Proceedings of the 2nd International Conference on Computing Innovation and Applied Physics
DOI: 10.54254/2753-8818/18/20230321

A comparative study between SA and GA in solving MTSP

Wenze Zhang[1*+], Chenyang Xu[2+]

[1]Electrical and Computer engineering, National University of Singapore, 119077, Singapore
[2]Mechanical Engineering, Shanghai Institute of Technology, Shanghai, 201418, China

*Corresponding author email: E0902047@u.nus.edu
+These authors contributed equally to this work and should be considered co-first authors

Abstract. The multiple traveling salesmen problems (MTSP) is a combinatorial optimization and np-hard problem. In practice, the computational resource required to solve such problems is usually prohibitive, and, in most cases, using heuristic algorithms is the only practical option. This paper implements genetic algorithms (GA) and simulated annealing (SA) to solve the MTSP and does an experimental study based on a benchmark from the TSPLIB instance to compare the performance of two algorithms in reality. The results show that GA can achieve an acceptable solution in a shorter time for any of the MTSP cases and is more accurate when the data size is small. Meanwhile, SA is more robust and achieves a better solution than GA for complex MTSP cases, but it takes more time to converge. Therefore, the result indicates that it is hard to identify which algorithm is comprehensively superior to the other one. However, It also provides an essential reference to developers who want to choose algorithms to solve MTSP in real life, facilitating them to balance the algorithm's performance on different metrics they value.

Keyword: multiple travelling salesmen problem (MTSP), NP-hard, optimization, Simulated Annealing (SA), genetic algorithms (GAs), Algorithm comparison, algorithm selection.

1. Introduction

The TSP problem is a classical combinatorial optimization problem that can be described as follows: a Salesman has to visit a number of cities, each city can only be visited once, start from one city, visit all cities and return to the departure city, and is required to choose a route that minimizes the total distance traveled. This is an np-hard problem that has $n!$ combinatorial solutions of various combinations, the number of which grows exponentially as the number of cities increases. Suppose there are 4 cities, which leads to 6 routes, and 5 cities, which leads to 24, then if the cities are scaled up to 20 cities, the number of routes scales up to 2.43×10^{18}, and 50 cities have 6.08×10^{62} seed routes. With a large number of cities, it is almost impossible to traverse all the routes.

In the real world, if the number of cities to be visited by a salesman is too large, then he has to travel a long distance to visit those cities. The problem is to reduce the distance travelled by the Salesman. A simple way to solve this problem is to make the number of Salesmen larger and the cities are divided into groups, each group is visited by one Salesman, which introduces the MTSP problem which is an

DOI: 10.54254/2753-8818/18/20230321

extension of the TSP problem. Both are the same np-hard problem, but the MTSP problem is more complex. The MTSP problem can be divided into four cases [1]:

(1) M salesmen start from the same city and return to that city.

(2) M salesmen leave from N different cities and return to the city they each left.

(3) M salesmen leave from the same city and return to N different cities.

(4) M salesmen leave from N different cities and return to the same city.

The first scenario is studied and modeled in this paper. The following additional qualifications are added: each city can only be travelled once; there can be no spare salesman; and MinMax (minimizes the maximum length of the journey between salesmen). There are four methods for solving the MTSP problem [2]: MTSP degenerates to TSP, exact algorithm, heuristic algorithm, and meta-heuristic algorithm. The MTSP degenerates to TSP, which makes the solution space more complex. Exact algorithms take much time. Heuristic algorithms are very dependent on problem-specific conditions. Meta-heuristics are a refinement of heuristics and are a combination of stochastic and local search algorithms, which tend to be more robust without recourse to problem-specific conditions and are thus used in a broader range of applications. Typical metaheuristics such as genetic algorithms and simulated annealing algorithms are also methods that will be used later in the paper. This paper will test the advantages and disadvantages of simulated annealing algorithms and genetic algorithms in solving the MTSP problem and attempt to find out the range of optimization of both algorithms. Furthermore, the paper will compare the two algorithms in terms of time complexity, space complexity, convergence capability, and final results.

2. Related Work

2.1. Simulated Annealing

The simulated annealing algorithm (SA) was proposed in 1953 by N. Metropolis et al. [3]. In 1983, S. Kirkpatrick et al. successfully introduced the idea of annealing to the field of combinatorial optimization. The simulation is a stochastic optimization algorithm based on the Monte-Carlo iterative solution strategy [4]. In the previous section, it was mentioned that mtsp is an np-hard problem, and in solving such problems, some algorithms, such as hill-climbing, are highly dependent on initial values and tend to fall into local optimum solutions. The advantage of SA is that it can accept poorer solutions with some probability and thus has some ability to jump out of the local optimum and tend to find the global optimum solution. Moreover, the initial solution of SA is generated randomly and has almost no relationship with the final solution, and all SA does not rely on reasonable initial solutions [5].

