Metric TSP Problem

Problem Formulation

- Fully connected graph
- Each node is connected to all other nodes
- The edge weight is given by Euclidean distance between the vertices that form the edge
- Triangular inequality holds

Steps to solve the TSP problem (2 – Approximation algorithm)

1. Find MST
2. Depth first method to travel through nodes to form a tour

Finding MST in Graph (Kruskal's Algorithm)

Step 1	Initialize empty set $T = \{\emptyset\}$
Step 2	For each node in list of vertices Make SET
Step 3	Arrange the edges by length in ascending order
Step 4	For each edge in the list: If the nodes are present in different sets: Join the sets Add the edge to set T Else: Discard the edge
Step 5	Form Tree from the edges in set T

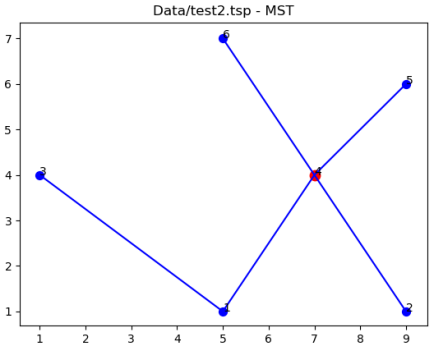
The algorithm is based on 'Disjoint Sets' data structure. This data structure is used to find if there is any cycles in a graph.

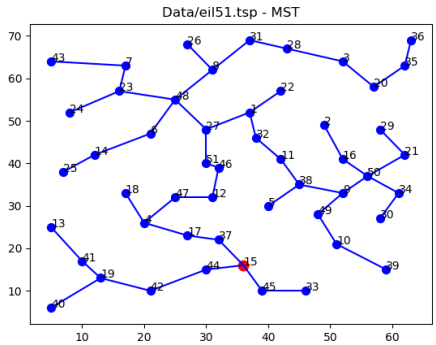
The complexity if the algorithm is given by

$$O(E \log(V))$$

Where,

$$E - \quad Number\ of\ edges$$
$$V - \quad Number\ of\ Vertices$$





In the above figure, Red node is the root node of MST

Depth First Traversal

The MST is collection of edges that connects all the vertices without any cycles in minimum possible edge length.

Step 1	Find root node
Step 2	Get all children for the node
Step 3	Select one child node that is not added to tour
Step 4	If the is leaf node: Go to Step 5 Else: Go to Step 2
Step 5	If all nodes are visited: Terminate Else: Go to parent node

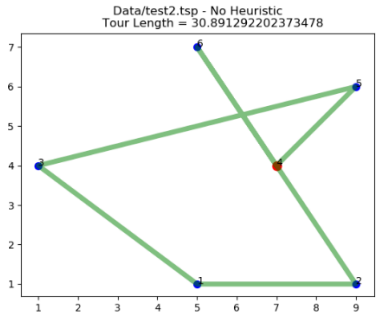
Figure 1: Depth First Traversal

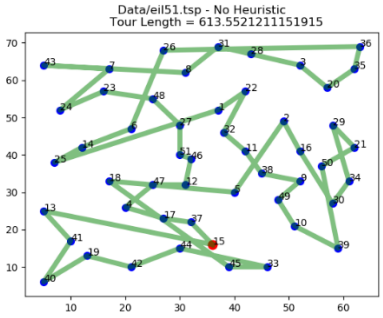
To keep track of the visited nodes and nodes to be visited, a stack is maintained. To the stack, we add the child nodes and visit these nodes in last in first out fashion.

It should be noted that the tree is traversed form parent to child nodes are added to tour but when we traverse from child to parent, nodes are not added to the tour. This form a shortcut from leaf node to next node. The length of this edge will be less than the sum of edge lengths if travelled backwards. This property holds when the graph is fully connected, and the edge weights are Euclidian distances. i.e. triangle inequality holds.

$$L_{AB} + L_{BC} + L_{CD} \leq L_{AD}$$

This approach ensures that the tour will be at most $2 \times L(opt)$





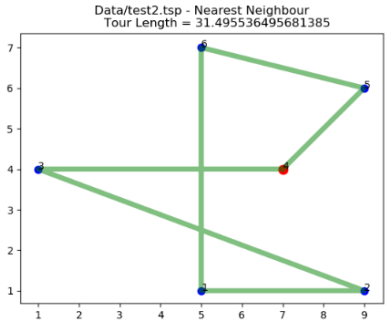
Heuristics on Depth first traversal

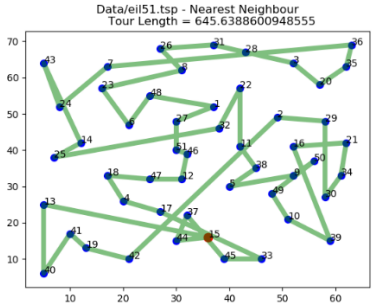
From the above figures it is very evident that there are lot of cross over in the tour path. These can be reduced to some extent by making few changes in traversing the MST. Based on certain heuristics, the nodes are popped from the stack which holds the nodes to be visited

Nearest Neighbor First

In this method, the stack is considered as a priority stack and the priority is decided on the distance between the current node and node to be visited. The highest priority goes to the node with least distance i.e. nearest node from current node

This method need not reduce the tour length and instead it often increases the tour length

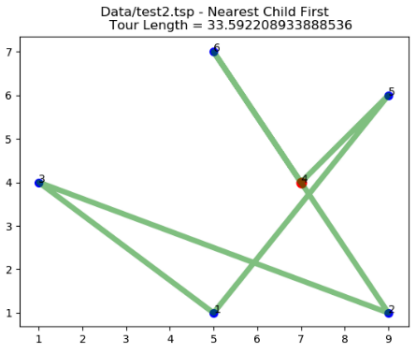


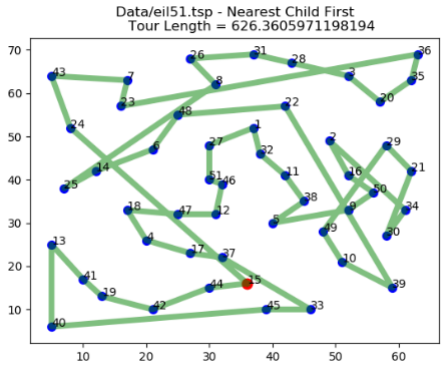


Nearest Child First

In this method, the stack is normal. When adding the child nodes to the stack, the nodes are sorted in descending order of the distance from its parent node. Thus, when popping nodes form stack, the node which is nearest to parent is visited first (since stack follows LIFO pattern).

