Genetic Algorithm with Comprehensive Sequential Constructive Crossover for the Travelling Salesman Problem

Zakir Hussain Ahmed
Department of Mathematics and Statistics, College of Science
Al Imam Mohammad Ibn Saud Islamic University (IMSIU)
Riyadh, Kingdom of Saudi Arabia

***Abstract*—The travelling salesman problem (TSP) is a very famous NP-hard problem in operations research as well as in computer science. To solve the problem several genetic algorithms (GAs) are developed which depend primarily on crossover operator. The crossover operators are classified as distance-based crossover operators and blind crossover operators. The distance-based crossover operators use distances between nodes to generate the offspring(s), whereas blind crossover operators are independent of any kind of information of the problem, except follow the problem's constraints. Selecting better crossover operator can lead to successful GA. Several crossover operators are available in the literature for the TSP, but most of them are not leading good GA. In this study, we propose reverse greedy sequential constructive crossover (RGSCX) and then comprehensive sequential constructive crossover (CSCX) for developing better GAs for solving the TSP. The usefulness of our proposed crossover operators is shown by comparing with some distance-based crossover operators on some TSPLIB instances. It can be concluded from the comparative study that our proposed operator CSCX is the best crossover in this study for the TSP.**

Keywords*—Genetic algorithm; reverse greedy sequential constructive crossover; comprehensive sequential constructive crossover; travelling salesman problem; NP-hard

I. INTRODUCTION

The travelling Salesman Problem (TSP) is an old and famous combinatorial optimization problem in computer science and operations research which was documented in 1759 by Euler, of course not by that name, whose aim was to solve the Knights' tour problem. A solution of the problem is a knight visit of each of the 64-squares of a chessboard exactly once in its tour. In 1932, the term 'travelling salesman' was first used in a German book 'The travelling salesman, how and what he should do to get commissions and be successful in his business', written by a veteran travelling salesman. In 1948, the problem was formally introduced by RAND Corporation. The problem became popular by the Corporation's reputation and then by introducing linear programming for solving combinatorial optimization problems. The problem aims to obtain a least cost Hamiltonian cycle in a network of nodes [1]. The TSP has applications in automatic drilling of printed circuit boards and circuits, very-large-scale-integrated (VLSI) circuit, X-ray crystallography, movement of people, computer wiring [2]. The problem can be defined as follows:

A A network of n nodes, with 'node 1' as 'depot' and a travel cost (or distance, or travel time etc.,) matrix C= [c_{ij}] of order n related to ordered pairs (i, j) of nodes is given. The aim of the problem is to obtain a least cost Hamiltonian cycle. Symmetric and asymmetric TSPs are two cases of TSP. It is symmetric if $c_{ij} = c_{ji}$, for all i, j; else, asymmetric. For asymmetric case, there are possibly $(n-1)!$ solutions out of which at least one provides the minimum cost, and for symmetric case, there are likely $\frac{(n-1)!}{2}$ solutions along with same cost opposite cyclic permutations in n-node network. The possible number of solutions is very huge in both cases and the problem is NP-Hard [3]. The problem is very famous, and it is researched by many researchers because of its difficulty and its usability to model many other difficult real-life problems.

In the literature for the TSP, numerous exact and heuristic methods have been proposed, and most of them are heuristic/metaheuristic methods. Genetic algorithm, tabu search, artificial neural network, ant colony optimization, particle swarm optimization, etc. are some of the effective metaheuristic algorithms [1]. Among them genetic algorithm is seen to be one of the best metaheuristic algorithms for the problem [4].

Genetic algorithm (GA) is a search process that is inspired by natural biological evolution process. It was proposed by John Holland in 1970s [5]. It starts initially with a population of strings, called chromosomes, that encode solutions to a problem, and operates probably three operators - selection, crossover and mutation, to produce new and probably better populations in successive generations. Crossover operator is the leading operator in GAs [1]. Mutation enlarges search space as well as protects GAs from damaging any genetic element resulted from crossover and selection operators. Among three operators, crossover is the primary operator, and hence, numerous crossover operators have been suggested as well as improved for finding better solution to the TSP [5].

In this present study, we propose first reverse greedy sequential constructive crossover (RGSCX) and then comprehensive sequential constructive crossover (CSCX) by combining greedy sequential constructive crossover (GSCX) [6] with RGSCX for the TSP. We then applied the proposed operators manually on two chromosomes to produce offspring(s). Finally, the usefulness of our proposed crossover

operators is shown by comparing with some distance-based crossover operators on some TSPLIB instances. One can conclude from the comparative study that our proposed operator CSCX is the best crossover operator in this study for the TSP.

This paper is organized as follows: A survey on related work for the TSP is reported in Section II. Section III develops proposed crossover operators for the problem, whereas, Section IV reports computational experiments for six crossover operators. Finally, Section V presents conclusion and future works.

II. Related Work

There are numerous crossover operators suggested for solving the TSP. Two kinds of crossover operators have been suggested for the problem, namely distance-based crossover and blind crossover. In distance-based crossover operators, offspring are created using distance between nodes, whereas, in blind crossover operators, offspring are not created using any information of the problem, rather they bother only about the problem's constraints. The partially mapped crossover (PMX) [7], ordered crossover (OX)[8], order based crossover (OBX) and position based operator (PBX) [9], alternating edges crossover (AEX) [10], cycle crossover (CX) [11], edge recombination crossover (ERX) [12], generalized N-point crossover (GNX) [13], etc. are some of the blind crossover operators, and greedy crossover (GX) [10], heuristic crossover (HX) [14], distance preserving crossover (DPX) [15], sequential constructive crossover (SCX) [4], etc. are some of the popular distance-based crossover operators.

Selecting better crossover operator can lead to successful GA. Though several crossover operators are available in the literature for the TSP, but most of them are not leading good GA. So, our aim is to develop distance-based crossover operators and then compare them with some existing crossover operators for showing the effectiveness of the proposed operators. Osaba et al. [16] pointed that comparisons between heuristic/metaheuristic methods using blind operators and heuristic methods using optimizing functions must be avoided. Otherwise, comparison would not be reliable, because nature of the methods is different. If someone wants to verify the quality of any distance-based crossover operator, suppose SCX, the results found by the operator should be compared against the one found by another distance-based crossover operator, such as HX, GX or DPX. Now, if performance of SCX is compared against other blind operators, such as the PMX, OX, CX, ERX or AEX, the comparison will not be fair. Since, we propose to modify SCX, hence modified SCXs will be compared with only some distance-based crossovers such as HX, GX and SCX. So, we are going to explain some of these operators only through an example of pair of parent chromosomes. To represent solution by chromosome, the path representation that lists the permutation of node is considered here. For example, let {1, 2, 3, 4, 5, 6, 7, 8, 9} is the list of node labels for a 9-node problem example, then the tour {1→7→6→3→8 → 9→2→4→ 5 →1} is represented by (1, 7, 6, 3, 8, 9, 2, 4, 5). The objective function cost (value) is the total cost of all edges in this tour.

