**Project 1: TSP Brute Force**

Stone Barrett

Computer Science Engineer

Speed School of Engineering

University of Louisville, USA

[Scbarr04@louisville.edu](mailto:Scbarr04@louisville.edu)

**1 Introduction**

In this project, the goal was to implement a predetermined algorithm in a language to be selected to solve the NP-complete Traveling Salesperson Problem. The premise of the problem is that there is a given list of geographical coordinates that represent places a traveling salesperson will need to stop at. These points all have distances between one another and because of these varying distances that have yet to be calculated, the salesperson cannot simply look at the map and immediately determine what the fastest route to take would be. So, the project needed a computer program to determine this optimal route automatically. This would allow for the traveling salesperson to waste as few resources as possible in his route. This route also must be Hamiltonian, meaning the place the salesperson starts is also the place they end; no other place can be visited twice.

**2 Approach**

To approach this problem, I decided to use the Python language to write the program. As far as the algorithm beyond employed, a certain limitation was put on the problem for this one time. I was required to use the brute force method of solving this problem, almost certainly to demonstrate how dangerous computational times can become with larger datasets. To brute force means to attempt every single possible outcome in order to find the best one.

**3 Results**

Given the specific instruction to keep optimization out of the question, I can only assume the algorithm performed as poorly as it was expected to. Brute forcing, as I have learned, should most likely be a last-ditch effort when no other solutions seem to solve the problem at hand – that is, when dealing with large datasets. The algorithm was lightning fast when it came to the smaller sets of data that were supplied. As the data grew larger, processing time for this algorithm grew exponentially larger. This obviously cannot be feasible past a certain point, as computational limits apply.

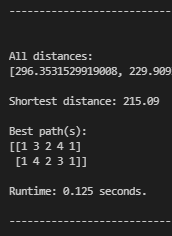
**3.1 Data**

The data that was supplied to test my program with were .tsp (text) files containing randomly generated geographical coordinates in increasing quantities. The first file had four points to account for. The fifth had eight. This grew until the ninth contained twelve data points. My program read these files one at a time and line by line to gather the relevant information from them. With this information, data structures were used to store, organize, and comb through to find optimal paths.

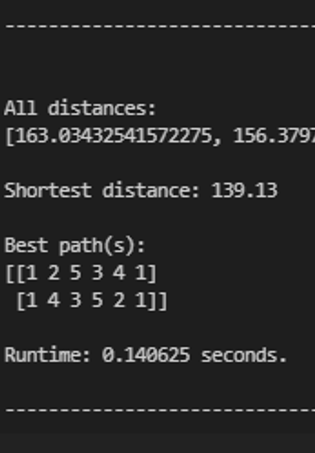
**3.2 Results**

The results of the program can be seen in this section.

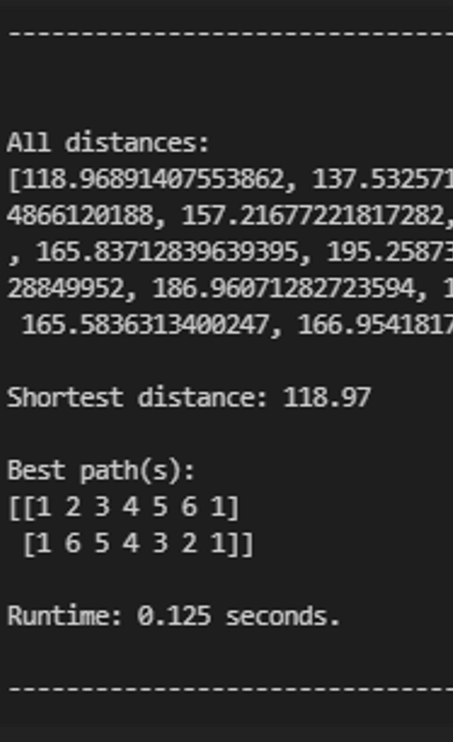
4 points:



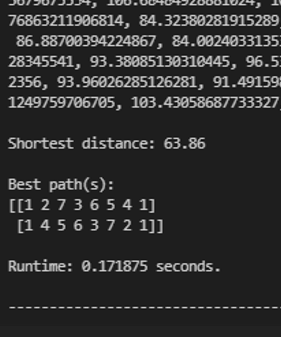
5 points:



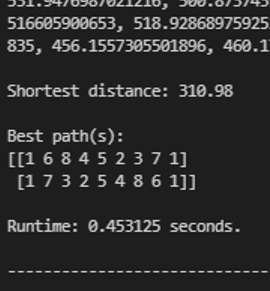
6 points:



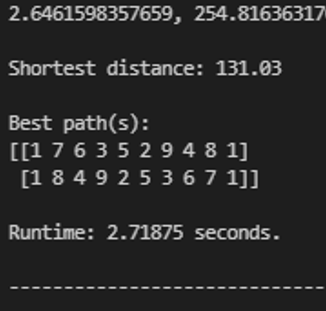
7 points:



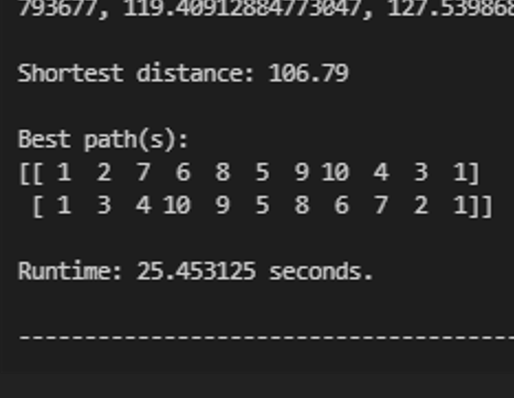
8 points:



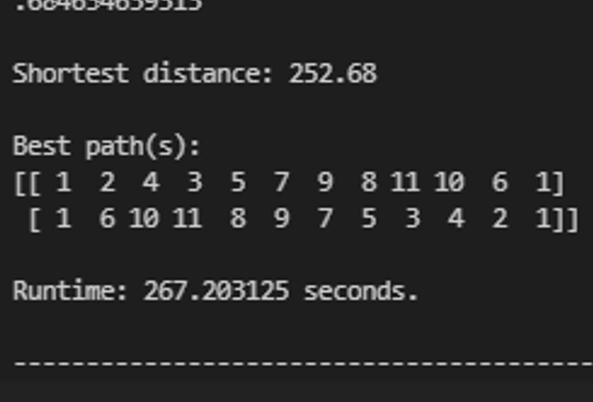
9 points:



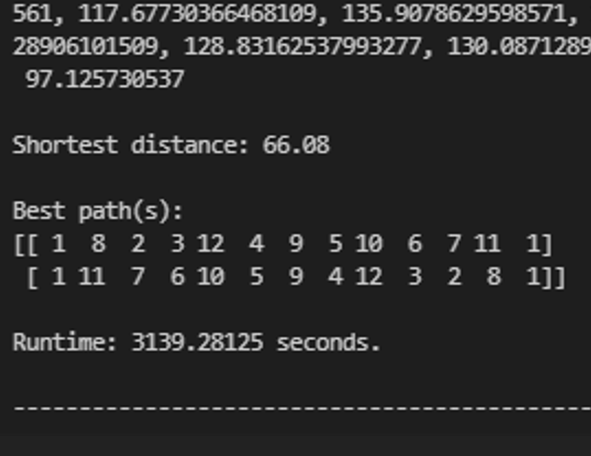
10 points:



11 points:



12 points:



I have constructed a graph plotting the number of stops the salesperson must make versus the time it takes my program to compute the shortest path. This demonstrates the intensity of the exponential growth of computational resource requirement.

**4 Discussion**

As mentioned before, the results I got were within expectations. Seeing the sheer difference between running the program with 11 data points and with 12 very clearly demonstrated that there must be better ways of handling similar problems, as larger data sets would eventually become virtually impossible to handle using this method.

**5 References**

Itertools - <https://docs.python.org/3/library/itertools.html>

NumPy - <https://numpy.org/>