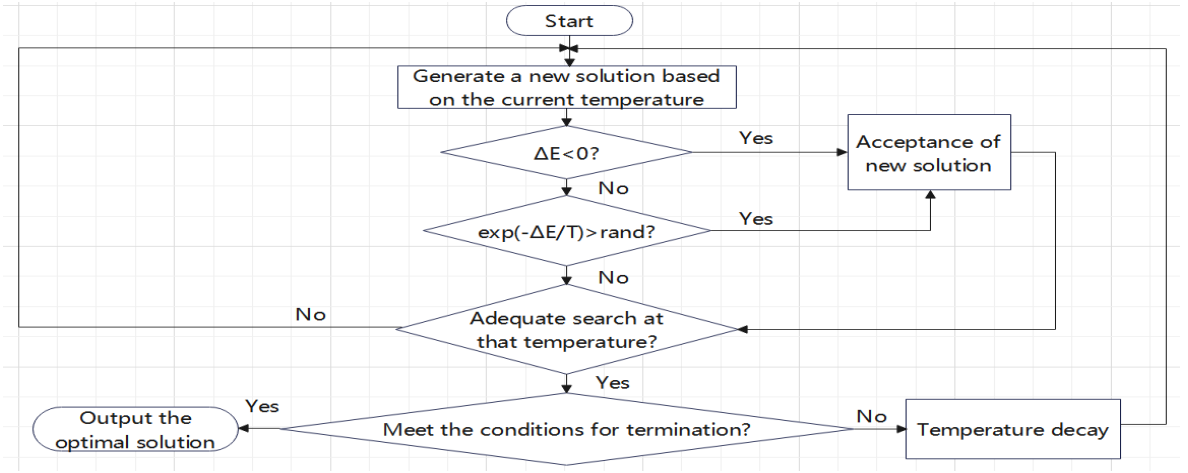

Figure 1. Flow-chart of simulated Annealing [6].

Figure 1 illustrates the exact flow of the simulated annealing algorithm. A very important property of SA is that when a poor solution is encountered, a probability is obtained according to the metropolis

DOI: 10.54254/2753-8818/18/20230321

criterion [3]: $\exp(\frac{-\Delta E}{T})$ to decide whether to accept the poor solution or not. When the temperature is high, SA will look for many solutions and accept even the worse solution with a higher probability. This property helps SA to jump out of the local optimum and eventually converge to the global optimum to some extent, and also frees SA from the dependence on the initial solution. When the temperature drops to a lower level, the probability of SA accepting a poor solution becomes very low, so it hardly jumps out of the current local optimum anymore, but converges further to the current found optimal solution.

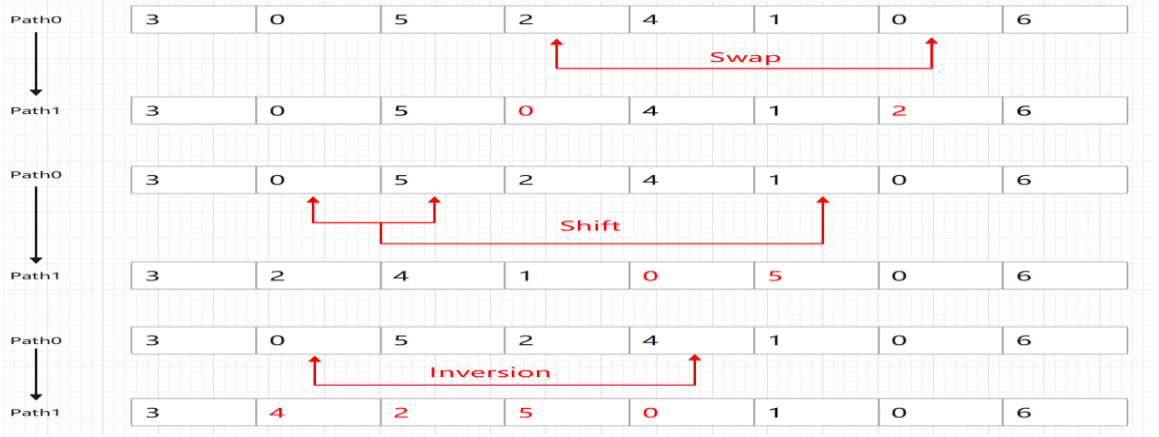

Figure 2. Three methods for generating new solutions.

