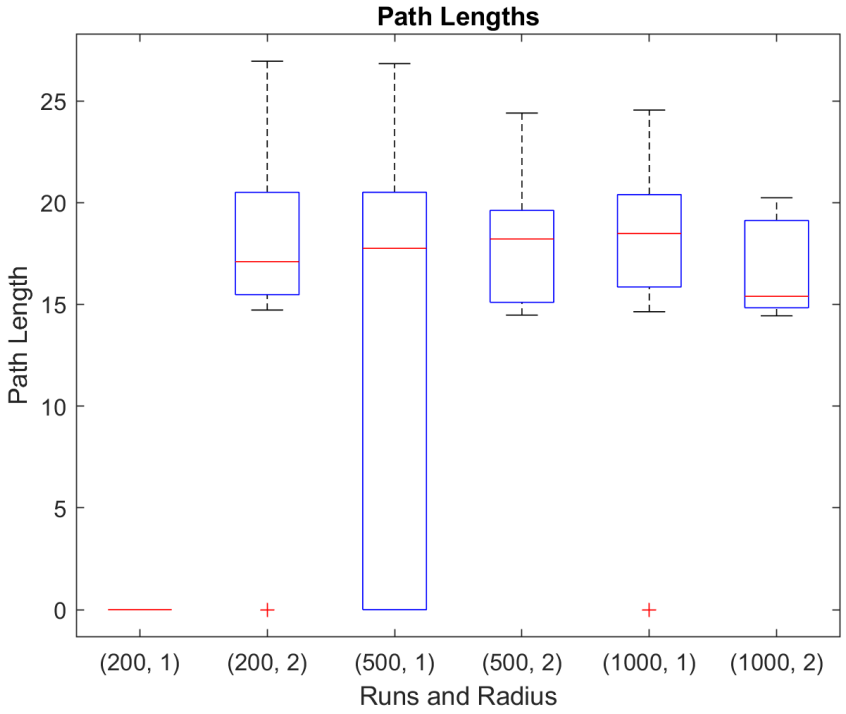


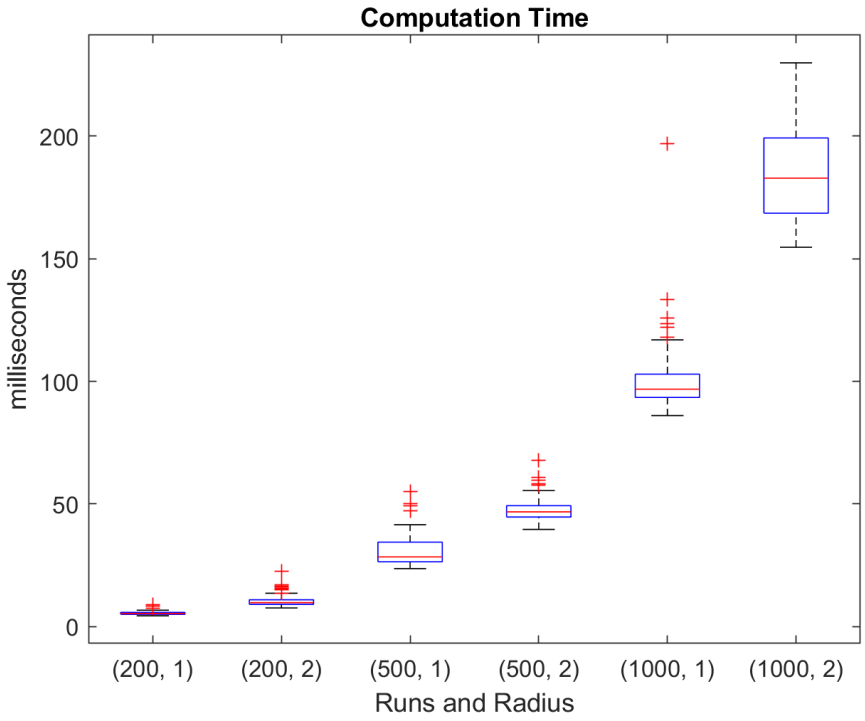
ii)

