International Journal of Computational and Experimental Science and ENgineering (IJCESEN)
Vol. 6-No.1 (2020) pp. 23-28
http://dergipark.org.tr/en/pub/ijcesen

ISSN: 2149-9144

Research Article

Analysis of TSP: Simulated Annealing and Genetic Algorithm Approaches

A. Reha BOTSALI*, Kemal ALAYKIRAN

N. Erbakan University, Engineering and Architecture Faculty, Industrial Engineering Dept., Konya –Turkey

*** Corresponding Author** : rbotsali@erbakan.edu.tr
ORCID: 0000-0002-8809-9353

Article Info:

DOI: 10.22399/ijcesen.637445
Received : 24 October 2019
Accepted : 12 March 2020

Keywords

Traveling Salesman Problem (TSP)
Simulated Annealing (SA)
Genetic Algorithms
Integer Programming

Abstract:

This paper analyzes the performance of the popular heuristic methods 'Simulated Annealing (SA)' and 'Genetic Algorithm (GA)' on the symmetric TSP. TSP is a well-known combinatorial optimization problem in NP-complete class. NP-completeness of TSP originates many specific approximation algorithms to find optimal or near optimal solutions in a reasonable time. On the other hand, both SA and GA are general purpose heuristic methods that are applicable to almost every kind of problem whose solution lies inside a search space. The performance of SA and GA depends on many factors such as the nature of the problem, design of the algorithm, parameter values, etc. In this paper, a GA and an SA algorithm are given and their performance with respect to several factors is analyzed. The algorithms are tested on some benchmark problems (TSPLIB) which are obtainable via Internet from http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html.

1. Introduction

Traveling Salesman Problem (TSP) is an NP-complete problem [1] and it is one of the most popular problems in combinatorial optimization. As a brief definition of TSP, there are *n* cities and there is a distance between each city pair (If there is no road between two cities then the distance can be taken to be ∞). The distance between two cities may depend on the travel direction as in the case of asymmetric TSP, or it can be independent of the travel direction which is called symmetric TSP. The objective is to find a tour of minimum total length in which every city is visited exactly once.

There are numerous ways to find the optimal solution. The simplest and the most costly way may be to enumerate all possible combinations which has a size of *n*! for an *n* city problem. Other than this, branch and bound (B&B) and mixed integer programming (MIP) methods can be used to find the optimal solution. However, none of these have polynomial runtime and this makes TSP attractive for heuristics.

In the literature, there are several variations of TSP that are analyzed by the researchers. For example, Cacchiani et al. [2] analyzes TSP for time-dependent service times. TSP can be seen as a sub-problem of vehicle routing problems (Ex. [3], [4]). This problem also gathers attention from researchers in computer science and electronics fields (Ex. [5], [6]). Regardless of the context in which TSP is analyzed, heuristics are the mostly used solution methodology for this problem and there are several heuristics to solve TSP. Some of these heuristics are designed for only solving TSP such as "nearest neighborhood algorithm" (NNA) or "Christofides Heuristic" [7]. However, it is also possible to apply general heuristic methods such as "neural networks", "simulated annealing" (SA) or "genetic algorithm" (GA). In this paper, the performance of the two of these methods, SA and GA are tested on several symmetric TSP instances. The outline of the paper is as follows: In section 2, the GA and SA algorithms used in this study are presented. In section 3 and 4, the results and the discussion of the results are given respectively. In the last section, the comments and further possible

extensions of the methods used in this study are suggested.

2. Design of the Algorithms

Both GA and SA methods rely on some main ideas during the search of the solution space. Simply, SA algorithm starts with an initial solution at some specific temperature *T*. At each iteration, it checks the solutions in the neighborhood of the current solution. If a better solution is found, then it is accepted. If a solution with worse objective value than the current solution is found, then this worse solution is accepted with some probability depending on the temperature of the current iteration. Throughout the iterations, by gradually decreasing the temperature, for later iterations the probability of accepting a worse solution is decreased continuously and the convergence to a solution is satisfied.

