Faculty of Engineering and Natural Sciences

Department of Computer Engineering

CMP3004 - Formal Languages and Automata Theory Term Project Report

Onur Güzel
Anıl Tekli
Mert Yerlikaya

June 13, 2020

Abstract

The traveling salesman problem which is also known as TSP, was mathematically formulated in the 1800s by the Irish mathematician W.R. Hamilton and by the British mathematician Thomas Kirkman. The objective of the problem is, given an array of cities and their locations, to find the shortest possible path which passes through each city once. There are two primary forms of the problem. One of them is the graph-based version, where the cities are given as a structure of a weighted graph which illustrates the connections between cities, the problem is to be solved within the restrictions on the order of the cities should be visited in. The other form of the traveling salesman problem is the Euclidean version, where cities are given as coordinates on a two-dimensional plane and the salesman can moves from one city to any other. In this form of the problem, distances between cities are calculated using the Euclidean distance formula. For this project, we implement solutions the Euclidean version of the problem using different techniques.

Contents

List of Figures

1 Responsibilities

1.1 Responsibility Matrix

Code	Task	Onur G.	Anıl T.	Mert Y.
I.	Nearesr Neighbour Algorithm	R	-	-
II.	Opt-Nearesr Neighbour Algorithm	S	R	-
III.	Greedy Algorithm	S	-	R
IV.	Divide and Conquer Algorithm	-	-	R
V	Tour Visualization	R	-	-
VI.	Planning Responsible	R	S	S
VII.	Research Responsible	R	R	R
VIII.	Documentation Responsible	R	S	S

R: Responsible S: Supportive

Figure 1: Responsibility Matrix

1.2 Responsibilities List

- Nearest Neighbour Algorithm Implementation – Onur Güzel
- Opt-Nearest Neighbour Algorithm Implementation – Anıl Tekli
- Greedy Algorithm Implementation – Mert Yerlikaya
- Divide and Conquer Algorithm Implementation – Mert Yerlikaya
- Tour Visualization – Onur Güzel
- Planing Responsible – Onur Güzel – Anıl Tekli - Mert Yerlikaya
- Research Responsible – Onur Güzel – Anıl Tekli – Mert Yerlikaya
- Documentation Responsible – Onur Güzel

2 Introduction

Algorithms for optimizing the traveling salesman problem, are usually based on re-construction of a tour, improvement upon an existing tour, or a combination of both. Algorithms, main objective is to construct a short tour from the beginning. There are variations of this approach, one of them is to construct a short tour in the first try by finding the best order of cities as it runs. Another approach is to construct multiple random tours either or based on some logic and use the one with the best solution of all. One the common example of this is the brute force algorithm, which computes every possible solution by checking all permutations of the cities and opts the best one. This method will find the optimal tour, however brute force algorithms are known for their poor running time. Optimization algorithms usually takes an existing tour and perform various operations which opts the best total tour distance.

3 Nearest Neighbour Algorithm

The principle behind this algorithm is simple. It starts by constructing a tour from the list of all the cities and their coordinates. The starting city is chosen randomly and moved to the top of the list of cities representing the solution. Then, out of the remaining cities, the city nearest in distance to the starting city is moved to the solution list. This operation is repeated until all cities have been moved to the solution.

3.1 Algorithm Output

Algorithm output is shown in the figure below;

```
 ------------------------------------
| Nearest Neighbour Algorithm Solution |
 ------------------------------------

Run Time: 0.009644

Shortest Path Distance: 35227

Shortest Path is:

0--8--14--11--10--22--39--2--21--15--40--28--1--41--25--3--34--44--9
--23--31--38--47--4--33--13--24--12--20--46--19--32--45--35--29--42-
-16--26--18--36--5--27--6--17--43--30--37--7--
```

Figure 2: "Nearest Neighbour Algorithm" Output

4 Opt-Nearest Neighbour Algorithm

This algorithm is the optimized version of the Nearest Neighbor Algorithm in order to find the shortest path possible. It takes the shortest path distance from the user as a parameter and generates tours until the distance is smaller than the distance provided by the user. Algorithm generates random tours using Nearest Neighbor algorithm and stops the generation until the path distance requirements meets the parameter given by the user, in this sense this algorithm is a mesh of Nearest Neighbor and Brute Force approach.

