**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

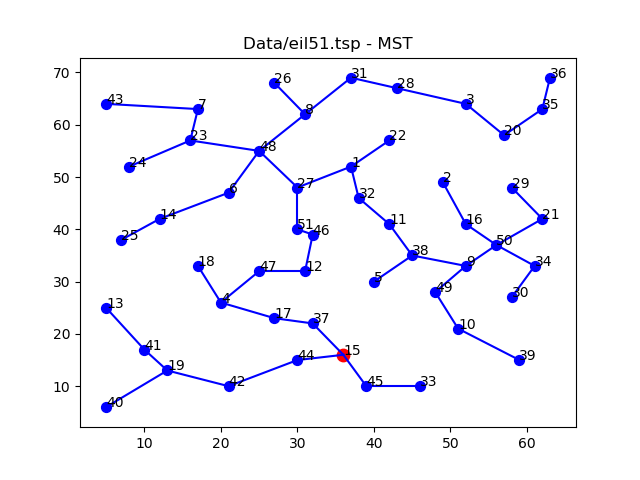
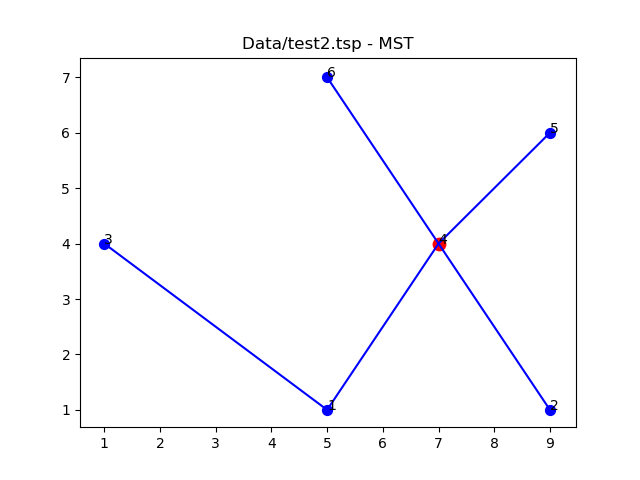
**Finding MST in Graph (Kruskal’s Algorithm)**

|  |  |
| --- | --- |
| Step 1 | Initialize empty set |
| 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  Else:  Discard the edge |
| Step 5 | Form Tree from the edges in set |

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

Where,



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.

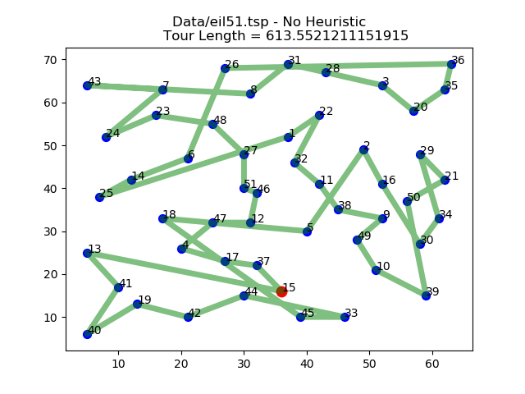
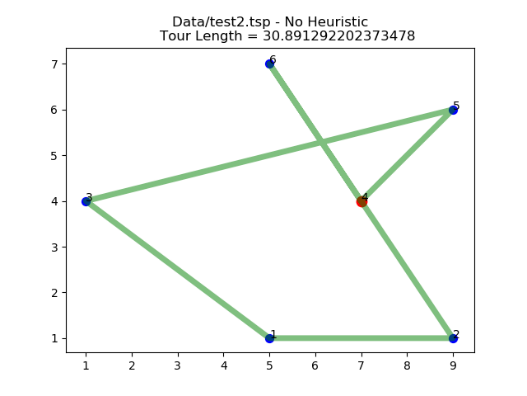
|  |  |
| --- | --- |
| Step 1 | Find root node |
| Step 2 | Get all children for the node |
| Step 3 | Select one child node that is not added to tour |
| Step 4 | If the is leaf node:  Go to Step 5  Else:  Go to Step 2 |
| Step 5 | If all nodes are visited:  Terminate  Else:  Go to parent node |

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.