A. Modified Heuristic Crossover Operator

Liepins et al. [17] proposed a modified HX for the TSP that initially selects a node and copy it to the offspring then its nearest node is copied to the offspring until the offspring is complete. Jog et al. [18] described a modified heuristic crossover (MHX) of the HX [17] that creates an offspring chromosome from two parent chromosomes as follows. Select a node randomly as the starting node for offspring chromosome. Compare two arcs leaving the starting node in both parents and select the cheaper one. Continue to copy the cheaper arc of two arcs in both parents into the offspring. If the cheaper parent arc introduces a cycle (repeating any node) into offspring (illegal chromosome), check whether the other parent arc introduces a cycle. If the second arc does not introduce a cycle, copy this arc into the offspring, otherwise, copy the cheaper arc into the offspring from a group of maximum 20 randomly selected arcs that do not introduce a cycle. Continue until a complete offspring is created and the first parent chromosome is replaced by this offspring. This operator generates an offspring using two parent chromosomes.

We illustrate the MHX using the 9-node problem example shown in Table I as the cost matrix. Suppose parent chromosomes P1: (1, 2, 3, 4, 6, 9, 5, 7, 8) and P2: (1, 3, 5, 7, 8, 9, 4, 2, 6) with costs 83 and 75 respectively are selected as parent chromosomes. We use these same chromosomes to illustrate all crossover operators. Also, as we fixed 'node 1' as depot node (first gene), so, we always start the procedure for all crossover operators from the 'node 1'.

Since the procedure starts from the node 1, so, initially the offspring is (1). The arcs in both parents going out from the node 1 are considered, i.e. 1→2 and 1→3 with costs 7 and 15 respectively. So, node 2 is added and the incomplete offspring becomes: (1, 2). Next, the parent arcs leaving the node 2 are considered, i.e. 2→3 and 2→6 with costs 8 and 3 respectively. Among them 2→6 is cheaper, So, node 6 is added and the incomplete offspring becomes: (1, 2, 6). Then the arcs 6→9 and 6→1 must be considered, but the second one is illegal it leads to already visited node, so, 6→9 is chosen which leads to the incomplete offspring: (1, 2, 6, 9). Continuing in this way, one can obtain a complete offspring as: (1, 2, 6, 9, 4, 8, 5, 7, 3) having cost 56.

TABLE I. The Cost Matrix

Node	1	2	3	4	5	6	7	8	9
1	999	7	15	9	10	6	8	9	10
2	11	999	8	7	11	3	6	4	3
3	15	5	999	16	12	5	8	13	4
4	2	5	11	999	9	13	14	4	2
5	8	6	3	5	999	6	7	10	9
6	6	13	8	11	5	999	5	4	5
7	5	15	3	7	12	6	999	8	9
8	9	3	9	14	3	11	8	999	10
9	11	16	3	9	10	7	9	10	999

B. Very Greedy Crossover Operator

Julstrom [19] proposed a greedy extension of HX [18], named very greedy crossover (VGX), for solving the TSP, which is as follows. It selects a starting node randomly, then constructs a tour by investigating the four arcs leaving that node in both parents, according to their costs. It copies the first arc that does not make a cycle. If all four parent arcs from any node make cycles, then it uses the cheapest arc to an unvisited node. If the parents share an edge from the current node to one unvisited node, the operator copies that arc, even if it is not cheapest. The VGX operator creates an offspring using two parents. We illustrate the VGX using same example shown above.

Initially, the offspring is (1). The arcs in both parents leaving the node 1 are considered, i.e. 1→2 and 1→8 with costs 7 and 9 respectively in P_1 and 1→3 and 1→6 with costs 15 and 6 respectively in P_2. So, node 6 is copied that gives the incomplete offspring as: (1, 6). Next, the parent arcs leaving the node 6 are considered, i.e. 6→9, 6→4, 6→1 and 6→2 having their respective costs 5, 11, 6 and 13. Among them 6→9 is cheaper, So, node 9 is copied that gives an incomplete offspring as: (1, 6, 9). Continuing in this way, one can obtain a complete offspring as: (1, 6, 9, 4, 2, 3, 5, 7, 8) having cost 69.

C. Adaptive Sequential Constructive Crossover Operator

The sequential constructive crossover (SCX) is proposed in [4] for solving the TSP which found very good solution for symmetric and asymmetric TSPLIB instances. In [20], SCX is modified as follows: after current node, if no any legitimate node is available in any parent, then it searches from the starting of the parent chromosome and the first legitimate node is selected as next node. In [21], a comparative study is reported that shows that SCX is the best among eight crossover operators. A modified SCX, named bidirectional circular SCX (BCSCX) is developed in [22]. In [23], an adaptive SCX (ASCX) is proposed that creates an offspring adaptively, either in forward or in backward or in mixed direction that depends on next node's cost. Hence, eight neighbour nodes of any current node is considered, four for each of the two nodes (genes).

Since in a chromosome number of genes is n, the 'node 1' is selected as the first as well the $(n+1)^{th}$ genes. The algorithm for the ASCX is stated as follows [23].

Step 1: Start from the first gene, 'node 1' (i.e., current node p =1 in position i=1) in forward direction and from the $(n+1)^{th}$ gene, 'node 1' (it is not shown in the chromosome), (i.e., current node q =1 in position j=n+1) in backward direction.

Step 2: Sequentially search both parent chromosomes in right direction and consider the first 'legitimate node' (the node that is not yet visited) appeared after 'node p' in each parent. If no 'legitimate node' after 'node p' is present in any of the parents, search sequentially from the starting of the parent (wrap around) and consider the first 'legitimate node'. Suppose the 'node α' and the 'node β' are found in 1^{st} and 2^{nd} parent respectively. Go to Step 3.

Step 3: Sequentially search both parent chromosomes in left direction and consider the first 'legitimate node' appeared after 'node p' in each parent. If no 'legitimate node' after 'node p' is present in any of the parents, search sequentially from the end of the parent (wrap around) and consider the first 'legitimate node'. Suppose the 'node γ' and the 'node δ' are found in 1^{st} and 2^{nd} parent respectively. Now, suppose among four nodes, 'node u' is the cheapest with cost s=min. $\{c_{p\alpha}, c_{p\beta}, c_{p\gamma}, c_{p\delta}\}$. Go to Step 4.

