© International Journal of Combinatorial Optimization Problems and Informatics, Vol. 2, No. 1, Jan-April 2011, pp. 2-13. ISSN: 2007-1558.

New Variations of Order Crossover for Travelling Salesman Problem

Kusum Deep, Hadush Mebrahtu
Department of Mathematics, Indian Institute of Technology, Roorkee, India
kusumfma@iitr.ernet.in, hhadumk@gmail.com

Abstract. Davis presented the order crossover operator for solving the travelling salesman problem (TSP) using genetic algorithm and has described two basic variants of this order crossover operator. In order to improve the efficiency and validity of these variants, in this paper three new variations of the order crossover operator are presented. These are programmed in C++ and implemented on a set of benchmark test problems taken from TSPLIB. The analysis of results based on the numerical and graphical analysis indicated that the new variations of the order crossover proposed in this paper have a definite supremacy over the existing variants for TSP with number of cities less than or equal to 100 as well as greater than 100 for some instances.

Keywords: Variations of Order crossover; The Travelling salesman problem; Genetic algorithms

1 Introduction

One of the most difficult problems in mathematics is the travelling salesman problem(TSP). It is a well known NP-hard combinatorial problem [1] and as a result of this any problem which belongs to the NP-class can be formulated as a TSP problem. It has attracted the attention of many researchers and remains an active research area since a large number of real world problems can be modeled by TSP.

The application of TSP is varied , for example on some instances of vehicle routing, computer wiring, scheduling of jobs on a single machine, over hauling gas turbine engines in aircrafts etc makes TSP a popular problem. Following [2] it can be stated mathematically as follows:

If a travelling salesman were to visit a given finite set of cities with cost (c_{ij}) of travelling from city *i* to *j* exactly once and return to its original city, then which tour would obtain the minimum cost? Formally let {1, 2, 3, …, *n*} be labels of the *n* cities , then the total cost *TC* of a TSP tour is

$$TC(n) = \sum_{i=1}^{n-1} C_{i,i+1} + C_{n,1} \tag{1}$$

Here we consider $c_{ij} = c_{ji}$ which is a symmetric TSP.

A method that provides an exact optimal solution to a problem is called exact method. An implicit way of solving the TSP is simply to list all the feasible solutions, evaluate their objective function values and pick out the best. However it is obvious that this “exhaustive search” is grossly inefficient and impracticable because of vast number of possible solutions to the TSP even for problem of moderate size. Since practical applications require solving larger problems, hence emphasis has shifted from the aim of finding exactly optimal solutions to TSP, to the aim of getting, heuristically, ‘good solutions’ in reasonable time and ‘establishing the degree of goodness’. Some of the new heuristic approaches that are being adapted to solve the NP-hard TSP are Genetic algorithm [3], Particle swarm [4], Ant Colony [5] and Simulated Annealing [6].

Genetic algorithms (GAs)[7] are population based search techniques which mimics the principles of natural selection and natural genetics laid by Charles Darwin. Since Holland introduced the genetic algorithm (GA) in the early 1970’s, many researchers have become interested in it as a new method of solving real life problems. As it is a promising heuristic approach to locate near optimal solution in large spaces, it is not surprising that it is a target of many researchers.

Received July 10, 2010 / Accepted Oct 20, 2010

In Genetic algorithm, a population of potential solutions termed as chromosomes/individuals are evolved over successive generations using a set of genetic operators called selection, crossover and mutation. First of all, based on some criteria, every chromosome is assigned a fitness value (in our case the above formulation of TSP) and then the selection operator is applied to choose relatively fit chromosomes to be part of the reproduction process. In reproduction process new individuals are created through application of operators. Large number of operators has been developed for improving the performance of GA, because the performance of algorithm depends on the ability of these operators. One of the operators, Crossover operator, blends the genetic information between chromosomes to explore the search space, where as mutation operator is used to maintain adequate diversity in the population of chromosomes and avoid premature convergence.

Crossover operator plays a critical role in solving TSP by performing an exchange of information between individuals during generation and helps in obtaining global optimal solution. In the last 2-3 decades several crossovers for TSP have been proposed. One of them is the order crossover (Ox) of Davis [8] and [9] that has two basic variants Ox_1 and Ox_2.

TSP as opposed to most problems tackled by GAs is a pure ordering problem [3]. Accordingly specialized permutation operators like crossover must be developed for this problem. In [10] it is developed a coding scheme for one point crossover and generated feasible offspring, but the sequencing information in the two parent chromosomes is not well transferred to off springs and the resulting search becomes close to a random search.TSP tours can have different representations, such as ordinal, path, adjacency and binary matrix representation. From these, the path representation is the most natural representation of a tour and we will focus on it. For example a tour

7 – 1 – 6 – 2 – 5 – 8 – 9 – 3 – 4 is represented simply as
(7 1 6 2 5 8 9 3 4)

In terms of path representation there are two basic groups of crossover operators proposed by researchers that preserves either the relative order or the absolute position of cities/chromosomes/ from the parent chromosomes. Cycle and partially matched crossovers are operators that preserve absolute position. While modified, order and order based crossovers are some examples of crossovers that preserves relative order of cities/chromosomes/.

In this paper we propose three new variations of order crossover operator and evaluate their performances over the existing variants. This paper is organized as follows: Section 2 is a literature review on crossover operators for TSP. Section 3 explains the existing variants of order crossover. Section 4 describes the variations proposed. In section 5, we will have experimental results for comparison. Finally conclusion and future work will be presented in section 6.

2 Literature review on crossover operators for TSP

Several heuristic and exact algorithms have been developed in the field of optimization to solve TSP. The exact algorithms [11] are designed to find the optimal solution, that is, the tour of minimal length. They are computationally expensive since they consider all solutions in order to identify the optimum.

Branch and bound algorithms are also commonly used to find an optimal solution to TSP which are good for less than or equal 70 cities. Running an exact algorithm for hours on a powerful computer is not cost- effective if a solution within a few percent of the optimal can be found quickly on small computers. Accordingly heuristic algorithms are often preferred to exact algorithms to solve TSP. Generally, TSP heuristics can be classified as tour construction procedures [12], tour improvement procedures [13] and composite procedures [14]. In this research we used evolutionary algorithms especially genetic algorithm to solve the travelling salesman problem (TSP).

According to [15] evolutionary algorithms are randomized search techniques aimed at stimulating the natural evolution of asexual species. In this model individuals were created via random mutation to the existing individuals. Holland and his students extended this model by allowing “sexual reproduction” i.e. the combination or crossover of genetic materials from two parents to create a new offspring. These algorithms were Genetic algorithms but they were not designed to solve combinatorial optimization problems like TSP. Consequently improving these algorithms were needed. Off course fitness function with penalty terms and repair operators to transform infeasible solutions to feasible ones were proposed by [16] to alleviate these problems. However the approaches were designed for very specific application domains and are not relevant in a TSP context.

The TSP as opposed to most problems tackled by GAs is a pure ordering problem. Accordingly specialized permutation operators like crossover must be developed for this problem. In [17] developed a coding scheme for one point crossover and generated feasible offspring, but the sequencing information in the two parent chromosomes is not well transferred to off springs and the resulting search becomes close to a random search.

In [18] partially matched crossover (PMX) was developed that preserves absolute position using two cut points in parents. This operator first randomly selects two cut points on both parents. In order to create an offspring, the substring between the two cut points in the first parent replaces the corresponding substring in the second parent. Then, the inverse replacement is applied outside of the cut points, in order to eliminate duplicates and recover all cities.

Cycle crossover which focuses on subsets of cities that occupy the same subset of positions in both parents was also proposed [19]. In this crossover a subset of cities are copied from the first parent to the offspring (at the same positions), and the remaining positions are filled with the cities of the second parent. In this way, the position of each city is inherited from one of the two parents. However, many edges can be broken in the process, because the initial subset of cities is not necessarily located at consecutive positions in the parent tours. In relation to order preserving crossover operators the following were proposed.

A modified crossover which is an extension of the one- point crossover for permutation problems is proposed in [8]. By [8] and [19] an order crossover had been proposed by extending the modified crossover by two cut points and builds offspring by choosing a subsequence of a tour from one parent and preserving the relative order of chromosome from the other parent.

Order based and position based crossover operators were also proposed [20].For order based crossover, first a subset of cities is selected in the first parent. In the offspring, these cities appear in the same order as in the first parent, but at positions taken from the second parent. Then, the remaining positions are filled with the cities of the second parent. While in position based crossover a subset of positions is selected in the first parent. Then, the cities found at these positions are copied to the offspring (at the same positions). The other positions are filled with the remaining cities, in the same relative order as in the second parent. Studies [19] and [21] demonstrated that order preserving crossover were mostly superior to operators preserving absolute position. Similarly Alternate edge and heuristic crossover [10], Edge recombination crossover [22] for adjacency representation that uses an edge to construct an offspring that inherits as much information as possible from the parent structures and matrix based crossover [23] as matrix representation have been studied. Moon proposed a new crossover operator named Moon Crossover (MX), which mimics the changes of the moon such as waxing moon → half moon → gibbous → full moon. Performance of MX operator and OX (order crossover) operator is reported to be almost same.

3 The Order crossover

From the fact that the order of cities (not the position of cities) is important for tours we analyze the order crossover which was proposed by [8] for its simplicity and convenience to application. The order crossover has two basic variants.

3.1 The first variant of order crossover (Ox_1)

The first variant is the standard crossover (Ox_1) [8] that build off springs by choosing a subsequence of a tour from one parent and preserving the relative order of cities from the other parent. For example, consider two parents p_1 and p_2 (with two points marked by "│")

p_1 = (1 2 │3 4 5 │ 6 7 8 9) and

p_2 = (8 5 │ 7 1 2 │ 4 9 3 6)

Produce off springs in the following way. First the segments between the cut points are copied in to off springs.

O_1 = (- - │3 4 5 │ - - - -) and

O_2 = (- - │7 1 2 │ - - - -)

Then, starting from the second cut point of one parent, the cities from the other parent are copied in the same order, omitting those which already exist. Reaching the end of the string /chromosome/ we continue from the first place of the string.

The sequence of cities in the second parent (from the second cut point) is

4 9 3 6 8 5 7 1 2

From this, after removal of cities 3, 4 and 5 which are already in the first offspring we get

9 6 8 7 1 2

Placing this sequence in the first offspring (starting from the second cut point, we have

$O_1 = (1\ 2 \mid 3\ 4\ 5 \mid 9\ 6\ 8\ 7)$

Similarly $O_2 = (4\ 5 \mid 7\ 1\ 2 \mid 6\ 8\ 9\ 3)$.

3.2 The second variant of order crossover (Ox_2)

The second variant of order crossover (Ox_2) proposed by Davis [9] is similar as to cut point selection of the first (Ox_1), except the repairing procedure. Two cut points are selected and the elements between them are copied. The rest elements are copied from the beginning of the second parent respecting their relative order omitting those which already exist in the first parent.

Consider the above example.

$O_1 = (-\ - \mid 3\ 4\ 5 \mid -\ -\ -\ -)$ and

$O_2 = (-\ - \mid 7\ 1\ 2 \mid -\ -\ -\ -)$

The sequence of cities in the second parent is

4 5 7 1 2 6 8 9 3

Removing 3,4 and 5 we get

7 1 2 6 8 9

Placing this sequence in the first offspring from left to right according to the order we have

$O_1 = (7\ 1\ 3\ 4\ 5\ 2\ 6\ 8\ 9)$

Similarly $O_2 = (3\ 4\ 7\ 1\ 2\ 5\ 6\ 8\ 9)$

4 Proposed variations of order crossover

The idea of variations of order crossover (VOX) is inspired from the fact Davis used cut point positions. Why does he use the same cut point positions in both parents? The variations are designed using cut point analysis on the standard ordered crossover operator Ox_1.

4.1 The first variation Ox_3

In this variation the cut points in both parents are at different positions, but size of substrings between the cut points in both parents is the same. Obviously Ox_1 is a special case of Ox_3. The repairing procedure is the same as Ox_1.For example consider

$p_1 = (8\ 3\ 1\ 5 \mid 2\ 7\ 6 \mid\ 9\ 4)$ and

p_2 = (1 │ 9 8 3 │ 6 5 2 4 7)

Replacing 5 4 1 9 8 3 starting from the second cut point in p_1 we have the first offspring as

O_1 = (1 9 8 3 2 7 6 5 4)

Replacing 4 1 5 2 7 6 starting from the second cut point in p_2 we have the second offspring as

O_2 = (6 9 8 3 4 1 5 2 7)

4.2 The second variation Ox_4

Here not only the positions of the cut points are different but the sizes of the substrings between the cut points in both parents are also different. The remaining steps are similar to Ox_1.Consider the following example

p_1 = (7 1 6 2 │ 5 8 9 │ 3 4) and

p_2 = (3 │ 8 4 1 9 │ 5 6 7 2)

Replacing 6 7 2 3 4 1 after the second cut point in p_1 and 3 7 6 2 5 after the second cut point in p_2 we have the two off springs as

O_1 = (2 3 4 1 5 8 9 6 7)

O_2 = (5 8 4 1 9 3 7 6 2)

4.3 The third variation Ox_5

The only difference from Ox_1 is, here we use two pairs of cut points. The rest is the same as Ox_1.For example let us Consider p_1 = (1 │ 7 3 │ 9 5 │ 4 2 6│ 8) and

p_2 = (2 │ 4 6 │ 8 1 │ 3 5 7│ 9)

Replacing 9 8 1 5 after the fourth cut point in p_1 and 8 1 9 2 after the fourth cut point in p_2 we have the two off springs as

O_1 = (8 7 3 1 5 4 2 6 9) and

O_2 = (1 4 6 9 2 3 5 7 8)

The pseudo code for the GA-TSP using the variations can be described as follows.

```
begin GA-TSP

    Create initial population with Nearest-Neighbor Heuristic
    while generation count < m do
        /*m = max. Number of generations. */
        begin
            Roulette wheel selection
            VOC
            Elitism
            Mutation
            Increment generation count
        end
        Output the best and worst individuals found
end GA-TSP
```

The Pseudo-code for the GA-TSP

5 Experimental results and discussion

In our experiment all computations are performed on a 1.66 GHz processor PC with 2 GB of RAM. The programs were written in the C++ programming language. We tested all the cut point variations on six bench mark problem instances taken from the TSPLIB [24]. They are categorized in to two catagories.The first in Table 1 are those instances whose number of cities is less than or equal to 100, while in Table 2 the number of cities is greater than 100.

For all instances probability of crossover (pc) is 0.9 and probability of mutation (pm) is 0.01 after a continuous exhaustive fine tuning .The mutation used here is inversion mutation after experiments are done to compare with reciprocal exchange mutation. The nearest-neighbor heuristic for creating initial population, have the advantage that they only contain a few severe mistakes, while there are long distances connecting cities with short distances. Therefore since such tours can serve as good starting tours, our initial population is generated using nearest–neighbor initialization strategy [25]. As to the selection we used roulette wheel for the first (0-generation) and then elitism.

Elitism ensures the survival of best individuals so far by preserving them and replacing the old members of the population with new best individuals. In the selection using elitism, we took two best fit individuals/chromosomes according to their fitness after two parents produce two off springs by crossover (two best from four). The maximum generation in all problems was 30000.

The solution quality is measured by calculating the percentage of excess above the optimal value reported in TSPLIB [24].We used the formula given by

$$\text{Excess (\%)} = \frac{\text{Obtained value} - \text{Optimal value}}{\text{Optimal value}} \times 100$$

We calculated the percentage of excess for the best value obtained over the known optimal value of 20 runs. The tables also report the time of convergence (in second) and worst values obtained by the algorithms.

From Table 1 we can observe that Ox_3 and Ox_4 have definitely a better performance than especially the existing variants Ox_1 and Ox_2. From the fact that Ox_3 is best for eil51 and kroA100 and the difference between Ox_3 and Ox_4 is 0.207 for eil76, Ox_3 is dominant for those instances whose number of cities is less than or equal to 100.Interms of time Ox_2 is better in all the three instances. In kroA100 three of the proposed variations outperform the existing variants. In comparison to order crossover with simple inversion mutation (OX-SIM) and Swap-GATSP [26] Ox3, Ox4 and Ox5 have shown a far better result.

For number of cities greater than 100 in the three instances, Ox_5 and Ox_4 have shown a better performance. Especially Ox_5 shows best performance being superior in two instances out of three. One important thing we observed in Table 2 is that the three proposed cut point variations have all performed better than the existing variants of order crossover operator. Even though Ox_3 is neither best nor next best, it still shows a better result than Ox_1 and Ox_2 in three of the instances. In terms of time, even if the convergence time of Ox_2 is better than the others its results are not promising.Ox_5 performed best, while its convergence time is longer relative to others.

Table 1. The experimental results of problems with number of cities less than or equal to100.

Problem and optimal value	**Pop. size**	**Cost and Time /sec**	**Ox_1**	**Ox_2**	**Ox_3**	**Ox_4**	**Ox_5**	**Best**	**Next best**
eil51 (51cities) (426)	50	best	432.8	431.1	**428.9**	**430.2**	442.2	Ox_3 (0.7)	Ox_4 (0.99)
		worst	714.5	779.5	769.6	771.6	865.3		
		time	359	289	357	355	410		
eil76 (76cities) (538)	76	best	555.8	557.5	**554.7**	**554.5**	558.7	Ox_4 (3)	Ox_3 (3.1)
		worst	850.6	841.8	878.5	889.2	1047.0		
		time	767	639	754	761	891		
kroA100 (100 cities) (21282)	100	best	21916.8	21390.8	**21294.4**	**21307.4**	21381.8	Ox_3 (0.058)	Ox_4 (0.12)
		worst	39565.6	37225.4	38520.6	42032.9	43764.0		
		time	1325	1189	1275	1285	1590		

Note: In column 9 and 10 the number inside the brace () is percentage of excess from the optimal.

Table 2. The experimental results for problems with number of cities greater than 100.

Problem and optimal value	**Pop. size**	**Cost and Time /sec**	**Ox_1**	**Ox_2**	**Ox_3**	**Ox_4**	**Ox_5**	**Best**	**Next Best**
eil101 (101cities) (629)	100	best	657.9	675.6	659.9	**653.0**	**651.3**	Ox_5 (3.55)	Ox_4 (3.82)
		worst	1011.1	960.9	1025.9	1016.7	1128		
		time	1309	1111	1275	1272	1494		
lin105 (105cities) 14379	100	best	14587.1	14664.3	14543.8	**14497.7**	**14524.7**	Ox_4 (0.825)	Ox_5 (1.01)
		worst	27588.7	25067.8	26793	26081.6	33701.8		
		time	1362	1126	1281	1250	1466		
rat195 (195cities) (2323)	100	best	2451.5	2418.3	2417	**2410.9**	**2394.7**	Ox_5 (3.08)	Ox_4 (3.78)
		worst	3692.8	3430.5	3856.1	3755.9	4218.4		
		time	2581	2258	2449	2482	3006		

Note: In column 9 and 10 the number inside the brace () is percentage of excess from the optimal.