This approach ensures that the tour will be at most



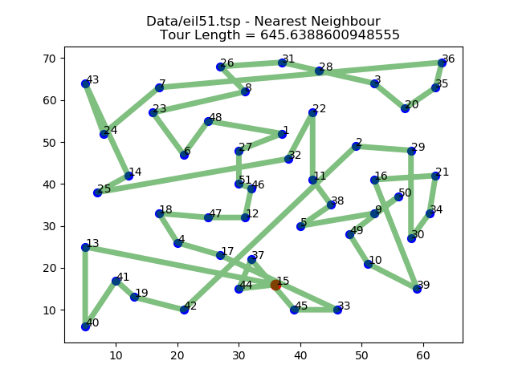
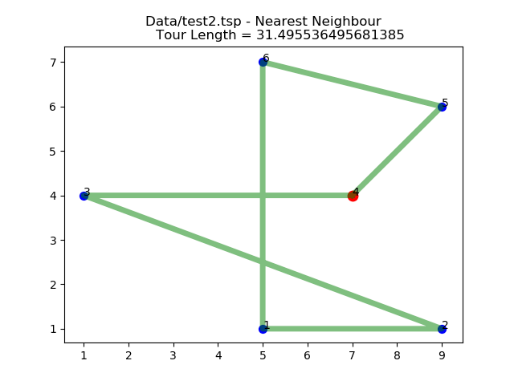
**Heuristics on Depth first traversal**

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

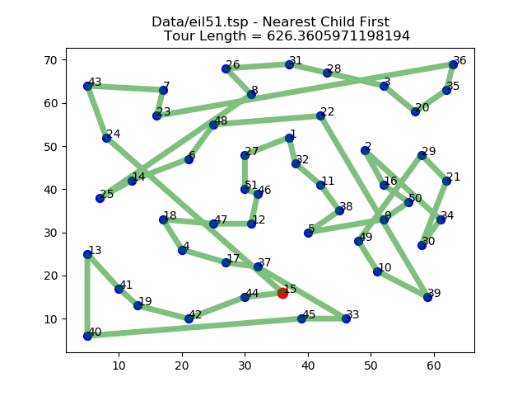
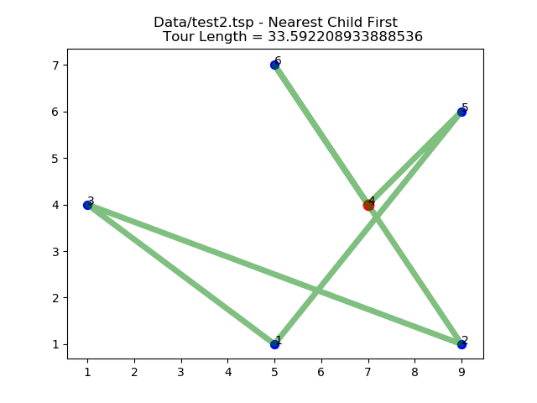
This method need not reduce the tour length and instead it often increases the tour length



**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).

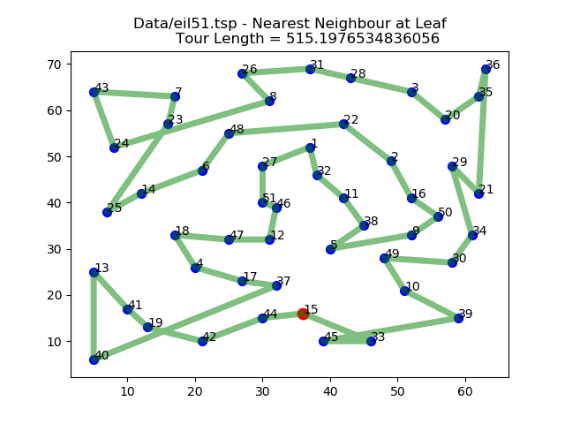
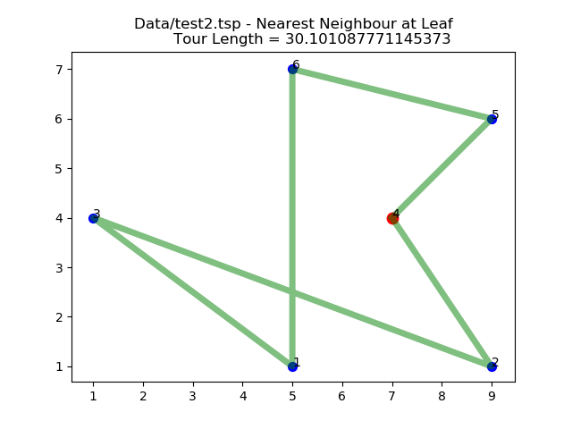
This method also doesn’t guarantee reduction in tour length in all cases



