Implementing Ant Colony Optimization to Solve the Traveling Salesman Problem

Dawson Merkle and Thomas Levitt

1. Introduction

In this report we will be examining and forming conclusions about the implementation of Ant Colony Optimization (ACO) in solving the Traveling Salesman Problem (TSP). This implementation was developed using the Python programming language using the Visual Studio Code source code editor. The Ant Colony Optimization algorithm used in this report emphasizes various functions to create ants that take a specific path through a set of cities, according to pheromones and distance. These operations are repeated, creating iterations, and run until the number of iterations is met. Using this algorithm, the goal is to find the shortest route possible using a sequence of the created cities.

2. Research and Planning

Within this project we both wanted to do the research, programming, and report as collaborative partners. We read chapter 6 from the "*Grokking Artificial Intelligence Algorithms*" textbook by Rishal Hurbans to familiarize ourselves with the general structure of Ant Colony Optimization. After reading chapter 6, we used the Ant Colony Optimization steps as a key piece of information to base our program off of. We planned to use each of the steps as a central idea for our functions. After reading the textbook, we looked at the "carnival_aco.py" file provided from the textbook. A complication we had when designing our program was how to update the pheromone matrix when using a list of cities and how to keep track of which specific route the ant took during its tour. We debugged the provided code to see how the pheromone

matrix was updated. To our surprise, we noticed that the provided code was not updating the pheromone matrix appropriately. We saw that every index in the matrix was being updated with the same value, which is not correct. Upon this discovery, we had to do more research on how to design our program. We found a YouTube video from Ali Mirjalili titled, “How the Ant Colony Optimization algorithm works,” which was a low-level description on the mathematics of the algorithm (no programming). To get our feet wet, we inherited some elements such as the city class and create_city function from our previous project, “Traveling Salesman Problem using Genetic Algorithms”. We also wanted to keep a repository for our source control so we would be able to compare our changes and revert if needed.

3. Program Design

Our program starts with the initialization of the City class which returns an object of an individual city with x and y coordinates attached. The city class is then used to make ‘N’ number of cities which are then contained in a list. There is also an Ant class which returns an object of an individual ant with a list of cities associated with it. The Ant class uses the City class to store its route. The solution space is real encoding. When the program starts, a list of cities is made, a distance matrix of those cities are made, the pheromone matrix is made, and the ant colony is made. This is the core of the program before the Ant Colony Optimization algorithm begins. The initial ant population is created by looping the ant object creation and assigning a random city as its start point. Then, the do_tour function is called with an individual ant as one of the parameters which does the calculations for the tour the ant does. Inside the do_tour function, the pheromone matrix gets updated but stores the updated matrix until the next iteration. The pheromone matrix is updated by getting the index of the city the ant is currently at and the next city the ant is going to. This gives an x and y position for the matrix. To compute the best solution, the ant does a full tour and compares its total distance to the global best distance. We also store the best distance for

the ants in a single iteration for plotting purposes. The stopping condition for the program is a predefined number of iterations. In the main function of our program there is a hyperparameter named “NUM_ITERATIONS” that can be changed depending on what the user wants. The reason that we chose to have the stopping condition be a parameter and not include something that stops the program if there is no change in the best route is because of what we found when testing. We noted that the randomness of the ants starting point causes the ants to not converge onto a single best route. This is both a good thing and a bad thing for many reasons, but relating to the stopping point, we wouldn’t want the program getting cut short because there is still a chance that deep into the run a new best route could be found. The stopping point really depends on the use of the algorithm, and for right now, we thought it would be best to have a hyperparameter that controls the execution. The experiments we did during testing of our program were extensive and at every part of progression. When making the distance matrix, we ran multiple tests to make sure our distance matrix was configured properly. When choosing the probabilities for the cities that the ant could travel to next, we wanted to make sure our probabilities were not random and actually reflected the distance between the cities and the pheromones which lie on that path. Another crucial test we ran was making sure the pheromone matrix was being updated correctly. We tested this by looking at the index of the two cities that the ant traveled to and from, and matching those indices with the indices on the pheromone matrix. This was most notably our most rigorous test because we wanted to ensure that the tour the ants were taking influenced the next generations. We also plotted the best route distance over the iterations for a visual of the ants learning. Another design specification that we decided on was to have explicit names associated with each city. This was useful in the development process when we were making sure that the coordinates of the cities weren’t changing when we were creating the program. Also due to this, the maximum number of cities that we can simulate is 58, because of the amount of cities in the state of Florida. In the main function of our program, there