Figure 2 shows three methods of generating new solutions, where there are two zeros in each path, which are used to split the path into three segments, and a salesman is assigned a segment of the path. For example, in the path in Figure 2, there are six cities assigned by three travellers.

2.2. Genetic Algorithm

Genetic algorithms (GAs) are an intelligent optimization algorithm that John Holland first developed in the 1970s. Based on the biological concept "survival-of-the-fittest". In GAs, an initial population is randomly generated. The population will undergo three operations, which are respectively selection, crossover and mutation, to produce a new generation of population that is more fit to the final goal. The general simulation process is shown in Figure 3.

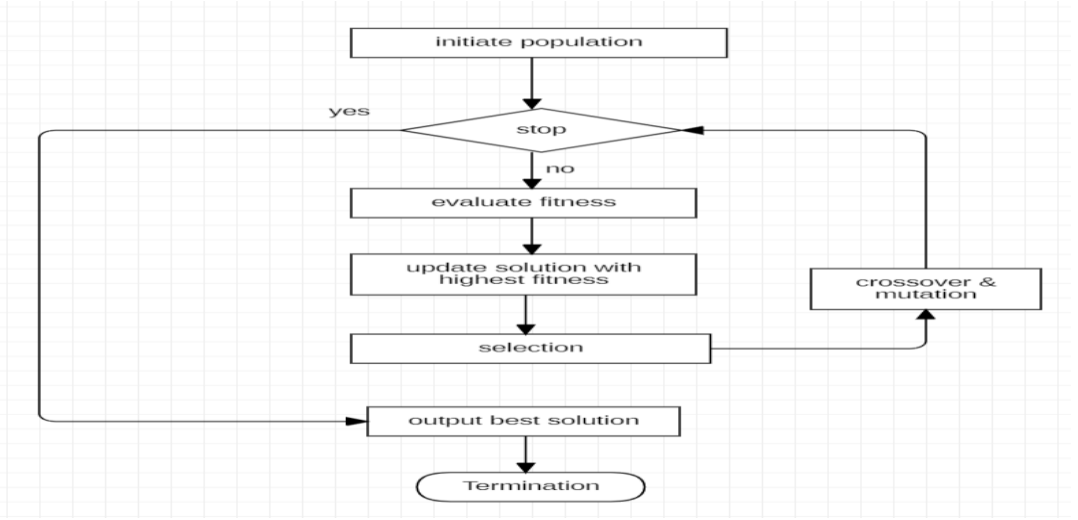

Figure 3. Flow-chart of typical Gas.

In selection operation, different genes (chromosomes) are selected to be in the future next generation based on their fitness value, and chromosomes with higher fitness value have more possibility of being

DOI: 10.54254/2753-8818/18/20230321

selected. On the other hand, the crossover operation randomly selects two parent chromosomes and mates them to produce offspring. The mutation alters some segments of chromosomes according to a predetermined possibility. Among these operations, selection and crossover are most influential to the final performance of the algorithms, and thus various operators are developed to fulfill different performance requirements of the algorithm. Moreover, as Goldberg found in his research paper, the possibility of crossover is usually fixed high, and the probability of performing mutation is fixed low [7]. Additionally, when implementing GAs to solve real-world problems, the method of encoding prospective planning solutions into chromosomes which usually contain numbers, also plays a crucial role in determining the final performance of the algorithm, and the most common practice in solving TSP is using a sequence of cities as the chromosome.

To solve the hot rolling scheduling problem in Shanghai Baoshan Iron & Steel Complex, Tang et al. (2000) model the problem as a MTSP whose goal is to minimize the total penalty cost, i.e., the setup cost in the procedure [8]. They solve this MTSP by converting it to a TSP by adding dummy points with the same position as the starting point to the graph. For example, if there are N nodes (including the starting node) and M salesmen, M dummy nodes, indexed as N+1….. Then, N+M was added to the chromosome. Moreover, distances between dummy nodes are set to infinite to avoid the solution of MTSP regressing to TSP.