In order to see the comparative convergence of the problems eil51, eil76, kroA100, eil101, lin105 and rat195 clearly, let us see the following figures which contain graphs of each problem.

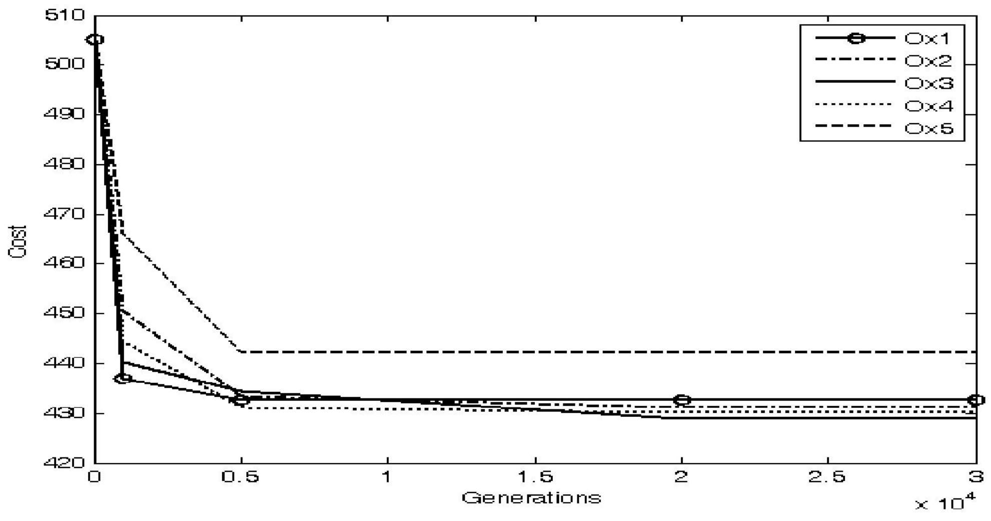

Fig. 1. Comparative convergence of the problem eil51.

In figure 1 above we observe that the performance of Ox_1 is better than the others up to 1000 generations. After 5000 generations, Ox_4 is the best performer. But after 20000 generations Ox_3 performs best with a cost of 428.9 and Ox_4 is next best with a cost of 430.2.In addition to this, the minimum cost of its initial population was 496.4 and now it seems that all are converging after 20,000 generations. Ox_5 does not perform well.

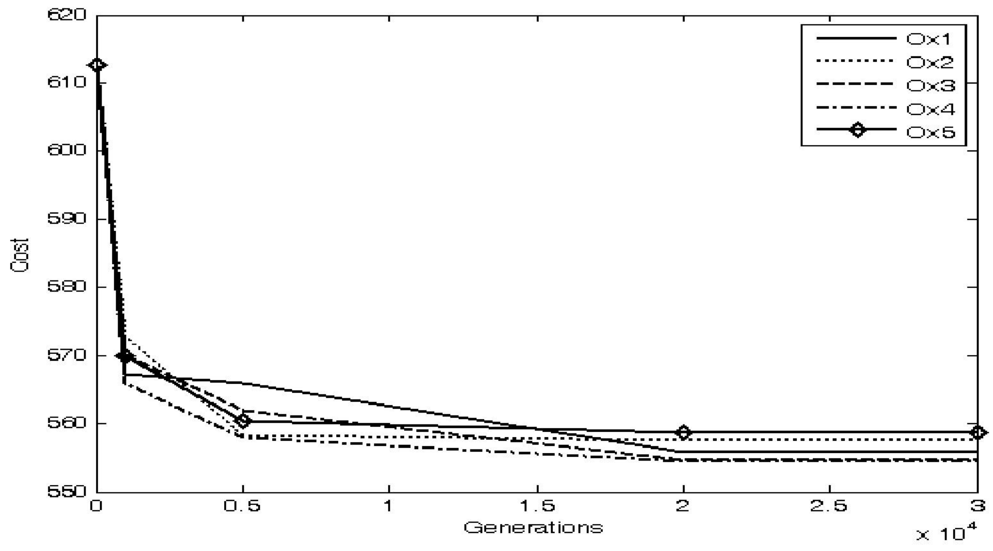

Fig. 2. Comparative convergence of the problem eil76.

Initially the minimum cost of the problem eil76 was 612.6.After using crossover and mutation operators Ox_4 and Ox_1 performed best and next best respectively at the end of 1000 generations. After 20000 generations Ox_4 is best and Ox_3 is next best. In eil76, the difference between the minimum costs obtained using all variations is not that much exaggerated.

