A Comparative Study between the Nearest Neighbor and Genetic Algorithms: A revisit to the Traveling Salesman Problem

Besan A. AlSalibi, Marzieh Babaeian Jelodar and Ibrahim Venkat

***Abstract*—**Traveling Salesman Problem (TSP) is one of the most widely studied optimization problems in computational mathematics. The simplest heuristic approach to solve TSP is the Nearest Neighbor (NN) algorithm. Bio-inspired approaches such as Genetic Algorithms (GA) are providing better performances in solving TSP. GA is based on mimicking the survival of the fittest among species generated by random changes in the gene-structure of chromosomes. The goals of this paper are to explain the concept of symmetric TSP, implementation of TSP using GA and NN, investigating the effects of GA parameters such as mutation rate, crossover rate and population size on GA performance and providing a performance comparison between GA and NN in solving the TSP problem for a small case study.

***Keywords*—**Control Parameters, GA, NN, Symmetric TSP

I. Introduction

THE TSP Problem which is known as one of the most extensively studied problems in optimization [1] is performed by a salesman who travels from a city to all other cities exactly once and returns back to the starting city to find the shortest path. Deterministic approaches for solving TSP, usually test all the possibilities for N city tour which requires N! math additions. The number of total possible paths increases dramatically in terms of powers of N. Among the meta-heuristic algorithms which could to solve the TSP problem, we focus on two typical ones viz., NN and GA. The nearest neighbor algorithm is a greedy algorithm that finds the candidate solution to TSP using simple means. On the other hand GA follows evolutionary principles to solve optimization problems, in our case the TSP. GA is a search heuristic that mimics the process of natural evolution. GA parameters such as mutation rate, crossover rate, population size and maximum number of generations can affect the performance of the algorithm in solving the optimization problems.

This paper is organized as follows: Section 2 presents the concept of TSP. Section 3 describes the implementation of TSP using NN and GA. Section 4 investigates the effects of GA parameters such as mutation rate, crossover rate, population size and maximum number of generations on GA performance. Section 5 provides a performance comparison between GA and NN in solving TSP problems. Finally, Section 6 concludes the paper.

Besan. A. Alsalibi is with the School of Computer Science, universiti Sains Malaysia, Pulau Penang, CO 11800 Malaysia (corresponding author's phone: 0016040176424721; e-mail: besansalipi@gmail.com).

Marzieh. Babaeian Jelodar is with the School of Computer Science, universiti Sains Malaysia, Pulau Penang, CO 11800 Malaysia (e-mail: marzbar.j@gmail.com).

Ibrahim Venkat is with the School of Computer Science, Universiti Sains Malaysia, Pulau Penang, CO 11800 Malaysia (email: ibrahim@cs.usm.my).

II. TSP Concept

TSP is a well known problem in the field of combinatorial optimization. Its concept is deceptively simple, but yet it remains as one of the most challenging problems. It is known as a classical NP-complete problem, which has extremely large search spaces and is very difficult to solve.

The basic definition of TSP problem is: given N cities, a salesman has to travel from his own city to visit each city exactly once and then return to the starting city. The main goal is to find the order of a tour such that the total travelled distance (cost) is minimized. TSP attracts many mathematicians and scientists which have worked on developing efficient solving methods since 1950's. TSP has been employed in application areas such as designing hardware, electronic devices, architecture of computational networks, and so on. TSP problem can be classified into different classes based on the arrangement of distance metrics between the cities. In symmetric TSP, the distance between two cities is the same in each direction, forming an undirected graph. This symmetry halves the number of possible solutions. In contrast to the symmetric version, paths in the Asymmetric TSP may not exist in both directions or the distances might be different, forming a directed graph. Symmetric TSP will be considered in this paper. TSP problem can be represented as a weighted graph [2] G= (N, E) as shown in Fig. 1, where N is the set of n cities and E is the set of edges (paths) fully connecting all cities. Each edge $(i,j) \in E$ is assigned a cost d_{ij}, which is the Euclidean distance between cities i and j. The coordinates of node i are x[i] and y[i]. The distance between i and j in the 2-dimentional case is calculated as in (1):