A recent study evaluates different crossover methods for solving MTSP using the same approach as Tang [8,9]. The result indicates that Sequential constructive crossover (SCX) is the best crossover operator. However, different from this study, Al-Omeer's study aims at minimizing the total distance, and SCX uses a greedy algorithm, utilizing parents' chromosomes to construct an offspring that confirms to be better than its predecessors. However, this method is too complex in terms of implementation and time complexity to apply to minimize the maximum distance each salesman goes. What is more, Yuan et al. developed a new encoding and crossover method called two-part chromosome (TCX) (see Figure 4) [10]. According to their study, this method derives better results than SCX. Meanwhile, TCX can be applied to MinMax MTSP, and thus TCX is implemented in this study.

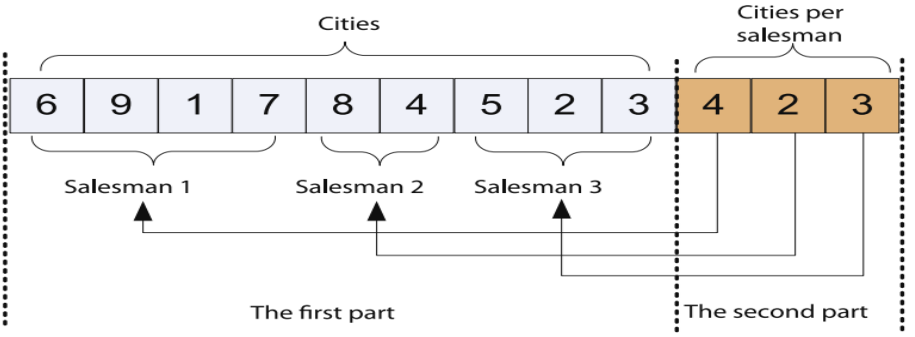

Figure 4. structure of two-part chromosome [9].

3. Computational Experiment

3.1. General

The proposed algorithms, GAs and Simulated Annealing, are implemented using python3. The experiment is conducted using the dataset TSPLIB, and the results are compared with the benchmarks obtained by Necula using the same TSPLIB data to evaluate algorithms' accuracy [11]. Considering that two different algorithms, in which the time complexities of one iteration are different, it is not just to compare the current best result after each iteration. Moreover, since the efficiency of existing algorithms cannot be affected, comparing the convergence of results based on real-time is more realistic. Therefore, a multi-process script is written. A subprocess will monitor the change of the current best result in the primary process every 0.05 seconds and plot a curve of two algorithms' convergence behavior over time.

DOI: 10.54254/2753-8818/18/20230321

The final result's stability is another metric that needs to be evaluated. Thus each algorithm is run in different MTSP settings 160 times (30 seconds per time). Afterward, Results are collected and analyzed by Excel to calculate the variance and average in various MTSP situations for each algorithm.

3.2. Simulated Annealing

Table 1. Parameters of Simulated Annealing algorithm.

Parameters	Value
initial temperature	*1000*
temperature iterations	*150*
number of iterations per temperature	*800*
cooling factor	*0.88*
acceptance probabilities	*Metropolis–Hastings algorithm*
methods for generating new solutions	*Swap*, *Shift*, *Inversion*

Setting reasonable SA parameters has a great impact on the running time, convergence ability and final results of the program. According to the study by Liang and the book by Bao, this paper has made some improvements to the parameters. Table 1 shows the improved parameters [6,12].

3.2.1. Initial Temperature. A larger value should be selected to ensure that the search range is large enough in the initial stage, and then narrow the search range when the temperature is reduced to eventually find the optimal global solution. The initial temperature should not be too large and should be chosen reasonably compared to other parameters. Otherwise, the computation is too large, and the running time is too long. The initial temperature was set to 1000.

3.2.2. Cooling Factor. The cooling factor and the number of iterations influence the optimal solution of SA. If the number of temperature iterations increases and the cooling factor is closer to 1, such as 0.9999, then SA is more likely to obtain the optimal solution, but the convergence speed becomes slower, and the search time increases. Most experiments with simulated annealing choose a cooling factor of 0.95 and above to get a better solution, but it takes more time. In later tests, the cooling factor was set to 0.88 for better comparison with the genetic algorithm to sacrifice the probability of finding an optimal solution to reduce the program's running time.

3.2.3. Temperature Iterations. Also known as the criterion for stopping the algorithm, the convergence theory of SA requires that the final temperature tends to 0. In the tests that follow, the number of temperature iterations was set to 150, which gives a termination temperature of order 10^-6 at an initial temperature of 1000 and a cooling factor of 0.88. This ensures that the search is stopped when the optimal value is continuously maintained.

