# Optimization Algorithms on Travelling Salesman Problem

Kelly Tan Kai Ling
*School of Computer Science*
*University of Nottingham Malaysia*
hfykt10@nottingham.edu.my

Lisa Ho Yen Xin
*School of Computer Science*
*University of Nottingham Malaysia*
hfylh2@nottingham.edu.my

Zhang Kun
*School of Computer Science*
*University of Nottingham Malaysia*
hfykz1@nottingham.edu.my

***Abstract*—**This paper provides a comprehensive review of Artificial Intelligence (AI) algorithms implemented to solve the Travelling Salesman Problem (TSP) over the past nine decades. Three algorithms are chosen for a critical evaluation for 107 cities, which includes Genetic Algorithm (GA), Simulated Annealing (SA), and Ant Colony Optimization (ACO). The results of the implementation of each algorithm are presented to provide a detailed analysis.

***Keywords*—***Travelling Salesman Problem, Genetic Algorithm, Simulated Annealing, Ant Colony Optimization*

## I. Introduction

The Travelling Salesman Problem (TSP) is a combinatorial optimization problem that is NP-hard and difficult to solve. The objective is to find the shortest possible route for a salesman to travel between *n* cities, visiting each city only once before returning to the starting city. Many exact algorithms have been proposed to find the optimal solution but are inefficient for large problem instances due to increasing computational complexity. To solve this problem, heuristic and metaheuristic algorithms have been developed to find good approximate solutions but do not guarantee optimality.

## II. Literature Review

### A. Exact Algorithms

Exact algorithms guarantee to find the optimal solution but are usually only feasible for small problem instances.

The Brute Force Algorithm calculates all possible solutions and filters out the optimal solution. It is easy to implement, but has an exponential time complexity. [3]

The branch-and-bound (BB) algorithm breaks down problems into smaller sub-problems and eliminates the non-optimal solutions. Its time complexity is better than the Brute Force Algorithm, but its implementation is more complex. [3]

### B. Approximate Algorithms

Approximate algorithms are heuristic algorithms that provide suboptimal solutions to optimization problems. These algorithms trade-off optimality for computational efficiency and can be used to solve problems for which finding an exact solution is computationally intractable.

2-opt is a local search algorithm that iteratively removes and reconnects two edges to form new tours. It has a relatively low computational cost but may get stuck in local optima. [3]

Christofides' Algorithm gurantees that its TSP solution will be within 1.5 times of the optimal solution. It may be computationally expensive for large-scale problems. [8]

### C. Metaheuristic Algorithms

Metaheuristic algorithms aim to find near-optimal solutions to a wide range of problems using advanced search strategies that incorporate heuristics as building blocks.

Genetic Algorithm (GA) selects and recombines the best solutions from a given population to produce new solutions. It can find the global optimum and handle large datasets, but is computationally expensive. [4]

Simulated Annealing (SA) modifies an initial solution by making random changes and accepting them based on a probability. It can escape local optima, but may require longer computational times and may not always find optimal solutions. [2]

Ant Colony Optimization (ACO) simulates ant behavior to build and evaluate solutions. It is effective at finding good solutions and can handle complex TSP instances with multiple criteria, but the solution quality depends on the choice of parameters, which cannot guarantee finding the global optimum. [4]

Tabu Search is a local search algorithm that avoids getting stuck in local optima by keeping track of recently visited solutions. It has less computational time. [4]

### D. Factors Affecting Performance of Optimization Algorithms on TSP

The main factors that influence the performance of optimization algorithms on TSP are dataset size, algorithm parameters, convergence criteria, and search space structure. [2]

Dataset size is an important factor that affects the performance of optimization algorithms for solving the TSP. For small datasets, exact algorithms can find optimal solutions efficiently, but as the dataset size grows, heuristic algorithms become more effective. [2]

Format adapted and modified from ©2023 IEEE.

In view of choice of algorithm parameters such as initial temperature, cooling schedule, and mutation rate, fine-tuning these parameters can improve the quality of the solution and reduce the computational time. [2]

Convergence criteria, also known as stopping criteria, determine whether the algorithm can stop running. Choosing an appropriate stopping criterion can help avoid premature convergence or excessive computational time. [2]

Lastly, the structure of the TSP search space can also affect the performance of optimization algorithms. For example, TSP instances with clusters of cities may be easier to solve using heuristics that exploit such clusters. [2]

### *E. Comparative Analysis of Optimization Algorithms on TSP for Various Dataset Sizes*

In 2017, Alhanjouri and Mohammed had proposed that for small datasets (less than 50 cities), the BB algorithm showed the best performance in terms of finding the optimal solution. The Dynamic Programming and SA algorithms also performed well, but they were not as efficient as the BB algorithm. [2]

For medium-sized datasets (between 50 and 200 cities), SA was found to be the most efficient in terms of both finding the optimal solution and computational time. ACO also showed good performance. [2]

For large datasets (more than 200 cities), GA was found to be the most efficient in terms of finding the optimal solution, followed by the ACO algorithm. However, the computational time for both algorithms was longer than the other algorithms studied in the paper. [2]

### *F. Overview of Optimization Algorithms on TSP*

Exact algorithms provide optimal solutions but can be computationally expensive for large problems. Heuristic algorithms are faster than exact algorithms but provide suboptimal solutions. Metaheuristic algorithms provide good solutions by using higher-level strategies to explore the search space more efficiently. [4]

Metaheuristic algorithms are considered the best approach for solving TSP because they can handle large-scale problems with reasonable computational effort and provide high-quality solutions. [4]

According to a comparison between metaheuristic algorithms by Sathya and Muthukumaravel, GA is known for its efficiency in handling large-scale problems while ACO is known for its accuracy in finding optimal solutions. [4] SA provides a better solution quality compared to GA, but it may get stuck in local optima. [5]

### *G. Considerations for Optimization Algorithms Selection on TSP*

When selecting an optimization algorithm for a TSP problem, it is important to consider factors such as the size and complexity of the problem, the required solution quality, and the available computational resources.

For example, if the problem consists of 107 cities, which is medium-sized, heuristic algorithms such as GA, SA, ACO or Tabu Search may be more appropriate. Although both GA and Tabu Search require memory to store fitness values or the Tabu List, GA has the advantage of being able to handle a wider range of problem types. Additionally, GA can be customized for a specific problem by tuning the selection, crossover, and mutation parameters. [2] In contrast, Tabu Search may require more memory and computational resources to maintain and update the Tabu List. [1] Therefore, GA, SA, and ACO are the best optimization algorithms to solve TSP of 107 cities.

## III. METHDOLOGY

This section presents three different optimization algorithms that will be used to solve TSP. The stopping criteria for each algorithm is set to 30 iterations. During each iteration, a new solution is generated and the best solution found up to that point is updated.

### *A. Genetic Algorithm (GA)*

GA is a metaheuristic algorithm that simulates the process of natural selection, based on the concept of “survival of the fittest”. Every generation consists of a population of solutions which are analogous to the chromosome. Each solution is evaluated according to a fitness score, where the fittest solutions will advance to the next generation to produce a more optimal solution. [8]

In the initialization phase, an initial population of random individuals is generated, and each individual is treated as a chromosome. The chromosomes are to undergo a **selection** process where chromosomes with greater fitness value are selected. Based on the crossover probability, the selected chromosomes are to perform a **crossover** on random crossover sites. The offspring of the chromosomes are to perform a **mutation** on random mutation sites based on the mutation probability.

The parameters that control the outcome of the algorithm include the *population size*, *tournament size*, *mutation rate*, and *elitism count*. The total distance of route is taken as the fitness value, while the stopping criteria is the *maximum generation*.

The *population size* parameter determines the number of individuals in the population while the *tournament size* parameter determines the number of

individuals to be selected in the selection phase. Larger population and tournament sizes improve the solution quality but increases computational time.

The *mutation rate* parameter determines the probability of mutation, which is the process of introducing new genetic information into a population. A small mutation rate may get stuck in local optima while a large mutation rate may destroy good solutions.

The *elitism count* parameter determines the number of best individuals to be carried over unchanged from a generation to another, which ensures the survival of best genes across generations to avoid getting stuck in local optima.

The *maximum generation* parameter represents the number of repetitions the algorithm should undergo and determines the algorithm's time of termination. A higher number of repetitions improves solution quality but increases computational time.

The pseudocode of the GA implemented is given in Fig. 1.

```
1.  Procedure Genetic Algorithm
2.  Begin
3.    Initialize population P with random
    chromosomes
4.    Evaluate fitness of each chromosome in P
5.    best_chromosome = chromosome with best
    fitness in P

6.    While (max generations not met)
7.      new_population = empty list

8.      For i = 1 to population_size:
9.        parent1 = tournament selection from P
10.       parent2 = tournament selection from P
11.       child = crossover(parent1, parent2)
12.       mutate(child)
13.       Add child to new_population

14.     Set worst elitism_count chromosomes in
    new_population = best_chromosome
15.     Evaluate fitness of each chromosome in
    new_population
16.     best_chromosome = chromosome with best
    fitness in new_population

17.     P = new_population
18.     Update number of generations
19.   End While

20.   Return best_chromosome
21. End
```

Figure 1. Pseudocode of GA

### *B. Simulated Annealing (SA)*

SA is a metaheuristic algorithm that simulates the process of annealing in metallurgy. Similar to the process of heating and cooling of metal during annealing, the algorithm starts with an initial temperature which is gradually decreased at each iteration until it reaches the minimum temperature. [2]

```
1.  Procedure Simulated Annealing
2.  Begin
3.    Initialize current_route = route with all cities
4.    Set initial_temperature, cooling_rate
5.    Set maximum number of iterations

6.    While (maximum iteration not met)
7.      Generate random number (i, j) between 1
    and total_cities - 1
8.      Clone current_route to next_route
9.      Swap cities i and j in next_route
10.     delta = distance of next_route - distance of
    current_route

11.     If (delta < 0):
12.       Set current_route to next_route
13.       Update temperature based on cooling_rate
14.   Update number of iterations
15.   End While

16.   Return current_route
17. End
```

Figure 2. Pseudocode Of SA

The process of SA is described as below:

The process starts with an initial route and random swaps of cities. It calculates the difference in distance between the new and current routes, accepts the new route if it is better, and sometimes accepts a worse route based on the temperature and cooling rate. The process repeats until the stopping criteria is met.

The parameters that control the outcome of the algorithm include the *initial temperature* and *cooling rate*. The stopping criteria is the number of repetitions.

The *initial temperature* parameter determines the starting temperature, where a high temperature increases the randomness of the search while a low temperature limits the search space.

The *cooling rate* parameter determines the rate at which the temperature decreases and determines the rate of solution convergence. A higher cooling rate results in faster convergence to the optimal solution but increases the risk of getting trapped in local optima. Conversely, a lower cooling rate allows the algorithm to explore the search space more thoroughly. [8]

### C. Ant Colony Optimization (ACO)

ACO is a metaheuristic algorithm that simulates the behaviour of ant colonies. It uses artificial ants that move through a graph of nodes and leave pheromone trails. The amount of pheromone deposited is based on the quality of the solution. This trail guides future ants to follow the best paths in the graph. ACO keeps constructing solutions and updating the pheromone trails until a stopping criterion is reached. [8]

```
1.  Procedure Ant Colony
2.  Begin
3.    Initialize pheromone levels for each edge
4.    Set number of ants, α, β, evaporation rate
5.    Set maximum number of iterations

6.    While (maximum iteration not met)
7.      For each ant:
8.        Choose a starting city at random
9.        While there are unvisited cities:
10.         Compute probabilities for every next
      city based on pheromone levels
11.       Use roulette wheel to select next city
12.       Move to the selected city
13.       Update the pheromone level on the
      edge just travelled
14.     End While
15.     Update the best_route

16.   Evaporate pheromone levels on all edges
17.   Update the pheromone levels on edges
      according to the ants' tours
18.   Update number of iterations
19.   End While
20.   Return the best_route
21. End
```

Figure 3. Pseudocode Of ACO

The process of ACO is described as below:

In the initialisation phase, parameters are initialised, and ants are placed randomly at different starting points. The path length of each ant is calculated, and the optimal solution is recorded. After updating the pheromone, the previous steps are repeated until the stopping criteria is met.

The parameters that control the outcome of the algorithm include the *number of ants*, pheromone ***α***, heuristic function operator ***β***, and *evaporation rate*. The stopping criteria is the number of repetitions.

The *number of ants* parameter determines the number of artificial ants used to build the solution. A higher number of ants improves the exploration space but increases computational time.

The parameter ***α*** represents the weight given to the pheromone trail in the decision making of the ants, and parameter ***β*** represents the weight given to the heuristic information. A higher value of ***α*** makes the ants more likely to choose paths with higher pheromone concentration, and a higher value of ***β*** makes the ants more likely to choose shorter paths based on the distance between the cities.

The *evaporation rate* parameter determines the rate at which the pheromone evaporates from the edges of the graph. A higher value of evaporation rate causes the pheromone to evaporate faster, leading to a faster solution convergence rate.