are many hyperparameters that we decided on making. These include, the number of ant factor (NUM_ANT_FACTOR), the number of cities (NUM_CITY), the alpha (ALPHA), the beta (BETA), the random attraction factor (RANDOM_ATTRACTION_FACTOR), the pheromone evaporation rate (EVAPORATION_RATE), and the number of iterations the program runs for (NUM_ITERATIONS). Creating these to be used as arguments for our functions allows for the algorithm to be changed however the user desires. For example, using these values, diversity can be increased or decreased, the program can be run for much shorter or longer, and the number of overall cities can be changed to create a more complex problem.

4. Representative Results

The results of our Ant Colony Optimization algorithm are represented easily by the plots we have made. The first output of our plotting shows the initial route made with no computation.

```
# hyperparameters
NUM_ANT_FACTOR = 1
NUM_CITY = 25
ALPHA = 1
BETA  = 1
RANDOM_ATTRACTION_FACTOR = 0.05
NUM_ITERATIONS = 500
EVAPORATION_RATE = 0.2
```

Figure 1.1

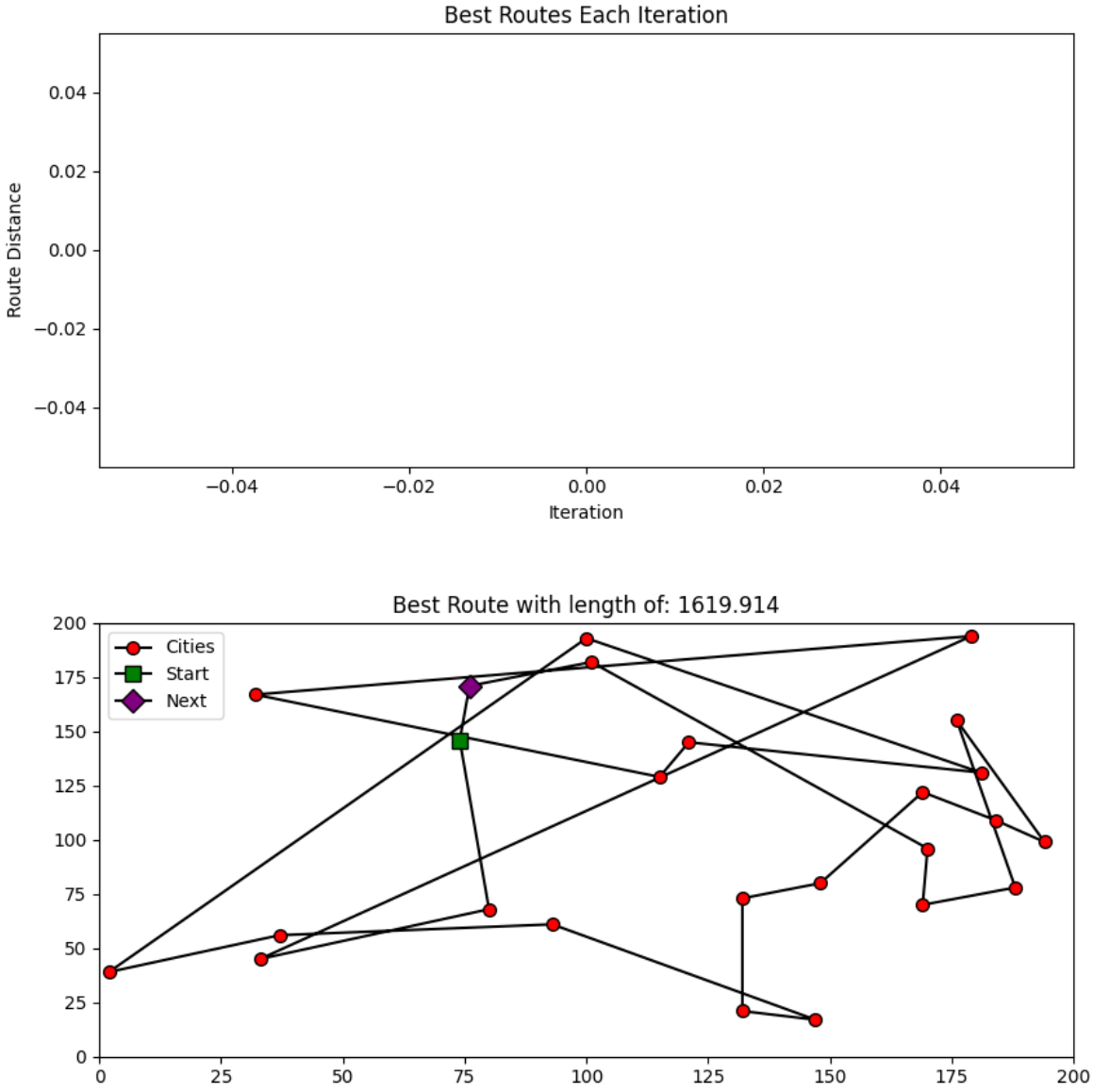

Best Routes Each Iteration
Route Distance
Iteration
0.04
0.02
0.00
−0.02
−0.04
Best Route with length of: 1619.914
Cities
Start
Next
0
25
50
75
100
125
150
175
200