On the other hand, GA, as opposed to SA, works on more than one solution at each iteration (generation). Starting with an initial pool of solutions, at each generation, with some rate (cross over rate), the solutions are combined with each other to generate new solutions (off-springs). Other than this, also with some specific rate (mutation rate), the existing solutions in the population are modified. After a predetermined number of generations or by some other stopping criterion, the algorithm stops after hopefully converging to a solution.

Although GA and SA depend on some basic ideas, the performance of their solutions highly depends on how these algorithms are implemented. In below subsections, the implementation details of the algorithms are given.

2.1 Design of the Genetic Algorithm

Generally, binary strings are used to represent the solutions in GA implementation. However, the nature of TSP allows us to use directly the permutation of the cities as strings. So each individual in the GA population consists of a permutation of cities and the total distance cost for the corresponding tour.

Each generation of GA consists of *(population size × crossover rate)* number of new individuals and *(population size × (1-crossover rate))* of the best individuals from the previous iteration. The individuals for crossover operation are selected by a tournament of size 2. At a cross over operation, for each i^{th} city position in the tour of the off-spring, a coin is tossed. Depending on coin toss, the i^{th} city from parent 1 or parent 2 is taken for the off-spring. If this city is visited before in the off-springs' uncompleted tour string, then starting from the beginning the first unvisited city from the chosen parent is taken as the i^{th} city for the off-spring.

Two types of mutation operations are used in the algorithm. When it is decided to have mutation on a tour, randomly two different city positions are selected from the tour. If it is a type 1 mutation then, all the cities between the chosen city positions are flipped in the reverse order. On the other hand, if it is a type 2 mutation, then just the positions of the selected cities are changed.

Other than the general mutation rate, another mutation rate is defined to decide on the type of the mutation. The rate of type 1 and type 2 mutations are defined to be dynamic depending on the generation number. Whenever a mutation occurs, with probability:

$$1 - \frac{(generation\ \#)}{(max.\#\ of\ generations)} \qquad (1)$$

Then, it is a type 1 mutation. As it is seen, type 1 mutation occurs mostly at the initial generations, whereas type 2 mutation occurs mostly in later generations. The logic behind this is that type 1 mutation radically changes the tour, so it is more appropriate to make radical changes on the individuals during the initial generations where the population is unstable. On the other hand, type 2 mutation is more appropriate to make fine tuning kinds of changes on the individuals which is better when the population is about to converge. The crossover and mutation operations are done respectively for each generation. Iteratively new generations are produced until a previously defined number of generations limit is reached.

2.2 Design of the Simulated Annealing Algorithm

The SA algorithm starts with an initial tour which is randomly chosen and with some initial temperature T_o. The temperature is decreased throughout the iterations with a cooling rate *R*. At each temperature, for a specified number of times (number of cycles at a temperature), the neighbor-hood of the current solution is checked. The neighborhood of a solution is defined as the tours that are different from the current solution by just two city positions. If a better solution is found, then

it is accepted, otherwise it is accepted with probability:

$$e^{(\text{cost difference between solutions/ current temperature})}. \quad (2)$$

If *i* and *j* denote the city locations, then the neighborhood search process can be summarized as in Fig. 1. After a predetermined number of temperature decreases, the algorithm stops.

TSP has the constraint of visiting each city exactly once in a given tour. When we apply GA or SA on a constrained problem, we may choose two ways of search strategies, we can either accept infeasible solutions with some penalty or we may just search over the feasible solutions. In this project, the second method is applied. In the iterations of both algorithms, only feasible tours are considered

```
For i=1 to problem size-1
  For j=i+1 to problem size
    Swap the cities on ith and jth locations.
    If the new tour is better, then current tour is new tour
  Else current tour is new tour with probability e^(Δcost/T)
```

Figure 1. *Summary of Neighborhood Search Loop for SA.*

3. Results

The algorithms are coded in C programming language and tested on a SGA Unix machine with 1 GB main memory. For GA code, upon a series of trials, a set of effective parameter values is found to be as 500 for population size, 0.6 for crossover rate, 0.5 for mutation rate. It was observed that a good maximum number of generations is problem dependent. If the number of cities is higher, then a higher maximum number of generations is better. For this reason, depending on problem size (size<50, 50<size<100, size>100) different number of maximum generations (500, 1000, 1500) are used.

For SA code, an initial temperature of 110 is found to be effective. 60 temperature decreases are allowed throughout the iterations. Depending on problem size, a different value for the number of cycles at each temperature is used (60 for size<50, 80 for 50<size<100, 100 for size>100).

The algorithms are tested using the symmetric TSP instances obtained from TSPLIB [9]. The numbers at the end of the problem names show the size of the problem. The results of GA and SA algorithm can be seen in Tables 1 and 2 respectively. In these tables, the column OPT contains the optimal solution for the TSP instance. Column BEST shows the best solution obtained by the algorithm over the trials. AVGBEST shows the average of the best solutions from the trials. Column DEVBEST shows the standard deviation of the best solutions. GENAVG shows the overall final average of the populations from all trials. DEVAVG contains the standard deviation info for the population averages over all the trials. TIME shows the average time required per trial. Finally, DEV shows the percentage deviation of the best solution obtained over the trials from the optimal solution.
For SA, since the 20 trials of problem instance Gr202 is not completed within 48 hours, no result is presented. One of the initial objectives was to solve TSP by a MIP model and compare the solution time. For problems Burma14 and Uly16 the optimal solution is obtained in seconds when corresponding MIP model is solved by CPLEX [8]. However, after observing for a 22 city problem (Uly22) the solution is not found in 8000 sec, it is decided to use the optimal solutions provided by TSPLIB. The optimal costs are calculated by using the optimal tours given for the problem instances.

To test the effect of the quality of the initial population on GA, for some problem instances, the tours found by NNA are included in the initial population. The results of this experiment are given in Table 3.

As first remarks, it is seen that the algorithms generally perform well on small sized problems and as the problem size increases, the deviation of the solutions from the optimal solution increases. Also, on average SA requires more time than GA. Further discussion and analysis of the results are given in the next section.

4. Discussion

As mentioned in previous section, it is observed that SA requires more time than GA. As the problem size increases, this difference becomes more remarkable. The main reason for this can be the neighborhood search procedure that is summarized in Fig. 1. This search loop is $O(n^2)$ and as the problem size increases, the search time increases in a quadratic fashion. This also explains the reason why SA gives better results than GA in trial runs. Simply more search effort gives better results. As mentioned in previous section, different number of maximum generation limits are used

depending on the problem size. Given a fixed population size, GA converges more quickly for small size problems. This is also clear in Fig. 2 where the convergence of two problem instances (Burma14 and St70) are compared. It is seen that a small size problem converges much faster than a larger problem. One of the questions related to GA is the effect of the quality of the initial population on the performance of the algorithm. This case is analyzed by placing some tours found by NNA in the initial population for some problem instances. The results in Table 3 show that except one instance, having better individuals in the initial population have positive effect both on the best solution and average solution of GA. For the problem instance Uly22, placing better individuals on the initial population led to convergence to a sub-optimal tour. Most probably, for this problem instance, the individuals found by NNA and placed in the initial population become dominant and push the population towards a sub-optimal tour.

