Turk. J. Math. Comput. Sci.
9(2018) 1–13

http://dergipark.gov.tr/tjmcs
http://tjmcs.matder.org.tr

MATDER

An Improved Genetic Algorithm Crossover Operator for Traveling Salesman Problem

Abid Hussain[a,*], Yousaf Shad Muhammad[a], Muhammad Nauman Sajid[b]

[a]*Department of Statistics, Quaid-i-Azam University, Islamabad, Pakistan.*
[b]*Department of Software Engineering, Foundation University, Islamabad, Pakistan.*

Received: 04-03-2018 • Accepted: 06-08-2018

Abstract. The genetic algorithm is one of the best algorithms in order to solve many combinatorial optimization problems, especially traveling salesman problem. The application of genetic algorithms to problems which are not amenable to bit string representation and traditional crossover has been a growing area of interest. One approach has been to represent solutions by permutations of a list, and permutation crossover operators have been introduced to preserve the legality of offspring. There are many existing schemes for permutation representation like PMX, OX, and CX etc. In this paper, we extend the CX scheme which produces healthy offspring based on survival of the fittest theory. Comparison of the proposed operator with other ones for ten benchmarks TSPLIB instances vividly show its pros at the same accuracy level. Also, it requires less time for tuning of genetic parameters and provides narrower confidence intervals on the results than other operators.

2010 AMS Classification: 65C60, 90C05, 97K80.

Keywords: Genetic algorithm, NP-hard, traveling salesman problems, path-representation, crossover operators.

1. Introduction

Genetic algorithms (GAs) are stochastic-based approaches which depend on biological evolutionary processes proposed by Holland [15]. After Holland's work, his students made development in his idea with some new directions and today it is a powerful tool to solving search and optimization problems. A lot of work and applications have been done about GAs in a frequently cited book by Goldberg [13]. The selection criteria, crossover and mutation are three major operators but crossover play a vital role in GAs. A lot of crossover operators have been introduced in literature and all have their own significant importance. With the help of these operators, GAs consumes less memory than other optimization algorithms and work efficiently. GAs work with population of chromosomes that are represented by some underlying parameters set codes. They operate in cycles called generations that produce successive individuals by survival-of-the-fittest selection followed by genetic operators and other die-off. Till the time a desirable condition is achieved, the process is continued to be repeated. The flow chart that how GAs work is given in Figure 1.

The traveling salesman problem (TSP) is one of the typical benchmark, significant, historic and very hard combinatorial optimization problem. TSP was documented by Euler in 1759, whose interest was in solving the knight's tour problem [20]. In the literature of computer science, engineering, operations research, discrete mathematics and graph theory etc., TSP is a fundamental problem. TSP can be described as the minimization of the total distance traveled by

*Corresponding Author
Email addresses: abid0100gmail.com (A. Hussain), yousuf@qau.edu.pk (Y. S. Muhammad), mn.nauman@gmail.com (M. N. Sajid)

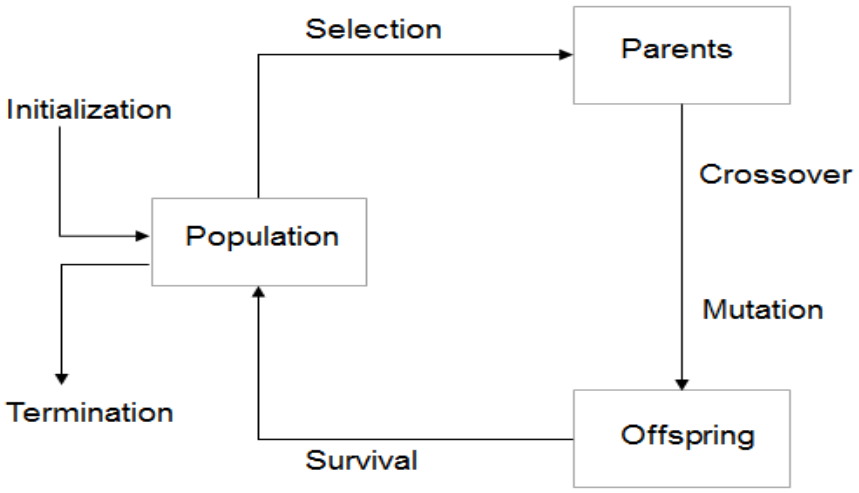

FIGURE 1. Layout of a Typical Genetic Algorithm

touring all cities exactly once and return to depot city. The traveling salesman problems (TSPs) are differentiated into two groups on the basis of the structure of the distance matrix as symmetric and asymmetric.The TSP is symmetric if $c_{ij} = c_{ji}, \forall i, j$, where i and j represent the row and column of a distance (cost) matrix respectively, otherwise asymmetric. The given n cities, a distance matrix $C = [c_{ij}]$ is searched for a permutation $\lambda : \{0, ..., n-1\} \longrightarrow \{0, ..., n-1\}$, where c_{ij} is the distance from city i to j, which minimizes the traveled distance, $f(\lambda, C)$.

$$f(\lambda, C) = \sum_{i=0}^{n-1} d(c_{\lambda(i)}, c_{\lambda(i+1)}) + d(c_{\lambda(n)}, c_{\lambda(1)})$$

where $\lambda(i)$ represents the location of city i in each tour, $d(c_i, c_j)$ is the distance between city i to j and (x_i, x_j) is a specified position of each city in a tour in the plane, and the Euclidean distances of the distance matrix C between the city i and j is expressed as:

$$c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

For n cities, there are $(n-1)!$ possible ways to find the tour after fixing the starting city for asymmetric and its half for symmetric TSP. If we have only 10 cities then 362,880 and 181,440 ways for asymmetric and symmetric TSP respectively. This is the reason to say TSP is a non-deterministic polynomial (NP-hard) problem. This type of problems cannot be solved using traditional optimization approaches like derivative-based methods. To achieve the optimal solution within reasonable time is only by heuristic approaches because they are efficient as handling the NP-hard problems [21].

Over the last three decades, TSP received considerable attention and various approaches are proposed to solve the problem, such as branch-and bound [9], cutting planes [23], 2-opt [21], particle swarm [17], simulated annealing [18], ant colony [7, 8], neural network [2], tabu search [12], and genetic algorithms [4, 16, 20, 22, 24, 28]. Some of these methods are exact, while others are heuristic algorithms. A comprehensive study about GAs approaches are successfully applied to the TSP [10]. A survey of GAs approaches for TSP presented [28]. A new sequential constructive crossover generates high quality solution to the TSP [1]. A new genetic algorithm for asymmetric TSP is proposed [25]. Three new variations for order crossover are presented with improvements [6]. A study presented about an algorithm with MATLAB programming to solve TSP [11]. A study associated with profit based genetic algorithm for TSP and obtaining good results to tested on networks of cities in some voivodships of Poland [27]. A comparative analysis of different crossover operators for TSP and showed partially-mapped crossover gives shortest path in [19]. The latest study to compare the various crossover operators with the modified form of cycle crossover operator for TSP is presented by Hussain et al. [16].

TSP has many applications such as variety of routing and scheduling problems, computer wiring, movement of people, X-ray crystallography [3] and automatic drilling of printed circuit boards and threading of scan cells in a testable Very-Large-Scale-Integrated (VLSI) circuits [29].

Rest of the paper is organized as: Section 2 reprints the background of crossover operators for TSP, proposed a new crossover operators for path representation in Section 3, computational results and discussion in Section 4 and summary in Section 5.

2. Crossover Operators for TSP

In literature, there are binary, path, adjacency, ordinal and matrix crossover representations to solve the TSP with using the GAs. A path representation is our desired because it is most natural and legal way to represent a tour.

2.1. **Path Representation.**

The most natural way to present a legal tour is probably by using path representation. For example, a tour 3⟶4⟶8⟶2⟶7⟶1⟶6⟶5 can be represented simply as (3 4 8 2 7 1 6 5). Since the TSPs in combinatorial with the path representation and the classical crossover operators such as one-point, two-point and uniform crossovers are not suitable. Further classification of path representation as:

2.1.1. *Partially-mapped Crossover Operator.* The partially-mapped crossover (PMX) was proposed by Goldberg et al. [14]. After choosing two random cut points on parents to builds offspring, the portion between cut points, the one parent's string is mapped onto the other parent's string and the remaining information is exchanged. Consider, for an example of the two parents tours with randomly one cut point between 3rd and 4th bits and other cut point between 6th and 7th bits are (the two cut points marked with '|'):

$$P_1 = (3 \quad 4 \quad 8 \,|\, 2 \quad 7 \quad 1 \,|\, 6 \quad 5) \quad \textit{and}$$
$$P_2 = (4 \quad 2 \quad 5 \,|\, 1 \quad 6 \quad 8 \,|\, 3 \quad 7).$$

The mapping sections are between the cut points. In this example, the mappings are $2 \longleftrightarrow 1$, $7 \longleftrightarrow 6$ and $1 \longleftrightarrow 8$. Now two mapping sections are copied with each other to make offspring as:

$$O_1 = (\times \quad \times \quad \times \,|\, 1 \quad 6 \quad 8 \,|\, \times \quad \times) \quad \textit{and}$$
$$O_2 = (\times \quad \times \quad \times \,|\, 2 \quad 7 \quad 1 \,|\, \times \quad \times).$$

Then we can fill further bits (from the original parents), for those which have no conflict as:

$$O_1 = (3 \quad 4 \quad \times \,|\, 1 \quad 6 \quad 8 \,|\, \times \quad 5) \quad \textit{and}$$
$$O_2 = (4 \quad \times \quad 5 \,|\, 2 \quad 7 \quad 1 \,|\, 3 \quad \times).$$

Hence, the first × in the first offspring is 8 which comes from first parent but 8 is already in this offspring, so we check mapping $1 \longleftrightarrow 8$ and see again 1 is exist in this offspring, again check mapping $2 \longleftrightarrow 1$, so 2 occupy at first ×. Similarly, the second × in first offspring is 6 which comes from first parent but 6 is exist in this offspring, check mapping as well $7 \longleftrightarrow 6$, so 7 occupy at second ×. Thus the offspring 1 is:

$$O_1 = (3 \quad 4 \quad 2 \,|\, 1 \quad 6 \quad 8 \,|\, 7 \quad 5).$$

Analogously, we complete second offspring as well:

$$O_2 = (4 \quad 8 \quad 5 \,|\, 2 \quad 7 \quad 1 \,|\, 3 \quad 6).$$

2.1.2. *Order Crossover Operator.* The order crossover (OX) was proposed by Davis [5]. It builds offspring by choosing a sub-tour of a parent and preserving the relative order of bits of the other parent. Consider, for a example of the two parents tours (with randomly two cut points marked by '|'):

$$P_1 = (3 \quad 4 \quad 8 \,|\, 2 \quad 7 \quad 1 \,|\, 6 \quad 5) \quad \textit{and}$$
$$P_2 = (4 \quad 2 \quad 5 \,|\, 1 \quad 6 \quad 8 \,|\, 3 \quad 7).$$

The offspring are produced in the following way. First, the bits are copied down between the cuts with similar way into the offspring, which gives:

$$O_1 = (\times \quad \times \quad \times \,|\, 2 \quad 7 \quad 1 \,|\, \times \quad \times) \quad \textit{and}$$
$$O_2 = (\times \quad \times \quad \times \,|\, 1 \quad 6 \quad 8 \,|\, \times \quad \times).$$

After this, starting from the second cut point of one parent, the bits from the other parent are copied in the same order omitting existing bits. As the sequence of the bits in the second parent from the second cut point is:

$$3 - 7 - 4 - 2 - 5 - 1 - 6 - 8$$

after removal of bits 2, 7 and 1, which are already in the first offspring, the new sequence is:

$$3 - 4 - 5 - 6 - 8.$$

This sequence is placed in the first offspring starting from the second cut point:

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

Analogously, we complete second offspring as well:

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

2.1.3. *Cycle Crossover Operator.* The cycle crossover (CX) operator was first proposed by Oliver et al. [26]. Using this technique to create offspring in such a way that each bit with its position comes from one of the parents. For example, consider the tours of two parents:

$$P_1 = (1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8) \quad \textit{and}$$

$$P_2 = (8 \quad 5 \quad 2 \quad 1 \quad 3 \quad 6 \quad 4 \quad 7)$$

Now its up to us that how we choose the first bit for the offspring to be either from the first or from the second parent. In our example, the first bit of the offspring has to be a 1 or a 8. Let we choose it be 1,

$$O_1 = (1 \quad \times \times \times \times \times \times \times)$$

Now every bit in the offspring should be taken from one of its parents with the same position, it means, further we do not have any choice, so the next bit to be considered must be bit 8, as the bit from the second parent just below the selected bit 1. In first parent this bit is at 8th position, thus

$$O_1 = (1 \times \times \times \times \times \times \ 8)$$

This turn out, implies bit 7, which is the bit of second parent just below the selected bit at 7th position in first parent. Thus

$$O_1 = (1 \times \times \times \times \times 7 \ 8)$$

The next it forced us to put the 4 at 4th position, as

$$O_1 = (1 \times \times 4 \ \times \times 7 \ 8)$$

After this, 1 comes which is already in the list, thus we have completed a cycle and filling the remaining blank positions with the bits of those positions which are in second parent.

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

Similarly the second offspring is:

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

But there is a drawback that some times this technique produces same offspring, for example the following two parents:

$$P_1 = (3 \quad 4 \quad 8 \quad 2 \quad 7 \quad 1 \quad 6 \quad 5) \ \textit{and}$$

$$P_2 = (4 \quad 2 \quad 5 \quad 1 \quad 6 \quad 8 \quad 3 \quad 7)$$

After applying CX technique, the resultant offspring are:

$$O_1 = (3 \quad 4 \quad 8 \quad 2 \quad 7 \quad 1 \quad 6 \quad 5) \ \textit{and}$$

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

The offspring looks similar to their parents.

2.1.4. *Modified-cycle Crossover Operator.* The modified-cycle crossover (CX2) operator was proposed by Hussain et al. [16]. Using this technique to create offspring in such a way that first bit of second parent is the the first bit of first offspring and then search that bit in first parent and choose the exact same location bit from second parent and again search it in first parent and again choose exact same location bit from second parent and that bit is the first bit of the second offspring. For example, consider the tours of two parents:

$$P_1 = (3 \quad 4 \quad 8 \quad 2 \quad 7 \quad 1 \quad 6 \quad 5) \quad \textit{and}$$

$$P_2 = (4 \quad 2 \quad 5 \quad 1 \quad 6 \quad 8 \quad 3 \quad 7).$$

The first bit of second parent is the first bit of first offspring:

$$O_1 = (4 \quad \times \times \times \times \times \times \times).$$

The selected bit is 4 and 4 is searching at second position in first parent and the bit at this position in second parent is 2. For again searching 2 is at fourth position in first parent and 1 is at same position in second parent, so 1 is selected for second offspring as:

$$O_2 = (1 \quad \times \times \times \times \times \times \times).$$

The previous bit was 1 and it is locates at 6th position in first parent and at this position bit is 8 in second parent, so

$$O_1 = (4 \ 8 \quad \times \times \times \times \times \times).$$

And for two moves as below 8 is 5 and below 5 is 7, so

$$O_2 = (1 \ 7 \quad \times \times \times \times \times \times)$$

Hence similarly;

$$O_1 = (4 \quad 8 \quad 6 \quad 2 \quad 5 \quad 3 \quad 1 \quad 7) \quad \textit{and}$$

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

We see that the last bit of second offspring is 3 which was the 1st bit of first parent. Hence this scheme is over within one cycle. Sometimes it is not over within one cycle, consider another example as:

$$P_1 = (1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8) \quad \textit{and}$$

$$P_2 = (2 \quad 7 \quad 5 \quad 8 \quad 4 \quad 1 \quad 6 \quad 3)$$

To work as previous example as:

$$O_1 = (2 \ 1 \ 6 \ 7 \quad \times \times \times \times) \quad \textit{and}$$

$$O_2 = (6 \ 7 \ 2 \ 1 \quad \times \times \times \times)$$

Now stop because the bit 1 has comes in second offspring which was in 1st position of first parent. One cycle is over and before starting other cycle, we match first offspring's bits with second parent or vice versa and left out the existing bits with their position in both parents as:

$$P_1 = (\bullet \ \bullet \quad 3 \quad 4 \quad 5 \quad \bullet \ \bullet \quad 8) \quad \textit{and}$$

$$P_2 = (\bullet \ \bullet \quad 5 \quad 8 \quad 4 \quad \bullet \ \bullet \quad 3)$$

Now filled positions of parents and '$\times$' positions of offspring are considered 1st, 2nd and 3rd positions etc., so we can completes it as usual:

$$O_1 = (2 \quad 1 \quad 6 \quad 7 \,|\, 5 \quad 3 \quad 8 \quad 4) \quad \textit{and}$$

$$O_2 = (6 \quad 7 \quad 2 \quad 1 \,|\, 8 \quad 4 \quad 5 \quad 3)$$

Hence this scheme is over.

3. Proposed Crossover Operator

We extend our previous study CX2, which was the modified form of CX. In CX2, both offspring built on the same time as one bit for first offspring, other bit for second offspring. But in this study, one offspring will be complete first before starting the other one, similarly as original CX pattern. This is the much better improved form of CX, that is reason to suggest it as an improved form of cycle crossover (ICX). We provide the complete scenario of the algorithm step-by-step in following example:

Consider the tours of two parents:

$$P_1 = (3 \quad 4 \quad 8 \quad 2 \quad 7 \quad 1 \quad 6 \quad 5) \quad \textit{and}$$
$$P_2 = (4 \quad 2 \quad 5 \quad 1 \quad 6 \quad 8 \quad 3 \quad 7).$$

The first bit of second parent is the first bit of first offspring as:

$$O_1 = (4 \quad \times \times \times \times \times \times \times).$$

When 4 in first parent was searched out, it was located at second position. Keeping this in view, the bit of same position which is in the other parent is to be chosen for the next bit of offspring.

$$O_1 = (4 \quad 2 \times \times \times \times \times \times).$$

Continuing the process, searching 2 in first parent again which is at fourth location and taking the bit which is located at exact position in second parent as the next bit of first offspring leads to the following result:

$$O_1 = (4 \quad 2 \quad 1 \times \times \times \times \times)$$

Similarly, the process is going on and first offspring comes as:

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

The process is over with in one stage and for more understanding we display it in Figure 2.

Analogously, for second offspring:

$$P_2 = (4 \quad 2 \quad 5 \quad 1 \quad 6 \quad 8 \quad 3 \quad 7)$$
$$P_1 = (3 \quad 4 \quad 8 \quad 2 \quad 7 \quad 1 \quad 6 \quad 5). \quad \textit{and}$$

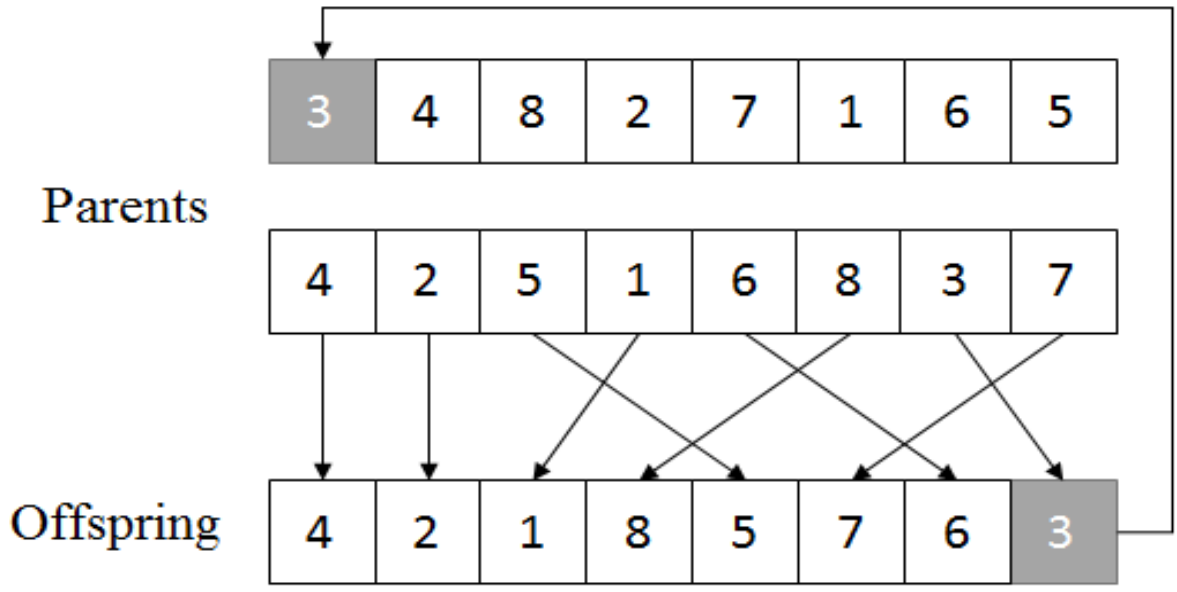

Figure 2. The ICX operator (within one step)

The first bit of first parent is the first bit of second offspring as:

$$O_2 = (3 \ \times \times \times \times \times \times \times).$$

When 3 in second parent was searched out, it was located at seventh position. Keeping this in view, the bit of same position which is in the other parent is to be chosen for the next bit of offspring.

$$O_2 = (3 \ \ 6 \times \times \times \times \times \times).$$

Similarly, the process is going on and second offspring comes as:

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

The last bit of first offspring must be the first bit of first parent (highlighted with gay color in Figure 2) and similar condition must exist in second offspring and parent. Such conditions are not met i.e. the first bit of first (second) parent is come in the first (second) offspring before the ending bit (highlighted with gray color in Figure 3), then the algorithm will go in the following way (more stages):

For this, we take two new tours as:

$$P_1 = (1 \ \ 2 \ \ 3 \ \ 4 \ \ 5 \ \ 6 \ \ 7 \ \ 8) \quad \textit{and}$$

$$P_2 = (6 \ \ 8 \ \ 4 \ \ 3 \ \ 1 \ \ 2 \ \ 5 \ \ 7).$$

To work as previous example and first offspring is:

$$O_1 = (6 \ \ 2 \ \ 8 \ \ 7 \ \ 5 \ \ 1 \times \times).$$

The process is halted at this stage because the bit of last filled-position in first offspring and the first bit of first parent are same. So stage one is over and now we start stage two as:
All bits are left out from both parents which are exiting in first offspring as:

$$P_1 = (\bullet \ \bullet \ \ 3 \ \ 4 \ \ \bullet \ \bullet \ \bullet \ \bullet) \quad \textit{and}$$

$$P_2 = (\bullet \ \bullet \ \ 4 \ \ 3 \ \ \bullet \ \bullet \ \bullet \ \bullet).$$

Now filled positions of parents and '×' positions of offspring are considered 1st, 2nd and 3rd positions etc., so we can completes it as is afore-mentioned.

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

Analogously, we complete the second offspring as well:

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

For better understanding we display it in Figure 3.

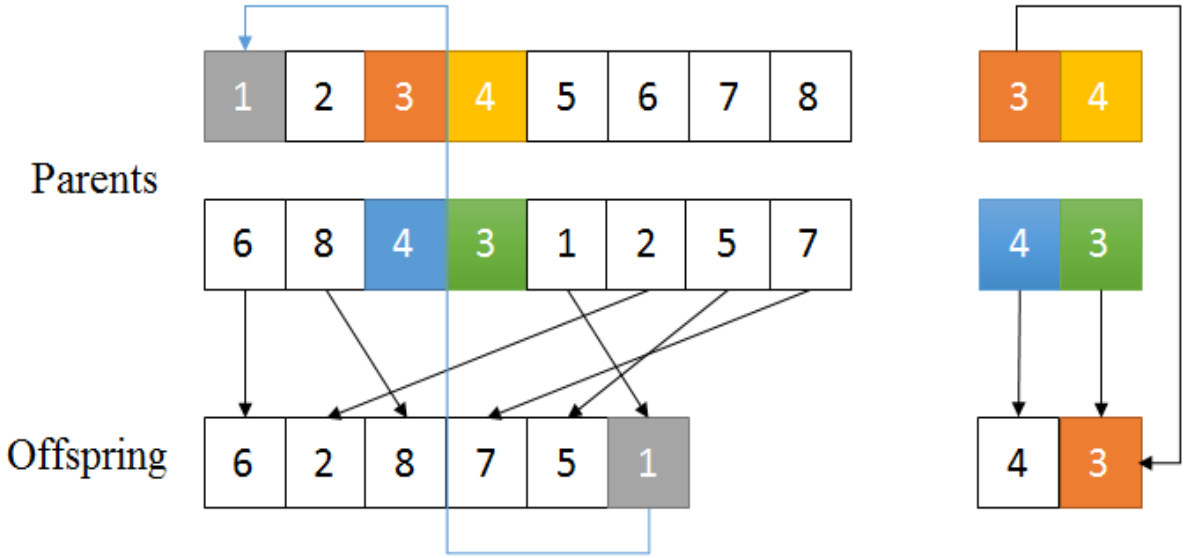

FIGURE 3. The ICX operator (within two steps)

To apply this crossover operator, we made a MATLAB code for GAs and have given pseudo-code in Figure 4.

```
N ← Number of cities
P1 ← select random parent by selection method
P2 ← select random parent by selection method
Index1 ← 1  // for child1
Index2 ← 1  // for child2
while (length of child1 != length of P1) do
    child1(index1) ← P2(index2)
    while (index1 < N) do
        location ← find (P1 == child1(index1))
        value ← P2(location)
        location ← find(P1 == value)
        value ← P2(location)
        child2(index2) ← value
        index2 + +
        location ← find(P1 == Child2(index2))
        value ← P2(location)
        if (find(child1 == value))
            break
        else
            child1(index1) ← value
            index1 + +
        end if
    end while
end while
```

FIGURE 4. The Pseudo-code of ICX

4. Computational Results and Discussion

To evaluate the performances of the proposed ICX, computational experiments have been tested by using ten benchmark instances which are taken from traveling salesman problem library (TSPLIB) [30]. We divide these instances into two parts as symmetric traveling salesman problems (TSPs) and asymmetric traveling salesman problems (ATSPs). The experiments are performed 30 times (30 runs) for each instance. In our simulation experiments, all GA programs were implemented in MATLAB version R2017a. The common parameters used in simulations are given in Table 1.

