CS312:Artificial Intelligence Laboratory

Lab 4 Report Group 24

Utkarsh Prakash - 180030042 Manjeet Kapil - 180010021

1. Introduction

Given a set of cities (coordinates) and distances between them, find the best (shortest) tour (visiting all the cities exactly one and returning to the origin city) in a given amount of time. In other words, in this assignment we solve the Travelling Salesman Problem.

2. Ant Colony Optimization

The ACO algorithm can be seen as a parallel, randomized search algorithm that converges towards good solutions by a process of learning in which each agent communicates some information about the goodness of solution components toa common pool. As more and more agents explore the good components, more agents use them to build solutions. In some sense, this behaviour is similar to the one in GA, except that there it is the solutions themselves that make up the populations. In ACO, the solutions are not coded, but simple agents repeatedly construct solutions, exploiting the information (or experience) of earlier attempts by the entire population.

Pseudo Code:

```
1. Initialize tij = Small values for all segments i-j in the
   problem
2. repeat
3.       Construct the tour for each of the m ants
4.       Remember the best tour when a better one is found
5.       Update the pheromone levels for each segment tij(t+n)
6. until some termination criteria
7. return the best tour
```

Let N represent the number of cities in the problem. The values of the parameters used are:

Alpha = 4, Beta = 4, rho = 0.1, Q= 0.1

Input	Cost of tour Found		
	Ants=N/10	**Ants = N/5**	**Ants = N**
euc_100	1532.85	1501.57	1530.71

noneuc_100	5852.47	5197.91	5197.91

3. Improvements

Ant system with elitist strategy and ranking (ASrank + ASelite)

The concept of ranking can be applied and extended to the ant system as follows: after all m ants have generated a tour, the ants are sorted by tour length (L1<= L2<= ::: <= Lm), and the contribution of an ant to the trail level update is weighted according to the rank of the ant. In addition to that, only the best ants are considered. Thus, the danger of over-emphasized pheromone trails caused by many ants using suboptimal paths can be avoided.

The update equations are given by:

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \Delta\tau_{ij} + \Delta\tau^*_{ij}$$

$$\text{where } \Delta\tau_{ij} = \sum_{\mu=1}^{\sigma-1} \Delta\tau^{\mu}_{ij}$$

$$\text{and } \Delta\tau^{\mu}_{ij} = \begin{cases} (\sigma-\mu)\frac{Q}{L_\mu} & \text{if the } \mu\text{-th best ant travels on edge } (i,j) \\ 0 & \text{otherwise} \end{cases}$$

$$\text{and } \Delta\tau^*_{ij} = \begin{cases} \sigma\frac{Q}{L^*} & \text{if edge } (i,j) \text{ is part of the best solution found} \\ 0 & \text{otherwise} \end{cases}$$

where

μ	ranking index
$\Delta\tau^{\mu}_{ij}$	increase of trail level on edge (i,j) caused by the μ-th best ant
L_μ	tour length of the μ-th best ant
$\Delta\tau^*_{ij}$	increase of trail level on edge (i,j) caused by the elitist ants
σ	number of elitist ants
L^*	tour length of best solution found

Empirically, we found sigma = 1 to perform the best.

Nearest Neighbor Algorithm (Greedy Approach)

We have also tried nearest neighbour approach on this problem-:

The process is as follows:

1. Select a starting city.
2. Find the nearest city to your current one and go there.
3. If there are still cities not yet visited, repeat step 2. Else, return to the starting city.

Once we find a best tour according to the algorithm above we use 2-opt optimisation strategy, we keep on changing locally two edges in the solution and try to find optimal path.

The solution found by the Greedy approach forms the initial pheromone solution for the Ant colony optimisation.

```
def tsp_2_opt(city_distances, tour, cost):
    """
    Approximate the optimal path of travelling salesman according to 2-opt algorithm
    Args:
        graph: 2d numpy array as graph
        route: list of nodes
    Returns:
        optimal path according to 2-opt algorithm
    """
    improved = True
    best_found_route = tour
    best_found_route_cost = cost
    while improved:
        improved = False

        for i in range(1, len(best_found_route) - 1):
            for k in range(i + 1, len(best_found_route) - 1):
                new_route_cost = (best_found_route_cost -
                                  city_distances[best_found_route[i-1]][best_found_route[i]] -
                                  city_distances[best_found_route[k]][best_found_route[k+1]] +
                                  city_distances[best_found_route[i-1]][best_found_route[k]] +
                                  city_distances[best_found_route[i]][best_found_route[k+1]])

                if new_route_cost < best_found_route_cost:
                    best_found_route_cost = new_route_cost
                    best_found_route = _swap_2opt(best_found_route, i, k)
                    improved = True
                    break
            if improved:
                break

    return best_found_route, best_found_route_cost
```

4. Conclusion

- Bigger alpha (pheromone factor) means the convergence speed increases but it may get stuck in local optima and lower alpha can lead to exceeded time-limit. So, we set alpha = 4 for Euclidean and alpha=15 for Non-Euclidean.

- Bigger beta (visibility/cost factor) means almost greedy algorithm and smaller leaves it random. So we set beta = 4 for Euclidean and beta=14 for Non-Euclidean.
- Bigger rho (evaporation coefficient) reduced convergence speed and smaller rho increased the global search ability so we set rho = 0.1 for Euclidean and rho=0.2 for Non-Euclidean
- Bigger Q(pheromone deposit factor) will allow it to fall into local optimum whereas smaller will give slow optimization speed. Q=0.1 for Euclidean and Non-Euclidean.
- Larger ants means a larger time period to complete 1 iteration but gives more optimum solutions at cost of time and smaller ants means less pheromone deposits and less directed next iteration. So we have chosen it to (no. of ants = no. of cities/5)