BIS LAB 10/10/2025

Sharada Koundinya(1BM23CS310)

CODE:

```
import random

import numpy as np


# Define the Problem - Create a set of customers with their coordinates and demands

class VRP:

    def __init__(self, depot, customers, capacities):

        self.depot = depot

        self.customers = customers

        self.capacities = capacities

        self.num_customers = len(customers)

        self.num_vehicles = len(capacities)

        self.distance_matrix = self.create_distance_matrix()


    def create_distance_matrix(self):

        dist_matrix = np.zeros((self.num_customers + 1, self.num_customers + 1))  # +1 for
depot

        for i in range(self.num_customers + 1):

            for j in range(i + 1, self.num_customers + 1):

                if i == 0:

                    dist = np.linalg.norm(np.array(self.depot) - np.array(self.customers[j-1]))

                elif j == 0:

                    dist = np.linalg.norm(np.array(self.depot) - np.array(self.customers[i-1]))

                else:

                    dist = np.linalg.norm(np.array(self.customers[i-1]) - np.array(self.customers[j-1]))

                dist_matrix[i][j] = dist_matrix[j][i] = dist
```

```
        return dist_matrix


# Ant Colony Optimization (ACO) for VRP
class AntColony:
    def __init__(self, vrp, num_ants, alpha=1, beta=5, rho=0.5, q0=0.9, iterations=100):
        self.vrp = vrp
        self.num_ants = num_ants
        self.alpha = alpha
        self.beta = beta
        self.rho = rho
        self.q0 = q0
        self.iterations = iterations
        self.pheromone = np.ones((vrp.num_customers + 1, vrp.num_customers + 1))  # Initial
pheromone matrix
        self.best_solution = None
        self.best_solution_length = float('inf')

    # Step 1: Construct Solutions
    def construct_solution(self, ant_idx):
        unvisited = set(range(1, self.vrp.num_customers + 1))  # Set of unvisited customers
        routes = {vehicle: [] for vehicle in range(self.vrp.num_vehicles)}
        demands = {vehicle: 0 for vehicle in range(self.vrp.num_vehicles)}  # Capacity tracker
        current_city = 0  # Start from depot (index 0)

        while unvisited:
            vehicle = random.choice(range(self.vrp.num_vehicles))  # Random vehicle selection
            if demands[vehicle] < self.vrp.capacities[vehicle]:
                next_city = self.select_next_city(current_city, unvisited)
```

```
            routes[vehicle].append(next_city)
            unvisited.remove(next_city)
            demands[vehicle] += 1  # We assume 1 unit demand per customer for simplicity
            current_city = next_city
        else:
            continue  # Skip vehicle if it's at full capacity

    # Return to the depot for each vehicle
    for vehicle in range(self.vrp.num_vehicles):
        routes[vehicle].append(0)  # Returning to depot

    return routes

# Step 2: Calculate Transition Probabilities
def calculate_probabilities(self, current_city, unvisited):
    probabilities = []
    for city in unvisited:
        pheromone = self.pheromone[current_city][city] ** self.alpha
        distance = self.vrp.distance_matrix[current_city][city]
        heuristic = (1 / distance) ** self.beta
        probabilities.append(pheromone * heuristic)

    total_prob = sum(probabilities)
    if total_prob == 0:  # If no pheromone, distribute probabilities evenly
        return [1 / len(unvisited)] * len(unvisited)

    probabilities = [p / total_prob for p in probabilities]
    return probabilities
```

```
# Step 3: Select the Next City Based on Probabilities
def select_next_city(self, current_city, unvisited):
    probabilities = self.calculate_probabilities(current_city, unvisited)
    return random.choices(list(unvisited), probabilities)[0]

# Step 4: Update Pheromones
def update_pheromones(self, solutions, lengths):
    # Evaporate pheromones
    self.pheromone *= (1 - self.rho)

    # Update pheromones based on solutions found by ants
    for idx, solution in enumerate(solutions):
        length = lengths[idx]
        for route in solution.values():
            for i in range(len(route) - 1):
                self.pheromone[route[i]][route[i + 1]] += 1 / length

# Step 5: Run the ACO Algorithm
def run(self):
    for iteration in range(self.iterations):
        all_solutions = []
        all_lengths = []

        # Construct solutions for each ant
        for ant_idx in range(self.num_ants):
            solution = self.construct_solution(ant_idx)
            length = self.calculate_solution_length(solution)
```

```
                all_solutions.append(solution)
                all_lengths.append(length)

                # Update the best solution found
                if length < self.best_solution_length:
                    self.best_solution = solution
                    self.best_solution_length = length

            # Update pheromones after all ants have completed their tours
            self.update_pheromones(all_solutions, all_lengths)

            print(f"Iteration {iteration + 1}/{self.iterations}: Best Length =
{self.best_solution_length}")

        return self.best_solution, self.best_solution_length

    # Step 6: Calculate the Length of a Solution (Total Distance)
    def calculate_solution_length(self, solution):
        length = 0
        for vehicle in solution.values():
            for i in range(len(vehicle) - 1):
                length += self.vrp.distance_matrix[vehicle[i]][vehicle[i + 1]]
        return length

# Main function to run the ACO
if __name__ == "__main__":
    # Define the depot (0, 0) and customers with (x, y) coordinates and demand
    depot = (0, 0)
```

```python
customers = [(2, 4), (3, 2), (6, 5), (8, 3), (7, 8), (5, 7)]
capacities = [3, 3]  # Two vehicles with a capacity of 3 each

# Initialize VRP and Ant Colony
vrp = VRP(depot, customers, capacities)
aco = AntColony(vrp, num_ants=10, alpha=1, beta=2, rho=0.5, q0=0.9, iterations=50)

# Run ACO to find the best solution
best_solution, best_solution_length = aco.run()

# Print the results
print("Best Solution (Routes):")
for vehicle, route in best_solution.items():
    print(f"Vehicle {vehicle + 1}: {route}")

print(f"\nBest Solution Length (Total Distance): {best_solution_length}")
```