## IV. Result and Discussion

The following analysis is conducted using IntelliJ IDEA 2022.3 on a computer with Intel Core i5-3550S 3.0 GHz processor to simulate the problem of TSP.

The values of the parameters for each algorithm are tuned and chosen based on trial and error in accordance with the characteristics of the problem being solved and the problem space.

### A. Genetic Algorithm (GA)

The simulation has a population size of 107 individuals, a mutation rate of 0.001, a tournament size of 5, an elitism count of 10, and a maximum generation of 100.

TABLE I. Experimental Results of GA

| Run | Solution | Time | Run | Solution | Time |
|---|---|---|---|---|---|
| 1 | 571996.65407 | 0.098 | 16 | 610595.89032 | 0.083 |
| 2 | 563561.55964 | 0.110 | 17 | 581191.19233 | 0.076 |
| 3 | 602502.19348 | 0.107 | 18 | 555966.31647 | 0.072 |
| 4 | 611061.49069 | 0.114 | 19 | 616221.03061 | 0.071 |
| 5 | 551798.94622 | 0.107 | 20 | 574914.53924 | 0.078 |
| 6 | 612153.32986 | 0.091 | 21 | 555922.38854 | 0.120 |
| 7 | 562057.46198 | 0.080 | 22 | 591289.19028 | 0.115 |
| 8 | 596044.67829 | 0.068 | 23 | 584318.98042 | 0.104 |
| 9 | 583872.59343 | 0.071 | 24 | 626097.09968 | 0.129 |
| 10 | 597711.67196 | 0.078 | 25 | 556357.47598 | 0.103 |
| 11 | 559153.46496 | 0.087 | 26 | 555739.74771 | 0.111 |
| 12 | 565156.52116 | 0.088 | 27 | 516710.98992 | 0.167 |
| **13** | **494630.31612** | **0.088** | 28 | 568177.13633 | 0.115 |
| 14 | 565872.39209 | 0.079 | 29 | 514071.09787 | 0.113 |
| 15 | 585816.06243 | 0.076 | 30 | 605193.07549 | 0.11 |

Table I shows the results for GA. The shortest distance found is 494,630.31612, which is the best solution and takes 0.088 seconds. The longest distance found is 626,097.09968, which is the worst solution and takes 0.129 seconds. Among the 30 runs, the average distance for the final route is 573,710.769602, and the average time taken is 0.095 seconds. However, the large gap between the shortest and longest distances

found suggests that the algorithm may not be stable and may not be suitable for the given medium-sized dataset.

### *B. Simulated Annealing (SA)*

In simulation, the initial temperature is set to 1000, and the cooling rate is set to 0.0000005 for a dataset of 107 cities.

TABLE II. EXPERIMENTAL RESULTS OF SA

| Run | Solution | Time | Run | Solution | Time |
|---|---|---|---|---|---|
| 1 | 49853.026178 | 40.585 | 16 | 44564.303608 | 39.760 |
| 2 | 47090.886815 | 40.143 | 17 | 44705.166437 | 40.238 |
| 3 | 49890.532426 | 41.584 | 18 | 46585.052680 | 37.103 |
| 4 | 44813.502663 | 39.931 | 19 | 49963.250471 | 35.906 |
| 5 | 49524.070048 | 39.875 | 20 | 44720.747098 | 36.033 |
| 6 | 49733.773348 | 39.844 | **21** | **44337.363848** | **36.633** |
| 7 | 48279.057201 | 40.052 | 22 | 44857.402874 | 40.530 |
| 8 | 45246.170651 | 39.483 | 23 | 44579.406887 | 37.616 |
| 9 | 49724.541331 | 39.774 | 24 | 44748.449801 | 38.122 |
| 10 | 50291.207507 | 38.672 | 25 | 47793.258985 | 36.822 |
| 11 | 47845.107238 | 39.630 | 26 | 49248.597052 | 36.884 |
| 12 | 45300.541695 | 39.674 | 27 | 48840.155389 | 36.229 |
| 13 | 48034.448027 | 44.357 | 28 | 44909.323376 | 36.047 |
| 14 | 45277.154193 | 39.653 | 29 | 48016.776695 | 36.294 |
| 15 | 44775.291587 | 39.657 | 30 | 44876.332266 | 39.049 |

Based on the results in Table II, SA appears to be stable for the given medium-sized dataset. The range between the shortest distance of 44,337.363848 and the longest distance of 50,291.207507 is moderate, indicating a normal level of stability. The average distance of 46,947.496466 and average time of 38.871 seconds are consistent across the 30 runs, further supporting the stability of the algorithm.

