**HALF TSP**

Before starting to research how we could solve this problem, we have to decide on if we are going to choose half of the cities before computing a path, or while computing. We decided that precomputing would be a lot less costly and more advantageous. So, our solution is split into 2 distinct parts: choosing half of the cities and solving TSP for chosen cities. While choosing the cities, our criteria is closeness. We want to find the closest cluster of cities. This can be done with statistics.

**Choosing Cities**

Our program calculates the means of the x and y variables while reading the cities. These are then passed into out **chooseCities** methods which chooses half of the cities using statistics. First, it calculates the variance of both the x data set and the y data set. Then we can get the standard deviation by taking the square root of the variance. Using this, we calculate the Z-Scores for each city. The Z-Score in statistics represents how far away a piece of data is from the mean. Since we want the closest cluster possible, the method sorts the cities by their x and y Z-Scores by descending order into separate arrays. Then, it removes cities by switching between these arrays until half of the cities remain. This practically creates a square and removes any outliers which would extend our rout. The difference in route is immediately apparent when compared to random selection of cities, but will have the most effect after optimization is done.

**Solving TSP**

As said in the report, it would be unfeasible to try and solve exactly. So, this means we can’t use brute force, dynamic programming, backtracking or branch and bound solutions. So, we started researching heuristics and approximation algorithms. It is also important to keep in mind that in our problem every city can go to every other city, as there are no predefined paths. This makes our graph a **complete** graph. This limits the different type of algorithms we can use, since there are so many edges.

After researching many academic papers, we found that to solve this problem would consist of 2 parts in of itself:

1. Constructing a tour
2. Tour improvement

The first step means that we need to create a route, even if it may not be optimal at first. The second step means optimizing this route. The details of our implementation are below.

**Constructing a Tour**

There are a few options here, but we chose to use a nearest neighbor approach. The reasoning behind this is that the extra calculations of the insertion heuristics or Christofides methods would not be worth the extra calculations. Since we are going to optimize the route after constructing anyway, we chose the one with the least number of calculations. The tour construction is contained in the nearestNeighborSolution method. It starts the tour from the city given in its parameter, which is the middlemost city. Then it creates an array of cities that are not visited, and finds the closest one. Then it removes the city it went to; this goes on until no cities are left. This path is at worst 25% worse than the theoretical best route. Now what we have to do is reduce this number using optimization techniques.

**Tour Improvement**

For this part, we chose a 2-opt approach.

**REFERENCES**

1. Average-Case Analysis of the 2-opt Heuristic for the TSP by Jaap J. A. Slootbeek ([Slootbeek\_MA\_EEMCS.pdf (utwente.nl)](https://essay.utwente.nl/72060/1/Slootbeek_MA_EEMCS.pdf))
2. The Travelling Salesman Problem by Libby Daniells

([The Travelling Salesman Problem – Libby Daniells (lancaster.ac.uk)](https://www.lancaster.ac.uk/stor-i-student-sites/libby-daniells/2020/04/21/the-travelling-salesman-problem/))

1. Heuristics for the Traveling Salesman Problem by Christian Nilsson

[Heuristics for the Traveling Salesman Problem By Christian Nillson.pdf (free.fr)](http://160592857366.free.fr/joe/ebooks/ShareData/Heuristics%20for%20the%20Traveling%20Salesman%20Problem%20By%20Christian%20Nillson.pdf)

1. Comparison of TSP Algorithms by Byung-In Kim, Jae-Ik Shim, Min Zhang

[Document3 (mykhi.org)](https://pja.mykhi.org/0sem/NAI/rozne/Comparison%20of%20TSP%20Algorithms/Comparison%20of%20TSP%20Algorithms.PDF)

1. 2-opt

[2-opt - Wikipedia](https://en.wikipedia.org/wiki/2-opt)