3.2.4. Generating New Solutions. The swap, shift, and inversion methods are mentioned in Figure 2 for using the three methods. One of these is used randomly for the old solution to produce the new solution,

DOI: 10.54254/2753-8818/18/20230321

and a mixture of the three methods keeps the difference between the NT and the old solution within certain limits. The new solution is neither too close nor too far from the old solution.

3.3. Genetic Algorithm

Table 2. Parameters of Genetic Algorithms.

Parameters	Value
population size	*150*
selection method	*Roulette-wheel selection with scaling*
crossover probability	*0.85*
crossover method	*two-part chromosome crossover*
mutation probability	*initially 0.2 and then gradually reduce to less than 0.1*
mutation method	*swap*
iteration	*13000*

3.4. Selection

The selection operator that is utilized is roulette-wheel with scaling (see Table 2). In this operator. Each individual's probability to be selected is the proportion of its fitness in the sum of the population's fitness value. A random number from 0 to 1 is generated and the chromosomes whose proportion covers the number is selected as the parents in new generations. The whole process can be thought of using an imaginary roulette-wheel where all individuals occupy different proportions on it based on their fitness and the random number is the pointer. However, the goal is to minimize the maximum distance and thus the result of superior solutions are smaller. Since roulette-wheel selection requires better solutions to occupy more space in the imaginary roulette wheel. The fitness value is calculated using the equation:

$$fitness = (C/d)^3 \ldots \quad (1)$$

In this equation, d refers to the MinMax value of the solution, C is a constant but might vary when the graph of cities changes. In this case, better solutions have larger value. C/d is powered by 3 in order to make the difference of fitness values between solutions more pronounced, confirming that the algorithm will converge finally.

3.5. Crossover

The crossover method that is implemented, as shown in Table 2, is based on the two-part chromosome (TCX). By changing the first part of the chromosome, which is the sequence of cities, the second part changes accordingly. Through this method, it is possible to maintain some of the superior gene segments of parents while generating offspring that have enough diversity.

For example, if the scenario is 8 cities (excluding the starting city) and 3 salesmen and two parent (P1 is the base chromosome) is respectively:

P1 = [(1 5)(7 8)(2 3 4 6) | 2 2 4]

P2 = [(3 8 1)(5 6 7)(4 2) | 3 2 3]

A segment of gene is randomly selected for each salesman from the base chromosome(P1) and extract genes that remain not in these segments in order from P2.

P1 = [(1 5)(7 8)(2 3 4 6) | 2 2 4] select: [(5)(7 8)(2 3)]

P2 = [(3 8 1)(5 6 7)(4 2) | 3 2 3] remain: [1 6 4]

Then randomly divide the extracted genes into as many parts as salesmen number (parts can be void) and concatenate them with selected segments respectively. Then finally update the second part of the chromosome.

select: [(5)(7 8)(2 3)]

remain: [(1)(6 4)()]

concatenate: [(5 1)(7 8 6 4)(2 3)]

DOI: 10.54254/2753-8818/18/20230321

update city number: [(5 1)(7 8 6 4)(2 3) | 2 4 2]
one of the final offspring: [5 1 7 8 6 4 2 3 2 4 2]

3.6. Mutation

Based on swap mutation, two genes in each part of the two-part chromosome are randomly selected and swapped [7]. Unlike selection which homogenizes the population, mutation prevents algorithms from falling into local optimality and maintains an appropriate and acceptable diversity of population. To find an optimal solution, algorithms need to explore various prospective solutions as much as possible initially and converge quickly afterwards. Therefore, the initial mutation probability is set to be 0.2 and this probability will decrease every iteration until it reaches a minimum value.

4. Result

4.1. Plot

Based on three benchmarks in TSPLIB, the experiment is conducted on six scenarios which are respectively 20 cities with 2 salesmen, 20 cities with 3 salesmen, 51 cities with 3 salesmen, 51 cities with 3 salesmen, 76 cities with 3 salesmen, and 76 cities with 5 salesmen. As shown in Figure 5, charts are plotted to represent algorithms' convergence performance with regard to time. In each chart of Figure 5, the blue curve represents GAs, orange curve represents SA and the y-axis represents the current best value (best solution). However, since these two algorithms are randomized and results are based on probability, The code is run multiple times and the most typical diagrams that best reflects the majority of results and important properties are selected and shown below.

