*BRIAN THOMAS PHELAN*

*Student Number: 18401696 | Email: brian.phelan.2019@mumail.ie*

*CS211 PROJECT*

2020

When taking on the CS211 (Algorithms & Data Structures II) Project, I researched many different methods of solving the problem. Analysing the brief, we have 1,000 different GPS Co-ordinates around the globe which must all be visited at least once to deliver the required supplies. This variation of the “Travelling Salesman Problem” came with one constraint, the distance between any two locations had to be greater than 100km. Originally I had tried to design a Genetic Algorithm to solve the problem, but I ran into a number of issues regarding the constraint, so I opted to implement a variation on the Nearest Neighbour Algorithm.

The first problem I had to tackle was loading the GPS Co-ordinates. I decided to convert the file to .csv format and use a Buffered Reader to read in the information to the program, directly referencing the file within the directory. Within this method (generateArrayListFromFile) I converted the read in stream to an Array List.

A screenshot of a cell phone

Description automatically generated

I wanted to have the Co-ordinates in the form of a two-dimensional array where there would be 1001 rows and 2 columns, i.e. [Longitude][Latitude]. To do this I had to convert the Array List to a one-dimensional String array and then to a two-dimensional double array as seen below. Please Note: I added location 0 to the end of the array also so it would be a round trip.

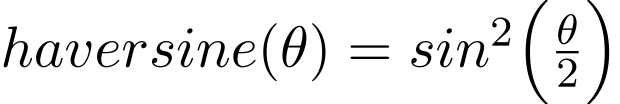
A screenshot of a cell phone

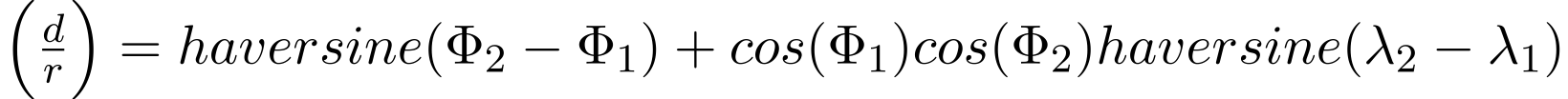
Description automatically generated

From there I just created a Population object and called the necessary method to solve the problem.

The Population class controls all the methods that find the solution. For the constructor, it was just a matter of letting a global double array be equal to the GPS Co-ordinates passed into it so I could use it with ease throughout the class. I also created a global variable in the same form called “path” that will be the solution path. Lastly, an array of Booleans that show whether a location had been visited or not.

Next, I needed a way to calculate the distance between two GPS Co-ordinates. To do this I used the “Haversine Formula” which calculates the distance between two points on a sphere,





Where  are the two latitude points and  are the two longitude points respectively (r = Earth’s radius).

A screenshot of a cell phone

Description automatically generated

Lastly we had to sort the locations. To do this, I used an algorithm which would compare the distance from the current location to all other locations in the array. The element that gave the smallest distance above 100km (I had to use 200km as using 100km would leave out certain points) would be added to the next point in the array. That next point would then become the current point and would repeat the process until all locations have been added, completing the path.

The algorithm itself is as follows:

1. Set all points as unvisited.
2. Set the starting point as point 0 and the ending point as the same location as point 0.
3. Mark point 0 as visited.
4. Find all the distances between the current point and the rest of the points.
5. If the distance is greater than 200 (instead of 100 due to errors) and less than the best distance so far, assign that distance as the “best” and it’s index as the “bestIndex”.
6. Once all distances have been calculated, whatever the best value is, we assign the index of the value to the next point in the path.
7. Repeat from step 4 until the path is completed.
8. If path is completed, then terminate the program (Swap 5 random elements (deterministic outcome).

This is a deterministic algorithm whereby the resulting path will always be the same. I opted to choose this algorithm as the run time is very short compared to other algorithms (Program took 0.318906513 seconds to run) and it also gives a very good solution compared to others.

I used java because it is a powerful language and also because I am proficient in it so it allowed ease of development.