IE-716: Project Report

Developing a Hybrid Algorithm based on Ant Colony Optimization (ACO) Heuristic for Solving the Travelling Salesman Problem (TSP)

By Team Optimizers,
Chinmay Naik (203190016)
Amit Gadekar (203190022)
Anand Siyote (203195001)

Under Guidance of
Prof. Ashutosh Mahajan, IEOR, IITB.

Abstract - Ant Colony Optimization algorithm (ACO) is a well known heuristic in computer science as well as operations research which belongs to the category of 'swarm intelligence'. It is a probabilistic technique for solving computational problems which can be ultimately reduced to finding good paths through graphs. The first ACO algorithm was put forth by M. Dorigo in 1992, known as 'Ant System', which later evolved into several different variants like Ant Colony System (ACS), Elitist Ant System (EAS), Max-Min Ant System (MMAS) and many more. In the first part of the project, we have tried to implement ACS, EAS and MMAS through a python program and done some basic comparative study among them using the famous 'Travelling Salesman Problem'(TSP). In the second part, we have combined ACO with a couple more simple heuristics (Nearest Neighbor Search and 2-opt local search) to create a system that dynamically keeps generating solutions for the given TSP problem. At the end, we used a few standard TSP problems from TSPLIB-95, an online available library of TSP datasets with known best solutions, and compared the performance of our system with them, by running the code on our local machines.

I. Introduction

Ant Colony Optimization falls under a field in artificial intelligence known as 'Swarm Intelligence'. Swarm Intelligence (SI) is the collective behavior of a decentralized, self-organized system, which can be natural or artificial. SI systems consist typically of a population of simple agents or boids interacting locally with one another and with their environment. The inspiration mostly comes from nature, especially biological systems. The agents follow some very simple rules, and although there is no centralized control structure dictating how they should behave individually; the interactions between these agents lead to the emergence of some kind of intelligent behavior which is unknown to those agents themselves.

This quality is observed in a colony of ants as well, when they need to search for some food and bring it back to their colony. Ants are almost blind. They cannot see the food which is far away. However they have a pretty interesting mechanism to search for food. Ants while travelling tend to release a kind of chemical secretion from their bodies on the path known as ‘pheromone’. This pheromone helps the ant to find its way back to the colony, as well as the other ants to reach the food source by following it.

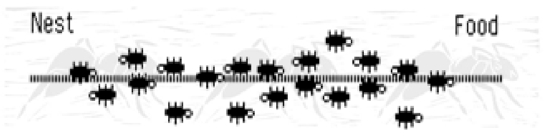


Fig.1: Ants following their normal path

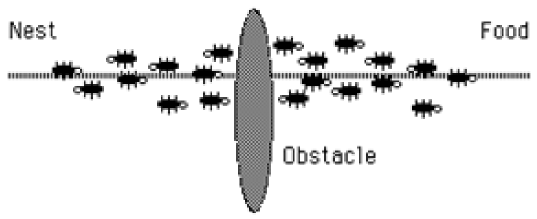


Fig.2: An obstacle is inserted in their path

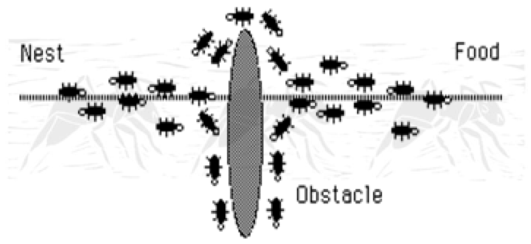


Fig.3: Ants randomly choosing one of the two paths

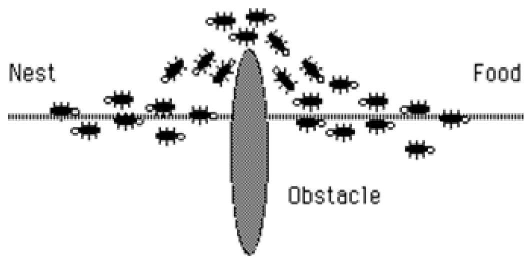


Fig.4: Ants finally selecting shortest path based on amount of pheromone deposition w.r.t time.

