Swarm Intelligence and Sinergy: Ant Colony for the Traveling Salesman Problem

Nicolás Romero Rodríguez

cod: 20222020023

Systems Analysis

Lecturer: Carlos Andrés Sierra Virgüez

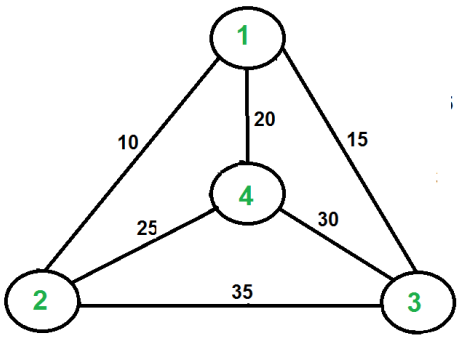
Universidad Distrital Francisco José de Caldas

The objective of this document is to give a simple analysis of an application to solve the Traveling salesman problem (TSP) by using the ant colony optimization algorithm, next the reader is going to find a series of diagrams explaining the problem, tests of the algorithm by changing its parameters and finally some conclusions based on the data obtained.

1. Diagrams and explanations

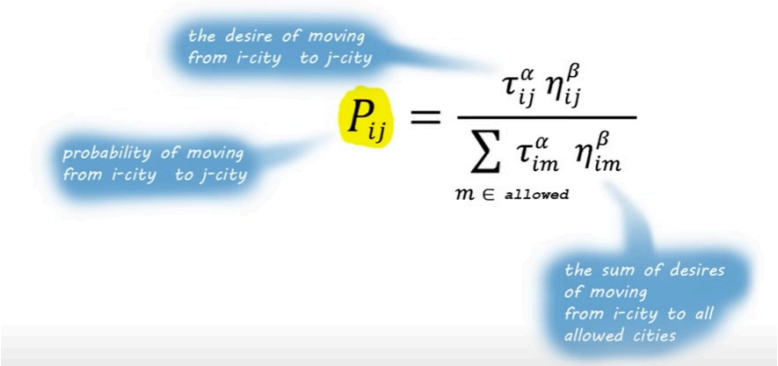
The traveling salesman problem (TSP) is a logistical problem enunciated as: Given a list of cities and distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the original city?

This problem can be represented as a node map as shown below.



In the picture each node represents a city and the number between each pair represents the distance between them.

In order to find the solution to this problem the program proposed for this exercise uses the ant colony optimization algorithm (ACO) The principle of this solution is that iteratively ants are going to travel the paths leaving a trace of pheromones on the edges of the solution graphs. Ants will then chose the next city based on the presence of the pheromone and the distance between each city, in this particular case the formula used to calculate this probability has the following form



Where T represents the amount of pheromone in the trail, n, The distance between the paths, alpha is a constant that defines the influence of the pheromone in the ant's decision, and beta the influence of the distance in the ant's decision.

The program is then going to compare the routes used by the ants and choose the shortest in order to solve the TSP.

2.Implementation

In this case cities are represented by points generated randomly in the space by using the numpy library in python.

```
    cities = []
    for _ in range(number_cities):
        # Generate random coordinates for each city
        x, y, z = np.random.rand(3)
        np.array(cities.append(np.array([x, y, z])))
    return cities
```

Here an empty array is created, then the program generates three random numbers representing the coordinates x, y and z, then it puts them in an array representing the city, and then the city is included in an array that contains every city.

Then the equation that defines the probability for any ant to take a parth was defined applying the equation shown previously and it was implemented as follows:

```
# based on pheromone, distance and alpha and beta parameters, define the preference
# for an ant to move to a city
for i, unvisited_city in enumerate(unvisited):
    # HERE add equation to calculate the probability of moving to a city based on pheromone, distance and alpha and beta parameters

    pheromone_factor = pheromone[current_city, unvisited_city] ** alpha
    distance_factor = 1.0 / calculate_distance(cities[current_city], cities[unvisited_city]) ** beta
    probabilities[i] = pheromone_factor * distance_factor

# normalize probabilities, it means, the sum of all probabilities is 1
# HERE add normalization for calculated probabilities
probabilities /= np.sum(probabilities)
```

The equation here is divided in two parts, the pheromone factor and the distance factor. This was done in order to make the code more readable and in hopes of minimizing the error caused by approximations and memory limitations.

3. Output analysis

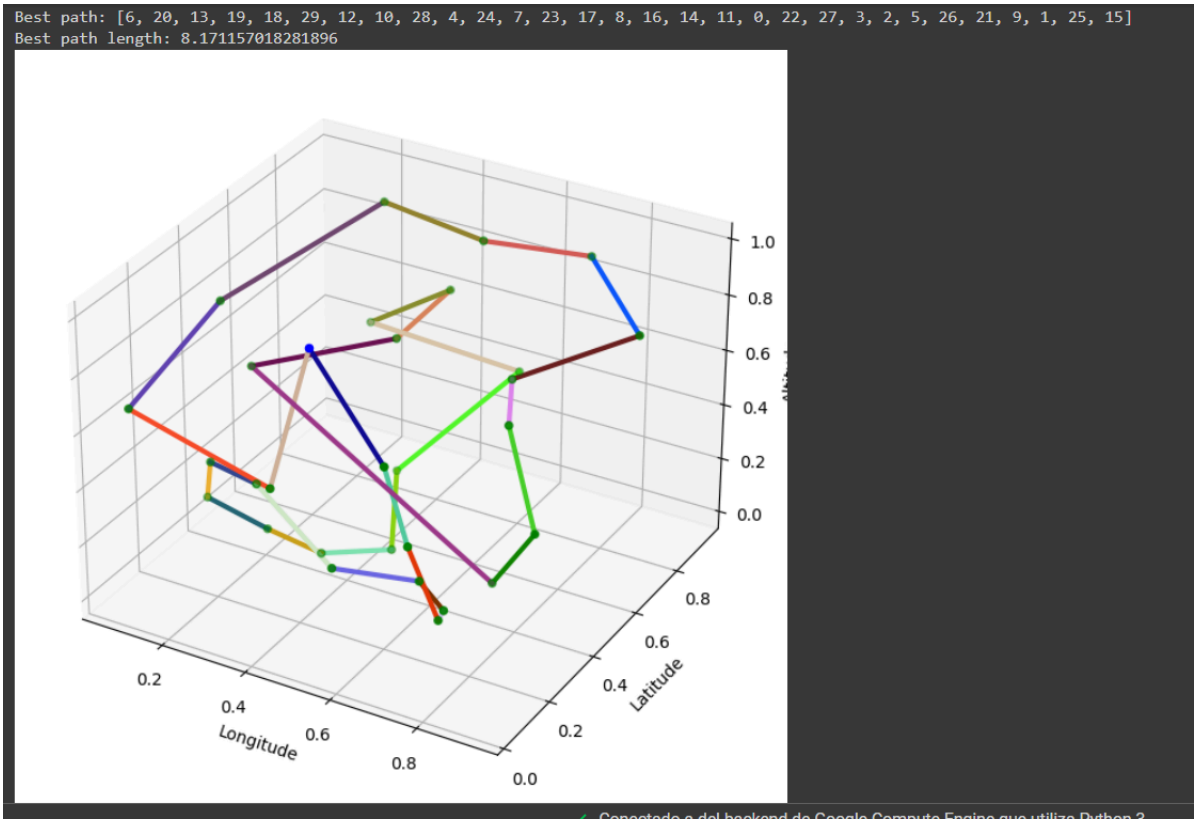
In this section, various outputs of the execution of the program are going to be shown using different initial values for the execution of the program, and an analysis of the output will be presented.

For the first test the initial values are as follows:

```
# model parameters
number_cities = 30
number_ants = 100
number_iterations = 100
alpha = 1
beta = 1
evaporation_rate = 0.5
Q = 1
```

The only parameters not previously explained are evaporation rate and Q, in this case the former represents how much of the pheromone is evaporate in a given amount of time, and the later represents the intensity of the pheromone.

This use case gives the following output:

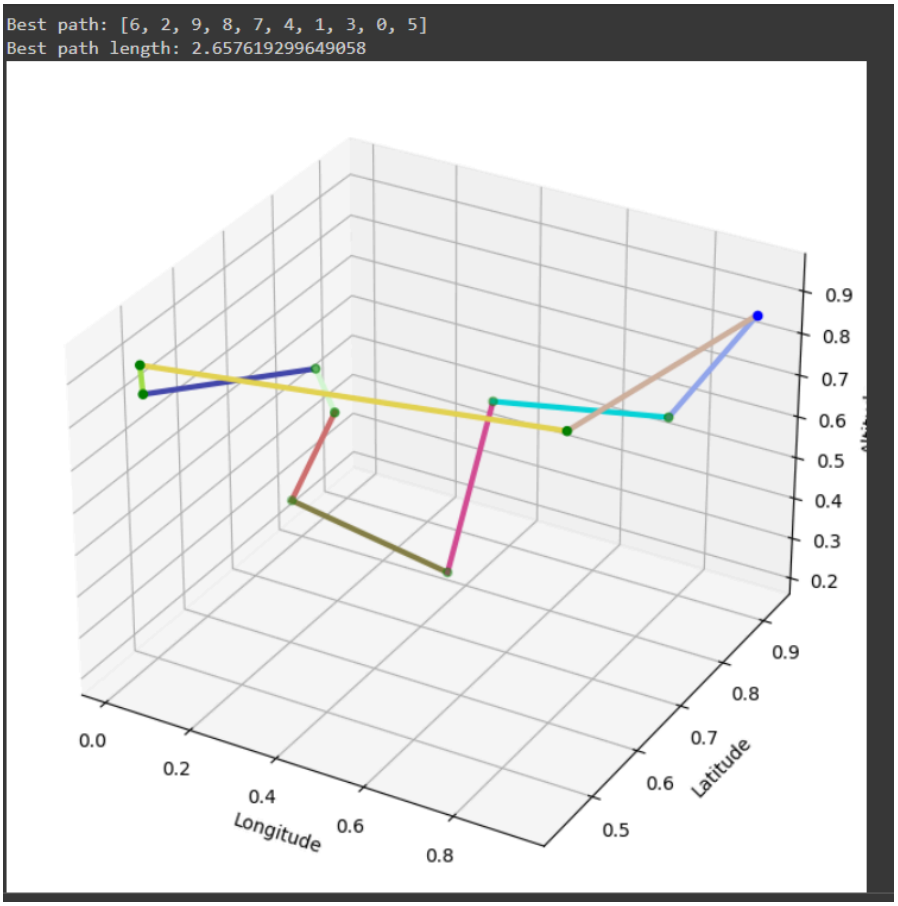


This output will serve as a reference point for other tests.

Now in the next test the number of cities will be reduced from 30 to 10 while maintaining the same values for the other parameters

```
# model parameters
number_cities = 10
number_ants = 100
number_iterations = 100
alpha = 1
beta = 1
evaporation_rate = 0.5
Q = 1
```

which gives the following output:

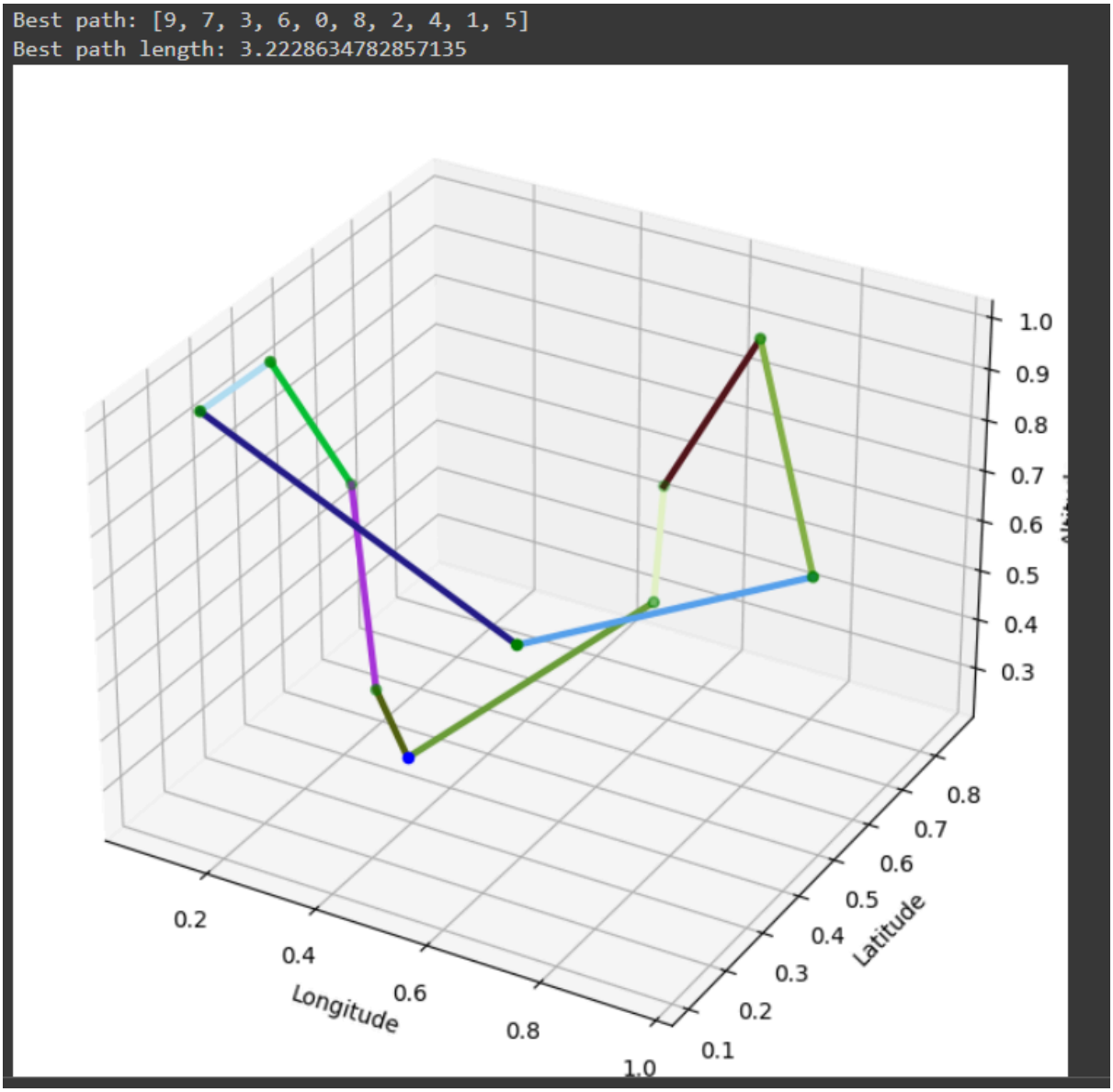


Naturally the reduction of the number of cities will reduce the length of the path proportionally.

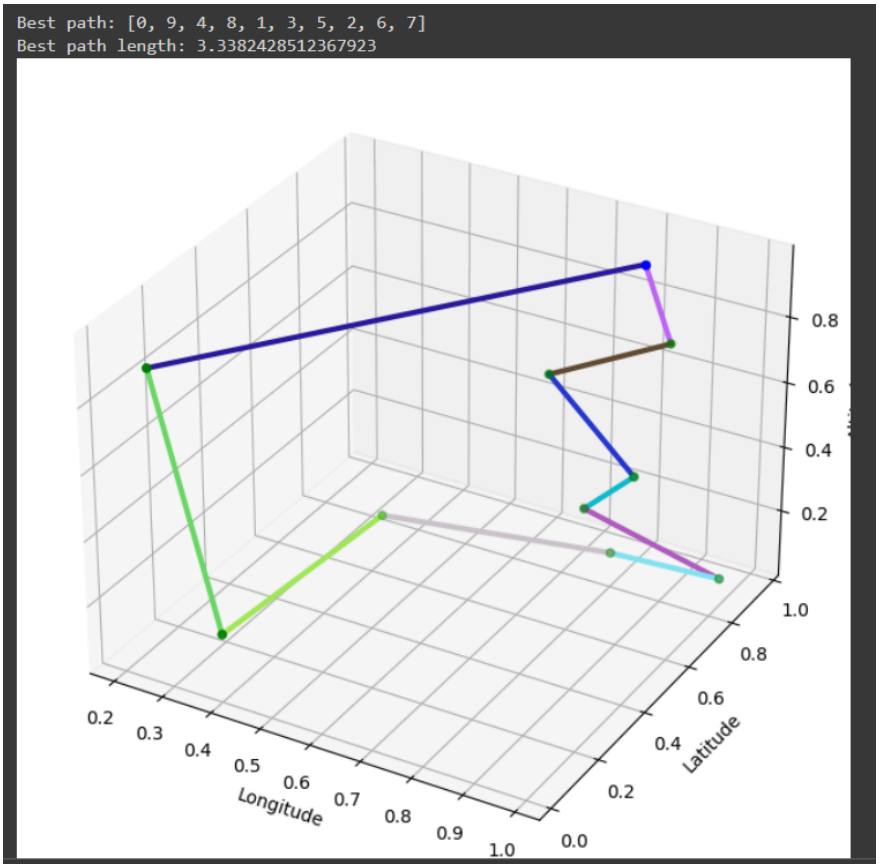
Now the number of cities will be kept the same and the number of ants will be modified

```
# model parameters
number_cities = 10
number_ants = 5
number_iterations = 100
alpha = 1
beta = 1
evaporation_rate = 0.5
Q = 1
```

```
Best path: [9, 7, 3, 6, 0, 8, 2, 4, 1, 5]
Best path length: 3.2228634782857135
```



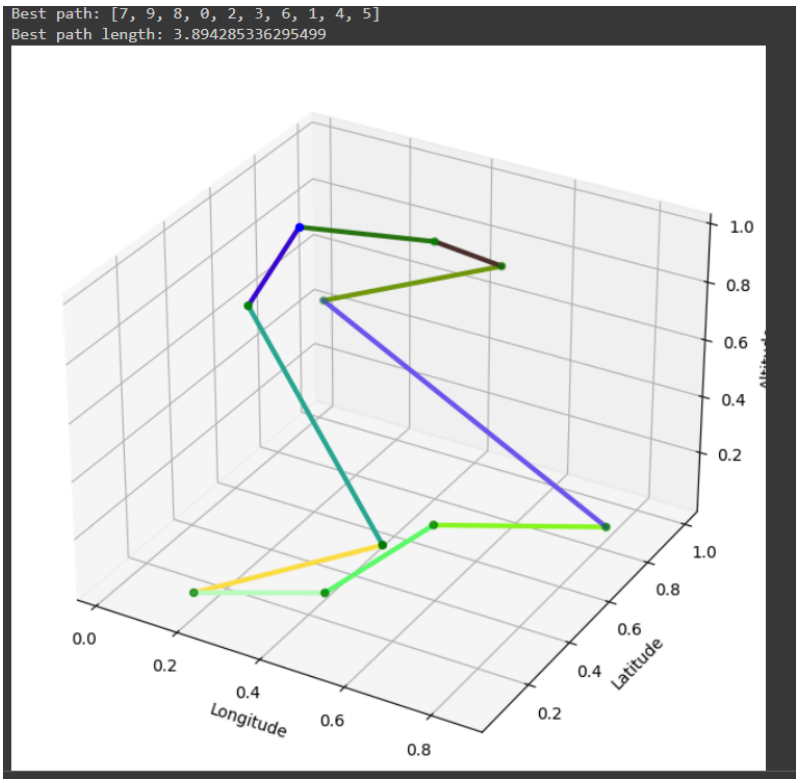
```
# model parameters
number_cities = 10
number_ants = 10
number_iterations = 100
alpha = 1
beta = 1
evaporation_rate = 0.5
Q = 1
```



By reducing the number of ants the best path obtained gets a slight increase and reduces the execution time

Increasing the number of ants also gives a similar result.

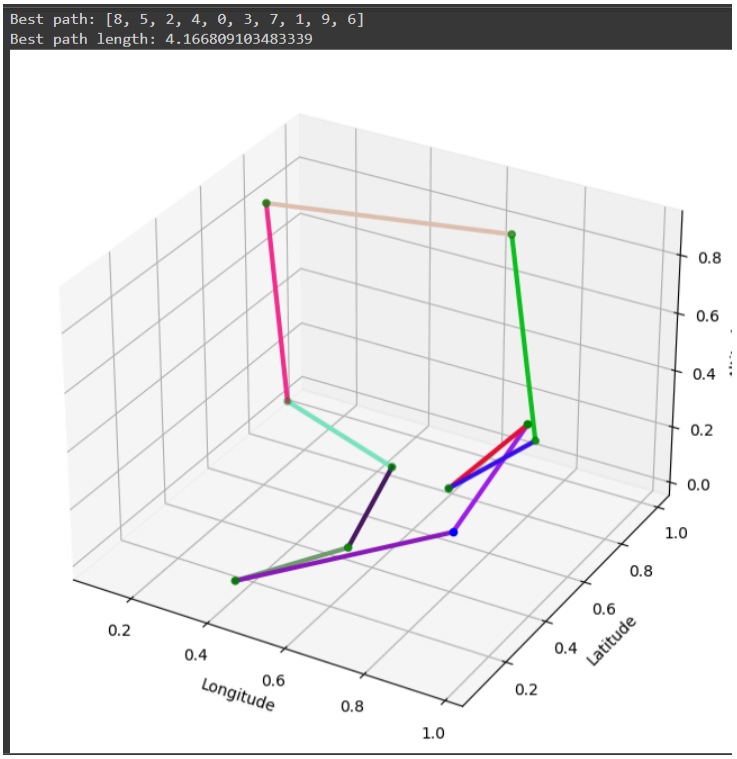
```
# model parameters
number_cities = 10
number_ants = 2000
number_iterations = 100
alpha = 1
beta = 1
evaporation_rate = 0.5
Q = 1
```



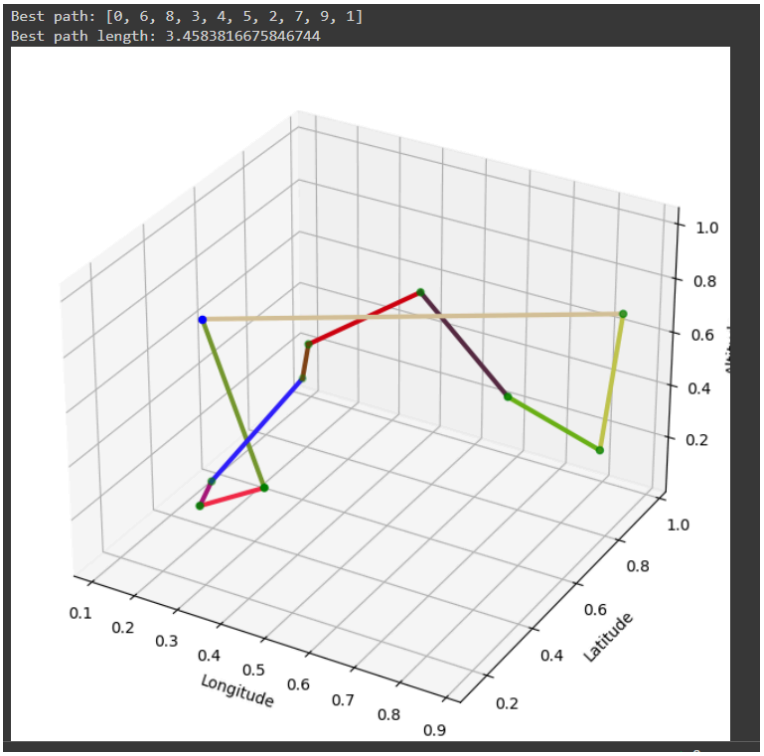
The best length keeps mostly the same but increases the execution time exponentially, but in some executions it does reduce the path length from 3 to 2.

Solely increasing the number of iterations seems to heavily impact the output

```
# model parameters
number_cities = 10
number_ants = 100
number_iterations = 500
alpha = 1
beta = 1
evaporation_rate = 0.5
Q = 1
```



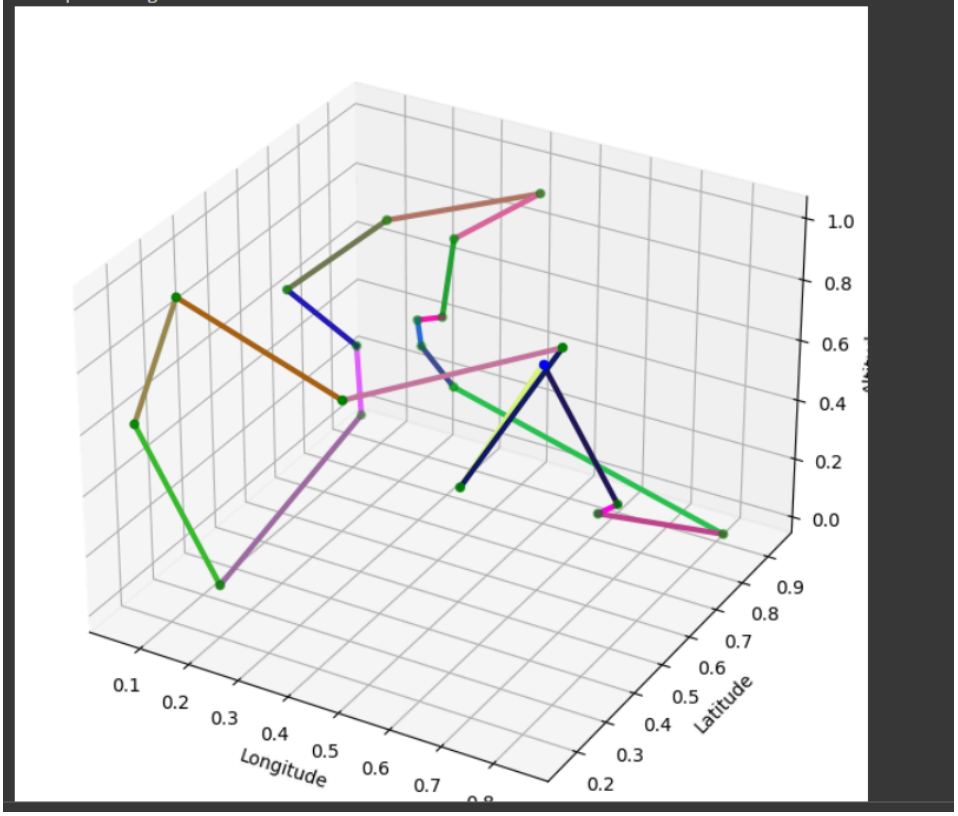
Naturally it increases the running time, but also causes the best length path to variate a lot



For the next test, 20 cities will be used, the same array of cities, but alpha and beta will be modified. This next output will serve as reference

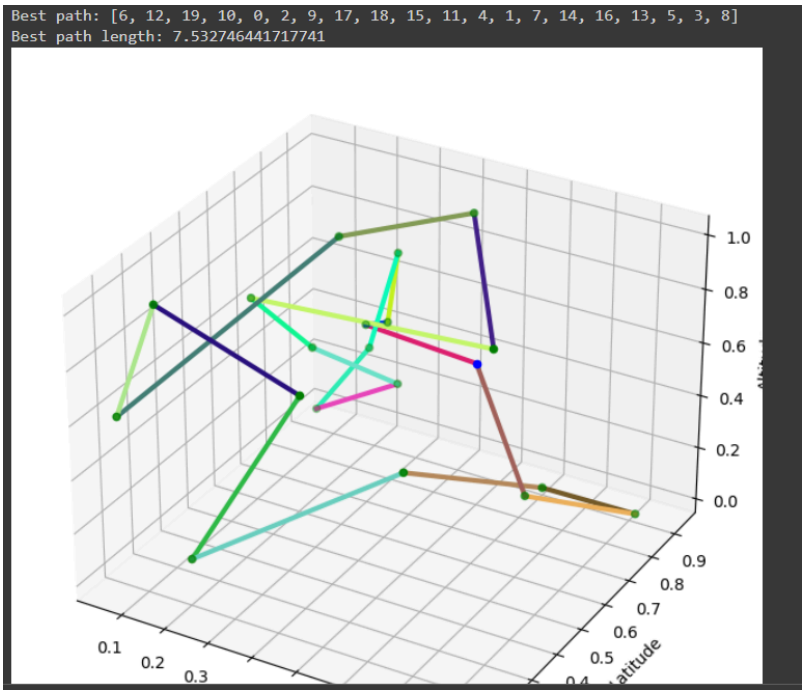
```
# model parameters
number_cities = 20
number_ants = 100
number_iterations = 100
alpha = 1
beta = 1
evaporation_rate = 0.5
Q = 1
```

```
Best path: [15, 4, 1, 16, 14, 17, 9, 2, 18, 11, 19, 10, 12, 0, 6, 7, 3, 5, 13, 8]
Best path length: 6.03757009874741
```



Now changing alpha:

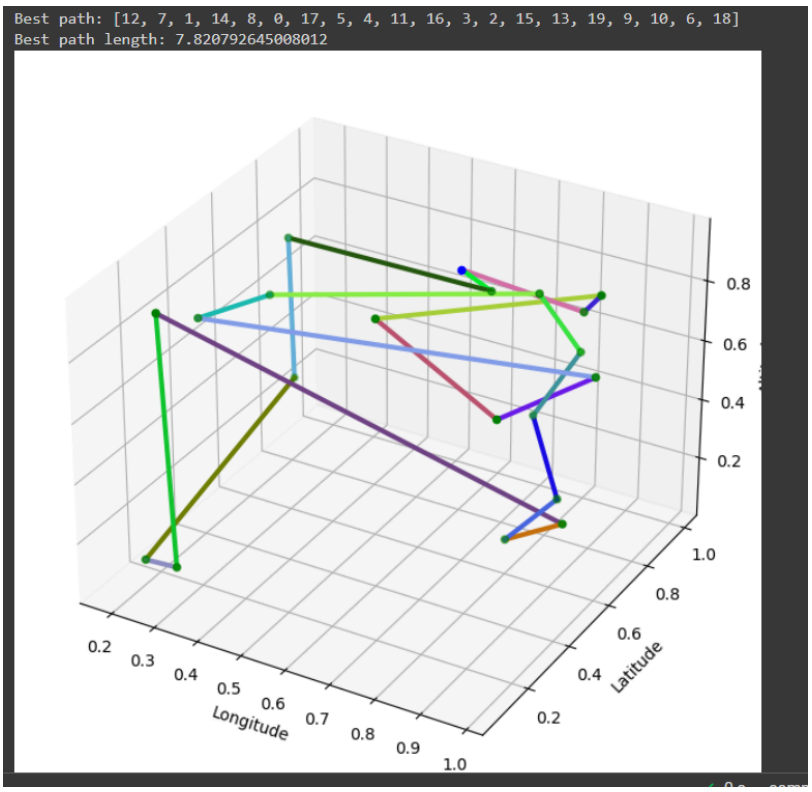
```
# model parameters
number_cities = 20
number_ants = 100
number_iterations = 100
alpha = 0.1
beta = 1
evaporation_rate = 0.5
Q = 1
```



In this case it's possible to see that the best path is now longer.

Now changing Beta:

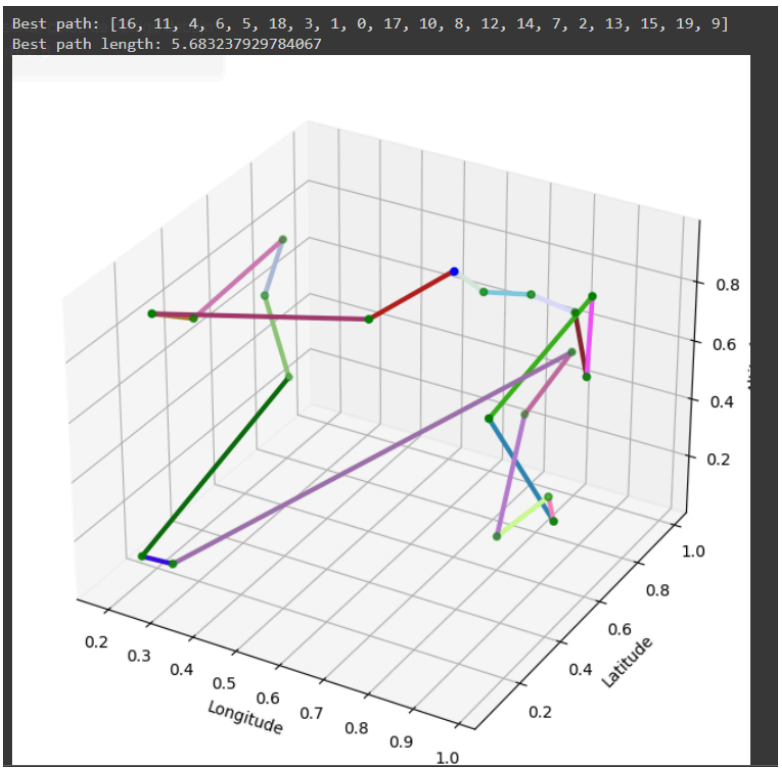
```
# model parameters
number_cities = 20
number_ants = 100
number_iterations = 100
alpha = 1
beta = 0.1
evaporation_rate = 0.5
Q = 1
```



The path is again longer but it doesn't seem to change much from the previous test

Now beta was instead increased considerably compared to alpha

```
# model parameters
number_cities = 20
number_ants = 100
number_iterations = 100
alpha = 1
beta = 20
evaporation_rate = 0.5
Q = 1
```

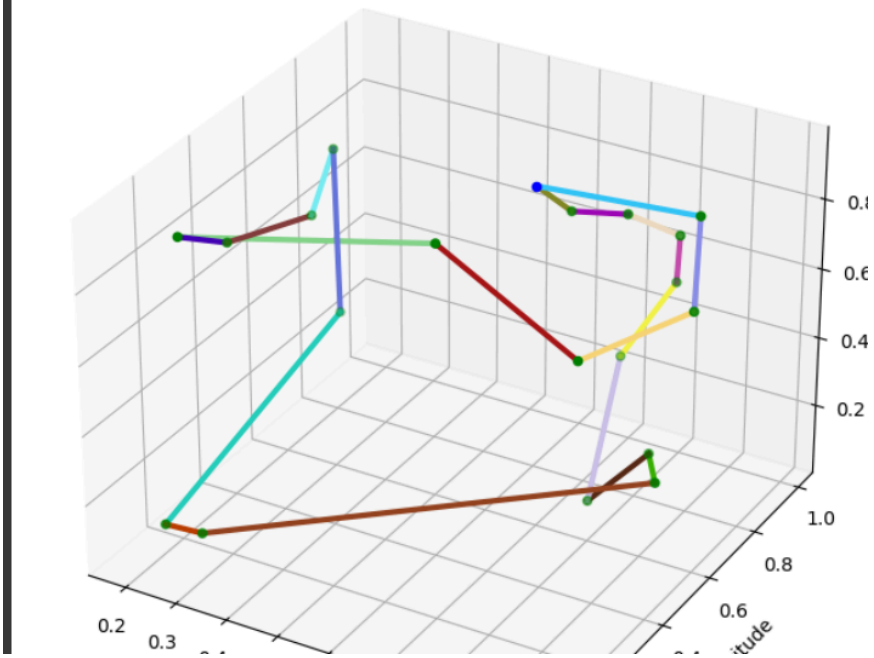


This iteration shows a considerably shorter best path

Increasing Beta again

```
# model parameters
number_cities = 20
number_ants = 100
number_iterations = 100
alpha = 1
beta = 40
evaporation_rate = 0.5
Q = 1
```

```
Best path: [2, 13, 15, 19, 9, 8, 10, 17, 0, 14, 12, 7, 1, 3, 18, 6, 5, 4, 11, 16]
Best path length: 5.683650082989756
```

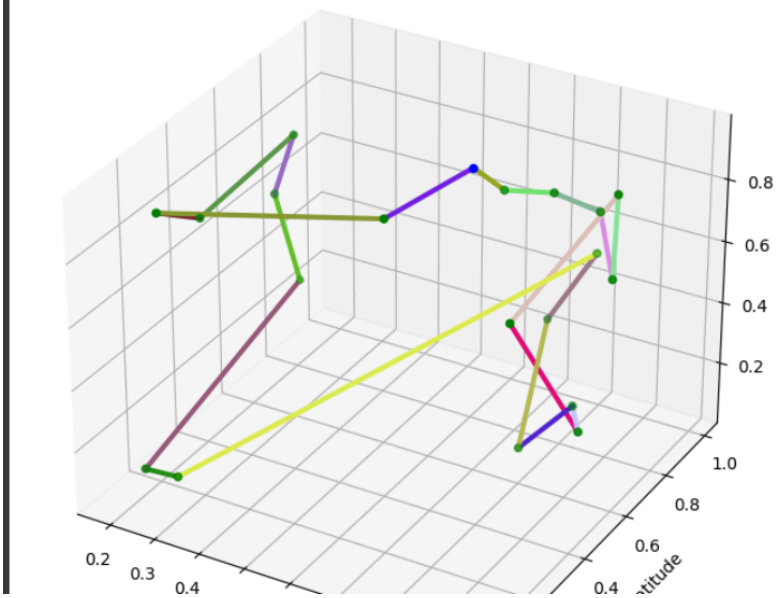


The length of the path keeps on the same range

```
# model parameters
number_cities = 20
number_ants = 100
number_iterations = 200
alpha = 1
beta = 40
evaporation_rate = 0.5
Q = 1
```

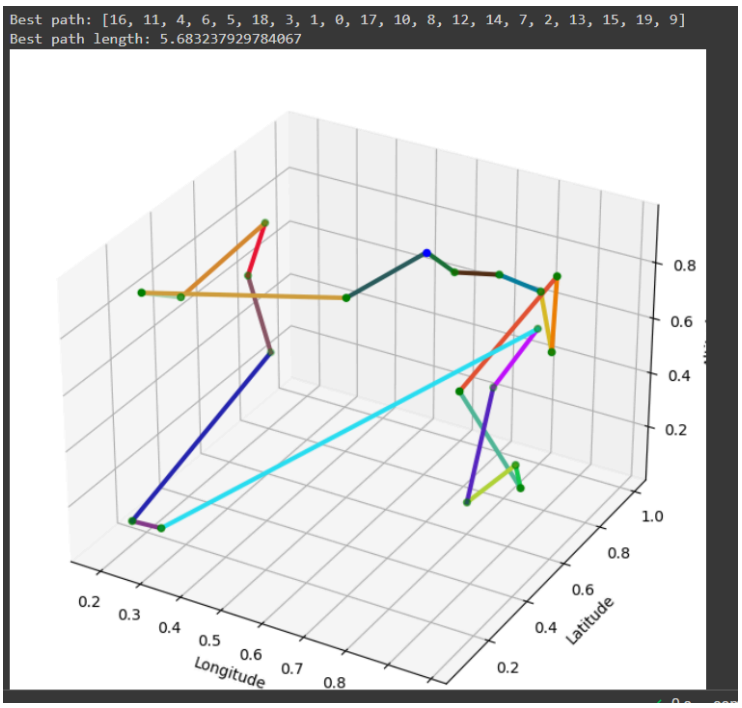
Keeping the increased Beta and doubling the iterations gives us a very similar range

```
Best path: [16, 11, 4, 6, 5, 18, 3, 1, 0, 17, 10, 8, 12, 14, 7, 2, 13, 15, 19, 9]
Best path length: 5.683237929784067
```



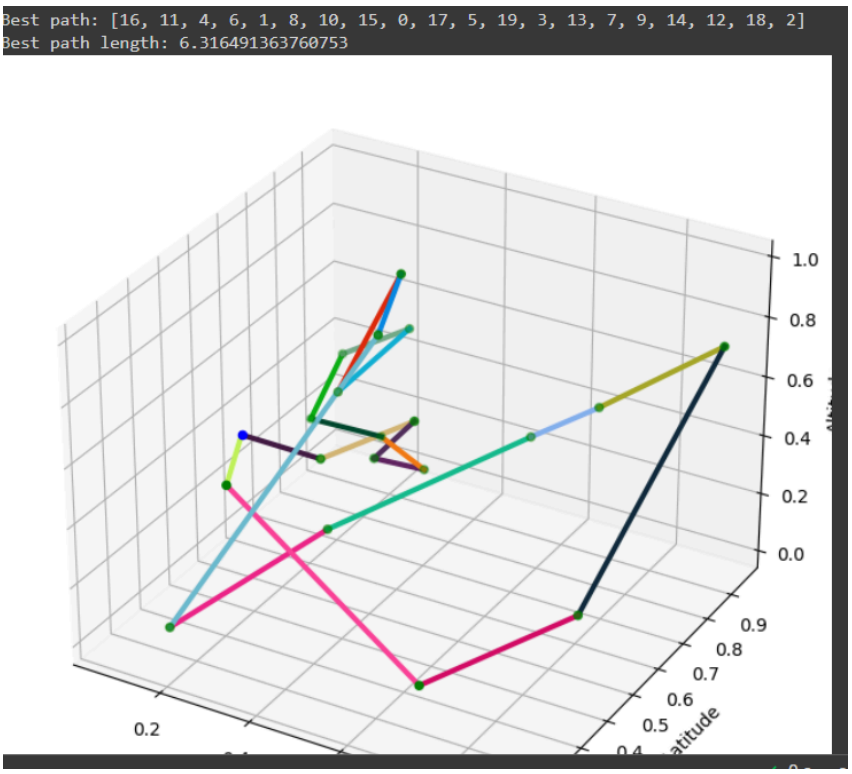
Changing the number of ants, iterations and beta still keeps the same range for this array of cities

```
# Model parameters
number_cities = 20
number_ants = 300
number_iterations = 200
alpha = 1
beta = 40
evaporation_rate = 0.5
Q = 1
```

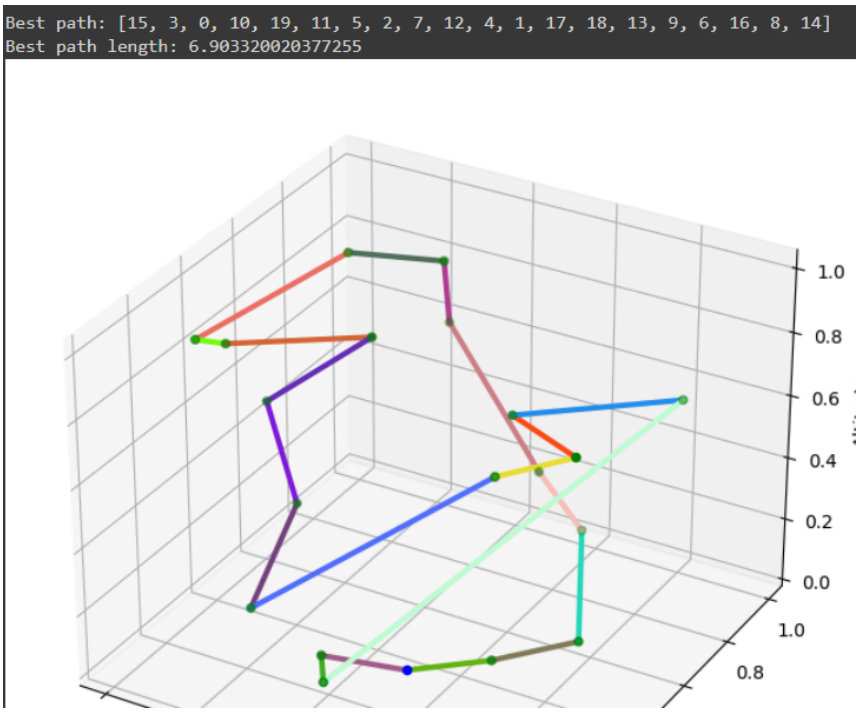


Now changing alpha

```
# model parameters
number_cities = 20
number_ants = 100
number_iterations = 100
alpha = 40
beta = 1
evaporation_rate = 0.5
Q = 1
```



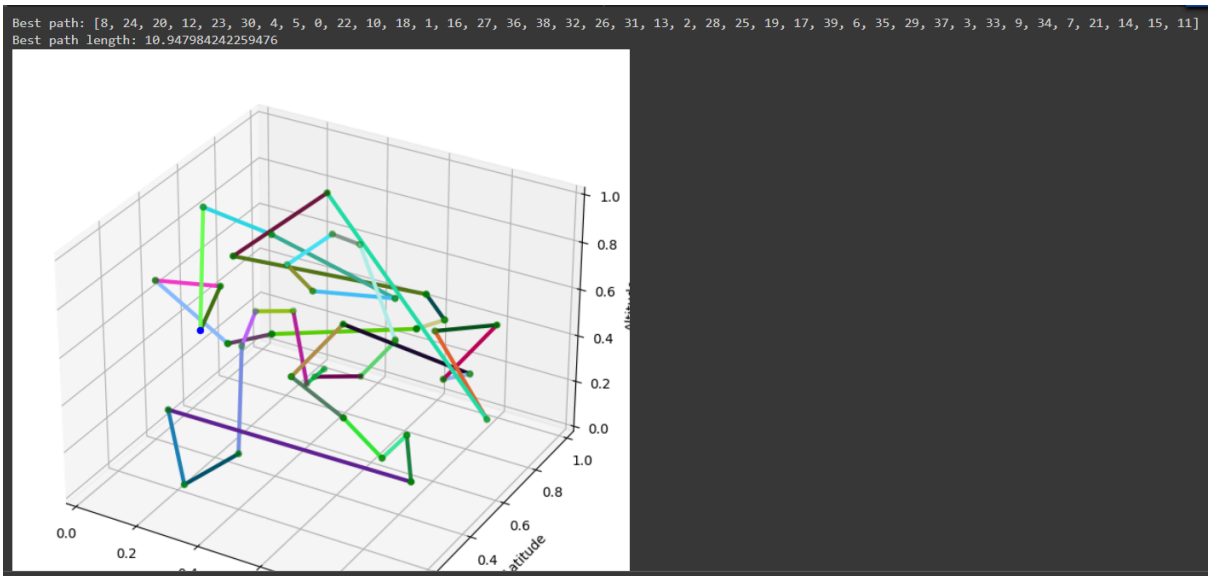
Now the best path is considerably longer than in previous attempts, but in this case this was a newly generated array of cities



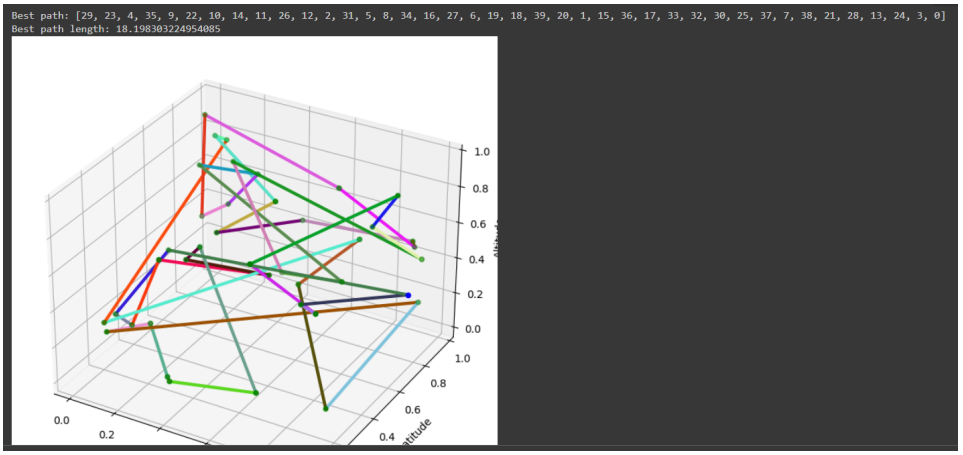
A second attempt with the same parameters and a different array of cities still keeps the best path at a length of 6

reducing the evaporation rate and increasing the number of cities.

```
# model parameters
number_cities = 20
number_ants = 100
number_iterations = 100
alpha = 40
beta = 1
evaporation_rate = 0.1
Q = 1
```




```
# model parameters
number_cities = 40
number_ants = 100
number_iterations = 1
alpha = 40
beta = 1
evaporation_rate = 0.1
Q = 1
```

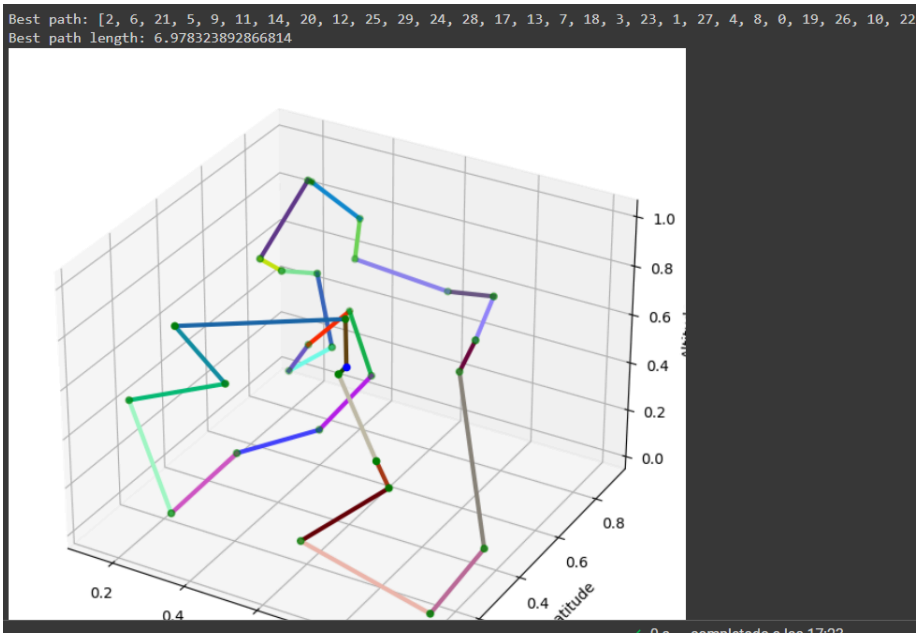


Curious thing to note is that with the changed alpha and reduced iterations in one execution the algorithm found a shorter path compared to another case with 100 iterations

The final experiment was to try and find the combination of parameters that could produce the shortest path possible with 30 cities. The initial parameters were as follows

```
# model parameters
number_cities = 30
number_ants = 100
number_iterations = 100
alpha = 1
beta = 1
evaporation_rate = 0.5
Q = 1
```

This combination of variables produce an output of around 7.5 after many tests though in one of them it produced the following output



Which was the shortest path found even after all the other experiments, but short paths are hard to replicate with these parameters compared to other combinations.

Decreasing the evaporation rates helps getting shorter paths, usually keeping the lengths below 7.4 after many tests.

The most consistent way to find short paths was to keep both alpha and beta the same, reduce the evaporation rate, and increasing the intensity of the pheromone, this way the shortest path found was 7.296127601568715 and most tests were a little above this result, but notably shorter compared to other variations, this were the specific parameters used

```
# model parameters
number_cities = 30
number_ants = 100
number_iterations = 100
alpha = 1
beta = 1
evaporation_rate = 0.1
Q = 4
```

In this experiment the variables that caused the path length to increase the most was the number of iterations, increasing the path length up to 8 while every other test kept the path length around 7, and the variable that reduced the lengths the most was Q, and alpha and beta start making notable changes when their values where above 4.

4. Difficulties

In general terms there weren't many problems in the development of this activity, the information required to complete the code were easily accessible in the internet and the implementation of those pieces of code was very simple, the only repetitive task was to execute each individual test for the code, copying and pasting cuts does get tedious at times, and due to the large amount of combinations to produce different results the exercise requires a lot of tests, which is not really a hard task to do, but a time consuming one.

5. Conclusions

In general the variable that changes the outputs the most is the amount of cities, but, changing individual variables (excluding number of cities) don't have a big impact in the length of the path obtained in the end

Having more iterations helps to have more uniform results, because by reducing the number of iterations the length of the path tends to change a lot.

The best combinations to get the shortest paths are when alpha and beta are the same, the evaporation rate is low and the concentration of pheromone is high.

Due to the random factor of the creation of cities certain results are difficult to replicate, and also the effects of changing the value of the parameters may not be so noticeable because of this.