Workshop2

Juliana Camila Rincón Rojas 20231020013

30/03/2024

The problem at hand is that of a delivery person who wishes to find the best routes for delivering their orders. To achieve this, an ant colony algorithm is programmed, which finds the optimal route among the desired points.

This work demonstrates how the ant colony algorithm can help solve various problems whose main focus is to navigate through different points and find the best routes, as in this case, for order delivery.

```python
import numpy as np


def generate_cities(number_cities: int) -> list:
    # HERE Generate random 3D Points using numpy generator
    cities = np.random.rand(number_cities, 3)
    return cities


def calculate_distance(point_1: np.array, point_2: np.array) -> float:
    point_1 = np.array(point_1)
    point_2 = np.array(point_2)

    # HERE return distance between two points using euclidean distance formula
    distance = np.linalg.norm(point_1 - point_2)
    return distance
```

The generate_cities function creates a list of cities with random coordinates in 3D space. It takes an integer parameter specifying the number of cities to generate and returns a list. calculate_distance function computes the Euclidean distance between two given points in 3D space.

```python
def ant_colony_optimization(
    cities, n_ants, n_iterations, alpha, beta, evaporation_rate, Q
):
    number_cities = len(cities)
    pheromone = np.ones((number_cities, number_cities))

    # initialize output metrics
    best_path = None
    best_path_length = np.inf

    # per each iteration the ants will build a path
    for iteration in range(n_iterations):
        paths = []  # store the paths of each ant
        path_lengths = []

        for ant in range(n_ants):
            visited = [False] * number_cities

            # you could start from any city, but let's start from a random one
            current_city = np.random.randint(number_cities)
            visited[current_city] = True
            path = [current_city]
            path_length = 0
```

```python
while False in visited:  # while there are unvisited cities
    unvisited = np.where(np.logical_not(visited))[0]
    probabilities = np.zeros(len(unvisited))

    # 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 / 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)

    next_city = np.random.choice(unvisited, p=probabilities)
    path.append(next_city)
    # increase the cost of move through the path
    path_length += calculate_distance(
        cities[current_city], cities[next_city]
    )
    visited[next_city] = True
    # move to the next city, for the next iteration
    current_city = next_city
```

```python
        paths.append(path)
        path_lengths.append(path_length)

        # update with current best path, this is a minimization problem
        if path_length < best_path_length:
            best_path = path
            best_path_length = path_length

    # remove a bit of pheromone of all map, it's a way to avoid local minima
    pheromone *= evaporation_rate

    # current ant must add pheromone to the path it has walked
    for path, path_length in zip(paths, path_lengths):
        for i in range(number_cities - 1):
            pheromone[path[i], path[i + 1]] += Q / path_length
        pheromone[path[-1], path[0]] += Q / path_length
return best_path, best_path_length
```

The ant_colony_optimization function takes input parameters such as cities' coordinates, the number of ants, iterations, and factors influencing ant decision-making.

