Homework 1 – Minimum Spanning Tree

Submitted in partial fulfilment of the requirements for the course of

CMSC818B – Decision Making for Robotics

By

Adheesh Chatterjee

Course Faculty

Prof. Pratap Tokekar

A. JAMES CLARK
SCHOOL OF ENGINEERING

INTRODUCTION

The aim of this project is to implement the Minimum Spanning Tree (MST) based 2 approximation algorithm for solving the metric Travelling Salesman Problem. The tour length is to further improved using a heuristic. The algorithms is tested on eil51,eil76 and eil101 input instances. The algorithm is also tested on 10 randomly generated instances of 100,200 and 300 vertices each. For problem 2, an algorithm for solving k robot metric TSP is designed. The pseudo code for the algorithm is written and the performance is analyzed.

Minimum Spanning Tree

A minimum spanning tree is a spanning tree such that all vertices in a graph are connected with weight less than or equal to the weight of every other such spanning tree. Here the weight of the spanning tree is defined as the sum of the weights given to the edges of the graph

So to create the MST for the problem given to us, we first check the input data

Node List

I use the basic file handling functions of python to open and read the .tsp file.

It is observed that points are given as 3 columns of node number, x coordinate, y coordinate. So I separate out the x-coordinate and y-coordinate into a list.

Edge List

Now since we have a list of xy coordinates, we use the euclidean distance to calculate the weight of the edges and create a final list which has 3 columns of x coordinate, y coordinate and the distance between the edges.

One thing to note, here since the distance between node1 and node2 is the same as the distance between node2 and node1 and we're dealing with undirected graphs, only the node pairs in ascending order are considered. I.e. only the edge (1,2) will be considered and not edge (2,1).

So the final list will only have a n*(n-1)/2 elements, where n is the number of nodes.

MST

Now that we have our nodelist and edgelist, we can proceed with the MST. Now since the MST has to be open loop with minimum weight, the following procedure is implemented.

First, the initial edge and weight are considered as minimum. Then, the list is traversed and any weight less than the initial is checked. If it exists, then the initial edge and weight is replaced by the one found and this procedure is repeated.

Once, the true minimum weight and its corresponding edge is found, the edge is popped out of the edgelist and appended to a new list called visited. So all the edges are sorted out by weight into the visited list.

Here it is observed, that if any 3 pair combinations of edges occurs, then it forms a closed loop. I.e. in the sorted list, if (0,4), (0,2) and (2,4) occur, then the three edges will form a closed loop. So since it is sorted by weight, the third occuring edge will be removed. So we end up with a list of n-1 edges where n is the number of nodes.

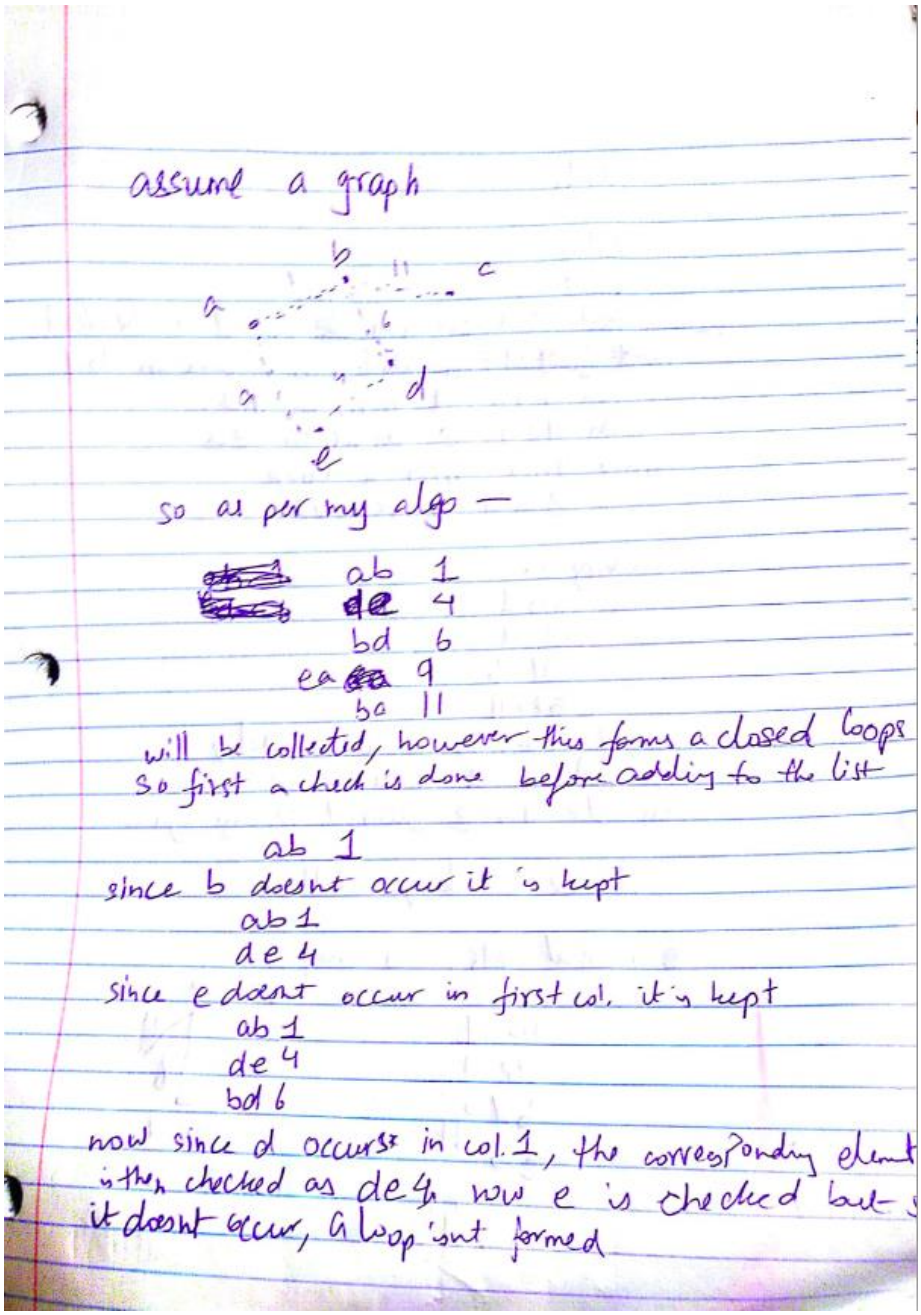
This will always be true and we can plot the MST for any given set of vertices. However, since identifying the 3 pair combinations is very time complex O(n^3), this algorithm

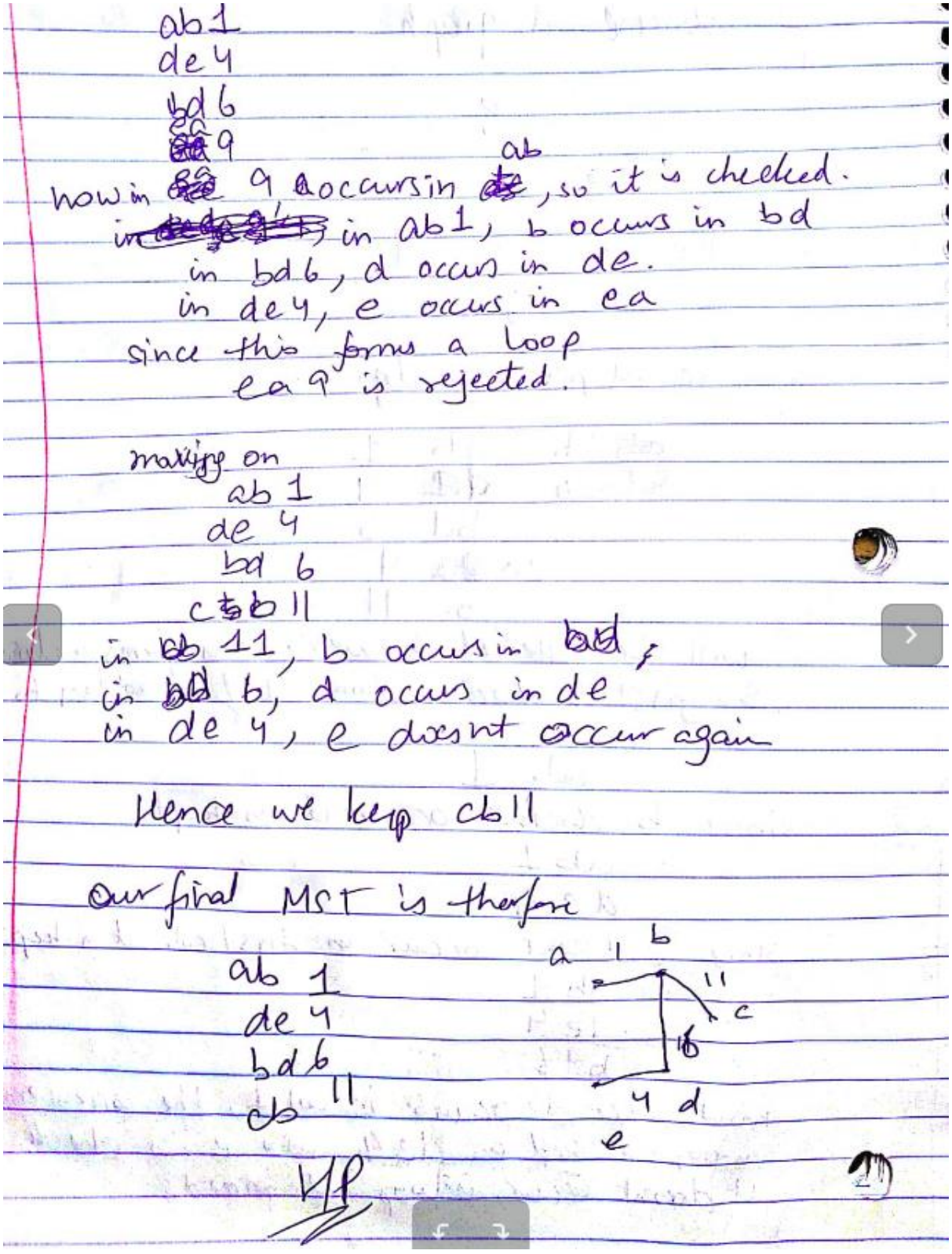
doesn't work well on a huge number of points. Upto 25-30 points, it can solve it within a 10-20 seconds, but beyond that, depending on how spread out and distant the points are, this algorithm may prove to be not that optimal.

As discussed with the professor, the method used by me above, causes the MST for the input files given, to be correct but not in optimal time.

The problem faced is with the checking if the loop closes due to any edge being joined. To solve this the the professor also suggested another method for finding MST in optimal time.

This is explained in the image below -





However, on implementing the algorithm suggested, it was observed that the complexity is exponential in this case as well. The recursion and checking every possible path to make sure no closed loops exist, increase the time taken.

This algorithm was much more complex and led to the same time constraint issue as the one previously proposed

The only possible method of solving the complexity, is the Union set concept as discussed with the professor. However that has not been implemented.

Heuristic Used

Please see file *init.py*

I use a heuristic to reduce the total cost of the problem. I select any vertex at random, and search for the next closest vertex. Once found, I use that vertex as the root vertex and find the next closest vertex. This process is repeated until all vertices are used up. Since choosing the initial vertex may also change the total tour cost, I used a loop to select a different starting root node every time.

Then the tour with the min cost is selected as the optimal tour. The length of the tour is calculated by traversing the distance backwards from the final node to the initial node.

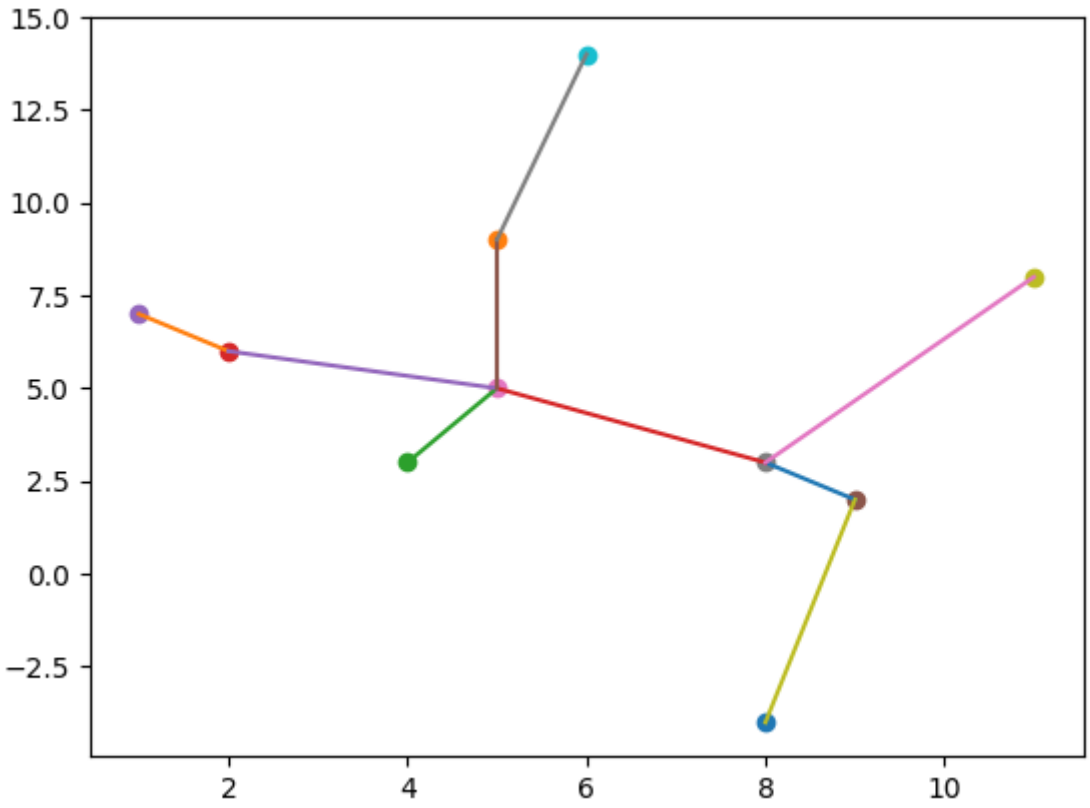
The tour is finally graphed and annotated.