OUTPUT:

```
Iteration 1/50: Best Length = 30.704096057970965
Iteration 2/50: Best Length = 25.85324233796014
Iteration 3/50: Best Length = 25.85324233796014
Iteration 4/50: Best Length = 25.85324233796014
Iteration 5/50: Best Length = 25.85324233796014
Iteration 6/50: Best Length = 25.85324233796014
Iteration 7/50: Best Length = 25.80789435080295
Iteration 8/50: Best Length = 25.80789435080295
Iteration 9/50: Best Length = 25.80789435080295
Iteration 10/50: Best Length = 25.80789435080295
Iteration 11/50: Best Length = 25.738177938973646
Iteration 12/50: Best Length = 25.738177938973646
Iteration 13/50: Best Length = 25.738177938973646
```

Iteration 14/50: Best Length = 22.65045276546171

Iteration 15/50: Best Length = 22.65045276546171

Iteration 16/50: Best Length = 22.65045276546171

Iteration 17/50: Best Length = 22.65045276546171

Iteration 18/50: Best Length = 22.65045276546171

Iteration 19/50: Best Length = 22.65045276546171

Iteration 20/50: Best Length = 22.65045276546171

Iteration 21/50: Best Length = 22.65045276546171

Iteration 22/50: Best Length = 22.65045276546171

Iteration 23/50: Best Length = 22.483842533421615

Iteration 24/50: Best Length = 22.483842533421615

Iteration 25/50: Best Length = 22.483842533421615

Iteration 26/50: Best Length = 22.483842533421615

Iteration 27/50: Best Length = 22.483842533421615

Iteration 28/50: Best Length = 22.483842533421615

Iteration 29/50: Best Length = 22.483842533421615

Iteration 30/50: Best Length = 22.483842533421615

Iteration 31/50: Best Length = 22.483842533421615

Iteration 32/50: Best Length = 22.483842533421615

Iteration 33/50: Best Length = 22.483842533421615

Iteration 34/50: Best Length = 22.483842533421615

Iteration 35/50: Best Length = 22.483842533421615

Iteration 36/50: Best Length = 22.483842533421615

Iteration 37/50: Best Length = 22.483842533421615

Iteration 38/50: Best Length = 22.483842533421615

Iteration 39/50: Best Length = 22.483842533421615

Iteration 40/50: Best Length = 22.483842533421615

Iteration 41/50: Best Length = 22.483842533421615

Iteration 42/50: Best Length = 22.483842533421615

Iteration 43/50: Best Length = 22.483842533421615

Iteration 44/50: Best Length = 22.483842533421615

Iteration 45/50: Best Length = 22.483842533421615

Iteration 46/50: Best Length = 22.483842533421615

Iteration 47/50: Best Length = 22.483842533421615

Iteration 48/50: Best Length = 22.483842533421615

Iteration 49/50: Best Length = 22.483842533421615

Iteration 50/50: Best Length = 22.483842533421615

Best Solution (Routes):

Vehicle 1: [5, 6, 1, 0]

Vehicle 2: [3, 4, 2, 0]

Best Solution Length (Total Distance): 22.483842533421615