Practical No. 5

Aim: 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.

- **Outcome:** At end of this experiment, student will be able understand Swam Intelligence
- **Hardware Requirement:** Computer System, Linux (Ubuntu)
- **Software Requirement:** PyCharm IDE

Theory:

Swarm Intelligence (S.I.) was introduced by **Gerardo Beni and Jing Wang in the year 1989**. S.I. simply means using the knowledge of collective objects (people, insects, etc.) together and then reaching the optimized solution for a given problem. **"Swarm" means a group of objects (people, insects**, **etc.)**. In other words, let's say we give a problem statement to a single person and tell him or her to go through this problem and then give the solution, then this means that we will consider the solution of that particular person only, but the problem is that the solution given by that person may not be the best solution or maybe, that solution is not good for others. So to avoid that, what we do is we give that problem to a certain amount of people together (swarm) and ask them to reach the best solution possible for that problem, and then computing all the responses together to reach the best solution possible, so here we are using the knowledge of the group as a whole to reach to the best solution or optimized solution for that problem and that solution will be good for all of them individually too, so that is the idea behind swarm intelligence.

Example

Let's say we have a jar containing 500 marbles in that. The question is without touching the jar a person needs to predict how many marbles are in that jar. Suppose we take only one response from a person and it predicts that according to him the jar contains 400 marbles. So by this result, we can conclude that this estimation of that person is not very bad since the **difference (error) is of 100** only, but this might not be the best solution, we can optimize this even more. So now what we will do is instead of taking response from only one person we will be taking response from 10 people let's say. Let 'P' denote a person therefore the responses are as follows:

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
400	450	550	600	480	390	520	490	510	450

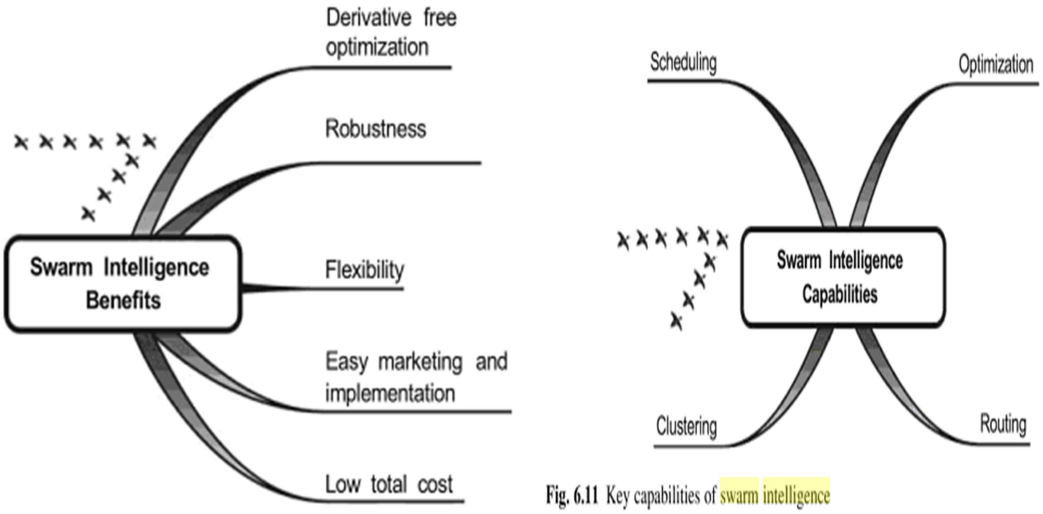
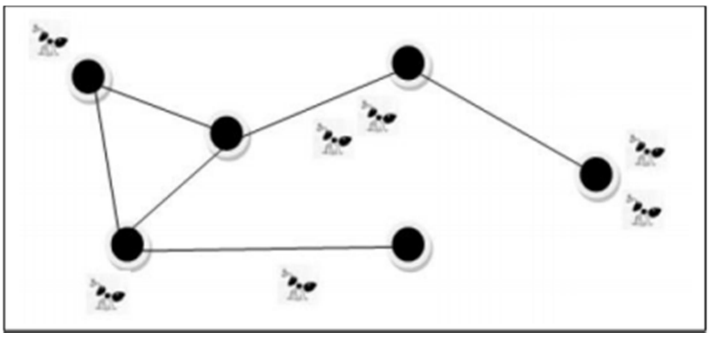