Workspace 1

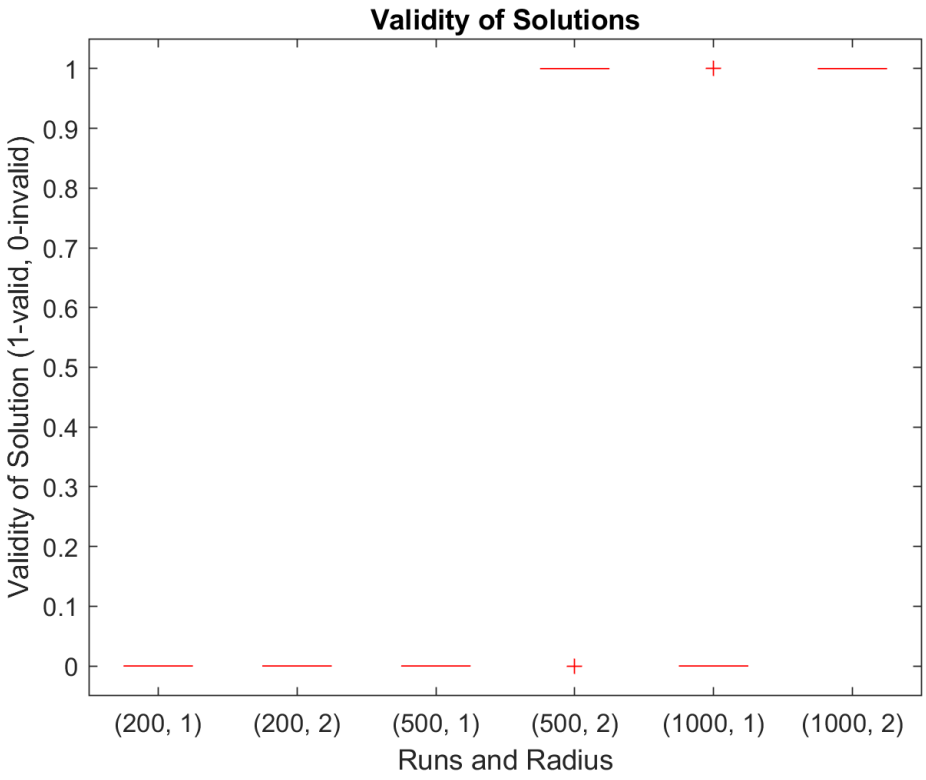


Note: Path lengths of zero are associated with invalid solutions

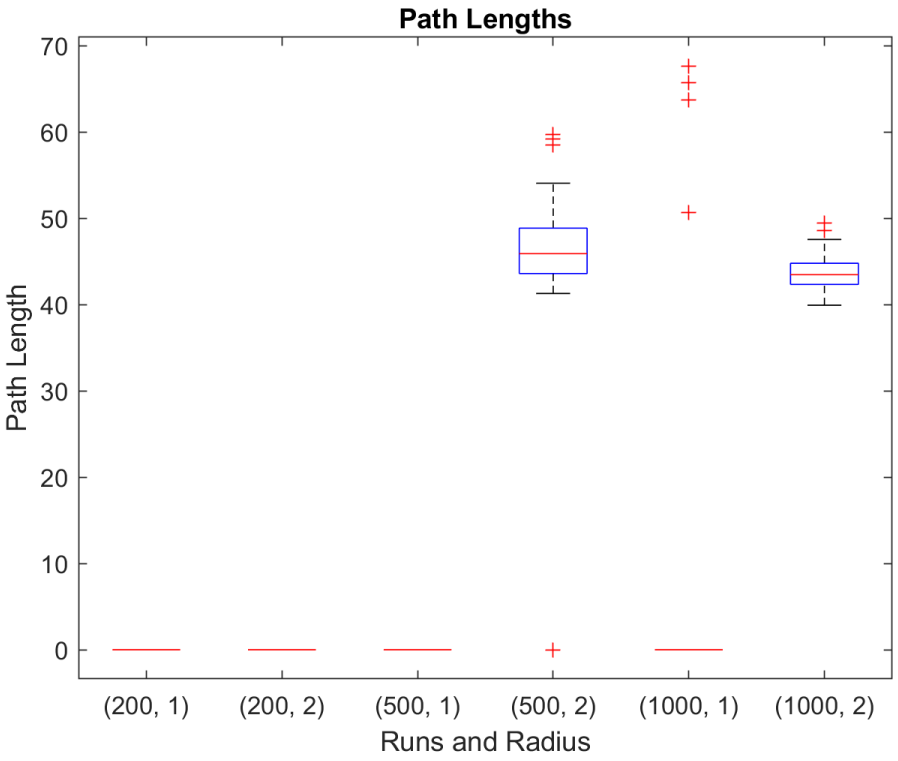

Path Lengths
Path Length
25
20
15
10
5
0
(200, 1)
(200, 2)
(500, 1)
(500, 2)
(1000, 1)
(1000, 2)
Runs and Radius

