ANT COLONY BY TRAVELLING SALESMAN

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:

