**Travelling Salesman Problem**

**Problem Definition:**

Given a list of cities and the distance between each pair of cities, find the shortest possible route that visits each city only once and returns to the origin city.

TSP can be modeled as an undirected weighted graph, such that cities are the graph's vertices, paths are the graph's edges, and a path's distance is the edge's weight. It is a minimization problem starting and finishing at a specified vertex, after having visited each other vertex exactly once.

**Complexity Analysis:**

1. Exact Exponential Method:

Suppose there are n different vertices. Exact exponential method computes all possible formulation of these vertices. Now a tour starts from vertex 0 and ends at the same vertex. So excluding the first vertex we are left with n-1 vertices. Counting all possible permutation of n-1 vertices takes (n-1)! time.

So the time complexity is O((n-1)!)

1. Branch and Bounding Method:

In the worst case scenario the time complexity of branch and bound is equal to the exponential method. O(n-1)! In the worst case, there may arise a scenario where no sub tree can be pruned. So the algorithm will traverse the full tree which is n-1 different permutation as mentioned above. So in worst case Branch and Bound takes O((n-1)!) time. In practice the algorithm performs very well depending on the difference instance of TSP.

For the best case, the first sub tree traversed is the solution and all the other sub trees are pruned. This takes for n vertices O(n) time.

1. Greedy 2-Approximation Algorithm:

I have used Kruskal’s Algorithm to implement this method. This gives a Minimum Spanning Tree (MST) as an output. Then to determine the Hamiltonian cycle Depth First Search (DFS) is done on the MST.

As I used adjacency matrix to represent the graph, for n vertices DFS take O(n^2) time. However, if m is the number of edges and n is the number of nodes, in Kruskal

* Sorting the edges : O(m log m)
* For each edge, calling Union Find: O(m \* alpha(n)) or basically just O(m)
* Total complexity for Kruskal : O(m log m + m \* alpha(n))

Total time complexity: O(m log m + m \* alpha(n) + n^2) where alpha(n) is the inverse of the Ackermann function, and, for all intents and purposes, can be considered constant.

For all three methods adjacency matrix is used to represent the graph. So the space complexity is O(n^2) for n vertices.

**Test Case Design:**

**Input:**

First line contains one integer which denotes the number of nodes.

From second line to EOF, each line contains x and y coordinate of a node. Both of

them are float values.

**Output**:

Output is contains two parts, one in console and another in file. Console output contains the number of nodes in the first line. Then the name of the algorithm and the Hamiltonian tour configuration. Next line contains the total cost of the cycle. One file contains the tuple in each line.

* Number of nodes
* Time needed for Exact Exponential
* Time needed for Branch and Bound
* Time needed for Greedy Approximation
* Approximation Ratio for Greedy Approximation

Charts: