Ant Colony Optimization for TSP

Mirali Ahmadli
20170847
School of Computing, KAIST
miraliahmadli@kaist.ac.kr

Abstract -

Traveling Salesman Problem is one of the most critical NP-hard problem and it has a lot of applications in many fields. In this report, I will propose a non-deterministic model based on Ant Colony Optimization (ACO) algorithm that finds relatively optimal solution in polynomial time compared to traditional deterministic non-polynomial time algorithms. Proposed model uses Ant Colony Optimization algorithm we covered in classroom with additional optimizations to make model converge faster. First, design decisions and implementation details will be outlined. Then, optimization techniques used in the model will be described with benchmarkings to verify the idea.

1 Introduction

The goals of this assignment were to research meta-heuristic algorithms and come up with stochastic model to solve TSP as well as possible. In this assignment, I used ACO algorithm which algorithm's nature and principles attracted me a lot and I wanted dive deep into it. Algorithm is very simple yet very effective to find good solution in very short time. In every iteration of algorithm, we randomly initialise starting vertices for ants (as given vertices are assumed to be randomly initialized, by randomly initializing, model can work well on without any assumption on vertices) and complete tours by choosing following vertices probabilistically using pheromone level and length of the edges. At each iteration we deposit some pheromone to used edges in tours and evaporate every edge's pheromone level a little to avoid using same edges every time and search other edges as well.

2 Design decisions

Firstly, I have implemented the simple algorithm given out in the lecture slides [1]. I tried to make fully utilise best programming approaches to make the model modular and scalable. I created *Vertex and Graph* classes to keep the core information about the given data. *Vertex* class contains index and location of vertex in 2D graph. *Graph* class contains all of the necessary information to use ACO (vertices, edge weights and pheromone levels). Since distance between two vertices will be constant over every iteration of ACO algorithm, they are initialized in the beginning to avoid additional complexity. And since we will apply pheromone updates to every edge, I am using vectorized numpy arrays to reduce runtime. I have also created *Ant and ACO* classes. These classes are loosely coupled and strongly encapsulated which made the development process faster and efficient, and debugging very easier. *Ant* class contains methods for finding probabilistic tour and depositing pheromone to edges in each iteration. *ACO* class is more of agent to solve TSP. It contains **run()** to solve TSP and methods to save and visualize the best tour as well as logging feature to see best distance after every iteration and convergence of ACO.

3 Interface

TSP package provides very simple interface to run model and control hyperparameters.
Model can be run by calling **python3 main.py -p path-to-tsp-file** and following hyperparameters can be adjusted (For more information, you can check Figure 1.):

-mode	Either use ACO or visualize solution.csv
-cs	Colony Size
-it	Number of Iterations
-er	Evaporation Rate
-a	Pheromone Weight
-b	Visibility Weight
-log	Visualizing learning curve

```
def parse_arguments():
    parser = argparse.ArgumentParser()
    parser.add_argument('-mode', type=str,
                        default="ACO", help="Choose mode")
    parser.add_argument('-p', type=str,
                        default="./data/a280.tsp",
                        help="path to the input file")
    parser.add_argument('-cs', type=int,
                        default=50, help="colony size")
    parser.add_argument('-it', type=int,
                        default=100, help="Number of iterations")
    parser.add_argument('-a', type=float,
                        default=1.0, help="pheromone weight")
    parser.add_argument('-b', type=float,
                        default=3.0, help="visibility weight")
    parser.add_argument('-er', type=float,
                        default=0.2, help="evaporation rate")
    parser.add_argument('-log', type=bool,
                        default=True, help="Visualize learning curve")
    args = parser.parse_args()
    return args
```

Figure 1.

4 Experiments and Optimizations

There were three main constraints in optimization of ACO algorithm. First one is deposit function which deposits some amount of pheromone to used edges in the tours. The second one is simply the hyperparameter optimization, which is changing the colony size, number of iterations, evaporation rate, weighting parametemers. And last one is fixing irregularities in the completed tours.
To quickly verify the idea all of the methods has been testes on *a280.tsp* and *rd100.tsp* files.

4.1 Pheromone deposit

The function for pheromone deposit is following:

$$\Delta\tau_{ij} = \frac{Q}{L_k} \tag{1}$$

where Q is the estimated shortest tour and L_k is the length of the tour by ant k. Since we don't know what is shortest tour, we need to make some good estimation. We will do this estimation based on our current best tour length.
In the early iterations, since every tour is almost equally likely to be chosen, we expect shortest tour to be much better that this and as we go on to later iterations, we will estimate that our best tour distance is close to optimal distance.
Therefore, inspired by the one cycle policy[2] for searching for good learning rate, we apply annealing of best tour length with following algorithm. In order to not increase the complexity of main loop, we just keep it simple with linear annealing:

$goal \leftarrow 0.5$
$rate \leftarrow 0.4/iters$
for $i \leftarrow 0$ to iters **do**
  $Q \leftarrow goal * bestdist$
  $goal \leftarrow goal + rate$
end for

As it is randomly chosen, chosing very low initial goal is not good as our algorithm may have already found good solution. Also, to always search for better result, incrementing rate is not 0.5 but 0.4. Also, since our deposit will be less than 1, we initialize all of pheromones to 1 in the beginning.
This scheduling algorithm helped ACO converge faster and always try to look for better result as shown in Figure 2 (For a280.tsp with cs=100, it=50, er=0.15, a=1, b=1.5, Best tour achieved = 3294.94, while optimal tour is 2568.88)

For verifying idea, we just iterated 50 times, but as shown from the graph, algorithm is tend to search for better results as iteration goes on.