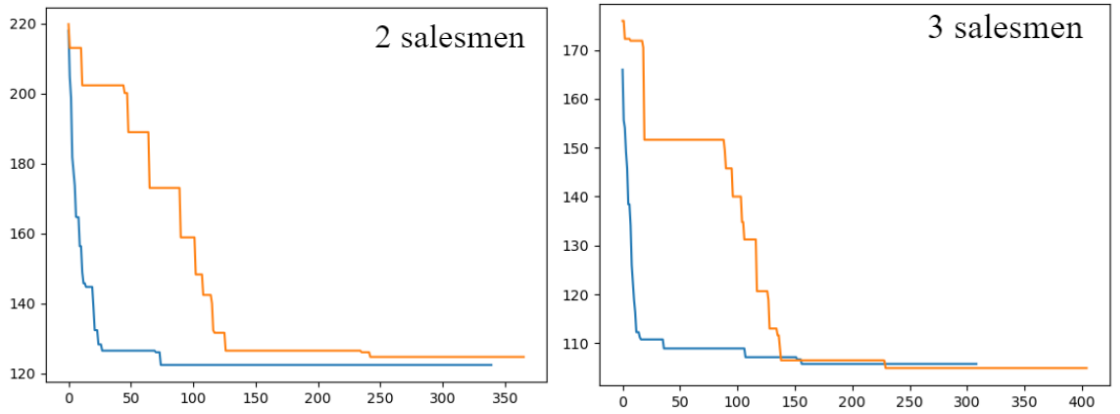

Figure 5. Convergence behaviour vs. time on 20cities.

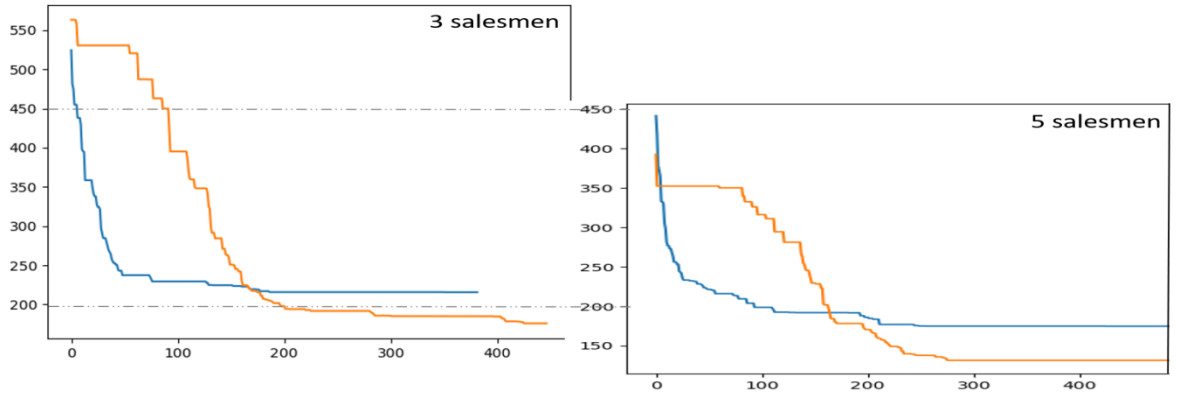

Figure 6. Convergence behaviour vs. time on 51cities.

DOI: 10.54254/2753-8818/18/20230321

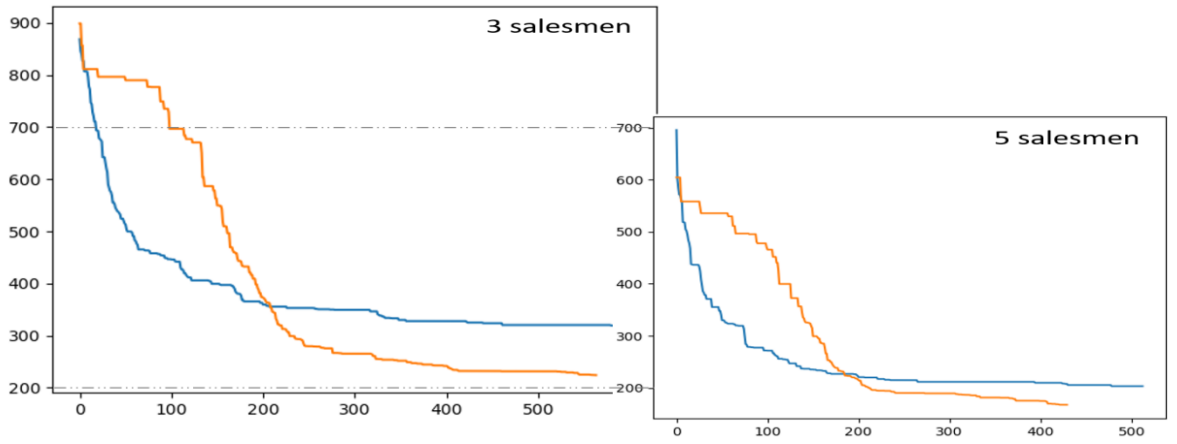

Figure 7. Convergence behaviour vs. time on 76cities.

