Adaptive Sequential Constructive Crossover Operator in a Genetic Algorithm for Solving the Traveling 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*—Genetic algorithms are widely used metaheuristic algorithms to solve combinatorial optimization problems that are constructed on the survival of the fittest theory. They obtain near optimal solution in a reasonable computational time, but do not guarantee the optimality of the solution. They start with random initial population of chromosomes, and operate three different operators, namely, selection, crossover and mutation, to produce new and hopefully better populations in consecutive generations. Out of the three operators, crossover operator is the most important operator. There are many existing crossover operators in the literature. In this paper, we propose a new crossover operator, named adaptive sequential constructive crossover, to solve the benchmark travelling salesman problem. We then compare the efficiency of the proposed crossover operator with some existing crossover operators like greedy crossover, sequential constructive crossover, partially mapped crossover operators, etc., under same genetic settings, for solving the problem on some benchmark TSPLIB instances. The experimental study shows the effectiveness of our proposed crossover operator for the problem and it is found to be the best crossover operator.**

***Keywords*—Genetic algorithm; adaptive sequential constructive crossover; traveling salesman problem; NP-hard**

I. Introduction

The usual travelling salesman problem (TSP) is very famous combinatorial optimization problem that finds a least cost Hamiltonian cycle in a network. The RAND Corporation introduced the TSP in 1948. The Corporation's reputation helped to make the TSP well-known and popular problem. The TSP also became popular at that time due to the new subject of linear programming and attempts to solve combinatorial optimization problems. It can be stated as.

A network with 'n' nodes, with 'node 1' as 'depot' and a travel cost (or distance, or travel time etc.,) matrix C= $[c_{ij}]$ of order n associated with ordered pairs (i, j) of nodes is given. The problem is to find a least cost Hamiltonian cycle. Based on the structure of the cost matrix, the TSPs are classified into two types as symmetric and asymmetric. If $c_{ij} = c_{ji}$, $\forall$ i, j, the TSP is symmetric, otherwise, it is asymmetric. For asymmetric TSP with n nodes, there are $(n-1)!$ possible solutions with at least one of them provide the minimum cost. For symmetric TSP, there are $\frac{(n-1)!}{2}$ possible solutions along with same valued reverse cyclic permutations. If there are only 10 nodes, then there are 362,880 and 181,440 tours for asymmetric TSP and symmetric TSP, respectively. The number of possible solutions in both types is very large for any size, n; so, a complete search is very difficult, if it is not impossible. That means, the problem is very difficult to solve. The TSP has been researched by several researchers for mainly three reasons. First, it can model many real-life problems. Second, it is NP-Hard [1]. Third, NP-Hard problems are so difficult that no one has found any efficient algorithm to solve them for large sized problem instances. Also, NP-hard problems are equivalent to each other; so, if one can develop efficient algorithm for solving one of them, then one could develop efficient algorithm for others.

The TSP has application in several situations such as automatic drilling of printed circuit boards and threading of scan cells in a testable very-large-scale-integrated (VLSI) circuit, automatic drilling of printed circuit boards and circuits, computer wiring, X-ray crystallography, movement of people [2].

Several exact and heuristic/metaheuristic algorithms have been reported for solving the TSP. Branch and bound [3], branch and cut [4], and lexisearch algorithm [5] are some exact algorithms. These algorithms provide the exact optimal solution to the problem, but as the problem size increases computational time increases exponentially. As reported by Deng et al. [6] only small sized TSP instances can be solved to exact optimality. Since some practically large problem instances must be solved, hence it is important to obtain heuristically optimal solution by ensuring the quality of the solution in reasonable time, rather to obtain exact optimal solution in hell of time. Heuristic/metaheuristics algorithms give near optimal solution in a reasonable computational time, but do not guarantee the optimality of the solution. Example of metaheuristic algorithms are ant colony optimization [7], genetic algorithm [8], simulated annealing [9], state transition algorithm [10], tabu search [11], artificial neural network [12], artificial bee colony [13], black hole [14], and particle swarm optimization [7]. Out of these metaheuristic algorithms, genetic algorithm (GA) is one of the best and widely used algorithm to solve the TSP as well as other combinatorial optimization problems in computer science and operations research.

GAs are proposed by John Holland in 1970s which are based on imitating Darwin's theory of 'the survival of the fittest' in natural biology [8]. To solve a real-world problem using GAs, two most important conditions are to be fulfilled: (i) a chromosome representing a solution, and (ii) an objective/fitness function can be defined. Any simple GA begins with random initial population, called gene pool of chromosomes, and operates three different operators to produce new, usually better, populations in consecutive iterations/generations. Selection is the 1st operator in which chromosomes are duplicated to next generation probabilistically. Crossover is the 2nd operator in which couples of chromosomes are selected randomly and mated to produce new and better chromosomes. Mutation is the 3rd operator which alters occasionally a chromosome position value. Crossover along with selection operator is the main leading procedure in GAs. Mutation expands search space and defends from loss of any genetic substance due to selection and crossover operators.

Though GA is one of the best algorithms, however, its performance depends on initial population, selection, crossover and mutation operators, and some parameters such as population size, crossover probability, mutation probability and stopping condition (Goldberg, 1989). Among different operators, crossover plays very important role in GAs, and accordingly many crossover operators have been developed and reported in the literature for solving the TSP [15]. This paper aims to propose a modified version of sequential constructive crossover (SCX) [16] named adaptive sequential constructive crossover (ASCX) and then compare with eight crossover operators including SCX to assess suitability for the TSP.

This paper is organized as follows: Section II discusses GAs using some existing crossover operators and our proposed crossover operator, named adaptive crossover sequential constructive crossover operator, for the TSP, while design of variant GAs is discussed in Section III. Section IV describes computational experiences for sixteen variant GAs using eight crossover operators with two possibilities of mutation operator and discussions. Finally, Section V presents concluding remarks and future works.

II. Genetic Algorithms for the TSP

For applying GA to any optimization problem, one must find a way for representing solutions as legal chromosomes such that crossovers of legal chromosomes result in legal chromosomes. The methods for representing solutions differ by problem and, contain a certain art. There are many representation methods for solving the TSP using GAs. Some of them are binary, adjacency, ordinal, matrix and path representations. We consider only the path representation that simply lists the node labels such that no node can appear twice in the same chromosome. For example, let {1, 2, 3, 4, 5, 6, 7, 8, 9} be the node labels in a 9-node instance, then a tour {1→ 9→6→ 2→7 → 4→3→8→ 5 →1} may be represented as (1, 9, 6, 2, 7, 4, 3, 8, 5). The objective function is the sum of the costs of all edges in the tour.

A. Initial Population and Selection Operator

In GAs, after generating the random population of chromosomes, selection operator is applied. In selection operator, chromosomes are copied into mating pool with a probability related to their fitness value. By transferring highly fit chromosomes to next generation mating pool, selection mimics the Darwinian theory of survival-of-the-fittest in the natural biology. In natural biology, fitness is determined by an individual's capability to survive predators, epidemic, and other difficulties to maturity and following selection. In this stage no new chromosome is created. The commonly used selection operator is the proportionate selection operator, where an individual is selected for the mating pool according to a probability related to its fitness value. We have considered the stochastic remainder selection process [17] for our GAs.

B. Existing Crossover Operators

Since the crossover operator plays a vital role in GA, so many crossover operators have been proposed for the TSP. However, the traditional crossover operators such as one-point, two-point, and uniform crossover operators are not suitable for the TSP. Two kinds of crossover operators have been developed for the TSP – distance-based and blind crossover operators [18]. We consider some of them from both kinds and compare our proposed crossover operator with them.

