**Project 3**

Jeremy Boyd

[Computer Science and Engineering]

Speed School of Engineering

University of Louisville, USA

[jrboyd03@louisville.edu](mailto:jrboyd03@louisville.edu)

1. **Introduction** (What did you do in this project and why?)

For this project I was responsible for solving the Travelling Salesperson Problem which is a problem programmers have been working on since forever. It involves a salesman being given a list of cities to travel to and their coordinates the problem is to find the most efficient route.

The approach I took to solve this problem was to create a greedy algorithm, this is an approach that chooses the locally optimal option in hopes of achieving a globally optimal solution. When implementing this type of algorithm you’re not guaranteed to get the shortest route. For this project I created a nearest neighbor program, that would start at one city and insert the cities one by one based off the city that is closest to an edge already in the tour. The route that is computed is then plotted using Matplotlib and displayed using its GUI.

1. **Approach** (Describe algorithm you are using for this project)

The programming language I used for this project was Python. The algorithm for this project was the easiest to implement so far in my opinion. I started by importing the math, collections, pandas, and numpy libraries. Then I brought in everything I’d be using from my previous project including my ‘distance’ function to calculate the distance between two points. Also the table function that would read in the data from the tsp file and store it in a table, a list that held all the x and y coordinates stored in the table, and a list called ‘distances’ that would iterate through the list of coordinates and find the distance between each set of points. Then in conjunction with an asarray, all the values that are stored in the ‘distances’ list are imported into the array along with their indexes so that the asarray functions as a sort of look up table.

Next, I created a greedy function that would take in a list of cities, the starting city, and the asarray that holds the distances between every city. Then I created two list one that would hold the path taken through the cities and another that would hold the remaining unvisited cities. After this I created a while loop that would proccess until the unvisited list was empty. And within that while loop a call was made to another function called ‘nearestNeighbor’. This function took in the last city added to the path, the list of unvisited cities, and the asarray. Within this function a for loop would iterate through the unvisited cities and using the asarray, look up all the distances from the current city to all other cities in the unvisited list of cities. Then it would append those distances into another list, after this I created a variable that would return the smallest number in the list of distances. This variable would be used to retrieve the index number of the value within the distances list because this value matches the index of the city that is the shortest distance from the current city. And that index number is used to return the number within the unvisited cities list that corresponds to the shortest distance.

After the ‘nearestNeighbor’ function returns the city that is the shortest distance away, it is stored in a variable. That city is then appended to the path list and removed from the unvisited cities list. This process will repeat and then the path will be returned once the while loop ends. Next to calculate the distance of the path returned I used a for loop that would iterate for the length of the returned path and use the asarray to calculate the distance from one city to the next in the path and add them up inside a variable. Then the route and its distance were printed to the screen.

Next in order to visually represent the path taken I imported matplotlib.pyplot. Which is a plotting library for the python programming language which includes a general purpose GUI that will display the path I generated. First I created two list that would take the x coordinates and y coordinates out of my list that I used from previous projects that held the x and y coordinates for each city and stored them in separate list. I then created two more list that would take the path generated by my greedy function. To use the indexes of the cities in the path in order to append the x and y coordinates in the proper order that corresponds to where the city is located in the generated path. So they can be plotted in the proper order, I then created a function that would take in a list of x coordinates and y coordinates it would then plot each of the points in the order indicated by the list and display them using the GUI.

1. **Results** (How well did the algorithm perform?)