All the charts above reveal that GA always converges faster than simulated annealing. Moreover, in this study's experiment settings, most of the time GA converges about 5 to 7 seconds faster than simulated annealing no matter what the scenarios are. Additionally, in spite of the scenario of 20 cities, in most cases the results of simulated annealing are closer to global optimal solutions. Comparing charts plotted in situations that have same city numbers, when the total searching space increases, for example, from $75! * \binom{74}{2}$ (76 cities, 3 salesmen) to $75! * \binom{74}{4}$ (76 cities, 5 salesmen), the time to converge does not change significantly. What can also be discovered is that simulated annealing performs better than GAs when salesmen are less while two algorithms approach similar results with the increment of salesmen.

4.2. Statistic

Using the mentioned 6 scenarios, a table is made to record the results from two algorithms' 160 runs . In table 3, avg is the average MinMax result, S.D is the standard deviation of results and min is the best results that algorithms derive in 160 runs. It is revealed that the simulated annealing performs more stable than GAs. Additionally, the standard deviations of two algorithms' results both increase when salesmen number decreases and the increment is more drastic in GAs.

Table 3. Standard deviation, average and best solution after 160 runs.

	2 salesmen			3 salesmen		
20 cities	Avg	S.D	Min	Avg	S.D	Min
SA	122.57	0.68	122.43	105.63	0.31	104.92
GA	123.72	1.89	122.43	106.29	1.07	104.92
	3 salesmen			5 salesmen		
51 cities	Avg	S.D	Min	Avg	S.D	Min
SA	173.42	5.57	162.47	126.34	3.54	119.57
GA	215.39	12.91	186.05	153.31	8.80	131.06
76 cities	Avg	S.D	Min	Avg	S.D	Min
SA	225.82	7.29	208.23	163.76	6.13	151.55
GA	317.38	20.35	273.33	221.64	12.22	193.97

DOI: 10.54254/2753-8818/18/20230321

Table 4. The accuracy of two algorithms compared with benchmark.

	SA	GA	Difference
20 cities （2 salesmen）	99%	99%	0
20 cities （3 salesmen）	99%	99%	0
51 cities （3 salesmen）	92%	74%	18%
51 cities （5 salesmen）	98%	81%	17%
76 cities （3 salesmen）	87%	62%	25%
76 cities （5 salesmen）	92%	68%	24%

Table 4 shows that when the number of travellers is fixed and the size of cities increases (20cities→51 cities→76 cities) and (51 cities→76 cities), the results for both SA and GA decrease to varying degrees.

GA: (3 salesmen) 99% → 74% → 62%, (5 salesmen) 81% → 68%

SA: (3 salesmen) 99% → 92% → 87%, (5 salesmen) 98% → 92%

The decline in GA is most pronounced, which shows that the increase in the size of the city has led to a sharp increase in the number of solutions and an increase in the number of local optima, resulting in poorer results for GA, revealing its tendency to fall into local optima. The decrease of SA is smaller, which indicates that SA is less influenced by the local optimum, and it is easy to jump out of the local optimum and tend to find the global optimum.

When the number of cities is increased from 3 to 5, the results for both GA and SA improve by different degrees.

GA: (51 cities) 74%→81%, (76 cities) 62%→68%

SA: (51 cities) 92% → 98%, (76 cities) 87% → 92%

GA increased by 6%-7% and SA increased by 5%-6%. It can be seen that when the number of travellers is boosted and the number of MTSP solutions decreases, the results of both algorithms improve, with GA improving slightly more than SA.

Overall, only in terms of accuracy, SA outperforms GA except for the 20city set of data, which should be because SA has a higher probability of jumping out of the local optimum initially, while GA has difficulty in jumping out of the local optimum. However, GA is not entirely without advantages, as shown in Figures 5, 6, and 7. When the city size is small, e.g., 20 cities and below, GA obtains almost the same results as SA but with a shorter running time.

