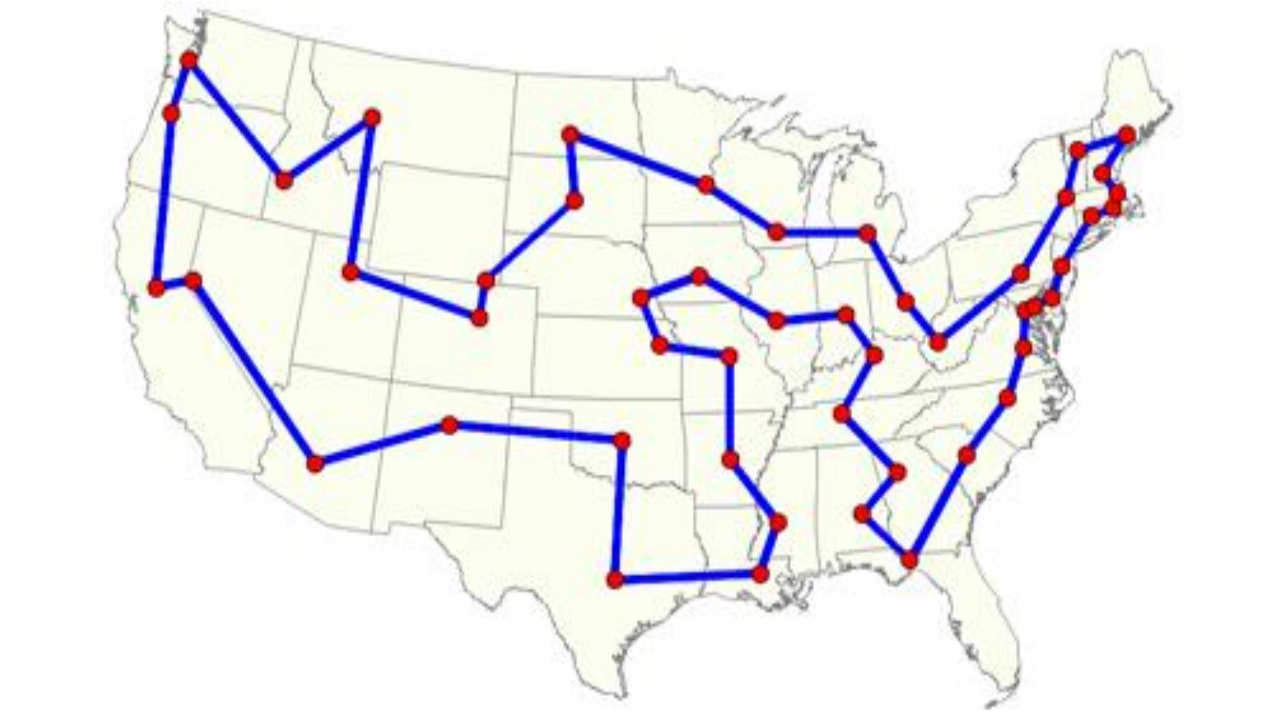
Middlesex University London



CST3170

Artificial Intelligence

Student ID- **M00735429**

Student Name- **Manjot Kaur**

Travelling Salesman Problem

Points	Area	
10	Self-Marking Sheet	10
10	Solve First Training Problem	10
10	Get Optimal Results for All Three Training Problems.	10
10	Describe Algorithm(s) Used	6
10	Quality of code	6
20	Get Optimal Results for the First Three Tests	20
20	Get Optimal Results for the First Three Tests in under a minute.	20
10	Best system on Fourth Test (Path length times time.)	0

INTRODUCTION –

In the Travelling salesman problem, different algorithms are used to find the distance between the paths, but the distance is calculated using the same Euclidean distance formula.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

EUCLIDEAN FORMULA

GENETIC ALGORITHM-

The genetic algorithm has proven to be the most successful. The algorithm gives optimal results within one minute for all train problems and the first three tests.

STEPS-

The process begins by creating a collection of fifty genes containing randomised indexes of possible paths and whose length equals the number of data points. Genes are chosen from a generation of genes and the best gene is chosen as the fittest gene. The leading gene has the shortest distance in the generation. It is the same fittest gene that produces fifty offspring, each of whom has parts from the same gene. There is a 15% probability that each index in an offspring will be mutated. A mutation is a swap between indexes.

RESULTS-

Hence, this algorithm is suitable for many cities, however, after 25 or 28 cities it may not give the optimal results. However, the results varied in the fourth test file (but not significantly) since there were 27 cities involved. It works perfectly for the first three tests and gives optimal results.

PERMUTATION ALGORITHM –

STEPS-

The cycle to all cities is represented by a sequence of consecutive integers. Calculating all the permutations of the paths using recursion.

RESULTS-

Considering there are n-factorial possibilities after 10 cities, the permutation method does not work. Arrays that grow beyond their maximum size will eventually exceed Java's heap space limit, meaning that Java will eventually run out of space while searching for all possible paths.

BRANCH AND BOUND ALGORITHM -

STEPS-

Branching breaks up the set of all tours (possible solutions) into successively smaller subsets. Using branch and bound algorithm recursively. If the non-visited list is empty and not visited, add the first vertex from the non-visited list into the path. If the path is empty, add the temporary path to the non-visited list. Replace the shortest distance with the saved distance if the path is complete.

RESULTS-

Despite this, this algorithm does not work well for large numbers of cities. However, for traveling salesman problems with 14 or fewer cities, it provides the most optimal solution.

NEAREST NEIGHBOUR ALGORITHM -

Begin with a randomly selected city and continue visiting the nearest cities until they are all visited.

STEPS-

Initialize all vertices as unvisited. Choose a random vertex and set it as the current vertex. Find the shortest edge between the current vertex and an unvisited vertex. Set the unvisited vertex as the current vertex and mark it as visited. The process terminates when all the vertices have been visited.

RESULTS-

Even though the nearest neighbour algorithm is fast, it may not always find the shortest path since it is not a reliable method of finding the shortest path. While it works for many cities, it doesn't offer optimal solutions in all situations.

2-OPT ALGORITHM -

STEPS-

Reconnects the two paths created by removing two edges from the path. To create a valid tour, that can connect the two paths. This can be done by the new shortest tour. Remove and reconnect the tour until no 2-opt improvements can be found. The tour is now 2-optimal.

RESULTS-

Consequently, the 2-opt algorithm works fine for all the train problems and test results. Yet, sometimes, it does not provide the optimal results for a particular dataset. Which is like 2% less optimal than the actual optimal path and distance.