1). Partially mapped crossover operator: Goldberg and Lingle [19] developed the partially mapped crossover (PMX) that used two crossover points. It defines an interchange mapping in the section between these points. PMX was the first crossover for the GA to solve the TSP. Consider, for example, the two parent chromosomes P_1: (1, 2, 3, 4, 6, 9, 5, 7, 8) and P_2: (1, 3, 5, 7, 8, 9, 4, 2, 6). We shall consider same pair of chromosomes for illustrating all the crossover operators considered here. Also, we fix headquarters (first gene) as 'node 1'. Suppose the randomly selected cut points are between 3rd and 4th genes and between 7th and 8th genes as follows (these cut points are marked with "|"):

P_1: (1, 2, 3 | 4, 6, 9, 5 | 7, 8) and

P_2: (1, 3, 5 | 7, 8, 9, 4 | 2, 6)

We always fix first gene as 'node 1'. The mapping sections are between the cut points. In this example, the mapping systems are 4↔7, 6↔8, 9↔9, and 5↔4. Now these mapping sections are copied with each other to build offsprings as follows:

O_1: (1, *, * | 7, 8, 9, 4 | *, *),

O_2: (1, *, * | 4, 6, 9, 5 | *, *)

Then we can add more genes from the original parents which do not result any conflict as follows:

O_1: (1, 2, 3 | 7, 8, 9, 4 | *, *),

O_2: (1, 3, * | 4, 6, 9, 5 | 2, *)

The first * in the first offspring should be 7 that comes from first parent, but it is already present in this offspring, so

we check mapping 4↔ 7, but 4 is also present in this offspring, again check mapping 5↔4, so 5 is added. Similarly, the second * in first offspring should be 8 that comes from first parent, but it is present in this offspring, so we check mapping 6↔8 and hence, we add 6 at second *. Thus, the first offspring becomes

O_1: (1, 2, 3 | 7, 8, 9, 4 | 5, 6),

Similarly, we build the second offspring as:

O_2: (1, 3, 7 | 4, 6, 9, 5 | 2, 8)

2). Ordered crossover operator: Davis [20] developed the ordered crossover (OX) that builds offspring by choosing a subsequence of a tour from one parent and preserving the relative order of nodes from the other parent. Consider the same example parent chromosomes with randomly chosen two cut points marked by "|":

P_1: (1, 2, 3 | 4, 6, 9, 5 | 7, 8) and

P_2: (1, 3, 5 | 7, 8, 9, 4 | 2, 6)

We always fix first gene as 'node 1'. At first, the offsprings are built by copying the genes between the cuts with similar way into the offsprings that lead the offsprings as:

O_1: (1, *, * | 4, 6, 9, 5 | *, *),

O_2: (1, *, * | 7, 8, 9, 4 | *, *)

Then beginning from the second cut point of one parent, the genes from the other parent are copied in the same order except the existing genes. The sequence of the genes in the second parent from the second cut point is "2 →6→3→5→7→8 →9 →4." After omitting the genes 4, 6, 9 and 5 that are already present in the first offspring, the sequence becomes "2→3→7→8", which is placed in the first offspring beginning from the second cut point:

O_1: (1, 7, 8 | 4, 6, 9, 5 | 2, 3).

Similarly, we build the second offspring as:

O_2: (1, 6, 5 | 7, 8, 9, 4 | 2, 3)

3). Alternating edges crossover operator: Grefenstette et al. [21] proposed alternating edges crossover (AEX) operator that assumes a chromosome as a directed cycle of arcs. Only one offspring is built by selecting alternative arcs from both parents, with some additional random selections in case of infeasibility. Consider the same example parent chromosomes P_1: (1, 2, 3, 4, 6, 9, 5, 7, 8) and P_2: (1, 3, 5, 7, 8, 9, 4, 2, 6).

First, the arc (1, 2) is first selected from the first parent and copied to the offspring. Then the arcs (2, 6) from second parent, (6, 9) from first parent and (9, 4) from second parent are selected and copied to the offspring. Then, arc (4, 6) is selected from first parent, however, this arc produces a cycle and a new arc leaving the node 4 to a node not yet visited is selected randomly. Suppose the arc (4, 3) is chosen. Then, the arcs (3, 5) from second parent, (5, 7) from first parent and (7, 8) from second parents are selected. This way the offspring is built as follows:

O: (1, 2, 6, 9, 4, 3, 5, 7, 8)

All arcs in the offspring are inherited from the parents, apart from the arc (4, 3).

4). Cycle crossover operator: Oliver et al. [22] developed cycle crossover (CX) that builds an offspring where every node and its corresponding position originated from one of the parents. Consider the same example parent chromosomes P_1: (1, 2, 3, 4, 6, 9, 5, 7, 8) and P_2: (1, 3, 5, 7, 8, 9, 4, 2, 6).

As we fix first gene as node 1, for the next position, we select randomly between 2 and 3. Suppose we select node 2, then the offspring becomes:

O_1: (1, 2, *, *, *, *, *, *, *)

Every gene in the offspring is taken from one of its parents with the same position, so the next gene to be considered must be bit 3, as this gene from the second parent is just below the selected gene 2. In the first parent this gene is at 3rd position; thus, the offspring becomes:

O_1: (1, 2, 3, *, *, *, *, *, *)

Next gene will be 5 of second parent as it is just below the current gene 3, which is present at 7th position in first parent. Thus, the offspring becomes:

O_1: (1, 2, 3, *, *, *, 5, *, *)

Next gene will be 4 of second parent as it is just below the current gene 5, which is present at 4th position in first parent. Thus, the offspring becomes:

O_1: (1, 2, 3, 4, *, *, 5, *, *)

Next gene will be 7 of second parent as it is just below the current gene 4, which is present at 8th position in first parent. Thus, the offspring becomes:

O_1: (1, 2, 3, 4, *, *, 5, 7, *)

Next, we have node 2, which is already present in the offspring; thus, we have completed a cycle and hence, we fill the remaining blank positions with the genes of those positions which are present in second parent. This way the offspring is built as follows:

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

Similarly, we build the second offspring as (same as P_2):

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

5). Greedy crossover operator: Grefenstette et al. [21] also proposed greedy crossover (GX) for the TSP that selects a starting node randomly. Then in each step, four neighbor nodes of currently selected node in both parents are considered, and the cheapest one (not present in the offspring) is selected. If the cheapest node or all four neighbour nodes are present in the offspring, then any node from the remaining is selected randomly. This operator creates only one offspring from two parents. Let us illustrate the GX through the 9-node example given as cost matrix in Table I and the same parent chromosomes considered above.

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

As we fixed first gene as 'node 1', the offspring is initiated as (1). The nodes 2 and 3 are neighbours of 1 with their costs 7 and 15 respectively. The node 2 is cheaper, so, it is copied into the incomplete offspring: (1, 2).

Next, the nodes 3, 1, 6 and 4 are neighbours of 2 with their costs 8, 11, 3 and 7 respectively. The node 6 is the cheapest, so, it is copied into the incomplete offspring: (1, 2, 6).

Next, the nodes 9, 4, 1 and 2 are neighbours of 6 with their costs 5, 11, 6 and 13 respectively. The node 9 is the cheapest so, it is copied into the incomplete offspring: (1, 2, 6, 9).

Next, the nodes 5, 6, 4 and 8 are neighbours of 9 with their costs 10, 7, 9 and 10 respectively. The node 6 is the cheapest, but it is already present in the offspring, so, node 3 is selected randomly and it is copied into the incomplete offspring: (1, 2, 6, 9, 3).

Next, the nodes 4, 2, 5 and 1 are neighbours of 3 with their costs 16, 5, 12 and 15 respectively. The node 2 is the cheapest, but it is already present in the offspring, so, node 4 is selected randomly and it is copied into the incomplete offspring: (1, 2, 6, 9, 3, 4).

Continuing in this way, we have the complete offspring: (1, 2, 6, 9, 3, 4, 5, 7, 8) with cost 67.

6). Sequential constructive crossover operator: Ahmed [16] proposed the sequential constructive crossover (SCX) operator which is modified in [23] that constructs an offspring using better arcs based on their cost present in the parents' structure. Furthermore, it also uses the better arcs that are present neither in the parents' structure. SCX sequentially searches both parent chromosomes and considers the first legitimate node (i.e. unvisited node) that appeared after the present node and in case, if no legitimate node is found in either of the parent chromosomes, it sequentially searches from the beginning of the chromosome and then compares their associated cost to decide the next node of the child chromosome. The SCX is compared with edge recombination crossover (ERX) and generalized N-point crossover (GNX) on symmetric and asymmetric TSPLIB instances. As reported, SCX is better than ERX and GNX. Khan [24] presented a comparative study among eight different crossover operators, namely, Two-Point Crossover, PMX, CX, Shuffle Crossover, ERX, Uniform Order-based Crossover, Sub-tour Exchange Crossover and SCX, and found that SCX outperformed other operators in achieving good quality solution for the TSP. Further, SCX is successfully applied to many other combinatorial optimization problems ([25]-[31], [32]). Let us recall the algorithm for the SCX [23].

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 no 'legitimate node' after 'node p' is present in any of the parents, search sequentially from the starting of the parent and consider the first 'legitimate node', and go to Step 3.

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.