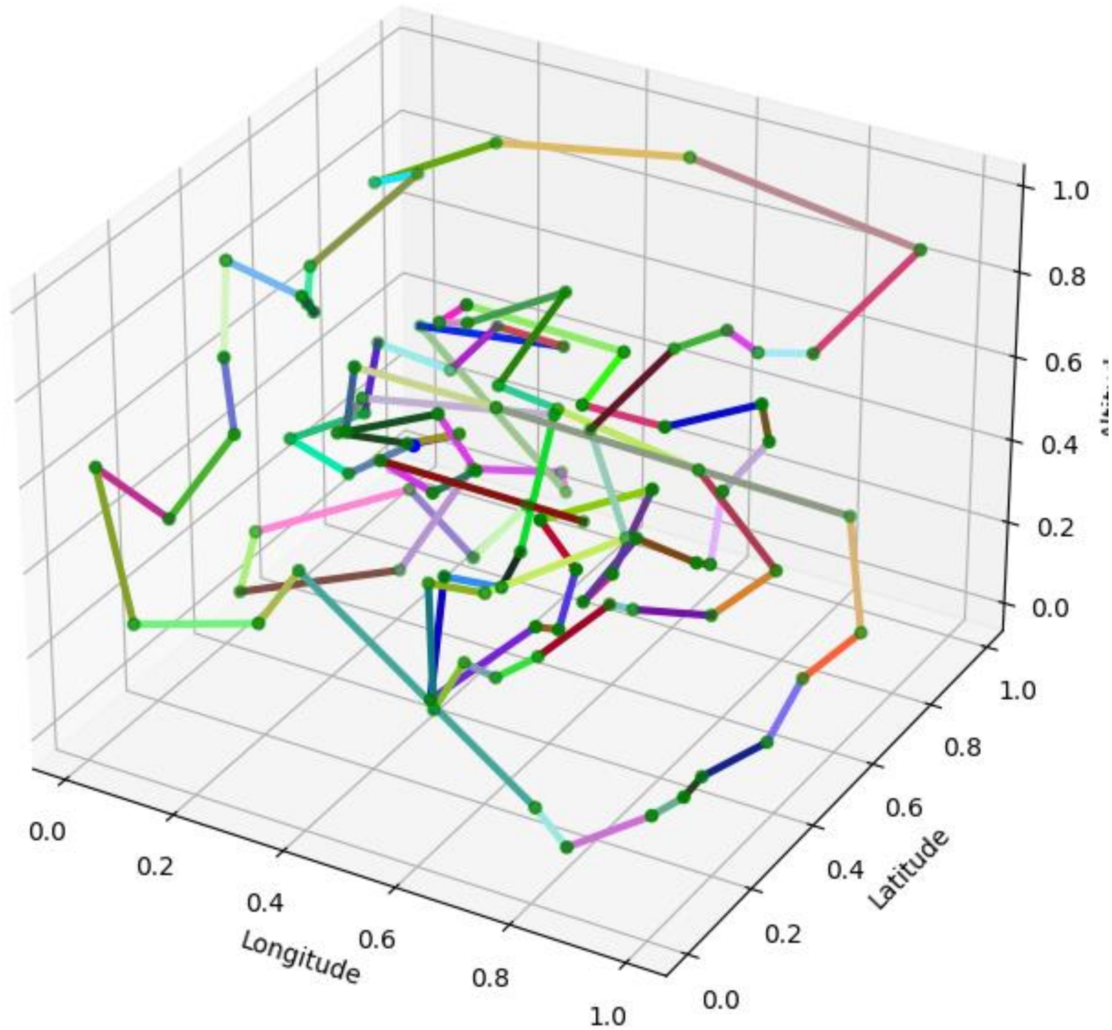
The algorithm initializes pheromone levels and tracks the best path found. Ants construct paths biased by pheromone levels and distances. Construct paths, pheromone levels are updated based on path quality. Finally, the function returns the best path found and its length.

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

# HERE create list of cities
cities = generate_cities(number_cities)

# HERE call ant_colony_optimization function
best_path, best_path_length = ant_colony_optimization(cities, number_ants, number_iterations, alpha, beta, evaporation_rate, Q)
```

Here are described the parameters used when running the program, you can modify them and obtain different results.

```python
def random_color() -> list:
    return [random.random(), random.random(), random.random()]


def plot_aco_route(cities: np.array, best_path: list):
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection="3d")

    for i in range(len(best_path) - 1):
        ax.plot(
            [cities[best_path[i], 0], cities[best_path[i + 1], 0]],  # x axis
            [cities[best_path[i], 1], cities[best_path[i + 1], 1]],  # y axis
            [cities[best_path[i], 2], cities[best_path[i + 1], 2]],  # z axis
            c=random_color(),
            linestyle="-",
            linewidth=3,
        )

    ax.plot(
        [cities[best_path[0], 0], cities[best_path[-1], 0]],
        [cities[best_path[0], 1], cities[best_path[-1], 1]],
        [cities[best_path[0], 2], cities[best_path[-1], 2]],
        c=random_color(),
        linestyle="-",
        linewidth=3,
    )

    ax.scatter(cities[0, 0], cities[0, 1], cities[0, 2], c="b", marker="o")
```

The random_color function simply produces a random color in RGB format. It returns a list containing these three random values. plot_aco_route function plots the cities and the best path found by the ACO algorithm.



Finally, if we run the program with 100 completely random cities, we obtain the graph. It starts at the blue point and traverses these 100 locations in the best route. We can conclude that the problem can be solved using the ant colony optimization algorithm.