This method also doesn't guarantee reduction in tour length in all cases

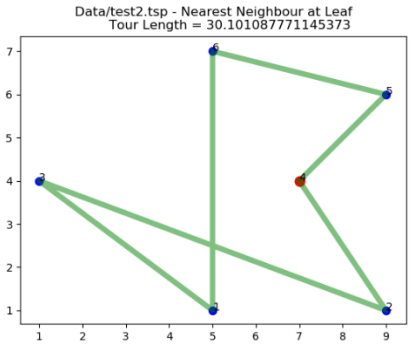


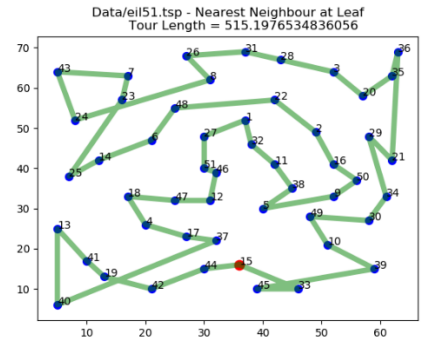


Nearest Node at Leaf

This method is a combination of above two heuristics. Here the nodes are sorted in descending order before adding to stack. Also, if the node has no child, i.e. the node is leaf node, the nearest node in stack with is chosen over other nodes in stack.

This heuristic guarantee reduction in tour length





Improvement Heuristics

By comparing the result of above heuristics, the Nearest Node at Leaf heuristic gives the best results. Also, it is observed that the crossover of the tour path is generally reduced. To reduce the cross over of tour path, some heuristics on the semi optimal path obtained above can be implemented.

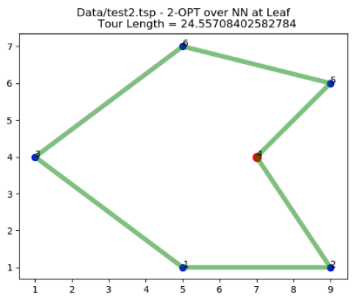
2- OPT algorithm

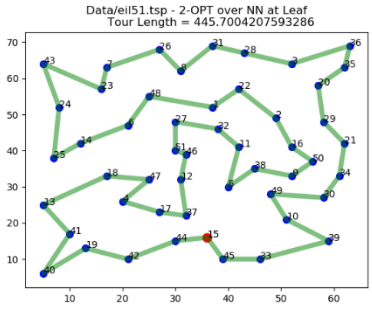
This is am improvement heuristics that operates on existing tour. In this method, two nodes are taken in the tour and their positions are switched. If the tour length thus obtained is less than the original tour length the new tour is considered. All possible swaps are performed and the tour with least distance is considered.

The time complexity of this method is higher than that of all other previous heuristics. At each iteration, there can be a maximum if $O(n^2)$ swaps possible. But it is possible that removing a crossover by a swap might create another intersection. Thus, at worst case, the complexity is $O(n!)$.

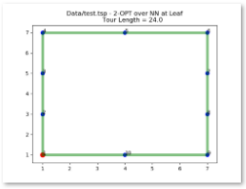
It happens that if the algorithm is performed on any arbitrary tour, multiple iterations are needed to remove all intersections. Since, we apply 2 – OPT on the tour which is at most $2 \times L(OPT)$, number of iterations required is greatly reduced. Even with 300 points it gives almost no intersections with single iteration.

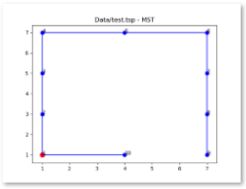
The current implementation of 2- OPT algorithm can be improved by vectorizing the tour length calculation and implementing C inline functions in python using libraries such as weaver.