$$d_{ij} = \sqrt{\left((x[i]-x[j])^2 + (y[i]-y[j])^2\right)} \qquad (1)$$

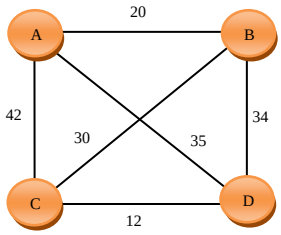


Fig.1 Graph Representation of TSP

III. TSP IMPLEMENTATION

A. Nearest Neighbor

The NN algorithm was first introduced by J. G. Skellam. The work on the NN was continued by P.J. Clark and F.C. Evans. The NN method compares the distribution of distances that occur from a data point to its nearest neighbour in a given data set with a randomly distributed data set. NN algorithm is a greedy algorithm that follows a very simple greedy procedure for solving TSP. The first strategy that has been introduced and used for solving TSP problem was NN algorithm [3]. NN starts with a randomly chosen city and it adds the nearest but not yet visited city to the last city in the tour until all the cities are visited.

The algorithm steps are described in the following:

1) A vertex will be picked randomly as the current vertex
2) The lightest edge will be chosen that connect current vertex to the nearest unvisited vertex V.
3) Our current vertex is vertex V.
4) Vertex V is marked as visited.
5) If all of the vertices in the domain are visited, then terminate the procedure.

The output of the algorithm is the sequence of all visited vertices. This implies a short tour but not an optimal one. Because of its greediness nature, NN algorithm misses some shorter routes that can easily be detected with human insight. It is possible that NN algorithm will not find a feasible tour even if there is one.

B. Genetic Algorithm

GA which is classified under the evolutionary algorithms was first introduced by John Holland in 1970s. Genetic Algorithm [4] is a search heuristic that mimics the process of natural evolution. GA solves many of the real world problems involving finding optimal parameters. GA has an outstanding performance in optimization. Implementation of GA starts with a population that consists of random chromosomes. These chromosomes are evaluated and reproductive opportunities are allocated. The chromosomes which have better solution to the target problem are chosen to reproduce.

1. Chromosome Encoding

The candidate solutions of TSP problem can be represented by chromosomes that consist of genes. Each chromosome gene indicates a city. The values of these genes and their position in the "gene string" tell the genetic algorithm what solution the individual represents. TSP problem has been solved in different chromosome representations such as binary strings and matrices by the GA algorithm. The binary and matrix based representations usually use the binary alphabets for the tour representation. A TSP tour can be encoded in terms of binary strings but this method is not much suitable because of the ordering dependencies. The TSP solution is represented often with the permutation representation, since it requires representing the cities without appearing twice. Two representation methods which are the adjacency representation and path representation are known for representing the TSP tour. The most natural representation of a tour is denominated by path representation. A tour is a list of n cities, and if the city i is the j-th element of the list, city i will be the j-th city to be visited. Therefore, an example tour, 3-2-4-1-7-5-8-6 is simply represented as it is, that is by 3-2-4-1-7-5-8-6. Path representation is considered in this paper for encoding the chromosomes.

2. Genetic Operators of GA

To make the GA work efficiently, genetic operators are needed. Generally, there are three basic operators: crossover operator, selection operator and mutation operator. These operators will be described in the following:

2.1. Crossover Operator

The crossover operator is the method for combining selected parents into new offspring. For TSP we have used the Ordered Crossover (OX). It chooses a substring randomly from a parent and produces a proto-child by copying the substring into the corresponding position of it. The cities which are in the substring from the 2nd parent will be deleted. The cities will be placed in the unfixed positions of the proto-child from left to right according to the order of the sequence to produce an offspring.