For better comparisons, we employ statistical hypothesis testing using the two-sampled pooled t-test because GAs belong to the group of stochastic search algorithms. We set the null hypothesis: 'ICX does provide better performance when performed for 30 trials′. The significance level at p = 0.05 (95% confidence) can be indicted according to the two-sample pooled t-test. The statistical value indicates whether a significant improvement by ICX ($t \leq -2.00$) or significant degradation by ICX ($t \geq 2.00$). The resulting result of the t-value ($-2.00 < t < 2.00$) do not ensure enough statistical evidence to confirm or refute the null hypothesis, which shows similar performance between the two crossover operators.

Table 1. Parametric configuration for GA

Parameter	Value
population size	150
Selection scheme	Roulette-wheel selection
Mutation method	Swap
Crossover probability	80%
Mutation probability	5%
Maximum generation	1000
Replacement percentage	20%

Table 2 summarizes the average result and standard deviation (S.D) of the 30 trials for of each method for the five symmetric benchmark instances. For instance gr21, the proposed crossover operator gives a significant improvement performance than PMX and CX operators but other two approaches are not statistically significant than it. For instance bays29, the t-Test values are indicated that proposed operator is not statistically significant with each other used crossover operators. CX2 outperforms than the proposed one for instance dantzig42 but ICX significant better performing than PMX and non-significant with all others. The only CX2 is statistically non-significant for the benchmark eil76 and ICX outperforms the other three crossover approaches (PMX, OX and CX). For instance brg180, the proposed operator ICX statistically significant with better performance than all used crossovers except OX. The overall results of Table 2 indicated that the proposed operator is significantly better than all other operators which are used in this paper. For more close comparison, we display all average results of Table 2 in Figure 5.

Table 3 summarizes the average result and S.D of the 30 trials for each method for the five asymmetric benchmark instances. The t-Test values of instance ftv33, ftv38, rbg323 and rbg443 do not provide us with enough statistical evidence to confirm that which operator is better than other. But negative values of these instances indicate an improved average performance of the proposed method. For the benchmark ft53, the proposed approach gives significantly improved performance than CX and CX2 and other two methods (PMX and OX) are statistically same behave with it. The overall results of Table 3 indicate that proposed method is performing statistically better or do not ensure evidence to confirm that which one is better with 95% confidence level. Moreover, the results confirm that there is no t-Test value which is significant degradation ($t \geq 2.00$) by ICX. For more close comparison, we display all average results of Table 3 in Figure 6.

TABLE 2. Comparison Results of Crossover Operators for TSPs

Instance	N	Optimum value	Results	PMX	OX	CX	CX2	ICX
			Average	2969	2843	2974	2902	2861
gr21	21	2707	S.D	96	117	145	149	81
			t-Test	-4.63	0.68	-3.66	-1.30	-
			Average	2599	2561	2610	2621	2588
bays29	29	2020	S.D	110	95	183	202	175
			t-Test	-0.29	0.73	-0.47	-0.66	-
			Average	1225	1101	1007	802	1048
dantzig42	42	699	S.D	188	219	166	98	103
			t-Test	-4.45	-1.18	1.13	9.32	-
			Average	562	555	563	546	549
eil76	76	538	S.D	9	10	11	5	8
			t-Test	-5.81	-2.52	-5.54	1.71	-
			Average	2148	2131	2221	2210	2113
brg180	202	1950	S.D	51	62	63	57	33
			t-Test	-3.10	-1.38	-8.18	-7.93	-