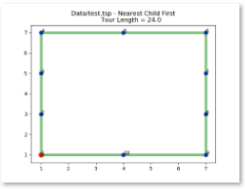


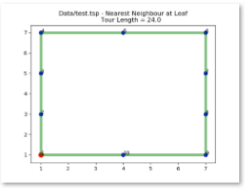


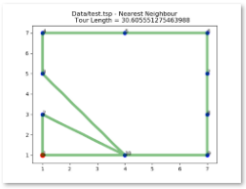
Simple Test Cases

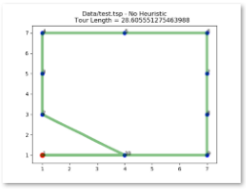

test.tsp - 2-OPT over NN at Leaf.png

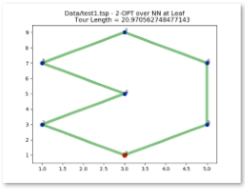

test.tsp - MST.png

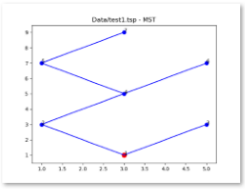

test.tsp - Nearest Child First.png

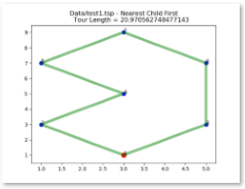

test.tsp - Nearest Neighbour at Leaf.png

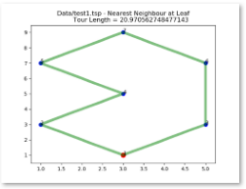

test.tsp - Nearest Neighbour.png

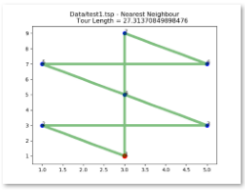

test.tsp - No Heuristic.png

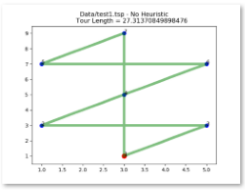

test1.tsp - 2-OPT over NN at Leaf.png

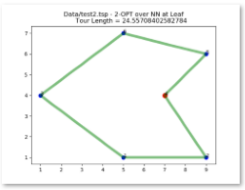

test1.tsp - MST.png

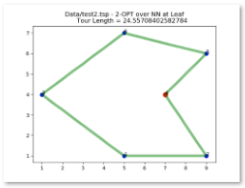

test1.tsp - Nearest Child First.png

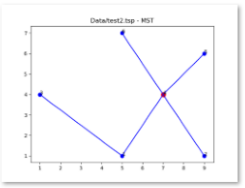

test1.tsp - Nearest Neighbour at Leaf.png

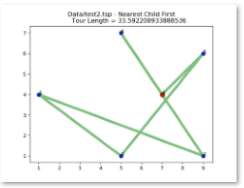

test1.tsp - Nearest Neighbour.png

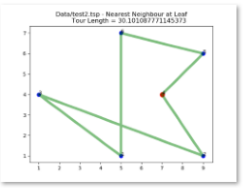

test1.tsp - No Heuristic.png

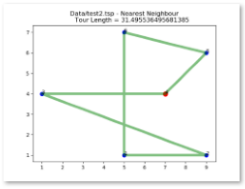

test2.tsp - 2-OPT over NN at Leaf.png

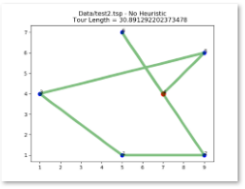

test2.tsp - 2-OPT over NN at Leaf.png


test2.tsp - MST.png


test2.tsp - Nearest Child First.png

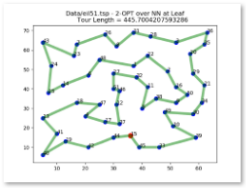

test2.tsp - Nearest Neighbour at Leaf.png

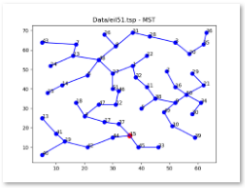

test2.tsp - Nearest Neighbour.png

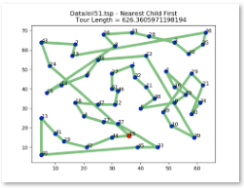

test2.tsp - No Heuristic.png

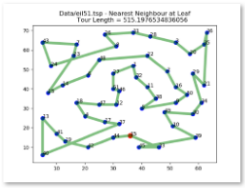
For simple cases, the 2-OPT and Nearest Neighbor at Leaf give same results but as complexity increases, 2-OPT gives best result

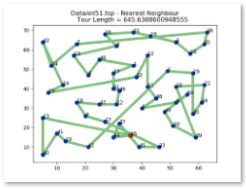
Sample Test Cases (eil51, eil76, eil101):