Let us illustrate the SCX through the same example given above. Select 'node 1' as the 1st gene. The legitimate nodes after node 1 in P_1 and P_2 are 2 and 3 respectively with $c_{12}=7$ and $c_{13}=15$. Since $c_{12}<c_{13}$, we accept node 2. So, the partially constructed chromosome will be (1, 2).

The legitimate nodes after node in P_1 and P_2 are nodes 3 and 6 respectively with $c_{23}=8$ and $c_{26}=3$. Since $c_{26}<c_{23}$, we accept node 6. So, the partially constructed chromosome will be (1, 2, 6).

The legitimate node after node 6 in P_1 is 9 with $c_{69}=5$, but none in P_2. So, for P_2, we sequentially search from the beginning of the chromosome and find the first legitimate node 3 with $c_{63}=8$. Since $c_{69}<c_{63}$, we accept node 9. So, the partially constructed chromosome will be (1, 2, 6, 9).

The legitimate nodes after node 9 in P_1 and P_2 are 5 and 4 respectively with $c_{95}=10$ and $c_{94}=9$. Since $c_{94}<c_{95}$, we accept node 4. So, the partially constructed chromosome will be (1, 2, 6, 9, 4).

The legitimate node after node 4 in P_1 is 5 with $c_{45}=9$, but none in P_2. So, for P_2, we sequentially search from the beginning of the chromosome and find the first legitimate node 3 with $c_{43}=11$. Since $c_{45}<c_{43}$, we accept node 5. So, the partially constructed chromosome will be (1, 2, 6, 9, 4, 5).

Continuing this way, we obtain the complete offspring chromosome: (1, 2, 6, 9, 4, 5, 7, 8, 3) with cost 72.

7). Bidirectional circular sequential constructive crossover operator: The bidirectional circular sequential constructive crossover (BCSCX) was proposed by Kang et al. [33] to modify SCX that searches for next neighbor in both left and right directions in both parents. Thus, four neighbor genes are considered. Also, during searching for the next neighbor gene, if it reaches to the end or to the beginning of

the genes list in any of the parents, it will wrap around. Let us illustrate the BCSCX through the same example given above.

Select 'node 1' as the 1st gene. The legitimate nodes after node 1 in both directions in P_1 are 2 and 8 (after wrapping around), and in P_2 are 3 and 6 (after wrapping around), with their costs 7, 9, 15 and 6 respectively. We accept node 6 as it is cheapest. So, the partially constructed offspring chromosome will be (1, 6).

The legitimate nodes after node 6 in both directions in P_1 are 9 and 4, and in P_2 are 3 (after wrapping around) and 2, with their costs 5, 11, 8 and 13 respectively. We accept node 9 as it is cheapest. So, the partially constructed chromosome will be (1, 6, 9).

The legitimate nodes after node 9 in both directions in P_1 are 5 and 4, and in P_2 are 4 and 8, with their costs 10, 9, 9 and 10 respectively. We accept node 4 and the partially constructed chromosome will be (1, 6, 9, 4).

The legitimate nodes after node 4 in both directions in P_1 are 5 and 3, and in P_2 are 2 and 8, with their costs 9, 11, 5 and 4 respectively. We accept node 8 and the partially constructed chromosome will be (1, 6, 9, 4, 8).

Continuing this way, we obtain the complete offspring chromosome: (1, 6, 9, 4, 8, 2, 7, 3, 5) with cost 56.

Among the above discussed crossover operators GX, SCX and BCSCX are called distance-based crossover operators because they care about the distance between nodes. On the other hand, crossover operators like PMX, OX, AEX, CX, GNX and ERX are called blind crossover operators because they only concern about to satisfy the constraints of the problem and do not use any information associated with the problem [18]. We propose to compare our proposed ASCX against both kind of crossover operators.

C. Proposed Crossover Operator: Adaptive Sequential Constructive Crossover Operator

We are going to propose a modification of the SCX operator, named adaptive SCX (ASCX). In BCSCX, four neighbor genes are considered. We propose to construct offspring either in forward direction from the first gene or in backward direction from the last gene or in mixed direction adaptively depending on the cost of the next node. Hence, we consider a total of eight neighbour nodes of a current node, four nodes for each of the two genes (nodes). Since there are n genes in a chromosome, we select 'node 1' as the first and $(n+1)^{th}$ (it is not shown in the chromosome) genes. Let us define the algorithm for the ASCX as follows.

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 1st and 2nd 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 1st and 2nd 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 1st and 2nd 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 1st and 2nd 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.

Let us illustrate the ASCX through the same example parent chromosomes given above. Since there are 9 genes in the parent chromosomes, we select 'node 1' as the first and 10^{th} gene (it is not shown in the chromosome). The legitimate nodes after first gene, node 1, in both directions in P_1 are 2 and 8 (after wrapping around), and in P_2 are 3 and 6 (after wrapping around), with their costs 7, 9, 15 and 6 respectively. Among them node 6 with cost 6 is the cheapest. On the other hand, the legitimate nodes before 10^{th} gene, node 1 (though it is not shown in the chromosome), in both directions in P_1 are 8 and 2 (after wrapping around), and in P_2 are 6 and 3 (after wrapping around), with their costs 9, 7, 6 and 15 respectively. Among them node 6 with cost 6 is the cheapest. Since both cheapest nodes are same 6, we add it as the 2nd gene in the offspring, and hence the partially constructed offspring chromosome will be (1, 6, *, *, *, *, *, *, *).

The legitimate nodes after 2nd gene, node 6, in both directions in P_1 are 9 and 4, and in P_2 are 3 (after wrapping around) and 2, with their costs 5, 11, 8 and 13 respectively. Among them node 9 with cost 5 is the cheapest. On the other hand, the legitimate nodes before 10^{th} gene, node 1, in both directions in P_1 are 8 and 2 (after wrapping around), and in P_2

are 2 and 3 (after wrapping around), with their costs 9, 11, 11 and 15 respectively. Among them node 8 with cost 9 is the cheapest. Since between two cheapest nodes, node 9 is cheaper, we add it as the 3rd gene in the offspring, and hence the partially constructed offspring chromosome will be (1, 6, 9, *, *, *, *, *, *).

The legitimate nodes after 3rd gene, node 9, in both directions in P_1 are 5 and 4, and in P_2 are 4 and 8, with their costs 10, 9, 9 and 10 respectively. Among them node 4 with cost 9 is the cheapest. On the other hand, the legitimate nodes before 10th gene, node 1, in both directions in P_1 are 8 and 2 (after wrapping around), and in P_2 are 2 and 3 (after wrapping around), with their costs 9, 11, 11 and 15 respectively. Among them node 8 with cost 9 is the cheapest. Since both cheapest nodes have same costs, we add node 4 as the 4th gene in the offspring, and hence the partially constructed offspring chromosome will be (1, 6, 9, 4, *, *, *, *, *).

The legitimate nodes after 4th gene, node 4, in both directions in P_1 are 5 and 3, and in P_2 are 2 and 8, with their costs 9, 11, 5 and 4 respectively. Among them node 8 with cost 4 is the cheapest. On the other hand, the legitimate nodes before 10th gene, node 1, in both directions in P_1 are 8 and 2 (after wrapping around), and in P_2 are 2 and 3 (after wrapping around), with their costs 9, 11, 11 and 15 respectively. Among them node 8 with cost 9 is the cheapest. Since between two cheapest nodes, node 8 is cheaper, we add it as the 5th gene in the offspring, and hence the partially constructed offspring chromosome will be (1, 6, 9, 4, 8, *, *, *, *).

The legitimate nodes after 5th gene, node 8, in both directions in P_1 are 2 (after wrapping around) and 7, and in P_2 are 2 and 7, with their costs 3, 8, 3 and 8 respectively. Among them node 2 with cost 3 is the cheapest. On the other hand, the legitimate nodes before 10th gene, node 1, in both directions in P_1 are 7 and 2 (after wrapping around), and in P_2 are 2 and 3 (after wrapping around), with their costs 5, 15, 5 and 15 respectively. Among them node 7 with cost 5 is the cheapest. Since between two cheapest nodes, node 2 is cheaper, we add it as the 6th gene in the offspring, and hence the partially constructed offspring chromosome will be (1, 6, 9, 4, 8, 2, *, *, *).

