Performance Analysis of Genetic Algorithm and Ant Colony Optimization on a given Symmetric Travelling Salesman Problem

Rishabh Narayan

Graduate Student-M.Eng in ECE

Department of Electronics and Computer Engineering,

University of Waterloo, 200 University Ave W, Waterloo, ON N2L 3G1

Abstract -

This paper describes an implementation of the Ant Colony Optimization (ACO) and genetic algorithm for a given dataset of a symmetric travelling salesman problem. These two optimization algorithms are then compared on the metrics of their performance, such as time taken to achieve the shortest distance and the actual shortest distance achieved for the travelling salesman problem. The Traveling Salesman Problem is a NP-hard problem which implies that the problem's solution is exponential. The use of probabilistic algorithms such as the ones implemented in this paper, i.e., genetic algorithms and ant colony optimization (ACO), prove to be extremely efficient in improving the solution.

The genetic algorithm and ant colony optimization are implemented in a Python environment and on a symmetric travelling salesman problem dataset. The results of this implementation conclude that the ant colony optimization method achieves a drastically better performance than the genetic algorithm. The ACO solution is further fine-tuned by varying its parameters to achieve the best optimum solution.

Index Terms – Travelling salesman problem, genetic algorithm, Ant Colony Optimization (ACO)

I. INTRODUCTION

The travelling salesman problem is a popular and well-known NP hard, i.e., non-deterministic polynomial time hard problem. This particular problem has no exact solution and hence, the problem is solved by not achieving an exact solution but instead by trying to achieve the closest optimum solution [1].

In this paper a comparison is conducted between the performance of heuristic algorithms of genetic algorithm and ant colony optimization. There are multiple heuristic algorithms devised for this problem which achieve better and more time efficient results as compared to the two algorithms we are implementing [2]. This paper focuses on using the heuristic algorithms which fall under the umbrella of the field of artificial life. Genetic algorithm is designed basis evolutionary computation, while the ACO is a form of swarm intelligence method which is inspired by biological swarm agents, i.e., ants. Hence, a comparative study is required to help establish the better performing artificial life algorithm. The comparison conducted is aimed at finding the algorithm which finds the shortest distance and does so efficiently within optimum time.

II. THEORETICAL BACKGROUND

A. TRAVELLING SALESMAN PROBLEM

The travelling salesman problem is essentially an optimization problem. Having no single solution available for it, it is instead resolved using heuristic algorithms. This problem is categorized as NP hard, i.e., non-deterministic polynomial time hard problem. This implies that the problem's solution is exponential in nature. This problem is one of the most popular and well researched problem in the field of computer science.

The problem states that, for a given list of cities or locations and the respective distances between them, a salesman having to travel to these cities must do so by finding the shortest path possible for visiting all the cities and returning to the initial starting point. The problem is a rather simple problem on face value, however upon doing so for a large list of cities, the computational time for finding the shortest path increases exponentially [1].

The aforementioned problem can be characterized by the following [3],

a. There are n points (cities)

b. For given points i and j, point i to point j is connected by edge E (i, j) with weight D (i, j)

c. The salesman begins and ends his travel at the same point.

d. All points must be visited once by the salesman

e. There are n! (factorial) combinations of solutions possible for this problem, out of which the most optimum shortest path is selected as the final approximate solution.

The travelling salesman problem has a multitude of applications ranging from microchip manufacturing, vehicle routing, network routing, logistic planning, etc. [4]

B. Genetic Algorithm

Genetic algorithms are heuristic search algorithms, they are devised using the principles of Darwinian evolution and natural selection. They are hence dependent on the continuous simulation of the most fit individual over multiple generations of evolution.

Sangwan and Shabnam in their paper state that, "genetic algorithms transfer evolution principles in living organisms into intelligent searching and model optimization in other fields." [4]

A genetic algorithm traditionally follows the following steps:

1. The algorithm first generates a population. This is essentially the initial set of chromosomes, which is referred to as the population.

2. The algorithm next generates a new generation of the population. For every such generation, the individuals who have a better fitness function survive to the next generation. The process of generating individuals for the next generation is performed by using crossover and mutation within the existing population.

3. For every generation, the genetic algorithm generates certain fitness function, the value of the best fitness function determines if the algorithm stops or not. If the value of the function does not improve over successive generations, the algorithm stops. If the fitness function improves instead, then the algorithm repeats the step 2. [5]

In this paper, the individuals, i.e., the chromosomes are the co-ordinates of the cities, the fitness function used to evaluate the fitness of the individuals is the distance between the cities. Then subsequently after selection, crossover and mutation is performed.

C. Ant Colony Optimization (ACO)

The ACO algorithm falls under swarm intelligence, which is the collaboration between virtual/artificial agents to solve a problem or perform a task. This behaviour is inspired by natural organisms which display swarm behaviour for fulfilling their diurnal tasks. Ants, bees and termites are a good example of swarm behaviour in nature.

The famous entomologist, Ernest Andre introduced the interest in the study of ants as artificial agents. The objective of studying the ant behaviour was to understand how ants despite being blind are capable of foraging for food away from the colony and further capable of finding the shortest distance from the colony to the food site. This was explained by the phenomenon of stigmergy [6], by which ants are capable of indirect communication with one another via the means of pheromones. Stigmergy is essentially the mechanism swarm agents use to communicate with one another by making changes in their environment and subsequently responding to it.

The ACO is solely based on this mechanism of ant colonies to forage for food. When foraging for food, ants move out of their colonies and search for food in a random manner. All while moving, the ants keep secreting pheromones on their paths. Upon finding a food source, the ant first decides the quality of food and consequently varies the pheromone concentration as per the quality of the food. Hence, for every such trip where an ant finds food it evaluates the food quality and releases either high or low concentrations of pheromones. This trail of pheromones which would have increased in concentration if the quality of food was assessed as ideal, would then guide the other ants to the same food source. In this manner, over multiple iterations or such cycles of food foraging, the pheromone concentration on the path to the best quality of food would be high enough to be considered as the shortest path from the ant colony to that food source. [7]

In the ACO algorithm, artificial agents represent the ants which are then simulated in a similar fashion as the ants forage for food and are also provided with pheromones. This swarm intelligent system is then used for searching for the shortest path in our travelling salesman problem.

III. HARDWARE AND SOFTWARE CONFIGURATIONS

For this experiment, a PC with AMD Ryzen 7 5800H @3.20 GHz and 8GB RAM were used to implement the algorithms on our given TSP dataset.

The code for the same was run in a Python 3 environment using the PyCharm IDE.

IV. DATASET USED

The Oliver30 dataset is a common travelling salesman problem dataset. The original source of this dataset is given in the paper by Oliver et al. [8] However, the consolidated dataset is also available online [9].

This dataset was stored in a csv file and subsequently used for our simulations.

V. METHODOLOGY

A) Genetic Algorithm for TSP

The genetic algorithm implemented is tested on the Olive30 dataset. The dataset which is in a csv format consists of the location coordinates of 30 cities.

In the program for the genetic algorithm, this dataset is imported and the location coordinates of the cities are stored as tuples within a list. Hence, for our genetic algorithm implementation, each chromosome here represents a list of tuples containing the city coordinates, which in turn represent the shortest path taken by the salesman. As the list of tuples is considered to be the chromosomes, the tuples themselves represent the genes in this algorithm. The collective storage of these represents the population of our given generation.

In our program first and foremost a class is created to define, control and modify our genetic algorithm. Upon instantiating the class object our variables get initialized. The list of cities obtained from Olive30 dataset is then used for generating the starting population. For each population generated, the cities are shuffled and then stored in a new population list. For this, the permutation encoding is done as we require our data, i.e., the cities to be ordered.

Start is called to begin the program. The core loop of the program involves calculating the fitness of the current population. It has to check if the shortest path obtained for each population is in fact the best shortest path available.

For this fitness function, the distance between the cities is calculated. This is calculated using the Pythagoras theorem. The distances between the cities which are calculated are then summed to give our total path length for the given population which the salesman traverses. This generated path is then compared with the all-time best path from the previous generations.

Following the calculation of the fitness values, a new population is then generated. This new population is initially set to an empty list. Now two parents are called using the select function in the program. The select function finds the maximum fitness values of the members from the current population. Once these two parents are selected, the crossover function is implemented for generating a new child member. This crossover function operates by selecting a random integer, n, between 0 and the length of the cities. The first n elements from the first parent are added to the child member which defines the start of the path. The remaining path elements are then inserted from the second parent in the same order as they appear and all while ignoring any sets of paths from the first parent. This helps ensure that the child generated has the correct number of unique cities in order, which has taken characteristics from the two parents.

Next the mutate function is iterated over the child member and for each city in the list a random number is generated. If this random number is found to be lower than the given mutation rate, the current city is swapped with another randomly selected city. Upon completion of mutation, we now have a new population list which is then stored as the current population list. This process in our program runs until the generation limit has been reached.

The visualization for the program is performed using the PyGame modules in python. The code for

this portion of the program is taken from a github repository [10].

The aforementioned flow of our program for solving a travelling salesman problem is shown in figure-1.

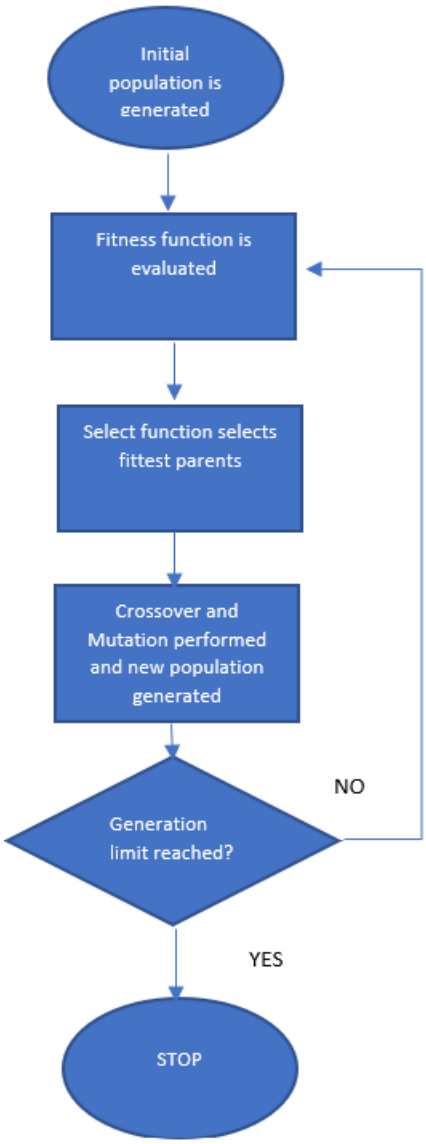

Fig. 1 – Genetic Algorithm for TSP - Flowchart

B) Ant Colony Optimization (ACO) for TSP

The Ant Colony Optimization (ACO) algorithm is implemented and tested on the Olive30 dataset. The dataset which is in a csv format consists of the location coordinates of 30 cities.

For the travelling salesman problem, the heuristic taken by the ACO is essentially the distance between the cities. The ants would pick the distance which proves to be the shortest.

The code for this was obtained from a github repository [11] and the original structure of the ACO algorithm was not further modified. Parts of unrequired code were dropped and changes were made to the manner of reading in input data and the manner of running the driver program file. Further, certain aesthetic changes were made for the output plot. A separate driver program file is used to run the ACO program.

In the code, the data of the cities is loaded and stored in a list of tuples. This list represents the nodes in our ACO algorithm. In our program a class is created to define, control and modify our genetic algorithm. Upon instantiating the class object our variables get initialized. Within our main class we create a subclass for defining the behaviour of the artificial ants. Next, we need to calculate the probability of selecting a solution, i.e., finding a node. The “choose node” function in conjunction with the “tour finder” function, implement a roulette-wheel type of selection process for selecting. An important point to note here is that at each step of finding such a solution, a constant amount of pheromones is deposited by the ants. This is governed by the adding pheromone function in the program.

During this running period the distances being travelled are calculated at each step. The optimal path obtained from this, i.e., the shortest path being taken is the only path for which we update the pheromone concentration. If all the ants take the same best path then we achieve our optimal path and the program ends, however if convergence is not reached, the program begins again from the first phase of initializing the ants to search for the cities again.

The final optimum path is further plotted and the time efficiency of achieving the path is recorded for analysis. One of the most interesting aspects of the ACO in comparison with the genetic algorithm is the availability of multiple customizable parameters such as pheromone exponential weight (α), heuristic exponential weight (β), etc. Varying combinations of these parameters are implemented depending on the nature of the problem at hand. In this paper, these parameters are varied and then

compared to see as to which combination yields the most accurate result.

V. RESULTS AND ANALYSIS

In this study, two metaheuristic algorithms, i.e., genetic algorithm and ACO were implemented for the TSP problem. The further objective is to conduct a comparison between the two which is aimed at finding the algorithm which finds the shortest distance efficiently within optimum time.

A) Genetic Algorithm Results

The genetic algorithm in this study, performed quite poorly on the given Oliver30 TSP dataset. The Oliver30 TSP has a benchmark solution of the shortest distance being 423.741 [9], while the genetic algorithm was able to achieve a shortest distance of 800.516 and that too with an extremely high computational time of 7580.52 seconds.

The results of the genetic algorithm are shown in table 1 and fig 2 displays the graphical output of the shortest path achieved by the genetic algorithm for the Oliver30 TSP dataset.

Algorithm	Dataset	Cost	Optimum Cost	Time(s)
Genetic Algorithm	Oliver30	800.516	423.741	7580.52

Table 1 – Genetic Algorithm Results

Fig 2 – Graphical output for genetic algorithm solution

B) Ant Colony Optimization (ACO) Results

The ant colony optimization algorithm yielded satisfactory results when implemented on the Oliver30 dataset.

The ACO algorithm was further fine-tuned by varying its customizable parameters of exponential weight (α), heuristic exponential weight (β), pheromone evaporation rate (ρ) and the colony size. The parameters for exponential weight (α), heuristic exponential weight (β) and pheromone evaporation rate (ρ) are taken from the parametric optimization studies conducted by Shrivastava et al. and Simpson et al. [12][13]. Further the colony size is varied to help achieve the most optimum solution.

The results of the ACO algorithm implementation on Oliver30 dataset with variation in α, β and ρ are shown in table 2.

Optimum Cost	Cost	Time(s)	α	β	ρ
423.741	424.635	9.76	1	3	0.3
423.741	431.267	10.49	3	3	0.8

Table 2 - ACO results with variation in α, β and ρ

The graphical output for the ACO algorithm with the variations in α, β and ρ are shown in Fig-3 and Fig-4.

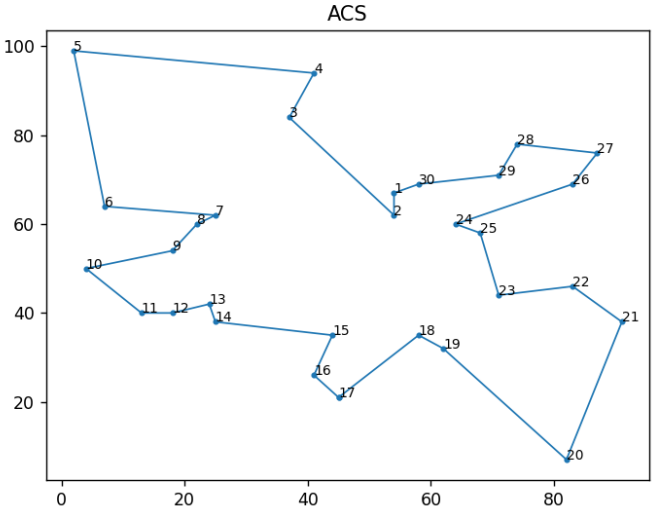

Fig 3 – Graphical output for ACO (α = 1, β = 3 and ρ = 0.3)

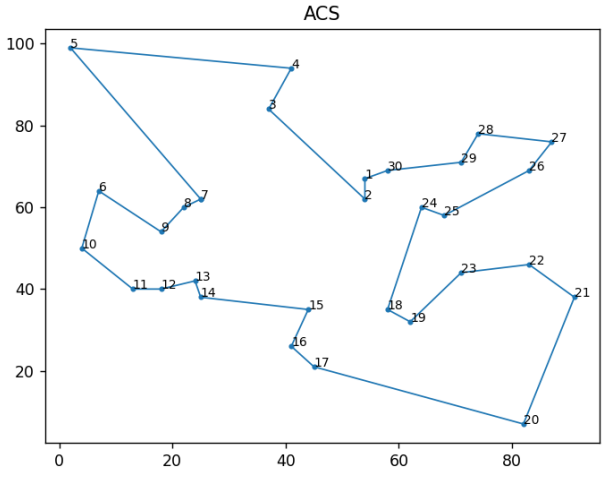

Fig 4 - Graphical output for ACO (α = 3, β = 3 and ρ = 0.8)

We find that for the optimized parameters of α = 1, β = 3 and ρ = 0.3 provided by Shrivastava et al. [12], the shortest path (cost) of 424.635 is almost the same as that of the optimum cost. The computational time taken however is relatively high, but it can be lowered by further optimizing the ant colony size. It was important to first determine the optimum α, β and ρ parameters since they are the major parameters influencing the algorithm [14]. Now the ant colony size is reduced to reduce the computational time. However, reducing the colony size by a large margin will result in a poor optimum solution. Therefore, it is essential to find a good trade off between the computational time and the optimum cost. The results of the ACO algorithm implementation with variation in ant colony size are shown in table 3 (it is important to note that the previously determined optimum α, β and ρ parameters are used)

Optimum Cost	Cost	Time(s)	Ant Colony Size
423.741	426.600	0.97	50
423.741	424.635	4.93	250
423.741	424.635	9.76	500

Table 3 - ACO results with variation in ant colony size

As seen from this table, for low values of ant colony size, the computational time is extremely fast and we achieve an optimum cost of 426.6 within 0.97s with the colony size at 50. Increasing the colony size to 500 provides us with our approximate optimum solution of 424.635 but it takes 9.76s which is computationally expensive. However, by setting the size at 250 gives us the same optimum path as the 500 sized colony but within a relatively shorter time of 4.93s. The larger the colony size the more the number of ants who work towards finding the shortest path, so it is expected to get better results for a higher value of colony size. This observation confirms the theory proposed by Ivan Brezina Jr. and Zuzana Čičková [15]. However, at the same time it is also important to prioritise computation time. Hence, we conclude that for the ACO algorithm, we get the most optimum result when the parameters are as follows: α = 1, β = 3, ρ = 0.3, colony size = 250).

The graphical output for the optimized ACO (α = 1, β = 3, ρ = 0.3 and colony size = 250) is shown in fig 5.

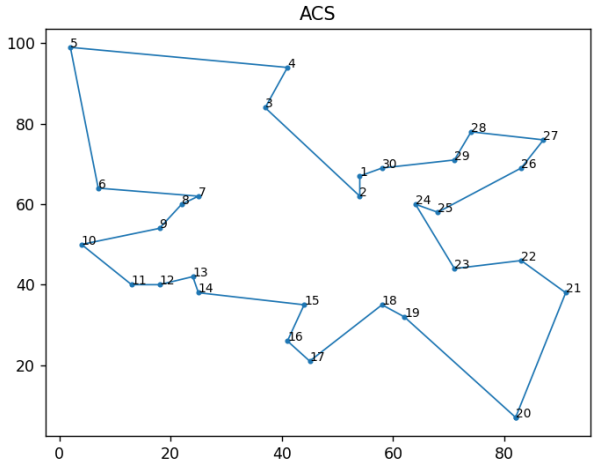

Fig 5 - Graphical output for the optimized ACO (α = 1, β = 3, ρ = 0.3 and colony size = 250)

C) Comparison between Genetic Algorithm and ACO

The two algorithms implemented on the Oliver30 dataset performed drastically differently. The ACO algorithm yielded us with a satisfactory optimum path within an acceptable range of computational time. The genetic algorithm on the other hand was unable to achieve a satisfactory optimum cost value and had an incredibly high computational time. The results of these two algorithms are tabulated in table 4.

Algorithm	Time(s)	Cost	Optimum Cost
ACO	4.93	424.635	423.741
Genetic Algorithm	7580.52	800.516	423.741

Table 4 – Comparison of results between genetic algorithm and ACO

The genetic algorithm is traditionally known to perform much better than our observed results and in some cases is known to outperform the ACO algorithm for TSP optimization problem [16]. The genetic algorithm is found to be much faster than the ACO at processing data for cities greater in number [3]. The genetic algorithm implemented in this study failed to achieve benchmark or close to benchmark results owing to the lack of optimization in the structure of the algorithm programmed. The genetic algorithm implemented in this study can further be optimized by implementing different crossover techniques. Various improvements on the crossover operator have been made which would significantly improve the performance of the genetic algorithm for the TSP problem. Fine tuning the crossover operator is one of such methods [17]. There is also a possibility that the mannerism of coding the genetic algorithm has unnecessarily increased our computation time and further debugging of the code could help improve the performance of the genetic algorithm.

The ACO algorithm on the other hand, while giving satisfactory results can also be improved to yield a better optimum path solution. The ACO having multiple variable factors, makes it tricky at times to get the exact right combination of parameters for a given problem.

VI. CONCLUSION

This study's objective was to implement a genetic algorithm and ant colony optimization (ACO) algorithm to solve the travelling salesman problem (TSP) and to further compare their respective performances. In our study we conclude that the genetic algorithm failed to achieve a satisfactory solution for the TSP problem and had a high computational time. The ACO algorithm on the other hand provided us with an approximate solution for the TSP problem and was further optimized to reduce its computational time.

VII. REFERENCES

[1] D.L. Applegate, R.E. Bixby, V. Chvátal and W.J. Cook, The Traveling Salesman Problem: A Computational Study, Princeton: Princeton University Press, 2007

[2] Halim, A.H., Ismail, I. Combinatorial Optimization: Comparison of Heuristic Algorithms in Travelling Salesman Problem. Arch Computat Methods Eng 26, 367–380 (2019).

[3] Alexander Alexander and Haris Sriwindono, The Comparison of Genetic Algorithm and Ant Colony Optimization in Completing Travelling Salesman Problem, Proceedings of the 2nd International Conference of Science and Technology for the Internet of Things, ICSTI 2019

[4] Sangwan, Shabnam. (2018). Literature Review on Travelling Salesman Problem. International Journal of Research. 5. 1152.

[5] Juneja, Sahib & Saraswat, Pavi & Singh, Kshitij & Sharma, Jatin & Majumdar, Dr & Chowdhary, Sunil. (2019). Travelling Salesman Problem Optimization Using Genetic Algorithm. 264-268.

[6] Marco Dorigo, Eric Bonabeau, Guy Theraulaz,,Ant algorithms and stigmergy, Future Generation Computer Systems, Volume 16, Issue 8,2000,Pages 851-871

[7] Y. Pei, W. Wang and S. Zhang, "Basic Ant Colony Optimization," 2012 International Conference on Computer Science and Electronics Engineering, 2012, pp. 665-667

[8] I. M. Oliver, D. J. Smith and J. R. C. Holland, "A study of permutation crossover operators on the travelling salesman problem," in Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application, 1987, pp. 224-230

[9] https://stevedower.id.au/research/oliver-30

[10]https://github.com/sit215-high-achievers/genetic_TSP/blob/master/TSPGenetic.py

[11] https://github.com/nishnash54/TSP_ACO

[12] Shrivastava, Kush & Kumar, Dr Shishir. (2018). The Effectiveness of Parameter Tuning on Ant Colony Optimization for Solving the Travelling Salesman Problem. 78-83.

[13] Simpson, Angus & Maier, Holger & Foong, W & Seah, H & Tan, C. (2001). Selection of Parameters for Ant Colony Optimisation Applied to the Optimal Design of Water Distribution Systems.

[14] Latheef, Ghaisan & Gan, Hao Zhan & Abdul Salam, Zailan. (2020). Comparative Analysis of Ant Colony Optimization and Genetic Algorithm on Solving Symmetrical Travelling Salesman

Problem. Journal of Advanced Research in Dynamical and Control Systems. 12. 2629-2635.

[15] I.J. Brezina and Z. Čičková, "Solving the Travelling Salesman Problem Using the Ant Colony Optimization," Management Information Systems, vol. 6, no. 4, pp. 10-14, 2011.

[16] M. Alhanjouri and B. Alfarra, "Ant Colony versus Genetic Algorithm based on Travelling Salesman
Problem," *International Journal of Computer Technology and Applications*, vol. 2, no. 3, pp. 570-578,
2011.

[17] Shrestha, Ajay & Mahmood, Ausif. (2016). Improving Genetic Algorithm with Fine-Tuned Crossover and Scaled Architecture. Journal of Mathematics. 2016. 1-10.