LAB-02 **(16-10-24)**

Ant Colony Optimization for the Traveling Salesman Problem:
The foraging behavior of ants has inspired the development of optimization algorithms that can solve complex problems such as the Traveling Salesman Problem (TSP). Ant Colony Optimization (ACO) simulates the way ants find the shortest path between food sources and their nest. Implement the ACO algorithm using Python to solve the TSP, where the objective is to find the shortest possible route that visits a list of cities and returns to the origin city.

Implementation Steps:
1.Define the Problem: Create a set of cities with their coordinates.
2. Initialize Parameters: Set the number of ants, the importance of pheromone (alpha), the importance of heuristic information (beta), the evaporation rate (rho), and the initial pheromone value.
3. Construct Solutions: Each ant constructs a solution by probabilistically choosing the next city based on pheromone trails and heuristic information.
4.Update Pheromones: After all ants have constructed their solutions, update the pheromone trails based on the quality of the solutions found.
5. Iterate: Repeat the construction and updating process for a fixed number of iterations or until convergence criteria are met.
6.Output the Best Solution: Keep track of and output the best solution found during the iterations.

ALGORITHM/LOGIC -

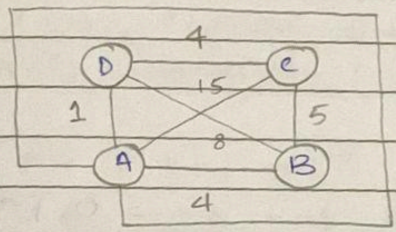
ANT COLONY ORGANIZATION

1) $\Delta Z_{i,j}^{k} = \begin{cases} 1/L_k & k^{th} \text{ ant travels on the edge } x, y \\ 0 & \text{otherwise} \end{cases}$

L_k = length of the path

2) $Z_{i,j}^{k} = \sum_{k=1}^{m} \Delta Z_{ij}^{k}$ {without vaporization}

3) $Z_{i,j}^{k} = (1-\rho) Z_{i,j}^{t} + \sum_{k=1}^{m} \Delta Z_{i,j}^{m}$

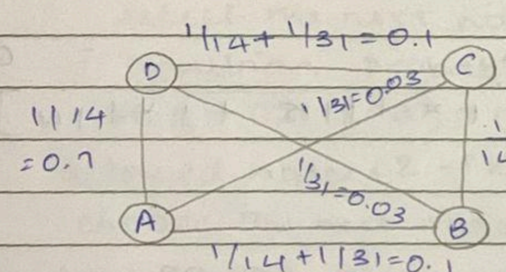


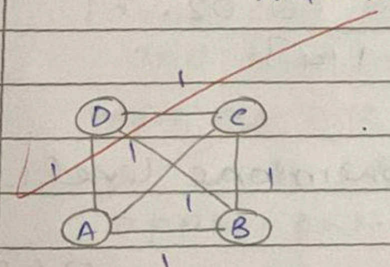
Cost Matrix

	A	B	C	D
A	0	4	15	1
B	4	0	5	8
C	15	5	0	4
D	1	8	4	0

$S_1 = 14 = 1/14$ 1pencil-thick

$S_2 = 31 = 1/31$



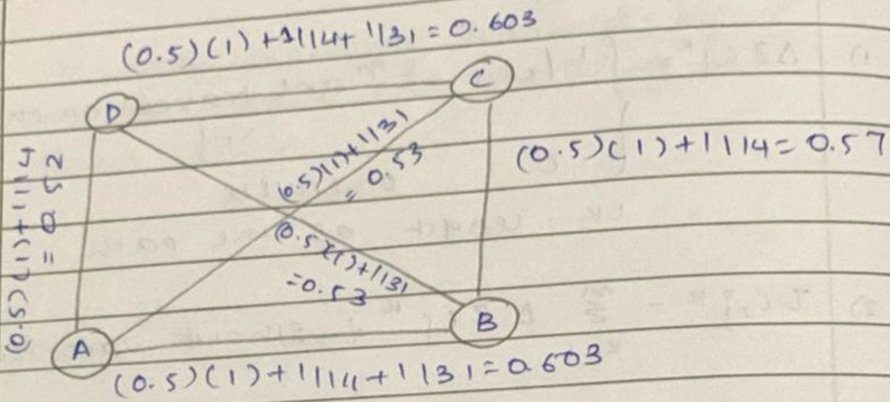


$\rho = 0.5$

$Z_{ij}^{k} = (1-\rho) Z_{ij} + \sum_{k=1}^{m} \Delta Z_{ij}^{k}$

$Z_{ij} \Rightarrow 1$ at first

(phermone level)



$$P_{ij} = \frac{(\tau_{ij})^{\alpha} (\eta_{ij})^{m}}{\Sigma\left[(\tau_{ij})^{\alpha} (\eta_{ij})^{B}\right]} \qquad \eta_{ij} = \frac{1}{d_{ij}}$$