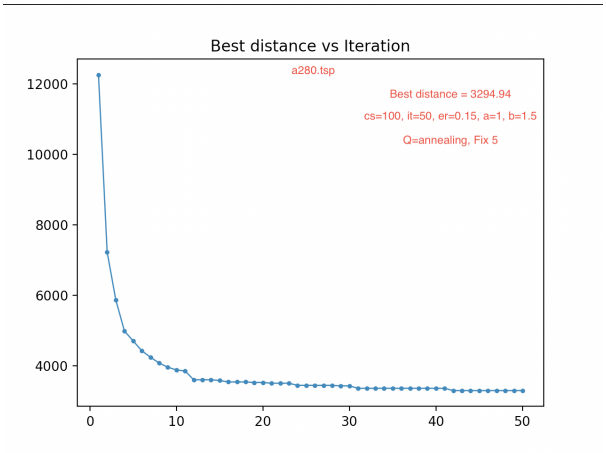

Figure 2.

4.2 Hyperparameter Tuning

Weights a and b are dependent on the given data and as we want our ACO to work for any given data. Values a and b can be adjusted accordingly after visualizing/knowing the distribution of vertices, but for general purpose, it might be better to keep them close by making b slightly larger.
For evaporation rate, if number of vertices in graph is large, it is better to set it to a little higher value to be able search more possible edges as we are not able to iterate much due to large number of vertices.
Finally, more colony size and iterations is always good as we do more iteration and our sample size is bigger, so we are more likely to get very good solution. But the problem is that they directly effect the complexity as they are the main two loops in ACO. For small to mid range size of vertices, they can be set around 100, while for large graph such as *rl*11849*.tsp* file, it is better to set around 20 as we will later show that our ACO algorithm's complexity is $O(C * S * |V|^2)$ where C is colony size, S is number of iterations and $|V|$ is number of vertices.

4.3 Fixing Irregularities in tour

As I plotted the final results, I saw that there were geometrical problem in the tour as edges were chosen probabilistically. So, I applied simple deterministic solutions for this. I have applied two similar techniques called *fix_tour _4* and *fix_tour _5* functions. First one iterates over the completed tour by ant and in each iteration, it looks at four consecutive vertices x, y, z, w in tour and chooses minimum of $x- > y- > z- > w$ and $x- > z- > y- > w$ paths. Similarly, the latter one takes five consecutive vertices x, y, z, u, v in tour and takes the minimum path from the permutations of $x- > \pi(y, z, u)- > v$. This algorithm's complexity for *fix_tour _n* is $O((n-2)!)$, so we

will stop here and we are not gonna apply it for six or more vertices to keep it as meta-heuristic.
Results from applying these functions is guaranteed that we will end up better solution, so it is safe to apply this algorithm.
Difference between with vanilla ACO and after applying these methods is shown in Figure 3, 4, 5, 6 (Optimal, Vanilla, Fix4, Fix5) and Table 1 for rd100.tsp file. Hyper-parameters were: cs=100, it=50, er=0.15, a=1.5, b=2.0.

Table 1. Fixing irregularities

Method	Best distance
Optimal	7888.73
Vanilla	8315.93
Fix 4	8445.24
Fix 5	8224.93

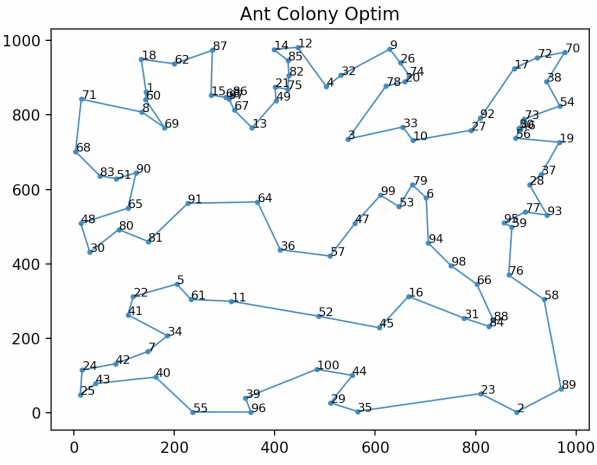

Figure 3.