From Fig.1 to Fig.4, it can be seen the way ants always tend to choose the shortest path over the course of time. As the speed of ants is constant, the shorter path gets more pheromone deposited on it as the time passes, and more and more ants tend to follow the path with more pheromone deposited on it. As a result, after some time, all the ants start following the shorter path in terms of length. This can be shown in a more mathematical way in Fig.5. We will be using the same concept in the Ant Colony Optimization algorithm by generating some artificial ants and by putting artificial pheromones on the edges.

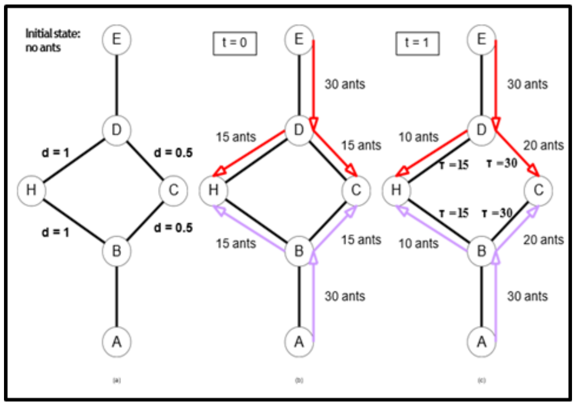


Fig. 5: Mathematical representation of working of ACO.

We will see in the upcoming sections how this decentralized system is converted into an algorithm and implemented on one of the most famous problems in combinatorial optimization known as the ‘Travelling Salesman Problem’ (TSP).

II. Travelling Salesman Problem (TSP)

Travelling Salesman Problem needs no introduction as almost everyone studying Integer Programming or Combinatorial Optimization is well aware of it. Still for the sake of formality we will briefly look into it:

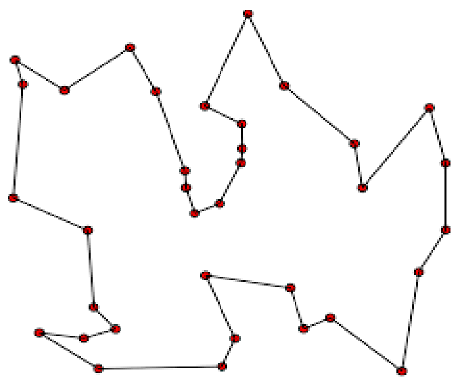


Fig.6: A typical image of a (solved) TSP problem

In TSP, we are generally given a list of cities (nodes) and the distance between each pair of cities (cost matrix). The salesman has to start from any of the nodes and visit all of them exactly once to return back to the initial one. The aim is to find the shortest path to achieve this goal.

TSP is an NP-hard problem. Which means that there isn't any algorithm that can solve the given TSP with polynomial time complexity. All the algorithms that guarantee the optimal solution, tend to have exponential time complexity. However there are many 'heuristics' which can lead to a suboptimal solution (which may not be the best, but is good enough to be considered) in polynomial time. ACO is one such heuristic which can keep running for as many number of iterations as we want, and keeps generating better and better suboptimal solutions, which may converge to the optimal one much faster than the exact algorithms in many cases (although there is no way to verify that the solution found is the optimal one or not).

III. ACO Algorithm

The first version of ACO was developed by M. Dorigo in 1992. It was called 'Ant System' (AS) and was particularly developed to solve the TSP. In this algorithm or in any of its other versions; there are some artificial ants generated in every iteration which choose to move from one node to another based on some predefined rules.

The algorithms works like this:

❏ At the beginning of every iteration, we will place some ants randomly on some cities. Each ant has to complete a full tour of the map covering each and every city by making its own choices. The choice will depend on two factors:

1. First factor is 'visibility'. A nearer node has more chance, whereas a distant one has less chance of being chosen.
2. Amount of pheromone. The more intense the pheromone trail laid out on an edge between two cities, the greater the probability of choice.

