LAURA DANIELA MUÑOZ IPUS

20221020022

SYSTEM ANALYSIS

WORKSHOP_2: Swarm Intelligence and Sinergy: Ant Colony for the Traveling Salesman Problem

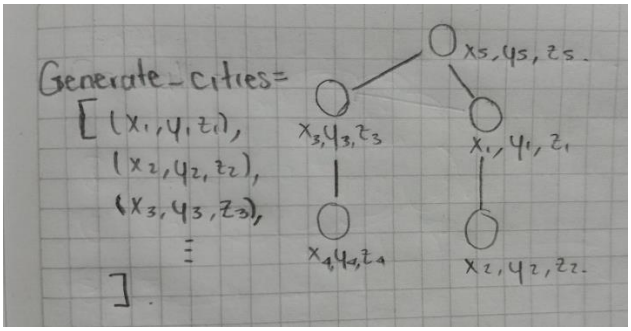
- Problem Analysis:

It´s wanted to obtain the shortest paths to ship some products, however, this is very similar to what TSP is, and to solve this problem you can use Ant Colony Optimization (ACO).

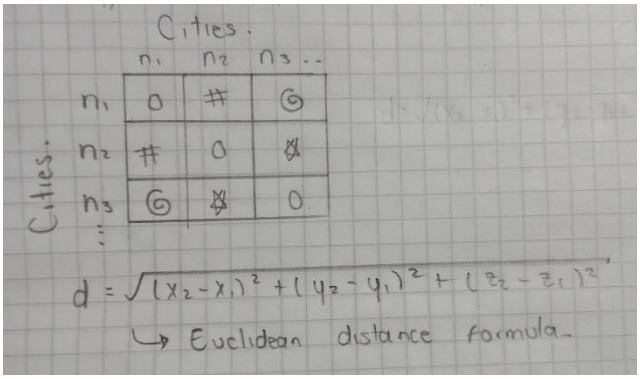
TSP is about a salesman must visit a set of cities exactly once and return to the original city, thus finding the shortest routes.

To address this problem, the following analysis was performed.

First, random points in space with coordinates x, y, and z, which represent the cities, must be generated.



The distances between the cities must be calculated using the euclidean distance formula, which calculates the distance between two points with coordinates x, y, z



To obtain the distance that exists between all the cities can be represented in a matrix of size M x M (M is the total size of cities) The distances are stored in the matrix, for the relation of the same city can be represented as 0.

By having a list of cities and a formula to calculate the distance from point to point, the ant colony optimization (ACO) can be implemented.

To go deeper into this algorithm, it is about a given number of ants that make multi-point foraging trips and at the same time look for the shortest routes to accomplish their goal. The ants are influenced by two factors, alpha (which determines the extent to which the ants are influenced by pheromone traces left by other ants) and beta (which determines the extent to which the ants are influenced by the distance to be traveled). It must be taken into account that the pheromone has a certain intentionality and can evaporate along the way. The ants start having a random behavior but then the probabilities that a path will be traveled by an ant are found by the factors mentioned above, resulting in a pattern of travel and thus find the best routes. The cities and the routes have to be plotted in 3D.

To start with the algorithm, the pheromone matrix is initialized with values of 1 (in the code this is done with the function np.ones from the numpy function package), which means that all routes have the same probability of being selected at the beginning.

The route of the ants is iterable, so when iterating we obtain a list with the routes made by the total number of ants.

To start the route and determine which cities have already been visited, a logical analysis has to be done.

when the ants make their routes is a repetitive cycle, so it can be iterated. At the moment of iterating we can obtain a list of routes and a list of the size of these, to later analyze which is the shortest route.

To analyze which cities have already been visited, a logical reasoning is needed,

Each ant will contain a list of unvisited cities, which it will have to travel only once.

0 1 2 3 4 n

unvisited = [F, F, F, F, F .... nF]

n la cantidad de ciudades -1

Each city will be represented by its index

From this, a random city (a random point in space) is taken from the list of false and is taken as the starting point (current_city) of an ant, this city when visited is taken as True and is added to the list of routes, and so on for all the ants and with all the cities.

PROBABILITY

To know the route that each ant is going to take, the probability has to be calculated, depending on alpha and beta, the distance and the pheromone that each ant expels.