Table 1. *GA Results*

	OPT	BEST	AVGBEST	DEVBEST	GENAVG	DEVAVG	TIME	DEV
Burma14*	4894	4894	4895.50	10.34	5746.18	78.38	<1min	0.00
Ulysses16*	8063	8063	8097.40	37.80	9089.64	119.68	<1min	0.00
Ulysses22*	8265	8265	8363.80	133.74	9592.90	232.44	<1min	0.00
Eil51**	426	441	473.50	15.94	508.09	22.98	<1min	3.52
Berlin52**	7542	8074	8579.38	303.04	9911.17	386.59	<1min	7.05
St70**	675	748	852.86	48.08	991.01	58.22	<1min	10.81
Pr76**	108159	117383	138968.00	7815.75	157887.30	9410.17	<1min	8.53
Gr96**	74804	90293	103668.90	6457.31	115594.60	6818.84	<1min	20.71
Ch150***	6528	9935	11410.50	661.56	12181.71	736.50	1.2min	52.19
Gr202***	53735	87345	94776.00	4075.45	98503.60	4064.86	1.7min	62.55

*over 100 trials , **over 50 trials, ***over 30 trials

Table 2. *SA Results*

	OPT	BEST	AVGBEST	DEVBES	TIME	DEV(%)
Burma14	4894	4894	4894.00	0.00	<1 min	0.00
Ulysses16	8063	8063	8069.96	23.84	<1 min	0.00
Ulysses22	8265	8265	8306.90	75.41	<1 min	0.00
Eil51**	426	432	443.87	5.30	1.5 min	1.41
Berlin52*	7542	7678	7988.97	167.53	1.6 min	1.80
St70**	675	690	721.63	18.45	1.8 min	2.22
Pr76**	108159	115527	121347.10	3121.22	4.9 min	6.81
Gr96**	74804	75343	80329.07	2028.98	9.6 min	0.72
Ch150**	6528	7412	7623.75	133.31	45.3 min	13.54
Gr202***	53735	NA	NA	NA	NA	NA

* over 50 trials, ** over 30 trials, *** over 20 trials

Table 3. *Effect of the Quality of the Initial Population on GA*

	OPT	BEST	AVGBEST	DEVBEST	GENAVG	DEVAVG	DEV(%)
Burma14*	4894	4894	4894.82	5.80	5721.58	77.09	0.00
Ulysses16*	8063	8063	8131.86	25.70	9016.50	59.80	0.00
Ulysses22*	8265	8446	8473.30	24.62	9468.52	58.56	2.19
Eil51**	426	428	439.78	6.18	507.35	15.32	0.47
Berlin52**	7542	7542	7802.28	116.74	8972.10	116.74	0.00
St70**	675	721	729.14	3.59	827.69	13.56	6.81
Pr76**	108159	112779	119955.60	3296.34	13885.70	5665.78	4.27