2.2. Mutation Operator

Mutation operator is intended to maintain the diversity of individuals in the population. For TSP problem, the mutation operator is only used for a single individual. In this paper, swap mutation is used. It works in the following way; the swap mutation picks two alleles randomly and swaps their positions. For example if we have: 1 2 3 4 5 6 7 8 9, after the swap we will have: 1 5 3 4 2 6 7 8 9.

2.3. Selection Operator

The selection operation is responsible for choosing the next generation. It has the ability to provide a mechanism for gaining promising solutions and passing them to the next generations. There are different types of selection operations such as Tournament, Roulette-wheel, Top Percent, Best and Random. In this implementation, we have used the Roulette-wheel concept. In this concept the chance of a chromosome getting selected is proportional to its fitness (or rank). The concept of the fittest comes into play from this part.

2.4. Stopping Criteria

The implementation of TSP problem using GA algorithm has a number of stopping operations. For our implementation we used a combination of three stopping operations which are as follows:

1) Reaching the max number of generations.
2) Getting the optimal solution.
3) Getting the same cost for 1000 generations.

2.5. GA steps

The general steps of GA are as follows:

1) Generate random population of n trips.
2) Evaluate the fitness of each trip.
3) Create a new population.
   - Select two parents from a population according to their fitness.
   - If there is an order crossover operation performed between the two parents, two children will be generated, and if there is no crossover performed the children will be copied from the parents.
   - If the swap mutation operation is performed this will change the trip from the output of step B, if not the children will be the same as step B.
   - Add new children in a new population.
4) Evaluate the fitness of each trip.
5) If the stopping criteria is satisfied, the algorithm stops and shows the best trip, if not it will start over from step 3 and continues the iteration.

IV. The Effects of Parameter Settings on GA Performance

The performance of GA algorithm largely depends on the appropriate selection of its parameter values; including crossover mechanism, probability of crossover, maximum number of generations, population size and mutation rate. To investigate how each parameter affects the performance of GA, the other parameters should be fixed while changing the value of the active parameter [5].

A. The Effect of Mutation Rate

As shown in Fig. 2, changing the mutation rate from 0.01 to 0.9 affects both the fitness (cost) and the execution time of the GA algorithm. The best solution with the smallest amount of time was obtained while adjusting the mutation rate to 0.4

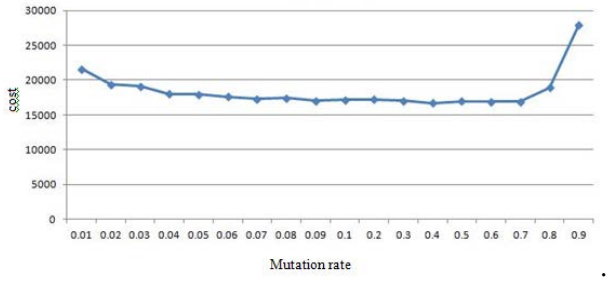


Fig.2 Plot of fitness against mutation rate

Increasing the mutation rate to a value greater than 0.4, negatively effects the fitness and the execution time. These results were obtained using *rd400* benchmark data set with a population size of 1000, maximum number of generations being set to 10,000 and a crossover rate of 0.5.

A. The Effect of Population Size

As shown in Fig. 3, changing the population size from 0 to 10,000 affects both the fitness (cost) and the execution time of the GA algorithm. While the population size is increasing, the cost is decreasing and the Solution becomes closer to the optimal one.

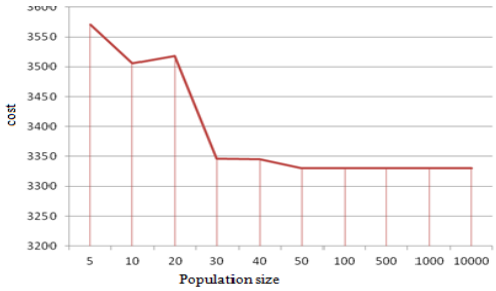


Fig.3 Plot of path cost against population size (*burma14 dataset*)

It is noticeable that for the *burma14* benchmark dataset which contains 14 cities, the best solution was obtained when the population size was 50. Increasing the population size for a value more than 50 has no effect on the cost. For small number of cities (less than 50), the population size of 50 is a good choice for both the execution time and the cost. In contrast to *burma14*, the performance of GA using *rd400* dataset is improving while the population size is increasing at the expense of the execution time which is increased as shown in Fig.4.