To develop an equation to do this you have to iterate the cities not visited with the index with which it corresponds (i), this is done with a python function called enumerate. To find the probability of each i the pheromone is multiplied by the alpha and summed with the distance multiplied by the beta.

When summing the pheromone trail and the inverse of the distance, the algorithm takes into account both the attractiveness of the pheromone trail and the distance. It has to be added because the distance is larger than the pheromone trail, so subtracting them would give negative numbers and you cannot do probabilistic data with negative numbers.

Each city will have a probability of being visited, some having more probabilities than others.

For each city will have a probability of a part of the total number of cities.

n = total of cities

cities = measured by their index

0 = n /a 1= n /b 2= n /c 3= n /d ….. k / n-1

The sum of all the denominators must give n for the probability to be 1.

$$\frac{n}{a+b+c+d+\cdots+k} = 1$$

This is how we normalize the probability, we have to normalize the probabilities to optimize the algorithm and not make the data so heavy to be implemented.

RELATIONSHIP BETWEEN PATH AND PATH_LENGHT

In this part of the code, the zip( ) function is used to combine two lists, paths and path_lengths, into a single iterable. This allows iterating over both lists simultaneously in the following for loop.

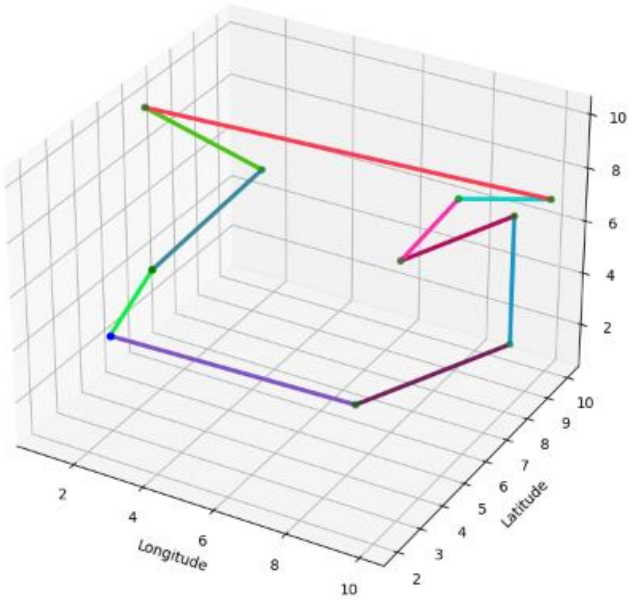
GRAPH OF THE DATA

To analyze the behavior of the algorithm we set the following parameters found in the code part, changing certain parameters to see the behavior of the 3D graph

- Analyze the graph with 10 points in space.

```python
# # model parameters
# cities = cities
# n_ants = 100
# n_iterations = 100
# alpha = 1
# beta = 1
# evaporation_rate = 0.5
# Q = 1

# HERE create list of cities
cities = generate_cities(10) #LLamado de la funcion

# HERE call ant_colony_optimization function
aco = ant_colony_optimization(cities,  100 , 100 , 1, 1, 0.5, 1 )

best_path = aco[0] # se toma el primer el elemento de la lista de best_path, que representa al mejor camino a recorrer
best_path_length = aco[1]

```

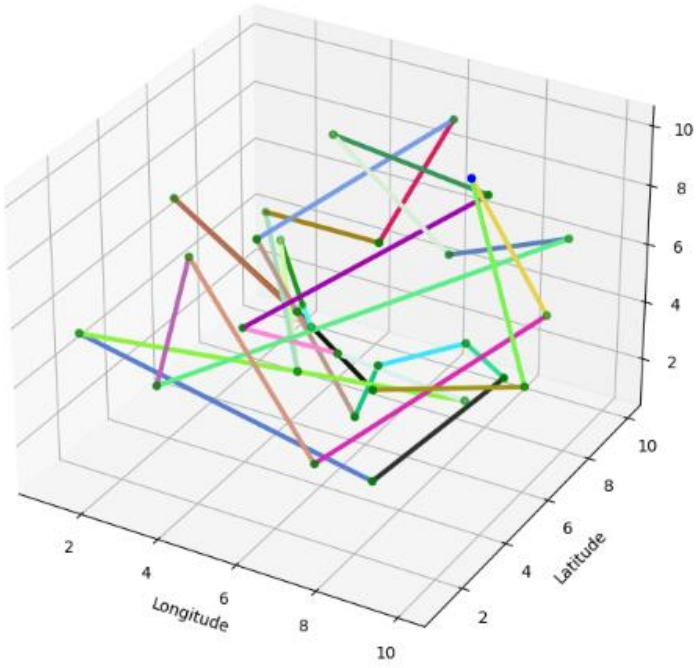
```
Best path: [1, 6, 7, 0, 8, 4, 3, 5, 2, 9]
Best path length: 38.58522752312703
```