From the analysis of obtained statistic, it can be concluded that for genetic algorithms, the advantages are: (1) GA has a short time to reach an acceptable solution in either MTSP case, which is ideal for situations where real-time planning is required and planning time is limited. (2) GA is well suited to situations where the city size is small, the number of salesmen is significant, where the number of solutions is small, and GA can obtain an optimal solution more quickly than SA and with acceptable solution quality. Meanwhile, tGAs also have disadvantages: (1) Weak robustness. When the city size is more extensive and the number of salesmen is smaller, its final result is farther away from the optimal solution, and the decline is significant. (2) It is easy to fall into a local optimum, resulting in a less than optimal result.

As For simulated annealing, it has advantages: (1) Stronger robustness. The most complex case in MTSP is when the city size is large, and the number of salesmen is small. The number of solutions, in this case, is the largest. However, SA is still applicable, and its final result is slightly lower than the most straightforward case (a few nodes and many salesmen), with a smaller drop. (2) It is easy to jump out of the local optimum and make the final result converge to the global optimum. While its disadvantages are: (1) In all cases of MTSP, it takes longer for SA to reach an acceptable solution than GA and is unsuitable for situations with short planning times.

5. Conclusion

In this study the performances of genetic algorithms and simulated annealing are compared and the result shows that it is hard to identify which algorithm is comprehensively superior to the other one.

DOI: 10.54254/2753-8818/18/20230321

Because the algorithms' performance mainly depends on the conditions and the metrics developers value. Thus, when trying to solve real-world problems, the strengths and weaknesses of two algorithms should be considered and combined with actual situations. According to the experiment results, this paper proposes that GA is more suitable in real-world problems such as peak-time food delivery planning and robot cluster control, where situations keep changing and real-time planning is required. Whereas, SA can be applied to large-scale nationwide distribution of supplies, such problems require exact solutions with more relaxed time constraints.

In this present study, all developed operators and components of GAs and SA are respectively based on literature of each algorithm. However, since there is similarity, for example, the generation process of new solution in these algorithms, between these algorithms, a better hybrid algorithm can be developed using various operators both from GAs and SA. Considering our conclusion that GA has better global searching capacity and SA are more likely to find optimal solution, we will try to combine two algorithms and develop new algorithms in our future study. For example, we will try to introduce temperature and metropolis criterion into the process of generating offspring and evaluate the performance of the new algorithms. Hopefully a superior algorithm can be developed under this study.

Acknowledgement

Zhang Wenze, and Xu Chenyang contributed equally to this work and should be considered co-first authors

References

[1] Estévez-Fernández, A., Borm, P., Hamers, H. (2006) On the core of multiple longest traveling salesman games. European journal of operational research, 174(3): 1816-1827.

[2] Xu, M., Li, S., Guo, J. (2017) Optimization of multiple traveling salesman problem based on simulated annealing genetic algorithm. Matec web of conferences, 100: 02025.

[3] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., Teller, E. (1953) Equation of state calculations by fast computing machines. The journal of chemical physics, 21(6): 1087-1092.

[4] Binder K (1978) Monte Carlo methods in statistical physics. Springer, Berlin

[5] Burke, E. K., Burke, E. K., Kendall, G., Kendall, G. (2014) Search methodologies: introductory tutorials in optimization and decision support techniques. Springer, Scotland.

[6] Bao, Z., Yu, J., Yang, S. (2016) Intelligent optimization algorithm and its MATLAB example. Publishing House of Electronics Industry, Beijing.

[7] Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. Reading, MA: Addison Wesley

[8] Tang, L., Liu, J., Rong, A., Yang, Z. (2000) A multiple traveling salesman problem model for hot rolling scheduling in Shanghai Baoshan Iron & Steel Complex. European Journal of Operational Research, 124(2), 267-282.

[9] Al-Omeer, M. A., Ahmed, Z. H. (2019, April) Comparative study of crossover operators for the MTSP. In 2019 International Conference on Computer and Information Sciences (ICCIS) (pp. 1-6). IEEE

[10] Yuan, S., Skinner, B., Huang, S., Liu, D. (2013) A new crossover approach for solving the multiple travelling salesmen problem using genetic algorithms. European journal of operational research, 228(1), 72-82.

[11] Necula, R., Breaban, M., Raschip, M. (2015, June) Performance evaluation of ant colony systems for the single-depot multiple traveling salesman problem. In International Conference on Hybrid Artificial Intelligence Systems (pp. 257-268). Springer, Cham.smns

[12] Liang, G., Zhang, S., Huang, F., He, S. (2007) A Kind of Renewed Simulated Annealing Algorithm Solves 0-1 Knapsack Problem. JOURNAL OF GUANGXI UNIVERSITY FOR NATIONALITIES(Natural Science Edition), 13(3): 91-93.