❏ Having completed its journey, the ant deposit will deposit a pheromone train on it's path inversely proportional to the length of the tour.

❏ After each iteration, the trails of pheromone will evaporate by some fraction.

❏ At the end of every iteration, the ants will die and new ants will be placed randomly.

❏ The best solution is stored and updated at the end of every iteration.

❏ This process continues for as many iterations as we want.

A schematic of this algorithm is shown in Fig. 7. This series of steps is followed by almost all variations of ACO, the only difference is the pheromone updating rules at the end of each iteration. Based on that we will see different variants of ACO like Ant Colony System (ACS), Elitist Ant System (EAS) and Max-Min Ant System (MMAS) in the upcoming section.

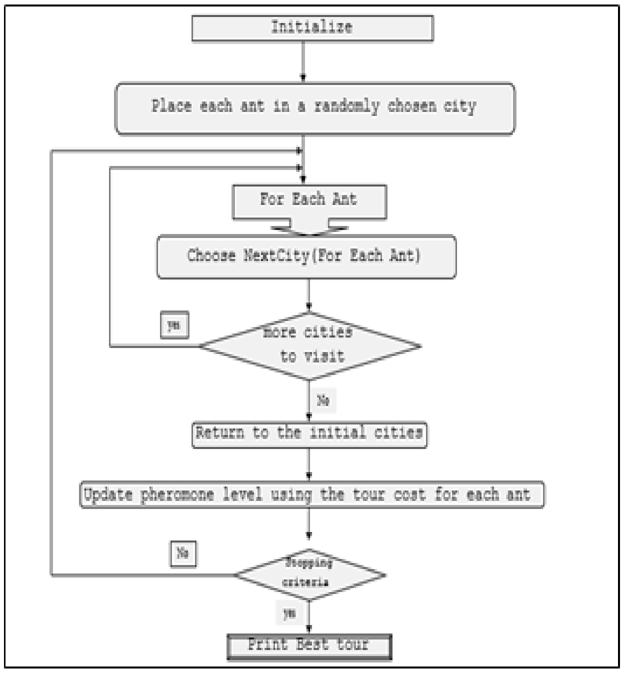

Initialize
Place each ant in a randomly chosen city
For Each Ant
Choose NextCity(For Each Ant)
yes
More cities to visit
No
Return to the initial cities
Update pheromone level using the tour cost for each ant
No
Stopping criteria
yes
Print Best tour

Fig.7: Schematic of ACO

Variables used in ACO:

1. $\tau_{x,y}$: Amount of pheromone present on the path between node x and y.
2. $\eta_{x,y}$: Visibility of city y from city x. (inversely proportional to distance d
3. α, β : Influence controller of amount of pheromone and visibility respectively for probability calculation.
4. ρ : Amount of pheromone evaporation after each iteration.

Probability of choosing next city:

$$P^{k}_{xy} = \frac{(\tau_{xy})^{\alpha}(\eta_{ny})^{\beta}}{\sum_{z \in allowed}(\tau_{xz})^{\alpha}(\eta_{xz})^{\beta}}$$

Pheromone update rule:

$$\tau_{xy} \leftarrow (1 - \rho)\tau_{xy} + \sum_{k} \Delta\tau^{k}_{xy}$$

Where, $\Delta\tau^{k}_{xy}$ is the amount of pheromone deposited by kth ant; which is given by:

$$\Delta\tau^{k}_{xy} = \begin{cases} \frac{Q}{L_k} & \text{if ant k has visited xy in previous tour.} \\ 0 & \text{otherwise.} \end{cases}$$

Versions of Ant Colony Optimization:

Ant Colony System (ACS):

❏ All ants complete their tour. The global best tour is updated if any.
❏ Pheromone is updated on every ant’s tour.
❏ Previous pheromone evaporates.

Elitist Ant System (EAS):

❏ All ants complete their tour. The global best tour is updated if any.
❏ Pheromone is updated on every ant’s tour.
❏ Some additional pheromone is added on the current Global best tour.
❏ Previous pheromone evaporates.

