ASSIGNMENT 10

PROBLEM STATEMENT

Implement Ant colony optimization by solving the Traveling salesman problem using python Problem statement- A salesman needs to visit a set of cities exactly once and return to the original city. The task is to find the shortest possible route that the salesman can take to visit all the cities and return to the starting city.

OBJECTIVE

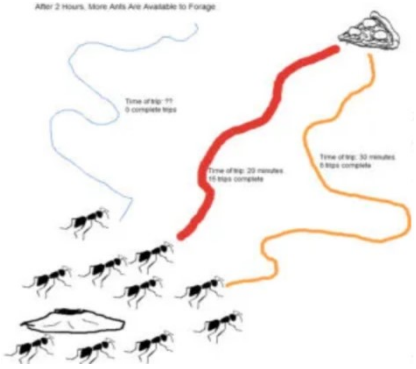
1. To learn and understand the Ant colony optimization algorithm.
2. Implement Ant colony optimization by solving the Traveling salesman problem.

THEORY

Ant colony optimization (ACO)

- Ant colony optimization (ACO) is a method inspired by the behavior of ants when they search for food.
- It is a metaheuristic optimization algorithm inspired by the behavior of ants. It is commonly used to solve combinatorial optimization problems like the Traveling Salesman Problem (TSP), the job shop scheduling problem, the vehicle routing problem, and the Knapsack problem.

- When ants look for food, they move randomly, leaving a pheromone trail behind them. Other ants follow these pheromone trails to find the food. The more ants that follow a trail, the stronger the pheromone trail becomes.



- In this example (figure), first, some Ant goes through the gray line, and it doesn't find the food. Since only some Ant's went through this way, the "pheromone" left by the ant will be minimal. These ants communicate with other ants not to follow this path.
- Second another set of some Ants goes through the orange line, and it finds food. These ants communicates with other ants that the food is found. So the other ant knows how to reach food following the "pheromone" train (orange path).
- Once the location of food is found, the other set of ants diverge to find the shortest path to the food from their ant colony. If the shortest path is found, then the shortest path is communicated to other ants. Now more ant's travel through the shortest path (red line) thus creating very strong "pheromone" trail. So finally ant's use the (red line) to go from their colony to food source.
- Similarly, in ACO, we have a group of artificial ants that are trying to find the best solution to a problem.
- For example, if we want to find the shortest path between two points, the artificial ants move randomly from one point to another, leaving a pheromone trail behind them.
- The pheromone trail indicates the quality of the path that the ant took. The ants prefer to follow the pheromone trail that is stronger, which corresponds to a better path.

Steps involved in Ant Colony Optimization:

1. Initialization: We start by placing the ants on the starting point.
2. Movement: Each ant selects a point to move to based on a probabilistic function that takes into account the pheromone level and the heuristic information. The heuristic information can be thought of as a measure of how good a particular point is.
3. Updating pheromone trails: The pheromone trail on each edge is updated based on the quality of the solution found by the ant that traversed that edge.
4. Termination: We stop the algorithm after a certain number of iterations or when a satisfactory solution is found.

By repeating these steps, the ants will gradually converge on the best solution to the problem. ACO can be used to solve a wide range of optimization problems, such as the traveling salesman problem, where the goal is to find the shortest path that visits a set of cities.

Pros & Cons of Ant colony optimization:

Advantages:

- Ant colony optimization is easy to implement and does not require complex mathematical knowledge.
- It is a very flexible algorithm and can be used in various problem domains.
- It can handle multiple objectives and constraints.
- It is a metaheuristic algorithm that can quickly find high-quality solutions in a large solution space.

- It has the ability to find near-optimal solutions, even in cases where the search space is very large or poorly understood.

Disadvantages:

- The algorithm may converge to a suboptimal solution if the parameter settings are not carefully selected.
- The algorithm may become computationally expensive if there are a large number of ants and/or iterations required to find a solution.
- The quality of the solution may be dependent on the pheromone initialization, which can be difficult to optimize.
- The algorithm may require a large number of iterations to converge to a solution, which may be a disadvantage if fast convergence is required.

