Survey of Nearest Neighbor & MST Heuristics AND Comparison With the Exact TSP Algorithm

https://github.com/theyusko/tsp-heuristics

Summary

20 Asymmetric, 20 Symmetric, 20 Euclidean and 20 Non-Euclidean graphs are randomly generated in Java. Then, two known heuristics for TSP, Nearest Neighbor and Minimum Spanning Tree, are proposed and implemented in Python. Then, an exact algorithm is implemented in Xpress. Finally, the exact algorithm and the heuristics are run on the dataset generated previously and compared.

Algorithms

Graph Generation Pseudocode

For the purpose of having a huge number of datasets, as a group, a Java code was implemented which creates a totally unbiased and random Symmetric, Asymmetric, Euclidean and Non-Euclidean graphs.

Distances Matrice: An (nxn) matrix that holds the distances where n denotes the number of nodes inside the graph.

Minimum Cycle Array: An array which creates a minimum cycle which is completely random. To clarify, assume user wants to generate an array with 4 nodes and assume their names are A, B, C, D. Minimum Cycle Array helps us to define Minimum Cycle inside the graph. To do that all the nodes are added to the array one by one (Minimum Cycle Array = [A, B, C, D]) and then shuffled (Assume, Minimum Cycle Array = [B, D, A, C]). This shows us that there are edges from B to D, D to A, A to C and lastly, C to B. By using that array, program guarantees that there is a cycle in the graph which is created completely randomly.

Locations Matrice: An (nxd) matrix where n denotes the number of nodes and d denotes the dimension. All the coordinates was generated randomly and stored in that matrice.

Pseudo code of the Symmetric Graph Creator:
To conduct a Symmetric Graph,

- A 2-D array (Distances Matrice) which holds the distances between nodes was created and all the values is assigned to zero.

- An array (Minimum Cycle Array) which holds a basic cycle that includes all the nodes was created. After that, this array is shuffled.
- Then, half of the Distances Matrice is traced until reaching to the point (nxn) where n denotes the number of nodes
  - All the Distances Matrice locations (n,m) where (m = n) is set to '-1' since there is no edge from a node to itself.
  - According to the Minimum Cycle Array, all the Distance Matrice locations was changed with the random distances
  - All other distances was first randomly selected. If the edge is selected, then the length of that edge is changed with a random value, otherwise, nothing done and the length of that edge is stayed as it was in the beginning which is equal to zero.
- Other half of the Distance Matrix was filled according to the generated half. By this way, program guarantees that the graph is symmetric.
- Distances Matrix was written to a text file for using it in XPress, Python and Java.

Pseudo code of the Asymmetric Graph Creator:

To conduct an Asymmetric Graph,

- A 2-D array (Distances Matrice) which holds the distances between nodes was created and all the values is assigned to zero.
- An array (Minimum Cycle Array) which holds a basic cycle that includes all the nodes was created. After that, this array is shuffled.
- Then, the Distances Matrice is traced until reaching to the point (nxn) where n denotes the number of nodes.
  - All the Distances Matrice locations (n,m) where (m = n) is set to '-1' since there is no edge from a node to itself.
  - According to the Minimum Cycle Array, all the Distance Matrice locations was changed with real distances that was calculated by using the nodes' random locations.
  - All other distances was first randomly selected. If the edge is selected, then the length of that edge is changed with with real distances that was calculated by using the nodes' random locations. Otherwise, nothing done and the length of that edge is stayed as it was in the beginning which is equal to zero.
- Distances Matrix was written to a text file for using it in XPress, Python and Java.

Pseudo code of the Euclidean Graph Creator:

To conduct an Euclidean Graph,

- A 2-D array (Distances Matrice) which holds the distances between nodes was created and all the values is assigned to zero.
- An array (Minimum Cycle Array) which holds a basic cycle that includes all the nodes was created. After that, this array is shuffled.
- Then, half of the Distances Matrice is traced until reaching to the point (nxn) where n denotes the number of nodes
  - All the Distances Matrice locations (n,m) where (m = n) is set to '-1' since there is no edge from a node to itself.

    - According to the Minimum Cycle Array, all the Distance Matrice locations was changed with the random distances
    - All other distances was first randomly selected. If the edge is selected, then the length of that edge is changed with a random value, otherwise, nothing done and the length of that edge is stayed as it was in the beginning which is equal to zero.
- Other half of the Distance Matrix was filled according to the generated half. By this way, program guarantees that the graph is symmetric.
- Distances Matrix was written to a text file for using it in XPress, Python and Java.

Pseudo code of the Non-Euclidean Graph Creator:

To conduct an Non-Euclidean Graph,

- A 2-D array (Distances Matrice) which holds the distances between nodes was created and all the values is assigned to zero.
- An array (Minimum Cycle Array) which holds a basic cycle that includes all the nodes was created. After that, this array is shuffled.
- Then, half of the Distances Matrice is traced until reaching to the point (nxn) where n denotes the number of nodes
    - All the Distances Matrice locations (n,m) where (m = n) is set to '-1' since there is no edge from a node to itself.
    - According to the Minimum Cycle Array, all the Distance Matrice locations was changed with the random distances
    - All other distances was first randomly selected. If the edge is selected, then the length of that edge is changed with a random value, otherwise, nothing done and the length of that edge is stayed as it was in the beginning which is equal to zero.
- Other half of the Distance Matrix was filled according to the generated half. By this way, program guarantees that the graph is symmetric.
- Lastly, all the triangles inside the graph was spotted and Euclidean rule was broke by changing one of the edges.
- Distances Matrix was written to a text file for using it in XPress, Python and Java.

