Rémi Wong

Traveling Salesman Problem

2019

Contents

Introduction

In this coursework, an algorithm based on java programming language has been devised to solve the traveling salesman problem (TSP) where the salesman begins his journey at any particular city and visits all the other cities only once. At the end of his journey upon reaching the last city, he goes back to the initial city. The nearest neighbour algorithm was chosen as the possible solution, then the latter was improved with the concept of Mexican Hat algorithm to try to optimise the TSP issue.

Problem Definition

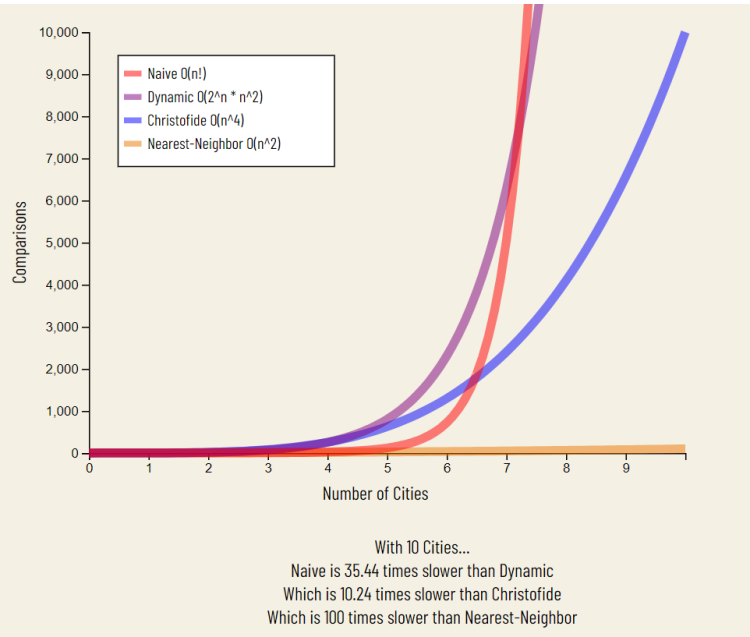
- The salesman can start from any city.
- A city is visited only once.
- The shortest possible path that can be utilised needs to be investigated.
- The algorithm needs to produce optimal results in terms of time and distance travelled.
- All cities datasets need to be processed in less than a minute.

Solution

The algorithm designed is based on two concepts

1. Nearest Neighbour
2. Mexican Hat

Nearest neighbour was chosen as the base of the algorithm because it is seen as one of the fastest travelling salesman problem algorithms with a time complexity of $O(n^2)$, thus performs exceptionally well in terms of time efficiency as the amount of cities to be visited is greatly increased (Roy Mathew, 2019). First the native version of neighbour has been investigated, then it was improved with the use of Mexican Hat algorithm to try reduce the shortest path undertaken by the salesman.



The distance between any two cities is determined using the Euclidean distance which is represented below.

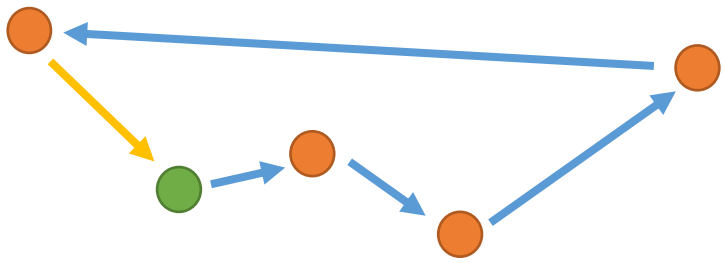
$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

The points; p and q are the two cities being compared and the difference in unit distance in the horizontal x-axis and the vertical y-axis is squared and summed. Then the value obtained is square rooted to find the distance between the two points.

The nearest neighbour algorithm selects a starting city randomly and determines the next city to go by comparing the possible distance that needs to be travelled towards the remaining cities. The smallest distance out of them is chosen as the ideal next city. The next process is to use the next city as the reference city which is first marked as visited so that the salesman does not consider it again. The nearest city's distance is again found. The process is repeated until all cities are visited and the salesman's last trip would be to go back to his initial starting city.