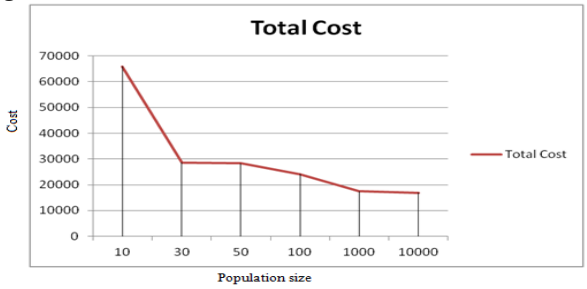


Fig.4 Plot of path cost against population size (rd400 dataset)

B. The Effect of Crossover Rate

As shown in Fig.5 and Fig.6, changing the crossover rate from 0.01 to 0.9 affects both the fitness and the execution time of GA. The results have been obtained using *rd400* and *burma14* datasets respectively with a mutation rate of 0.4, population size of 1000 and number of generations of 10,000. It is noticeable from these two plots that there is a contrast in the crossover rate effect according to the number of cities used. For *rd400* dataset, the best cost is obtained when a crossover rate of 0.5 was used. Increasing the rate for a value more than 0.5 will negatively affect the performance. In contrast to *rd400*, the performance of GA using *burma14* dataset is better when using a smaller crossover rate around 0.03.

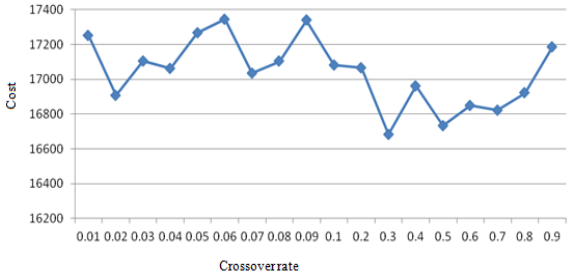


Fig.5 Plot of path cost against crossover rate (*red400 dataset*)

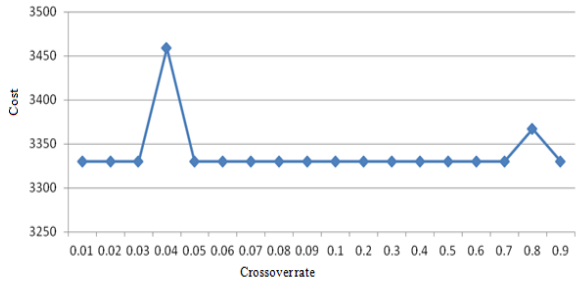


Fig.6 Plot of path cost against crossover rate (*burma14 dataset*)

We have observed that, increasing the crossover for a value more than 0.03 has no effect on the cost but interestingly it increases the execution time. Hence from these results, we recommend that when a tour has small number of cities (less than 50) a crossover rate with the value of 0.03 can be used whereas for a tour with large number of cities (about 400 something), a cross over rate of 0.5 can be used.

Further Results and Discussion

For this study we have used an Intel core I5 laptop with a 2.4 GHz processor and 4 Gigabytes RAM, which runs under Windows 7. Further, we have implemented NN and GA algorithms using C# which is a family of Visual Studio 2010. To demonstrate and test the effectiveness of the algorithms, four benchmark instances varying from 14-cities to 431-cities from the standard TSPLIB library have been considered [5].