Exact TSP Solver Pseudocode

- All the declarations are done according to the model.
- One of the graph matrices was used as an 2D Array Input.
- The given problem was formulated.
- Then, all the nodes are travelled by using the edges to see that if there is a shorter path or not.
- After all the edges is travelled, program picks the shortest edge and prints it out with the route and total distance of the route.

TSP Heuristics Pseudocode

Pseudo code of Nearest Neighbor Heuristic:

Step 1. Initialize a distance variable for each node to zero.

Step 2. For each node start_node in graph:

- Mark start_node as visited, mark rest of the nodes as unvisited.
- current_node = start_node

Step 2.1. Look at all the arcs coming out of current_node. Choose the arc with the least length among the nodes that are unvisited:

- If such a node exists:
  - name the node as next_node
  - Increment distance of start_node by the length of arc between current_node and next_node
  - Mark next_node as visited
  - next_node = current node
  - Repeat **Step 2.1.**
- If no such node exists
  - If there is an unvisited node, distance of start_node equals to infinity.

Step 3. Choose the smallest distance.

Pseudo code of Minimum Spanning Tree Heuristic:

Step 1. Initialize a distance variable for each node to zero.

Step 2. For each node start_node in graph:

Step 2.1. Generate a minimum spanning tree starting from start_node according to prims algorithm [1].

Step 2.2. List vertices in the preorder walk of the constructed minimum spanning tree.

Step 2.3. Increment distance of start_node according to the preorder walk. Increment distance of start_node with the arc between last node of the preorder walk and start_node. If any of the arcs needed to form the cycle doesn't exist distance equals infinity.

Step 3. Choose the smallest distance.

Discussion & Results

Asymmetric Graphs

Nearest Neighbor Heuristic

Figure 1.1 shows that Nearest Neighbor algorithm gave a solution at most twice as the actual TSP solution.

It is clear Table 1 that Nearest Neighbor algorithm always gave a feasible sub optimal solution, according to our experiments, as long as the symmetrics graph itself had a feasible solution. The only time Nearest Neighbor algorithm failed to provide a solution was when the graph didn't have

any feasible solution itself (in experiment 9). However, this doesn't indicate that Nearest Neighbor will give a solution no matter what, in fact, it might use up all the neighbor nodes of a given node and might be stuck while there are still other nodes left unvisited, because of the greedy logic.
It still can be mentioned that this heuristic performs fairly well with Asymmetric graphs.

Minimum Spanning Tree Heuristic

It is observable from Table 1 that MST fails to provide a suboptimal solution in all experiments other than experiment 1. This can be supported by the fact that MST algorithm assumes triangle inequality, which doesn't necessarily be satisfied by a asymmetric graph. In fact, majority of asymmetric graphs don't satisfy triangle inequality.

Symmetric Graphs

Nearest Neighbor Heuristic

Figure 2.1 shows that Nearest Neighbor algorithm gave a solution at most twice as the actual TSP solution.
It can be seen from Table 2 that Nearest Neighbor algorithm failed to provide a solution when in fact there was a feasible solution (experiment 8). This can be explained by the greedy logic of the algorithm. By the time a node-i gets visited, all of its neighbors might have already been visited, and there still might be a unvisited node connected to other nodes, but not to that particular node-i. Therefore, greedy logic might fail to provide a solution.
As seen from both Figure 2.1 and Figure 2.2, Nearest Neighbor algorithm usually provides a solution with high precision given a symmetric algorithm, and even when it fails to do so, it still gives a solution within the boundary of 2 times the exact solution.
The fact that a symmetric graph has bidirectional edges with the same cost eases the Nearest Neighbor approach, and causes a better solution to be given by the greedy logic of the heuristic.

Minimum Spanning Tree Heuristic

It is observable from Table 2 that MST fails to provide a suboptimal solution in all but three experiments (experiments 10, 13, 18) when the given graph is symmetric. This can be supported by the fact that MST algorithm assumes triangle inequality to be satisfied for the arcs of the given graph, however, a symmetric graph doesn't necessarily satisfy the inequality. In fact, triangle inequality is not satisfied for all the nodes for the majority of graphs.

Euclidean Graphs

Nearest Neighbor Heuristic

Figure 3.1 shows that Nearest Neighbor algorithm gave a solution at most twice as the actual TSP solution. Furthermore, it can be seen from Figure 3.1 that for a noticeable amount of

experiments Nearest Neighbor algorithm provides a solution roughly as good as the exact solution.
In fact, for experiments 1, 10, 15, 16 Nearest Neighbor algorithm provided a solution where the exact algorithm couldn't provide any solution due to time or memory limitations.
As seen from both Figure 2.1 and Figure 2.2, Nearest Neighbor algorithm usually provides a solution with high precision given a symmetric algorithm, and even when it fails to do so, it still gives a solution within the boundary of 2 times the exact solution.
It can be seen from Figure 3.2 that the Nearest Neighbor algorithm provides a solution almost as good as the exact one for all of the experiments with euclidean graphs.