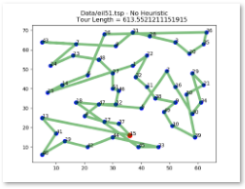

eil51.tsp - 2-OPT over NN at Leaf.png

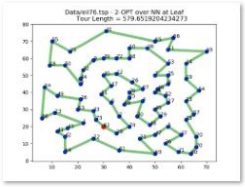

eil51.tsp - MST.png

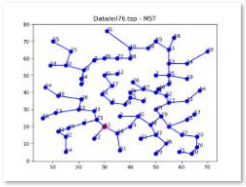

eil51.tsp - Nearest Child First.png

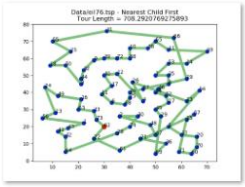

eil51.tsp - Nearest Neighbour at Leaf.png

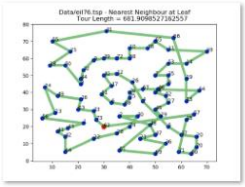

eil51.tsp - Nearest Neighbour.png

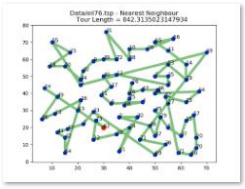

eil51.tsp - No Heuristic.png

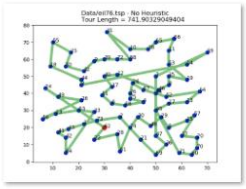

eil76.tsp - 2-OPT over NN at Leaf.png

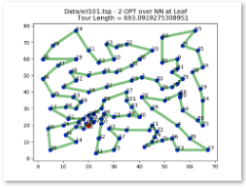

eil76.tsp - MST.png

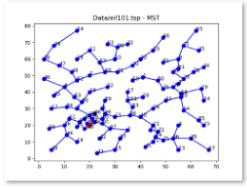

eil76.tsp - Nearest Child First.png

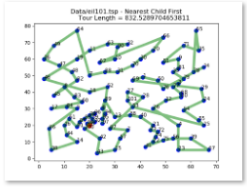

eil76.tsp - Nearest Neighbour at Leaf.png

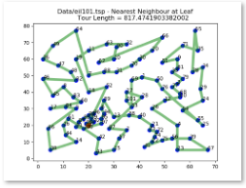

eil76.tsp - Nearest Neighbour.png

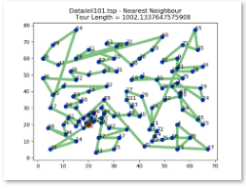

eil76.tsp - No Heuristic.png

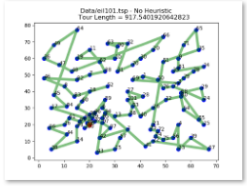

eil101.tsp - 2-OPT over NN at Leaf.png


eil101.tsp - MST.png


eil101.tsp - Nearest Child First.png

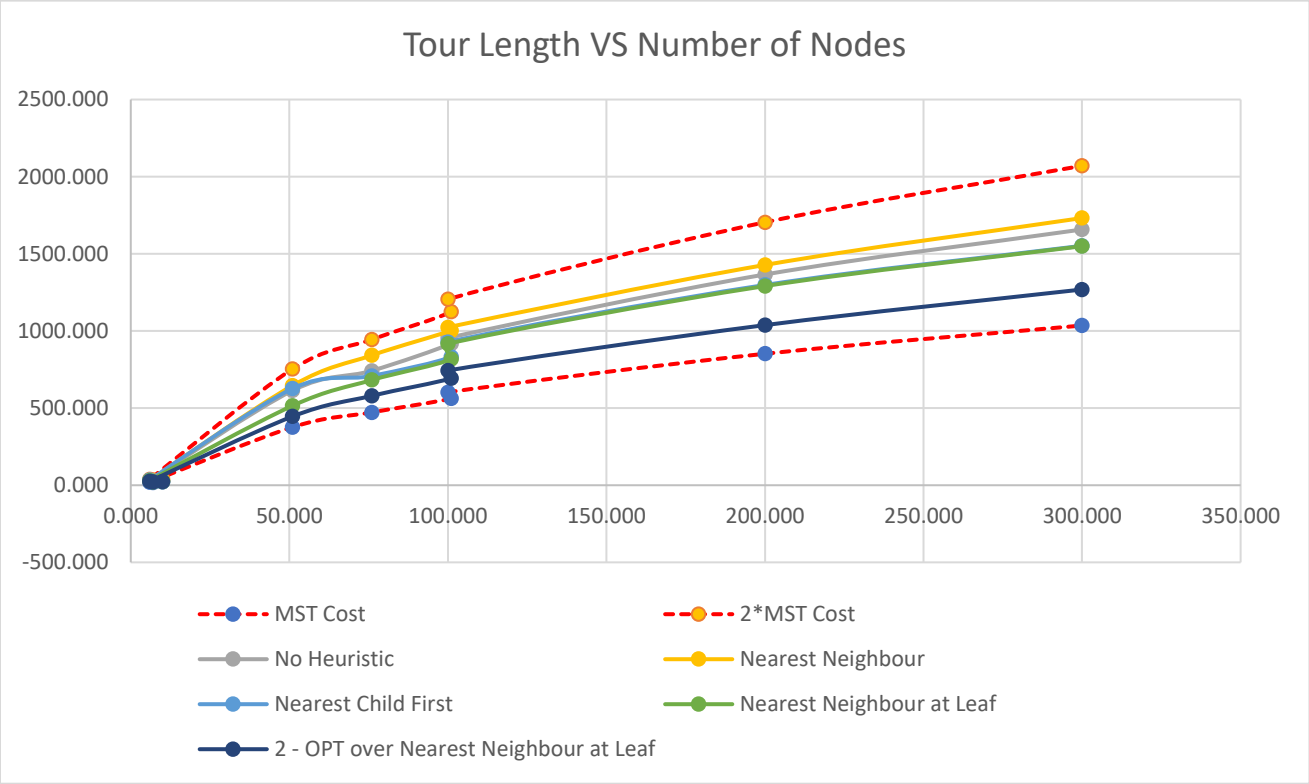

eil101.tsp - Nearest Neighbour at Leaf.png


eil101.tsp - Nearest Neighbour.png


eil101.tsp - No Heuristic.png

Comparison of different heuristics – Tour Length

File		test.tsp	test1.tsp	test2.tsp	eil51.tsp	eil76.tsp	eil101.tsp	100Random.tsp	200Random.tsp	300Random.tsp
Number of Nodes		10.000	7.000	6.000	51.000	76.000	101.000	100.000	200.000	300.000
Number of Edges		100.000	49.000	36.000	2601.000	5776.000	10201.000	10000.000	40000.000	90000.000
MST Cost		21.000	16.971	18.645	376.491	472.331	562.257	602.829	852.179	1035.075
2*MST Cost		42.000	33.941	37.290	752.981	944.661	1124.515	1205.658	1704.358	2070.149
Tour Length	No Heuristic	28.606	27.314	30.891	613.552	741.903	917.540	954.653	1365.972	1657.525
	Nearest Neighbour	30.606	27.314	31.496	645.639	842.314	1002.134	1023.862	1427.865	1731.873
	Nearest Child First	24.000	20.971	33.592	626.361	708.292	832.529	927.876	1297.951	1551.742
	Nearest Neighbour at Leaf	24.000	20.971	30.101	515.198	681.910	817.474	916.795	1290.999	1549.256
	2 - OPT over Nearest Neighbour at Leaf	24.000	20.971	24.557	445.700	579.652	693.092	743.687	1037.811	1268.699



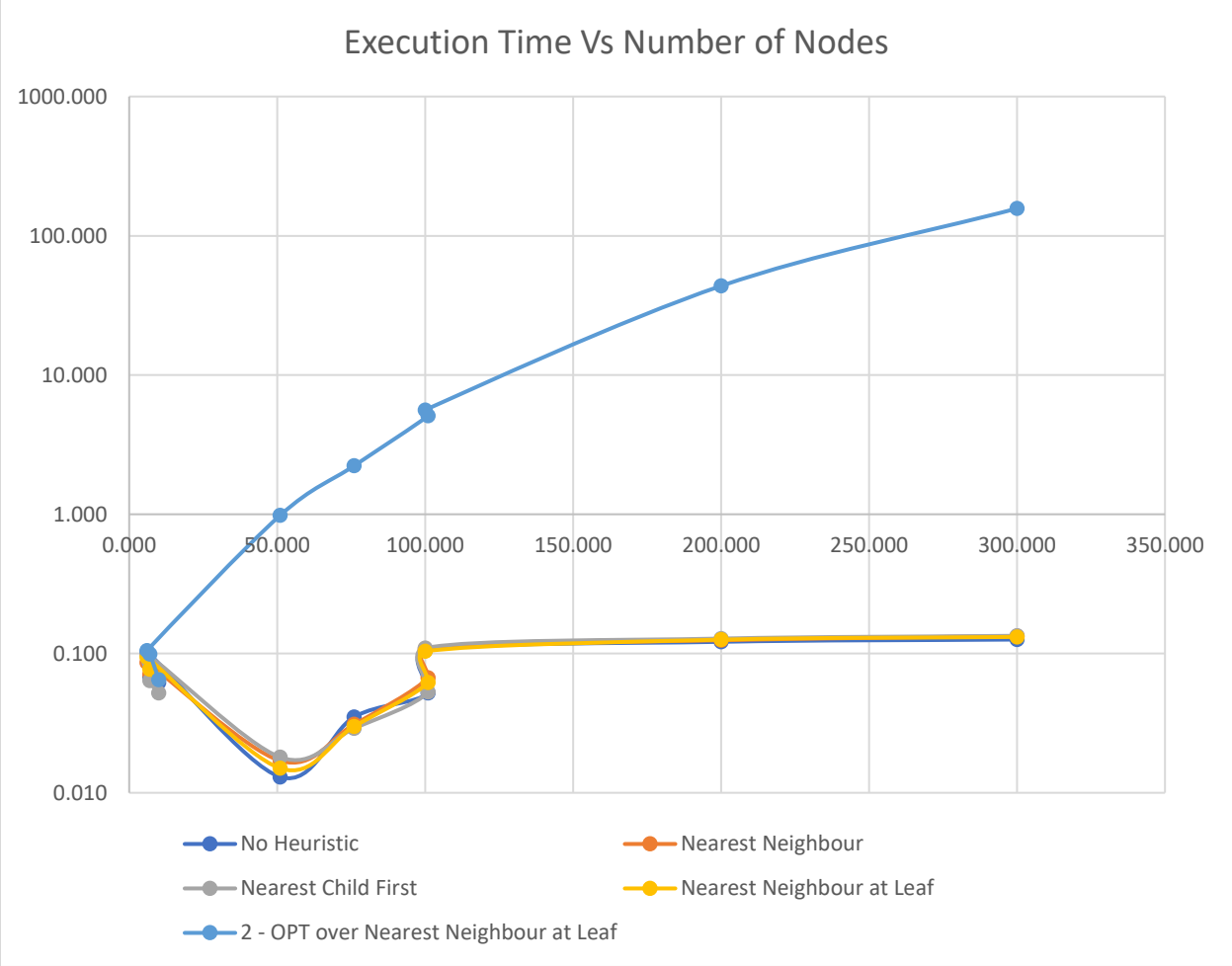
From the above graph, the tour length is $MST\ COST \leq Tour\ Length \leq 2 \times MST\ Cost$ irrespective of any heuristic chosen. Also, the 2-OPT Algorithm gives the best result of all the heuristics as expected.