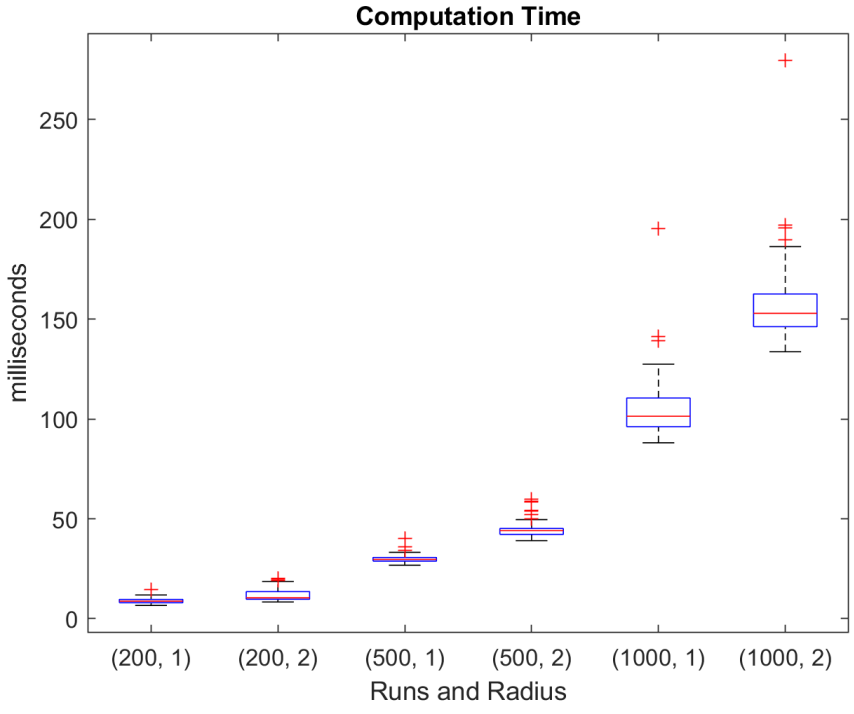

Computation Time
milliseconds
200
150
100
50
0
(200, 1)
(200, 2)
(500, 1)
(500, 2)
(1000, 1)
(1000, 2)
Runs and Radius

Workspace 2



Note: Path lengths of zero are associated with invalid solutions


Path Lengths
Path Length
70
60
50
40
30
20
10
0
(200, 1)
(200, 2)
(500, 1)
(500, 2)
(1000, 1)
(1000, 2)
Runs and Radius


Computation Time
milliseconds
250
200
150
100
50
0
(200, 1)
(200, 2)
(500, 1)
(500, 2)
(1000, 1)
(1000, 2)
Runs and Radius

iii)

Workspace 1:

The optimal values for n and r are 1000 and 2 respectively. This is true since for all simulations, a valid path was found, the path lengths are the least out of all configurations with 100% valid paths and the computation times are the longest but it's a necessary sacrifice for optimality w.r.t path length.

Workspace 2:

The optimal values for n and r are 1000 and 2 respectively. This is true since for all simulations, a valid path was found, the path lengths do not differ much from other configurations, and the computation times are tied for the lowest out of all configurations with 100% valid paths.

iv)

Workspace 1:

After path smoothing, the optimal values for n and r are still 1000 and 2 respectively. The validity of the solutions was the same as before smoothing thus all configurations lead to too many invalid solutions. Now, the path lengths are all shorter and closer to optimal but the optimal configuration is still (1000,2) w.r.t. valid solutions primarily.

Workspace 2:

After path smoothing, the optimal values for n and r are still 1000 and 2 respectively. The validity of the solutions was the same as before smoothing thus all configurations lead to too many invalid solutions. Now, the path lengths are all shorter and closer to optimal but the optimal configuration is still (1000,2) w.r.t. valid solutions primarily.

c)

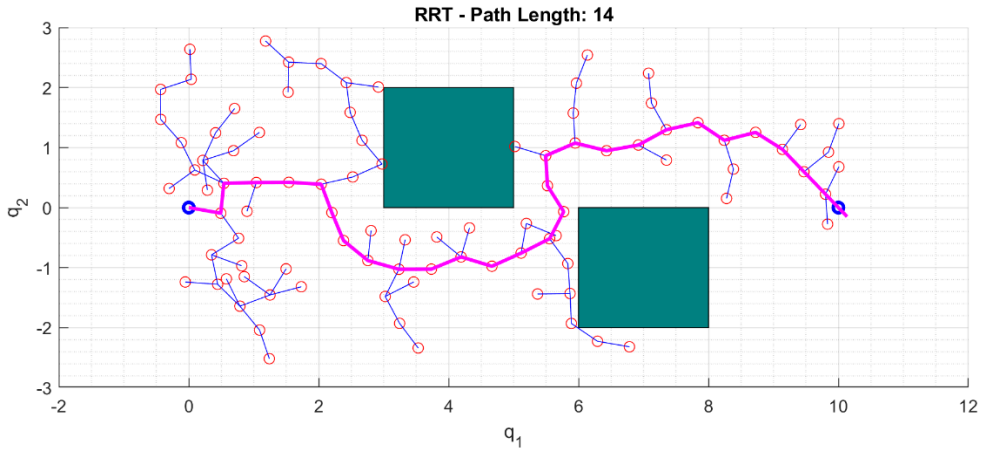
The only thing to look out for in exercise 4 of homework is the c-space obstacles. Those obstacles would need to be split up into smaller obstacles with defined primitives so that collisions can be detected when randomly sampling nodes to add to the PRM.

My PRM assumes a c-space is input and takes care of the rest. In other words as long as the robot is parameterized as a point, a PRM can be generated by randomly sampling the c-space and connected nodes. If a valid path exists, it will find it.