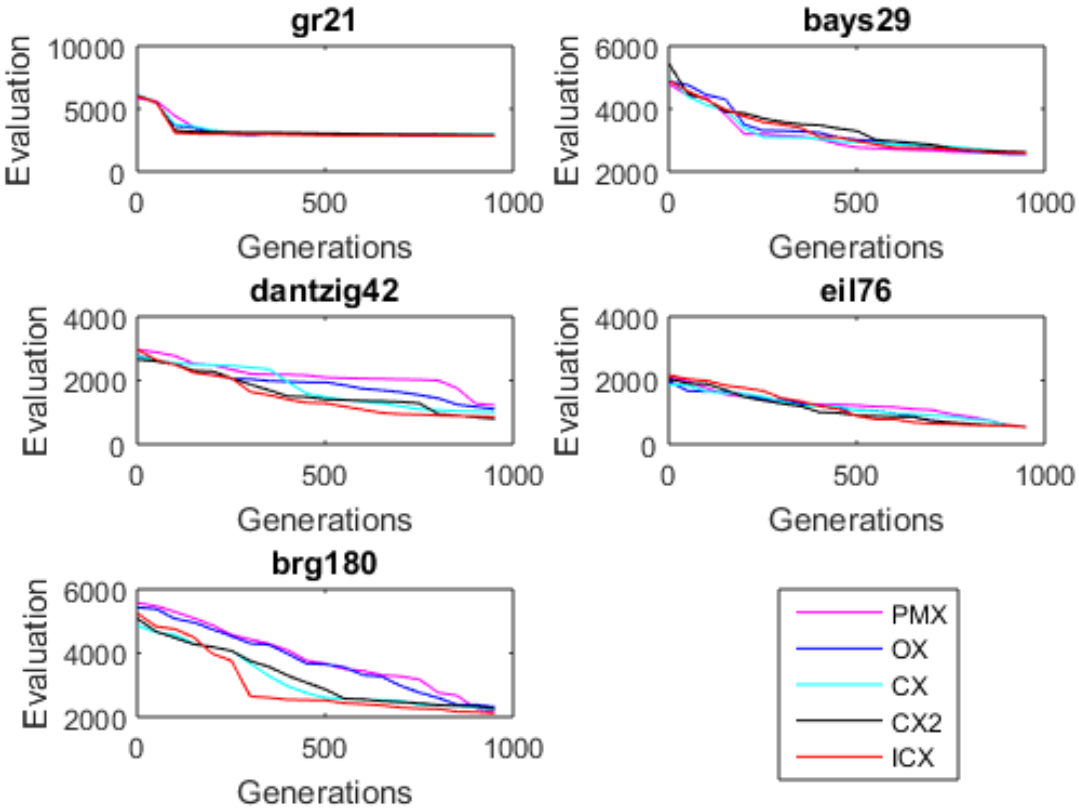

FIGURE 5. Comparative convergence of the TSPs problems

TABLE 3. Comparison Results of Crossover Operators for ATSPs

Instance	N	Optimum value	Results	PMX	OX	CX	CX2	ICX
			Average	1728	1832	1841	1887	1791
ftv33	34	1286	S.D	211	199	272	194	181
			t-Test	1.22	-0.81	-0.82	-1.95	-
			Average	2154	2077	2158	2199	2120
ftv38	39	1530	S.D	172	162	190	226	233
			t-Test	-0.63	0.82	-0.68	-1.31	-
			Average	11763	10887	11975	12498	11370
ft53	53	6905	S.D	628	949	804	1331	1069
			t-Test	-1.71	1.82	-2.43	-3.56	-
			Average	3099	2937	3109	3136	3008
rbg323	323	1326	S.D	971	848	1030	939	978
			t-Test	-0.36	0.30	-0.38	-0.51	-
			Average	4529	4445	4572	4681	4533
rbg443	443	2720	S.D	1025	1329	918	1169	1002
			t-Test	0.02	0.28	-0.15	-0.52	-

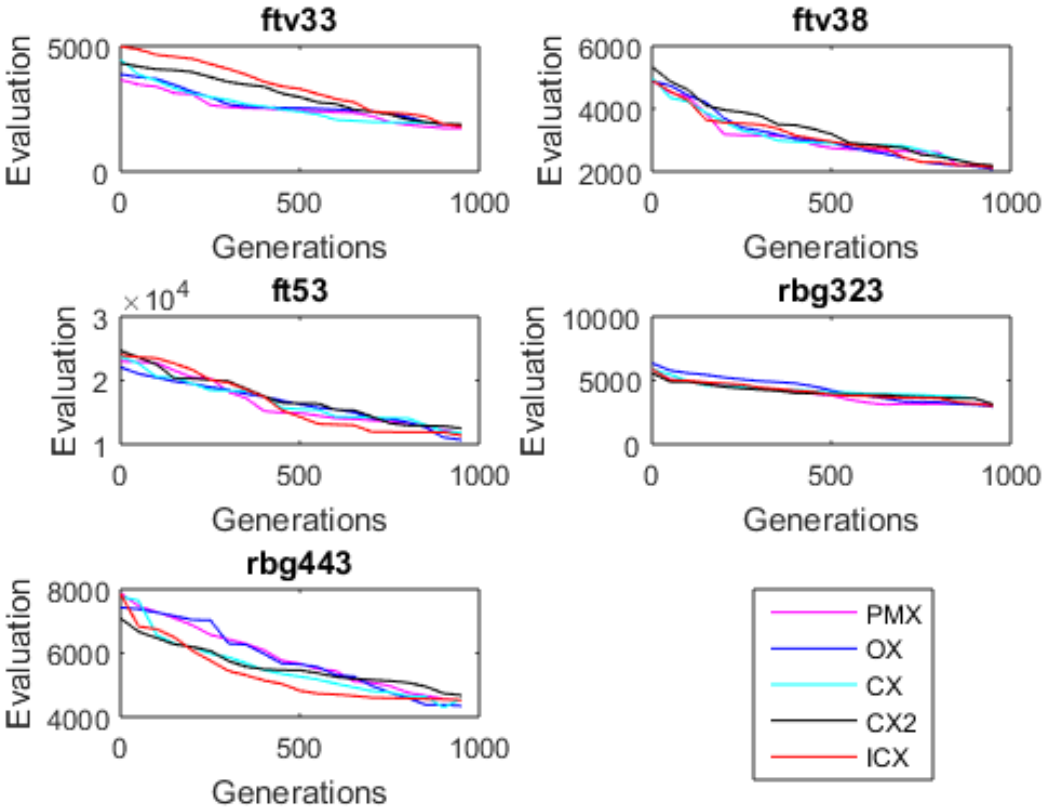

FIGURE 6. Comparative convergence of the ATSPs problems

5. Conclusion

This article represents a comprehensive overview of the performance of GAs in NP-hard problems like TSP and keenly observes how GAs create a solution without having any prior knowledge about the traveling routes. Unlike other heuristic methods, GA uses natural rules of selection, crossover and mutation to make the computation easier and fast. These things make it more valuable, better performing and efficient algorithm over those. The various crossover operators have been introduced for TSP by using GAs. We also proposed a new crossover operator for TSP. This proposed operator ICX upgrade the path-represented CX and used to improve the quality of offspring. ICX is easy to execute and always generates a valid tour of offspring. Ten benchmarks from the TSPLIB have been used to assess its performance along with other operators, for comparison and to investigate how they statistically better or degradation on the basis of t-Test results with 95% confidence level. All statistical results show that ICX is effective. Hence proposed operator might be a good candidate to get accurate convergent results. Moreover, researchers might be more confident to apply it for comparisons.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

