A New Hybrid Genetic and Simulated Annealing Algorithm to Solve the Traveling Salesman Problem

Younis Elhaddad, Omar Sallabi

***Abstract*-- The Traveling Salesman Problem (TSP) is one of the most widely known non- deterministic polynomial (NP-hard) problems. It is used to check the efficacy of any combinatorial optimization method, and is often used as a testbed. This paper proposes a new Hybrid Genetic and Simulated Annealing Algorithm (HGSAA) to solve the TSP. The basic idea behind the proposed HGSAA is that the genetic algorithm (GA) starts its process with its new techniques. When it seems that the GA is stuck after 20 consecutive generations, the hybrid algorithm switches the population to the simulated annealing (SA) which allows uphill jumps to a higher-cost solution in order to avoid getting trapped in local minima. After several SA iterations; the new population is returned to GA. The proposed algorithm was tested using benchmark datasets for symmetric TSPs from TSPLIB, and it provided good results within a reasonable time.**

Keywords*-- Genetic Algorithm, Simulated Annealing, and Traveling Salesman Problem

I. Introduction

The TSP is stated as: for a given complete graph, G, with a set of vertices, V, a set of edges, E, and a cost, c_{ij}, associated with each edge in E. T, where the value c_{ij} is the cost incurred when traversing from vertex, $i \in V$ to vertex, $j \in V$, a solution to the TSP must return the cheapest Hamiltonian cycle of G. Genetic algorithms and SA have been used to solve optimization problems. Current GAs may not produce optimal solutions, or if they do, they will not do so within a reasonable time [1]. Simulated annealing proceeds with only one candidate solution all the time, and therefore does not build up an overall view of the search space. In addition, it is sequential in nature, which makes it slow. According to previous studies of the optimization algorithms [2, 3], high-quality results cannot be obtained by distinct algorithm within a reasonable time, especially with large TSPs. Therefore, many researchers thought of combining two or more algorithms in order to improve solution quality and reduce execution time.

Younis Elhaddad, with Department of computer science Faculty of information Technology, Garyounis University, Benghazi, Libya,younis_hadded@garyounis.edu.

Omar Sallabi, with Department of computer science Faculty of information Technology, Garyounis University, Benghazi, Libya, Osallabi@garyounis.edu

This paper introduces a new hybrid algorithm by combining both the SA and GAs, in order to help each other overcome their problems to obtain the best results in the shortest time. The basic idea behind the proposed HGSAA is to overcome the problem of GA when it is stuck at any early local minima, by switching to SA, which has a better chance of jumping over this problem with its hill-climbing behavior.

II. Genetic Algorithm

The basic principles of GAs were introduced by Holland in 1975 [4]. The GA optimization and search technique emerged from a study of biological evolution [3]. Genetic algorithms operate on populations of potential solutions that are referred to as chromosomes. Each chromosome represents a set of parameters for a given problem. The chromosomes are evaluated according to a fitness function to select the best solutions for a recombination process, which produces new chromosomes. The new, improved chromosomes replace those with poorer solutions. In this way, each new generation gets closer to the optimal solution. This continues for many generations until the termination condition is met. Mutations and different combining strategies ensure that a large range of search space is discovered [3].

III. Simulated annealing algorithm

Simulated annealing algorithm is a general-purpose optimization technique that has been used to solve many combinatorial optimization problems [5]. It can be described formally as follows: start from a random solution x_p, select a neighboring solution x_n and compute the difference in the objective function values,

$\Delta f = f(x_n) - f(x_p)$. If the objective function is improved ($\Delta f < 0$), then replace the present solution x_p by the new one x_n; otherwise accept the solution that decrease the value of the objective function with a probability

$$pr = 1/(1 + e^{-\frac{\Delta f}{t}})$$

where pr is decreased as the algorithm progresses, and where (t) is the temperature or control parameter. This acceptance is achieved by generating a random number (nr) where $(0 \leq rn \leq 1)$ and comparing it against the threshold. If $pr > rn$, then replace the current solution by the new one. The procedure is repeated until a termination condition is satisfied.

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

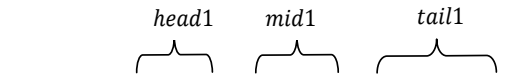
IV. THE PROPOSED HYBRID ALGORITHM (HGSAA)

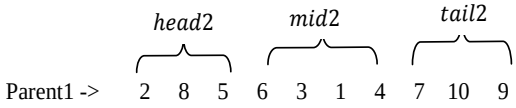
Unlike the classic GA, the HGSAA uses only two individuals as a population. Where the hybrid algorithm starts with a random population, it will be the input of the GA, and multi-crossover is then applied to it to produce 60 different children. The fitness of the parents and their offspring will be calculated, and depending on the results of this calculation a new population will be selected that is the same size as the original population. A partial local optimal mutation operation will then be applied on one individual (according to mutation probability) in order to improve its fitness value. The rearrangement operation is also used on the population; this process is continued until there is no improvement in results after ten consecutive iterations. The memorized population from GA which provides the best result will then be transferred to the SA. The SA processes will be used to improve the results by using the nearest solution technique. If no improvement results are achieved within ten consecutive iterations, then the best memorized population from SA will be moved to the GA to repeat the above process.