The legitimate nodes after 6th gene, node 2, in both directions in P_1 are 3 and 7 (after wrapping around), and in P_2 are 3 (after wrapping around) and 7, with their costs 8, 6, 8 and 6 respectively. Among them node 7 with cost 6 is the cheapest. On the other hand, the legitimate nodes before 10th gene, node 1, in both directions in P_1 are 7 and 3 (after wrapping around), and in P_2 are 7 and 3 (after wrapping around), with their costs 5, 15, 5 and 15 respectively. Among them node 7 with cost 5 is the cheapest. Since between two cheapest nodes, node 7 is cheaper, we add it as the 9th gene in the offspring, and hence the partially constructed offspring chromosome will be (1, 6, 9, 4, 8, 2, *, *, 7).

The legitimate nodes after 6th gene, node 2, in both directions in P_1 are 3 and 5 (after wrapping around), and in P_2 are 3 (after wrapping around) and 5, with their costs 8, 11, 8 and 11 respectively. Among them node 3 with cost 8 is the cheapest. On the other hand, the legitimate nodes before 9th gene, node 7, in both directions in P_1 are 5 and 3 (after wrapping around), and in P_2 are 5 and 3 (after wrapping around), with their costs 7, 8, 7 and 8 respectively. Among them node 5 with cost 7 is the cheapest. Since between two cheapest nodes, node 5 is cheaper, we add it as the 8th gene in the offspring, and hence the partially constructed offspring chromosome will be (1, 6, 9, 4, 8, 2, *, 5, 7).

Continuing this way, we obtain the complete offspring chromosome: (1, 6, 9, 4, 8, 2, 3, 5, 7) with cost 59.

D. Mutation Operator

After applying crossover operator, mutation operator is applied. The mutation operator randomly selects a position in the chromosome and changes the corresponding allele (value of a gene), thereby modifying information. The need for mutation comes from the fact that as the less fit members of successive generations are discarded; some aspects of genetic material could be lost forever. By performing occasional random changes in the chromosomes, GAs ensure that new parts of the search space are reached, which selection and crossover alone couldn't fully guarantee. In doing so, mutation ensures that no important features are prematurely lost, thus maintaining the mating pool diversity. For the TSP, the classical mutation operator does not work. For this investigation, we have considered the reciprocal exchange mutation that selects two nodes randomly and swaps them.

III. Design of Our Genetic Algorithms

A simple GA may be summarized as follows:

Step 1: Create initial random population of chromosomes of size Ps and set generation = 0.

Step 2: Evaluate the population.

Step 3: Set generation = generation + 1 and select good chromosomes by selection procedure.

Step 4: Perform crossover with crossover probability P_c.

Step 5: Perform bit-wise mutation with mutation probability P_m.

Step 6: Replace old population with new one.

Step 7: Repeat Steps 2 to 6 until the terminating criterion is satisfied.

As suggested in [18] if the performance of the distance-based crossover is compared with blind crossovers, the comparison is not going to be as fair as it should be. So, we consider both types of crossover operators. There are eight possible selections for crossover operator, which are: PMX, OX, AEX, CX, GX, SCX, BCSCX and ASCX respectively. Within one selection, a single crossover operator is executed.

However, we apply two possibilities of selecting mutation–presence or absence of mutation. There are eight possible selections for crossover operator along with two possibilities of mutation, thus providing altogether sixteen variants of GAs. The goal of such separate execution is to measure effectiveness of specific operator and to find their comparative ranking. Note that each variant GA is purely simple or non-hybrid, which is built of GA procedures and operators, and it does not combine elements of any other

heuristic or metaheuristic algorithm. However, GA search process is guided by some parameters, namely, population size that determines number of chromosomes in a population, crossover probability that states the probability of performing crossover between two parent chromosomes, mutation probability that specifies the probability of performing bit-wise mutation, and termination condition that specifies condition to stop the GA search.

IV. Computational Experiences and Discussions

In order to compare the efficiency of the different crossover operators, variant GAs using different crossovers have been encoded in Visual C++ on a Laptop with i3-3217U CPU@1.80 GHz and 4 GB RAM under MS Windows 7, and run for twelve benchmark TSPLIB instances [34]. In these twelve problem instances, the ftv33, ftv38, ft53, kro124p, ftv170, rbg323, rbg358, rbg403 and rbg443 are asymmetric, and gr21, fri26 and dantzig42 are symmetric TSPs. Initial population of chromosomes is generated randomly. The following common parameters are selected for all algorithms: population size is 50, crossover probability is 1.0 (i.e., 100%), mutation probability is 0.09 (i.e., 9%), and maximum of 1,000 generations is the terminating condition. Though GA is structured, yet randomized, so, its repeated execution on the same input data with the same number of procedures usually gives slightly different results. To compensate this randomization effect, the experiments have been repeated 50 times for each instance. The results of experiments by the sixteen GA variants are summarized in Tables II and IV. All tables are organized in the same way: a row corresponds to a problem instance (its best known solution is reported within brackets) and a column to a GA variant considered by a certain selection of crossover operator. Thus, a table entry presents the summary of results of the corresponding instance by the corresponding GA variant. The result is described by its best solution cost, average solution cost, average percentage of excess to the best known solution, standard deviation (S.D.) of costs, and average convergence time (in second). The best result for a chosen instance over all variants is marked by bold face. The percentage of excess above the best known solution, reported in TSPLIB website, is given by the

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

Fig. 1 and Fig. 2 present results for the instance ftv170 (considering only 30 generations). Fig. 1 refers to the GA variants without mutation, and Fig. 2 to the variants with mutation, respectively. In both figures, each graph corresponds to a crossover operator, and it shows how the current solution improves depending on the number of generations. Only the three best performing crossover operators, namely, SCX, BCSCX and ASCX, are reported.

In the figures, the labels on the left margin denote the solution cost, while the labels on the right margin refer to percentage of excess to the best known solution (Excess (%)). All crossover operators have some randomized factors that make them more effective when trying to add an allele. The more randomized these operators are, the more possibilities of progress should have. Fig. 1 shows that SCX has some variations, but it is not the best. Though BCSCX and ASCX have less variations and are competing each other, still ASCX provides us best results. But it has limited variation range and gets stuck in local minimums very quickly. From Fig. 2, it is observed that mutation always improves performance by helping crossovers to escape from local minima.

Table II reports results by the eight GA variants where mutation is not applied. With respect to the average cost, it is very clear from Table II that distance-based crossovers are far better than blind crossovers. Among the crossovers, GX, SCX and BCSCX obtain lowest average cost with lowest S.D for the instances danzig42, gr21 and fr26 respectively. The crossovers SCX and BCSCX are competing. The proposed crossover ASCX obtains lowest average costs with lower S.D. for remaining nine instances, namely, ftv33, ftv38, ft53, kro124p, ftv170, rbg323, rbg358, rbg403 and rbg443. So, among all the crossovers ASCX is found to be the best. Based on best solution costs also ASCX is found to be the best. The results are depicted in Fig. 3, which also shows the effectiveness of our proposed crossover operator ASCX.

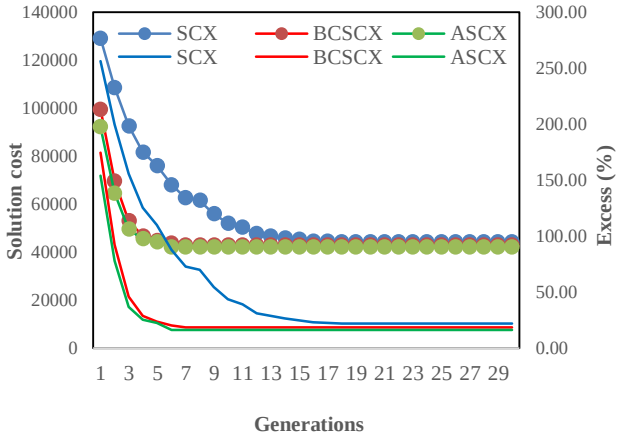

Fig. 1. Performance of Three Crossover Operators without Mutation for the Instance ftv170.

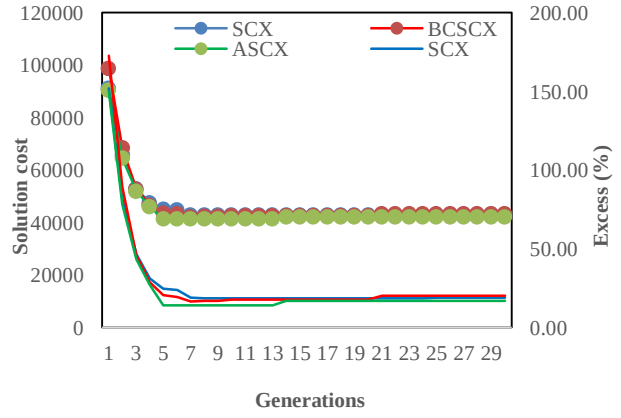

Fig. 2. Performance of Three Crossover Operators with Mutation for the Instance ftv170.

TABLE. II. SUMMARY OF THE RESULTS BY THE VARIANT GAS WITHOUT MUTATION FOR TSPLIB INSTANCES

Instance	Results	PMX	OX	AEX	CX	GX	SCX	BCSCX	ASCX
gr21	Best Sol	3393	2927	3887	5112	3821	2707	2707	2707
(2707)	Avg. Sol	4289.74	3806.40	4462.80	5767.94	4282.04	**2907.20**	2924.26	2916.04
	AvgExc(%)	58.47	40.61	64.86	113.07	58.18	7.40	8.03	7.72
	S.D.	447.10	485.88	339.14	284.30	219.95	112.17	100.03	67.24
	Avg. Time	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
fri26	Best Sol	1193	1014	1100	1799	993	941	937	937
(937)	Avg. Sol	1520.24	1364.70	1305.86	2060.34	1071.26	981.38	**957.70**	959.74
	AvgExc(%)	62.25	45.65	39.37	119.89	14.33	4.74	2.21	2.43
	S.D.	158.59	149.79	91.51	101.83	41.12	32.33	14.51	14.16
	Avg. Time	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ftv33	Best Sol	2236	1722	1843	3269	1604	1380	1420	1378
(1286)	Avg. Sol	2695.08	2352.22	2282.60	3539.30	1770.02	1489.20	1487.62	**1412.68**
	AvgExc(%)	109.57	82.91	77.50	175.22	37.64	15.80	15.68	9.85
	S.D.	231.60	277.64	187.21	122.80	114.42	37.26	36.08	44.42
	Avg. Time	0.00	0.01	0.00	0.00	0.00	0.00	0.01	0.01
ftv38	Best Sol	2730	2288	2244	3806	1860	1635	1619	1629
(1530)	Avg. Sol	3281.28	2853.34	2699.46	4267.20	2098.12	1772.80	1720.80	**1707.50**
	AvgExc(%)	114.46	86.49	76.44	178.90	37.13	15.87	12.47	11.60
	S.D.	267.82	324.38	221.03	164.85	119.59	47.61	36.42	32.79
	Avg. Time	0.00	0.01	0.00	0.00	0.00	0.00	0.01	0.01
dantzig42	Best Sol	1430	1159	937	2167	699	750	736	699
(699)	Avg. Sol	1786.76	1570.24	1153.74	2425.48	**711.72**	814.34	812.16	746.94
	AvgExc(%)	155.62	124.64	65.06	246.99	1.82	16.50	16.19	6.86
	S.D.	168.28	165.08	91.39	111.80	22.30	32.43	29.54	27.99
	Avg. Time	0.00	0.01	0.01	0.00	0.00	0.01	0.01	0.01
ft53	Best Sol	13539	11923	13058	20977	11504	8188	8082	7970
(6905)	Avg. Sol	17059.50	14595.36	14691.82	22342.38	12736.38	8626.44	8611.80	**8472.16**
	AvgExc(%)	147.06	111.37	112.77	223.57	84.45	24.93	24.72	22.70
	S.D.	1541.62	1746.14	856.81	620.64	602.92	239.55	228.45	272.27
	Avg. Time	0.00	0.02	0.01	0.00	0.01	0.03	0.02	0.27
kro124p	Best Sol	109400	84177	122824	148310	97683	41392	41396	40308
(36230)	Avg. Sol	130592.18	108875.06	143543.54	165422.36	107546.14	43789.36	42625.90	**42156.86**
	AvgExc(%)	260.45	200.51	296.20	356.59	196.84	20.86	17.65	16.36
	S.D.	9738.64	10308.85	9821.05	4375.44	4150.34	682.39	568.96	598.17
	Avg. Time	0.01	0.08	0.03	0.01	0.02	0.04	0.09	0.06
ftv170	Best Sol	17932	13271	9501	22545	5245	3696	3255	3258
(2755)	Avg. Sol	19522.98	16231.42	10765.90	23785.48	6158.00	3719.26	3611.54	**3473.52**
	AvgExc(%)	608.64	489.16	290.78	763.36	123.52	35.00	31.09	26.08
	S.D.	1048.53	1629.45	620.84	427.26	319.37	179.77	137.69	112.91
	Avg. Time	0.03	0.25	0.11	0.03	0.07	0.13	0.19	1.44
rbg323	Best Sol	4675	3616	5050	5645	2651	1731	1657	1620
(1326)	Avg. Sol	5014.40	4292.42	5259.02	5797.06	2985.10	1840.80	1747.90	**1689.16**
	AvgExc(%)	278.16	223.71	296.61	337.18	125.12	38.82	31.82	27.39
	S.D.	212.01	366.03	214.99	62.72	141.85	62.96	54.36	27.95
	Avg. Time	0.08	0.91	0.37	0.12	0.50	0.83	2.04	10.33
rbg358	Best Sol	5014	4081	5225	6307	2705	1678	1586	1393
(1163)	Avg. Sol	5562.44	4641.02	5600.40	6481.82	3010.70	1740.04	1713.26	**1453.60**
	AvgExc(%)	378.28	299.06	381.55	457.34	158.87	49.62	47.31	24.99
	S.D.	237.13	350.20	260.00	73.70	167.65	78.91	78.86	26.85
	Avg. Time	0.09	1.33	0.46	0.14	0.65	0.96	2.34	6.18
rbg403	Best Sol	5972	4931	6253	7031	4080	3483	3229	2928
(2465)	Avg. Sol	6346.12	5428.20	6360.18	7215.74	4310.88	3510.74	3403.54	**3012.70**
	AvgExc(%)	157.45	120.21	158.02	192.73	74.88	42.42	38.07	22.22
	S.D.	255.46	346.60	234.33	77.70	102.96	88.64	96.20	36.90
	Avg. Time	0.11	1.95	0.55	0.25	0.80	1.28	3.22	4.84
rbg443	Best Sol	6574	5538	6622	7615	4533	3731	3699	3333
(2720)	Avg. Sol	6933.96	6030.42	7076.30	7816.20	4730.06	3904.62	3881.90	**3404.44**
	AvgExc(%)	154.93	121.71	160.16	187.36	73.90	43.55	42.72	25.16
	S.D.	232.79	417.55	243.32	87.11	104.65	85.01	82.04	38.80
	Avg. Time	0.13	2.31	0.71	0.27	1.09	1.52	4.76	5.16

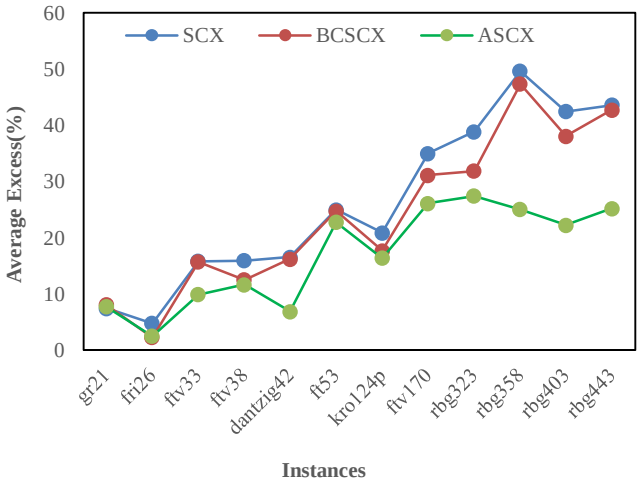

Fig. 3. Average Excess (%) by GA Variants without Mutation.

Among the blind crossovers, OX and AEX are competing. OX obtains lower average solutions for seven instances, namely, gr21, ft53, kro124p, rbg323, rbg358, rbg403 and rbg443, whereas AEX obtains lower average costs for five instances, namely, fri26, ftv33, ftv38, dantzig42 and ftv170. From this observation, one can tell that OX is better than AEX, and PMX and CX show very bad performances.

In order to decide if ASCX-based GA average (without mutation) is significantly different than the averages obtained by other GA variants, we performed Student's t-test. It is to be noted that we performed 50 runs for every problem instance considered here. We used the following t-test for the case of two big independent samples [35]:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{SD_1^2}{n_1 - 1} + \frac{SD_2^2}{n_2 - 1}}}$$

where,

$\bar{X}_1 - average\ of\ first\ sample,$

$SD_1 - standard\ deviation\ of\ first\ sample,$

$\bar{X}_2 - average\ of\ second\ sample,$

$SD_2 - standard\ deviation\ of\ second\ sample,$

$n_1 - first\ sample\ size,$

$n_2 - second\ sample\ size,$

The values of $\bar{X}_2$ and SD_2 are obtained by the ASCX-based GA, while of $\bar{X}_1$ and SD_1 values are obtained by other GA variants. The calculated values of the t statistic are reported in the Table III.