Minimum Spanning Tree Heuristic

It is observable from Table 3 that MST provides a solution for euclidean graph, each time there is a feasible solution. It can be explained by the fact that a euclidean graph has to satisfy triangle inequality for all of its arcs, and MST algorithm makes heavy use of this fact. Therefore, MST leans to solve euclidean graphs better than the non-euclidean counterparts.
It can be seen from Figure 3.1 that MST provides a solution withing 2 times the exact solution, and even provides solution for an experiment where the exact algorithm couldn't finish due to computational limitations.

Non-euclidean Graphs

Nearest Neighbor Heuristic

Figure 4.1 and 4.2 show the anomaly that happened in experiments 13 and 19: Nearest Neighbor algorithm provided a solution at least 40 times larger then the exact solution. For these situations, Nearest Neighbor algorithm cannot give an acceptable solution, even it gave some solution.
If we remove the outlier experiments and investigate the rest, iit can be observed from Table 4.3 that Nearest Neighbor algorithm provided solutions consistently almost as good as the exact algorithm.

Minimum Spanning Tree Heuristic

It is observable from Table 4 MST fails to provide a solution for all experiments except for experiment 9, which can be assumed an outlier.
This is easily explainable due to MST algorithm's assumption of triangle inequality for arcs.

Graph Types and Heuristics Discussion

It is observed from the data that Nearest Neighbor algorithm provides a more optimal solution than MST algorithm for almost all experiments for all graph types. MST is generally infeasible since most graphs don't satisfy triangle inequality.
MST algorithm favors euclidean graphs over non-euclidean ones since euclidean graphs satisfy triangle inequality.
After comparing Figure 1.2 to Figure 2.2 and Figure 3.2 to Figure 4.2/4.3, Nearest Neighbor algorithm also observed to favor symmetric graphs over asymmetric ones and euclidean graphs over non-euclidean ones, since solutions tend to fit into exact algorithm's solution more when the graph is either euclidean or symmetric or both.

Computation Limitations, Exact Algorithm versus Heuristics

Exact algorithm was stuck when a graph with approximately 90 nodes was given (Table 4 experiment 6). Therefore the experiments with symmetric and euclidean graphs had to be renewed with smaller graphs.
On the other hand, heuristics either gave a solution or quitted in a seconds for graphs with more than 100 nodes. This is explained by the fact that exact algorithm has O(n!) time complexity, where n is the number of nodes in graph, while heuristics provide a polynomial time result instead. Hence, number of nodes heavily limits the exact algorithm in finding a solution within a feasible time limit and within a feasible amount of memory.

Furthermore, exact algorithm was limited by the arc lengths as well, as we have observed in Table 4 experiment 16, Table 3 experiment 1 & 15 & 16. This is explained by the fact that larger integers complicate calculations.

Appendix

Asymmetric Graphs

Experiment	# Nodes	TSP	Nearest Neighbor	MST
1	6	1819	1997	3072
2	82	3293	5521	infeasible
3	28	3345	4890	infeasible
4	24	2922	4614	infeasible
5	76	3915	7710	infeasible
6	43	2981	5307	infeasible
7	64	3153	6081	infeasible
8	20	2793	4327	infeasible
9	6	infeasible	infeasible	infeasible
10	74	3252	5273	infeasible
11	86	2929	5623	infeasible
12	78	2888	5536	infeasible
13	72	3436	5482	infeasible
14	26	3944	5085	infeasible
15	16	2958	3873	infeasible
16	63	2799	5257	infeasible
17	41	3634	5289	infeasible
18	59	3370	4620	infeasible
19	86	3262	6544	infeasible
20	39	2775	4056	infeasible

Table 1. Minimum Total Distances of TSP & Nearest Neighbor & MST for asymmetric graphs