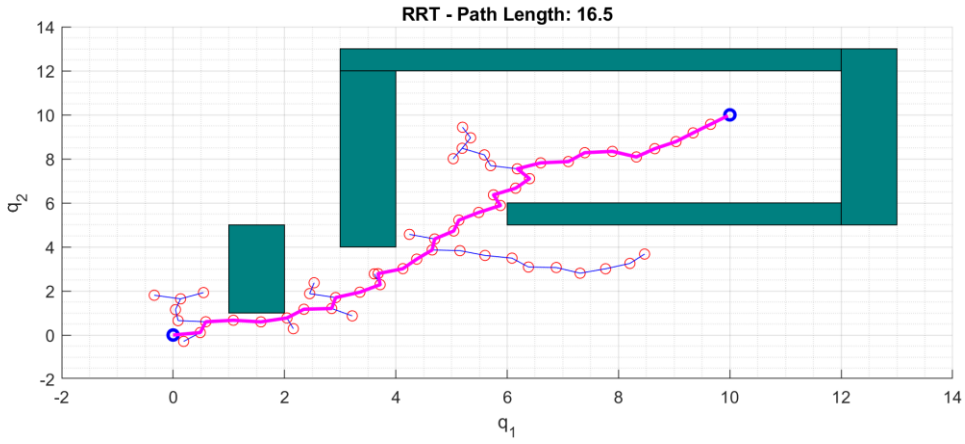
3)

a)