Figure 1.2

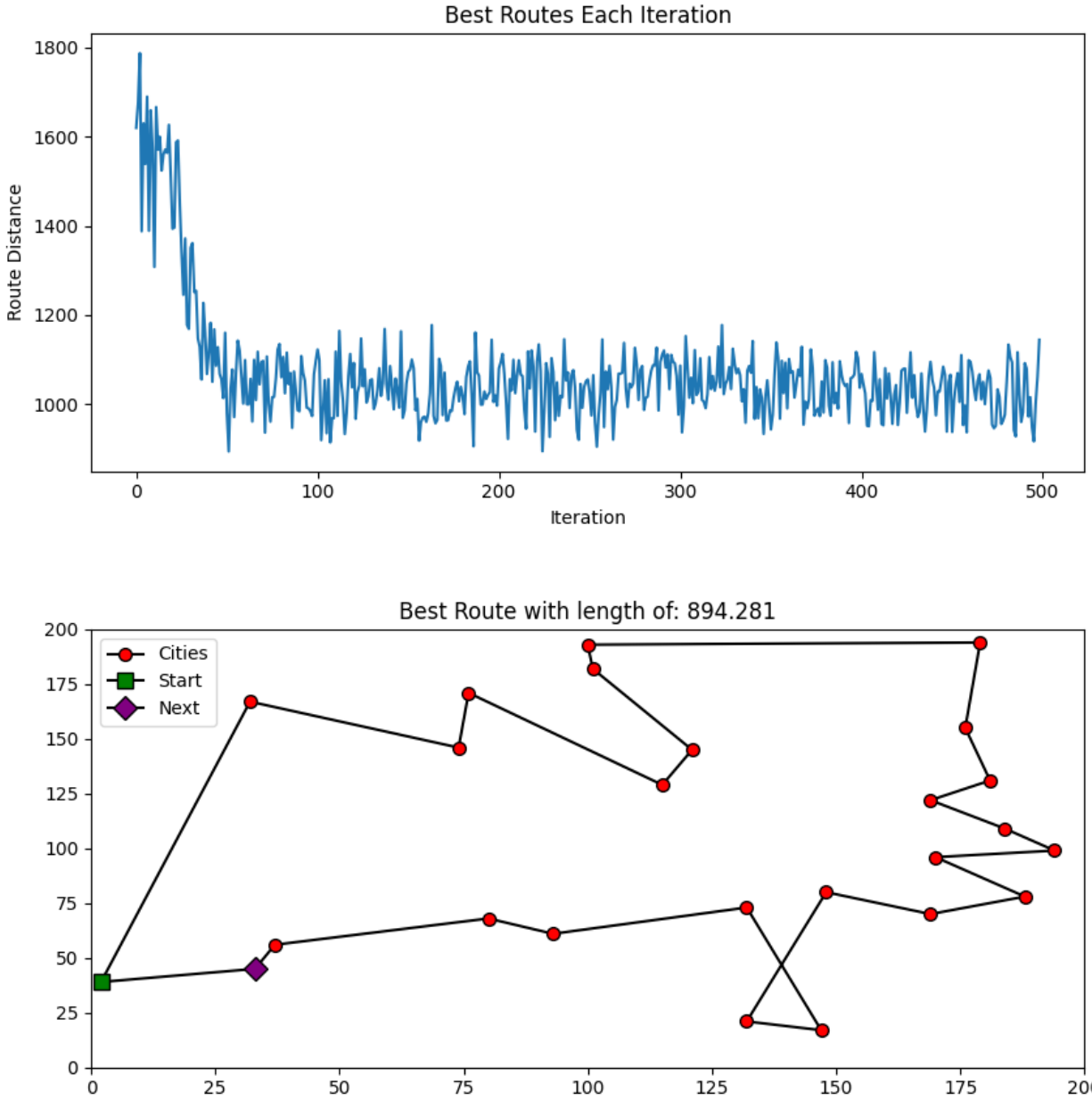
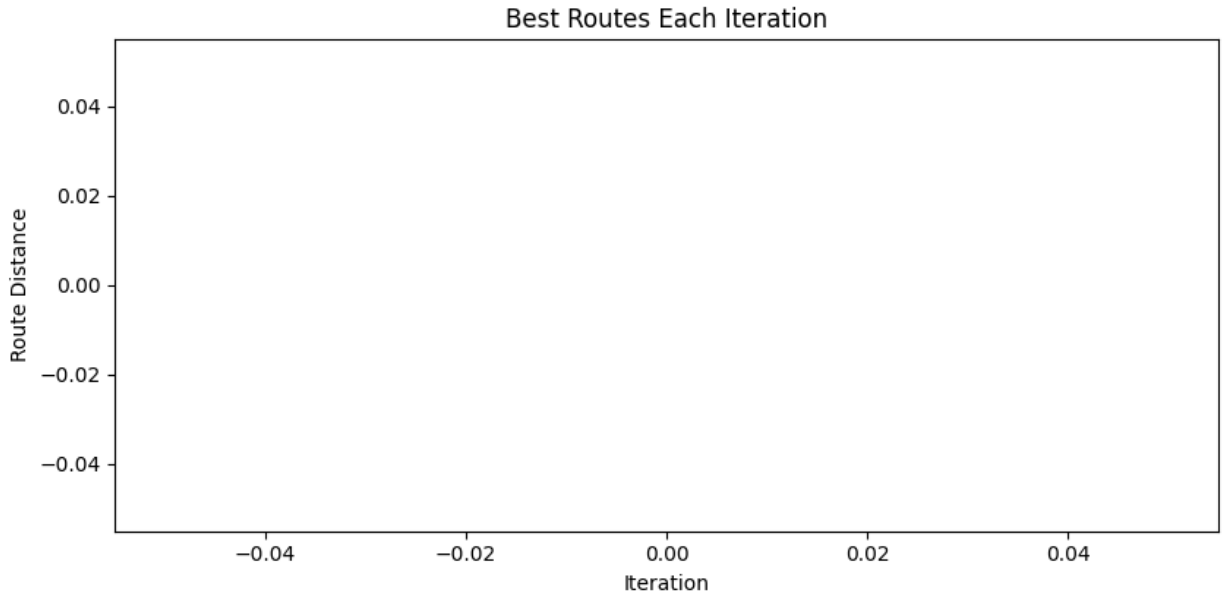


Figure 1.3

Figure 1.1 shows the hyperparameters that define the run. As shown, the "Best Route Each Iteration" plot is empty as there is no data at the beginning. We programmed the plotting to occur every 10 iterations so it could be evident to see the improvement. When the program is finished, the "Best Route Each Iteration" plot provides a curve that allows us to see if progress is still being made and where the best route was found. The Best Route diagram, seen in Figures 1.2 and 1.3, also provides a much cleaner route for the Traveling Salesperson.

```
# hyperparameters
NUM_ANT_FACTOR = 1
NUM_CITY = 25
ALPHA = 5
BETA  = 1
RANDOM_ATTRACTION_FACTOR = 0.05
NUM_ITERATIONS = 500
EVAPORATION_RATE = 0.2
```

