Lab-3-Ant Colony Optimization for the Traveling Salesman Problem

Code:

```
import numpy as np
import random

class AntColony:
    def __init__(self, cities, num_ants, alpha, beta, rho, q0, iterations):
        self.cities = cities
        self.num_ants = num_ants
        self.alpha = alpha
        self.beta = beta
        self.rho = rho  # pheromone evaporation rate
        self.q0 = q0    # exploration vs. exploitation parameter
        self.iterations = iterations
        self.distance_matrix = self.calculate_distance_matrix()
        self.pheromone = np.ones(self.distance_matrix.shape) / len(cities)

    def calculate_distance_matrix(self):
        num_cities = len(self.cities)
        distance_matrix = np.zeros((num_cities, num_cities))
        for i in range(num_cities):
            for j in range(num_cities):
                distance_matrix[i][j] = np.linalg.norm(np.array(self.cities[i]) - np.array(self.cities[j]))
        return distance_matrix

    def select_next_city(self, current_city, visited):
        probabilities = []
        for next_city in range(len(self.cities)):
            if next_city not in visited:
                pheromone = self.pheromone[current_city][next_city] ** self.alpha
                heuristic = (1 / self.distance_matrix[current_city][next_city]) ** self.beta
                probabilities.append(pheromone * heuristic)
            else:
                probabilities.append(0)
        probabilities = np.array(probabilities)
        probabilities /= probabilities.sum()  # Normalize
        return np.random.choice(range(len(self.cities)), p=probabilities)
```

```
    def construct_solution(self):
        for _ in range(self.num_ants):
            visited = [0]
            current_city = 0
            for _ in range(len(self.cities) - 1):
                current_city = self.select_next_city(current_city, visited)
                visited.append(current_city)
            visited.append(0)  # Return to starting city
            yield visited

    def update_pheromones(self, solutions):
        self.pheromone *= (1 - self.rho)  # Evaporation
        for solution in solutions:
            distance = self.calculate_tour_length(solution)
            for i in range(len(solution) - 1):
                self.pheromone[solution[i]][solution[i + 1]] += 1 / distance

    def calculate_tour_length(self, tour):
        return sum(self.distance_matrix[tour[i]][tour[i + 1]] for i in range(len(tour) - 1))

    def run(self):
        best_solution = None
        best_length = float('inf')

        for _ in range(self.iterations):
            solutions = list(self.construct_solution())
            self.update_pheromones(solutions)

            for solution in solutions:
                length = self.calculate_tour_length(solution)
                if length < best_length:
                    best_length = length
                    best_solution = solution

        return best_solution, best_length


def main():
    # User input for cities
    num_cities = int(input("Enter the number of cities: "))
```

```
    cities = []

    for i in range(num_cities):
        x, y = map(float, input(f"Enter coordinates for city {i + 1} (x y): ").split())
        cities.append((x, y))

    # Parameters for ACO
   num_ants = int(input("Enter the number of ants: "))
   alpha = float(input("Enter the importance of pheromone (alpha): "))     # Importance of
pheromone
    beta = float(input("Enter the importance of heuristic (beta): "))          # Importance of heuristic
    rho = float(input("Enter the pheromone evaporation rate (rho): "))         # Evaporation rate
    q0 = float(input("Enter the exploration parameter (q0): "))               # Exploration parameter
    iterations = int(input("Enter the number of iterations: "))              # Number of iterations



    # Run the ACO algorithm
    aco = AntColony(cities, num_ants, alpha, beta, rho, q0, iterations)
    best_solution, best_length = aco.run()

    print("Best solution:", best_solution)
    print("Best tour length:", best_length)


if __name__ == "__main__":
    main()
```

Output:

```
Enter the number of cities: 5
Enter coordinates for city 1 (x y): 0 0
Enter coordinates for city 2 (x y): 1 5
Enter coordinates for city 3 (x y): 5 2
Enter coordinates for city 4 (x y): 3 3
Enter coordinates for city 5 (x y): 6 1
Enter the number of ants: 1000
Enter the importance of pheromone (alpha): 1.0
Enter the importance of heuristic (beta): 2.0
Enter the pheromone evaporation rate (rho): 0.5
Enter the exploration parameter (q0): 0.5
Enter the number of iterations: 150
Best solution: [0, 4, 2, 3, 1, 0]
Best tour length: 17.66049070851008
```