In the solution designed, instead of randomising the starting city, all cities are considered as possible starting cities to give the best possible shortest path. A diagram is shown below to show the nearest neighbour thinking process where city Green is chosen as the starting city.



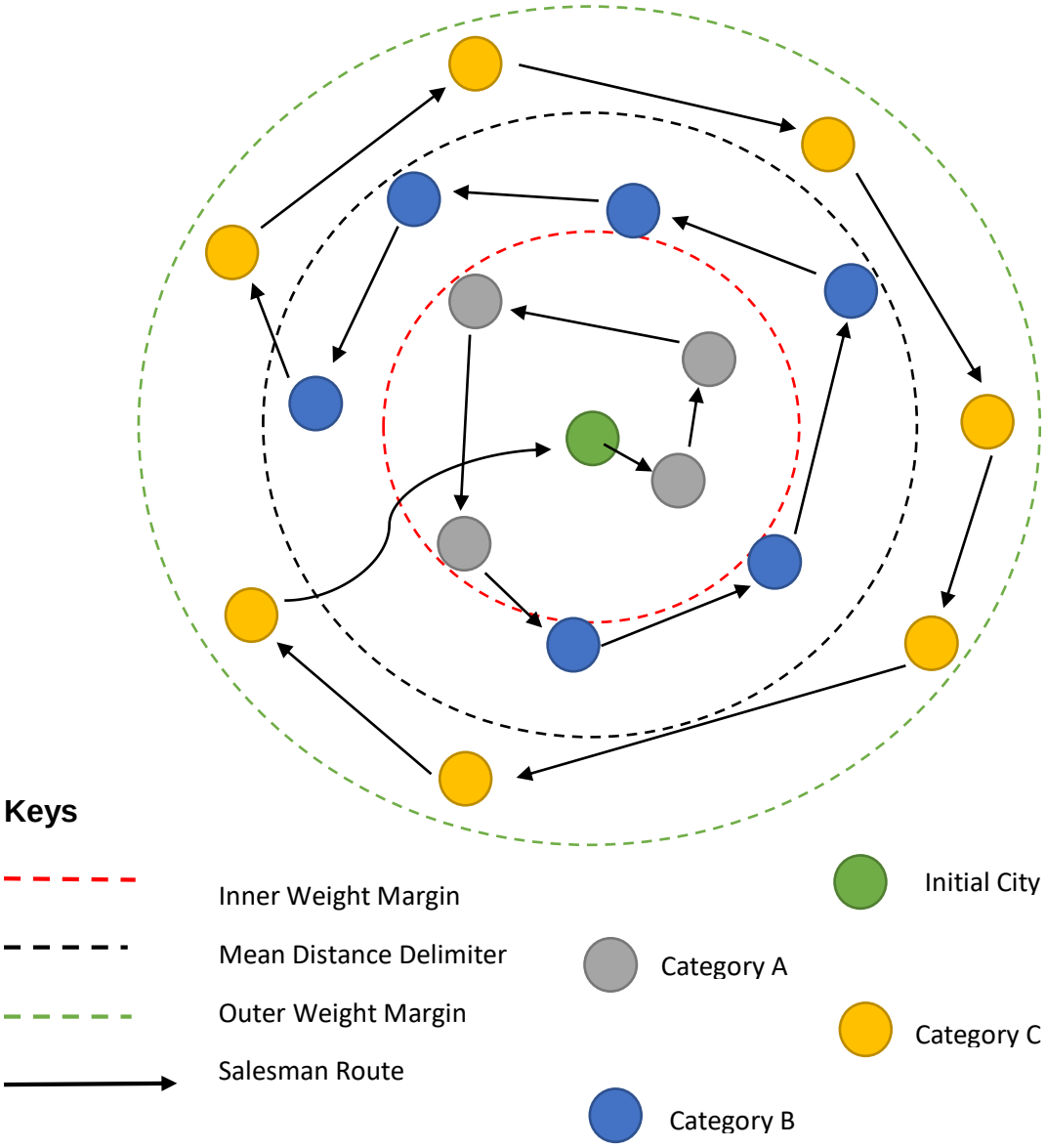
It was found that the solution was performing adequately but that the use of the Mexican Hat concept could be utilised to potential decrease the shortest path results obtained under certain conditions. With this improvement, the cities have been classified in three categories A, B, and C before the nearest neighbour algorithm kicks in. Instead of considering all remaining unvisited cities as next destination, the cities marked as A are first candidates, then B, and last C. The categories are determined based on the inner and outer weights around the chosen initial city that the salesman starts his journey. The weights are first valued on the mean distance between the starting city and the unvisited cities using the same Euclidean distance formula. The mean would represent the circular path around the starting city where the categoriser decides; either A/B, B/C. Initially it would be A/C since the weights have a learning rate which is the ratio, n of the difference between the furthest city and closest city from the starting city to the desired number. The inner weight is decremented with the learning rate and the outer weight is incremented respectively. The formula is shown below. The value of learning rate is iterated to have different weights lengths, thus different outputs.