**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.

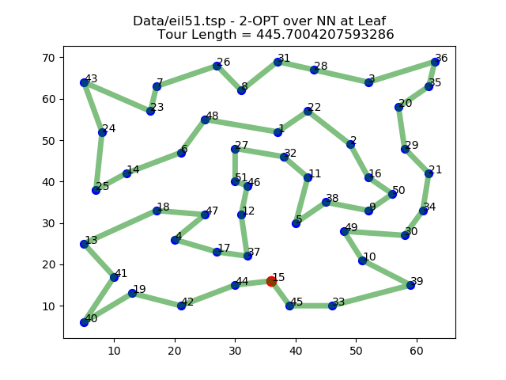
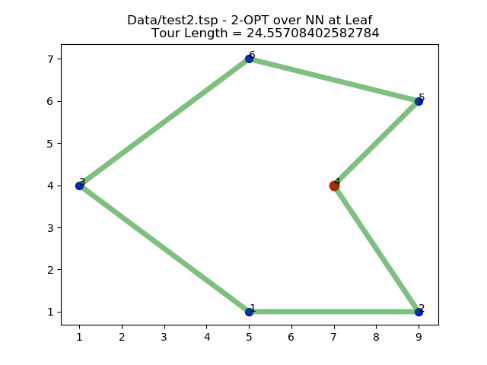
**2- OPT algorithm**

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.

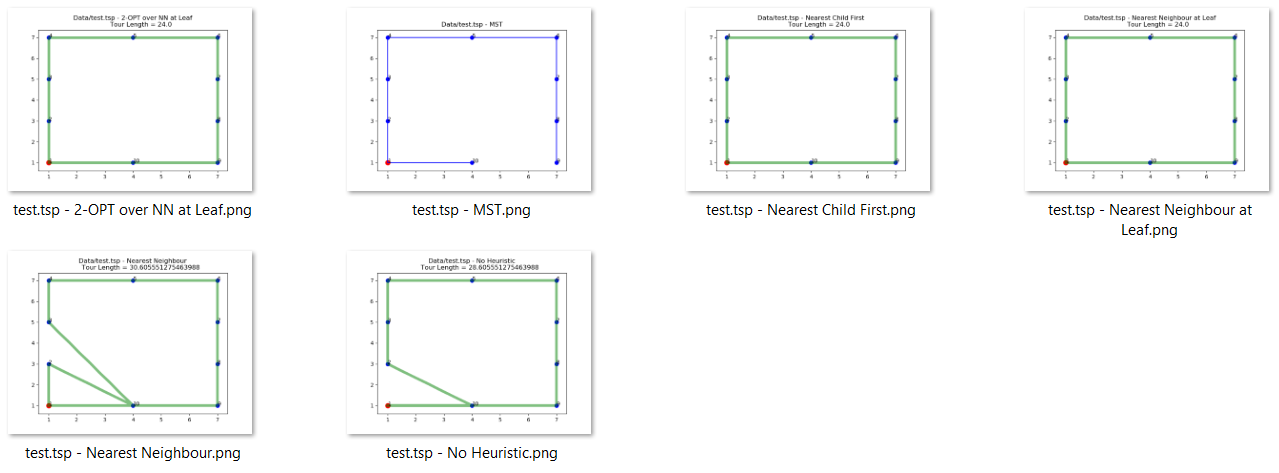
The time complexity of this method is higher than that of all other previous heuristics. At each iteration, there can be a maximum if swaps possible. But it is possible that removing a crossover by a swap might create another intersection. Thus, at worst case, the complexity is .