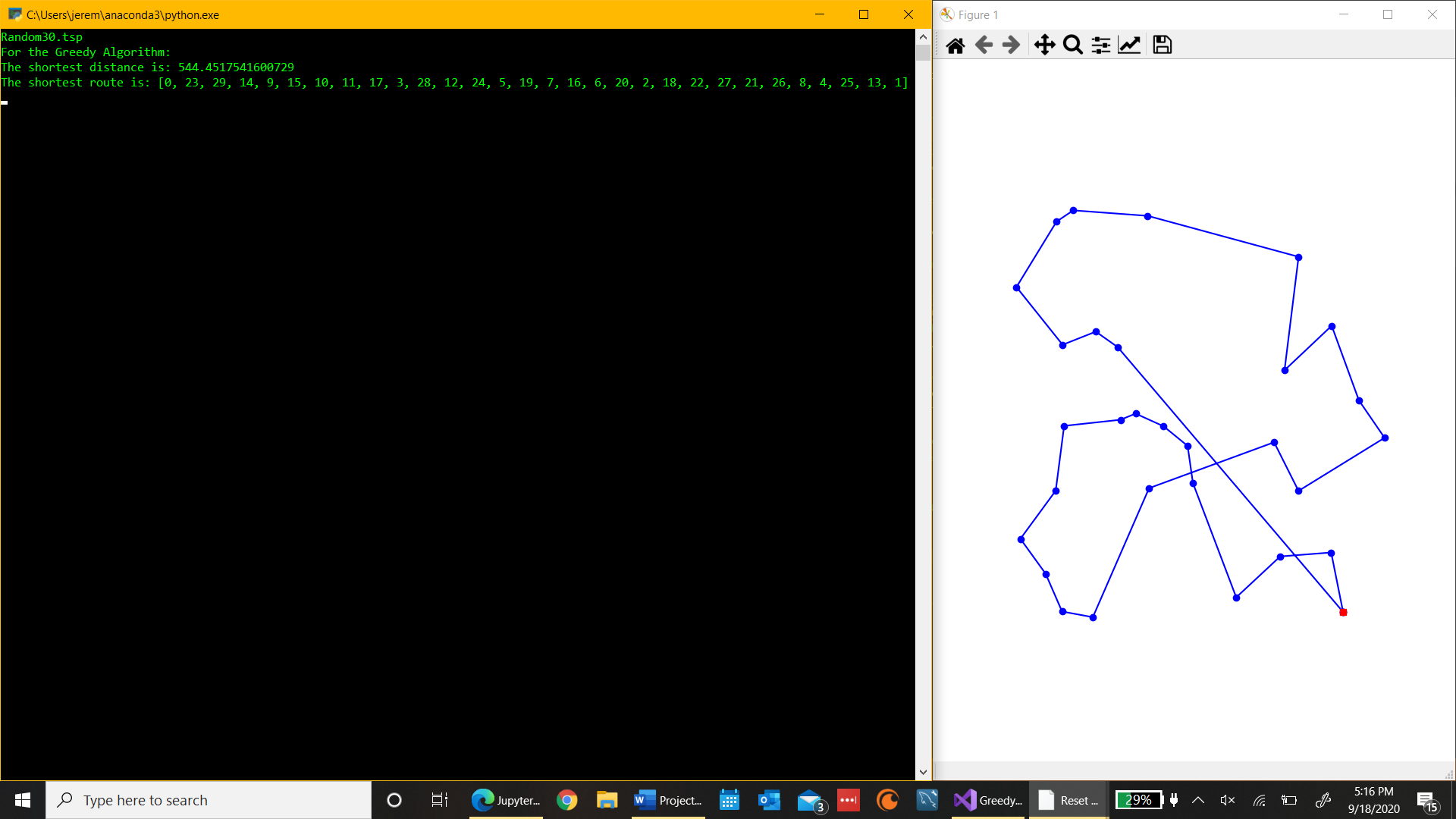
The algorithm performs well and is able to generate a path and its distance within seconds for 30 and 40 cities.