Step 4: Sequentially search both parent chromosomes in left direction and consider the first 'legitimate node' appeared after 'node q' in each parent. If no 'legitimate node' after 'node q' is present in any of the parents, search sequentially from the end of the parent (wrap around) and consider the first 'legitimate node'. Suppose the 'node w' and the 'node x' are found in 1^{st} and 2^{nd} parent respectively. Go to Step 5.

Step 5: Sequentially search both parent chromosomes in right direction and consider the first 'legitimate node' appeared after 'node q' in each parent. If no 'legitimate node' after 'node q' is present in any of the parents, search sequentially from the beginning of the parent (wrap around) and consider the first 'legitimate node'. Suppose the 'node y' and the 'node z' are found in 1^{st} and 2^{nd} parent respectively. Now, suppose among four nodes, 'node v' is the cheapest with cost t=min. $\{c_{wq}, c_{xq}, c_{yq}, c_{zq}\}$. Now, for selecting the next node as well as adding it in a position in the offspring chromosome go to Step 6.

Step 6: If $s \leq t$, then add 'node u' in position 'i' in the partially constructed offspring chromosome and set p=u, i=i+1. Otherwise, add 'node v' in position 'j' in the partially constructed offspring chromosome and set q=v, j=j-1. Now, If the offspring is a complete chromosome, then stop, otherwise, go to Step 2.

We illustrate the ASCX using same example shown above. As number of genes in the chromosomes is 9, the 'node 1' is the first as well as the 10^{th} gene (not displayed in the chromosomes). After 'node 1' (first gene), the legitimate nodes in P_1 in forward direction is 2 and in backward direction (after wrapping around) is 8, and in P_2 they are 3 and (after wrapping around) 6, having their respective costs 7, 9, 15 and 6. So, the cheapest is node 6 having cost 6. From the end, before 'node 1' (10^{th} gene), the legitimate nodes in P_1, in backward direction is 8 and in forward direction (after wrapping around) is 2, and in P_2 they are 6 and (after wrapping around) 3, having their respective costs 9, 7, 6 and 15. So, the cheapest is node 6 having cost 6. As both cheapest nodes are 6, it is added as the second gene in the current offspring that leads the incomplete offspring to (1, 6, *, *, *, *, *, *, *, *).

After 'node 6' (second gene), the legitimate nodes in P_1 in forward direction is 9 and in backward direction is 4, and in P_2 they are (after wrapping around) 3 and 2, having their respective costs 5, 11, 8 and 13. So, the cheapest is node 9 having cost 5. From the end, before 'node 1' (10^{th} gene), the legitimate nodes in P_1, in backward direction is 8 and in forward direction (after wrapping around) is 2, and in P_2 they are 2 and (after wrapping around) 3, having their respective costs 9, 11, 11 and 15. So, the cheapest is node 8 having cost 9. As node 9 is cheaper between the cheapest nodes, it is added as the third gene in the current offspring that leads the incomplete offspring to (1, 6, 9, *, *, *, *, *, *, *). Continuing in this way, one can obtain a complete offspring as: (1, 6, 9, 4, 8, 2, 3, 5, 7) having cost 59.

D. Greedy Sequential Constructive Crossover Operator

Recently, Ahmed [6] proposed the greedy SCX (GSCX) by introducing a greedy method, which is as follows.

Step 1: Start from 'node 1' (i.e., current node p =1).

Step 2: Sequentially search both parent chromosomes and consider the first 'legitimate node' (the node that is not yet visited) appeared after 'node p' in each parent. If 'legitimate node' after 'node p' is found in both parents, then go to Step 3, otherwise, consider the cheapest 'legitimate node' from the group of remaining legitimate nodes and concatenate it to the partially constructed offspring chromosome. If the offspring is a complete chromosome, then stop, otherwise, rename this present node as 'node p' and repeat this Step 2

Step 3: Suppose the 'node α' and the 'node β' are found in 1st and 2nd parent respectively, then for selecting the next node go to Step 4.

Step 4: If $c_{p\alpha} < c_{p\beta}$, then select 'node α', otherwise, 'node β' as the next node and concatenate it to the partially constructed offspring chromosome. If the offspring is a complete chromosome, then stop, otherwise, rename the present node as 'node p' and go to Step 2.

We illustrate the GSCX using same example shown above. As 'node 1' is the first gene, after this node, the legitimate nodes in P_1 is 2 and in P_2 is 3 having $c_{12}=7$ and $c_{13}=15$. As $c_{12}<c_{13}$, the node 2 is added as the second gene in the current offspring that leads the incomplete offspring to (1, 2).

After 'node 2', the legitimate nodes in P_1 is 3 and in P_2 is 6 having $c_{23}=8$ and $c_{26}=3$. As $c_{26}<c_{23}$, the node 6 is added as the third gene in the current offspring that leads the incomplete offspring to (1, 2, 6).

After 'node 6', the legitimate nodes in P_1 is 9 and in P_2 is nothing. So, we search and find the cheapest legitimate node as 8, which is added as the fourth gene in the current offspring that leads the incomplete offspring to (1, 2, 6, 8). Continuing in this way, one can obtain a complete offspring as: (1, 2, 6, 8, 5, 7, 3, 9, 4) having cost 42.

III. Proposed Crossover Operators

We propose two crossover operators - reverse greedy sequential constructive crossover operator and comprehensive sequential constructive crossover operator.

A. Reverse Greedy Sequential Constructive Crossover Operator

In this proposed operator, we apply the GSCX in reverse direction and we name it as reverse GSCX (RGSCX). We construct the offspring in reverse direction, that is, from the last node (gene) of the offspring back to the first node (gene) of the same. So, we define RGSCX as follows.

Step 1: Suppose the 'node α' and the 'node β' are the last nodes in 1st and 2nd parent respectively. Since 'node 1' is the first node (gene), then for selecting the last node, we check whether $c_{\alpha 1} < c_{\beta 1}$. If yes, then select 'node α', otherwise, 'node β' as the last node and concatenate it to the partially constructed offspring chromosome. Then rename this present node as 'node p' and go to Step 2.

Step 2: Sequentially search both parent chromosomes in reverse direction and consider the first 'legitimate node' (the node that is not yet visited) appeared before 'node p' in each parent. If 'legitimate node' before 'node p' is found in both parents, then go to Step 3, otherwise, consider the cheapest 'legitimate node' from the group of remaining legitimate nodes and concatenate it to the partially constructed offspring chromosome. If the offspring is a complete chromosome, then stop, otherwise, rename this present node as 'node p' and repeat this Step 2.

Step 3: Suppose the 'node α' and the 'node β' are found in 1st and 2nd parent respectively, then for selecting the previous node go to Step 4.

Step 4: If $c_{\alpha p} < c_{\beta p}$, then select 'node α', otherwise, 'node β' as the previous node and concatenate it to the partially constructed offspring chromosome. If the offspring is a complete chromosome, then stop, otherwise, rename this present node as 'node p' and go to Step 2.

We illustrate the RGSCX using same example shown above. By default, the 10th node is 1. The last nodes (9th genes) are 8 and 6 in P_1 and P_2 respectively having $c_{81}=9$ and $c_{61}=6$. As $c_{61}<c_{81}$, the node 6 is considered as the 9th gene that initiated the incomplete offspring as (6).

Before 'node 6', the legitimate nodes in P_1 is 4 and in P_2 is 2 having $c_{46}=13$ and $c_{26}=3$. As $c_{26}<c_{46}$, the node 2 is added as the 8th gene in the current offspring that leads the incomplete offspring to (2, 6).

Before 'node 2', the legitimate node in P_1 is nothing. So, we search and find the cheapest legitimate node as 8, which is added as the 7th gene in the current offspring that leads the incomplete offspring to (8, 2, 6).

Before 'node 8', the legitimate nodes in both P_1 and in P_2 are node 7, so it is added as the 6th gene in the current offspring that leads the incomplete offspring to (7, 8, 2, 6).

Also, before 'node 7', the legitimate nodes in both P_1 and in P_2 are node 5, so it is added as the fifth gene in the current offspring that leads the incomplete offspring to (5, 7, 8, 2, 6).

Before 'node 5', the legitimate nodes in P_1 is 9 and in P_2 is 3 having $c_{95}=10$ and $c_{35}=12$. As $c_{95}<c_{35}$, the node 9 is added as the fourth gene in the current offspring that leads the incomplete offspring to (9, 5, 7, 8, 2, 6).

Before 'node 9', the legitimate nodes in P_1 is 4 and in P_2 is 3 having $c_{49}=2$ and $c_{39}=4$. As $c_{49}<c_{39}$, the node 4 is added as the third gene in the current offspring that leads the incomplete offspring to (4, 9, 5, 7, 8, 2, 6). Continuing in this way, one can obtain a complete offspring as: (1, 3, 4, 9, 5, 7, 8, 2, 6) having cost 70.

B. Comprehensive Sequential Constructive Crossover Operator

We propose a comprehensive SCX (CSCX) by combining two crossover operators GSCX and RGSCX that produces two offspring. So, by using above example parents, it produces both offspring (1, 2, 6, 8, 5, 7, 3, 9, 4) and (1, 3, 4, 9, 5, 7, 8, 2, 6) with cost 42 and 70 respectively which are less than costs of both the parent chromosomes.

Our GA is non-hybrid, simple, which uses basic GA processes and operators, but does not incorporate any other heuristic algorithm. In our simple GA, starting with random chromosome population, good chromosomes are selected by stochastic remainder selection technique, then population passes through one selected crossover operator and swap mutation operator. Our simple GA may be designed as follows.

```
SimpleGA ( )
{ Initialize random population of size P_s;
 Evaluate the population;
 Generation = 0;
 While stopping condition is not satisfied
 { Generation = Generation + 1;
 Select good chromosomes by selection operator;
 Select a crossover operator and do crossover with crossover
      probability P_c;
 Do swap mutation with mutation probability P_m;
 Evaluate the population;
 }
}
```

IV. Computational Experiments

The simple GAs using six crossover operators (MHX, VGX, ASCX, GSCX, RGSCX and CSCX) have been encoded in Visual C++. To compare the competence of these operators, simple GAs are applied on twenty eight TSPLIB instances [24] and then executed on a Laptop with specification i3-3217U CPU@1.80 GHz and 4 GB RAM under MS Windows 7. Among the twenty eight problem instances, instances ftv33, ftv35, ftv38, p43, ftv44, ftv47, ry48p, ft53, ftv55, ftv64, ft70, ftv70, kro124p, ftv170, rbg323, rbg358, rbg403 and rbg443 are asymmetric, and instances gr21, fri26, bayg29, dantzig42, eil51, berlin52, pr76, lin105, d198 and a280 are symmetric. For all simple GAs, the parameters are set as follows: 50 is population size, 1.0 is crossover probability, 0.20 is mutation probability, and 1,000 is maximum generations that is set as the stopping condition. For each instance, the experiments were repeated 50 times. Figures 1 shows results for rbg443 (considering only 100 generations) by all simple GAs. Each graph is for one crossover operator that shows the improvement of the solution as the number of generations increases. In the figure, the label on the left margin denotes the percentage of excess (Excess (%)) to the best known solution reported in TSPLIB website, which is calculated by the formula.

$$Excess\ (\%) = \frac{Solution\ Obtained\ -Best\ Known\ Solution}{Best\ Known\ Sol}\ x\ 100.$$

It is seen for the Figure 1 that MHX has some deviations, but not the best. GSCX has limited deviation but gets stuck very quickly in local minimum. Though ASCX and CSCX have less deviations and they are competing, however CSCX finds best results.

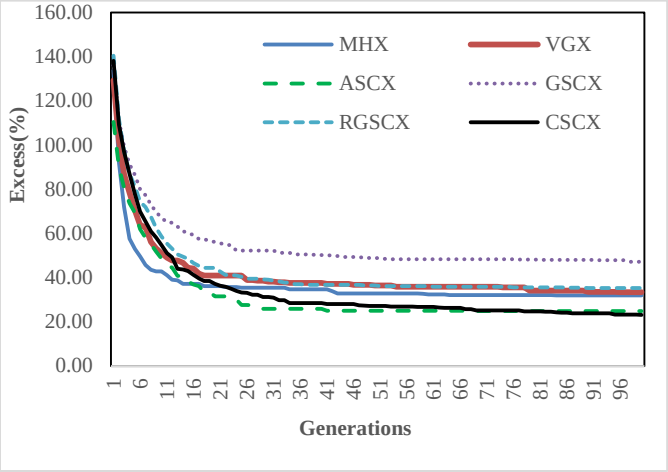

Fig. 1. Excess(%) by GAs using different Crossover Operators.

We summarize the results of our experiments using six crossover operators in Tables II and IV. We have organized the tables as follows: a row corresponds to the summarized results for a problem instance using variant GAs, first column reports a problem instance and its best-known solution (within brackets), second column reports the size of the instance, third column reports title of the summarized results and remaining each column is for GA using the mentioned crossover operator. The result using each crossover operator is designated by its best solution cost (Best Sol), average solution cost (Avg. Sol), percentage of excess of average solution to the best-known solution (Avg. Exc(%)), standard deviation of solution costs (S.D.), and average convergence time (Avg. Time) (in seconds). The best result over these six crossover operators for an instance is marked by bold face.