Random Points

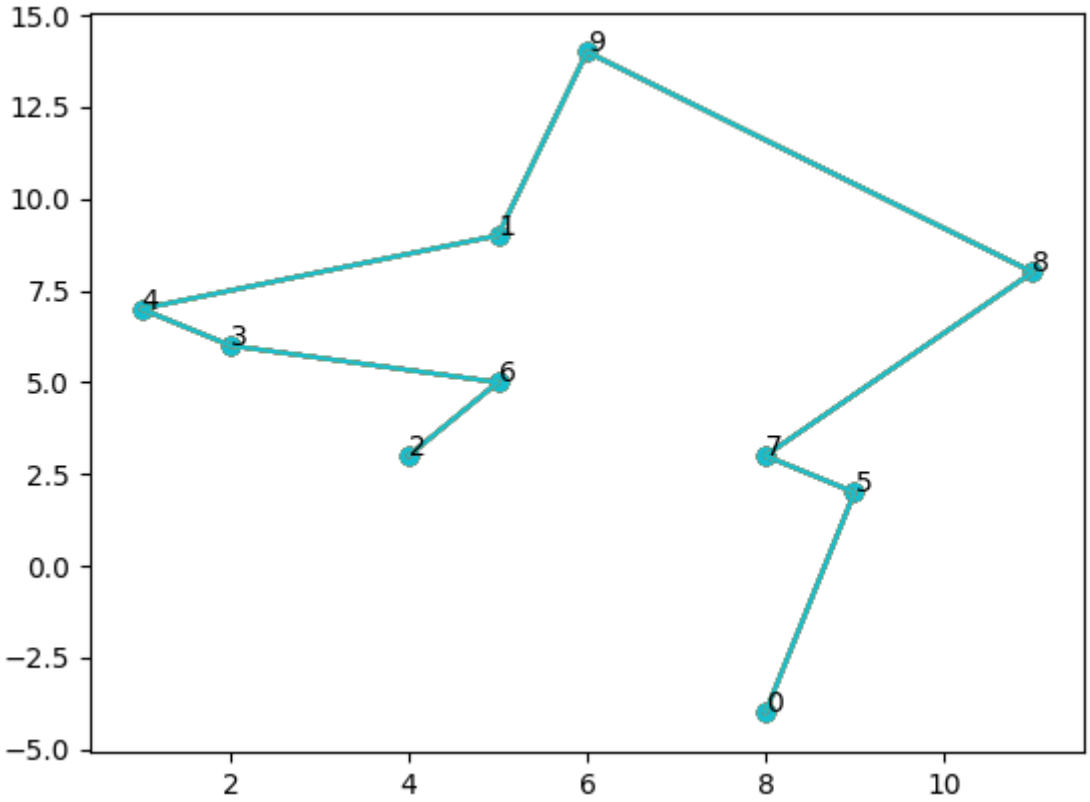
Random points are generated using np.random.rand(n,2) where n is the number of points reqd. This gives us a 2D array of points between 0 and 1. On multiplying the entire array by 100, we get the reqd points on which MST is to be implemented. The final tour is also found.

RESULTS

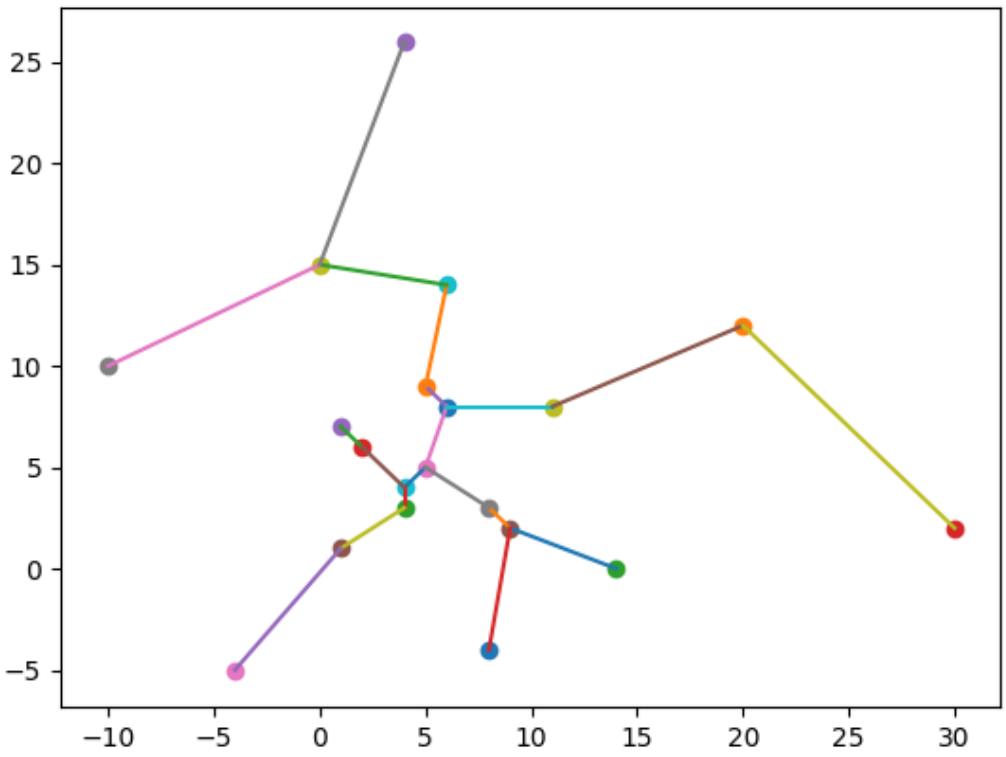
File Name	Length of FInal Tour	Optimal Tour Length	Length of MST	Time Taken to solve (tour only)	Time taken to solve MST
10 points	42	-	34	negligible	0:00:00.268279
20 points	145	-	127	negligible	0:00:03.901221
25 points	213	-	153	negligible	0:01:12.171757
eil51.tsp	489	426	376	0:00:00.008347	0:01:13.942919
eil76.tsp	598	538	472	0:00:00.024770	-
eil101.tsp	750	629	562	0:00:00.055821	-