The t values can be positive or negative. The positive value indicates that the ASCX obtained better solution than the competitive GA variant. In the negative case, the competitive algorithm obtained better solution. We used confidence interval at the 95% confidence level ($t_{0.05}$ = 1.96). When t-value is greater than 1.96, the difference between the two values is significant. In this situation, the ASCX solution is better, when t has positive value. Negative t value means that the competitive GA variant has better solution. The case when t-value is less than 1.96, it corresponds to the situation that the difference between the observed values is not significant. The table also reports the information about the GA variants that obtained significantly better solutions.

In the case of three instances there is no statistically significant difference between ASCX and BCSCX. On nine instances ASCX is better than BCSCX. There is no significant difference between ASCX and SCX on one instance only. ASCX performed better than SCX on eleven instances. Next, ASCX performed better than GX on eleven instances, GX performed better than ASCX on one instance only. Finally, when comparing ASCX against blind crossovers, PMX, OX, AEX and CX, we found that ASCX performed better on all twelve instances, but it is not reported.

Table III also reports calculated values of the t statistic of blind crossovers against OX. In the case of one instance, ftv33, there is no statistically significant difference between OX and AEX. On seven instances OX is better than AEX. OX performed better than PMX and CX on all twelve instances.

TABLE. III. THE CALCULATED VALUES OF THE T STATISTIC (VARIANT GAS WITHOUT MUTATION) AND THE INFORMATION ABOUT VARIANT GAS THAT OBTAINED SIGNIFICANTLY BETTER SOLUTIONS

Instance	t-values against OX			t-values against ASCX		
	PMX	AEX	CX	GX	SCX	BCSCX
gr21	5.12	7.75	24.39	41.57	-0.47	0.47
Better	OX	OX	OX	ASCX	-----	----
fri26	4.99	-2.34	26.88	17.94	4.29	-0.70
Better	OX	AEX	OX	ASCX	ASCX	----
ftv33	6.63	-1.45	27.37	20.37	9.23	9.17
Better	OX	---	OX	ASCX	ASCX	ASCX
ftv38	7.12	-2.74	27.20	22.05	7.91	1.90
Better	OX	AEX	OX	ASCX	ASCX	----
dantzig42	6.43	-15.45	30.03	-6.89	11.01	11.22
Better	OX	AEX	OX	GX	ASCX	ASCX
ft53	7.41	0.35	29.26	45.12	2.98	2.75
Better	OX	OX	OX	ASCX	ASCX	ASCX
kro124p	10.72	17.04	35.35	109.16	12.59	3.98
Better	OX	OX	OX	ASCX	ASCX	ASCX
ftv170	11.89	-21.94	31.39	55.47	8.10	5.43
Better	OX	AEX	OX	ASCX	ASCX	ASCX
rbg323	11.95	15.94	28.36	62.75	15.41	6.73
Better	OX	OX	OX	ASCX	ASCX	ASCX
rbg358	15.25	15.40	36.01	64.20	24.06	21.82
Better	OX	OX	OX	ASCX	ASCX	ASCX
rbg403	14.92	15.60	35.23	83.09	36.31	26.55
Better	OX	OX	OX	ASCX	ASCX	ASCX
rbg443	13.23	15.15	29.31	83.14	37.47	36.83
Better	OX	OX	OX	ASCX	ASCX	ASCX

TABLE. IV. SUMMARY OF THE RESULTS BY THE VARIANT GAS WITH MUTATION FOR TSPLIB INSTANCES

Instance	Results	PMX	OX	AEX	CX	GX	SCX	BCSCX	ASCX
gr21	Best Sol	2707	2707	2835	2707	3614	2707	2707	2707
(2707)	Avg. Sol	3122.58	2827.12	3056.16	3201.74	3988.84	2885.36	2900.82	**2826.20**
	AvgExc(%)	15.35	4.44	12.90	18.28	47.35	6.59	7.16	4.40
	S.D.	219.5	90.36	156.58	272.06	169.99	104.16	86.26	61.29
	Avg. Time	0.02	0.04	0.06	0.06	0.05	0.03	0.05	0.04
fri26	Best Sol	953	1165	956	999	955	937	937	937
(937)	Avg. Sol	1103.02	1234.3	993.24	1150.16	1012.08	979.04	**953.44**	954.04
	AvgExc(%)	17.72	31.73	6.00	22.75	8.01	4.49	1.75	1.82
	S.D.	69.85	36.02	25.56	62.95	5.51	24.54	9.37	11.27
	Avg. Time	0.04	0.05	0.07	0.1	0.12	0.04	0.02	0.13
ftv33	Best Sol	1577	1671	1436	1660	1510	1371	1405	1371
(1286)	Avg. Sol	1759.54	2209.82	1604.42	1922.98	1679.9	1474.48	1478.94	**1386.72**
	AvgExc(%)	36.82	71.84	24.76	49.53	30.63	14.66	15.00	7.83
	S.D.	109.81	86.85	62.93	125.13	52.31	49.02	42.04	2.85
	Avg. Time	0.06	0.07	0.13	0.17	0.07	0.04	0.00	0.18
ftv38	Best Sol	1723	2273	1794	2217	1746	1630	1619	1599
(1530)	Avg. Sol	2077.04	2458.48	1971.24	2465.84	1911.48	1705.68	1712.56	**1648.64**
	AvgExc(%)	35.75	60.68	28.84	61.17	24.93	11.48	11.93	7.75
	S.D.	112.98	92.71	84.32	113.34	43.37	30.78	20.85	21.88
	Avg. Time	0.07	0.09	0.16	0.20	0.13	0.10	0.07	0.12
dantzig42	Best Sol	845	1069	827	1170	699	723	725	699
(699)	Avg. Sol	989.80	1108.88	915.18	1297.50	704.18	808.72	810.08	**699.72**
	AvgExc(%)	41.60	58.64	30.93	85.62	0.74	15.70	15.89	0.10
	S.D.	88.68	62.68	31.93	66.57	3.64	30.50	28.83	24.41
	Avg. Time	0.08	0.10	0.18	0.25	0.04	0.06	0.01	0.11
ft53	Best Sol	10027	10597	10299	12629	10109	7678	7848	7631
(6905)	Avg. Sol	11796.86	13902.48	12273.08	13854.44	11144.14	8494.82	8524.50	**8127.34**
	AvgExc(%)	70.85	101.34	77.74	100.64	61.39	23.02	23.45	17.70
	S.D.	701.64	421.58	288.69	588.81	455.60	246.93	183.91	156.51
	Avg. Time	0.09	0.13	0.27	0.40	0.25	0.26	0.31	0.38
kro124p	Best Sol	106539	79811	109251	110833	81824	41331	41668	40246
(36230)	Avg. Sol	117138.20	100806.48	116768.26	120254.10	89253.80	43674.54	42544.46	**41471.58**
	AvgExc(%)	223.32	178.24	222.30	231.92	146.35	20.55	17.43	14.47
	S.D.	3153.95	2264.52	2428.83	2740.37	2557.47	638.43	566.01	432.72
	Avg. Time	0.18	0.43	0.66	1.22	0.57	1.25	0.13	2.22
ftv170	Best Sol	17088	13158	9482	18962	4667	3285	3257	3232
(2755)	Avg. Sol	18689.18	15389.62	10588.86	19630.42	4817.32	3523.74	3608.40	**3393.00**
	AvgExc(%)	578.37	458.61	284.35	612.54	74.86	27.90	30.98	23.16
	S.D.	305.95	282.54	231.40	219.10	71.52	113.55	92.49	95.42
	Avg. Time	0.27	1.15	1.83	3.41	4.17	3.77	3.23	0.93
rbg323	Best Sol	4583	3558	4809	5024	2102	1658	1660	1611
(1326)	Avg. Sol	5006.74	4263.20	5075.84	5150.86	2192.08	1718.76	1725.52	**1618.80**
	AvgExc(%)	277.58	221.51	282.79	288.45	65.32	29.62	30.13	22.08
	S.D.	50.16	42.92	34.15	39.96	31.70	22.19	20.63	17.70
	Avg. Time	0.90	3.05	5.59	10.75	15.51	12.37	24.79	23.53
rbg358	Best Sol	4988	3951	5034	5624	2054	1524	1582	1327
(1163)	Avg. Sol	5428.92	4583.32	4650.54	5740.92	2203.12	1699.20	1711.30	**1387.92**
	AvgExc(%)	366.80	294.09	299.87	393.63	89.43	46.10	47.15	19.34
	S.D.	53.53	41.32	42.26	45.63	52.02	29.22	23.23	24.05
	Avg. Time	1.01	3.56	6.91	12.16	17.28	16.71	30.14	30.77
rbg403	Best Sol	5809	4848	6079	6375	3760	3314	3229	2922
(2465)	Avg. Sol	6219.88	5363.96	6273.98	6543.88	3828.34	3401.18	3479.62	**2983.38**
	AvgExc(%)	152.33	117.60	154.52	165.47	55.31	37.98	41.16	21.03
	S.D.	49.76	37.89	34.63	46.26	39.16	26.48	24.06	21.43
	Avg. Time	1.14	4.48	9.25	17.81	21.43	19.98	33.89	38.93
rbg443	Best Sol	6401	5494	6411	7053	3705	3705	3710	3252
(2720)	Avg. Sol	6893.26	5935.72	6895.30	7123.44	3742.88	3882.52	3872.66	**3321.58**
	AvgExc(%)	153.43	118.23	153.50	161.89	37.61	42.74	42.38	22.12
	S.D.	43.17	44.32	36.77	34.14	22.11	26.78	25.53	20.95
	Avg. Time	1.37	6.35	10.41	18.65	37.87	26.82	42.64	50.82