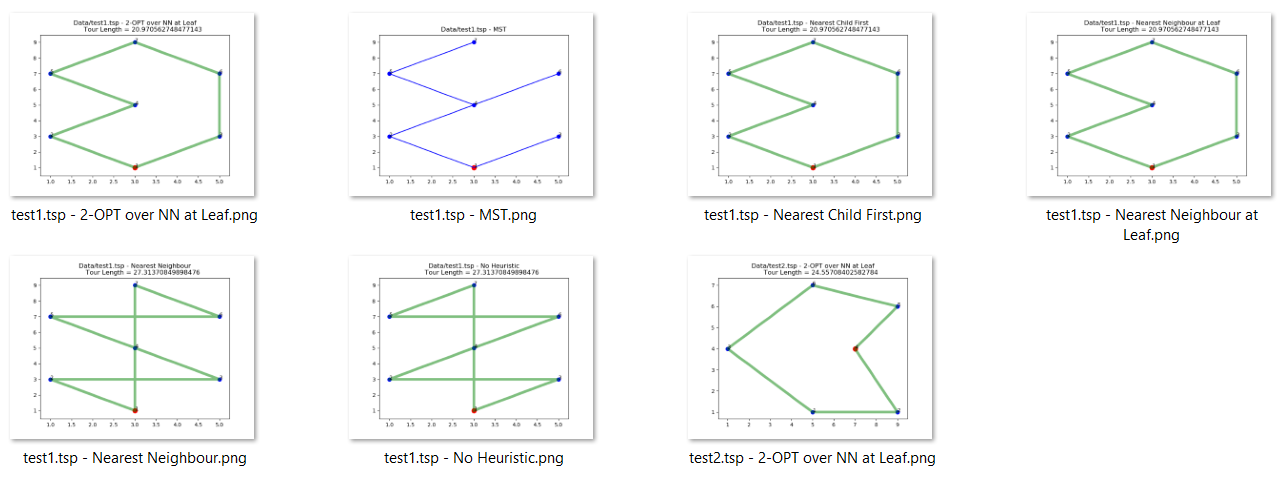
It happens that if the algorithm is performed on any arbitrary tour, multiple iterations are needed to remove all intersections. Since, we apply 2 – OPT on the tour which is at most , number of iterations required is greatly reduced. Even with 300 points it gives almost no intersections with single iteration.

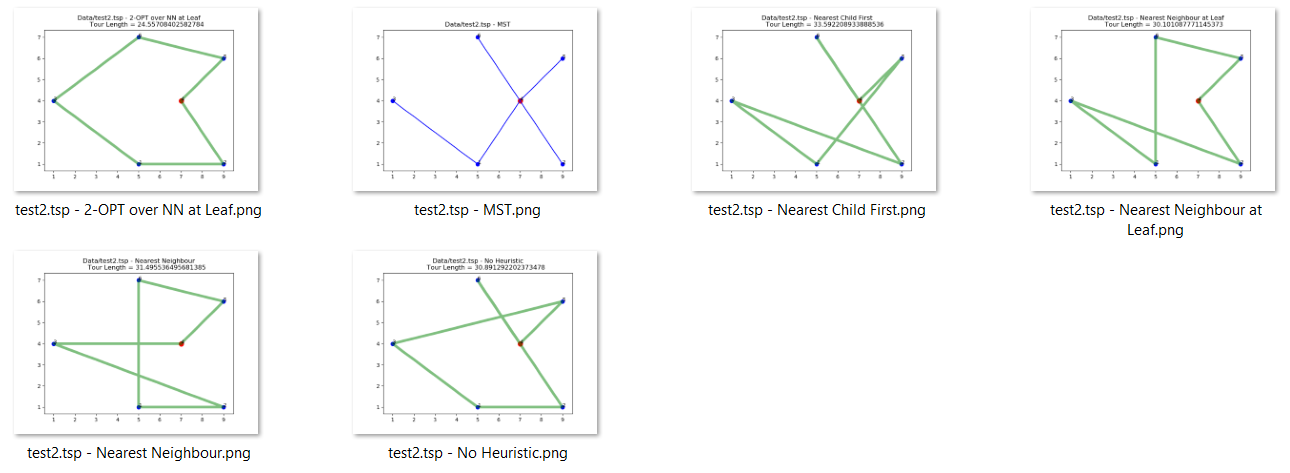
The current implementation of 2- OPT algorithm can be improved by vectorizing the tour length calculation and implementing C inline functions in python using libraries such as weaver.