Figure 2.1



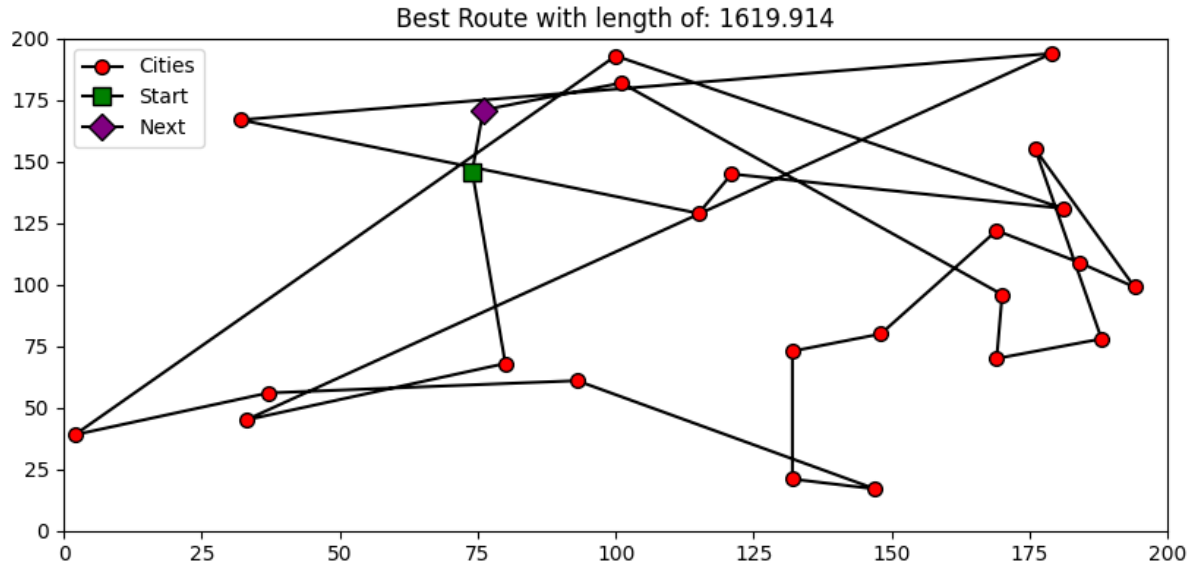


Figure 2.2