The Table II reports results using the GAs for the asymmetric instances. The crossover operators MHX and GSCX could not obtain either lowest best solution cost or average solution cost for any asymmetric instance. The crossover operators VGX and RGSCX obtain lowest best solution cost for the instance p43, whereas CSCX obtains lowest best solution costs for remaining seventeen instances. The crossover operator ASCX obtains lowest average solution cost for the instance p43 with lowest S.D., whereas CSCX obtains lowest average solution cost with lowest S.D. for remaining seventeen instances. By looking at average of Avg. Exc (%), one can make rank of the crossover operators. Accordingly, CSCX produce the best results, while ASCX is the second best, MHX and RGSCX are competing for the third best, and VGX is the worst. The results are also depicted in Figure 2, which also demonstrates the usefulness of our proposed crossover CSCX.

TABLE II. RESULTS BY THE CROSSOVER OPERATORS FOR ASYMMETRIC TSPLIB INSTANCES

Instance	n	Results	MHX	VGX	ASCX	GSCX	RGSCX	CSCX
ftv33 (1286)	34	Best Sol	1376	1404	1371	1380	1396	**1341**
		Avg. Sol	1479.56	1501.18	1394.72	1458.48	1464.16	**1382.86**
		Avg. Exc (%)	15.05	16.73	8.45	13.41	13.85	7.53
		S.D.	50.71	39.65	2.85	47.24	43.98	14.42
		Avg. Time	0.18	0.09	0.18	0.05	0.08	0.10
ftv35 (1473)	36	Best Sol	1520	1543	1586	1531	1583	**1499**
		Avg. Sol	1623.68	1649.28	1657.08	1631.32	1705.56	**1551.44**
		Avg. Exc (%)	10.23	11.97	12.50	10.75	15.79	5.33
		S.D.	55.93	46.01	31.98	47.09	56.93	32.17
		Avg. Time	0.18	0.15	0.07	0.05	0.08	0.17
ftv38 (1530)	39	Best Sol	1604	1618	1679	1613	1672	**1550**
		Avg. Sol	1678.08	1714.04	1748.64	1690.50	1722.22	**1605.72**
		Avg. Exc (%)	9.68	12.03	14.29	10.49	12.56	4.95
		S.D.	40.17	44.37	25.88	39.65	59.15	32.38
		Avg. Time	0.23	0.13	0.12	0.05	0.10	0.22
p43 (5620)	43	Best Sol	5631	**5625**	5631	5631	**5625**	5627
		Avg. Sol	5640.96	5636.52	**5635.70**	5641.20	5640.18	5639.30
		Avg. Exc (%)	0.37	0.29	0.28	0.38	0.36	0.34
		S.D.	5.91	5.35	1.96	6.97	8.73	7.09
		Avg. Time	0.29	0.34	0.37	0.17	0.14	0.27
ftv44 (1613)	45	Best Sol	1725	1686	1733	1706	1627	**1613**
		Avg. Sol	1843.98	1863.24	1796.12	1853.28	1793.12	**1669.48**
		Avg. Exc (%)	14.32	15.51	11.35	14.90	11.17	3.50
		S.D.	59.21	68.81	28.08	60.57	71.07	37.33
		Avg. Time	0.41	0.37	0.73	0.12	0.21	0.19
ftv47 (1776)	48	Best Sol	1860	1902	2054	1864	1919	**1833**
		Avg. Sol	2046.38	2065.80	2111.08	2021.72	2102.62	**1936.26**
		Avg. Exc (%)	15.22	16.32	18.87	13.84	18.39	9.02
		S.D.	84.14	91.77	21.98	70.92	69.51	43.12
		Avg. Time	0.57	0.46	0.74	0.26	0.23	0.27
ry48p (14422)	48	Best Sol	15629	15204	15290	15469	15293	**14983**
		Avg. Sol	16120.54	16062.36	15744.88	16150.78	15664.34	**15479.70**
		Avg. Exc (%)	11.78	11.37	9.17	11.99	8.61	7.33
		S.D.	287.99	306.73	200.95	278.65	186.04	278.44
		Avg. Time	0.45	0.36	0.50	0.17	0.18	0.13
ft53 (6905)	53	Best Sol	8061	7899	7631	7882	7973	**7486**
		Avg. Sol	8617.90	8529.00	8127.34	8614.86	8427.90	**7816.04**
		Avg. Exc (%)	24.81	23.52	17.70	24.76	22.06	13.19
		S.D.	278.54	291.14	156.51	277.13	224.61	194.86
		Avg. Time	0.49	0.59	0.38	0.35	0.31	0.34
ftv55 (1608)	56	Best Sol	1773	1753	1749	1723	1705	**1639**
		Avg. Sol	1872.60	1846.58	1798.86	1841.82	1773.14	**1712.58**
		Avg. Exc (%)	16.46	14.84	11.87	14.54	10.27	6.50
		S.D.	56.81	60.22	20.58	50.89	48.93	39.74
		Avg. Time	0.56	0.54	1.05	0.36	0.34	0.28
ftv64 (1839)	65	Best Sol	2010	2079	2145	1990	1999	**1879**
		Avg. Sol	2196.10	2228.14	2236.32	2140.28	2178.94	**1921.62**
		Avg. Exc (%)	19.42	21.16	21.61	16.38	18.49	4.49
		S.D.	85.06	71.79	44.42	76.32	89.40	39.74
		Avg. Time	0.66	0.90	1.01	0.29	0.53	0.49
ft70 (38673)	70	Best Sol	40976	40926	41592	41129	41445	**40050**
		Avg. Sol	42208.68	42210.74	42447.92	42185.60	42283.14	**41080.98**
		Avg. Exc (%)	9.14	9.15	9.76	9.08	9.34	6.23
		S.D.	517.87	480.29	292.36	571.46	442.49	376.74
		Avg. Time	0.99	0.91	1.80	0.64	0.72	0.53
ftv70 (1950)	71	Best Sol	2145	2154	2276	2118	2068	**1975**
		Avg. Sol	2326.06	2350.86	2332.22	2296.32	2294.38	**2065.54**
		Avg. Exc (%)	19.29	20.56	19.60	17.76	17.66	5.93
		S.D.	80.56	90.05	41.54	69.83	91.22	59.63
		Avg. Time	0.76	1.08	1.16	0.48	0.63	0.68
kro124p (36230)	100	Best Sol	41199	41764	41246	41251	40956	**38432**
		Avg. Sol	43371.14	43167.84	42471.12	42829.16	42967.98	**40303.68**
		Avg. Exc (%)	19.71	19.15	17.23	18.21	18.60	11.24
		S.D.	990.53	781.36	462.23	780.6	1057.48	877.77
		Avg. Time	0.94	1.12	0.58	0.42	0.92	0.79