**Simple Test Cases**

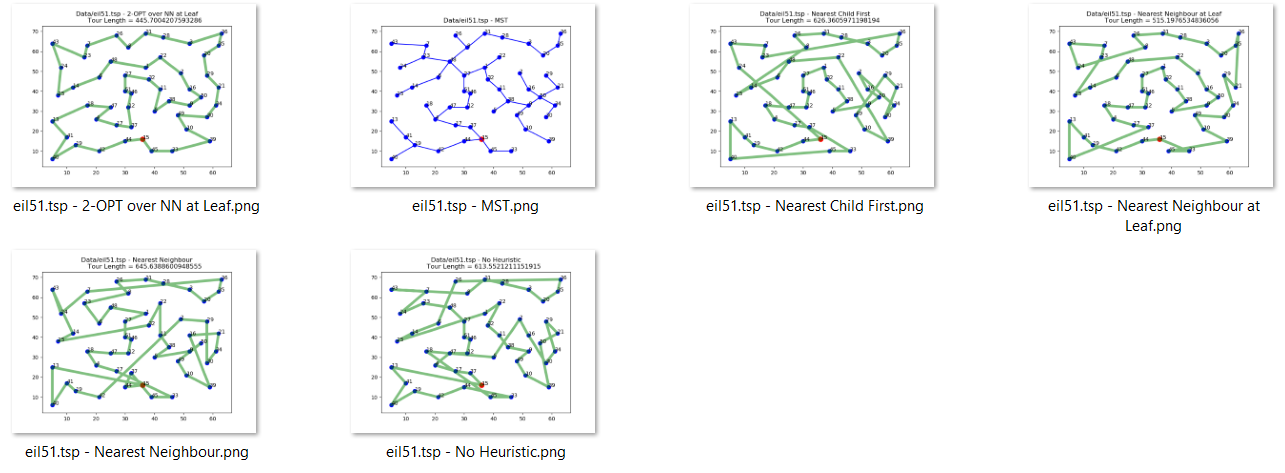


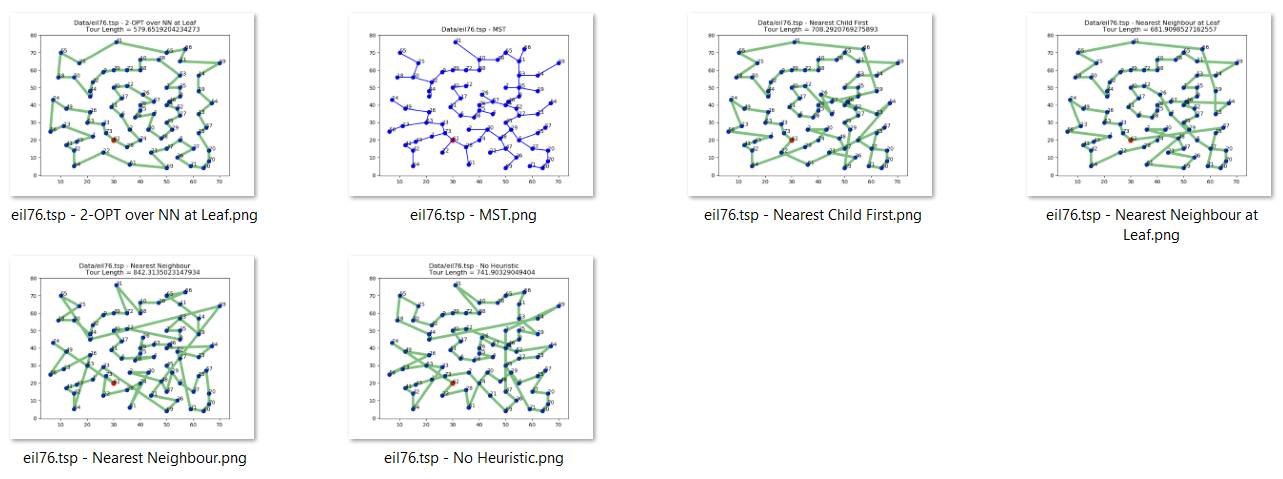


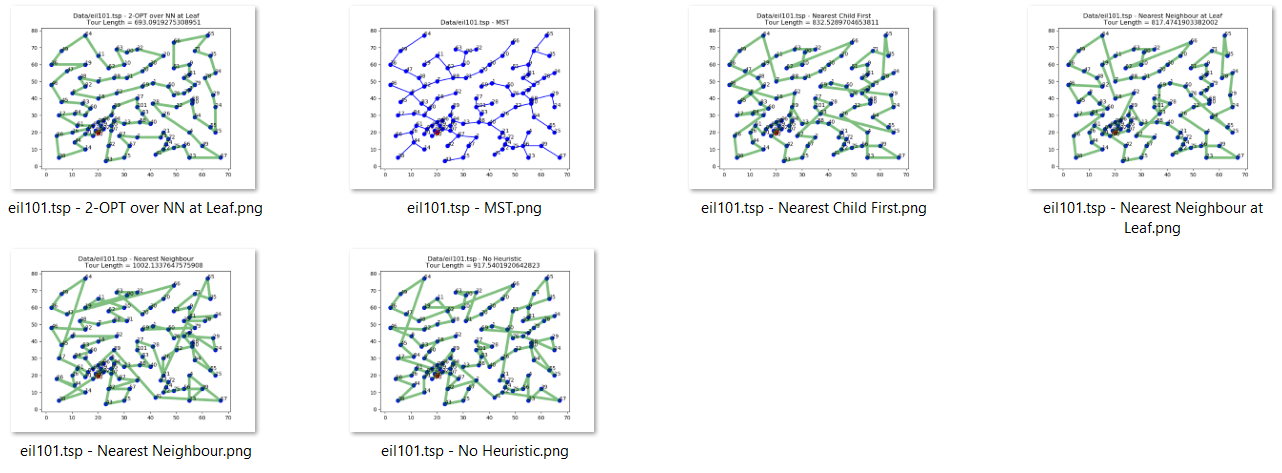


For simple cases, the 2-OPT and Nearest Neighbor at Leaf give same results but as complexity increases, 2-OPT gives best result

**Sample Test Cases (eil51, eil76, eil101):**







**Comparison of different heuristics – Tour Length**



From the above graph, the