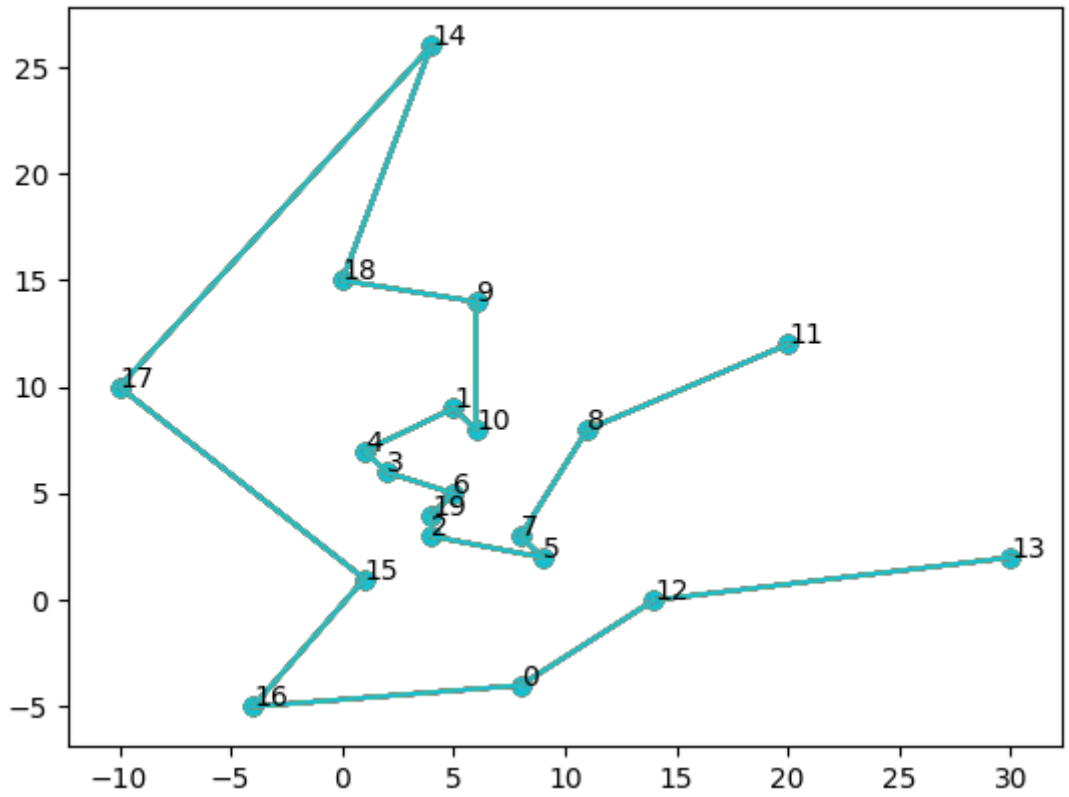
MST for 10 points



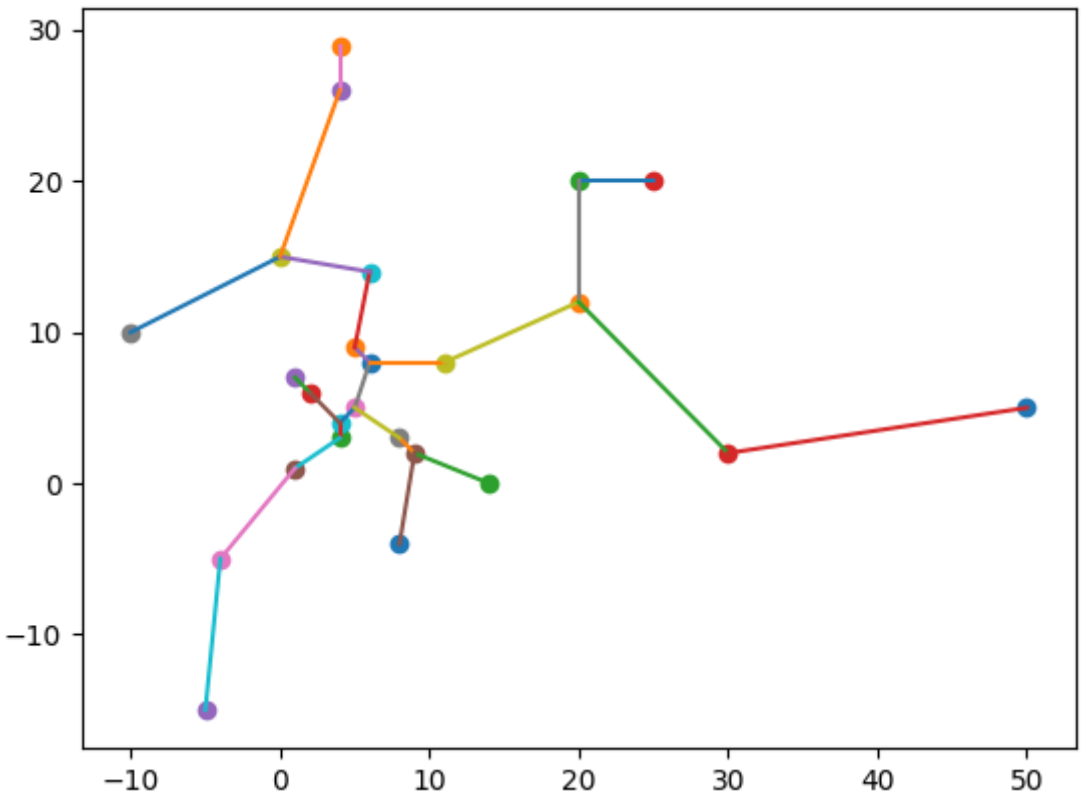
Tour for 10 points



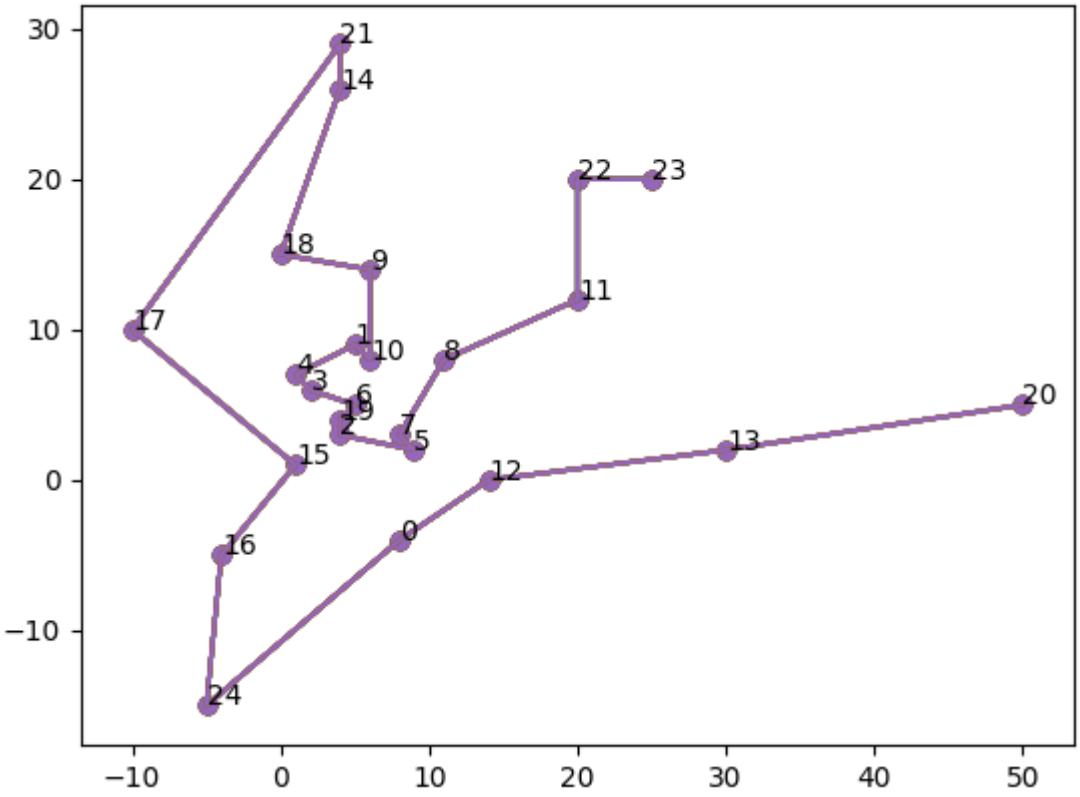
MST for 20 points



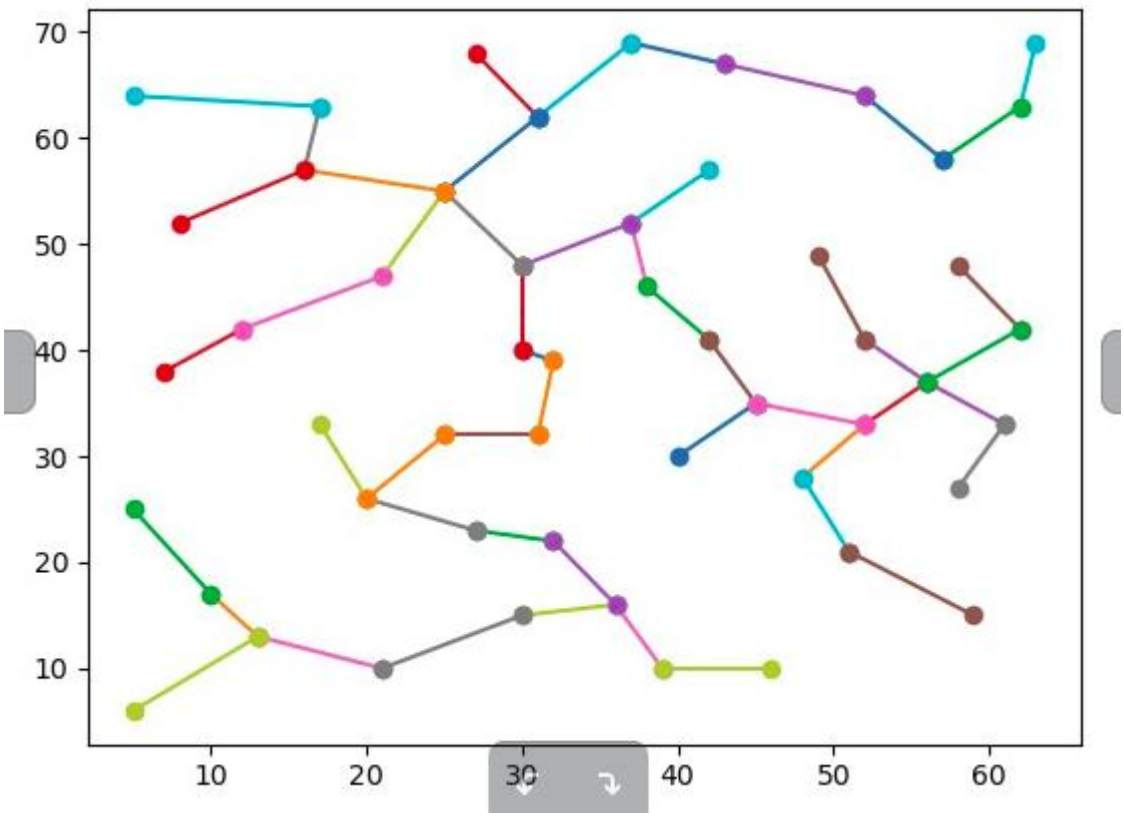
Tour for 20 points



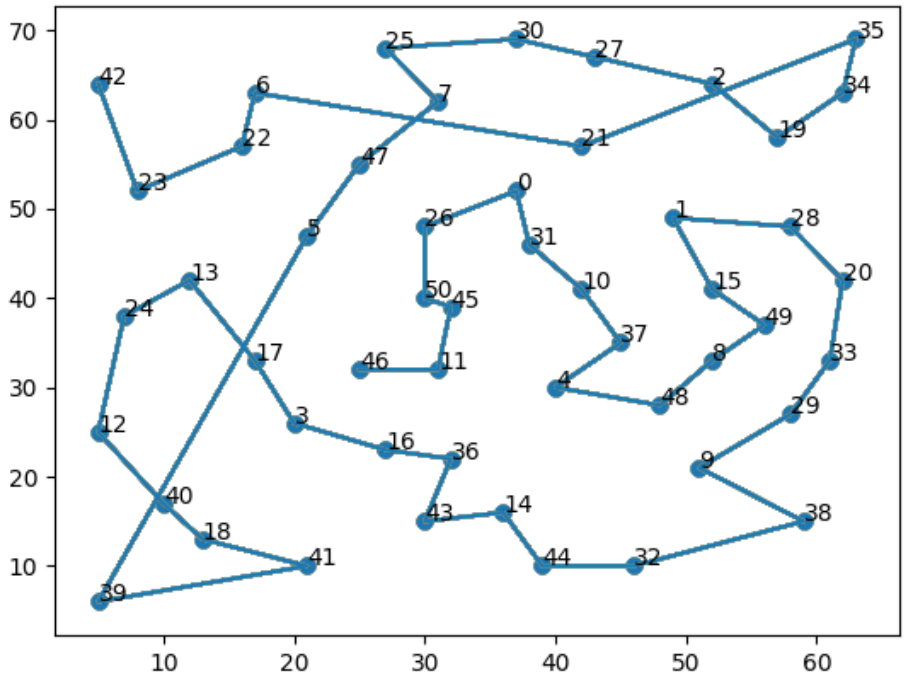