References

[1] Ahmed, Z.H., *Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator*, Int J Biom Bioinformatics. **3**(2010), 96–105. 1

[2] Bhide, S., John, N., Kabuka, M.R., *A Boolean neural network approach for the traveling salesman problem*, IEEE T COMPUT., **42**(1993), 1271–1278. 1

[3] Bland, R.G., Shallcross, D.F., *Large traveling salesmen problems arising from experiments in x-ray crystallography*, Oper. Res. Lett., **8**(1988), 125–128. 1

[4] Bolanos, R.I., Eliana, M.T.O., and Mauricio, G.E., . *A population-based algorithm for the multi traveling salesman problem*, International Journal of Industrial Engineering Computations, **7**(2016), 245–256. 1

[5] Davis, L., *Applying Adaptive Algorithms to Epistatic Domains*, In: Proceedings of the International Joint Conference on Artificial Intelligence, 1985, 162–164. 2.1.2

[6] Deep, K., Adane, H.M., *New variations of order crossover for traveling salesman problem* International Journal of Combinatorial Optimization Problems and Informatics, **2**(2011), 2–13. 1

[7] Dorigo, M., Gambardella, L.M., *Ant colony system: a cooperative learning approach to the traveling salesman problem*, IEEE T EVOLUT COMPUT., **1**(1997), 53–66. 1

[8] Dorigo, M., Maniezzo, V., Colorni, A., *Ant system: optimization by a colony of cooperating agents*, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), **26**(1996), 29–41. 1

[9] Finke, G., Claus, A., Gunn, E.,*A two-commodity network flow approach to the traveling salesman problem*, Congressus Numerantium, **41**(1984), 167–178. 1

[10] Gen M, Cheng R. Genetic algorithms and Engineering design. John Wiley and Sons, London, UK. 1997. 1

[11] Ghadle, K.P., and Muley, Y.M., *Traveling salesman problem with MATLAB programming*, International Journal of Advances in Applied Mathematics and Mechanics, **2**(2015), 258–266. 1

[12] Glover, F., *Artificial intelligence, heuristic frameworks and tabu search*, Managerial and Decision Economics, **11**(1990):365–375. 1

[13] Goldberg, D.E., Genetic algorithms in search, optimization, and machine learning. Addison-Wesley Publishing Company, 1989. 1

[14] Goldberg, D.E., Lingle, R., *Alleles, loci, and the traveling salesman problem*, In: Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications. Hillsdale, New Jersey: Lawrence Erlbaum, 1985, 154–159. 2.1.1

[15] Holland, J.H., Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. University of Michigan Press, Oxford, UK, 1975. 1

[16] Hussain, A., Muhammad, Y.S., Sajid M.N., Hussain, I., Shoukry A.M., Gani, S., *Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator*, COMPUT INTEL NEUROSC., **2017**(2017), 1–7. 1, 2.1.4

[17] Kennedy, J., Eberhart, R.C., and Shi, Y., Swarm intelligence, morgan kaufmann publishers. Inc., San Francisco, CA, 2001. 1

[18] Kirkpatrick, S. and Toulouse, G., *Configuration space analysis of traveling salesman problems*, Journal de Physique, **46**(1985), 1277–1292. 1

[19] Kumar, N., Karambir, R.K., *A comparative analysis of PMX, CX and OX crossover operators for solving traveling salesman problem*, International journal of Latest Research in science and technology, **1**(2012), 98–101. 1

[20] Larranaga, P., Kuijpers, C.M., Murga, R.H., Inza, I., Dizdarevic, S., *Genetic algorithms for the traveling salesman problem: A review of representations and operators*, ARTIF INTELL REV, **13**(1999), 129–170. 1

[21] Lin, S., Kernighan, B.W., *An effective heuristic algorithm for the traveling salesman problem*, OPER RES, **21**(1973), 498–516. 1

[22] Michalewicz, Z., Genetic Algorithms+ Data Structures= Evolution Programs. Springer, 3rd edition, 1996. 1

[23] Miliotis, P., *Using cutting planes to solve the symmetric traveling salesman problem*, MATH PROGRAM, **15**(1978), 177–188. 1

[24] Moon, C., Kim, J., Choi, G., Seo, Y., *An efficient genetic algorithm for the traveling salesman problem with precedence constraints*, EUR J OPER RES., **140**(2002), 606–617. 1

[25] Nagata, Y., Soler, D., *A new genetic algorithm for the asymmetric traveling salesman problem*, EXPERT SYST APPL., **39**(2012), 8947–8953. 1

[26] Oliver, I.M., Smith, D., Holland, J.R., *Study of permutation crossover operators on the traveling salesman problem*, In: Grefenstette, J. J. (ed.) Genetic Algorithms and Their Applications, Proceedings of the Second International Conference. Hillsdale, New Jersey: Lawrence Erlbaum, 1987, 224–230. 2.1.3

[27] PiwoAska, A. *Genetic algorithm finds routes in traveling salesman problem with profits*, Zeszyty Naukowe Politechniki BiaAostockiej. Informatyka, (2010), 51–65. 1

[28] Potvin, J.Y., *Genetic algorithms for the traveling salesman problem*, ANN OPER RES., **63**(1996), 337–370. 1

[29] Ravikumar, C.P., *Parallel techniques for solving large scale traveling salesperson problems*, Microprocessors and Microsystems, **16**(1992), 149–158. 1

[30] Reinelt G. TSPLIB http://www. iwr. uni-heidelberg. de/groups/comopt/software. TSPLIB95. 2014. 4