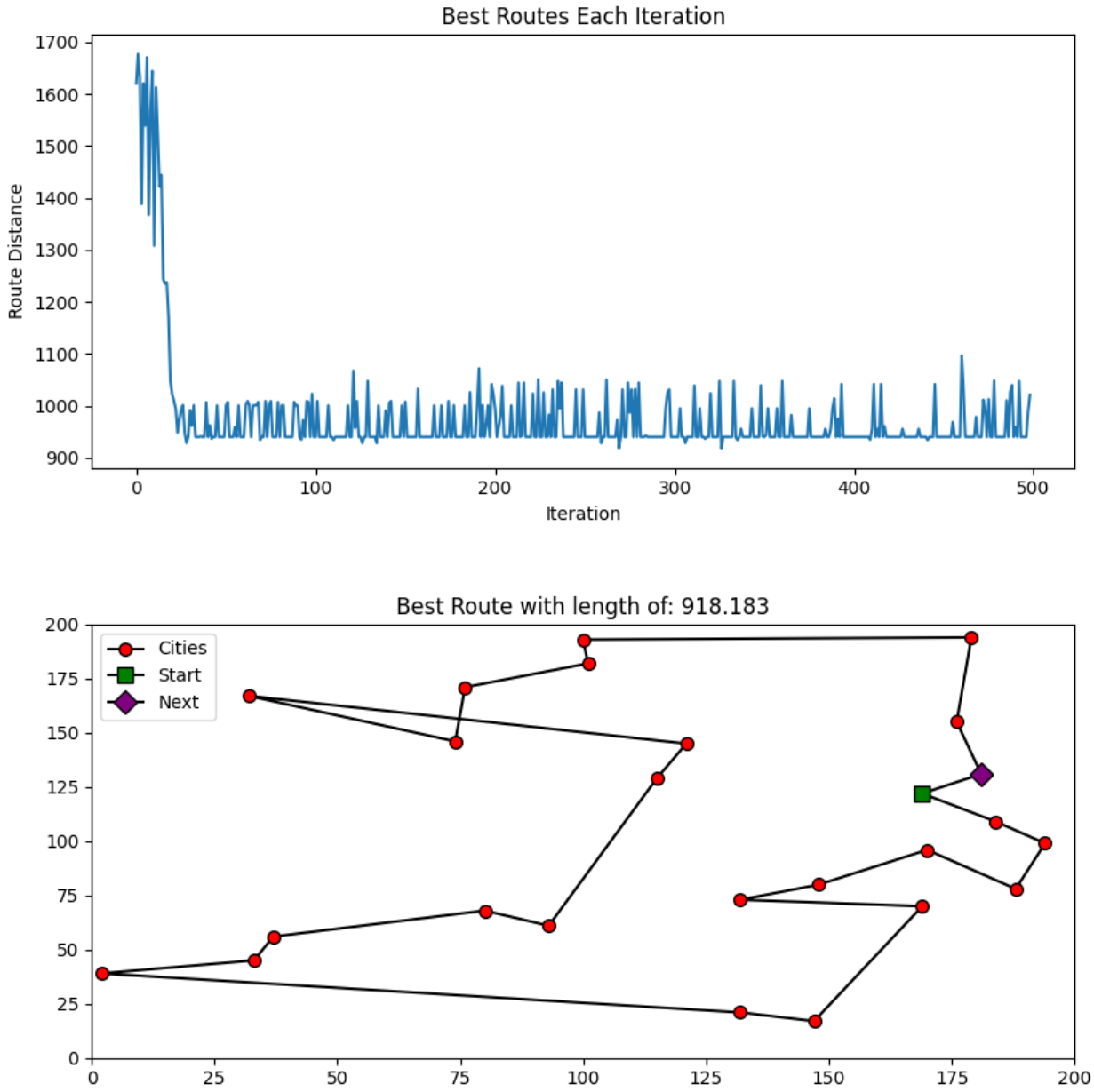
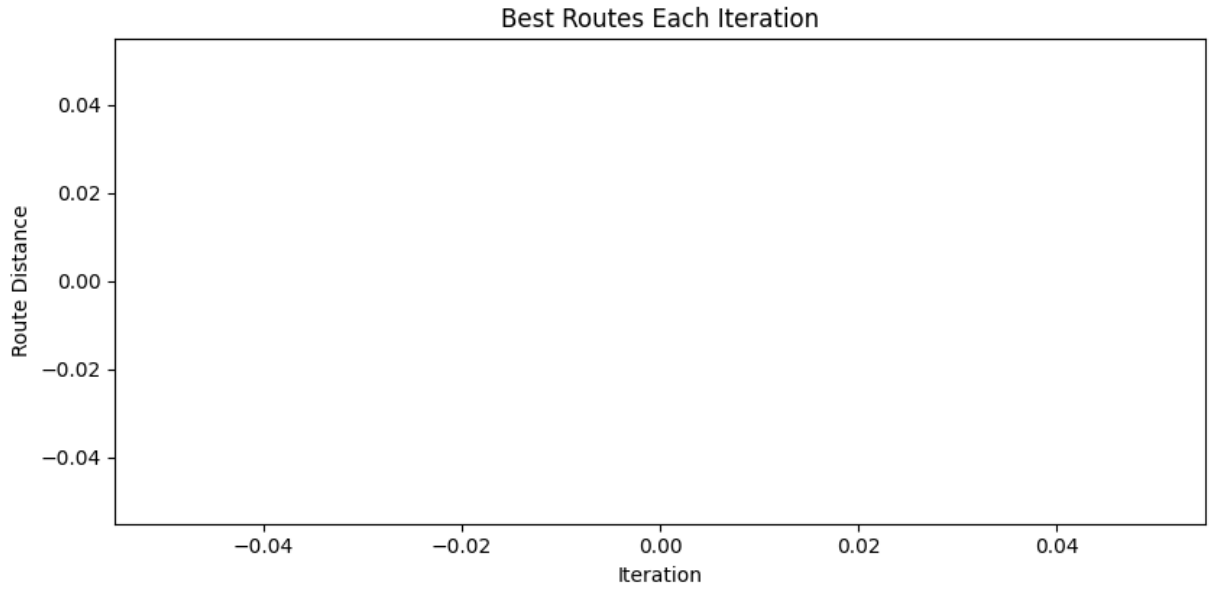


