Program Structures and Algorithms

Spring 2023(SEC –01)

Project Report Team-02

Team members:

NAME: Ashi Tyagi **NAME: Shivani Datar** **NAME: Swarag Sanjay Gutte**

NUID: 002706544 **NUID: 002772160** **NUID: 002728422**

Github Repository Link :- https://github.com/shivanidatar/INFO6205_FinalProject

Introduction (Aim & Approach)

One of the combinatorial optimization problems that has received the most research is the traveling salesman problem (TSP). It has turned into a testbed for new calculations as it is not difficult to think about the outcomes among the distributed works. A pure brute force algorithm requiring factorial time is required for this NP-hard problem. In plain words, the TSP poses the accompanying inquiry: Given a rundown of urban communities and the distances between each sets of urban communities, what is the most brief conceivable course that visits every city and gets back to the beginning city?

The idea of the traveling salesman problem (TSP) is to find a tour of a given number of cities, visiting each city exactly once and returning to the starting city where the length of this tour is minimized. In the symmetric TSP, the distance between two urban communities is similar in every contrary heading, framing an undirected diagram. This evenness parts the number of potential arrangements. Calculating the number of distinct tours through n cities would be the most straightforward approach to a TSP problem's resolution. It has n-1 choices for the second city, n-2 choices for the third city, etc., given a starting city. It is possible for paths to not exist in both directions and for the distances to be different in the asymmetric TSP, resulting in a directed graph.

Our aim is to find the optimal length of the tour in meters for the salesman with the cities plotted as the points provided to us in the dataset which is of about 585 points(crimeIDs). We have done so by making the Christofides algorithm the main or basis of our solution and then optimize this by using tactical and strategic algorithms. We use 2-opt and 3-Opt algorithms for our tactical approach and make use simulated annealing and ant colony optimization for our strategic algorithms. We have also visualized our findings by providing UI for all the tours plotted for different algorithms.

Program (Classes, Algorithms and invariants)

Classes

The basic folder structure has the files arranged in a stair cascade manner with the main classes arranged inside the src folder with working code in the info6205_team02 folder. It has different packages with names of different algorithms. The packages then all have a driver class which is the main class for that particular algorithm and a window_tsp class which is called from the driver class itself and it provides the UI part of the code. The rest of the classes are made dependent on the need of the algorithm. The christofides package has the 2-opt and 3-opt code inside it. The other two packages mainly are built on optimizing this tour with ant colony and simulated annealing method. Different data structures have been used according to the need of the code, but mostly linked lists, Arraylist have been utilized and 2-d matrices have been used to create the distance matrix for the christofides code.

Haversine Distance Formula

The haversine distance formula is a mathematical formula used to calculate the shortest distance between two points on a sphere, such as the Earth. The formula is named after the haversine function, which is used in the calculation.

The haversine distance formula is based on the law of haversines, which states that the haversine of half the angular distance between two points on a sphere is equal to the sine squared of the angular distance between the points. The formula considers the curvature of the Earth and is therefore more accurate than using a simple Euclidean distance formula.

Following way, we have calculated the distance in our program.

```
double radius = 6371;
double x1 = Math.toRadians(a.getX());
double y1 = Math.toRadians(a.getY());
double x2 = Math.toRadians(b.getX());
double y2 = Math.toRadians(b.getY());
double deltaLat = x2-x1;
double deltaLon = y2-y1;
double e = Math.pow(Math.sin(deltaLat/2),2)+(Math.cos(x1)*Math.cos(x2)) * Math.pow(Math.sin(deltaLon/2),2);
double c = 2 *Math.atan2(Math.sqrt(e),Math.sqrt(1-e));
double distance = (radius * c);
return distance;
```

Christofides Algorithm

In 1976, Nicos Christofides developed an algorithm to find approximate solutions for the Traveling Salesman Problem, provided the situation is symmetric and obeys the triangle inequality. The triangle inequality states that for every three vertices x, y, and z, the weight of the edges, w, should satisfy the condition w(x y) + w(y z) ≥ w(x z). The algorithm starts by constructing a minimum spanning tree, T, from a symmetric complete graph G. Next, all odd

degree vertices in T are found and placed in set O. An odd degree vertex has an odd number of edges connected to it. Then, the odd degree vertices are connected using the minimum weight perfect matching, adding edges to the matched odd degree vertices to create an Euler cycle for T. This step ensures that every vertex in T has an even degree. Finally, repeated edges are removed from vertices with a degree greater than 2 to create a Hamiltonian cycle. The Christofides algorithm is an approximation algorithm, as the solution is guaranteed to be within a factor of 3/2 of the optimal solution. However, it can achieve this result much faster than an exact algorithm. The time complexity of the algorithm depends on the algorithm used to find the minimum spanning tree.

2/3- OPT

The 2-opt algorithm is a heuristic method used to find a good solution to the traveling salesman problem. It involves removing two edges from the tour and reconnecting the resulting sub-tours in a way that maintains the tour's validity. This process is repeated until no further improvement can be made. The 3-opt algorithm works in a similar way but involves removing three edges instead of two. Reconnecting the sub-tours in a valid way can be done in two possible ways, which are equivalent to single 2-opt moves. However, 3-opt moves can be seen as two or three 2-opt moves and are slower than 2-opt moves due to their O(n3) complexity. A tour that is 3-optimal is also 2-optimal, but a 3-opt exchange can provide better solutions.

Simulated Annealing

Simulated Annealing is a stochastic global search algorithm which means it uses randomness as part of its search for the best solution. It derives its name and inspiration from a similar process named annealing in metallurgy whereby heating or cooling a metal affects its physical characteristics. SA uses this concept of temperature in determining the probability of transitioning (stochasticity of the search) to a worse solution in order to more widely explore the search space and have a better chance of finding the global optimum. This helps avoid getting stuck in a local optimum which algorithms such as hill climbing often do.

Ant Colony Optimization

The Ant Colony Optimization algorithm, or ACO, was proposed by Marco Dorigo in 1992 in his PhD thesis. ACO is an algorithm that searches and finds the most optimal solution for the Traveling Salesman Problem graph based on the behavior of ants between their colony and a food source. The algorithm is a probabilistic technique due to the ants wandering randomly. The idea is that as ants search for food, they leave a pheromone trail for other ants in the colony to follow. However, the pheromone trail starts to evaporate as time goes on. The longer it takes for

an ant to travel a path, the more the pheromone trail evaporates, leaving a weaker attractive strength. Thus, the shorter the path to the food source is, the stronger the attractive strength due to a greater number of ants following the stronger scent. Given an n-city Traveling Salesman Problem with distances di j , artificial ants are distributed to these n cities randomly. Based on the pheromone trail remaining on the paths, each ant will choose the next city to visit. The main differences between artificial ants and real ants are that artificial ants will not visit cities that they have already visited, and they can also know the distances between two cities, unlike real ants.

Invariants

2/3OPT :-

- Tour connectivity :- In the tour all cities stay connected
- Tour distance :- Upon each swap, if the new tour distance is less than tour distance, the vertices are swapped to give the optimal tour distance. Therefore we can say that the tour distance either stays the same or decreases after every swap.
- Edge distinctness :- As we use the Hamilton cycle for 2/3opt, we can say throughout the every swap iteration every edge is distinct.

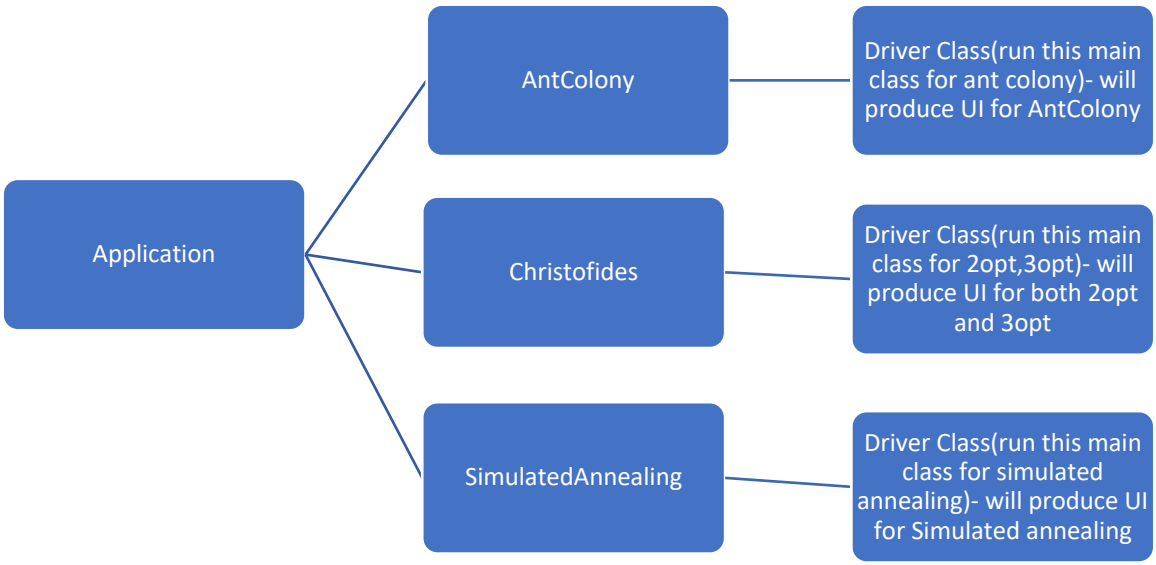
Simulated Annealing :-

- This algorithm is stochastic in nature.
- Starting Temperature :- In our implementation the starting temperature is 10, it changes according to the cooling schedule, in our case in every iteration t = t * cooling_rate. But the temperature stays the same for each iteration.
- Tour distance :- After swapping two cities randomly, it is checked if the new distance tour is less than existing tour distance, else it is checked if the Euler number of the difference of distances divided by the temperature is less than Math.Random() number, the swap is reverted. Therefore, we can say that each iteration the distance remains the same or decreases.

Ant Colony :-

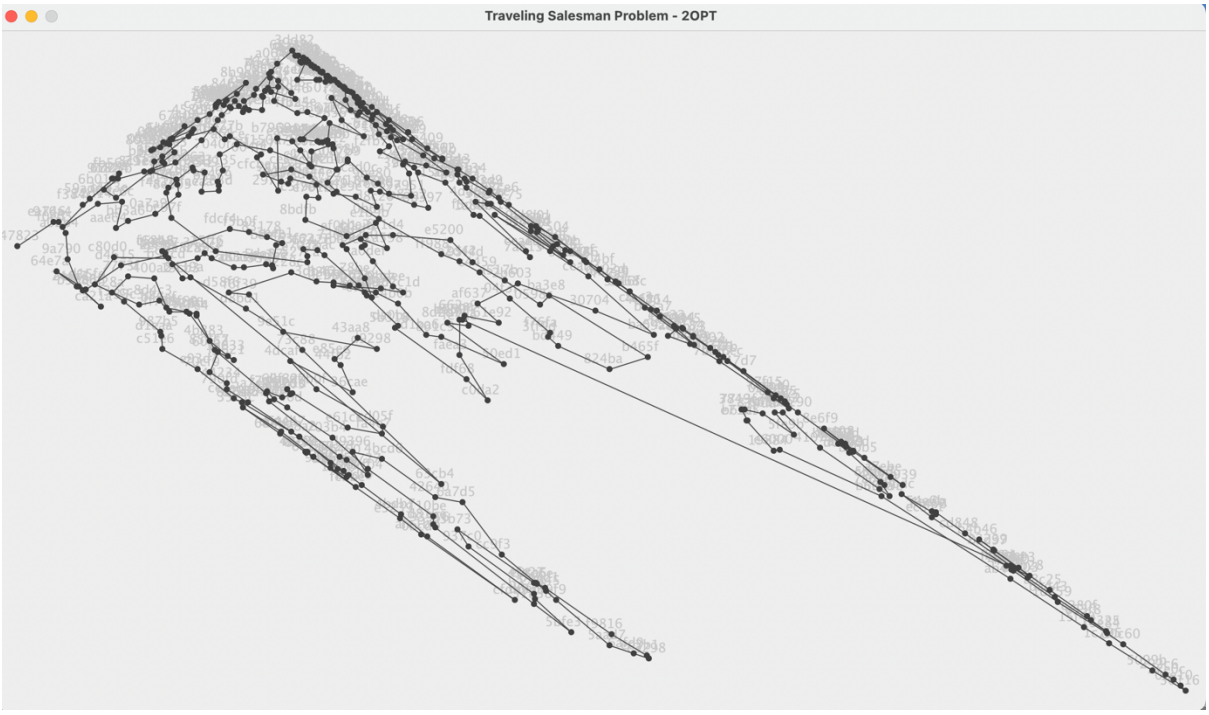
- Pheromones:- The trail of ants picks up the path where they sense more amount of pheromones. The trails corresponding distances between the cities and the alpha, beta factor helps in calculating the probability of how many pheromones will be on a given edge/path. Throughout the algorithm it stays constant that the next city is chosen with more amount of pheromones (probability)
- Evaporation rate :- The trails are also dependent on the evaporation rate. This is the rate by which the pheromones will be evaporating and eventually lose its effect. This is used to determine the current path with maximized pheromones.

Flow Charts (UI Flow)

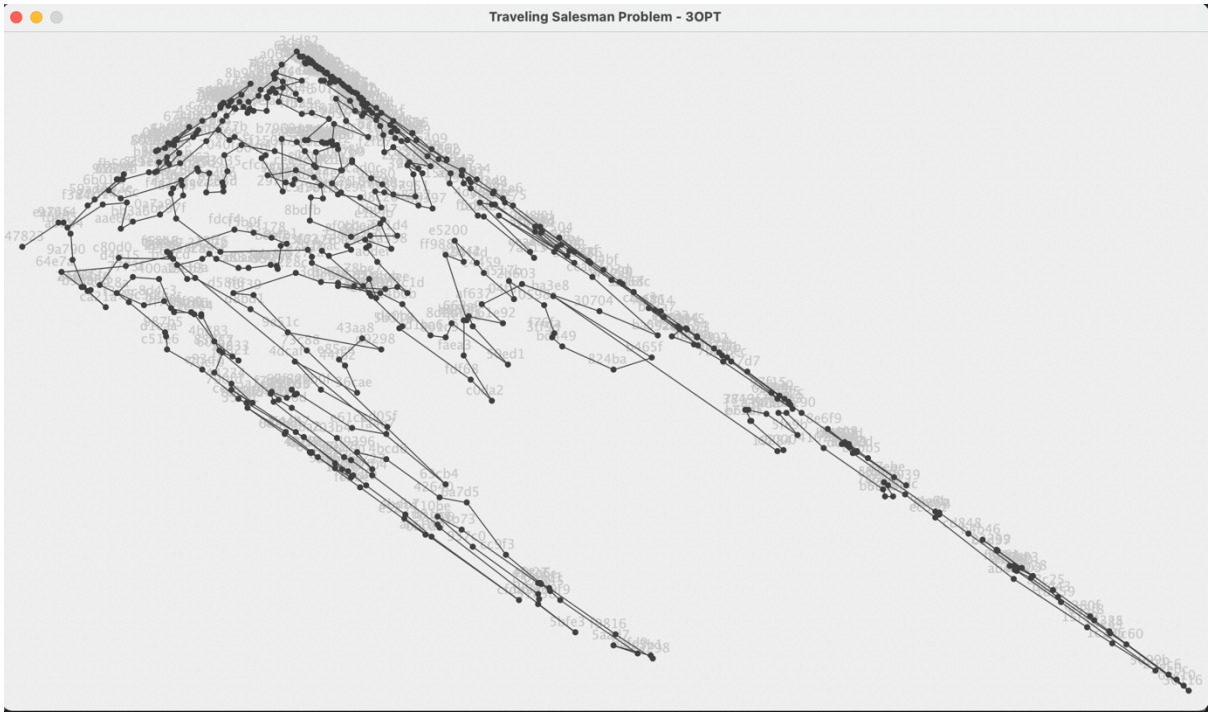


UI screenshots :-

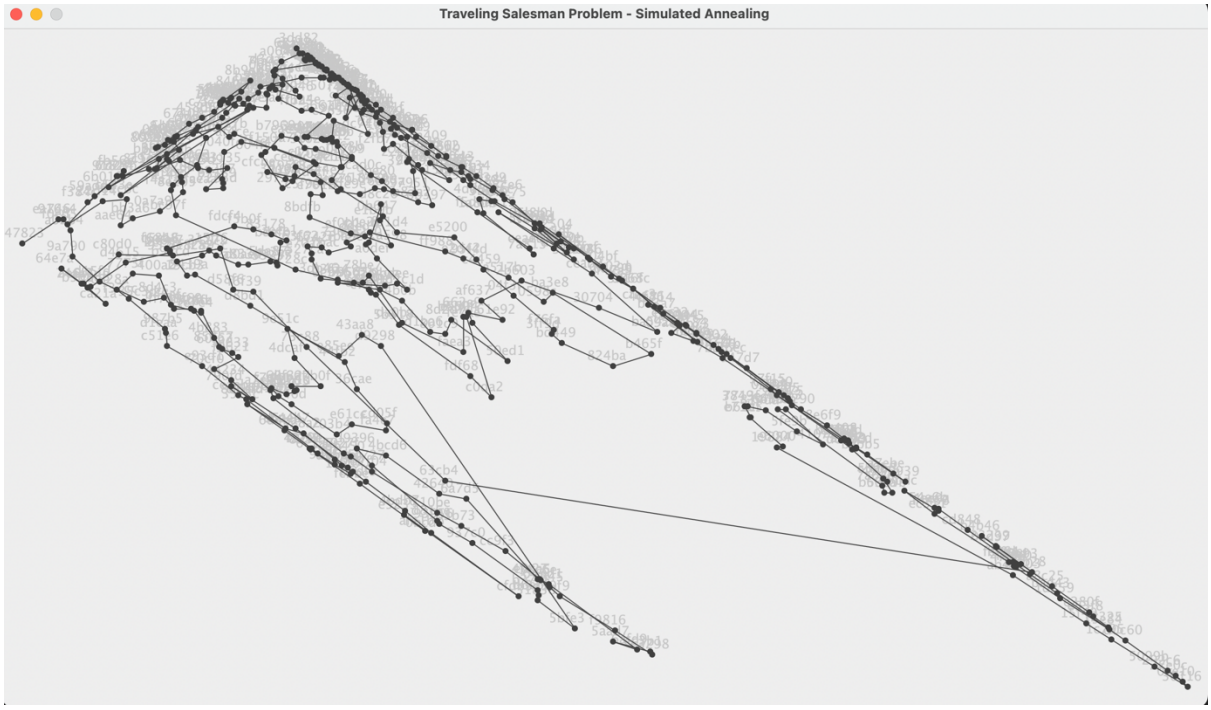
2Opt:-



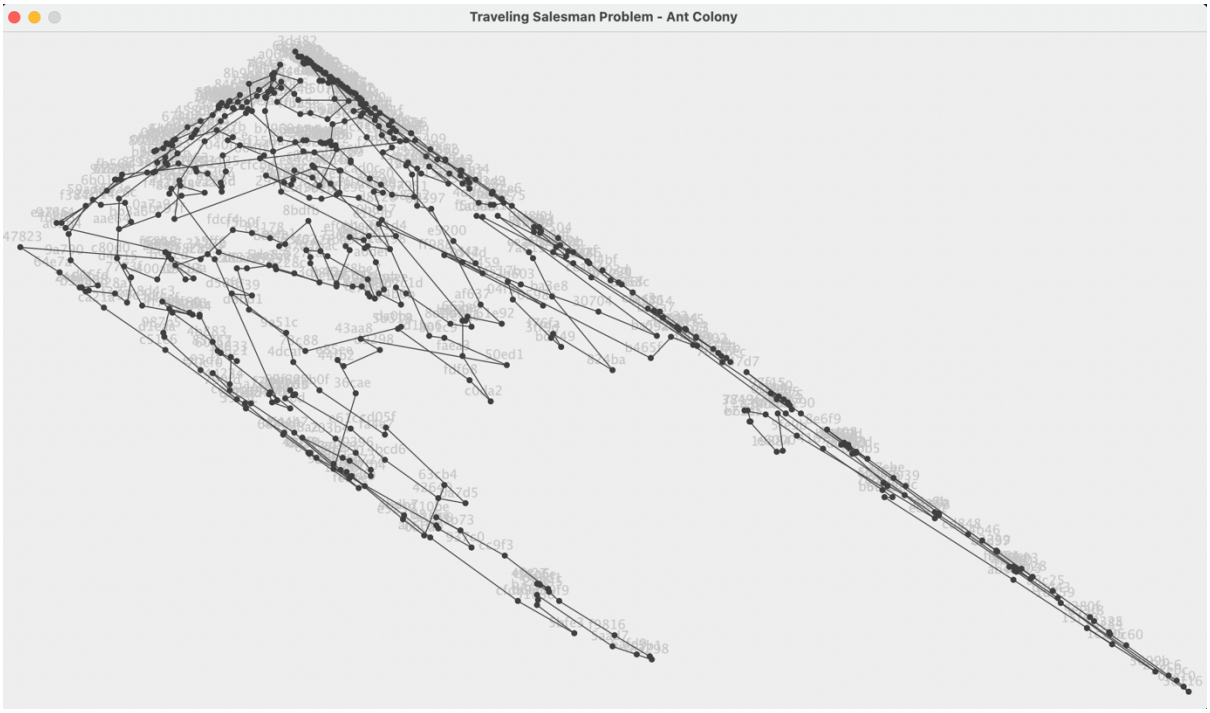
3Opt :-



Simulated Annealing :-



Ant Colony :-

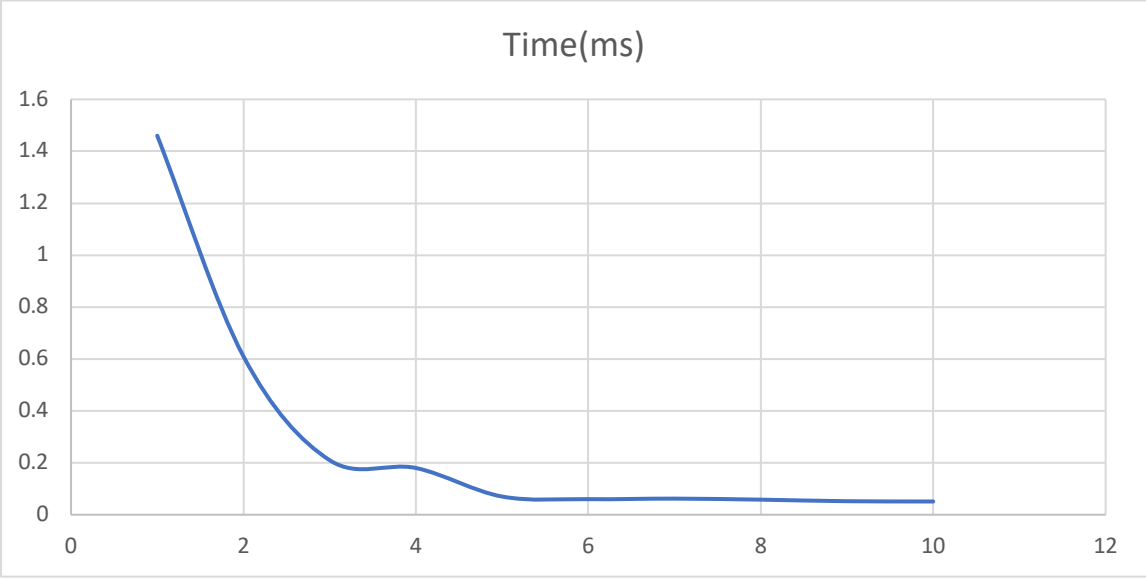


Observations and Results

Simulated annealing:

The graph here is plotted with taking 10 arbitrary iterations and their time taken. The trend of the graph shows an almost straight or constant curve at last few iterations with the time being around in the range of 0 - 0.2ms for all the last 200 iterations. The overall time required is 0.20 ms.

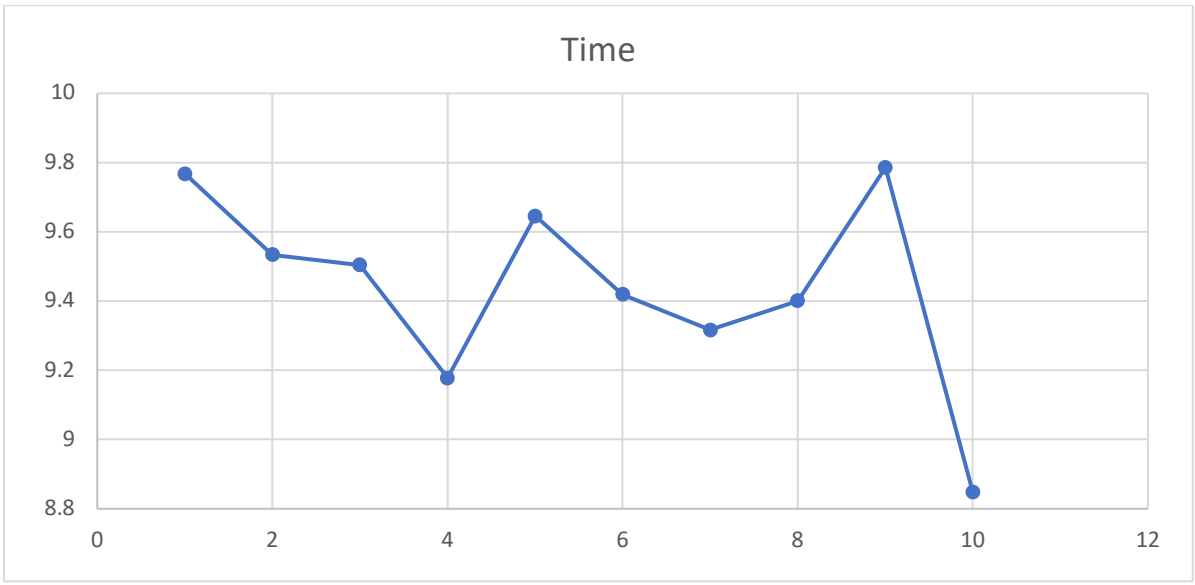
The best tour length given here is 680kms and the best tour provided is also shown in the screenshot below:



Ant Colony Optimization:

The graph here is plotted with taking 10 arbitrary iterations and their time taken. The trend of the graph shows the time constantly improving then taking a sharp turn upwards and then decreasing again. The best time and tour length is given by the last iteration of the code. The overall time taken is 43286 ms or 43.286 secs.

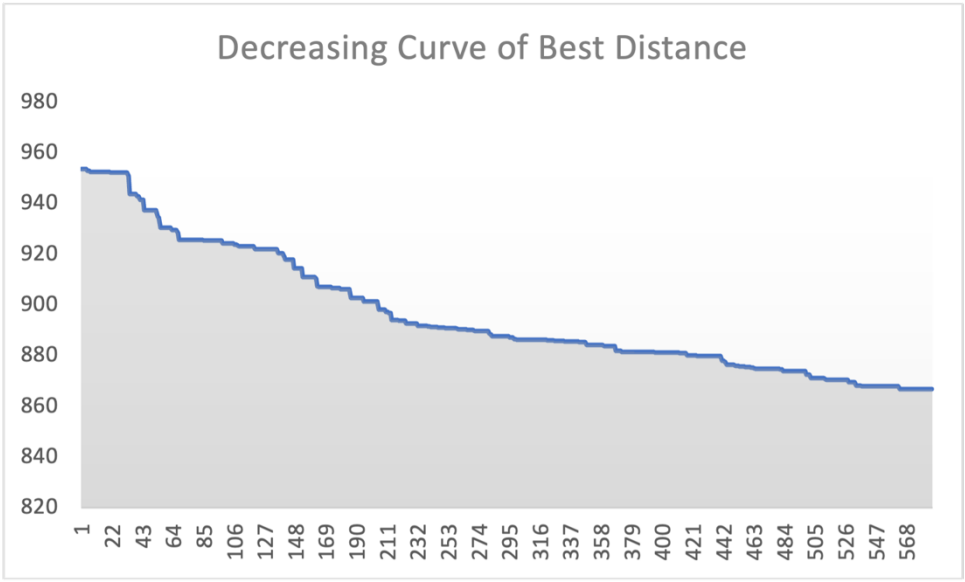
The best tour length given here is 1048kms and the best tour provided is also shown in the screenshot below:



2-Opt:

The graph here is plotted with the decreasing value of the best distance. The trend of the graph shows the distance constantly decreasing over the course of different iterations. The best distance is (866.708 kms -> 866708m) and tour length is given by the last iteration of the code. In the graph X axis has the number of iterations and notes at which iteration decrease in the best distance was observed. On the Y axis we have the best tour distance.

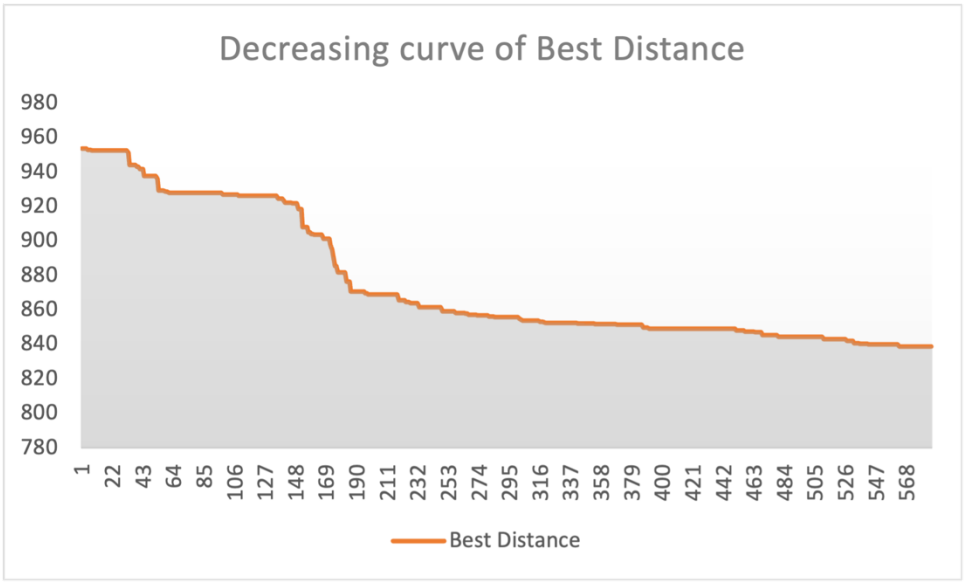
The time taken for 2OPT is 946ms, and it can be inferred from following screenshots :-



3-opt:

The graph here is plotted with the decreasing value of the best distance. The trend of the graph shows the distance constantly decreasing over the course of different iterations. The best distance is (838.890kms -> 838890m) and tour length is given by the last iteration of the code. In the graph X axis has the number of iterations and notes at which iteration decrease in the best distance was observed. On the Y axis we have the best tour distance.

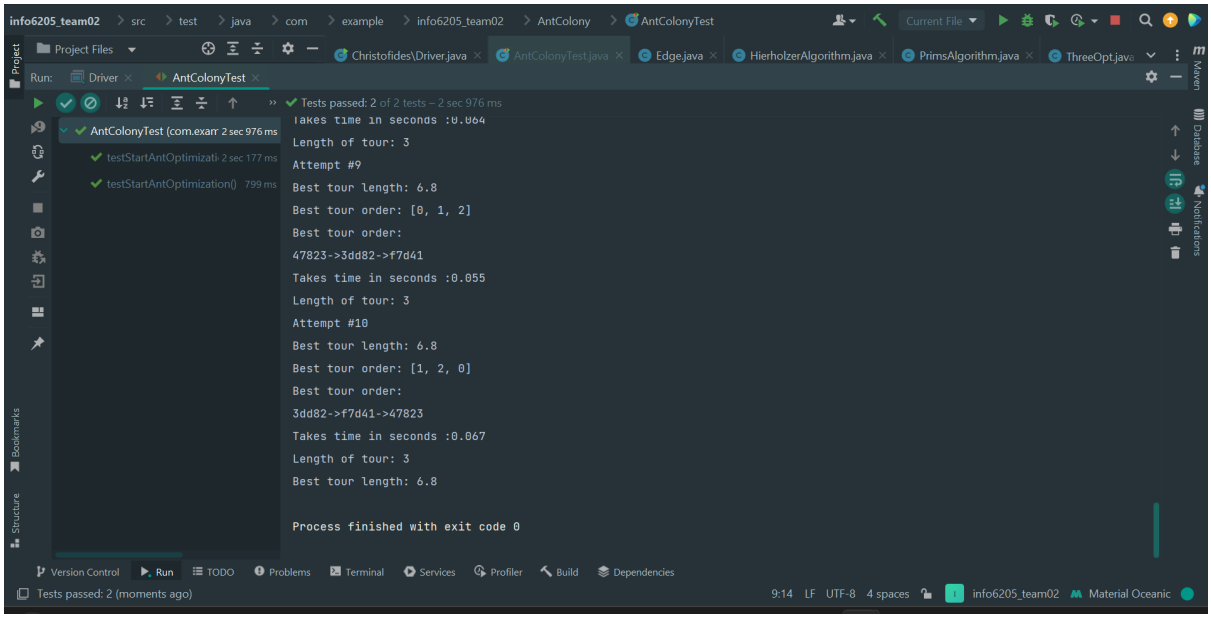
The time taken for 3OPT is 247310 ms, and it can be inferred from following screenshots :-



Unit Tests

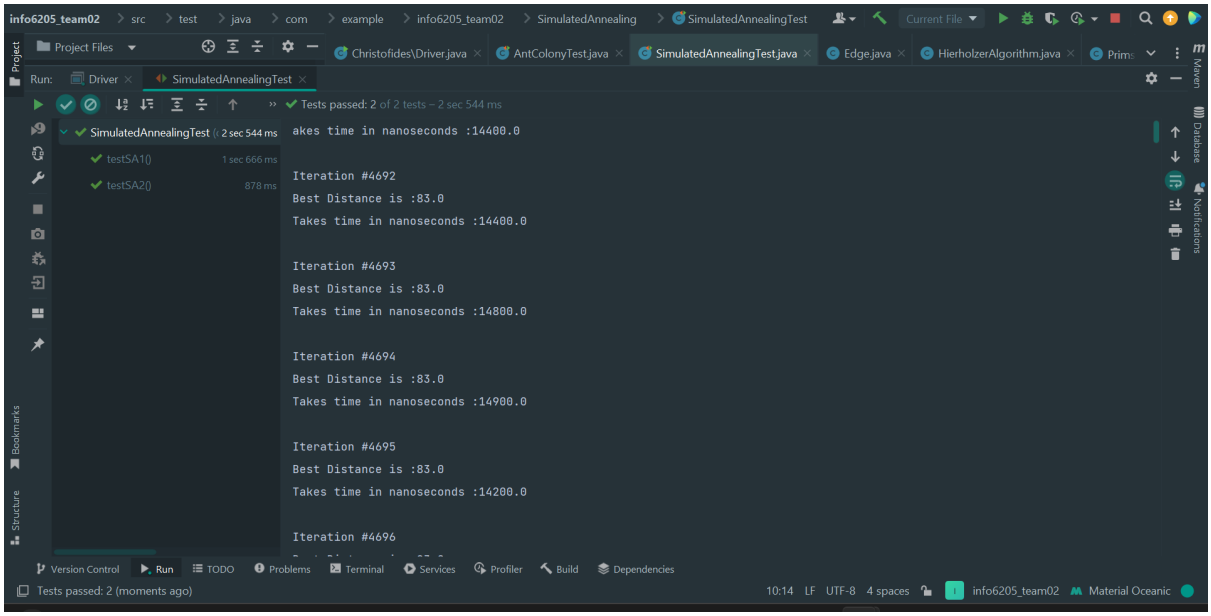
Ant colony unit test

In ant colony tests, we have created 2 scenarios having 3 and 5 cities repectively. And then have invoked the ant colony algorithm methods on them.



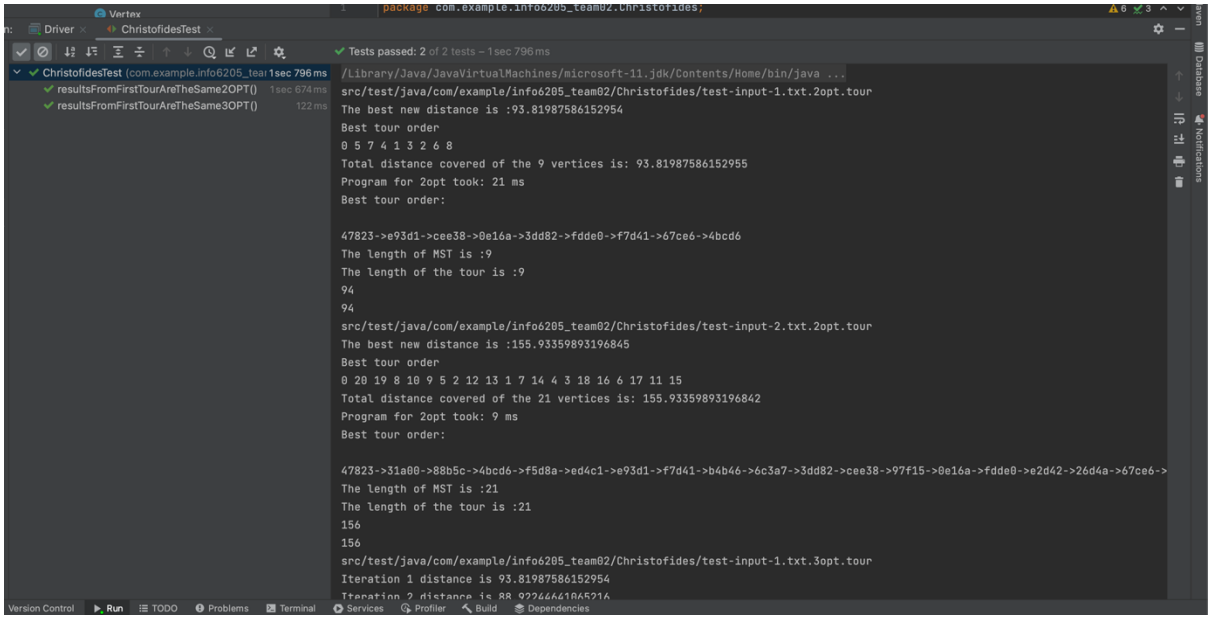
Simulated Annealing test

In Simulated Annealing we have 2 test csv files, and we parse graphs on the trial dataset of same format with 10,20 points. Once the graph is parsed, we invoke methods from simulated annealing to check the working.



2/3-opt test

In this we have 2 input files test-input-1.txt and test-input-2.txt in the same format as the dataset. We also have the output in form of files expected for 2opt and 3opt for each of these files. In the test we parse the graph from the input files, invoke methods from christofides and associated optimized methods and compare and check the input and output files.



Conclusion

- Usual time complexities of the algorithms,
  2Opt – O(n^2)
  3Opt – O(n^3)
  Simulated Annealing – O(n^2)
  Ant Colony – O(n^2 * m), where m is the ant factor or total number of ants.

- With respect to time following is the performance of all algorithms

Algorithm Name	Time in ms(MiliSeconds)
2Opt	946
3Opt	247310
Simulated Annealing	0.209417
Ant Colony	43286

We can see our algorithmic timings as per the Big(O) notation as follows,
Simulated Annealing < 2Opt < Ant Colony < 3Opt.

Here, Simulated annealing takes the least time and 3-Opt taking the most time to complete its execution. Hence, we can conclude for our algorithm, simulated annealing is best Algorithm regarding the time performance.

- The distances of the tours are as follows for all algorithms.

Algorithm Name	Distance in Km	Distance in meters
2Opt	866.780	866780
3Opt	838.890	838890
Simulated Annealing	680	680000
Ant Colony	1048.08	1048080

We can see for our algorithms following is the order of performance for distances, Simulated Annealing < 3Opt < 2Opt < Ant Colony.

We observe that 2Opt is better than 3Opt in case of time performance, whereas 3Opt is more efficient than 2Opt in case of distance performance.

However, Simulated Annealing Algorithm gives the shortest tour distance along with best time performance. Hence, we can conclude that Simulated Annealing is best algorithm for the given dataset for Travelling Salesman Problem

References

1. https://www.baeldung.com/java-ant-colony-optimization
2. https://gist.github.com/ljvmiranda921/ca93059bc213531fd99af22955b6bf5f
3. https://www.youtube.com/playlist?list=PLSM8fkP9ppPockBq1_GEG5umsz6qZb7-o
4. https://github.com/originalname51/Christofides-Algorithm
5. https://github.com/theyusko/tsp-heuristics/tree/master/algo
6. https://www.cs.nmsu.edu/~dcook/thesis/paper2.html
7. https://cse442-17f.github.io/Traveling-Salesman-Algorithms/
8. https://www.researchgate.net/publication/229476110_Accelerating_2-opt_and_3-opt_Local_Search_Using_GPU_in_the_Travelling_Salesman_Problem
9. https://baobabsoluciones.es/en/blog/2020/10/01/travelling-salesman-problem-methods/
10. https://www.ijirt.org/master/publishedpaper/IJIRT101672_PAPER.pdf
11. https://bochang.me/blog/posts/tsp/
12. https://strikingloo.github.io/ant-colony-optimization-tsp
13. https://blogs.oracle.com/javamagazine/post/how-to-solve-the-classic-traveling-salesman-problem-in-java
14. [2203.02201] Neural Simulated Annealing (arxiv.org)
15. How To Solve Travelling Salesman Problem With Simulated Annealing | by Egor Howell | Towards Data Science
16. Flood, Merrill M. “The Traveling-Salesman Problem.” Operations Research, vol. 4, no. 1, 1956, pp. 61–75. JSTOR, www.jstor.org/stable/167517.
17. Bellmore, M., and G. L. Nemhauser. “The Traveling Salesman Problem: A Survey.” Operations Research, vol. 16, no. 3, 1968, pp. 538–558. JSTOR, www.jstor.org/stable/168581.
18. Arora, S., 1996. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In: 37th Annual Symposium on Foundations of Computer Science (Burlington, VT, 1996), IEEE Comput. Soc. Press, Los Alamitos, CA, pp. 2– 11.

19. Arora, S., 1998. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. J. ACM 45(5), pp. 753–782