Overall performance of heuristics in terms of tour length is

$$MST\ Cost \leq 2-OPT \leq NN\ at\ leaf \leq Nearest\ Child\ First \leq No\ Heuristic \leq Nearest\ neighbour \leq 2 \times MST\ Cost$$

Comparison of different heuristics – Time Complexity

File		test.tsp	test1.tsp	test2.tsp	eil51.tsp	eil76.tsp	eil101.tsp	100Random.tsp	200Random.tsp	300Random.tsp
Execution Time	No Heuristic	0.062	0.068	0.102	0.013	0.035	0.052	0.105	0.121	0.126
	Nearest Neighbour	0.068	0.071	0.086	0.017	0.031	0.067	0.108	0.125	0.134
	Nearest Child First	0.052	0.064	0.098	0.018	0.029	0.053	0.110	0.128	0.134
	Nearest Neighbour at Leaf	0.066	0.077	0.094	0.015	0.030	0.062	0.104	0.126	0.132
	2 - OPT over Nearest Neighbour at Leaf	0.065	0.099	0.105	0.982	2.230	5.098	5.614	43.642	156.951



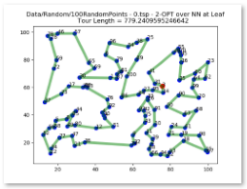
In general, the time complexity of different heuristics is given by

Heuristic	**Time Complexity**
MST	$O(E \log V)$
Nearest Neighbor	$O(V)$
Nearest Child First	$O(V)$
Nearest Neighbor at Leaf	$O(V)$
2-OPT	$O(V!)$

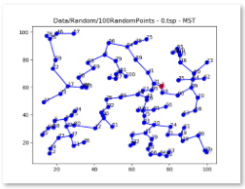
The time taken by 2-opt heuristic increases with number of vertices and from the above graph, it can be seen exponentially increasing for single iteration. As number of iterations increases the computation complexity further increases. The next best heuristic is Nearest Neighbor at leaf but the gap in tour length between the two increases with number of nodes. Thus, there must be a trade off between available resources and the tour length.

Random Scenario Testing

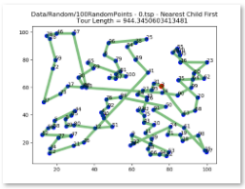
Tour for 100 uniformly distributed random points



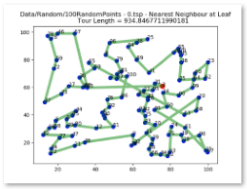
100RandomPoints - 0.tsp - 2-OPT over NN at Leaf.png



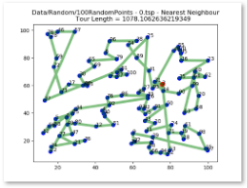
100RandomPoints - 0.tsp - MST.png



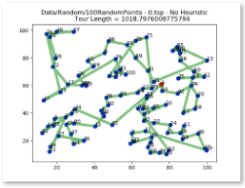
100RandomPoints - 0.tsp - Nearest Child First.png



100RandomPoints - 0.tsp - Nearest Neighbour at Leaf.png



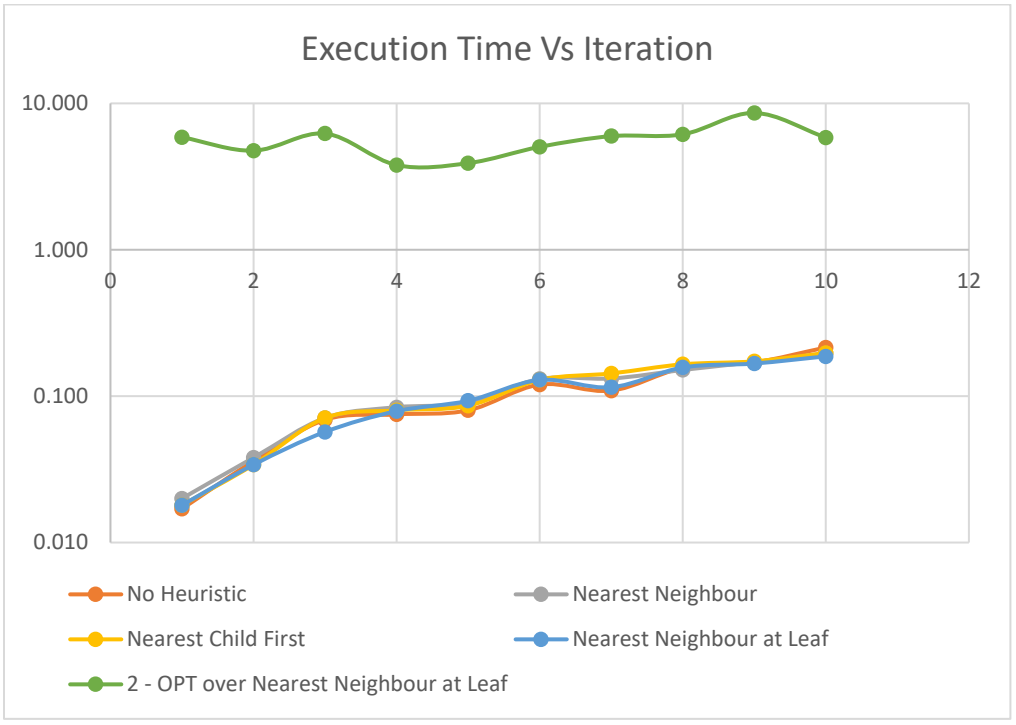
100RandomPoints - 0.tsp - Nearest Neighbour.png



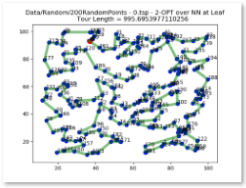
100RandomPoints - 0.tsp - No Heuristic.png

Iteration		1.000	2.000	3.000	4.000	5.000	6.000	7.000	8.000	9.000	10.000	Average
Number of Nodes		100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.000	**100.000**
Number of Edges		10000.000	10000.000	10000.000	10000.000	10000.000	10000.000	10000.000	10000.000	10000.000	10000.000	**10000.000**
Tour Length	MST Cost	617.999	551.952	644.911	588.457	613.041	626.693	586.694	614.057	586.463	598.025	**602.829**
	No Heuristic	1018.798	868.756	1053.849	919.501	942.939	994.766	903.828	932.545	938.698	972.847	**954.653**
	Nearest Neighbour	1078.106	965.782	1104.200	942.271	991.733	1053.457	956.001	1062.438	1004.723	1079.904	**1023.862**
	Nearest Child First	944.345	835.482	1043.527	919.932	878.934	979.629	898.195	942.628	900.317	935.773	**927.876**
	Nearest Neighbour at Leaf	934.847	775.614	1050.192	887.138	910.066	948.866	845.354	988.803	943.968	883.106	**916.795**
	2 - OPT over Nearest Neighbour at Leaf	779.241	694.620	772.882	686.639	780.788	797.112	723.415	765.635	711.077	725.459	**743.687**