Figure 2.3

Figure 2 pictures highlight a test run with the alpha value being 5 and the beta value being 1. This affects the program greatly because it changes the weight of the pheromones to be very high when the ant is choosing it's next city.

```
# hyperparameters
NUM_ANT_FACTOR = 1
NUM_CITY = 25
ALPHA = 1
BETA  = 5
RANDOM_ATTRACTION_FACTOR = 0.05
NUM_ITERATIONS = 500
EVAPORATION_RATE = 0.2
```

Figure 3.1



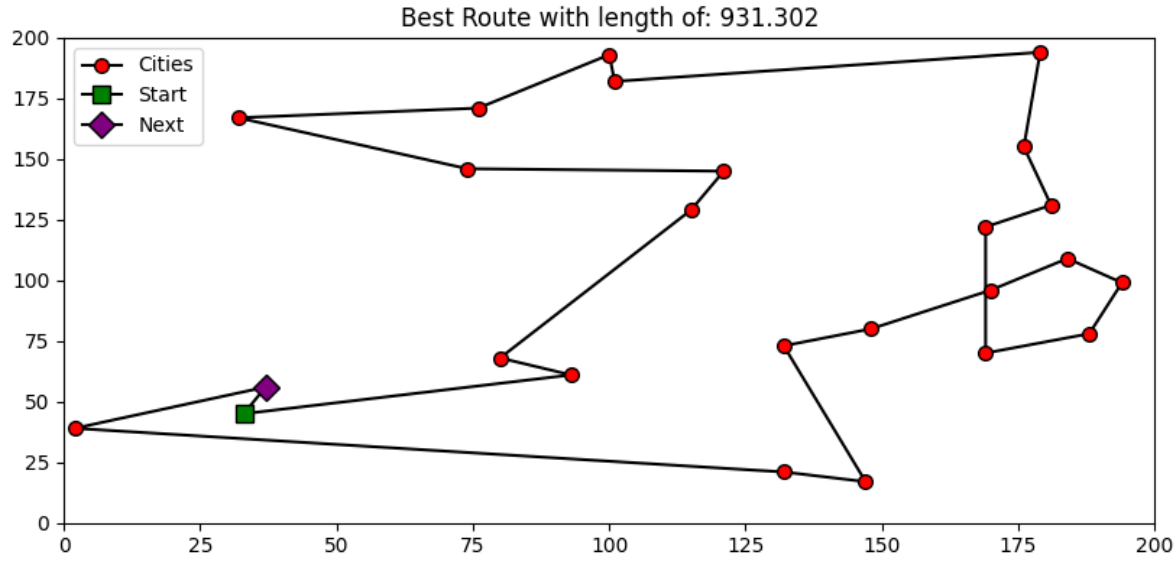


Figure 3.2

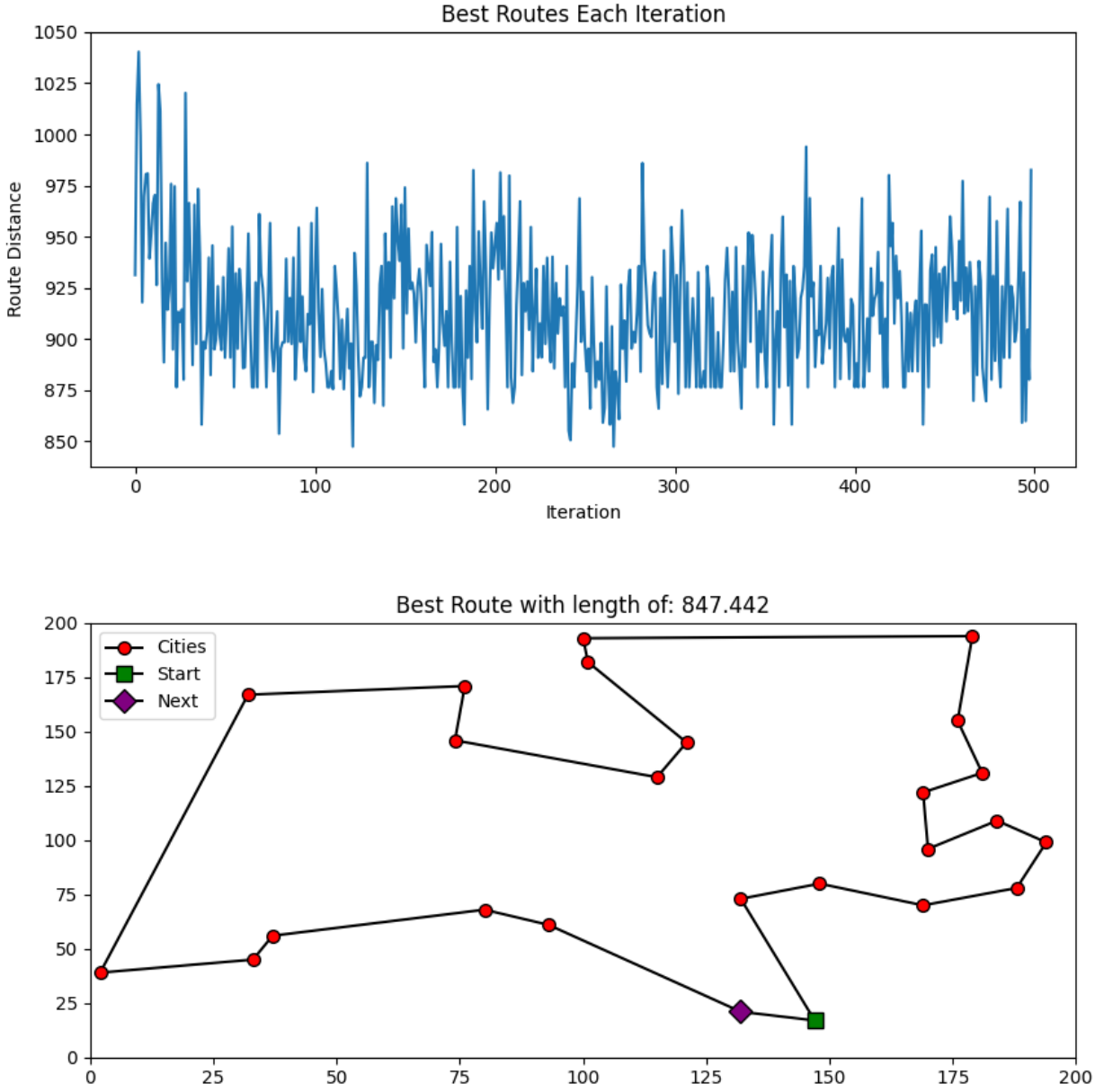


Figure 3.3

Figure 3 pictures highlight a run where the beta value was set to 5 and the alpha value was set to 1. This means that the distances were taken into account more than the pheromones. This causes the ant to select the next closest city most of the time. Relating to this seed, the algorithm performs very well under these parameters.

5. Future Improvements

While the project was completed and done to the best of our abilities, there are always

improvements that could be made. One example of these includes making a single animated plot that is constantly updated, instead of displaying multiple plots per however many iterations. This would allow for the user to see the best route being updated in real time. Along with this, we would choose to have the names of the cities included in this plot, allowing for visualization of the optimal route to have meaning.

The following link navigates to the GitHub Repository where project can be found: https://github.com/ddmerkle/TSPwithACO

6. *References*

Hurbans, Rishal. *Grokking Artificial Intelligence Algorithms*. Manning Publications Co. 2020, pp. 153-188.

Hurbans, Rishal.Carnival Ant Colony Optimization [source code]. https://github.com/rishal-hurbans/Grokking-Artificial-Intelligence-Algorithms/blob/master/ch06-swarm_intelligence-ants/carnival_aco.py.

Mirjalili, Ali. "How the Ant Colony Optimization algorithm works" *YouTube*, uploaded by Ali Mirjalili, 4 October 2018, https://www.youtube.com/watch?v=783ZtAF4j5g&ab_channel=AliMirjalili.