(CONTD.) RESULTS BY THE CROSSOVER OPERATORS FOR ASYMMETRIC TSPLIB INSTANCES

Instance	n	Results	MHX	VGX	ASCX	GSCX	RGSCX	CSCX
ftv170 (2755)	171	Best Sol	3303	3551	3232	3656	3517	**2968**
		Avg. Sol	3607.68	3835.46	3393	3799.50	3767.42	**3178.74**
		Avg. Exc (%)	30.95	39.22	23.16	37.91	36.75	15.38
		S.D.	130.91	159.74	95.42	130.15	214.40	79.22
		Avg. Time	1.79	4.05	0.93	1.74	2.74	2.99
rbg323 (1326)	323	Best Sol	1553	1558	1611	1597	1617	**1400**
		Avg. Sol	1594.20	1644.42	1618.8	1677.12	1677.76	**1443.04**
		Avg. Exc (%)	20.23	24.01	22.08	26.48	26.53	8.83
		S.D.	19.15	28.06	17.7	34.19	28.94	15.82
		Avg. Time	13.45	20.24	23.53	15.76	17.71	24.28
rbg358 (1163)	358	Best Sol	1481	1495	1327	1522	1514	**1325**
		Avg. Sol	1541.4	1555.36	1387.92	1591.04	1650.14	**1373.36**
		Avg. Exc (%)	32.54	33.74	19.34	36.80	41.89	18.09
		S.D.	26.05	27.79	24.05	36.93	48.64	22.04
		Avg. Time	8.86	25.67	30.77	18.16	24.36	19.5
rbg403 (2465)	403	Best Sol	3033	3104	2922	3149	2833	**2636**
		Avg. Sol	3110	3172.66	2983.38	3214.22	2980.9	**2704.58**
		Avg. Exc (%)	26.17	28.71	21.03	30.39	20.93	9.72
		S.D.	38.51	35.02	21.43	42.23	46.77	30.76
		Avg. Time	42.8	33.66	38.93	28.96	32.33	27.67
rbg443 (2720)	443	Best Sol	3399	3517	3252	3573	3188	**2932**
		Avg. Sol	3498.92	3604.94	3321.58	3678.86	3329.06	**2993.72**
		Avg. Exc (%)	28.64	32.53	22.12	35.25	22.39	10.06
		S.D.	41.53	33.71	20.95	48.04	60.93	28.09
		Avg. Time	53.28	45.97	50.82	30.78	38.33	33.07
Average of Avg. Exc (%)			**18.00**	**19.49**	**15.58**	**19.07**	**18.09**	**8.20**

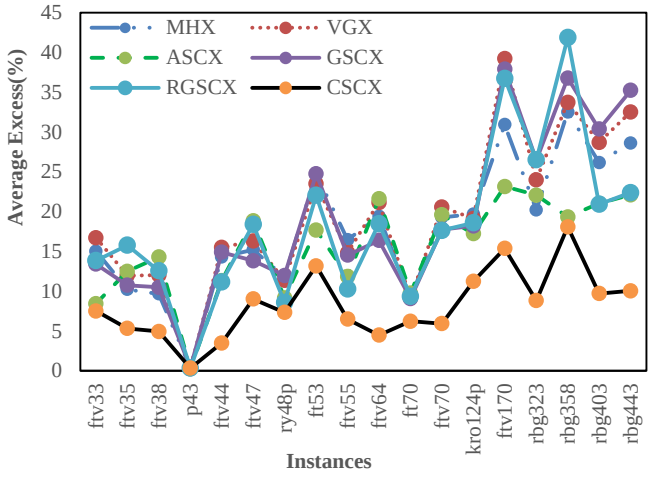

Fig. 2. Average Excess(%) by different GAs for Asymmetric Instances.

To validate the above observations, we also carried out an adequate statistical analysis. By considering that reported results in Table II are random and independent samples, a set of Student's t-tests were conducted. Indeed, for every pair of crossover operator, the hypothesis is tested whether one of the operators is better than the other. The efficiency of an operator is categorized by its average of average excess (%) computed over the all problem instances with best-known solutions.

The results of our hypotheses testing are summarized in Table III. In the table, each row contains two columns, where the first lists a crossover operator and the second column lists its inferior crossover operators. The results are statistically significant at the significance level 0.05 [25]. In Table III, each crossover is ranked according to its number of inferior crossover operators. It is found that there is statistically significant difference between CSCX and other crossover operators at level 0.05, and so, as expected the best ranked crossover is CSCX. Also, the second best is ASCX. No significant difference is found between RGSCX and MHX, as expected, they share the third rank. Also, no significant difference is found between GSCX and VGX, and hence, they share the worst rank.

The Table IV reports results by the GAs for the symmetric TSPLIB instances. The crossover MHX and ASCX obtain lowest best solution cost only for two instances - gr21 and fri26; VGX obtains lowest best solution cost for three instances – dantzig42, eil51 and lin105; GSCX and RGSCX obtain lowest best solution cost for three instances – gr21, fri26 and bayg29; and CSCX obtains lowest best solution cost for six instances – gr21, fri26, berlin52, pr76, pr226 and a280. The crossover operators MHX, VGX, ASCX and GSCX could not obtain lowest average solution cost for any symmetric instance. The proposed RGSCX obtains lowest average solution cost with lowest S.D. for the only one instance bayg29, and CSCX finds lowest average costs along with lower S.D. for remaining 9 instances. So, our proposed crossover CSCX is the best. Also, by looking at average of Avg. Exc (%), one can make it clear that CSCX produces best results, while VGX, RGSCX and GSCX are competing for the second best, and MHX is the worst. The results are also depicted in Figure 3, which also demonstrates the usefulness of our proposed crossover CSCX.

TABLE III. RESULTS OF STATISTICAL HYPOTHESES TESTING ON ASYMMETRIC INSTANCES

Crossover	**Inferior crossovers**
CSCX	MHX, VGX, ASCX, GSCX, RGSCX
ASCX	MHX, VGX, GSCX, RGSCX
RGSCX	VGX, GSCX
MHX	VGX, GSCX
GSCX	------
VGX	------

