Ant Colony Optimization for the Traveling Salesman Problem

```
import numpy as np

import random


class ACO:

    def __init__(self, n_ants, n_iterations, alpha, beta, rho, pheromone_deposit, cities):

        self.n_ants = n_ants

        self.n_iterations = n_iterations

        self.alpha = alpha  # importance of pheromone

        self.beta = beta    # importance of heuristic information

        self.rho = rho      # pheromone evaporation rate

        self.pheromone_deposit = pheromone_deposit  # pheromone deposit for the best path

        self.cities = cities

        self.num_cities = len(cities)

        self.distances = self.calculate_distances(cities)

        self.pheromone_matrix = np.ones((self.num_cities, self.num_cities))


    def calculate_distances(self, cities):

        distances = np.zeros((len(cities), len(cities)))

        for i, (x1, y1) in enumerate(cities):

            for j, (x2, y2) in enumerate(cities):

                distances[i, j] = np.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)

        return distances


    def run(self):

        best_distance = float('inf')

        best_path = None


        for iteration in range(self.n_iterations):

            all_paths = []
```

```
            all_distances = []

            for ant in range(self.n_ants):
                path = self.construct_solution()
                path_distance = self.calculate_path_distance(path)
                all_paths.append(path)
                all_distances.append(path_distance)

                if path_distance < best_distance:
                    best_distance = path_distance
                    best_path = path

            self.update_pheromones(all_paths, all_distances)
            print(f"Iteration {iteration+1}: Best Distance = {best_distance}")

        return best_path, best_distance

    def construct_solution(self):
        path = [random.randint(0, self.num_cities - 1)]
        while len(path) < self.num_cities:
            current_city = path[-1]
            next_city = self.choose_next_city(current_city, path)
            path.append(next_city)
        return path

    def choose_next_city(self, current_city, path):
        probabilities = []
        for next_city in range(self.num_cities):
            if next_city not in path:
                pheromone = self.pheromone_matrix[current_city, next_city] ** self.alpha
                heuristic = (1.0 / self.distances[current_city, next_city]) ** self.beta
```

```
            probabilities.append((next_city, pheromone * heuristic))
        else:
            probabilities.append((next_city, 0))

    total = sum(prob for _, prob in probabilities)
    probabilities = [(city, prob / total if total > 0 else 0) for city, prob in probabilities]

    selected_city = random.choices(
        [city for city, _ in probabilities],
        weights=[prob for _, prob in probabilities],
        k=1
    )[0]
    return selected_city

def calculate_path_distance(self, path):
    distance = 0
    for i in range(len(path) - 1):
        distance += self.distances[path[i], path[i + 1]]
    distance += self.distances[path[-1], path[0]]  # return to start
    return distance

def update_pheromones(self, paths, distances):
    self.pheromone_matrix *= (1 - self.rho)  # evaporate pheromones

    # deposit pheromones based on path quality
    for path, distance in zip(paths, distances):
        for i in range(len(path) - 1):
            self.pheromone_matrix[path[i], path[i + 1]] += self.pheromone_deposit / distance
        self.pheromone_matrix[path[-1], path[0]] += self.pheromone_deposit / distance  # return to
start
```

```
# Example usage with random cities

cities = [(random.randint(0, 100), random.randint(0, 100)) for _ in range(10)]

print(cities)

aco = ACO(n_ants=10, n_iterations=100, alpha=1, beta=2, rho=0.5, pheromone_deposit=10, cities=cities)

best_path, best_distance = aco.run()


print("\nBest Path:", best_path)

print("Best Distance:", best_distance)
```

```
Iteration 66: Best Distance = 326.99882689334635
Iteration 67: Best Distance = 326.99882689334635
Iteration 68: Best Distance = 326.99882689334635
Iteration 69: Best Distance = 326.99882689334635
Iteration 70: Best Distance = 326.99882689334635
Iteration 71: Best Distance = 326.99882689334635
Iteration 72: Best Distance = 326.99882689334635
Iteration 73: Best Distance = 326.99882689334635
Iteration 74: Best Distance = 326.99882689334635
Iteration 75: Best Distance = 326.99882689334635
Iteration 76: Best Distance = 326.99882689334635
Iteration 77: Best Distance = 326.99882689334635
Iteration 78: Best Distance = 326.99882689334635
Iteration 79: Best Distance = 326.99882689334635
Iteration 80: Best Distance = 326.99882689334635
Iteration 81: Best Distance = 326.99882689334635
Iteration 82: Best Distance = 326.99882689334635
Iteration 83: Best Distance = 326.99882689334635
Iteration 84: Best Distance = 326.99882689334635
Iteration 85: Best Distance = 326.99882689334635
Iteration 86: Best Distance = 326.99882689334635
Iteration 87: Best Distance = 326.99882689334635
Iteration 88: Best Distance = 326.99882689334635
Iteration 89: Best Distance = 326.99882689334635
Iteration 90: Best Distance = 326.99882689334635
Iteration 91: Best Distance = 326.99882689334635
Iteration 92: Best Distance = 326.99882689334635
Iteration 93: Best Distance = 326.99882689334635
Iteration 94: Best Distance = 326.99882689334635
Iteration 95: Best Distance = 326.99882689334635
Iteration 96: Best Distance = 326.99882689334635
Iteration 97: Best Distance = 326.99882689334635
Iteration 98: Best Distance = 326.99882689334635
Iteration 99: Best Distance = 326.99882689334635
Iteration 100: Best Distance = 326.99882689334635

Best Path: [7, 9, 5, 4, 1, 3, 8, 0, 6, 2]
Best Distance: 326.99882689334635
```