Table IV reports results by the eight GA variants where mutation is applied. With respect to the average cost, it is once again very clear from the Table IV that distance-based crossovers are superior to blind crossovers. Among the distance-based crossovers, GX is the worst, however, it obtains best cost for danzig42 only. Though SCX could not obtain lowest average cost, but it obtains best costs for the instances gr21 and fr26 at least once in 50 runs. The crossovers SCX and BCSCX are competing. SCX obtains lower average cost for nine instances, whereas BCSCX obtains lower average cost for three instances only. However, BCSCX obtains lowest average solution for the instance fri26 and ASCX obtains lowest average solutions with lower S.D. for eleven instances, namely, gr21, ftv33, ftv38, dantzig42, ft53, kro124p, ftv170, rbg323, rbg358, rbg403 and rbg443. So, among all the crossovers ASCX is found to be the best. Based on best costs also ASCX is found to be the best. The results are depicted in Fig. 4, which also shows the effectiveness of our proposed crossover operator ASCX. So, whether mutation is used or not, the best performance is accomplished by ASCX. However, based on convergence time blind crossovers found to be better than distance-based crossovers, and PMX is the best one.

Among the blind crossovers, CX show very bad performances that obtains lower average solution for no any instance; PMX obtains lower average costs for the instance ft53 only; OX obtains lower average solutions for six instances, namely, gr21, kro124p, rbg323, rbg358, rbg403 and rbg443; whereas AEX obtains lower average solutions for five instances, namely, fri26, ftv33, ftv38, dantzig42 and ftv170. From this observation one can say that OX is the best and CX is the worst. However, CX obtains best solution at least once in 50 runs for gr21.

Here also, in order to decide if ASCX based GA (with mutation) average is significantly different than the averages obtained by other GA variants, we perform Student's t-test and the calculated values of the t statistic are reported in the Table V.

In the case of one instance there is no statistically significant difference between ASCX and BCSCX, and ASCX and GX. On eleven instances ASCX is better than BCSCX and GX. ASCX performed better than SCX on all twelve instances. While comparing SCX against BCSCX (of course, not reported in any table here), we found that on most of the instances there is no statistically significant difference between them, we can treat them statistically equivalent. Finally, when comparing ASCX against blind crossovers, PMX, OX, AEX and CX, we found that ASCX performed better on all twelve instances, except for one instance, gr21, there is no statistically significant difference between ASCX and OX, but it is not reported.

Table V also reports calculated values of the t statistic of blind crossovers against OX. On two instances there is no statistically significant difference between OX and CX. On two instances CX is better than OX, whereas on eight instances OX is better than CX. On five instances PMX is better than OX, whereas on seven instances OX is better than PMX. On six instances OX and AEX are better than each other. So, among the blind crossovers OX is the best and CX is the worst.

Based on the above study it very clear that the proposed crossover operator ASCX is the best, BCSCX and SCX are equivalent and the second-best, and CX is the worst. Among blind crossovers OX is the best. About the performance of blind crossovers same observation is made in [36]. However, in terms of convergent time, PMX is found to be the best.

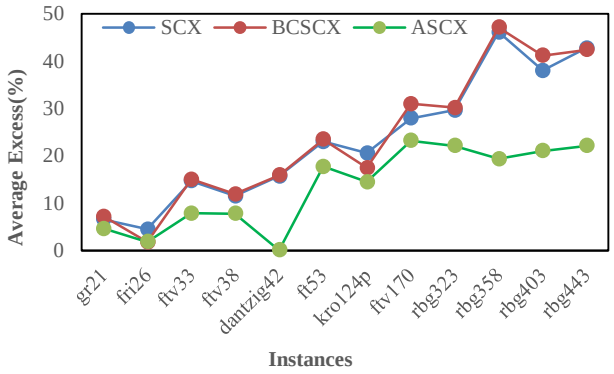

Fig. 4. Average Excess (%) by GA Variants with Mutation.

TABLE. V. THE CALCULATED VALUES OF THE T STATISTIC (CROSSOVERS WITH MUTATION) AND THE INFORMATION ABOUT VARIANT GAS THAT OBTAINED SIGNIFICANTLY BETTER SOLUTIONS

Instance	t-values against OX			t-values against ASCX		
	PMX	**AEX**	**CX**	**GX**	**SCX**	**BCSCX**
gr21	8.71	8.87	9.15	45.04	3.43	4.94
Better	OX	OX	OX	ASCX	ASCX	ASCX
fri26	-11.69	-38.21	-8.12	32.39	6.48	-0.29
Better	PMX	AEX	CX	ASCX	ASCX	----
ftv33	-22.51	-39.51	-13.18	39.17	12.51	15.32
Better	PMX	AEX	CX	ASCX	ASCX	ASCX
ftv38	-18.27	-27.22	0.35	37.88	10.57	14.80
Better	PMX	AEX	----	ASCX	ASCX	ASCX
dantzig42	-7.68	-19.28	14.44	1.26	19.53	20.45
Better	PMX	AEX	OX	----	ASCX	ASCX
ft53	-18.01	-22.32	-0.46	43.84	8.80	11.51
Better	PMX	AEX	----	ASCX	ASCX	ASCX
kro124p	29.44	33.65	38.29	128.95	19.99	10.54
Better	OX	OX	OX	ASCX	ASCX	ASCX
ftv170	55.46	-92.02	83.03	83.61	6.17	11.35
Better	OX	AEX	OX	ASCX	ASCX	ASCX
rbg323	78.84	103.71	105.96	110.53	24.65	27.48
Better	OX	OX	OX	ASCX	ASCX	ASCX
rbg358	87.53	7.96	131.63	99.57	57.58	67.70
Better	OX	OX	OX	ASCX	ASCX	ASCX
rbg403	95.80	124.10	138.13	132.50	85.85	107.81
Better	OX	OX	OX	ASCX	ASCX	ASCX
rbg443	108.34	116.64	148.61	96.82	115.48	116.81
Better	OX	OX	OX	ASCX	ASCX	ASCX

TABLE. VI. COMPARATIVE STUDY AMONG CX2 [37], HX AND OUR PROPOSED ASCX

Instance	Results	ASCX	CX2	HX
gr21	Best Sol	2707	2995	2707
	Worst Sol	2903	3576	2940
	Avg. Sol	**2790**	3245	2791
fri26	Best Sol	937	1099	937
	Worst Sol	970	1278	1014
	Avg. Sol	**953**	1128	967
ftv33	Best Sol	1341	1811	1397
	Worst Sol	1480	2322	1941
	Avg. Sol	**1393**	2083	1638
ftv38	Best Sol	1598	2252	1755
	Worst Sol	1703	2718	2389
	Avg. Sol	**1653**	2560	2068
dantzig42	Best Sol	699	699	699
	Worst Sol	795	920	985
	Avg. Sol	**703**	802	833
	Best Sol	7803	10987	11234
ft53	Worst Sol	8692	13055	13288
	Avg. Sol	**8192**	12243	12534
kro124p	Best Sol	40607	92450	103988
	Worst Sol	41543	121513	121239
	Avg. Sol	**41473**	101229	110447
ftv170	Best Sol	3244	6421	11215
	Worst Sol	3422	8416	13221
	Avg. Sol	**3384**	7019	10878
rbg323	Best Sol	1501	4212	5175
	Worst Sol	1589	5342	5341
	Avg. Sol	**1557**	4654	5072
rbg358	Best Sol	1339	5404	5560
	Worst Sol	1408	6004	5889
	Avg. Sol	**1394**	5622	5629
	Best Sol	2867	6257	6259
rbg403	Worst Sol	3023	6671	6885
	Avg. Sol	**2965**	6455	6632
	Best Sol	3268	6854	7218
rbg443	Worst Sol	3432	7388	7523
	Avg. Sol	**3356**	6981	7318