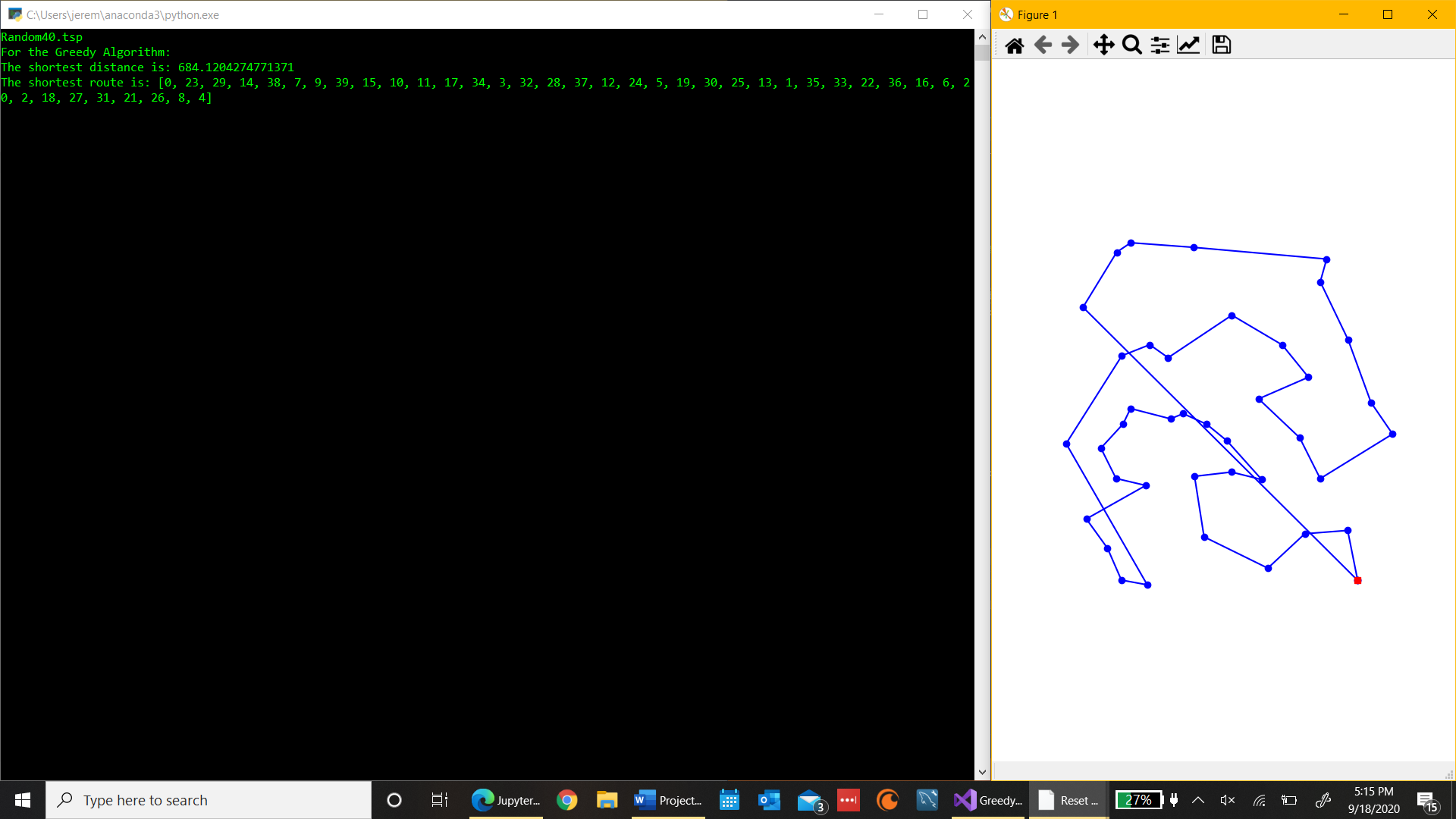
* 1. **Data** (Describe the data you used.)

I used the given data in the tsp files we were given, which included 30 and 40 numbered cities and their respective x and y coordinates.

I also used the tsp files from previous projects with less cities to check things like making sure the shortest distance from one neighbor to the next was actually being calculated and chosen correctly.

* 1. **Results** (Numerical results and any figures or tables.)

- 



1. **Discussion** (Talk about the results you got and answer any specific questions mentioned in the assignment.)

I was able to generate what would be the shortest path and its distance using a greedy algorithm for 30 and 40 cities as well as for smaller city counts like 12 and 11 very efficiently. In my program you can start at any node and it will calculate the best path using the greedy algorithm from that node to travel to all others. But if you look at the results the algorithm makes good decisions but not optimal ones as there are intersecting lines. This type of algorithm wouldn’t be optimal cause it doesn’t guarantee that this is actually the shortest path.

In comparison to using brute force the greedy algorithm is much better because on a regular computer like I have the program crashed and couldn’t even calculate the shortest path for 12 cities using brute force. On the other had I believe BFS or DFS would be better algorithms for this problem because the path they generate is guaranteed to be the shortest path. But if the greedy algorithm does generate the shortest path then I would say it performs the best it just isn’t full proof.

For my greedy function its time complexity is O(), which is the number of computations required too survey all the options left for every node and then pick the best one.

The computes I’m using is a LENOVO YOGA 720-12IKB with a Intel Core i7 CPU with a clock rate of 2.80 GHz, it has a 64-bit operating system, and I’m using Visual Studios as my IDE to run my code.

1. **References** (If you used any sources in addition to lectures please include them here.)