Max-Min Ant System (MMAS):

For first (say 75%) tours:

❏ The best path in that step is stored as ‘iteration best tour’.
❏ Pheromone is added only on the iteration best tour for each step.

For remaining (25%) tours:

❏ A variable Max_pheromone is set according to the last ‘iteration best tour’.
❏ Min_pheromone is set by scaling Max_pheromone with a constant.
❏ For each iteration, the amount of pheromone on each edge is made to fit in (Min_pheromone , Max_pheromone) range.
❏ For every iteration pheromone is added only on the global best tour.

Comparison between ACO and Exact methods:

We have compared the performance of different versions of ACO in terms of convergence. For this we randomly generated various datasets for TSP and observed their behavior. A sample observation is like this:

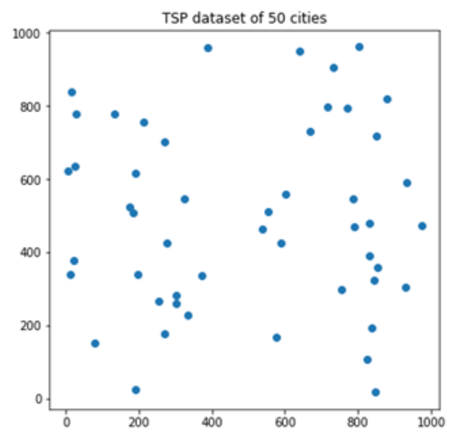


Figure 8: Dataset used for ACO methods comparison

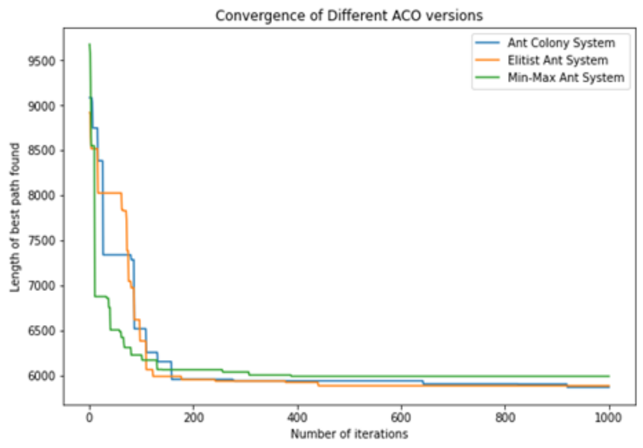
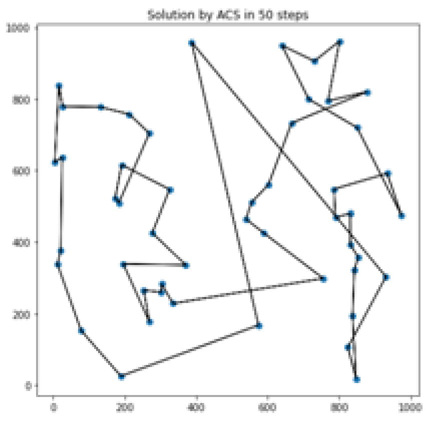


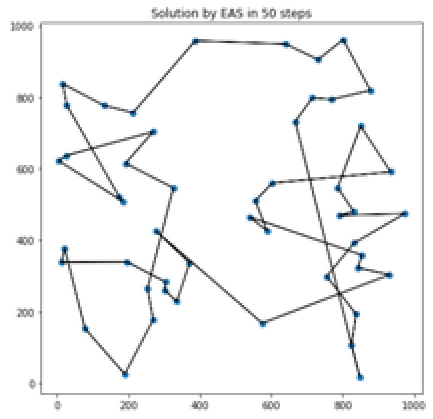
Figure 9 : Convergence of ACO methods

Our observations:

1. MMAS has much faster convergence compared to EAS and ACS in the first few steps.
2. However soon, the other two join MMAS and continue to converge
3. Elitist and ACS are often seen to perform slightly better than MMAS in long run

This is how the solutions looked after 50 steps:





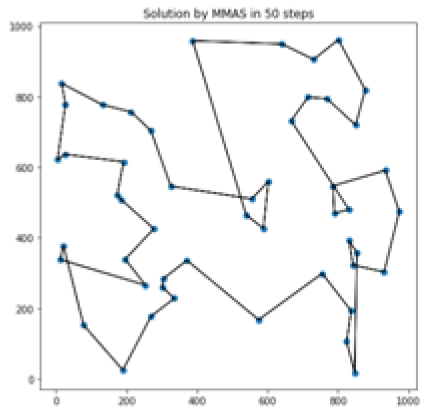


Figure 10: ACS vs EAS vs MMAS (50 steps)

Other Heuristics used:

Nearest-Neighbor search:

- ❑ NN search is a simple search algorithm to search for the shortest path.
- ❑ You start from a node.
- ❑ Look for all unvisited cities.
- ❑ Travel to the nearest one.
- ❑ Keep doing this till all cities are visited.
- ❑ Return back to where you started.

2-opt search heuristic:

- ❑ 2-opt search is a simple search algorithm.
- ❑ You start with an initial route.
- ❑ Check if swapping a pair of edges can make your path better.
- ❑ If so, swap them and continue.
- ❑ Keep doing this till the stopping condition is reached

Performance of NN and 2-opt on same dataset:

We tested the same dataset of 50 cities with NN, 2-opt and the combination of both

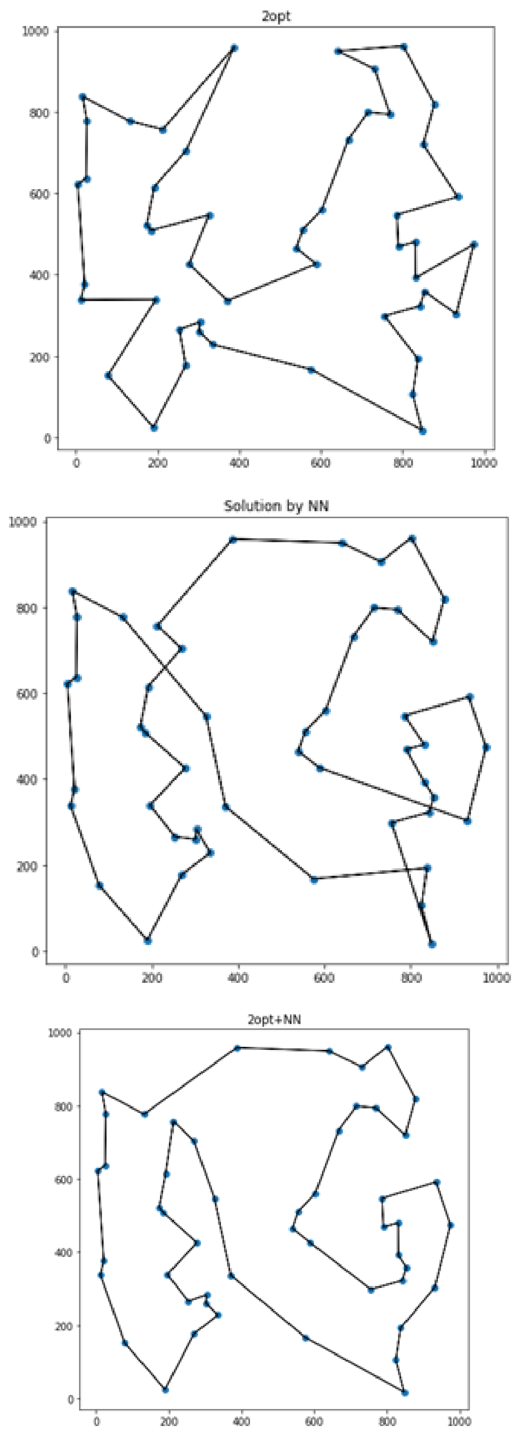


Figure 11

The Combination of NN and 2-opt search gives us a pretty good solution in a very short amount of time. The only drawback of this combination is that unlike an algorithm like ACO, the solution stops converging at a point and there is no way further!