tour length is 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

**Comparison of different heuristics – Time Complexity**



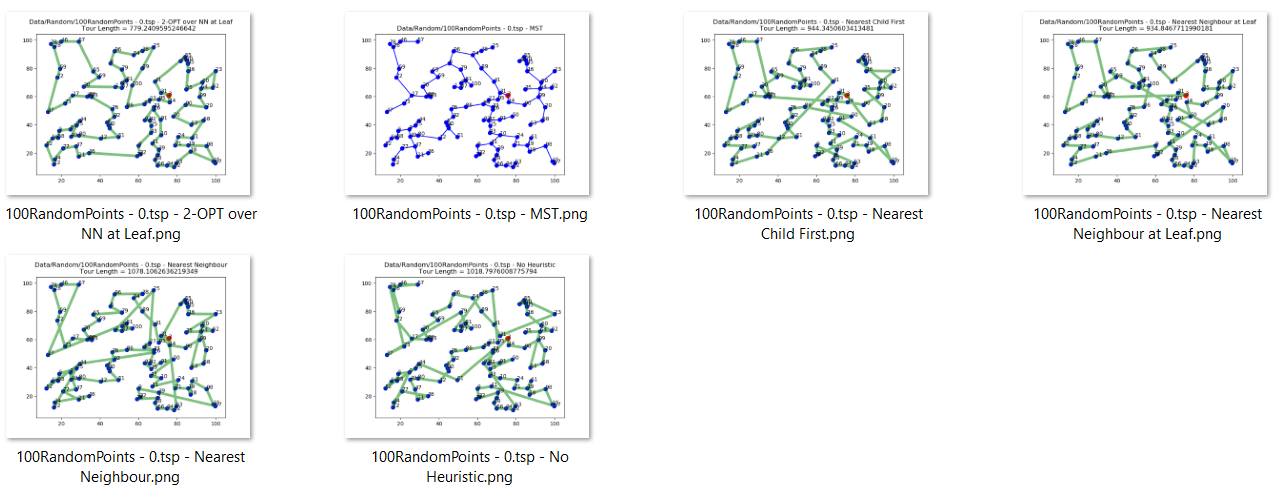
In general, the time complexity of different heuristics is given by

|  |  |
| --- | --- |
| **Heuristic** | **Time Complexity** |
| MST |  |
| Nearest Neighbor |  |
| Nearest Child First |  |
| Nearest Neighbor at Leaf |  |
| 2-OPT |  |

