Bilgehan Geçici 150117072 – Anıl Şenay 150117023

REPORT

Introduction

First of all, we read the file line by line with **readFromFile** function to get id, x and y coordinate values for cities. After that we create city objects with id, x and y values. Then we add these city objects to an array list named cityList. After that we apply **Randomized Nearest Neighbor Algorithm** with cityList array list to get shortest tour. To do that we indicate a start city.

Mainly we indicate start city as a first object of the cityList's first element. After that we find 3 cities which are closest to start city. We select randomly a city in 3 cities and then continue with that city. At the end of the algorithm we have re-organized cityList. Because of selecting nearest neighbors randomly, we observed that sometimes we get worse or better results than just implementing simple nearest neighbor algorithm. So, we decide to optimize the tour we created. To optimize the tour, we use **2-opt algorithm.**

Before implementing 2-opt algorithm, we use reorganized cityList, we call the **lineListCreate** function to create the array list that will hold objects from the Line class we have created, which we define as lineList. This function takes cities in 2 ways and creates Line2D objects from them and creates Line objects that hold the ID values and Line2D objects and add them to the lineList list. Then we use lineList array list to implement the **2-opt algorithm.** Basic idea behind the 2-opt algorithm is that changing selected 2 lines from lineList if there is an improvement on the results. If improvements exist, we simply change and update the 2 line's points and finally we reverse the path between points.

We observed that if we run the 2-opt optimization more times, we get better results. So, we decided to run 2-opt algorithm approximately 50 times to get a better tour. At the end of the program, we calculate the total distance of re-organized tour and then we simply print out the cities id's in given order to the output file.

Randomized Nearest Neighbor Algorithm

The nearest neighbor algorithm was one of the first algorithms used to solve the travelling salesman problem approximately. In that problem, the salesman starts at a random city and repeatedly visits the nearest city until all have been visited. The algorithm quickly yields a short tour, but usually not the optimal one. The nearest neighbor algorithm is easy to implement and executes quickly, but it can sometimes miss shorter routes which are easily noticed with human insight, due to its "**greedy**" nature. As a general guide, if the last few stages of the tour are comparable in length to the first stages, then the tour is reasonable; if they are much greater, then it is likely that much better tours exist. Another check is to use an algorithm such as the lower bound algorithm to estimate if this tour is good enough.

The only difference with the regular NN algorithm is that it is not completely '**greedy**', since at every step in building the route it considers multiple cities (in our algorithm we will find 3 cities) and randomly chooses 1. we can simply adjust 'greediness' of the algorithm by increasing or decreasing the number of cities. A less greedy algorithm will produce more variance in the generated routes.

2-Opt Optimization Algorithm

The 2-opt algorithm is a **local search algorithm** probably the most basic and widely used algorithm for solving Traveling Salesman Problems. Basically, it can be defined as deleting the two edges in the tour and connecting the tour, which is divided into two parts, in a way that reduces length of the tour. Once we choose the two edges to delete, we do not have a choice about which edges to add – there is only one way to add new edges that results in a valid tour. The 2-opt algorithm repeatedly looks for 2-opt moves that decrease the cost of the tour. A 2-opt move decreases the cost of a tour when the sum of the lengths of the two deleted edges is greater than the sum of the lengths of the two deleted edges.

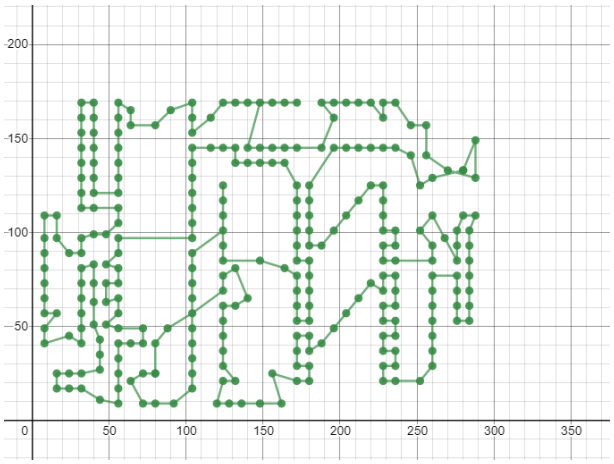
A complete 2-opt local search will compare every possible valid combination of the swapping mechanism. This technique can be applied to the travelling salesman problem as well as many related problems.