### *C. Ant Colony Optimization (ACO)*

The simulation includes a dataset of 107 cities with a set of parameters, including 100 ants, 100 iterations, α value of 10.0, β value of 30.0, and an evaporation rate of 0.5.

TABLE III. EXPERIMENTAL RESULTS OF ACO

| Run | Solution | Time | Run | Solution | Time |
|---|---|---|---|---|---|
| 1 | 46115.743006 | 3.070 | 16 | 46193.939939 | 3.258 |
| 2 | 46268.251834 | 2.871 | 17 | 46150.291033 | 3.323 |
| 3 | 46534.114430 | 2.843 | 18 | 46334.169769 | 3.275 |
| 4 | 46491.382139 | 2.865 | 19 | 46381.841754 | 3.341 |
| 5 | 46485.866795 | 2.852 | 20 | 46405.102530 | 3.237 |
| 6 | 46481.287695 | 2.900 | 21 | 46350.308064 | 3.180 |
| 7 | 46163.551171 | 2.921 | **22** | **45688.391490** | **3.264** |
| 8 | 46379.236772 | 3.082 | 23 | 46264.078042 | 3.232 |
| 9 | 46242.863027 | 3.274 | 24 | 46496.732430 | 3.293 |
| 10 | 46218.432929 | 3.239 | 25 | 46439.118061 | 3.312 |
| 11 | 46549.060377 | 3.313 | 26 | 46238.591610 | 3.233 |
| 12 | 46397.593983 | 3.259 | 27 | 46273.890378 | 3.281 |
| 13 | 46443.338964 | 3.255 | 28 | 46276.00629 | 3.310 |
| 14 | 46233.315326 | 3.230 | 29 | 46477.313516 | 3.187 |
| 15 | 46408.399084 | 3.232 | 30 | 46298.463388 | 3.229 |

The result in Table III shows that the shortest final route distances for ACO is 45,688.391490, which took 3.264 seconds as the best solution, and the longest is 46,496.732430, which took 3.293 seconds as the worst solution. In the results of these 30 runs, the average distance was 46,322.689195 in 3.178 seconds. The gap between the shortest (45,688.391490) and longest (46,496.732430) distances is small, which indicates high stability. In addition, ACO demonstrates favourable average distance and running time compared to the other two algorithms, making it a highly suitable option for the given medium-sized dataset.

### *D. Comparison of Result of Optimization Algorithms*

TABLE IV. EXPERIMENTAL RESULTS

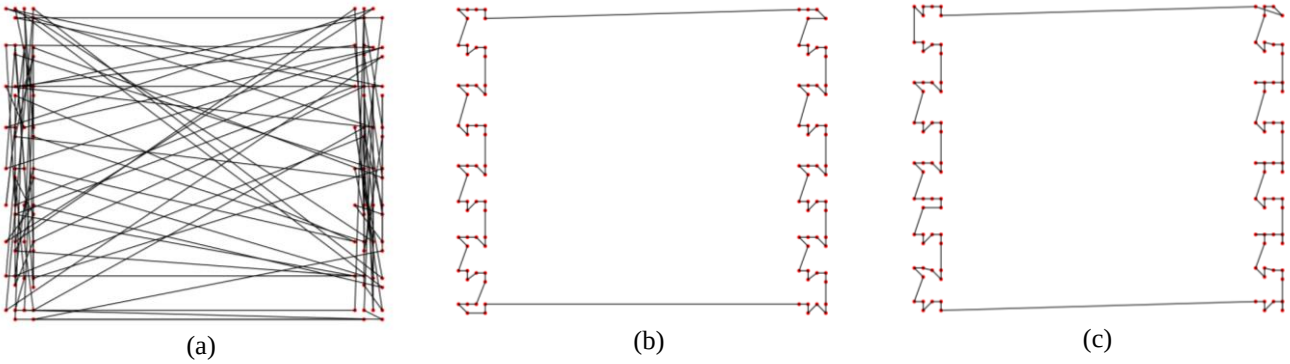
| Results | Models | | |
|---|---|---|---|
| | ***GA*** | ***SA*** | ***ACO*** |
| Final route distance | 494630.31612 | 44337.363848 | 45688.391490 |
| Time taken (seconds) | 0.088 | 36.633 | 3.264 |

Table IV shows the comparison of the best solution achieved by the three algorithms (GA, SA, ACO).