$$Learning\ Rate = \frac{(furthest\ city - \ closest\ city)}{\mathrm{n}}$$

$$Inner\ Weight = Mean\ distance - Learning\ Rate$$

$$Outer\ Weight = \ Mean\ distance + Learning\ Rate$$

A graphical representation of the logical path of the improved algorithm is shown below. The example shows the idea conditions which should occur to obtain the most accurate results for the shortest path.



Program Structure

TSP: Main Program for Improved TSP Algorithm

TSPNN: Old version of the TSP algorithm using only Nearest Neighbour

FileParser: Loading of Cities from text file

City: Object Constructor Class for Cities

CityCoordinates: Constructor class for city coordinates x and y

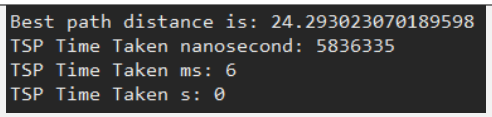
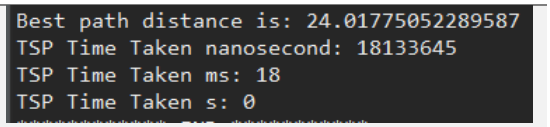
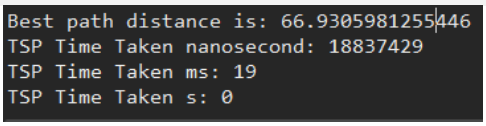
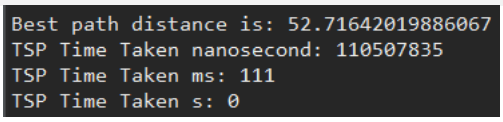
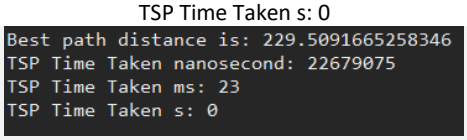
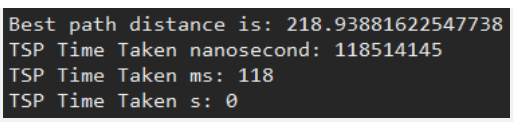
EuclideanDistance: Class called to called to calculate distance between two cities

Operating Instructions

1. Run the TSP class as a java application on Eclipse IDE
2. Insert the path and name of the text file where the dataset is stored
3. Validate file by pressing enter. If the File is not found, a file not found exception will be displayed, else if the dataset is not properly written in the text file, an error for city points will be shown.
4. Please note that before running the application, the console display limit on the preference settings of Eclipse need to be de-activated to see all program outputs after the program termination.

Training Set Results

After several executions of the program using the training datasets and the test datasets, the ratio at which the learning rate was set to 8 because it was found give the best shortest path during the different trials. Screenshots of the results have also been provided for the execution of each dataset.

