BIS LAB

LAB 4

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.

PYTHON CODE:

ACO for the Traveling Salesman Problem (TSP) — where the goal is to find the **shortest route visiting all cities and returning to the starting city**.

```
import numpy as np
import random
import math

# 1. Define the Problem – Cities and Coordinates
cities = {
    'A': (0, 0),
    'B': (1, 5),
    'C': (5, 2),
    'D': (6, 6),
    'E': (8, 3)
}

city_names = list(cities.keys())
num_cities = len(city_names)

# Distance Matrix
dist_matrix = np.zeros((num_cities, num_cities))
for i in range(num_cities):
    for j in range(num_cities):
        xi, yi = cities[city_names[i]]
        xj, yj = cities[city_names[j]]
        dist_matrix[i][j] = math.sqrt((xi - xj)**2 + (yi - yj)**2)

# 2. Initialize Parameters
num_ants = 10
num_iterations = 50
alpha = 1.0      # Pheromone importance
beta = 5.0       # Heuristic importance (visibility)
rho = 0.5        # Pheromone evaporation rate
Q = 100          # Pheromone deposit factor
```

```
initial_pheromone = 1.0

# Initialize pheromone trails
pheromone = np.full((num_cities, num_cities), initial_pheromone)

def route_length(route):
    length = 0
    for i in range(len(route) - 1):
        length += dist_matrix[route[i]][route[i + 1]]
    length += dist_matrix[route[-1]][route[0]]  # Return to start
    return length
best_route = None
best_length = float('inf')

for iteration in range(num_iterations):
    all_routes = []
    all_lengths = []

    for ant in range(num_ants):
        visited = [random.randint(0, num_cities - 1)]

        while len(visited) < num_cities:
            current = visited[-1]
            probabilities = []
            for j in range(num_cities):
                if j not in visited:
                    tau = pheromone[current][j] ** alpha
                    eta = (1.0 / dist_matrix[current][j]) ** beta
                    probabilities.append(tau * eta)
                else:
                    probabilities.append(0)
            probabilities = np.array(probabilities)
            probabilities /= probabilities.sum()

            next_city = np.random.choice(range(num_cities),
p=probabilities)
            visited.append(next_city)

        length = route_length(visited)
        all_routes.append(visited)
        all_lengths.append(length)
        if length < best_length:
            best_length = length
            best_route = visited

    # 4. Update Pheromones
    pheromone *= (1 - rho)  # Evaporation
    for i, route in enumerate(all_routes):
```

```
        for j in range(num_cities - 1):
            a, b = route[j], route[j + 1]
            pheromone[a][b] += Q / all_lengths[i]
            pheromone[b][a] = pheromone[a][b]
        # Add pheromone for return to start
        a, b = route[-1], route[0]
        pheromone[a][b] += Q / all_lengths[i]
        pheromone[b][a] = pheromone[a][b]

    print(f"Iteration {iteration + 1} | Best Length:
{best_length:.4f}")

# 6. Output the Best Solution
best_city_names = [city_names[i] for i in best_route]
print("\n🐜 Best Route Found:")
print(" → ".join(best_city_names + [best_city_names[0]]))
print(f"Total Distance = {best_length:.4f}")
```

```
Iteration 1 | Best Length: 22.3510
Iteration 2 | Best Length: 22.3510
Iteration 3 | Best Length: 22.3510
Iteration 4 | Best Length: 22.3510
Iteration 5 | Best Length: 22.3510
Iteration 6 | Best Length: 22.3510
Iteration 7 | Best Length: 22.3510
Iteration 8 | Best Length: 22.3510
Iteration 9 | Best Length: 22.3510
Iteration 10 | Best Length: 22.3510
Iteration 11 | Best Length: 22.3510
Iteration 12 | Best Length: 22.3510
Iteration 13 | Best Length: 22.3510
Iteration 14 | Best Length: 22.3510
Iteration 15 | Best Length: 22.3510
Iteration 16 | Best Length: 22.3510
Iteration 17 | Best Length: 22.3510
Iteration 18 | Best Length: 22.3510
Iteration 19 | Best Length: 22.3510
Iteration 20 | Best Length: 22.3510
Iteration 21 | Best Length: 22.3510
Iteration 22 | Best Length: 22.3510
Iteration 23 | Best Length: 22.3510
Iteration 24 | Best Length: 22.3510
Iteration 25 | Best Length: 22.3510
Iteration 26 | Best Length: 22.3510
Iteration 27 | Best Length: 22.3510
Iteration 28 | Best Length: 22.3510
Iteration 29 | Best Length: 22.3510
Iteration 30 | Best Length: 22.3510
Iteration 31 | Best Length: 22.3510
Iteration 32 | Best Length: 22.3510
Iteration 33 | Best Length: 22.3510
Iteration 34 | Best Length: 22.3510
Iteration 35 | Best Length: 22.3510
Iteration 36 | Best Length: 22.3510
Iteration 37 | Best Length: 22.3510
Iteration 38 | Best Length: 22.3510
Iteration 39 | Best Length: 22.3510
Iteration 40 | Best Length: 22.3510
Iteration 41 | Best Length: 22.3510
Iteration 42 | Best Length: 22.3510
Iteration 43 | Best Length: 22.3510
Iteration 44 | Best Length: 22.3510
Iteration 45 | Best Length: 22.3510
Iteration 46 | Best Length: 22.3510
Iteration 47 | Best Length: 22.3510
Iteration 48 | Best Length: 22.3510
Iteration 49 | Best Length: 22.3510
Iteration 50 | Best Length: 22.3510

🐜 Best Route Found:
D → B → A → C → E → D
Total Distance = 22.3510
```