A to D

$$\frac{1 * 1/1}{(1 * 1/1 + 1 * 1/15 + 1 * 1/14)} = 0.195$$

A to B

$$\frac{1 * 1/14}{\left[1 * 1/1 + 1 * 1/15 + 1 * 1/14\right]} = 0.18$$

A to C

$$\frac{1 * 1/15}{\left[1 * 1/1 + 1 * 1/15 + 1 * 1/14\right]} = 0.02$$

Making A → B pheromone levels.

$$A \text{ to } B = \frac{5 * 1/14}{1 * 1/1 + 1 * 1/15 + 5 * 1/14} = 0.53$$

A to C = $\frac{1 \times 1/15}{1 * 1/1 + 1 * 1/15 + 5 * 1/4}$ = 0.10

ALGORITHM.

Initialize parameters

N = no. of ants

T = max no. of iteration

a = influence of pheromone

B = influence of heuristic information

p = pheromone evaporation rate

Q = pheromone deposit constant

T-ij = initial pheromone level

n-ij = heuristic information

For ~~b~~ t = 1 to T

for each ant k = 1 to N

initialize ant's tour ~~for each~~

for each step of the ant's tour

select the next node (j) based on transition probability

P-ij = (2-ij ^ a) * (n-ij ^ B) / sum-over allowed-nodes (2-ik ^ a * n-ik ^ B)

choose the next node j using P-ij

favouring paths with more pheromone and better heuristic info

move the ant to selected node

for each edge (i, j)

apply pheromone evaporation:

2-ij = (1-p) * 2-ij

for each ant k:

calculate the tour length L-k

Deposit pheromone on the edges in ant's path:

$\Delta z\text{-}ij = Q/L - k$

$z\text{-}ij = z\text{-}ij$ for all edges (i,j) in ant k's tour optionally, keep track of the best solution so far check stopping criteria.

if stopping condition a met (e.g. maximum iterations or convergence), exit the loop

return the best solution found –

before updating the pheromone,

$$P1 = \frac{1 * 1/1}{1*1/1 + 1*1/15 + 1*1/4} = 0.75$$

$$P2 = \frac{1 * 1/4}{1*1/1 + 1*1/15 + 1*1/4} = 0.18$$

$$P3 = \frac{1 * 1/1}{1*1/1 + 1*1/15 + 5*1/4} = 0.05$$

After updating the pheromone,

$$P1 = \frac{1 * 1/1}{1*1/1 + 1*1/15 + 5*1/4} = 0.43$$

$$P2 = \frac{5*1/4}{1*1/1 + 1*1/15 + 5*1/4} = 0.53$$

$$P3 = \frac{1 * 1/15}{1*1/1 * 1*1/15 + 5*1/4} = 0.02.$$

INPUT-

```
import numpy as np
import random
def distance(city1, city2):
    return np.linalg.norm(np.array(city1) - np.array(city2))
cities = {
    0: (0, 0),
    1: (2, 4),
    2: (5, 2),
    3: (6, 6),
    4: (8, 3)
}
num_cities = len(cities)
distance_matrix = np.array([
    [distance(cities[i], cities[j]) for j in range(num_cities)] for i in range(num_cities)
])

# Initialize parameters
num_ants = 10
alpha = 1
beta = 2
rho = 0.5
pheromone_init = 0.1
num_iterations = 50
pheromone = np.full((num_cities, num_cities), pheromone_init)
heuristic = 1 / (distance_matrix + np.diag([np.inf] * num_cities))
def calculate_probabilities(ant_path, current_city):
    probabilities = []
    for next_city in range(num_cities):
        if next_city not in ant_path:
            tau = pheromone[current_city][next_city] ** alpha
            eta = heuristic[current_city][next_city] ** beta
            probabilities.append(tau * eta)
        else:
            probabilities.append(0)
    probabilities = np.array(probabilities)
    return probabilities / probabilities.sum()
def construct_solutions():
    all_paths = []
    all_distances = []
```

```python
    for ant in range(num_ants):
        ant_path = []
        current_city = random.randint(0, num_cities - 1)
        ant_path.append(current_city)
        while len(ant_path) < num_cities:
            probabilities = calculate_probabilities(ant_path, current_city)
            next_city = np.random.choice(range(num_cities), p=probabilities)
            ant_path.append(next_city)
            current_city = next_city
        ant_path.append(ant_path[0])
        total_distance = sum(
            distance_matrix[ant_path[i]][ant_path[i + 1]] for i in range(num_cities)
        )
        all_paths.append(ant_path)
        all_distances.append(total_distance)
    return all_paths, all_distances
# Update pheromones
def update_pheromones(all_paths, all_distances):
    global pheromone
    pheromone *= (1 - rho)
    for path, dist in zip(all_paths, all_distances):
        for i in range(num_cities):
            from_city = path[i]
            to_city = path[i + 1]
            pheromone[from_city][to_city] += 1 / dist
# Ant Colony Optimization
best_path = None
best_distance = float('inf')
for iteration in range(num_iterations):
    all_paths, all_distances = construct_solutions()
    update_pheromones(all_paths, all_distances)
    min_distance_idx = np.argmin(all_distances)
    if all_distances[min_distance_idx] < best_distance:
        best_distance = all_distances[min_distance_idx]
        best_path = all_paths[min_distance_idx]
    print(f"Iteration {iteration + 1}: Best Distance = {best_distance}")
print("Best path:", best_path)
print("Best distance:", best_distance)
```