C. NN Algorithm

Ten runs of NN algorithm using instances *burma14* and *rd400* from TSPLIB respectively were performed. From Fig.7 and Fig.8 we can see the oscillating nature and the randomness of solutions generated by NN, though it tends to approach towards the optimal solution. Fig.9 shows the best and average solutions generated by the NN algorithm for the four instances. Based on these results, we notice that NN algorithm is very suitable for small datasets which usually contains a small number of cities in which it can generate solutions very close to the optimal one and sometimes better than the optimal solution in a small amount of time.

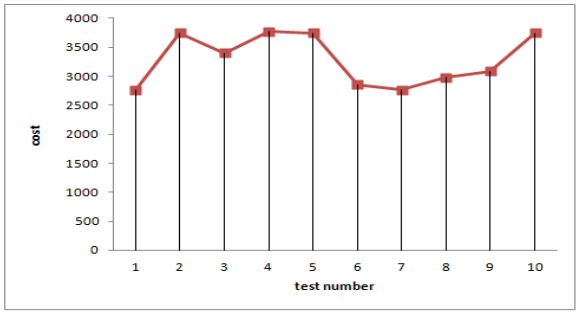


Fig.7 Random solutions generated by NN for burma14 dataset

For a large number of cities, NN generates solutions which are far from the optimal solution. However, when accuracy is not a major constraint, one can consider NN for such datasets because of its faster execution time. The average time of the NN algorithm for all the four instances are less than about one second which indicates that NN algorithm is faster than the GA as shown in Table I.

TABLE I
NN AVERAGE TIME IN SEC

dataset	Average time
burma14	0.00044778
rd400	0.50996
f1417	0.57162
gr431	0.642

Fig.8 Random solutions generated by NN for rd400 dataset

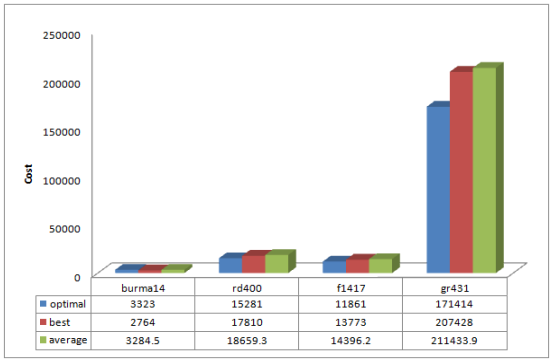


Fig.9 Optimal, best and average solutions generated by NN

D. Performance Comparison between NN and GA

To compare the performance between the NN and GA, we run 10 tests of each algorithm using the four instances and compare the cost and the execution time. For the instance of *burma14* as shown in Fig. 10 and Fig. 12, it can be observed that GA is quite stable in terms of cost as it generates solutions which are much closer to each other.

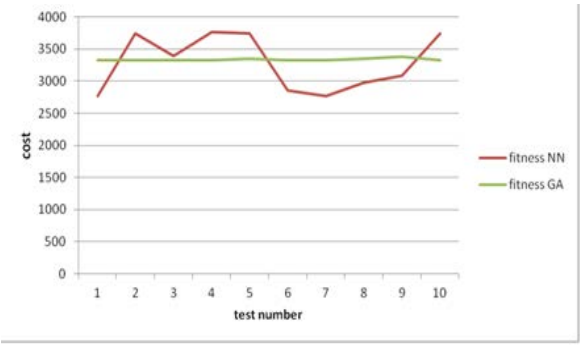
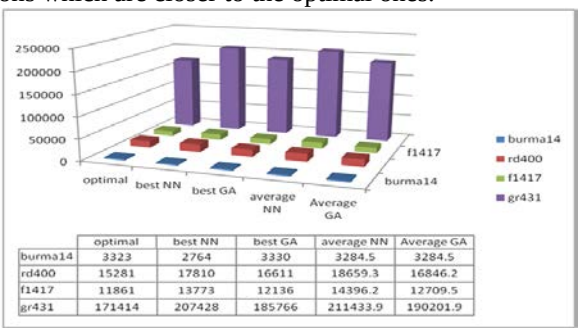


Fig. 10 Cost comparison between NN and GA using burma14 dataset

Fig.11 Cost comparison between NN and GA using rd400 dataset

In contrast to GA, NN algorithm generates a range of solutions where some of them are far from the optimal solution and some are very close to the optimal solution. The runtime of NN algorithm is better than GA.