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**

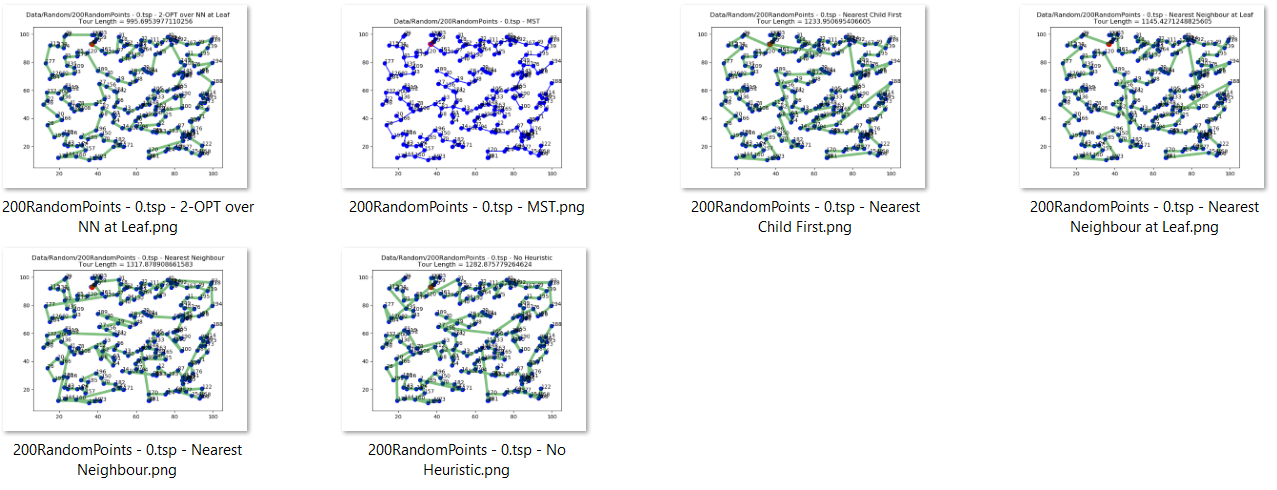
**Tour for 100 uniformly distributed random points**







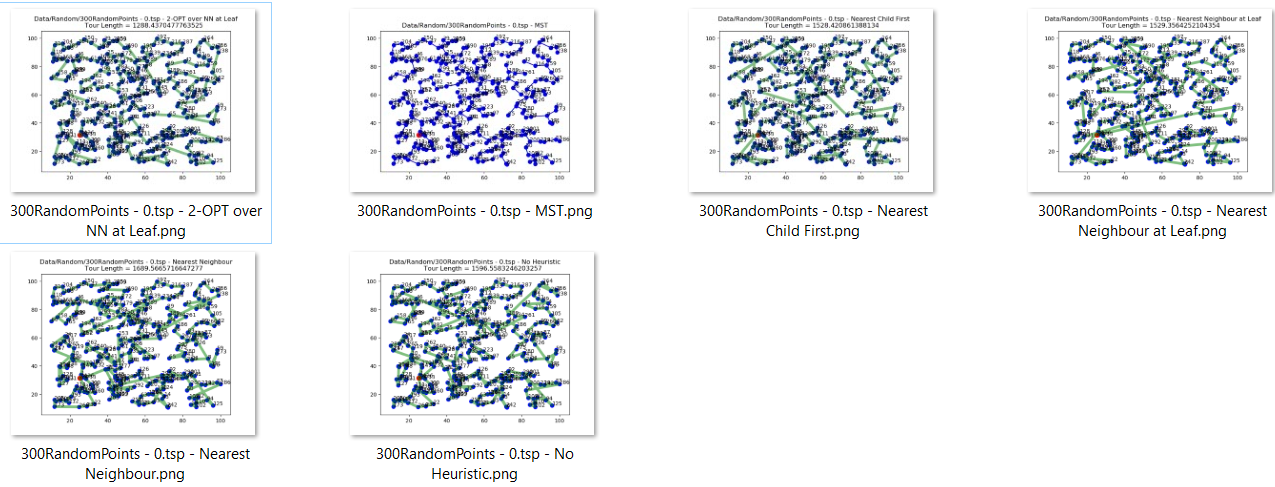
**Tour for 200 uniformly distributed random points**







**Tour for 300 uniformly distributed random points**







**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