Novel proposed system to solve TSP:
We did some experiments and came up with a novel system that keeps solving the TSP problem and dynamically keeps showing the current best solution after a few number of steps.
(This is a combination of NN search + Elitist ACO + 2opt search.)

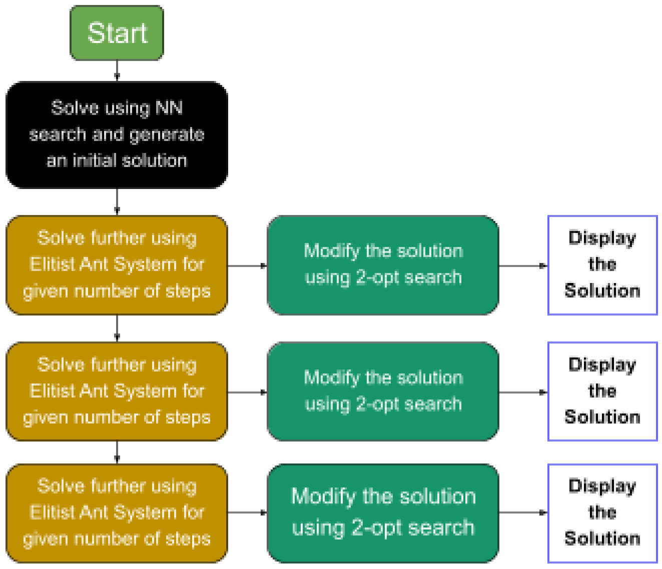

Start
Solve using NN search and generate an initial solution
Solve further using Elitist Ant System for given number of steps
Modify the solution using 2-opt search
Display the Solution
Solve further using Elitist Ant System for given number of steps
Modify the solution using 2-opt search
Display the Solution
Solve further using Elitist Ant System for given number of steps
Modify the solution using 2-opt search
Display the Solution

Figure 12

Steps:

1. Solve given problem using NN search and generate an initial solution
2. Solve further using Elitist Ant System for given number of steps
3. Modify the solution using 2-opt search
4. Again solve further using Elitist Ant System for given number of steps
5. Modify the solution using 2-opt search
6. Again solve further using Elitist Ant System for given number of steps
7. Modify the solution using 2-opt search
8. Display the solution

In short algorithm is like:

- ❏ Generate initial solution using NN search algorithm
- ❏ For given number of iterations:
    - Solve using Elitist Ant System
    - Modify the solution using 2-opt search
    - Store the solution
- ❏ Display best solution

Performance of our system with some standard datasets:

We tested our system with some well known datasets in TSPLIB-95 online dataset library with known best solutions given. Here is how our system performed:

TSPLIB Dataset	**Fri26**
No. of nodes	26
Best known Solution	**937**
NN solution	1112
NN+2opt solution	953
Our system's solution	**937**
Time taken to reach the solution	0.87 Sec
Error (%)	0.00%

Table 1

TSPLIB Dataset	**Berlin52**
No. of nodes	52
Best known Solution	**7542**
NN solution	8980.92
NN+2opt solution	8114.35
Our system's solution	**7544.37**

Time taken to reach the solution	5.29 Sec
Error (%)	0.03%

Table 2

TSPLIB Dataset	**Eli-101**
No. of nodes	101
Best known Solution	**639**
NN solution	825.24
NN+2opt solution	690.88
Our system's solution	**653.74**
Time taken to reach the solution	18.05 Sec
Error (%)	0.00%

Table 3

TSPLIB Dataset	**CH-150**
No. of nodes	150
Best known Solution	**6528**
NN solution	8194.61
NN+2opt solution	6837.27
Our system's solution	**6620.8**
Time taken to reach the solution	34.48 Sec
Error (%)	1.42%

Table 4

A sample result: (Visual representation)

CH-150 Dataset:

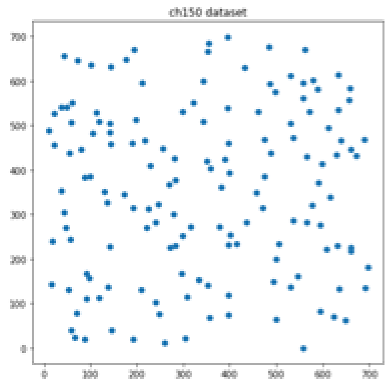


Figure 13: **Dataset**

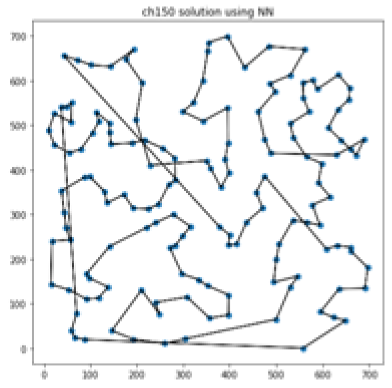


Figure 14: **NN solution (8194)**
Error = 25.52%

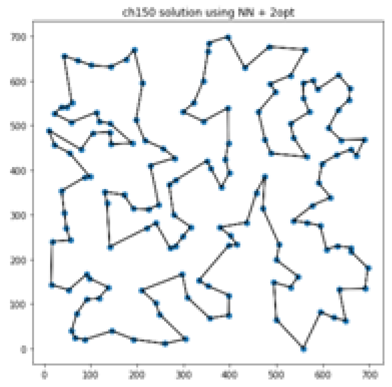


Figure 15: **NN + 2opt solution (6837)**
Error = 4.73%

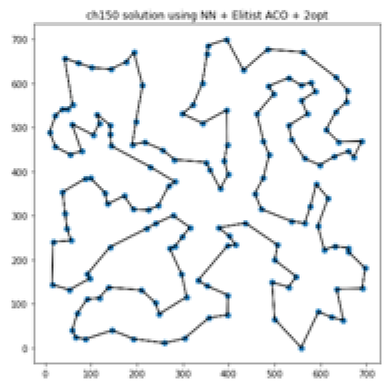


Figure 16: **Our solution (6620)**
Error = 1.42%

Results:

From the above observations:

- Errors in the case of our model are less than other discussed heuristics. In the above mentioned problems, our model achieved **below 2% deviation from global optimal solution.**
- Time required for convergence in our model is very less as compared to exact methods. The above mentioned problems were solved to a reasonably suboptimal solution **within 40 seconds** on a regular laptop.

CONCLUSION:

Advantages:

- **The hybrid system** made by combining **NN search + Elitist ACO + 2opt search** seems faster and more powerful compared to the individual algorithms.
- The system **keeps generating solutions dynamically** and goes on converging as per our observations. So we can get a good early solution as well as a better and better solution with time.
- **Parallel computation** is possible in this system. The output of Elitist ACO can be sent to another machine to perform 2-opt search.

Disadvantages:

- The convergence and speed seen in the previous slide is not as clean as it always appears. Often we need to tune the parameters in the initial stages before proceeding further to get a good convergence
- There is no way to know if we have reached to or near the optimal solution.
- In some cases, the solution might even stop converging after reaching some good result for a long time (we don't know how long)

Future work :

- Better combinations of other Heuristics available can be tried as well.
- Use of a better search algorithm like K-opt local search instead of 2-opt search.
- Need to build a method to pre-tune the parameters better for a given dataset.

References:

[1] Nwamae, Believe B., and G. Ledisi. "Solving Traveling Salesman Problem (TSP) Using Ant Colony Optimization (ACO)." International Journal Of Engineering Research & Technology (IJERT) 7.07 (2018).

[2] Stützle, Thomas, and Marco Dorigo. "ACO algorithms for the traveling salesman problem." Evolutionary algorithms in engineering and computer science 4 (1999): 163-183.

[3] Nilsson, Christian. "Heuristics for the traveling salesman problem." *Linkoping University* 38 (2003): 00085-9.

[4] Datasets : TSPLIB_95
(http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/)

[5] Reference code for basic structure of ACO:
(https://github.com/rochakgupta/aco-tsp)