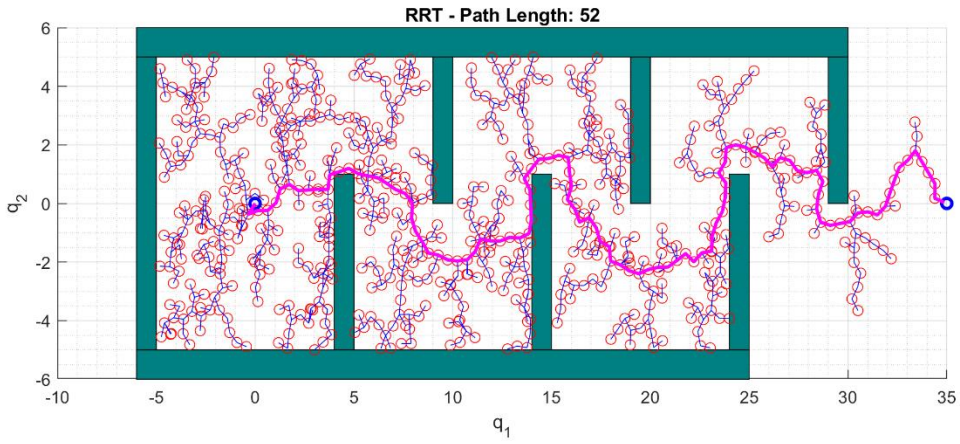
Map 1



Map 2

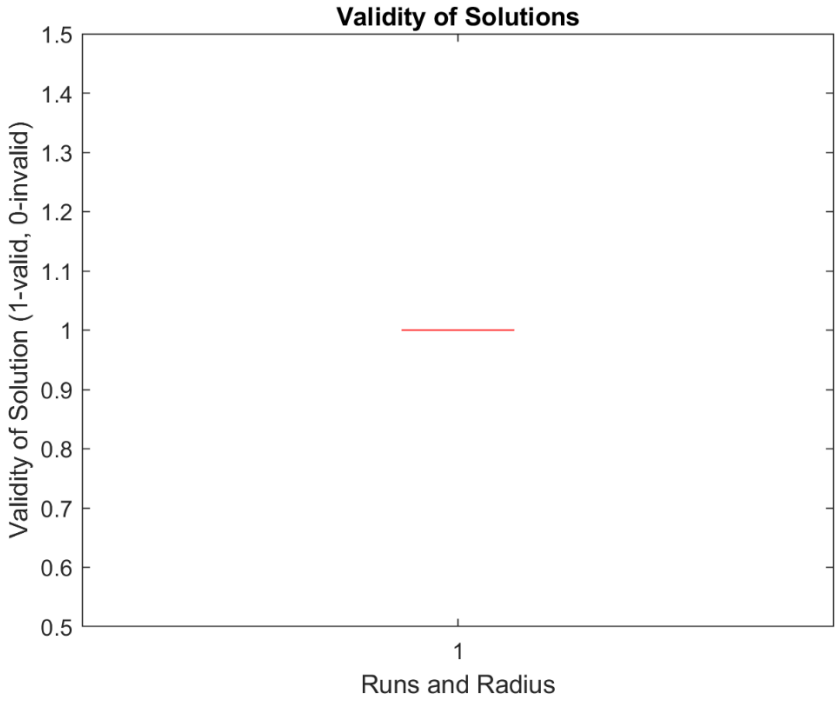


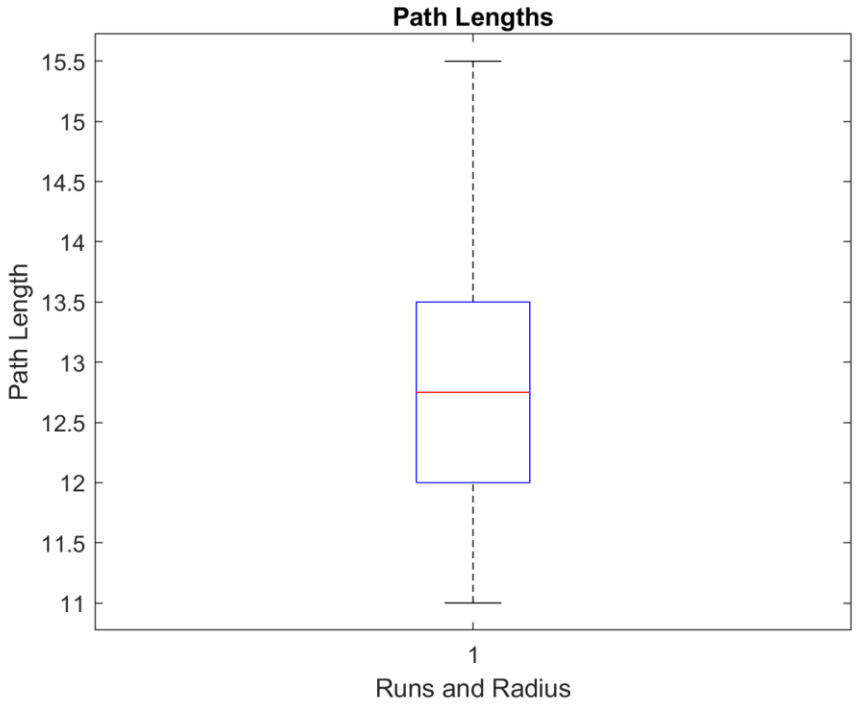
Map 3

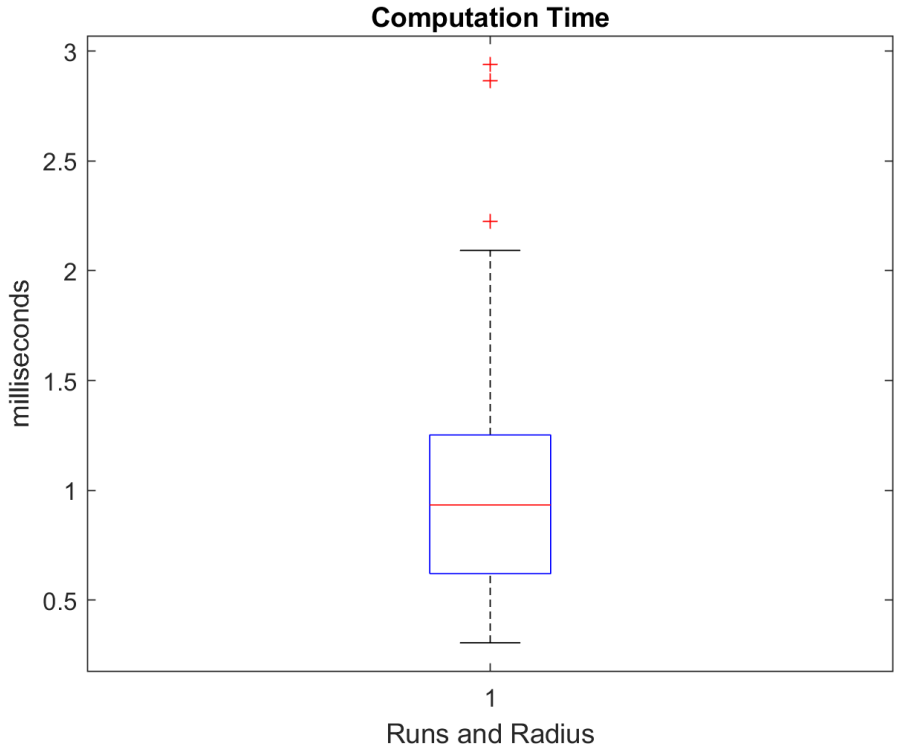

RRT - Path Length: 52
6
4
2
0
-2
-4
-6
-10
-5
0
5
10
15
20
25
30
35
q_2
q_1

b)

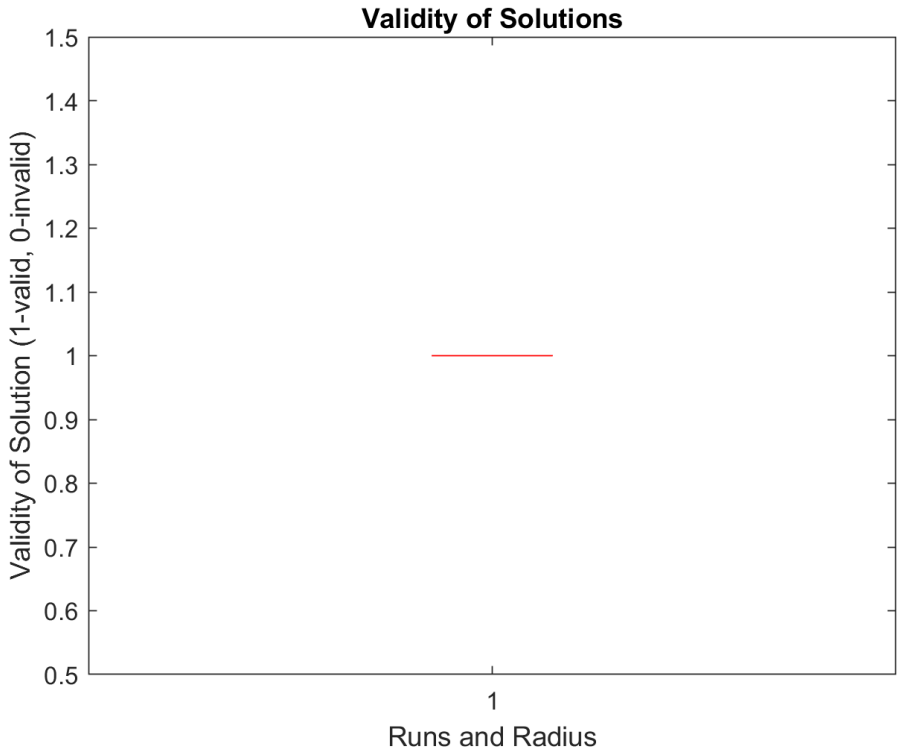
Map 1

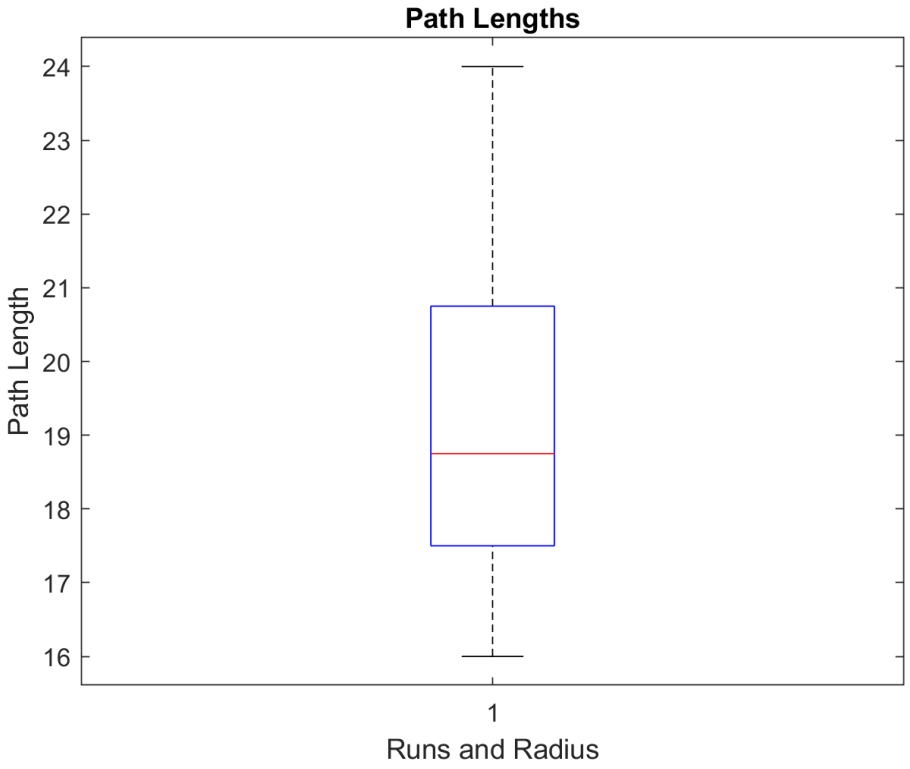

Validity of Solutions
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
Validity of Solution (1-valid, 0-invalid)
1
Runs and Radius