Fig. 6.11 Key capabilities of swarm intelligence

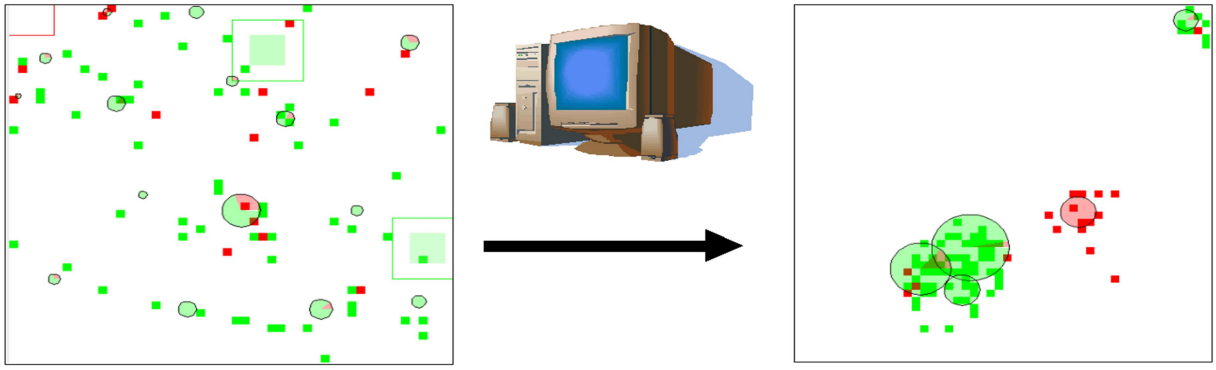
Swarm Intelligence Capabilities:

1. **Scheduling / Load Balancing:**

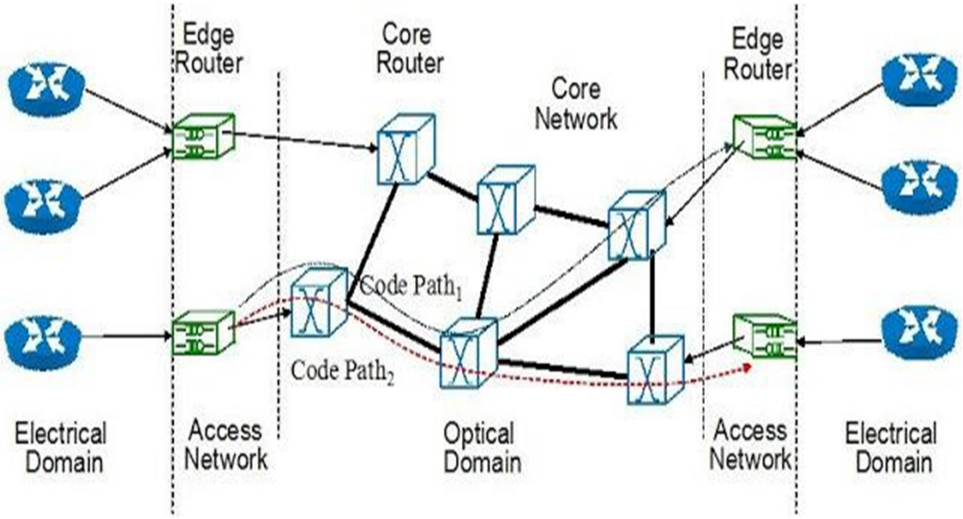
The emphasis is on he elative position of the job rather than its direct predecessor or its direct successor in the schedule and summation evaluation rule / global pheromone evaluation rule is followed.



2. **Clustering:** A cluster is a collection of agents which are similar and are dissimilar to the agents in other clusters.



Optimization: An optimization problem is the problem of finding the Best Solution / Minimal Cost Solution from all the feasible solutions.



3. **Routing:** This is based on the principle that backward ants utilize the useful information gathered by the forward ants on their trip from source to destination.

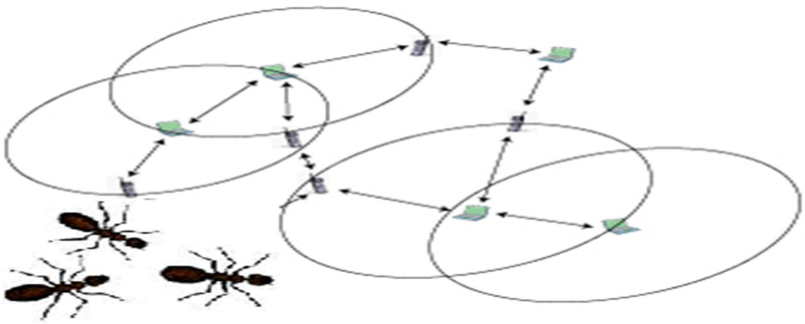


Fig: The AntHocNet routing algorithm for MANETs (mobile ad hoc networks)

Advantages:

1) **Flexible:** The colony respond to internal disturbances and external challenges.

2) **Robust:** Tasks are completed even if some agents fail.

3) **Scalable:** From a few agents to millions

4) **Decentralized:** There is no central control in the colony.

5) **Self-organized:** The solutions are emergent rather than pre-defined.

6) **Adaptation:** The swarm system can not only adjust to predetermined stimuli but also to new stimuli.

7) **Speed:** Changes in the network can be propagated very fast.

8) **Modularity:** Agents act independently of other network layers.

9) **Parallelism:** Agents' operations are inherently parallel.

Conclusion: In this way Ant colony optimization by solving the Traveling salesman problem using python is implemented successfully.

//Code

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.

```
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)
```

```
Best route: [3, 0, 1, 2]
Shortest distance: 95
```