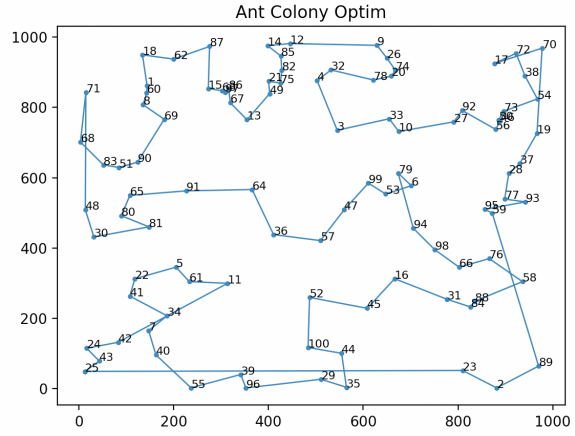

Figure 4.

We can see that proposed method (Figure 5, 6) fixes problems in vanilla (Figure 4) method that can be easily

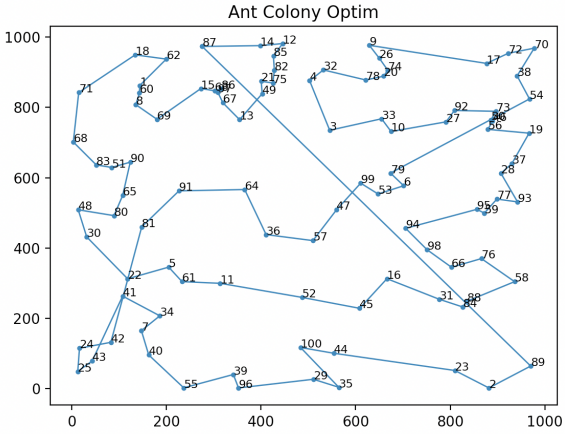

Figure 5.

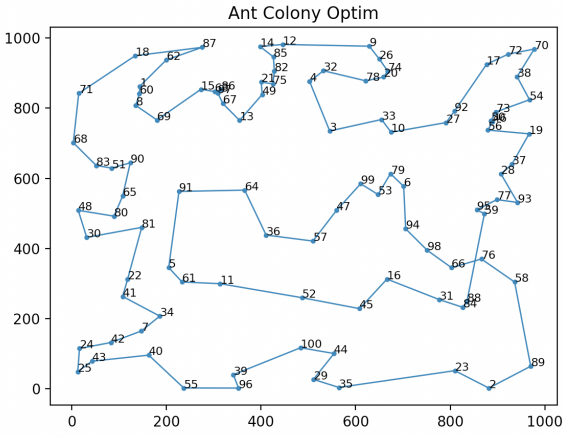

Figure 6.

seen by human eyes.
Purpose of these methods is not that it gives us best result in the end (as everything is probabilistic), but it at least fixes obvious mistakes that can be determined by human eyes and guarantees to give smaller best tour for chosen complete tour by ant.

5 Complexity Analysis

Before we start, lets denote some variables: V is set of vertices, E is set of edges, C is colony size, and S is number of iterations.
In preprocessing step of model, we first read the vertices. For given vertices V, we set up graph $G = (V, E)$ where this graph is isomorphic to $K_{|V|}$ and E will have weights, which is length of edges, and pheromone levels of edges. So, the total complexity of preprocessing will be

$$O(|V| + |E|) = O(|V|^2)$$

In the main loop of algorithm, in each iteration we will

complete the tour for each ant and then we will update the pheromones in all edges as well as pheromones in used edges by ants.

- Updating pheromone level is $O(|V|^2 + C * |V|)$
- In getting complete tour, we first choose initial vertex randomly, then we iterate $|V| - 1$ times to choose remaining vertices. For selecting each vertex, we iterate over unvisited vertices, and compute $phrm^a * (1/dist)^b$ and in the end, we divide all of them to their sum to get the probabilities. In the end we will randomly choose the next vertex by given probabilities with $O(|V|)$. Therefore, time complexity of completing tour will be $O(|V|^2)$
- After completing tour, we also apply fixing function which costs $O(|V|)$ time

So, time complexity of the ACO model is

$$O(S * C * |V| * (|V| + C))$$

.

6 Conclusion

In this report, I have proposed Ant Colony Optimization method with annealing for estimating shortest tour and deterministic approach to fix the irregularities occured in the graph.
While checking researches on ACO, all of them used constant Q which creates critical issue that algorithm may not work for any given TSP problem. Estimating shortest tour is useful since we cannot set one constant value to be general purpose for any given TSP problem and I proposed method to solve it by changing this fixed factor during runtime instead of treating it as a constant, by expecting that as we go on we will have better solution. This method showed great results on any given TSP problem and helped model to converge faster.
Also, I applied deterministic path fixing to get shorter solution for given path.
For the future work, we can apply better estimation for the shortest tour Q. As used in learning rate schedulers, we can apply few iterations for warm-up or instead having linearly increasing goal factor, we can make it more sophisticated.

7 References

[1] Lecture Slides on Ant Colony Optimization
[2] One Cycle Policy `https://sgugger.github.io/the-1cycle-policy.html`