MST for 25 points



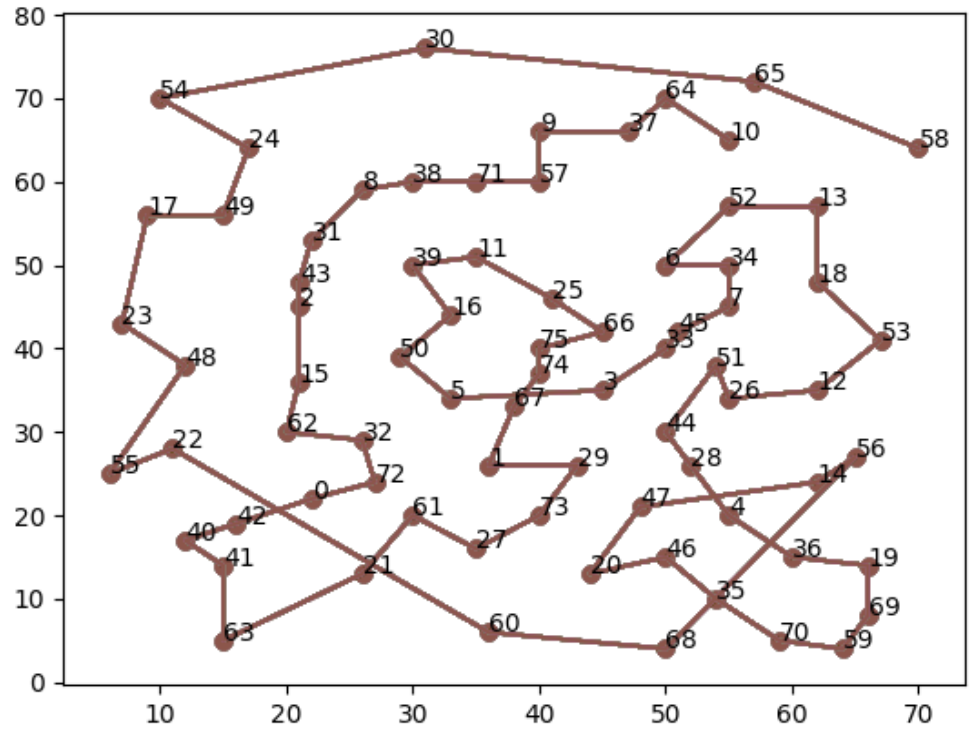
Tour for 25 points



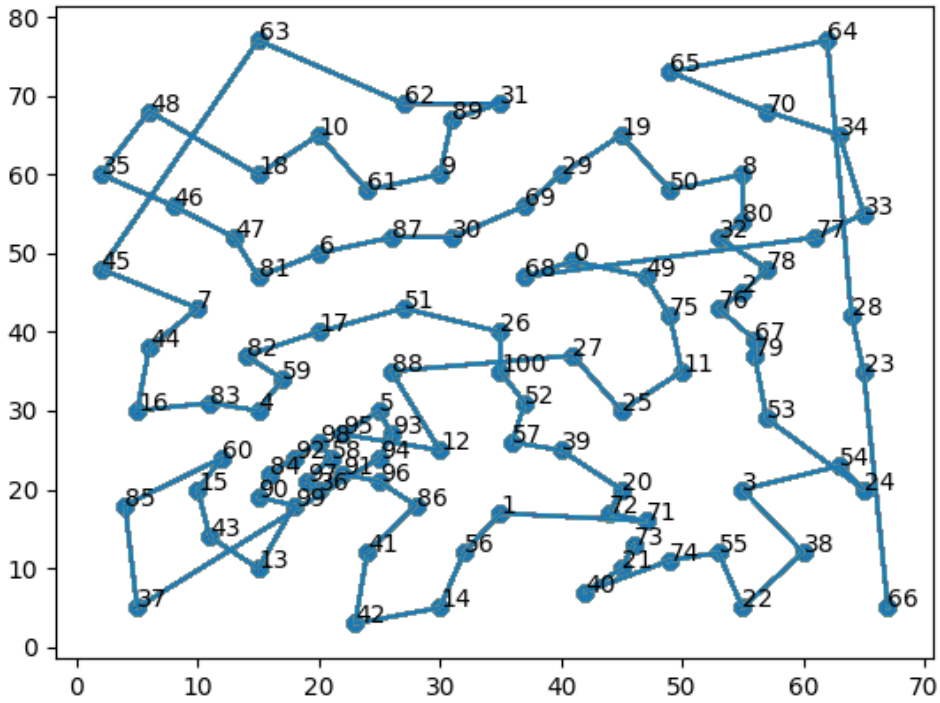
MST for eil51.tsp



Tour for eil51.tsp



Tour for eil76.tsp

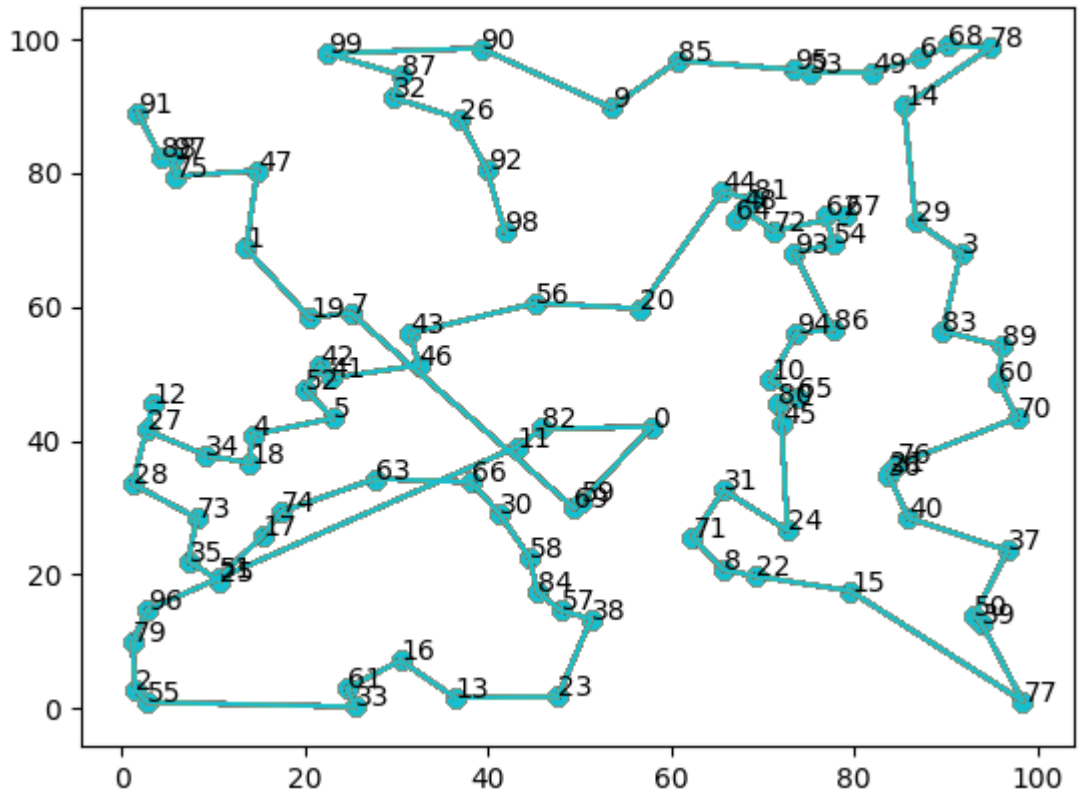


Tour for eil101.tsp

100 vertices