Division of Labor

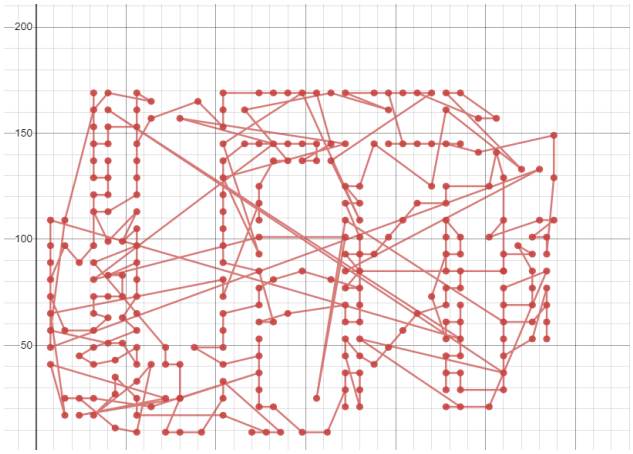
We mainly worked together via **VsCode Live Share Extension** during coding process. On the hand for researching and development we always kept in touch in this process. We tried to show our best work to finish the project successfully.

Visualization of Data's

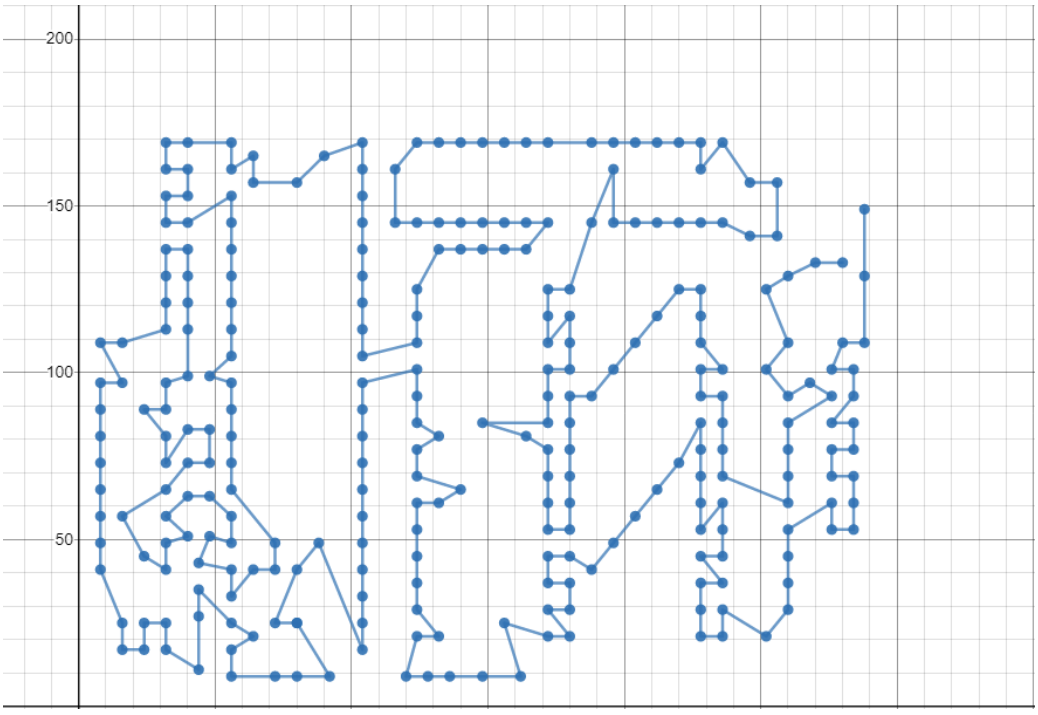
We use test-input-1.txt (280 cities) as example input for visualization process to understand better how **Randomized Nearest Neighbors Algorithm** and **2-Opt Optimization Algorithm** works.



(Figure 1.1 Before RNN Algorithm)



(Figure 1.2 After RNN Algorithm)



(Figure1.3 Final Result After 2-Opt Algorithm)