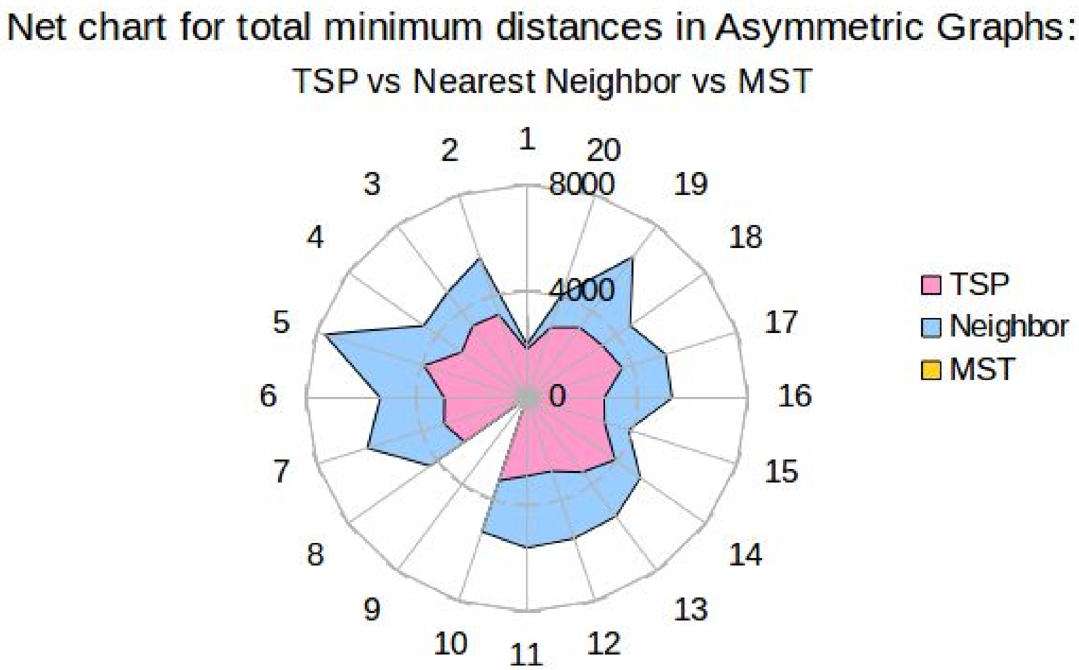

Figure 1.1 Minimum Total Distances of TSP & Nearest Neighbor & MST for asymmetric graphs

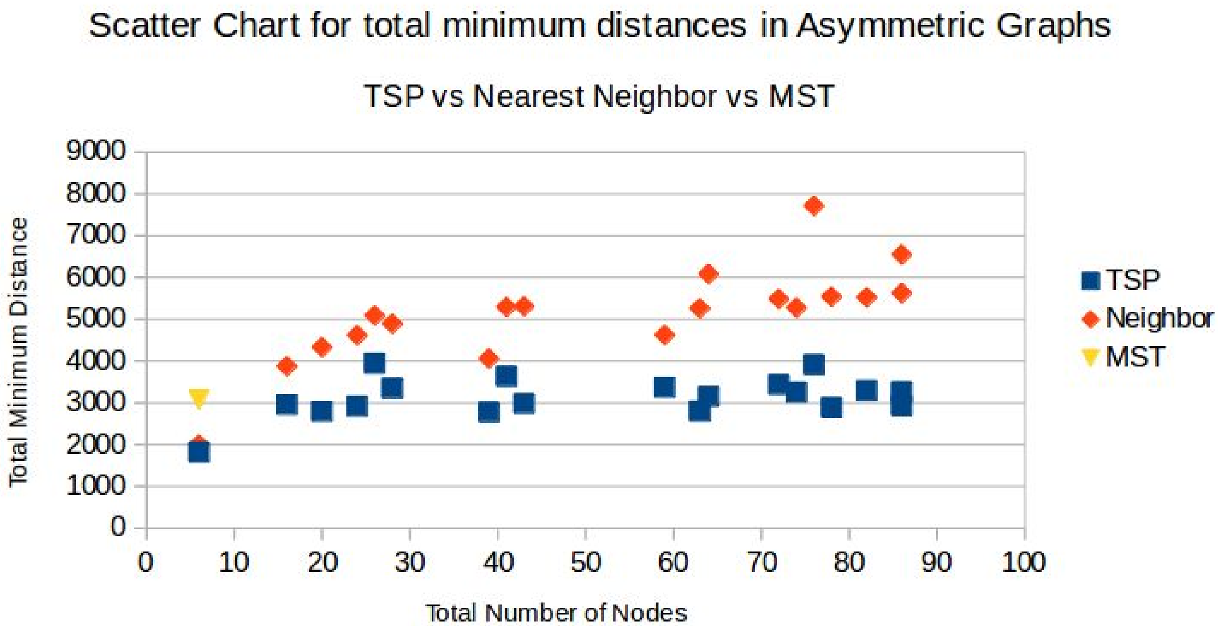

Figure 1.2 Minimum Total Distances of TSP & Nearest Neighbor & MST for asymmetric graphs

Symmetric Graphs

Experiment	# Nodes	TSP	Nearest Neighbor	MST
1	33	1156	1506	infeasible
2	20	863	946	infeasible
3	36	1063	1736	infeasible
4	38	1210	1685	infeasible
5	16	1033	1731	infeasible
6	28	1111	1614	infeasible
7	12	571	709	infeasible
8	23	1057	inf	infeasible
9	18	744	1105	infeasible
10	3	515	515	515
11	24	850	926	infeasible
12	34	1097	2021	infeasible
13	10	461	461	1263
14	13	611	611	infeasible
15	33	1017	1878	infeasible
16	4	infeasible	infeasible	infeasible
17	4	infeasible	infeasible	infeasible
18	7	389	395	425
19	39	1205	1491	infeasible
20	19	754	1077	infeasible

Table 2. Minimum Total Distances of TSP & Nearest Neighbor & MST for symmetric graphs