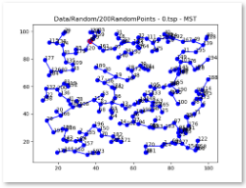
Iteration		1.000	2.000	3.000	4.000	5.000	6.000	7.000	8.000	9.000	10.000	Average
Execution Time	No Heuristic	0.017	0.036	0.069	0.075	0.080	0.120	0.109	0.156	0.170	0.215	**0.105**
	Nearest Neighbour	0.020	0.038	0.071	0.084	0.090	0.131	0.132	0.151	0.171	0.196	**0.108**
	Nearest Child First	0.018	0.034	0.071	0.080	0.086	0.129	0.143	0.165	0.173	0.197	**0.110**
	Nearest Neighbour at Leaf	0.018	0.034	0.057	0.079	0.093	0.129	0.115	0.157	0.167	0.187	**0.104**
	2 - OPT over Nearest Neighbour at Leaf	5.872	4.751	6.220	3.788	3.904	5.044	5.974	6.143	8.603	5.841	**5.614**



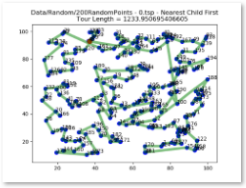
Tour for 200 uniformly distributed random points



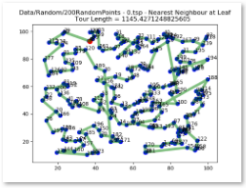
200RandomPoints - 0.tsp - 2-OPT over NN at Leaf.png



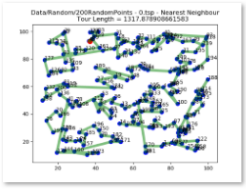
200RandomPoints - 0.tsp - MST.png



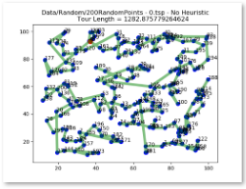
200RandomPoints - 0.tsp - Nearest Child First.png



200RandomPoints - 0.tsp - Nearest Neighbour at Leaf.png

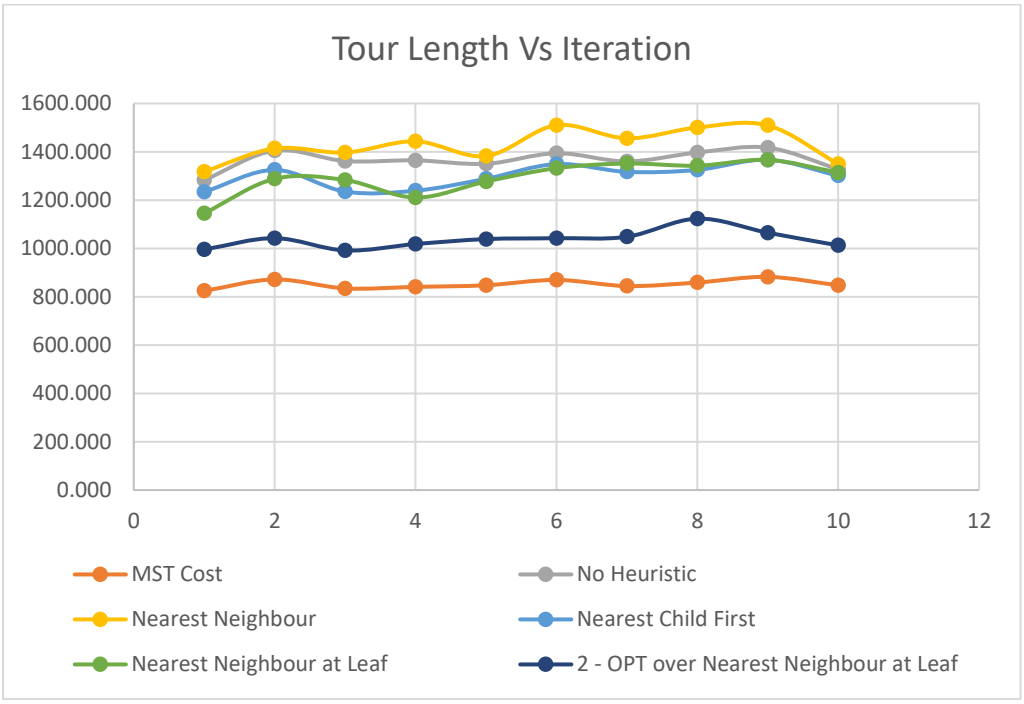


200RandomPoints - 0.tsp - Nearest Neighbour.png

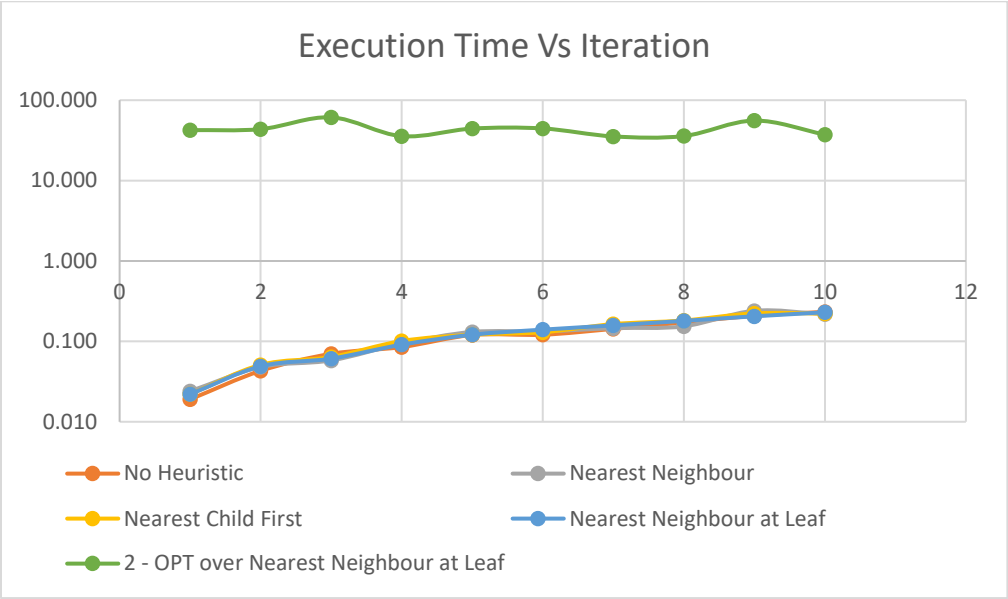


200RandomPoints - 0.tsp - No Heuristic.png

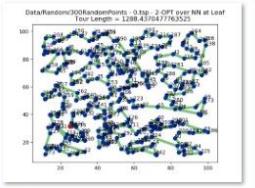
Iteration		1.000	2.000	3.000	4.000	5.000	6.000	7.000	8.000	9.000	10.000	Average
Number of Nodes		200.000	200.000	200.000	200.000	200.000	200.000	200.000	200.000	200.000	200.000	**200.000**
Number of Edges		40000.000	40000.000	40000.000	40000.000	40000.000	40000.000	40000.000	40000.000	40000.000	40000.000	**40000.000**
Tour Length	MST Cost	824.321	871.688	834.668	840.819	847.613	869.579	844.389	858.879	882.195	847.639	**852.179**
	No Heuristic	1282.876	1404.706	1361.602	1364.422	1350.316	1393.790	1360.152	1396.906	1417.057	1327.897	**1365.972**
	Nearest Neighbour	1317.879	1413.648	1396.917	1443.327	1382.173	1509.852	1455.591	1500.376	1509.150	1349.739	**1427.865**
	Nearest Child First	1233.951	1324.793	1235.130	1238.879	1288.050	1349.038	1316.912	1324.903	1366.893	1300.965	**1297.951**
	Nearest Neighbour at Leaf	1145.427	1288.071	1283.170	1210.502	1276.745	1332.064	1351.363	1342.294	1366.243	1314.109	**1290.999**
	2 - OPT over Nearest Neighbour at Leaf	995.695	1042.450	991.679	1018.159	1038.658	1042.323	1048.225	1123.216	1065.093	1012.616	**1037.811**

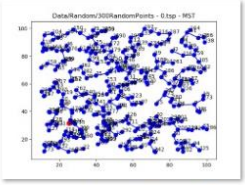


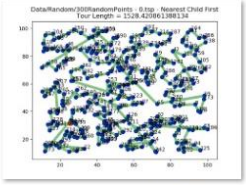
Iteration		1.000	2.000	3.000	4.000	5.000	6.000	7.000	8.000	9.000	10.000	Average
Execution Time	No Heuristic	0.019	0.043	0.070	0.085	0.120	0.121	0.143	0.172	0.208	0.233	**0.121**
	Nearest Neighbour	0.024	0.048	0.058	0.094	0.131	0.133	0.146	0.154	0.240	0.218	**0.125**
	Nearest Child First	0.022	0.051	0.064	0.101	0.121	0.130	0.164	0.182	0.223	0.222	**0.128**
	Nearest Neighbour at Leaf	0.022	0.049	0.061	0.091	0.121	0.140	0.158	0.180	0.205	0.230	**0.126**
	2 - OPT over Nearest Neighbour at Leaf	42.436	43.684	61.211	35.710	44.380	44.507	35.440	35.839	55.809	37.405	**43.642**

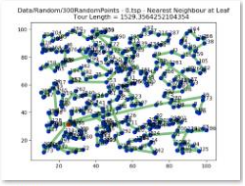


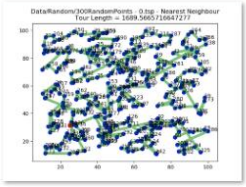
Tour for 300 uniformly distributed random points