A. Multi-crossover operation

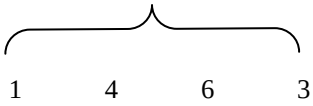
Crossover is the most important operation of GA because it exchanges characteristics between the individuals, and according to that many types of crossover operations are used to produce offspring with different attributes in order to build up an overall view of the search space. Multi-crossover works as mentioned below:

The basic principle of this crossover is two random cut points (p_1 and p_2), a head, containing $(1,2 \ldots\ldots, p_1 - 1)$, the middle containing $(p_1, p_1 + 1, \ldots\ldots, p_2)$, and the tail containing ($p_2 + 1, p_2 + 2, \ldots\ldots\ldots, n$) .The head and tail of each parent are flipped, and then the head of the first parent is swapped with the tail of the other parent, and vice versa. For example, if the selected random two crossover points are $p_1 = 4$ and $p_2 = 7$, and two parents tours are:

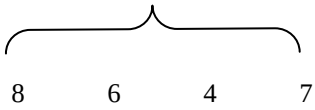




For a valid tour the elements of head2 and tail2 are removed from the parent1 to give mid1



In the same way, elements of head1 and tail1 are removed from the parent2 to give mid2.



Step 1: If the parts (head2, mid1, tail2) are reconnected using all possible permutations, six different children can be obtained(3!).

child1 → 2 8 5 1 4 6 3 7 10 9

In the same way for (head1, mid2, tail1), six other children are produced: i.e.

child2 → 9 1 5 8 6 4 7 2 10 3

Step 2: If the two heads are flipped, as in step 1, 12 new different children are produced:

child3 → 5 8 2 1 4 6 3 7 10 9

*child*4 → 5 1 9 8 6 4 7 2 10 3

Step 3: If the two tails are flipped and as in step 1, 12 new different children are produced:

child5 → 2 8 5 1 4 6 3 9 10 7

*child*6 → 9 1 5 8 6 4 7 3 10 2

Step 4: If the two mid are flipped and as in step 1; 12 new different children are produced:

*child*7 → 2 8 5 3 6 4 1 7 10 9

*child*8 → 9 1 5 7 4 6 8 2 10 3

Step 5: If the two heads and tails are flipped and as in step 1, 12 new different children are produced:

*child*9 → 5 8 2 1 4 6 3 9 10 7

*child*10 → 5 1 9 8 6 4 7 3 10 2

In each step 12 children are produced; therefore $5 \times (3!) \times 2 = 60$ completely different children are produced from just two parents.

B. Selection operation

Using the rank selection, the best two individuals are selected for the next operations in order to reduce the execution time.

C. Mutation

The inversion mutation operation is used here, where random subtour is selected from the second individual then is inversed.

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

D. The Rearrangement operation

This operation is applied to both individuals. $c_{i,j}$ is the cost between the two adjacent cities $city_i$ and $city_j$, where $i = 1,2,3,\ldots,n-1$ and $j = i+1,$. The aim of this operation is to find the greatest (max) value of $c_{i,j}$ among all the adjacent cities on the tour, and then swap $city_i$ with three other cities, one at a time. These cities are located on This operation works in a random matter, and while it may not achieve any improvement after several iterations, it might instead (or is just as likely to) take a big jump and improve the result.

E. Partial local optimal mutation operation

In this operation, the subtour of individuals is selected randomly within the range of

$3 \leq \text{size of subtor} < n/4$. We then find the tour that produces the local minima of this subtour and exchange it with the original subtour. This operation is undertaken on one of the selected individuals after the mutation operation is performed.

V. Experiments and Results

We use instances that are ≥ 100 from TSPLIB [6] and used by J. Zhang and C. Tong [7]; same number of generation for each instance are used in order to compare both results of HGSAA and local search heuristic genetic algorithms (LSHGA) [3]. The HGSAA is implemented on a 3.4 GHz Pentium® D CPU, 512 MB of RAM with Matlab 7.0. The HGSAA was run for 10 trials corresponding to each instance, and the summarized results are shown in Table 1, where column 2 shows the known optimal solutions; column 3 shows the best result obtained by HGSAA; column 4 indicates the number of generations performed, with the number of generations needed to obtain optimal result in parentheses; Column 5 indicates the time in seconds used for each instance, with the time to obtain optimal result in parentheses; column 6 shows the average of the ten results for each instance; column 7 shows the standard deviation of the ten results for each instance; column 8 shows the error ratio between the best result and the optimal, which is calculated according to the following equation:

$$Error = \frac{Best\ result - optimal}{optimal} \times 100.$$

For results at [7], of LSHGA are summarized at Table 2. The notations, PS, CN, OS and error, denotes the population size of the algorithm, the convergence iteration number, the best solution of the LSHGA and the error respectively; error are calculated according to above equation.

Table 1. Results of HGSAA

Problem	Optimal	Best result	Iteration	Time Sec.	Avg.	St. dev.	error
eil101	629	629	400	17(15)	632.9	2.8	0.
ch130	6110	6126	500	26	6146.7	14.8	0.6%
ch150	6528	6528	750 (292)	46(18)	6540.4	13.9	0
korA100	21282	21282	400 (171)	18 (7)	21319.8	32.5	0.
kroA150	26524	26524	800 (407)	53 (27)	26588.7	62.3	0.
kroA200	29368	29382	1100	85	29434.9	45.7	0.23%

Table 2. The result of solution with LSHGA

problem	PS	CN	BS	error
eil101	300	400	640	1.75%
ch130	350	500	6164	0.88%
ch150	400	750	6606	1.19%
korA100	300	400	21296	0.66
kroA150	450	800	26775	0.95%
kroA200	500	1100	29843	1.62%

From Table 1 and Table 2 it is clear that the HGSAA performed better than the LSHGA. The HGSAA can find optimal solution for four instances out of six, while LSHGA cannot find an optimal solution for any of these six instances. The error ratios in both tables indicate that HGSAA is much better than LSHGA.

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

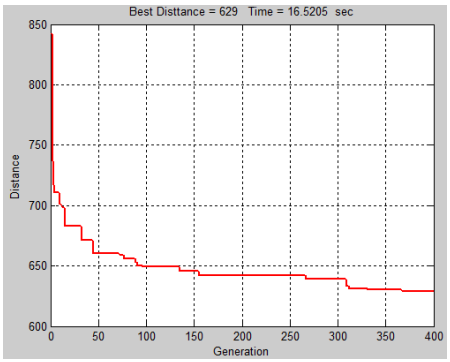


Fig 1. Performance of HGSAA for eil101

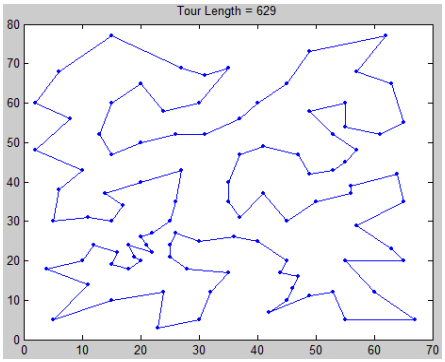


Fig 2. Best result for eil101 problem

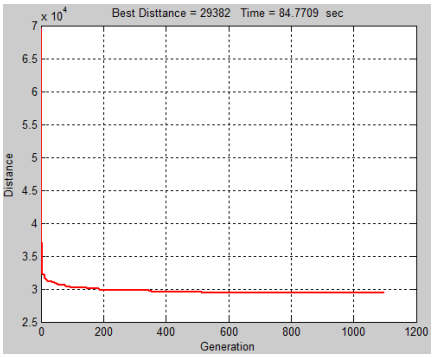


Fig3. Performance of HGSAA for kroA200

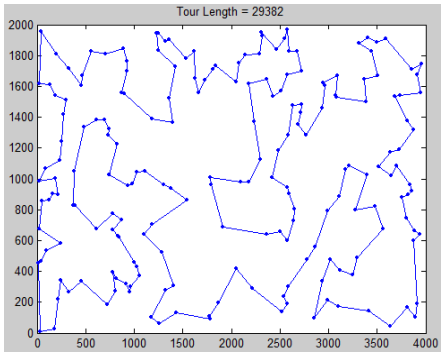


Fig 4. Best result for kroA200

Fig.1 shows the performance of HGSAA for the eil101 problem and Fig.2 shows the best result for the eil101 problem; Fig.3 shows the performance of HGSAA for the kroA200 problem, while Fig.4 shows the best result for the kroA200 problem.

VI. Conclusion

In this paper a new hybrid algorithm (HGSAA) is proposed with new heuristics techniques and operations which improve the convergence rate of the algorithm with better solutions to TSP compared with other algorithms. The hybrid algorithm uses GA and SA, which switches the population to SA in order to allow uphill jumps to a higher-cost solution in order to avoid getting trapped in local minima when the GA stuck after 20 consecutive generations. The proposed hybrid algorithm has been tested using benchmark datasets for symmetric TSPs from TSPLIB, and provides good results within a reasonable time.

References

[1]. P. Larranaga, C.M.H. Kuupers, R.H. Murga, I. Inza and S. Dizdarevic. "Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators," 1999.

[2] P. Merz and B. Freisleben. "Genetic Local Search for the TSP:New results" in proceeding of the IEEE International Conference on Evolutionary Computation, pp,159-164,IEEE Press,1977.

[3] S. Ray, S. Bandyopadhyay and S.K. Pal. "Genetic operators for combinatorial optimization in TSP and microarray gene ordering," Springer Science + Business Media, LLC, 2007.

[4] J. Holland "Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence," The University of Michigan Press, 1975.

[5] Y. Habib, M. Sait and H. Adiche, *"Evolutionary algorithms, simulated annealing and tabu search: a comparative study,"* Engineering Applications of Artificial Intelligence 14, pp. 167-181, 2001

[6] TSPLIB: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html

[7] J. Zhang and C. Tong. "*Solving TSP with Novel Local Search Heuristic Genetic Algorithms*" Fourth International Conference on Natural Computation IEEE 2008

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)