Abhinav Modi
116346849

Homework-1

CMSC818B
September 18, 2019

Travelling Salesman Problem

The Travelling Salesman Problem is to find the shortest possible route that visits every city in a given list of cities and returning to the origin for the tour to be complete. It is an NP-hard problem in combinatorial optimization.

MST Algorithm

For this project I have followed MST algorithm which requires a weighted graph as an input and then tries to find a tour which visits each vertex at least once.

The basic approach is as follows:

1. Find the Minimum Spanning Tree(MST).
2. Traverse the edges in a depth-first fashion.
3. When going up the tree, skip an already visited vertex and add a shortcut to the next unvisited one.

Implementation

My implementation of the algorithm is in C++. The brief description of the implementation is as follows:

1. The information of the nodes is read from the '.tsp' files and stored in a vector of the format (int city_num,(xCord, yCord).

2. A graph which contains all possible edges(connections between the cities) is then generated and stored in a vector (edge_length(node_index1, node_index2)) and then this list is sorted according to the edge lengths.
3. This graph is then used to calculate MST. To generate the MST I start with an empty tree and add edges starting with the smallest one in the graph. Each time an edge is added in the MST, it is checked if a cycle is formed. These edges are then merged and stored in the same format as the graph was done in the previous step. This is done till all the edges have been looked at exactly once.
4. Once the MST is generated, Greedy algorithm is followed to traverse all the vertices in a depth-first fashion. Start a vector (tour) to store the output path(indices of nodes) and a stack to take care of the visited nodes. Start traversing in a depth first fashion

and keep appending vertices in both the vector and the stack till we reach a leaf node. At the leaf node since there is no possibility of going to a next node, go to the previous node and pop the first element in the stack. Using this as the current node find the next possible node and add it to the tour. Keep repeating this process until the tree is empty. Then add the start node again to the path to complete the cycle.

Results

Note: Screen capture showcasing how to run the code.

$Dataset$	$OptimalLength$	$Hueristic$	$MSTLength$	$Timetaken(s)$
*eil*51	426	641	376	0.0002
*eil*76	538	707	472	0.0003
*eil*101	629	842	542	0.0007
*input*100_1	–	1072	680.25	0.001
*input*100_2	–	1039	646	0.00098
*input*100_3	–	977	663	0.001
*input*100_4	–	1080	695	0.001
*input*100_5	–	1079	685	0.001
*input*100_6	–	996	692	0.001
*input*100_7	–	969	648	0.0008
*input*100_8	–	1048	681	0.0009
*input*100_9	–	1005	668	0.0009
*input*100_10	–	1050	669	0.001
*input*200_1	–	2835	1821	0.005
*input*200_2	–	2784	1889	0.004
*input*200_3	–	2907	1899	0.006
*input*200_4	–	2896	1908	0.005
*input*200_5	–	2783	1848	0.005
*input*200_6	–	2969	1903	0.005
*input*200_7	–	2978	1898	0.005
*input*200_8	–	2864	1933	0.004
*input*200_9	–	2786	1801	0.005
*input*200_10	–	2916	1898	0.0045
*input*300_1	–	3577	2320	0.105
*input*300_2	–	3625	2343	0.106
*input*300_3	–	3611	2322	0.108
*input*300_4	–	3548	2313	0.107
*input*300_5	–	3648	2344	0.110
*input*300_6	–	3576	2346	0.111
*input*300_7	–	3497	2304	0.113
*input*300_8	–	3617	2322	0.109
*input*300_9	–	3571	2271	0.108
*input*300_10	–	3376	2320	0.112

Table 1: Results for given Problem sets