OUTPUT-

```
Iteration 1: Best Distance = 21.09726565276603
Iteration 2: Best Distance = 21.09726565276603
Iteration 3: Best Distance = 21.09726565276603
Iteration 4: Best Distance = 21.09726565276603
Iteration 5: Best Distance = 21.09726565276603
Iteration 6: Best Distance = 21.09726565276603
Iteration 7: Best Distance = 21.09726565276603
Iteration 8: Best Distance = 21.09726565276603
Iteration 9: Best Distance = 21.09726565276603
Iteration 10: Best Distance = 21.09726565276603
Iteration 11: Best Distance = 21.09726565276603
Iteration 12: Best Distance = 21.09726565276603
Iteration 13: Best Distance = 21.09726565276603
Iteration 14: Best Distance = 21.09726565276603
Iteration 15: Best Distance = 21.09726565276603
Iteration 16: Best Distance = 21.09726565276603
Iteration 17: Best Distance = 21.09726565276603
Iteration 18: Best Distance = 21.09726565276603
Iteration 19: Best Distance = 21.09726565276603
Iteration 20: Best Distance = 21.09726565276603
Iteration 21: Best Distance = 21.09726565276603
Iteration 22: Best Distance = 21.09726565276603
Iteration 23: Best Distance = 21.09726565276603
Iteration 24: Best Distance = 21.09726565276603
Iteration 25: Best Distance = 21.09726565276603
Iteration 26: Best Distance = 21.09726565276603
Iteration 27: Best Distance = 21.09726565276603
Iteration 28: Best Distance = 21.09726565276603
Iteration 29: Best Distance = 21.09726565276603
Iteration 30: Best Distance = 21.09726565276603
Iteration 31: Best Distance = 21.09726565276603
Iteration 32: Best Distance = 21.09726565276603
Iteration 33: Best Distance = 21.09726565276603
Iteration 34: Best Distance = 21.09726565276603
Iteration 35: Best Distance = 21.09726565276603
Iteration 36: Best Distance = 21.09726565276603
Iteration 37: Best Distance = 21.09726565276603
Iteration 38: Best Distance = 21.09726565276603
Iteration 39: Best Distance = 21.09726565276603
Iteration 40: Best Distance = 21.09726565276603
Iteration 41: Best Distance = 21.09726565276603
Iteration 42: Best Distance = 21.09726565276603
Iteration 43: Best Distance = 21.09726565276603
Iteration 44: Best Distance = 21.09726565276603
Iteration 45: Best Distance = 21.09726565276603
```


Iteration 41: Best Distance = 21.09726565276603
Iteration 42: Best Distance = 21.09726565276603
Iteration 43: Best Distance = 21.09726565276603
Iteration 44: Best Distance = 21.09726565276603
Iteration 45: Best Distance = 21.09726565276603


```
Iteration 46: Best Distance = 21.09726565276603
Iteration 47: Best Distance = 21.09726565276603
Iteration 48: Best Distance = 21.09726565276603
Iteration 49: Best Distance = 21.09726565276603
Iteration 50: Best Distance = 21.09726565276603
Best path: [4, 3, 1, 0, 2, 4]
Best distance: 21.09726565276603
```