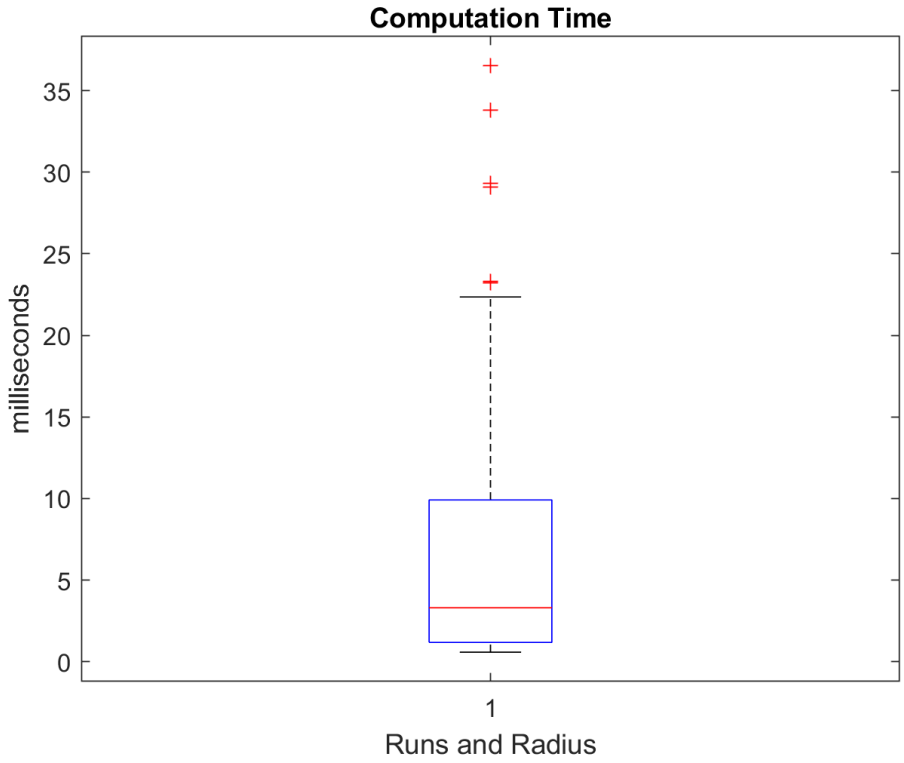

Path Lengths
15.5
15
14.5
14
13.5
13
12.5
12
11.5
11
Path Length
1
Runs and Radius