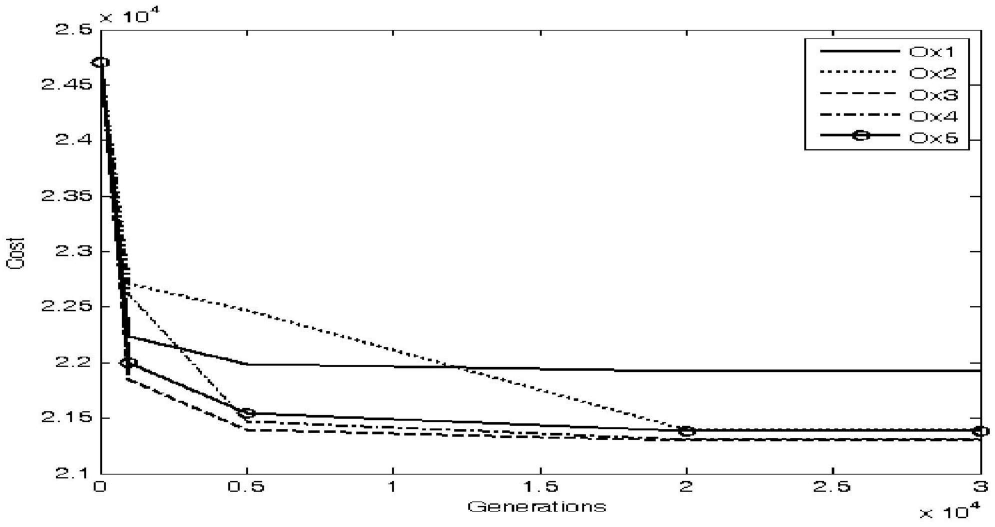

Fig. 3. Comparative convergence of the problem kroA100.

The initial population of the problem kroA100 had a minimum cost of 24698.5. At the end of 1000 generations Ox_3 performed best. While the performance of Ox_1 is better than Ox_4 .After 5000 generations the performance of Ox_3 is still best and Ox_4 is next best. What makes kroA100 different from eil51 and eil76 is that all the new variations Ox_3, Ox_4 andOx_5 performed better than the existing variants.

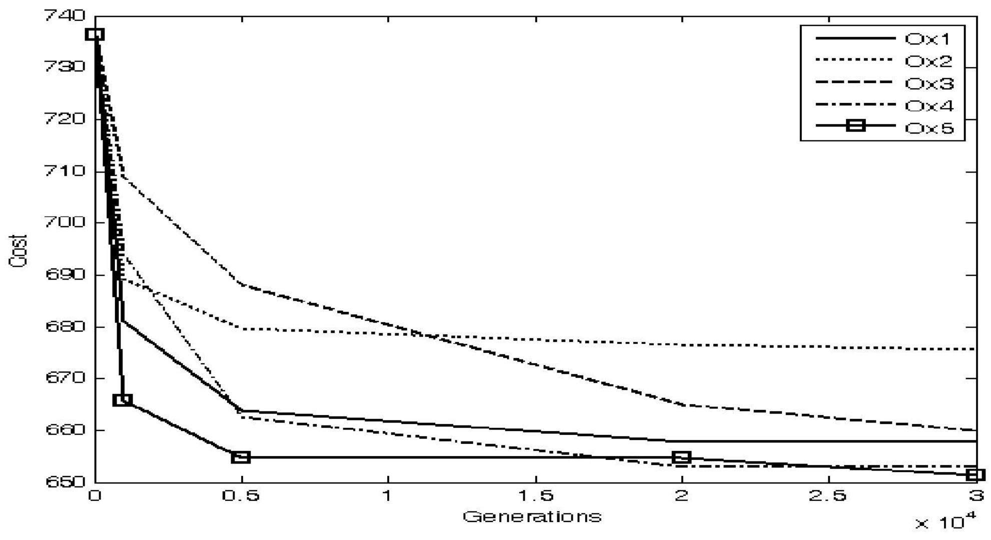

Fig. 4. Comparative convergence of the problem eil101.

eil101 is a hundred one city problem. The minimum cost of its initial population was 736.3.At the end of 1000 generations Ox_5 and Ox_1 performed best and next best respectively. But after 1000 generations until the end of 30000 generations Ox_5 and Ox_4 performed best and next best respectively.

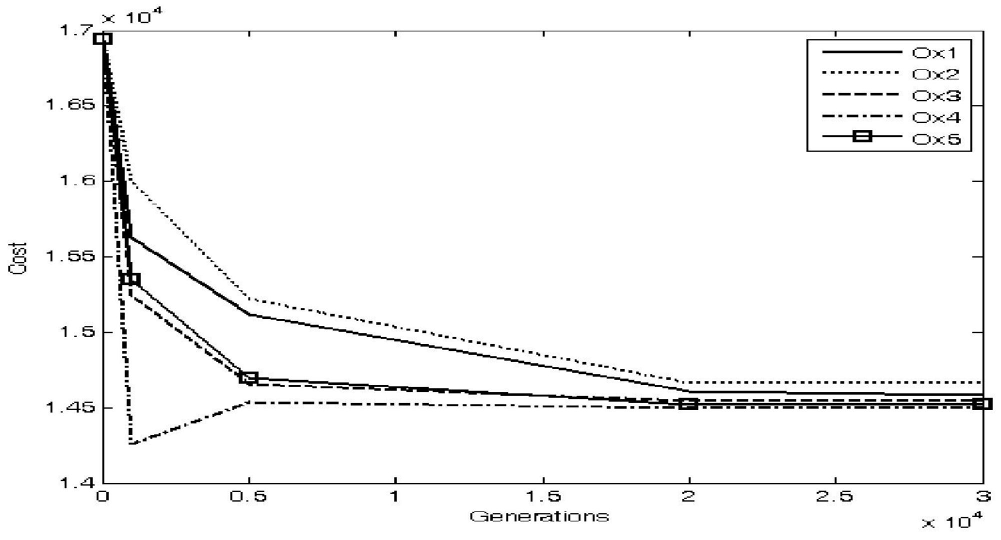

Fig. 5. Comparative convergence of the problem lin105.

For lin105 the minimum cost of its initial population is 16939.4. At the ends 1000,5000,20000 and 30000 generations the new variations of the order crossover (Ox_3, Ox_4 and Ox_5) has shown a clear superiority over the existing variants of order crossover(Ox_1 and Ox_2).The best and next best performers are Ox_4 and Ox_5 respectively.

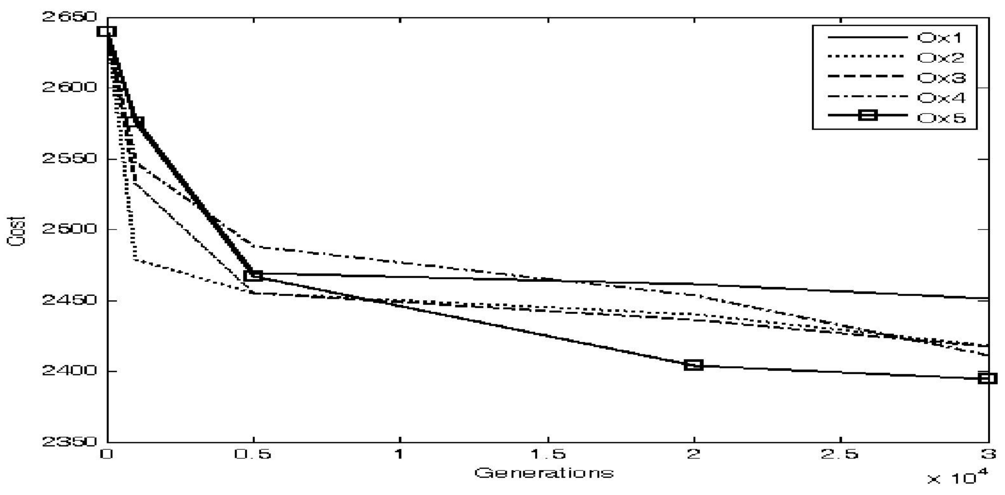

Fig. 6. Comparative convergence of the problem rat195.

Rat195 is a one hundred ninety five city problem. The initial population had a minimum cost of 2639.4.At the end of 1000 generationsOx$_2$ performed best .Ox$_2$ and Ox$_3$ performed best at the end of 5000 generations. But finally at the end of 30000 generations, the best and next performers become Ox$_5$ and Ox$_4$ respectively. Still in rat195 all the new variations of order crossover (Ox$_3$, Ox$_4$ and Ox$_5$) has shown a clear superiority over the existing variants of order crossover (Ox$_1$ and Ox$_2$).

6 Conclusion and future work

In this paper three new variations of order crossover operator are proposed in addition to the existing two basic variants of order crossover operators based on cut point analysis for solving the TSP. From the experimental results it has been observed that the existing variants of order crossover operator were neither best nor next best in all the six instances from the TSP library. Hence since fixing the cut point positions at the same position with equal length in both parents as in Ox_1 and Ox_2 has shown a poor performance in relative to the proposed ones, from now we can use the proposed variations in order to get better results. In particular Ox_3 for number of cities less than or equal 100 and Ox_5 for number of cities greater than 100 are recommended according to the experimental results.

However in order to conclude precisely which cut point variation is best for which instance, more experiment has to be done by considering more instances from the TSP library. By now we are planning in this direction to get better or best optimal values including the ideas of Lin-Kernighan and a modified order crossover operators. In addition to application of these variations to real life problem the behavior of these new variations of order crossover should be analyzed with respect to different mutations.

References

[1] Garey, M.R. and Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co., San Francisco (1979).

[2] Sur-Kolay, S., Banerjee, S. and Murthy, C.A.: Flavours of Travelling Salesman problem In VLSI Design. 1st Indian international conference on Artificial Intelligence, Hyderabad (2003).

[3] Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution programs, Berlin: Springer-Verlag (1992).

[4] Kennedy, J. and Eberhart, R.C.: Swarm Intelligence. Morgan Kauffman publishers, San Francisco (2001).

[5] Dorigo, M. and Gambardella, L.M.: Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1 (1997) 53-66.

[6] Laarhoven, P.V. and Aarts, E.H.L.: Simulated Annealing: Theory and Applications, Kluwer Academic (1987).

[7] Goldberg, D.E.: Genetic Algorithms in Search. Optimization and Machine Learning. Addison-Wesley (1989).

[8] Davis, L.: Applying Adaptive Algorithms to Epistactic Domains. Proceedings of the International Joint Conference on Artificial Intelligence – IJCAI85, Vol. 1 (1985) 162-164.

[9] Davis, L.: Handbook of Genetic Algorithms. New York: Van Nostrand Reinhold (1991).

[10] Grefenstette, J.: Incorporating Problem Specific Knowledge into Genetic Algorithms. Genetic Algorithms and Simulated Annealing, L. Davis Eds, Morgan Kaufmann (1987) 42-60.

[11] Laporte, G.: The Traveling Salesman Problem: An Overview of Exact and Approximate Algorithms. European Journal of Operational Research, Vol. 59, No. 2 (1992) 231-247.

[12] Rosenkrantz, D.J., Sterns, R.E. and Lewis, P.M.: An Analysis of several Heuristics for the Traveling Salesman Problem. SIAM Journal on Computing, Vol. 6, No. 3 (1977) 563-581.

[13] Lin, S. and Kernighan, B.: An Effective Heuristic Algorithm for the Traveling Salesman Problem. Operations Research, Vol. 21, No. 2 (1973) 498-516.

[14] Gendreau, M., Hertz, A. and Laporte, G.: New Insertion and Post-Optimization Procedures for the Traveling Salesman Problem. Operations Research, Vol. 40, No. 6 (1992) 1086-1094.

[15] Fogel, L.J., Owens, A.J. and Walsh, M.J.: Artificial Intelligence through Simulated Evolution. John Wiley (1966).

[16] Richardson, J.T., Palmer, M., Liepins, G.E. and Hilliard, M.: Some Guidelines for Genetic Algorithms with Penalty Functions. Proceedings of the Third International Conference on Genetic Algorithms (1989) 191-197.

[17] Grefenstette, J., Gopal, R., Rosmaita, B.J. and Gucht, D.V.: Genetic Algorithms for the Traveling Salesman Problem. Proceedings of the First International Conference on Genetic Algorithms (1985) 160-168.

[18] Goldberg, D.E. and Lingle, R.: Alleles, Loci and the Traveling Salesman Problem. Proceedings of the First International Conference On Genetic Algorithms (1985) 154-159.

[19] Oliver, I.M., Smith, D.J and Holland, J.R.C.: A Study of Permutation Crossover Operators on the Traveling Salesman Problem. Proceedings of the Second International Conference on Genetic Algorithms (1987) 224-230.

[20] Syswerda, G.: Schedule Optimization using Genetic Algorithms. Handbook of Genetic Algorithms, L. Davis Eds, Van Nostrand Reinhold (1990) 332-349.

[21] Starkweather, T., McDaniel, S., Mathias, K., Whitley, D. and Whitley, C.: A Comparison of Genetic Sequencing Operators. Proceedings of the Fourth International Conference on Genetic Algorithms (1991) 69-76.

[22] Whitley, L.D.: The Genitor Algorithm and Selection pressure: Why Rank-Based Allocation of Reproductive Trials is Best. Proceedings of the Third International Conference on Genetic Algorithms (1989) 116-121.

[23] Homaifar, A., Guan, S. and Liepins, G.: A New Approach on the Traveling Salesman Problem by Genetic Algorithms. Proceedings of the Fifth International Conference on Genetic Algorithms (1993) 460-466.

[24] TSPLIB: (http://www.iwr.uniheidelberg.de/groups/comopt/software/TSPLIB95/

[25] Athitos, V. and Sclaroff, S.: Approximate Nearest Neighbor Retrieval Using Euclidean Embeddings. Workshop of Approximate Nearest Neighbors Methods for Learning and Vision (2003).

[26] Shubhra, S.R., Sanghamitra, B. and Sankar, K.P.: New Operators of Genetic Algorithms for Traveling Salesman Problem. Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04), Vol. 2 (2004) 497-500.