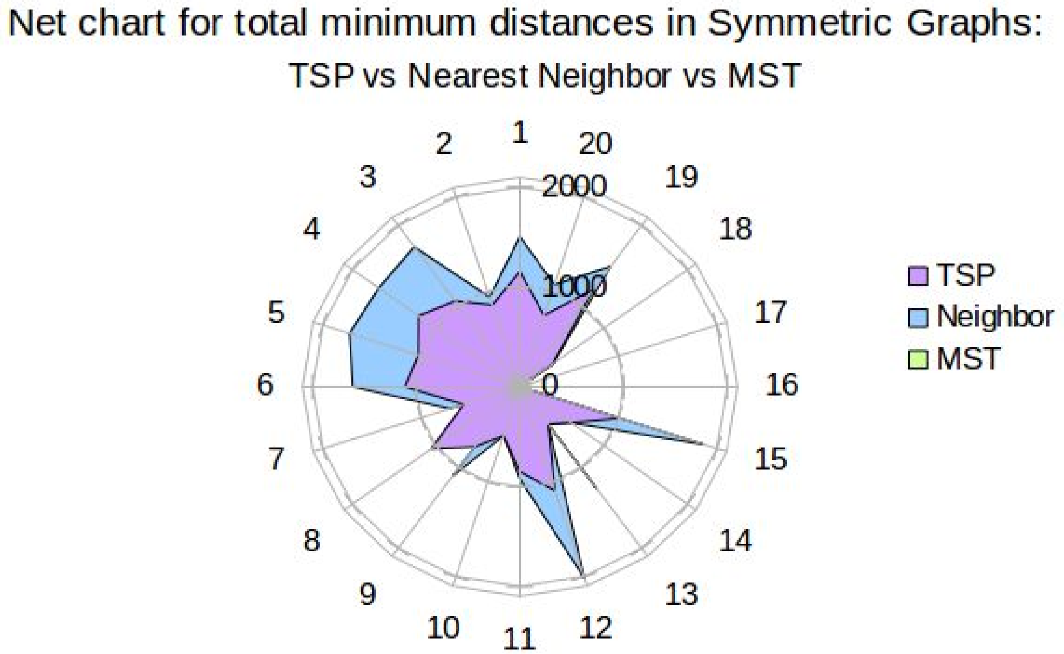

Figure 2.1 Minimum Total Distances of TSP & Nearest Neighbor & MST for symmetric graphs

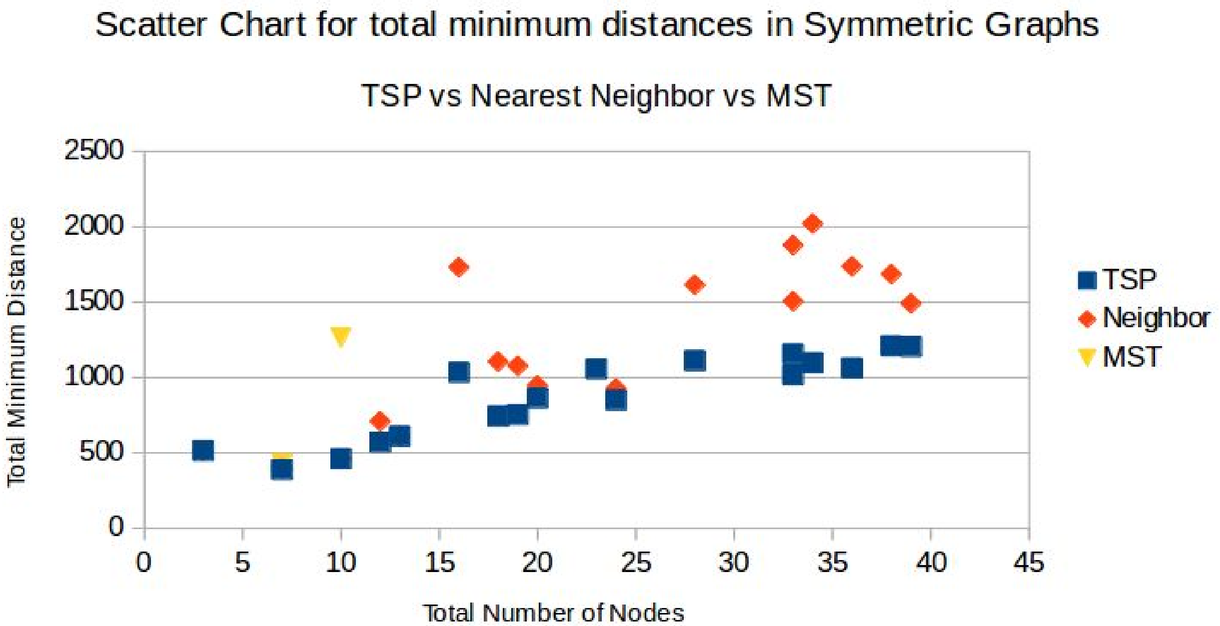

Figure 2.2 Minimum Total Distances of TSP & Nearest Neighbor & MST for symmetric graphs