TABLE IV. RESULTS BY THE CROSSOVER OPERATORS FOR SYMMETRIC TSPLIB INSTANCES

Instance	**n**	**Results**	**MHX**	**VGX**	**ASCX**	**GSCX**	**RGSCX**	**CSCX**
gr21	21	Best Sol	**2707**	2754	**2707**	**2707**	**2707**	**2707**
(2707)		Avg. Sol	2874.78	2927.70	2825.2	2845.28	2829.58	**2806.88**
		Avg. Exc(%)	6.20	8.15	4.37	5.11	4.53	3.69
		S.D.	109.00	89.03	61.29	86.33	87.07	55.14
		Avg. Time	0.03	0.02	0.04	0.01	0.02	0.04
fri26	26	Best Sol	**937**	953	**937**	**937**	**937**	**937**
(937)		Avg. Sol	989.30	987.30	944.04	972.62	969.62	**937.00**
		Avg. Exc(%)	5.58	5.37	0.75	3.80	3.48	0.00
		S.D.	29.86	24.51	11.27	17.72	20.05	0.00
		Avg. Time	0.08	0.03	0.13	0.02	0.03	0.04
bayg29	29	Best Sol	1642	1646	1686	**1634**	**1634**	1639
(1610)		Avg. Sol	1741.78	1767.18	1756.52	1719.42	**1718.42**	1719.54
		Avg. Exc(%)	8.19	9.76	9.10	6.80	6.73	6.80
		S.D.	69.49	55.81	39.51	56.77	49.15	47.16
		Avg. Time	0.08	0.04	0.02	0.03	0.05	0.1
dantzig42	42	Best Sol	753	**714**	754	723	724	723
(699)		Avg. Sol	807.44	788.60	813.36	781.80	792.60	**774.26**
		Avg. Exc(%)	15.51	12.82	16.36	11.85	13.39	10.77
		S.D.	29.64	33.92	22.3	26.27	26.83	23.89
		Avg. Time	0.35	0.24	0.29	0.08	0.14	0.02
eil51	51	Best Sol	444	**432**	444	436	442	437
(426)		Avg. Sol	470.76	463.76	466.66	463.94	462.40	**458.78**
		Avg. Exc(%)	10.51	8.86	9.54	8.91	8.54	7.69
		S.D.	13.31	10.64	8.24	12.01	10.81	9.96
		Avg. Time	0.46	0.49	0.90	0.38	0.37	0.30
berlin52	52	Best Sol	7885	7919	7910	7926	7891	**7646**
(7542)		Avg. Sol	8346.32	8217.40	8429.48	8156.70	8162.72	**7995.52**
		Avg. Exc(%)	10.66	8.96	11.77	8.15	8.23	6.01
		S.D.	247.93	224.64	154.44	189.50	237.35	137.83
		Avg. Time	0.62	0.39	0.70	0.22	0.29	0.22
pr76	76	Best Sol	118331	117411	120729	116844	117724	**113676**
(108159)		Avg. Sol	129036.50	123427.82	128392.50	127293.76	127868.12	**123337.58**
		Avg. Exc(%)	19.30	14.12	18.71	17.69	18.22	14.03
		S.D.	4517.92	3464.34	2791.62	5273.80	5243.46	4268.20
		Avg. Time	0.65	0.78	1.05	0.48	0.53	0.18
lin105	105	Best Sol	16369	**15245**	15978	15921	15627	15622
(14379)		Avg. Sol	17635.34	16616.80	16920.84	17118.08	17068.94	**16575.84**
		Avg. Exc(%)	22.65	15.56	17.68	19.05	18.71	15.28
		S.D.	772.10	635.92	320.38	557.25	637.83	413.79
		Avg. Time	1.34	1.79	1.97	1.10	1.03	0.23
pr226	226	Best Sol	93260	88768	91723	92428	90724	**87477**
(80369)		Avg. Sol	103462.00	93261.56	95511.32	95315.32	94946.68	**90411.98**
		Avg. Exc(%)	28.73	16.04	18.84	18.60	18.14	12.50
		S.D.	6228.71	2996.66	1326.34	4740.22	5643.82	1395.34
		Avg. Time	4.70	5.05	5.09	3.18	3.30	2.02
a280	280	Best Sol	3022	2905	3059	2980	2891	**2833**
(2579)		Avg. Sol	3197.00	3059.62	3179.67	3111.00	3094.10	**2958.34**
		Avg. Exc(%)	23.96	18.64	23.29	20.63	19.97	14.71
		S.D.	69.14	72.46	59.45	99.19	106.69	53.85
		Avg. Time	2.80	4.07	12.56	3.34	4.80	8.08
Average of Avg. Exc (%)			**15.13**	**11.83**	**13.04**	**12.06**	**11.99**	**9.15**

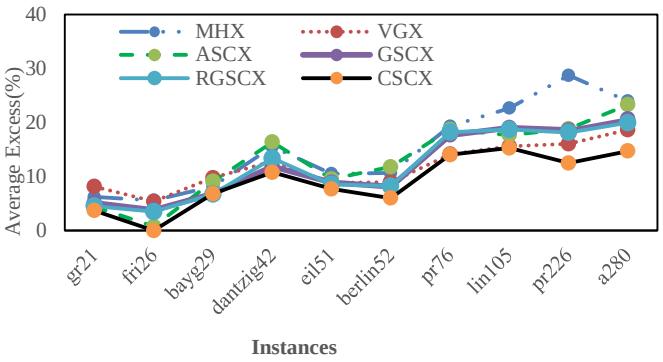

Fig. 3. Average Excess(%) by different GAs for Symmetric Instances.

To validate the above observations, we carried out statistical analysis for these instances also, and the results are summarized in Table V. It is seen that there is statistically significant difference between CSCX and other crossover operators at the significance level 0.05, and so, as expected the best ranked crossover is CSCX. However, there is no significant difference found among VGX, RGSCX, GSCX and ASCX, so, they share the second rank, and MHX is the worst in the rank. From this whole study one can conclude that the proposed crossover CSCX is the best.

TABLE V. RESULTS OF STATISTICAL HYPOTHESES TESTING ON SYMMETRIC INSTANCES

Crossover	Inferior crossovers
CSCX	MHX, VGX, ASCX, GSCX, RGSCX
VGX	MHX
RGSCX	MHX
GSCX	MHX
ASCX	MHX

V. CONCLUSION AND FUTURE WORKS

The crossover operators are classified as distance-based crossover operators and blind crossover operators. There are several crossover operators available in the literature. In this study, we proposed reverse greedy sequential constructive crossover (RGSCX) and then comprehensive sequential constructive crossover (CSCX) for the TSP. To show the usefulness of our proposed crossover operators, we compared with four distance-based crossover operators, such as MHX, VGX, ASCX and GSCX. We applied these crossover operators manually on two chromosomes to produce offspring(s) and found that our proposed crossover CSCX is the best. After that, GAs using all six crossover operators are developed and performed comparative study among them on eighteen asymmetric and ten symmetric TSPLIB instances. In terms of solution quality, it is found that our proposed crossover CSCX is the best. The observation is confirmed by Student's t-test at the significance level 0.05. So, CSCX might be worked good for other associated combinatorial optimization problems. However, the proposed RGSCX could not obtain good solutions and it is competing for third position on asymmetric instances.