Length of FInal Tour	Length of MST	Time Taken to solve
819	625	0:00:00.065279
814	673	0:00:00.064800
860	702	0:00:00.069824
760	521	0:00:00.065590
838	628	0:00:00.065694
748	598	0:00:00.085190
777	572	0:00:00.066849
827	645	0:00:00.064801
779	538	0:00:00.063807

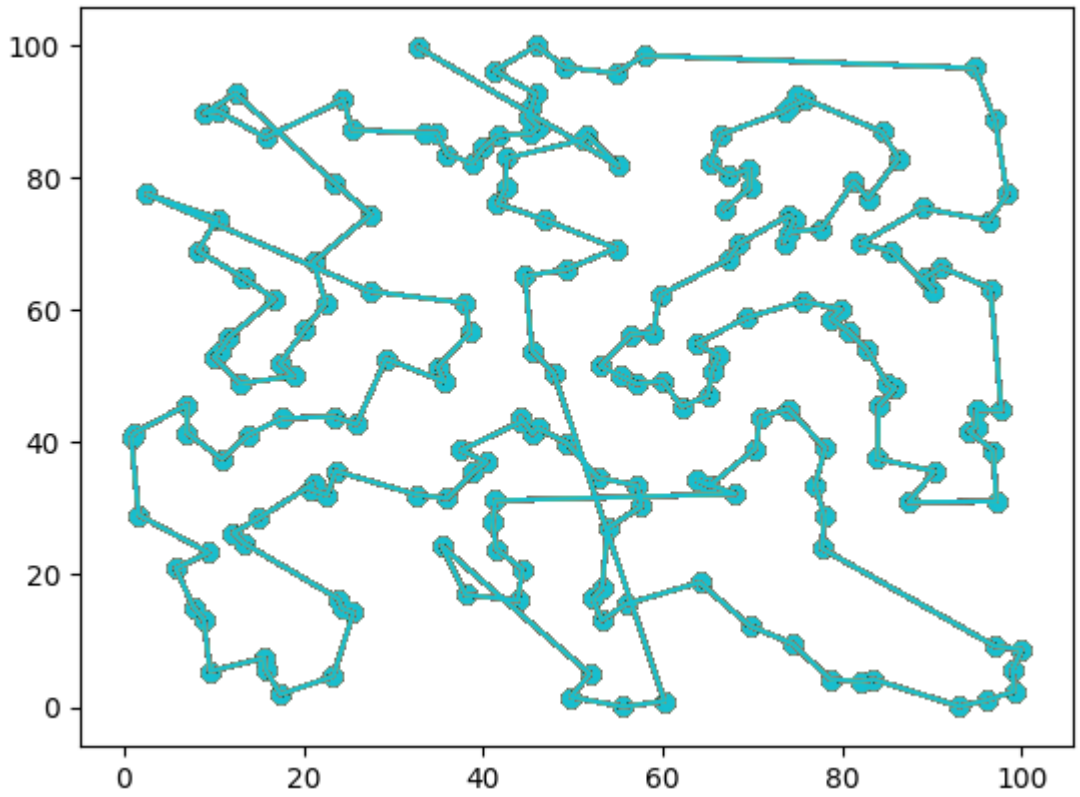
873	628	0:00:00.065238



200 vertices

Length of FInal Tour	Length of MST	Time Taken to solve
1161	935	0:00:00.563393
1164	958	0:00:00.598888
1157	998	0:00:00.553228
1211	1035	0:00:00.542720
1178	1032	0:00:00.526524
1158	972	0:00:00.552397
1129	956	0:00:00.529127