Euclidean Graphs

Experiment	# Nodes	TSP	Nearest Neighbor	MST
1	19	infeasible	964	964
2	36	7036	7452	12554
3	11	982	982	1322
4	33	4356	4546	8445
5	2	164	164	164
6	10	2347	2347	3220
7	39	6588	7132	11673
8	18	3512	3619	5493
9	15	2731	2786	4059
10	38	infeasible	6914	11212
11	6	1468	1468	1886
12	35	6598	6844	11893
13	23	3158	3244	5070
14	16	3384	3580	4824
15	37	infeasible	986	1012
16	28	infeasible	964	1528
17	29	5296	5543	8626
18	37	6603	6794	10934
19	35	6279	6847	10841
20	9	2721	2805	3215

Table 3. Minimum Total Distances of TSP & Nearest Neighbor & MST for euclidean graphs

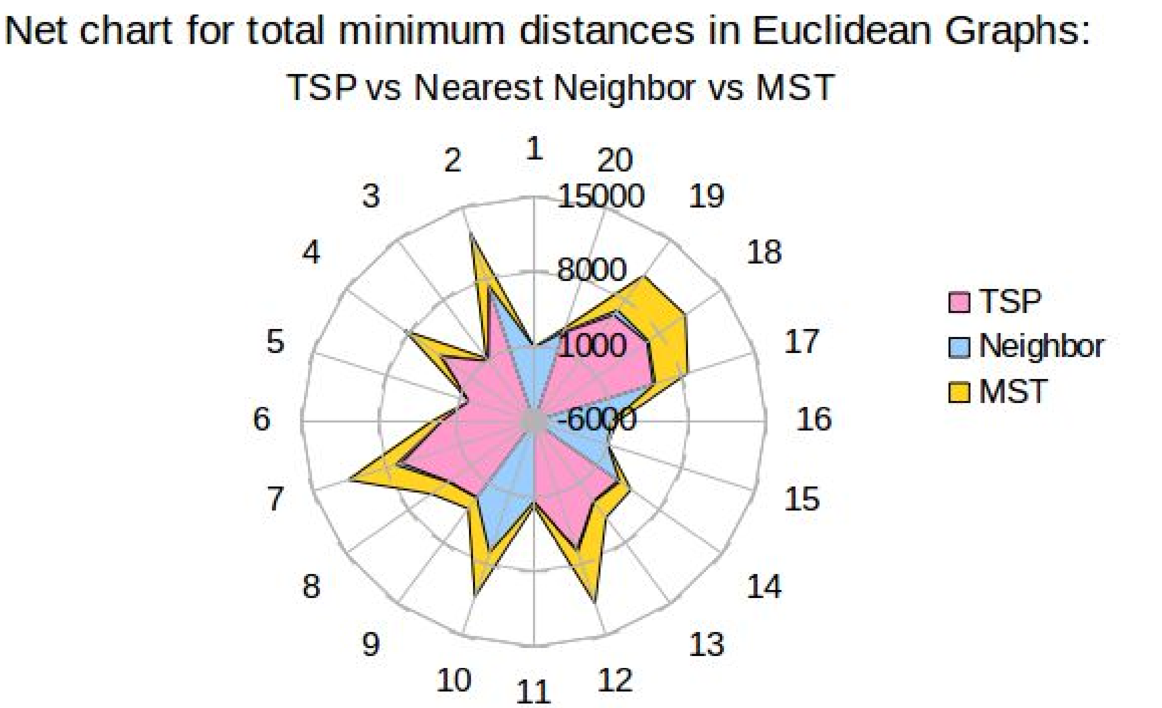

Figure 3.1 Minimum Total Distances of TSP & Nearest Neighbor & MST for euclidean graphs

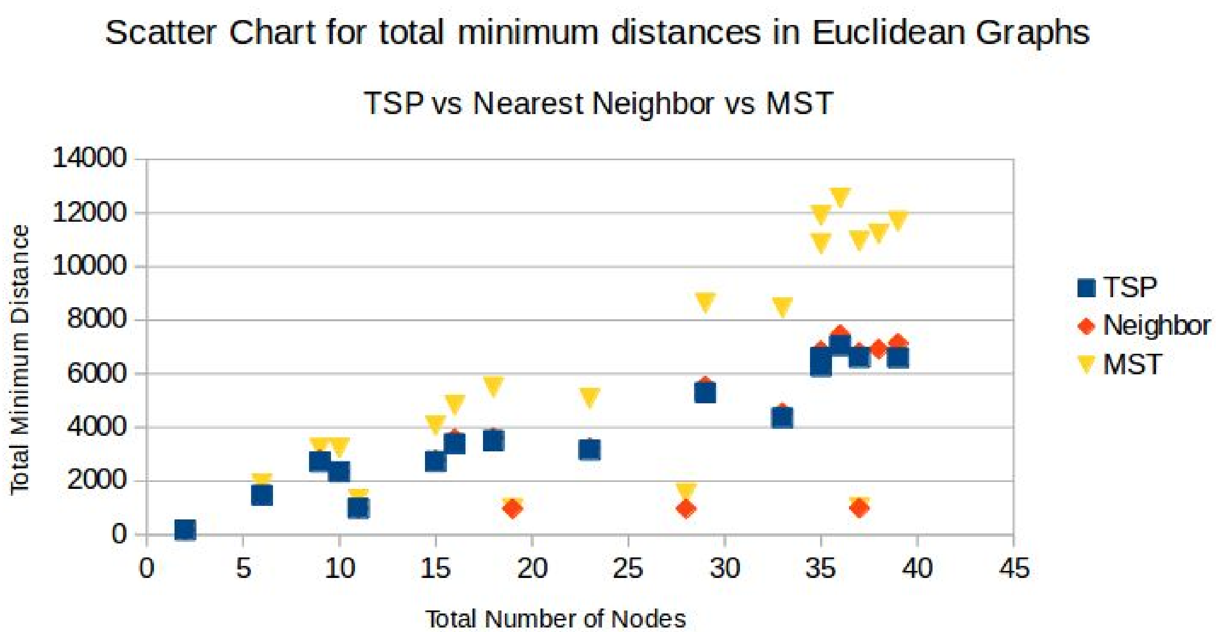

Figure 3.2 Minimum Total Distances of TSP & Nearest Neighbor & MST for euclidean graphs

Non-euclidean Graphs

Experiment	# Nodes	TSP	Nearest Neighbor	MST
1	39	18527	20869	infeasible
2	49	14910	16514	infeasible
3	48	14295	16896	infeasible
4	58	24504	26578	infeasible
5	7	8172	8630	infeasible
6	81	infeasible	infeasible	infeasible
7	15	9902	9968	infeasible
8	25	14339	infeasible	infeasible
9	6	7129	7129	10125
10	20	10726	11490	infeasible
11	51	9635	10981	infeasible
12	34	7092	9618	infeasible
13	49	2860	197786	infeasible
14	19	10143	11844	infeasible
15	60	9708	11865	infeasible
16	49	infeasible	3834	infeasible
17	62	17601	20838	infeasible
18	48	7964	10387	infeasible
19	43	9058	399177	infeasible
20	47	9041	9521	infeasible

Table 4. Minimum Total Distances of TSP & Nearest Neighbor & MST for non-euclidean graphs

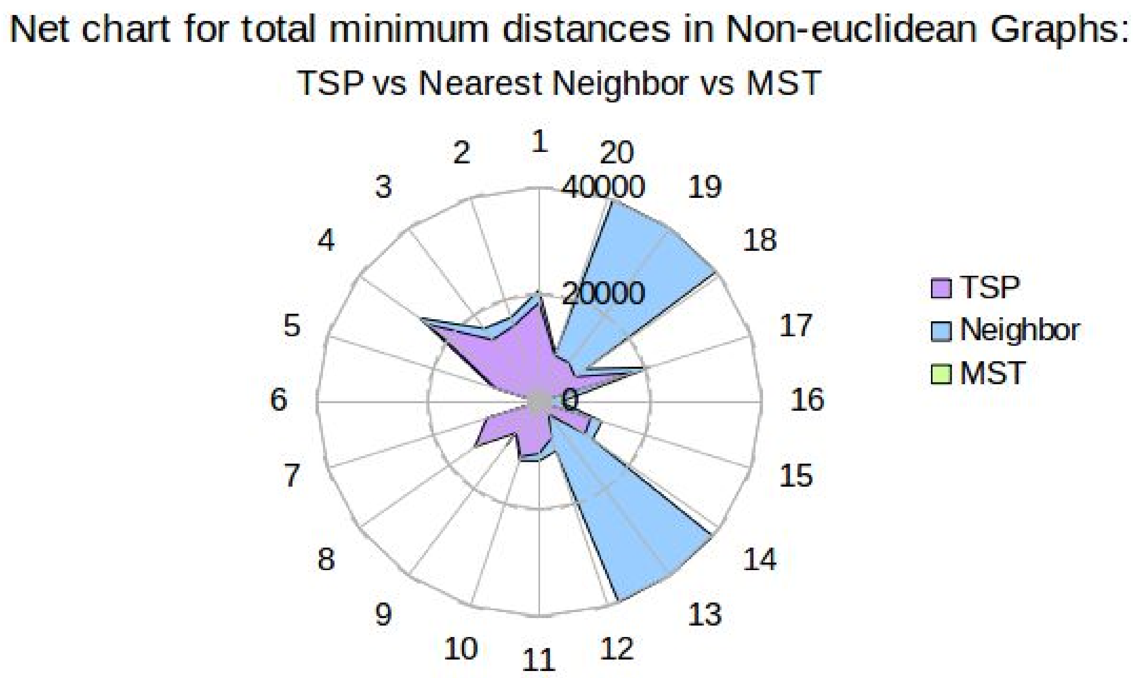

Figure 4.1 Minimum Total Distances of TSP & Nearest Neighbor & MST for non-euclidean graphs

