07-11-24

Ant Colony Algorithm

Algorithm:-

#Travelling Sales Man

```
import numpy as np
import random


# Function to calculate the total distance of a given path
def calculate_total_distance(distance_matrix, path):

    total_distance = 0

    for i in range(len(path) - 1):
        total_distance += distance_matrix[path[i]][path[i + 1]]
    total_distance += distance_matrix[path[-1]][path[0]] # Returning to the origin city
    return total_distance


# Function to perform the Ant Colony Optimization

def ant_colony_optimization(distance_matrix, num_ants, num_iterations, alpha, beta, rho,
pheromone_initial):

    num_cities = len(distance_matrix)



    # Initialize pheromone matrix with the initial pheromone value
    pheromone = np.ones((num_cities, num_cities)) * pheromone_initial


    # Initialize the best solution
    best_solution = None best_distance
    = float('inf')


    # Main ACO loop

    for iteration in range(num_iterations):
```

```
        # Ants' paths and their corresponding distances
        paths = []

        distances = []



        # Generate solutions for each ant
        for ant in range(num_ants):
            path = generate_path(distance_matrix, pheromone, alpha, beta)
            total_distance = calculate_total_distance(distance_matrix, path)
            paths.append(path)

            distances.append(total_distance)



            # Update the best solution if a new better one is found if
            total_distance < best_distance:

                best_solution = path best_distance
                = total_distance


        # Update pheromones

        pheromone = update_pheromones(pheromone, paths, distances, rho, best_solution,
best_distance)


    return best_solution, best_distance


# Function to generate a solution (path) for an ant

    def generate_path(distance_matrix, pheromone, alpha, beta):
        num_cities = len(distance_matrix)
    path = [random.randint(0, num_cities - 1)] # Start at a random city
    visited = set(path)


        while len(path) < num_cities:
            current_city = path[-1]
            probabilities = []
```

```
        # Calculate the probabilities for all unvisited cities for
        next_city in range(num_cities):

            if next_city not in visited:

                pheromone_strength = pheromone[current_city][next_city] ** alpha
                distance_heuristic = (1.0 / distance_matrix[current_city][next_city]) ** beta
                probabilities.append(pheromone_strength * distance_heuristic)

            else:

                probabilities.append(0)


        # Normalize the probabilities
        total_prob = sum(probabilities)

        probabilities = [p / total_prob for p in probabilities]



        # Choose the next city based on the calculated probabilities
        next_city = np.random.choice(range(num_cities), p=probabilities)
        path.append(next_city)
        visited.add(next_city) return

    path


# Function to update the pheromone matrix after each iteration

    def update_pheromones(pheromone, paths, distances, rho, best_solution, best_distance):
        num_cities = len(pheromone)


    # Apply pheromone evaporation


    pheromone *= (1 - rho)


    # Deposit pheromones based on the paths and their distances for
    path, dist in zip(paths, distances):
            for i in range(len(path) - 1):
                pheromone[path[i]][path[i + 1]] += 1.0 / dist
```

```
        pheromone[path[-1]][path[0]] += 1.0 / dist # Returning to the origin city



    # Deposit more pheromone on the best path found so far for
    i in range(len(best_solution) - 1):
        pheromone[best_solution[i]][best_solution[i + 1]] += 1.0 / best_distance
    pheromone[best_solution[-1]][best_solution[0]] += 1.0 / best_distance # Returning to the

origin city



    return pheromone



# Input the distance matrix and parameters from the user



num_cities = int(input("Enter the number of cities: "))
distance_matrix = []
print("Enter the distance matrix (row by row):") for
i in range(num_cities):
    row = list(map(int, input(f"Row {i+1}: ").split()))
    distance_matrix.append(row)


num_ants = int(input("Enter the number of ants: "))
num_iterations = int(input("Enter the number of iterations: "))

alpha = float(input("Enter the value of alpha (importance of pheromone): "))

beta = float(input("Enter the value of beta (importance of heuristic information): ")) rho
= float(input("Enter the evaporation rate (rho): "))

pheromone_initial = float(input("Enter the initial pheromone value: "))



# Run the ACO algorithm

best_solution, best_distance = ant_colony_optimization(
    distance_matrix, num_ants, num_iterations, alpha, beta, rho, pheromone_initial
```

```
)

# Display the results

print("Best Solution (Path):", list(map(int, best_solution))) # Fix for clean output
print("Best Distance:", best_distance
```

OUTPUT:

```
Enter the number of cities: 4
Enter the distance matrix (row by row):
Row 1: 0 12 23 45
Row 2: 12 0 43 23
Row 3: 23 67 87 9
Row 4: 14 32 26 17
Enter the number of ants: 10
Enter the number of iterations: 100
Enter the value of alpha (importance of pheromone): 2
Enter the value of beta (importance of heuristic information): 1
Enter the evaporation rate (rho): 0.3
Enter the initial pheromone value: 0.5
Best Solution (Path): [0, 2, 3, 1]
Best Distance: 76
```