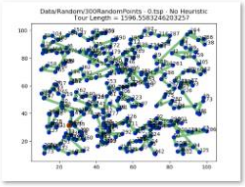

300RandomPoints - 0.tsp - 2-OPT over NN at Leaf.png


300RandomPoints - 0.tsp - MST.png

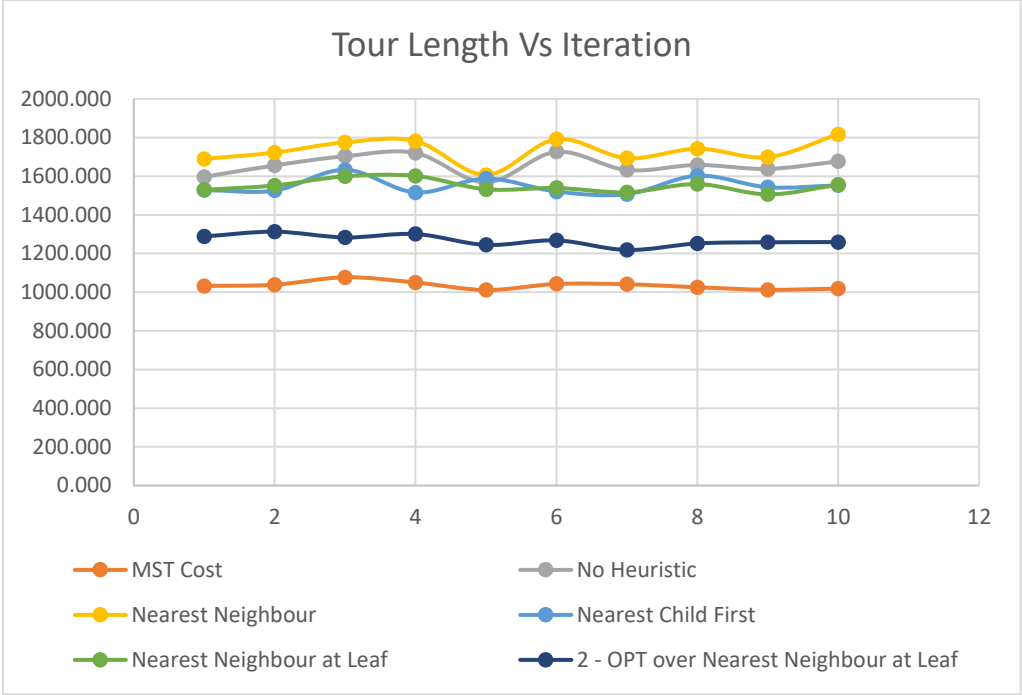

300RandomPoints - 0.tsp - Nearest Child First.png


300RandomPoints - 0.tsp - Nearest Neighbour at Leaf.png

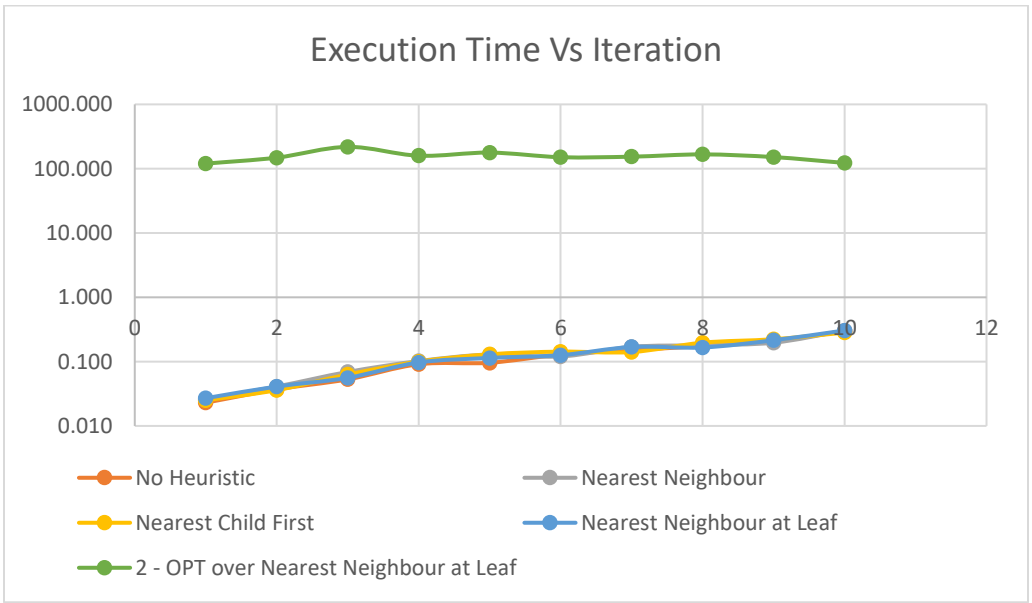

300RandomPoints - 0.tsp - Nearest Neighbour.png


300RandomPoints - 0.tsp - No Heuristic.png

Iteration		1.000	2.000	3.000	4.000	5.000	6.000	7.000	8.000	9.000	10.000	Average
Number of Nodes		300.000	300.000	300.000	300.000	300.000	300.000	300.000	300.000	300.000	300.000	**300.000**
Number of Edges		90000.000	90000.000	90000.000	90000.000	90000.000	90000.000	90000.000	90000.000	90000.000	90000.000	**90000.000**
Tour Length	MST Cost	1032.504	1038.631	1077.240	1050.419	1011.433	1042.617	1040.959	1025.673	1012.545	1018.726	**1035.075**
	No Heuristic	1596.558	1656.002	1703.459	1720.452	1568.072	1726.200	1631.879	1658.267	1637.814	1676.548	**1657.525**
	Nearest Neighbour	1689.567	1722.422	1774.938	1781.910	1606.804	1791.480	1693.236	1742.274	1698.936	1817.165	**1731.873**
	Nearest Child First	1528.421	1525.944	1633.353	1515.899	1587.781	1520.757	1507.003	1602.635	1542.976	1552.648	**1551.742**
	Nearest Neighbour at Leaf	1529.356	1552.278	1599.621	1601.541	1532.474	1539.192	1516.707	1559.446	1506.055	1555.893	**1549.256**
	2 - OPT over Nearest Neighbour at Leaf	1288.437	1313.167	1283.391	1300.904	1244.753	1268.054	1218.503	1252.278	1258.276	1259.230	**1268.699**



Iteration		1.000	2.000	3.000	4.000	5.000	6.000	7.000	8.000	9.000	10.000	Average
Execution Time	**No Heuristic**	0.023	0.037	0.053	0.091	0.096	0.129	0.151	0.174	0.207	0.299	**0.126**
	Nearest Neighbour	0.026	0.041	0.069	0.102	0.129	0.120	0.170	0.180	0.197	0.303	**0.134**
	Nearest Child First	0.025	0.036	0.063	0.100	0.130	0.143	0.141	0.197	0.223	0.282	**0.134**
	Nearest Neighbour at Leaf	0.027	0.041	0.056	0.097	0.114	0.126	0.169	0.166	0.215	0.305	**0.132**
	2 - OPT over Nearest Neighbour at Leaf	119.944	148.482	218.191	159.124	177.935	150.825	153.719	166.863	151.541	122.887	**156.951**



K Robot TSP

K robot TSP is a variation of Metric TSP where instead of single agent covering all the nodes, there are K agents that cover nodes. There the length of the largest tour must be minimized

General Assumption

Since the graph is fully connected and all the edge weights are Euclidean, the triangle inequality holds. Thus it is safe to assume that if the sum of distances of the nodes form centroid for cluster A is greater than that of cluster B , then, MST cost of nodes in cluster A will also be greater than MST cost of nodes in cluster B.

Algorithm 1 – Simple N Division

In this method all the nodes are taken and grouped into K clusters (excluding the starting node). MST is formed by including the starting point of all robots and for each cluster and a tour is generated. Each tour is assigned to one robot.

This algorithm will be highly inefficient since the nodes are assigned at random. There will be lot of intersections even if we optimize the tour of each cluster without any intersections in sub tours.

Algorithm 2 – K Means Clustering

In this method instead of randomly assigning the nodes to clusters, K Means algorithm is used for grouping the nodes. In this method also the starting node is excluded from the clustering problem. MST and thus tour is formed for each cluster. An example is as shown below

This method highly depends on the position of starting node. The tour length will be longer or shorter depending on the start node position. An Example is shown below were even though the clusters are optimal the Route 1 much larger than any other route.

Another case where this method fails is when the clusters are skewed. If the nodes are clustered in such a way that few points are grouped and far off from other points, since his method doesn't care for number of nodes in a cluster, the tour length might be very large for these set of point.

Presence of outliers to each cluster also greatly affects the optimal olution

Algorithm 3 – Balanced K Means Clustering

In this method instead of clustering based on distance from centroid alone, number of nodes in each cluster is also considered.

The cost function is two-fold

1. Sum of distance from centroid.
2. Difference in distance of each node from nearest cluster centroid and farthest cluster centroid.

The balanced K – Means algorithm is as follows

1. Compute current cluster means
2. For each object, compute the distances to the cluster means
3. Sort elements based on the delta of the current assignment and the best possible alternate assignment.
4. For each element by priority:
    1. For each other cluster, by element gain, unless already moved:
        1. If there is an element wanting to leave the other cluster and this swap yields and improvement, swap the two elements
        2. If the element can be moved without violating size constraints, move it
    2. If the element was not changed, add to outgoing transfer list.
5. If no more transfers were done (or max iteration threshold was reached), terminate

In this algorithm also depending on the position of the starting node, the tour lengths will not be equal for each cluster

Algorithm 4 – Balanced K means with Weighted Clusters

This method is similar to Algorithm 3, but the cost function here is 3-fold

1. Sum of distance from centroid.
2. Difference in distance of each node from nearest cluster centroid and farthest cluster centroid.
3. Sum of distance of the clusters from centroid is scaled by inverse of distance between cluster centroid and starting node.

Since the cost function is scaled based on the start node position, the effect of start node can be reduced. The minimization of this cost is through iterative algorithm as mentioned in Algorithm 3.

As shown in the above diagram, Route1 covers less nodes compared to other route. This is due to inclusion of position on starting node in cost function.

Conclusion

All the above algorithms are based on assumption as mentioned. In case the assumption fails, the cost function has to be changed to include the cost of MST for each cluster and the distance of non-cluster nodes has to be calculated not to centroid but to nearest leaf node in MST