* over 100 trials, ** over 50 trials

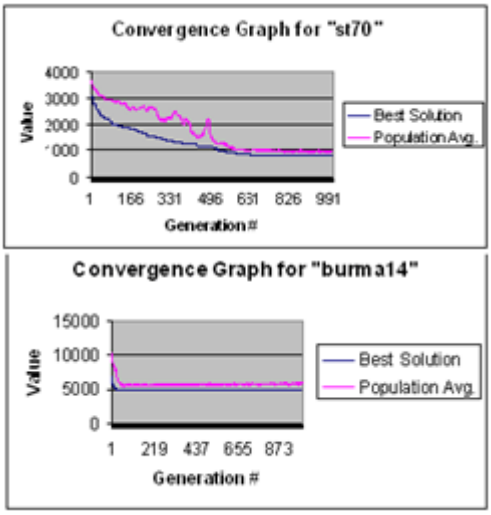

Figure 2. *Convergence and Problem Size Relationship*

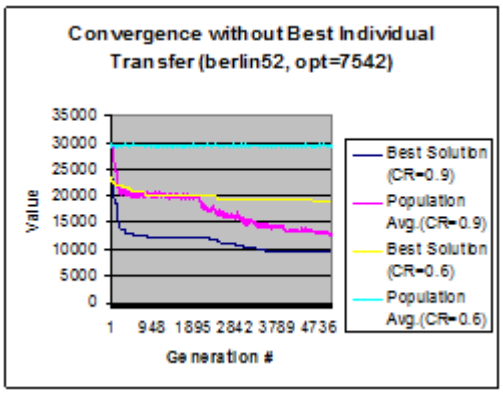

Figure 3. *Convergence of GA without Best Individual Selection for Berlin52*

In GA, at each generation, *((1-crossover rate)×population size)* of the best individuals from the previous generation are included. The purpose of this is to maintain a good population and force the population to converge to an optimal or near optimal solution. One argument may be that this approach may lead GA to converge to a sub-optimal solution quickly. However, in fact the reverse of this argument may be correct which is "neglecting the best values in the previous generation may delay the convergence". This is also supported by the trial run of problem in-stance "Berlin52". For this problem instance, GA is run without selecting best individuals from the previous solution and as seen in Fig. 3, it is observed that even for a maximum generation limit of 5000, the population does not converge to the optimal solution. The reason for this may be the low crossover rate (0.6) used by the best individual selection algorithm. However, when the crossover rate is set to 0.9, the algorithm still does not converge to a near optimal solution in 5000 generations as seen in Fig. 3.

5. Conclusion

In this study, the performance of GA and SA methods are tested on TSP. Although, the initial results show that SA performs better on the average, there are many factors affecting the performance of the algorithms and it is hard to give a clear cut answer such as "SA performs better" or vice versa.

It is shown that having better individuals lead GA to give better solutions in general. This result brings up the idea of focusing on hybrid algorithms. For example, when GA is applied on a problem, the initial population of individuals may be formed by applying a greedy heuristic which is NNA in this case. On the other hand, one thing should be noticed that sometimes having good individuals in the initial population may cause GA to converge to a sub-optimal solution. So, analyzing the initial population characteristics may be beneficial for GA designer.

Another point that is focused on in this study is the relationship between the consecutive generations of GA. The methods like tournament selection or transferring some proportion of the best individuals from one generation to the next one affect the convergence of the algorithm. By a careful analysis, GA designer may find the balance point where the algorithm converges to optimal or a near optimal solution with minimum computational effort.

About SA algorithm, it is observed that as the computational effort in search process increases, SA gives better results. This means designing SA algorithm requires attention for carefully defining the tradeoff between the search procedure and the computational effort.

One interesting extension may be to combine SA and GA as a hybrid method, basically, at each generation of GA, we can apply a limited neighborhood search on the best solution of that generation as done in SA. By this way, it may be possible to both improve the current best solution of the generation and search over other solutions which are not in the neighborhood of the best solution.

References

[1] Michael R. Garey and David S. Johnson. Computers and intractability. A guide to the theory of NP-completeness. WH Freeman and Company, San Francisco 1979.

[2] Cacchiani, V., Contreras-Bolton, C., Toth, P. "Models and algorithms for the Traveling Salesman Problem with Time-dependent Service times", V. 283, No. 3, 2020, P. 825-84.

[3] Saxena, R., Jain, M., Malhotra, K., Vasa, K.D. "An Optimized OpenMP-Based Genetic Algorithm Solution to Vehicle Routing Problem", Advances in Intelligent Systems and Computing, V. 767, 2020, P. 237-245.

[4] Singh, V., Ganapathy, L., Pundir, A.K. "An improved genetic algorithm for solving multi depot Vehicle Routing Problems". International Journal of Information Systems and Supply Chain Management, V. 12, No. 4, 2019, P. 1-26.

[5] Lu, Y., Tian, H., Yin, J. "A real-time routing protocol in wireless sensor-actuator network", Communications in Computer and Information Science, V. 931, 2019, P. 111-120.

[6] Wang, H., Zhang, N., Créput, J.-C. "A massively parallel neural network approach to large-scale Euclidean traveling salesman problems", Neurocomputing, V. 240, 2017, P. 137-151.

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein 2001. Introduction to Algorithms. The MIT Press Cambridge, Massachusetts.

[8] ILOG Cplex. World Wide Web, https://www.ibm.com/tr-tr/products/ilog-cplex-optimization-studio .

[9] TSPLIB. Library of Sample Instances for the TSP. http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html