- Analyze the graph with 30 points in space and 50 iterations.

```
# HERE create list of cities
cities = generate_cities(30) #LLamado de la funcion

# HERE call ant_colony_optimization function
aco = ant_colony_optimization(cities,  100 , 50 , 1, 1, 0.5, 1 )

best_path = aco[0] # se toma el primer el elemento de la lista de best_path, que representa al mejor camino a recorrer
best_path_length = aco[1]

```

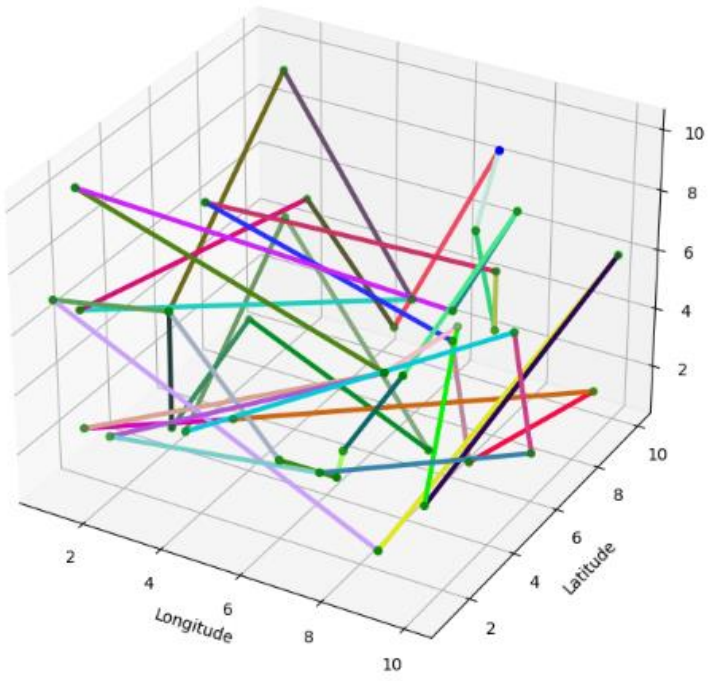
```
Best path: [10, 20, 12, 23, 11, 29, 7, 15, 19, 5, 1, 9, 17, 24, 8, 21, 3, 13, 26, 4, 18, 25, 6, 27, 14, 2, 22, 28, 0, 16]
Best path length: 166.3257676102993
```



- Analyze the graph with 500 ants, 50 iterations and 40 points in space.

```
# HERE create list of cities
cities = generate_cities(40) #LLamado de la funcion

# HERE call ant_colony_optimization function
aco = ant_colony_optimization(cities,  500 , 50 , 1, 1, 0.5, 1 )

best_path = aco[0] # se toma el primer el elemento de la lista de best_path, que representa al mejor camino a recorrer
best_path_length = aco[1]

```

```
Best path: [28, 35, 37, 25, 38, 17, 36, 2, 30, 20, 10, 0, 27, 8, 5, 26, 7, 19, 11, 14, 31, 24, 15, 29, 34, 1, 23, 33, 32, 22, 4, 16, 6, 18, 13, 39, 21, 9, 12, 3]
Best path length: 233.11199317258067
```



CONCLUSIONS

- In all the graphs there is a blue point, which indicates the starting point (current city), where the route starts and returns to that starting point.
- Increasing the number of ants does not necessarily increase the lines that connect the cities, it increases the iterations and makes the process heavier.
- In the best route it shows us the index of the city.
- Increasing the number of cities increases the number of the size of the best path.