DATASET	UNIMPROVED NEAREST NEIGHBOUR	IMPROVED ALGORITHM
TRAINING PROBLEM 1	 Best path distance is: 24.29 Units TSP Time Taken ns: 5836335 TSP Time Taken ms: 6 TSP Time Taken s: 0	 Best path distance is: 24.02 Units TSP Time Taken ns: 18133645 TSP Time Taken ms: 18 TSP Time Taken s: 0
TRAINING PROBLEM 2	 Best path distance is: 66.93 Units TSP Time Taken ns: 18837429 TSP Time Taken ms: 19 TSP Time Taken s: 0	 Best path distance is: 52.72 Units TSP Time Taken ns: 110507835 TSP Time Taken ms: 111 TSP Time Taken s: 0
TRAINING PROBLEM 3	 Best path distance is: 229.51 Units TSP Time Taken ns: 22679075 TSP Time Taken ms: 23 TSP Time Taken s: 0	 Best path distance is: 218.94 Units TSP Time Taken ns: 118514145 TSP Time Taken ms: 118 TSP Time Taken s: 0

The order of the best route is displayed in the console of Eclipse based on the ranking of each listed city.

Test Set Results

The same unmodified codes were used for the execution of the program for the provided test datasets for different cities. The results are shown below same as with training sets.

DATASET	UNIMPROVED NEAREST NEIGHBOUR	IMPROVED ALGORITHM
FINAL TEST 1	Best path distance is: 11269.38703536242 TSP Time Taken nanosecond: 30197338 TSP Time Taken ms: 31 TSP Time Taken s: 0 Best path distance is: 11269.39 Units TSP Time Taken ns: 30197338 TSP Time Taken ms: 31 TSP Time Taken s: 0	Best path distance is: 11383.982537117512 TSP Time Taken nanosecond: 132802041 TSP Time Taken ms: 133 TSP Time Taken s: 0 Best path distance is: 11383.98 Units TSP Time Taken ns: 132802041 TSP Time Taken ms: 133 TSP Time Taken s: 0
FINAL TEST 2	Best path distance is: 208.5004992857619 TSP Time Taken nanosecond: 92192455 TSP Time Taken ms: 92 TSP Time Taken s: 0 Best path distance is: 208.50 Units TSP Time Taken ns: 92192455 TSP Time Taken ms: 92 TSP Time Taken s: 0	Best path distance is: 202.39392979514503 TSP Time Taken nanosecond: 164612719 TSP Time Taken ms: 165 TSP Time Taken s: 0 Best path distance is: 202.39 Units TSP Time Taken ns: 164612719 TSP Time Taken ms: 165 TSP Time Taken s: 0
FINAL TEST 3	Best path distance is: 354.23591345646764 TSP Time Taken nanosecond: 103643818 TSP Time Taken ms: 104 TSP Time Taken s: 0 Best path distance is: 354.24 Units TSP Time Taken ns: 103643818 TSP Time Taken ms: 104 TSP Time Taken s: 0	Best path distance is: 326.94995053767013 TSP Time Taken nanosecond: 313425329 TSP Time Taken ms: 313 TSP Time Taken s: 0 Best path distance is: 326.95 Units TSP Time Taken ns: 313425329 TSP Time Taken ms: 313 TSP Time Taken s: 0
FINAL TEST 4	Best path distance is: 39236.884898455035 TSP Time Taken nanosecond: 1122320323 TSP Time Taken ms: 1122 TSP Time Taken s: 1 Best path distance is: 39236.88 Units TSP Time Taken ns: 1122320323 TSP Time Taken ms: 1122 TSP Time Taken s: 1	Best path distance is: 38568.948841580735 TSP Time Taken nanosecond: 7616331419 TSP Time Taken ms: 7616 TSP Time Taken s: 7 Best path distance is: 38568.95 Units TSP Time Taken ns: 7616331419 TSP Time Taken ms: 7616 TSP Time Taken s: 7

Conclusion

After the implementation and testing of the traveling salesman problem algorithm, it can be seen that the nearest neighbour algorithm generally performs very fast. The improved version consumes more time to go through more loops but outputs a better route for the salesman depending on the location of all the cities. For the largest dataset, it can be seen that the improvement brought to the nearest neighbour did not produce significant difference in the shortest distance covered. Hence in terms of efficiency of time against distance of TSP solution, the nearest neighbour algorithm is better for a very large set of cities.

References

Roy Mathew, D. C. K. P. K. Z., 2019. *Traveling Salesman Algorithm.* [Online]
Available at: https://cse442-17f.github.io/Traveling-Salesman-Algorithms/
[Accessed 13 December 2019].