In this study, our aim was to propose crossover operators and compare them against some existing crossover operators. It was not aimed to improve solution quality using them, and so, no local search procedure was used for developing state-of-art algorithm for the problem. Also, highest crossover probability was used to display the exact characteristics of operators. Though our proposed CSCX obtains best solutions, still it gets stuck in local minima in the first half of the generations. Hence, good local search along with immigration methods [26-30] may be incorporated to it to develop hybrid genetic algorithm to find better quality solutions to the problem instances, which is under our investigation.

ACKNOWLEDGMENT

The author is very much thankful to the honourable anonymous reviewers for their constructive comments and constructive suggestions which helped the author to improve this paper.

REFERENCES

[1] Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolution Problems," Second Edition, Springer-Verlag, New York, 1994.

[2] C.P. Ravikumar, "Solving large-scale travelling salesperson problems on parallel machines," Microprocessors and Microsystems, vol. 16, no. 3, pp. 149-158, 1992.

[3] S. Arora, "Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems," Journal of ACM, vol. 45, no. 5, pp. 753–782, 1998.

[4] Z.H. Ahmed, "Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator," International Journal of Biometrics & Bioinformatics, vol. 3, pp. 96-105, 2010.

[5] D.E. Goldberg, "Genetic algorithms in search, optimization, and machine learning," Addison-Wesley, New York, 1989.

[6] Z.H. Ahmed, "Solving the traveling salesman problem using greedy sequential constructive crossover in a genetic algorithm," IJCSNS International Journal of Computer Science and Network Security, vol. 20, no. 2, pp. 99-112, 2020.

[7] D.E. Goldberg, and R. Lingle, "Alleles, loci and the travelling salesman problem," In J.J. Grefenstette (ed.) Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications. Lawrence Erlbaum Associates, Hilladale, NJ, 1985.

[8] L. Davis, "Job-shop scheduling with genetic algorithms," Proceedings of an International Conference on Genetic Algorithms and Their Applications, pp. 136-140, 1985.

[9] G. Syswerda, "Schedule optimization using genetic algorithms," In Davis, L. (ed.) Handbook of Genetic Algorithms, New York: Van Nostrand Reinhold, pp. 332–349, 1991.

[10] J. Grefenstette, R. Gopal, B. Rosmaita, and D. Gucht, "Genetic algorithms for the traveling salesman problem," In Proceedings of the First International Conference on Genetic Algorithms and Their Applications, (J. J. Grefenstette, Ed.), Lawrence Erlbaum Associates, Mahwah NJ, pp. 160–168, 1985.

[11] I.M. Oliver, D. J. Smith and J.R.C. Holland, "A study of permutation crossover operators on the travelling salesman problem," In J.J. Grefenstette (ed.). Genetic Algorithms and Their Applications: Proceedings of the 2nd International Conference on Genetic Algorithms. Lawrence Erlbaum Associates, Hilladale, NJ, 1987.

[12] D. Whitley, T. Starkweather and D. Shaner, "The traveling salesman and sequence scheduling: quality solutions using genetic edge recombination," In L. Davis (Ed.) Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York, pp. 350-372, 1991.

[13] N.J. Radcliffe and P.D. Surry, "Formae and variance of fitness," In D. Whitley and M. Vose (Eds.) Foundations of Genetic Algorithms 3, Morgan Kaufmann, San Mateo, CA, pp. 51-72, 1995.

[14] J. J. Grefenstette, "Incorporating problem specific knowledge into genetic algorithms," In L. Davis (Ed.), Genetic algorithms and simulated annealing, London, UK: Pitman / Pearson, pp. 42–60, 1987.

[15] B. Freisleben and P. Merz, "A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems," in Proc. the 1996 IEEE International Conference on Evolutionary Computation, (Nagoya, Japan), pp.616-621, 1996.

[16] E. Osaba, R. Carballedo, F. Diaz, E. Onieva, A.D. Masegosa, and A.Perallos, "Good practice proposal for the implementation, presentation, and comparison of metaheuristics for solving routing problems," Neurocomputing, vol. 271, no. 3, pp. 2-8, 2018.

[17] G. E. Liepins, M. R. Hilliard, M. Palmer and M. Morrow, "Greedy genetics," in Proc. the Second International Conference on Genetic algorithms and their application, pp.90-99, October 1987.

[18] P. Jog, J.Y. Suh and D. Van Gucht, "The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem," in Proc. 3rd Int. Conf. Genetic Algorithms, Morgan Kaufmann Publishers, pp. 110–115, 1989.

[19] B.A. Julstrom, "Very greedy crossover in a genetic algorithm for the traveling salesman problem," In: ACM symposium on Applied computing, pp. 324-328, 1995.

[20] Z.H. Ahmed, "Improved genetic algorithms for the traveling salesman problem," International Journal of Process Management and Benchmarking, vol. 4, no. 1, pp. 109-124, 2014.

[21] I. H. Khan, "Assessing different crossover operators for travelling salesman problem," IJISA International Journal of Intelligent Systems and Applications, vol. 7, no. 11, pp. 19-25, 2015.

[22] S. Kang, S.-S. Kim, J.-H. Won, and Y.-M. Kang, "Bidirectional constructive crossover for evolutionary approach to travelling salesman problem," 2015 5th IEEE International Conference on IT Convergence and Security (ICITCS), pp. 1-4, 2015.

[23] Z.H. Ahmed, "Adaptive sequential constructive crossover operator in a genetic algorithm for solving the traveling salesman problem," IJACSA International Journal of Advanced Computer Science and Applications, vol. 11, no. 2, pp. 593-605, 2020.

[24] G. Reinelt, TSPLIB, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

[25] M. Spiegel and L. Stephens, "Schaum's Outline of Statistics," Fourth Edition, McGraw-Hill, New York, 2011.

[26] Z.H. Ahmed, "A hybrid genetic algorithm for the bottleneck traveling salesman problem," ACM Transactions on Embedded Computing Systems, vol. 12, Art. No. 9, 2013.

[27] Z.H. Ahmed, "An experimental study of a hybrid genetic algorithm for the maximum travelling salesman problem," Mathematical Sciences, vol. 7, pp. 1-7, 2013.

[28] Z.H. Ahmed, "The ordered clustered travelling salesman problem: A hybrid genetic algorithm," The Scientific World Journal, vol. 2014, Art ID 258207, 13 pages, 2014.

[29] Z.H. Ahmed, "The minimum latency problem: a hybrid genetic algorithm," IJCSNS International Journal of Computer Science and Network Security, vol. 18, no. 11, pp. 153-158, 2018.

[30] Z.H. Ahmed, "Performance analysis of hybrid genetic algorithms for the generalized assignment problem," IJCSNS International Journal of Computer Science and Network Security, vol. 19, no. 9, pp. 216-222, 2019.