Further, we considered results reported in [37] for comparing with our proposed crossover ASCX. Recently Weise et al. [38] made a comparative study among eleven crossover operators for the TSP and found that heuristic crossover (HX) [39] is the best performing operator. The HX applies a greedy heuristic to create an offspring from two parents. So, we implemented the GA using HX and run on the above twelve problem instances. It is to be noted that the same common parameters' values selected for GAs in [37] are used for ASCX and HX. The parameters are as follows: population size, maximum generation, crossover, and mutation probabilities are 150, 500, 0.80, and 0.10, respectively, for less than 100 size instances, whereas population size and maximum generation are 200 and 1000, respectively for more than 100 size instances. Also, the experiments are performed 30 times (30 runs) for each instance. Table VI reports best, worst and average solution costs in 30 runs by ASCX, CX2 and HX.

The crossovers ASCX and HX hit best known solutions for the instances gr21 and fri26, whereas ASCX, CX2 and HX hit best known solution for the instance dantzig42 at least once in 30 runs. In terms of best solution cost, except for the instance dantzig42, ASCX is found better than CX2, and except for gr21, fri26 and dantzig42, ASCX is better than HX. However, in terms of worst and average solution costs ASCX is found to be the best among three crossover operators for all instances. From this study it is very clear that our proposed crossover ASCX is far better than CX2 and HX.

V. CONCLUSION AND FUTURE WORKS

Several crossover operators have been proposed and reported for the TSP by using GAs. We have proposed a new crossover operator, named ASCX, for the TSP. This proposed operator upgrades the SCX and improved the quality of offspring. It is easy to execute and always generates a valid offspring. We focused on some blind crossover operators, namely, PMX, OX, AEX and CX, and distance-based crossover operators, namely, GX, SCX and BCSCX along with ASCX. Firstly, we applied these operators on a pair of chromosomes in manual experiment and found that ASCX performed very good. Then for a significant performance, twelve benchmark instances from the TSPLIB (traveling salesman problem library) have been considered. We developed sixteen variant GAs using crossovers with/without mutation and carried out comparative study of the GAs on the instances. In terms of solution quality, our comparative study showed that distance-based crossovers are far superior than the blind crossovers, and our proposed crossover operator ASCX is the best, BCSCX and SCX are the second-best, and CX is the worst. However, among blind crossovers OX is found to be the best. This observation is verified by Student's t-test at 95% confidence level. Further, we carried out a comparative study among CX2, HX and ASCX, and found that our proposed crossover ASCX is the best. Thus, our proposed operator may be good operator to find more better and accurate results, researchers may use it for other related combinatorial optimization problems.

In this present study, we considered the original version of some crossover operators. Our objective was only to compare the quality of the solutions obtained by the crossover operators, neither to improve the solution quality by any of the operators nor to design the most competitive algorithm for the TSP. So, we neither used any local search technique to enhance the solution quality nor developed parallel version of algorithms to find exact solution. Consequently, we have limited ourselves to simple and pure GA process. Also, we set highest crossover probability to display exact nature of crossover operators. Mutation might be used with lowest probability just not to get stuck in local minima quickly. However, one can incorporate good local search procedure to the hybridize the algorithm, and thus, to solve problem instances exactly, which is under our investigation. Finally, the advantage and helpfulness of the ASCX can be tested on other combinatorial optimization problems.

ACKNOWLEDGMENT

The author thanks the honourable anonymous reviewers for their constructive comments and suggestions which helped the author to improve this paper.

REFERENCES

[1] 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.

[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] J.D.E. Little, K.G. Murthy, D.W. Sweeny, and C.Karel, "An algorithm for the travelling salesman problem," Operations Research, vol. 11, pp. 972-989, 1963.

[4] M. Padberg, and G. Rinaldi, "Optimization of a 532-node symmetric traveling salesman problem by branch and cut," Operations Research Letter, vol. 6, no. 1, pp. 1–7, 1987.

[5] S.N.N. Pandit, and K. Srinivas, "A lexisearch algorithm for the traveling salesman problem," Proc. IEEE Int. Joint Conf. Neural Networks, vol. 3, pp. 2521–2527, 1991.

[6] Y. Deng, Y. Liu, and D. Zhou, "An improved genetic algorithm with initial population strategy for symmetric TSP," Mathematical Problems in Engineering, vol. 2015, Article ID212794, 6 pages, 2015.

[7] M. Mahi, Ö.K. Baykan, and H. Kodaz, "A new hybrid method based on particle swarm optimization, ant colony optimization and 3-opt algorithms for traveling salesman problem," Applied Soft Computing, vol. 30, pp. 484–490, 2015.

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

[9] S.-M. Chen, and C.-Y. Chien, "Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques," Expert Systems with Applications, vol. 38, no. 12, pp. 14439–14450,) 2011.

[10] X. Zhou, D. Gao, C. Yang, and H. Gui, "Discrete state transition algorithm for unconstrained integer optimization problems," Neurocomputing, vol. 173, pp. 864–874, 2016.

[11] J. Knox, "Tabu search performance on the symmetric traveling salesman problem," Computer and Operations Research, vol. 21, no. 8, pp. 867–876, 1994.

[12] J.-Y. Potvin, "State-of-the-art survey—the traveling salesman problem: a neural network perspective," ORSA Journal of Computing, vol. 5, no. 4, pp. 328–348, 1993.

[13] M.S. Kıran, H. Iscan, and M. Gündüz, "The analysis of discrete artificial bee colony algorithm with neighborhood operator on traveling salesman problem," Neural Computing and Applications, vol. 23, no. 1, pp. 9–21, 2013.

[14] A. Hatamlou, "Solving travelling salesman problem using black hole algorithm," Soft Computing, vol. 22, no. 24, pp. 8167–8175, 2018.

[15] Z.H. Ahmed, "Algorithms for the quadratic assignment problem," LAP LAMBERT Academic Publishing, Mauritius, 2019, 104 pages.

[16] 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.

[17] K. Deb, "Optimization for engineering design: algorithms and examples," Prentice Hall of India Pvt. Ltd., New Delhi, India, 1995.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] Z.H. Ahmed, "A simple genetic algorithm using sequential constructive crossover for the quadratic assignment problem," Journal of Scientific and Industrial Research, vol. 73, pp. 763-766, 2014.

[29] Z.H. Ahmed, "Experimental analysis of crossover and mutation operators for the quadratic assignment problem," Annals of Operations Research, vol. 247, pp. 833-851, 2016.

[30] 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.

[31] 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.

[32] M.A. Al-Omeer, and Z.H. Ahmed, "Comparative study of crossover operators for the MTSP," 2019 International Conference on Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, pp. 1-6, 3-4 April 2019.

[33] 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.

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

[35] M. Nikolić, and D. Teodorović, "Empirical study of the bee colony optimization (BCO) algorithm," Expert Systems with Applications, vol. 40, pp. 4609–4620, 2013.

[36] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: a review of representations and operators," Artificial Intelligence Review, vol. 13, pp. 129–170, 1999.

[37] A. Hussain, Y. S. Muhammad, M. N. Sajid, I. Hussain, A. M. Shoukry, and S. Gani, "Genetic algorithm for traveling salesman problem with modified cycle crossover operator," Computational Intelligence and Neuroscience, vol. 2017, Article ID 7430125, 7 pages, 2017.

[38] T. Weise, Y. Jiang, Q. Qi, and W. Liu, "A branch-and-bound-based crossover operator for the traveling salesman problem," IJCINI International Journal of Cognitive Informatics and Natural Intelligence, vol. 13, no. 3, pp. 1-18, 2019.

[39] 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.