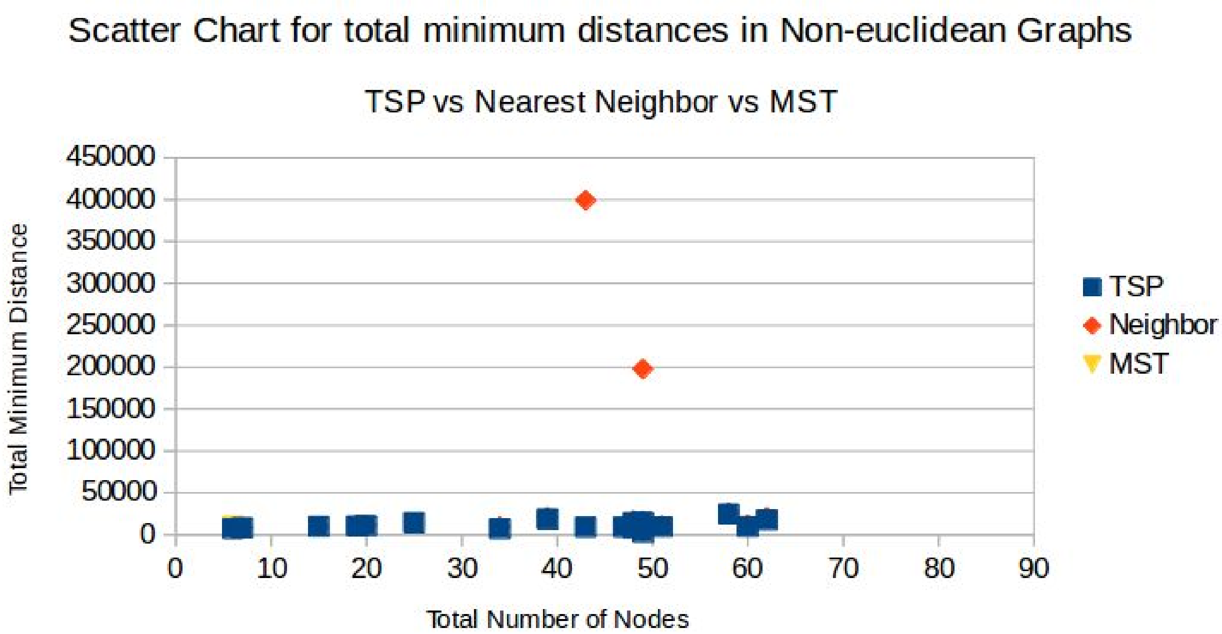

Figure 4.2 Minimum Total Distances of TSP & Nearest Neighbor & MST for non-euclidean graphs

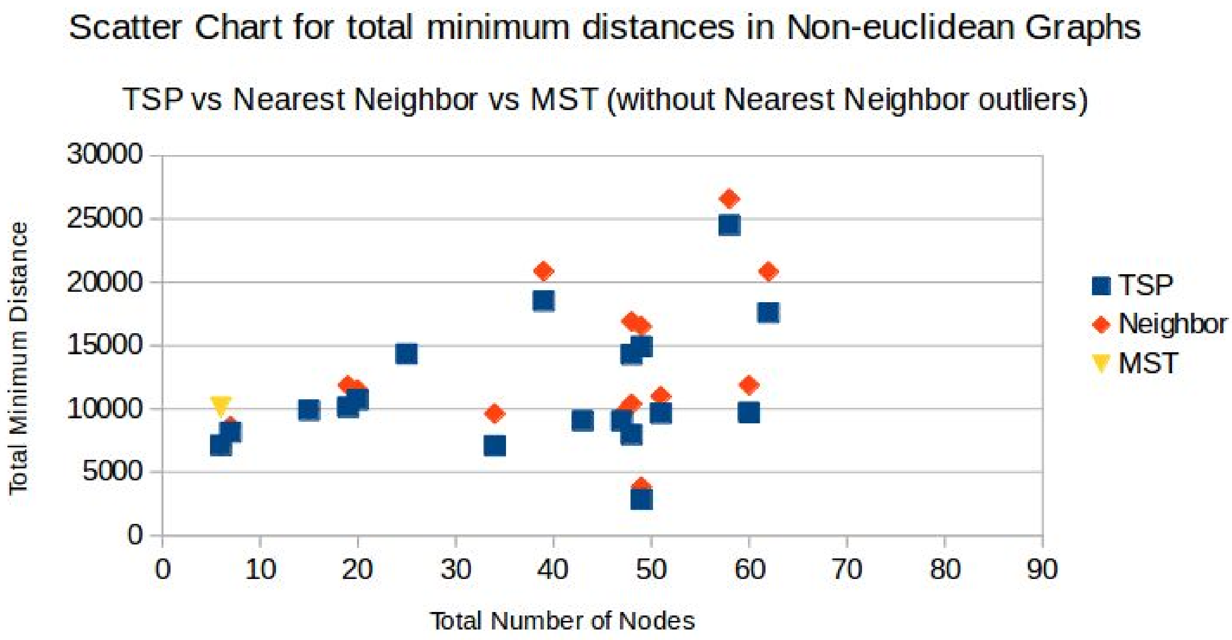

Figure 4.3 Minimum Total Distances of TSP & Nearest Neighbor & MST for non-euclidean graphs without Nearest Neighbor outliers

References

[1] https://www.geeksforgeeks.org/greedy-algorithms-set-5-prims-minimum-spanning-tree-mst-2/