**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:**