4.1 Algorithm Output

```
 ----------------------------------------
| Nearest Neigbour Algorithm Exact Solution |
 ----------------------------------------

Run time is: 10.5959

The shortest path distance is: 33523

The shortest path is:

33--28--1--25--3--34--44--9--23--41--4--47--38--31--20--46--19--32-
-45--35--29--42--16--26--18--36--5--27--6--17--43--30--37--7--0--8-
-39--14--11--10--12--24--13--22--2--21--15--40--
```

Figure 3: "Opt-Nearest Neighbour Algorithm" Output

5 Greedy Algorithm

The principle behind the Greedy Algorithms is pretty simple, basically algorithm picks a node and starts building a three structure. In the next iterations algorithm picks the lowest cost edge leaving that node. This operation is repeated until the algorithm constructs a full three, in the process is does not account for the edges that cannot yield a Hamiltonian cycle.

5.1 Algorithm Output

```
 --------------------------
| Greedy Algorithm Solution |
 --------------------------

Run Time: 0.010548

Shortest Path Distance: 34299

Shortest Path is:

0--8--37--30--43--17--6--27--5--36--18--26--16--42--29--35--45--14--39
--11--32--19--10--22--13--24--12--46--20--31--38--47--4--41--23--9--44
--34--3--25--1--28--40--33--2--21--15--7--
```

Figure 4: "Greedy Algorithm" Output

6 Divide and Conquer

As its name states, in Divide and Conquer method, the problem is taken and divided into sub-problems. These sub-problems are solved recursively to produce a solution for the main problem. To solve the sub-problems, another algorithm is used.

In our case, as seen in Figure 5, cities are divided into sub-cities. Firstly, cities are sorted in ascending order by their x coordinates. Then, this sorted array is divided into half as visualized by the red dashed line in the figure. Number of the cities are checked in every half and if the number is smaller than 4 (in this particular case), another algorithm such as *brute force* is applied. If the number is not smaller than 4, than it is divided again as visualized by the orange dashed lines in the figure. After the successful divisions, shortest path is found and these solutions are *merged* recursively.

Implementation of Divide and Conquer algorithm could not be achieved as it was too complicated to code. However, the basic idea and methodology is comprehended clearly.

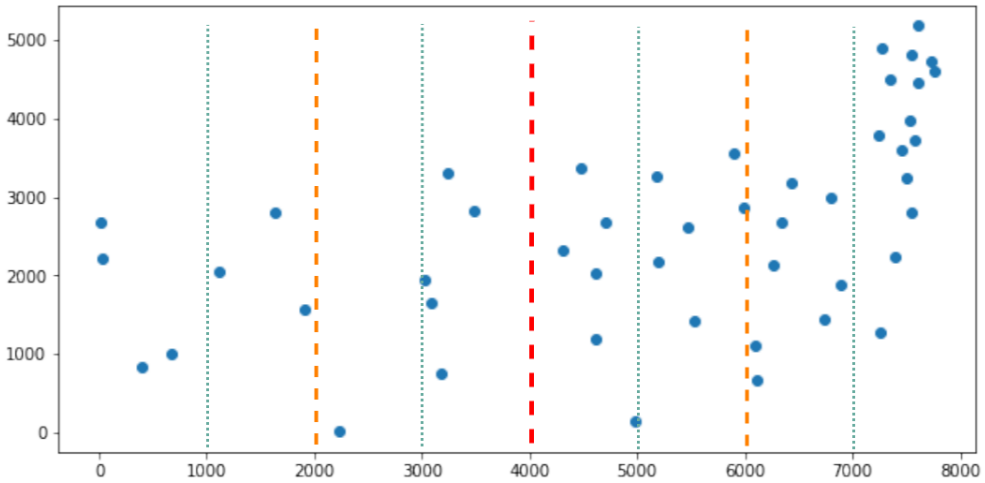

Figure 5: Divide and Conquer Visualization

7 Tour Visualization

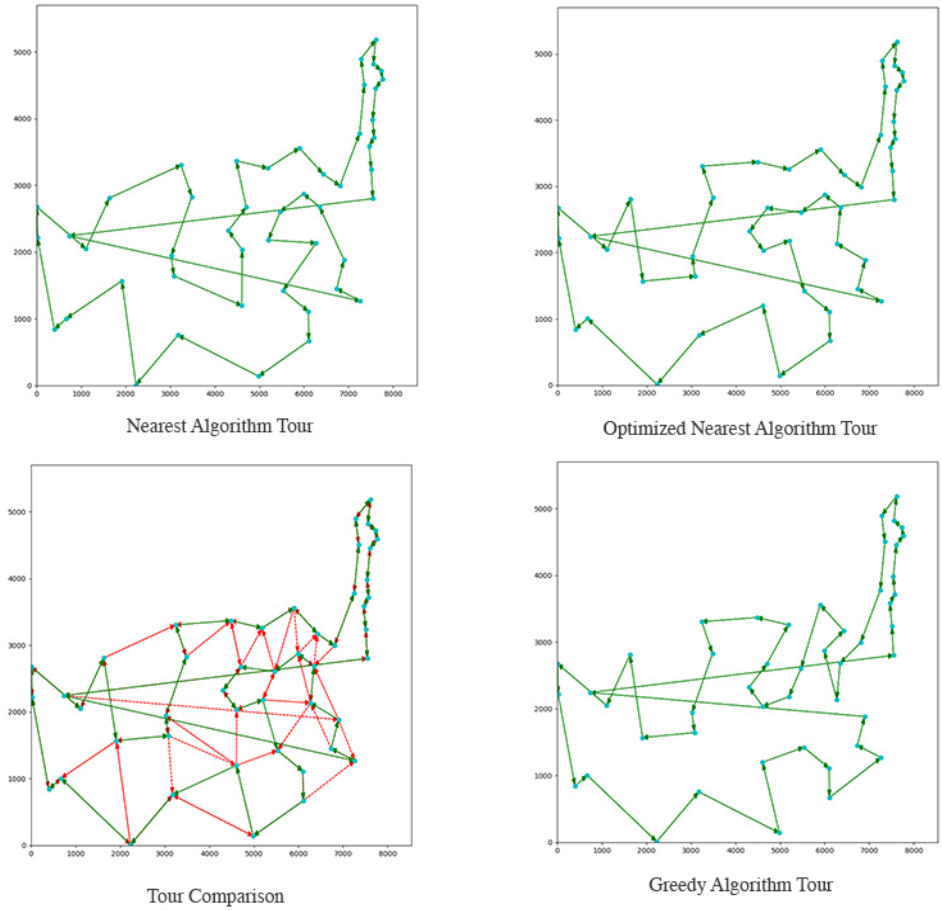

Figure 6: Tour Visualization

8 Time Complexity

Below is the time complexity of the algorithms implemented and time complexities of other approaches in order to provide a broader reference;

Greedy Algorithm - O(n^3)
– For each edge chosen, look through n^2 edges
Nearest Neighbor Algorithm – O(n^2)
– For each edge, chose from at most n edges
– Where n is the number of edges
Brute Force Algorithm – O(n!)
– Chose from i – 1 edges for every choice (n * n-1 * n-2 * . . .)
Best First Search Algorithm – O(n!)
– Worst case of no pruning (one node for every enumerable Hamiltonian cycle)

9 Conclusion

From the different techniques presented in this document, it is obvious that Nearest Neighbour Search algorithm is the most efficient in terms of run time. However, there is trade between the time complexity and the accuracy of the solution. Reaching the best solution is almost impossible without getting help of the Brute-Force approach, but also reaching the best solution is impossible using only Brute-Force approach. That's why in the project we implemented a solution using both Nearest Neighbour Search and Brute-Force algorithm. Based on the needs of a particular problem any of these algorithms can be implemented to provide a solution