# Group Pre-report:

# Traveling Salesperson Problem Analysis

**Team members:**

*Tong Wu, Xun Tang, Xinyu Lyu*

## 1.Definition

The TSP is a famous and popular mathematics problem that asks for the most efficient trajectory possible given a set of points and distances that must all be visited. To be more concrete, this problem describes a salesman who must travel between N cities, where distances between all the cities are known and each city should be visited just once, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.

Believe it or not, the problem actually manifests itself quite clearly in everyday life. Have you ever imagined that how do the delivery trucks bring you the fresh produce from the store? Or how do the school buses pick up and drop off your children? These are all about TSP, moving goods and providing services from one place to another, and another.

The TSP has several applications in many different areas either in its purest formulation or in its variants. In these applications, the concept of the city represents, for example, customers, or DNA fragments, the concept of distance represents traveling times or cost, or any other format.

In computer science, the problem can be applied to the most efficient route for data to travel between various nodes.

The TSP is typical of large class of NP-Hard or NP-Complete optimization problems that have intrigued mathematicians and computer scientists for years and it is used as a benchmark for many optimization methods. Even though the problem is computationally difficult, there still exists a large number of heuristics and exact algorithm, which can solve some instances with tens of thousands of cities completely.

The easiest and most expensive solution is the brute-force approach, which is to evaluate every possible tour and select the best one. For N vertices in a graph, there are (N-1)! numbers of possibilities. This will result in tremendous possibilities even if the number of vertices is not that large. So, in the following content, we will give several algorithms which can optimize this problem.

## 2. The Available algorithms

(1) The Greedy algorithm

The greedy algorithm is a simple and fast algorithm that is commonly used to solve optimization problems. It always makes the best/greedy choice at present and hopes to get the optimal solution to the problem based on each greedy choice. More specifically, it is to traverse all reachable nodes based on the current node and select the nearest node as the next node. The basic idea is to traverse all reachable next nodes from a node.

Select the nearest node as the next node, then mark the current node has been passed, and set the next node as the current node. Repeat steps above, until all nodes are marked as passed.

(2) The Hill-climbing algorithm

Hill-climbing algorithm is a locally preferred method. It is an improvement of depth-first search. It uses feedback information to help generate the solution. Each time the algorithm chooses an optimal solution from the solution space of the current solution until it reaches a locally optimal solution.

(3) The Simulated annealing algorithm

Simulated annealing is a random search algorithm based on the principle of annealing. It differs from the hill-climbing method. It selects the next node based on the difference after choosing the node with a certain probability. When the current solution is increased based on selecting the neighbor node, it will accept the improved solution as a new current solution. Contradictory, when compared with the current solution, the worse solution will also be accepted as a new current solution with a certain probability. The simulated annealing algorithm has been theoretically proved to be a global optimization algorithm that converges to the global optimal solution with a probability of 1.

(4) The Dynamic programming algorithm

Its basic idea is to decompose the problem to several sub-problems. First, solve the sub-problems, and then obtain the solution of the original problem from the solutions of these sub-problems.

Let V' denote a set of nodes. Assuming starting from the node s, d (i, V') represents the minimum cost for the current arrival of node i through all nodes in the V' set.

(i) When V’ is an empty set, d(i, V’) means that it returns to s from node i without going through any node, as shown in the figure above. In this case d(i, V’)=Cis (is the distance from node i to node s).

(ii) If V' is not empty, it is the optimal solution to the subproblem. You must try each one in the V’ set of cities and find the optimal solution--d(i, V’)=min{Cik + d(k, V’-{k})}. Cik represents the distance between your chosen node and node i. d(k, V’-{k}) is a sub-problem.

## 3. Performance Analysis

As the algorithms mentioned above, we are going to analyze and compare their performances. Mainly analyze the order of growth of running cost on average, best and worst cases. As for now, we can use C++ or Java as our programming language.