Based on Table IV, SA has the shortest route distance (44337.363848), followed by the ACO (45688.391490) and lastly GA (494630.31612). However, GA has the shortest time taken per iteration (0.088s), followed by the ACO (3.264s) and then SA (36.633s).

All algorithms work well with suitable running time for dataset with 107 cities. The results show that SA has the best trade-off between route distance and time taken, while GA has the advantage of being computationally efficient.

Figure 4 illustrates the route graph for the best solution obtained by the three algorithms (GA, SA, ACO). The red dots in the figure represent cities in its respecting coordinate, while the lines show the path connecting them.



(a) (b) (c)

Figure 4. Solution path for TSP: (a) Genetic Algorithm, (b) Simulated Annealing, (c) Ant Colony Optimization

The route graph can be used to determine the optimality of the solution. A solution with intersecting lines on its path is considered a negative solution as it is not the most efficient route. The optimal solution should have a clear and uninterrupted path. [2]

The dataset of 107 cities is best represented by Figure 4(b), by SA with no intersecting lines which shows the optimal solution. It gives the best solution at the shortest path level among the three algorithms.

The route graph in Figure 4(c) consists of more than one intersecting line, indicating that the ACO solution is slightly inferior to SA, which depicts the second-best solution.

In contrast, GA illustrates the most unexpected result as in Figure 4(a) that shows multiple intersecting lines, indicating that the solution path is not optimal.

### *E. Evaluation of Strength and Weakness of Optimization Algorithms*

Based on the results, GA shows the worst solution due to the value of input parameters. It may require a large population size and a high number of generations to obtain the optimal solution. This would cause extra memory and is computationally expensive.

On the other hand, SA shows the best result and is able to find the global optimum, indicating that it can easily handle complex and large-scale problems. However, it is computationally expensive compared with other two algorithms, and can be slow to converge to a solution which requires careful selection of initial parameters and cooling schedule.

Lastly, ACO shows the best stability, and have the most favourable average distance and computational time among three algorithms. Hence, it is suitable for complex problems and can efficiently explore a large search space. Although the route found is not as optimal as SA, the solution may be improved with parameter tuning. However, changes in choice of parameters and its values may result in slow convergence towards to the optimal solution, increasing the total computational time taken.

## CONCLUSION

This paper presents a comparative analysis of the performance of GA, SA, and ACO in solving a TSP of 107 cities. GA provides the worst solution but outperforms SA and ACO in terms of computational time. SA provides the best solution among the three algorithms by reaching global optimal within a reasonable time. The solution provided by ACO is less optimal than SA, but its computational time is lesser than SA and has the highest stability. Therefore, SA is proposed as the best algorithm for achieving the most optimal solution for TSP.

## REFERENCES

[1] F. Glover, “Tabu search—part I,” ORSA Journal on Computing, vol. 1, no. 3, pp. 190-206, 1989.

[2] M. Alhanjouri, “Optimization Techniques for Solving Travelling Salesman Problem,” International Journal of Advanced Research in Computer Science and Software Engineering, vol. 7, pp. 165-174, 2017, doi: 10.23956/ijarcsse/V7I3/01305.

[3] M. Chatting, “A Comparison of Exact and Heuristic Algorithms to Solve the Travelling Salesman Problem,” The Plymouth Student Scientist, vol. 11, no. 2, pp. 53-91, 2018.

[4] N. Sathya and A. Muthukumaravel, “A review of the optimization algorithms on traveling salesman problem,” Indian Journal of Science and Technology, vol. 8, no. 29, 2015, doi:10.17485/ijst/2015/v8i29/84652.

[5] P. Vaishnav and R. Patil, "A Comparative Study of GA and SA for Solving Travelling Salesman Problem," International Journal of Engineering Research & Technology (IJERT), vol. 3, no. 20, 2015.

[6] S. H. Zhan, J. Lin, Z. J. Zhang and Y. W. Zhong, “List-Based Simulated Annealing Algorithm for Traveling Salesman Problem,” Computational Intelligence and Neuroscience, vol. 2016, 2016, Article ID 1712630, doi: 10.1155/2016/1712630.

[7] S. Sahu, M. Nale and M. More, “Approximate Solution for NP Complete Problem,” International Journal of Science and Research, pp. 10-11, 2013.

[8] Z. Wu, “A Comparative Study of solving Traveling Salesman Problem with Genetic Algorithm, Ant Colony Algorithm, and Particle Swarm Optimization,” 2020 2nd International Conference on Robotics Systems and Vehicle Technology, 2020, doi: 10.1145/3450292.3450308.