As we can see in Fig. 11 and Fig.12, GA generates solutions which are closer to the optimal ones.



	optimal	best NN	best GA	average NN	Average GA
burma14	3323	2764	3330	3284.5	3284.5
rd400	15281	17810	16611	18659.3	16846.2
f1417	11861	13773	12136	14396.2	12709.5
gr431	171414	207428	185766	211433.9	190201.9

Fig. 12 Comparison between NN and GA for all datasets

The average errors of NN and GA algorithms have been calculated for the four datasets to examine the average performance for each algorithm using MAE equation as in (2).

$$MAE = \frac{(\text{test solution} - \text{optimal solution})}{\text{optimal solution}} * 100\% \qquad (2)$$

Fig.13 shows that the average error of GA algorithm is much less than NN algorithm for all datasets and this indicates that the performance of GA algorithm is much better than NN algorithm, pertaining to the instances we have used. Further, Table II shows the average running time taken by GA on the four various instances taken into consideration for our experiments.

TABLE II
GA AVERAGE TIME IN SEC

dataset	Average time
burma14	0.124
rd400	216.51
f1417	268.41
gr431	418.01

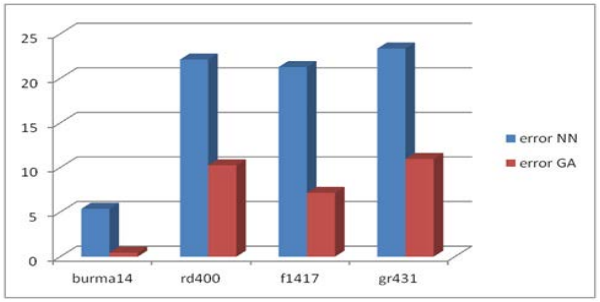


Fig. 13 Average errors of NN and GA

V. CONCLUSION

In this paper, we have investigated the TSP problem using two heuristic algorithms which are GA and NN using a small case study. The performance of the NN algorithm depends mainly on the number of cities in the sense that it performs very well for a small number of cities less than 50. However, for large number of cities its performance is lower and the solutions are not very accurate. Pertaining to our case study we also report the performance of GA in terms of some of its typical parameters. Finally, we have compared the performance of the two algorithms in terms of cost and runtime.

ACKNOWLEDGMENT

We would like to thank Dr. Wong Li Pei for providing us a guest lecture on the TSP problem.

REFERENCES

[1] G. Laporte, "The Traveling Salesman Problem: An overview of exact and approximate algorithms", *Elsevier Science Publishers*, vol. 59, pp. 231–247, July 1995.

[2] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, "The Traveling Salesman Problem: A Computational Study", *Princeton University Press*, 2007, pp 2-3.

[3] C. Chauhan, R. Gupta, K. Pathak, "Survey of Methods of Solving TSP along with its Implementation using Dynamic Programming Approach" *International Journal of Computer Applications*, vol. 52, no. 4, August 2012.

[4] G. R. Harik, F. G. Lobo and D. E. Goldberg, "The Compact Genetic Algorithm" *IEEE Trans on Evolutionary Computation*, Vol. 3, no. 4, November 1999.

[5] J.J. Grefenstette, "Optimization of Control Parameters for Genetic Algorithms", *IEEE Trans. On Systems, Man and Cybernetics*, vol. SMC-16, no. pp 122-128, 1986.

[6] G. Reinelt, TSPLIB http://www.iwr.uniheidelberg.de/groups/comopt/software/TSPLIB95/