1146	996	0:00:00.588369
1105	942	0:00:00.583878
1136	963	0:00:00.565552

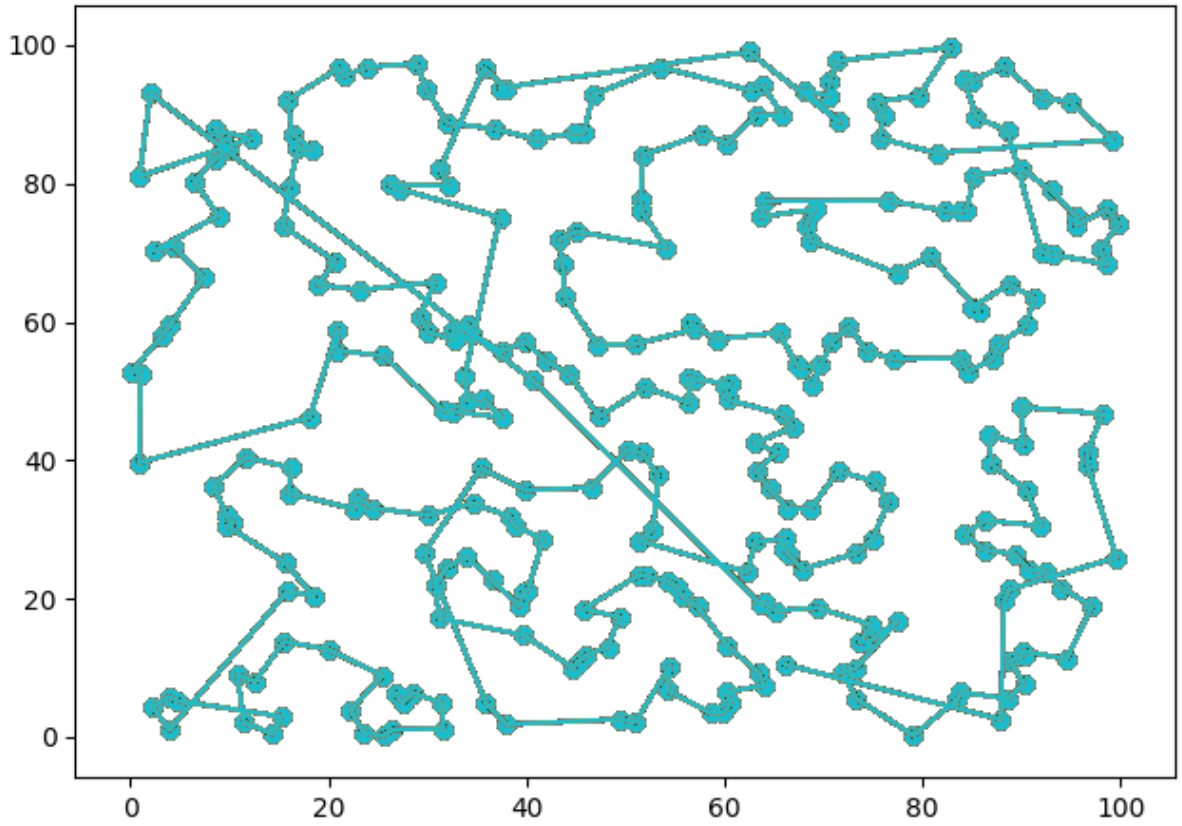


Tour for 200 vertices

300 vertices

Length of FInal Tour	**Length of MST**	**Time Taken to solve**
1492	1274	0:00:02.764483
1383	1230	0:00:02.797517
1307	1187	0:00:02.706242
1362	1146	0:00:02.750444
1340	1169	0:00:02.817431
1514	1387	0:00:02.928229
1407	1198	0:00:02.766036

1368	1083	0:00:02.788192
1422	1027	0:00:02.760441
1372	1006	0:00:02.854447



Tour for 300 vertices

PROBLEM 2
K ROBOTS

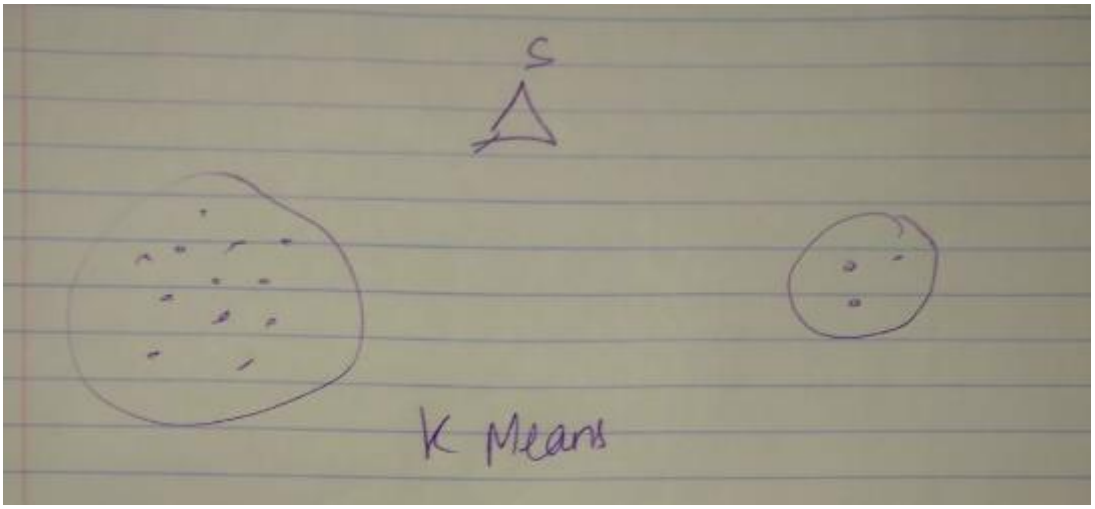
I had 3 ideas on how to implement K Robots metric

We know that the input is a complete metric graph, which has a starting vertex s as well as a an integer k>1 for the robots.

One method to do it is using **K Means Clustering**.

Let us assume N vertices are distributed randomly and we have k robots. We can use K means clustering to form K clusters among these N vertices with complexity as O(n^(2k+1)).

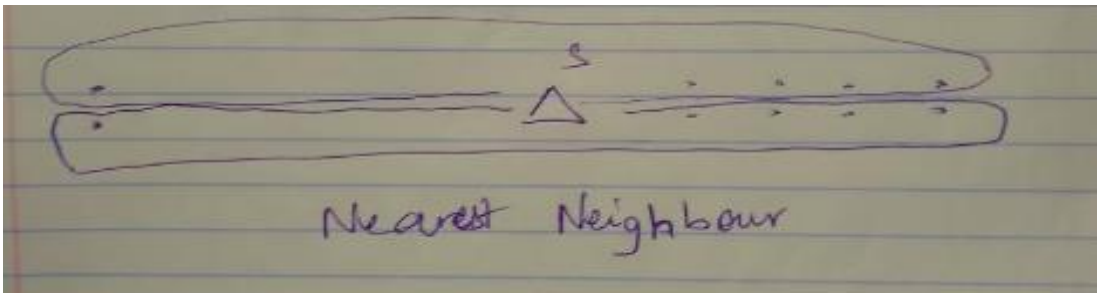
However, this method will fail if one area has more vertices and the other has less, as 2 clusters will be formed but one robot will take longer than the other and the other will be idle after completing the smaller cluster



Second method to do it is using **Nearest Neighbour**

Again, we have N vertices distributed randomly and we have k robots. We essentially find k nearest vertices from the starting point and move the 2 robots to that vertex. Again the nearest vertex is found for each robot from the current position and the robot is moved. This continues until all points are explored. In case, two points are equally close, it can be randomly assigned to any one robot.

However, this method fails as shown below -



2 robots will always move to the 2 closest nodes first and continue and then go back all the way around to the far away nodes.

Third method to do it is using **MST Clustering**

This is my proposed method to solve the K Robots Metric problem. Let us assume you have N vertices randomly distributed. K clusters will again be formed here. However, unlike the K means where the cost function is determined by the centroid to the outermost point in the cluster, here the cost will be determined by the size of the MST in the cluster. I.e an MST will be made for k clusters and points will be exchanged between the clusters until the MST of all the clusters are minimized.
This will form k optimal clusters with minimum MSTs and hence lead to minimum tour lengths.
It is very similar to the Radial Basis Function(RBF function) when graphed, however works on the basis of the MST