Computation Time
3
2.5
2
1.5
1
0.5
milliseconds
1
Runs and Radius

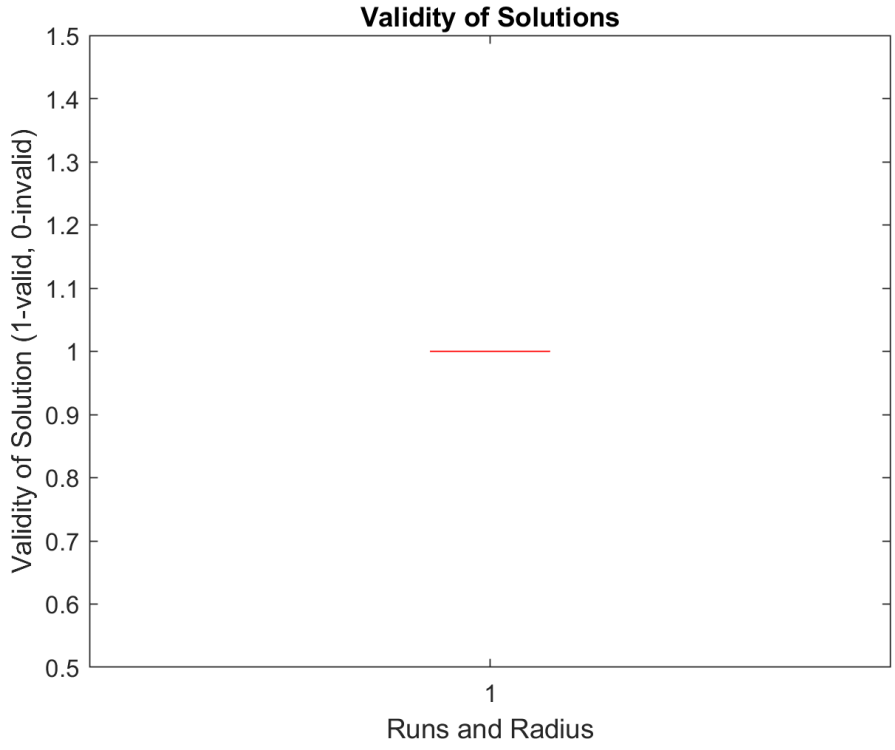
Map 2

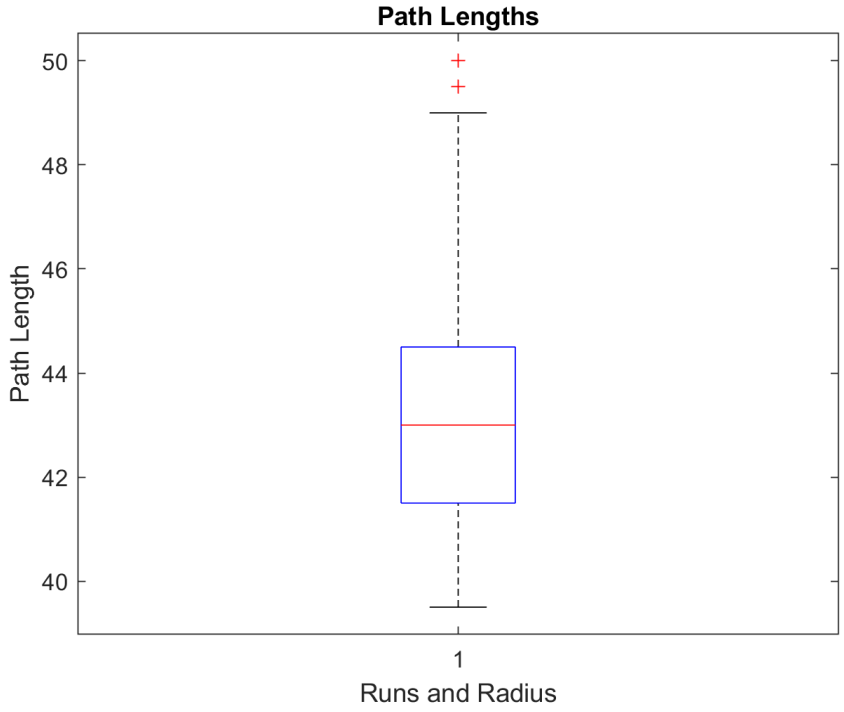

Validity of Solutions
Validity of Solution (1-valid, 0-invalid)
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
1
Runs and Radius

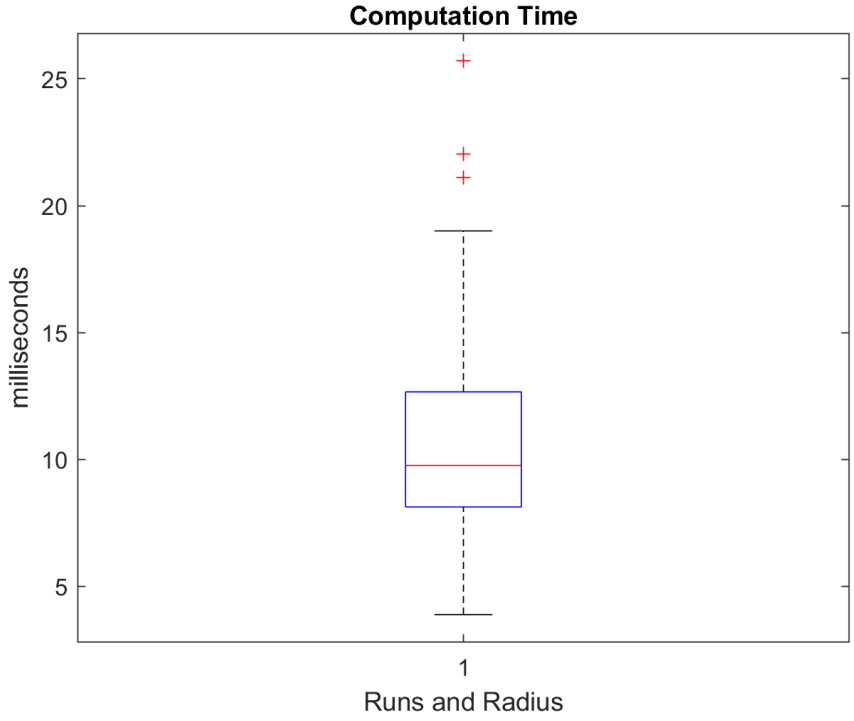

Path Lengths
Path Length
24
23
22
21
20
19
18
17
16
1
Runs and Radius


Computation Time
milliseconds
35
30
25
20
15
10
5
0
1
Runs and Radius

Map 3


Validity of Solutions
Validity of Solution (1-valid, 0-invalid)
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
1
Runs and Radius


Path Lengths
Path Length
50
48
46
44
42
40
1
Runs and Radius


Computation Time
milliseconds
25
20
15
10
5
1
Runs and Radius

c)

The only thing to look out for in exercise 4 of homework is the c-space obstacles. Those obstacles would need to be split up into smaller obstacles with defined primitives so that collisions can be detected when randomly sampling nodes to extend to.

4)

The gradient descent algorithm with a potential function was very tricky to tune. The path often devolved into local minimums due to the cancelling out of attractive and repulsive forces in all directions. The gains also had to be tuned properly to “push” the robot carefully around the obstacles to the goal.

The wave front planner was much easier to implement than the attractive/repulsive potential function approach to gradient descent. The discretization was simple. There was always only one local minimum which was the goal. However the algorithm is only resolution complete so the discretization had to be fine enough to find a path. This algorithm also performs poorly for large maps because it has to search the entire space. Grid based discretization planning takes a lot more time compared to sampling based methods.

The PRM algorithm was difficult to implement in terms of creating the graph in c++ and keeping track of all the pointers. However PRM works really quickly as it is a sampled based planning technique. However this algorithm, like gradient descent with potential functions, doesn’t guarantee completeness. If the PRM is not connected, more samples must be taken to connect more nodes to the graph and reach the goal. Thus this method is probabilistically complete.

GoalBias RRT is also a computationally efficient planning method like PRM. Instead of sampling random samples all the time like PRM, GoalBias RRT occasionally reaches out towards goal thus decreasing computation time and not having to search as much space.