Traveling Salesman Problem

- The Traveling Salesman Problem (TSP) is a famous mathematical problem in computer science and operations research.
- In this problem, a salesman needs to visit a set of cities exactly once and return to the original city.
- The task is to find the shortest possible route that the salesman can take to visit all the cities and return to the starting city.
- The TSP is an optimization problem and can be applied to various real-world scenarios.
- For example, a delivery person may want to visit a set of cities to deliver packages and return to the starting point while minimizing the distance traveled. Another example is a circuit board drilling machine that needs to drill holes at various locations on the board.
- Finding the exact solution for TSP becomes computationally expensive as the number of cities increases.
- The number of possible paths between cities grows exponentially with the number of cities. Hence, it is necessary to use heuristic algorithms like ant colony optimization or genetic algorithms to find an approximate solution to the TSP.

CONCLUSION

We have implemented Ant colony optimization by solving the Traveling salesman problem using Python.

ORAL QUESTION

1. What is the Traveling Salesman Problem (TSP)?
2. How does Ant Colony Optimization (ACO) work, and what are its key components?
3. What role do pheromone trails play in ACO?
4. What data structures would you use in Python to represent the graph of cities and distances between them?

.

Code:

```
import numpy as np
import random

# Define the distance matrix (distances between cities)
# Replace this with your distance matrix or generate one based on
your problem
# Example distance matrix (replace this with your actual data)
distance_matrix = np.array([
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
])

# Parameters for Ant Colony Optimization
num_ants = 10
num_iterations = 50
evaporation_rate = 0.5
pheromone_constant = 1.0
heuristic_constant = 1.0

# Initialize pheromone matrix and visibility matrix
num_cities = len(distance_matrix)
pheromone = np.ones((num_cities, num_cities))  # Pheromone matrix
visibility = 1 / distance_matrix  # Visibility matrix (inverse of
distance)

# ACO algorithm
for iteration in range(num_iterations):
    ant_routes = []
    for ant in range(num_ants):
        current_city = random.randint(0, num_cities - 1)
        visited_cities = [current_city]
        route = [current_city]

        while len(visited_cities) < num_cities:
            probabilities = []
            for city in range(num_cities):
                if city not in visited_cities:
                    pheromone_value = pheromone[current_city][city]
                    visibility_value = visibility[current_city][city]
                    probability = (pheromone_value **
pheromone_constant) * (visibility_value ** heuristic_constant)
                    probabilities.append((city, probability))
```

```
            probabilities = sorted(probabilities, key=lambda x: x[1],
reverse=True)
            selected_city = probabilities[0][0]
            route.append(selected_city)
            visited_cities.append(selected_city)
            current_city = selected_city

        ant_routes.append(route)

    # Update pheromone levels
    delta_pheromone = np.zeros((num_cities, num_cities))

    for ant, route in enumerate(ant_routes):
        for i in range(len(route) - 1):
            city_a = route[i]
            city_b = route[i + 1]
            delta_pheromone[city_a][city_b] += 1 /
distance_matrix[city_a][city_b]
            delta_pheromone[city_b][city_a] += 1 /
distance_matrix[city_a][city_b]

    pheromone = (1 - evaporation_rate) * pheromone + delta_pheromone

# Find the best route
best_route_index =
np.argmax([sum(distance_matrix[cities[i]][cities[(i + 1) %
num_cities]] for i in range(num_cities)) for cities in ant_routes])
best_route = ant_routes[best_route_index]
shortest_distance = sum(distance_matrix[best_route[i]][best_route[(i
+ 1) % num_cities]] for i in range(num_cities))

print("Best route:", best_route)
print("Shortest distance:", shortest_distance)
```

Output:

```
PS D:\BE SEM VIII> python -u "d:\BE SEM VIII\CL_III_Code\TSP.py"
d:\BE SEM VIII\CL_III_Code\TSP.py:24: RuntimeWarning: divide by zero encountered in divide
  visibility = 1 / distance_matrix  # Visibility matrix (inverse of distance)
Best route: [0, 1, 3, 2]
Shortest distance: 80
PS D:\BE SEM VIII>
```