Solving the Traveling Salesman Problem using Greedy Sequential Constructive Crossover in a Genetic Algorithm

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 efficiency of genetic algorithms (GAs) depends mainly on crossover operator. The choice of good crossover operators leads to effective GA. There are many existing crossover operators in the literature. In this paper, we propose a modified version of sequential constructive crossover (SCX), named greedy SCX (GSCX), for solving the benchmark travelling salesman problem. We then compare the efficiency of the proposed crossover operator with greedy crossover, SCX and bidirectional circular SCX operators for solving the TSP on some benchmark TSPLIB instances. The comparative study shows that our proposed GSCX operator is the best among these crossover operators for the problem.

Key words:
Genetic algorithm; Greedy sequential constructive crossover; Traveling salesman problem; NP-hard.

1. Introduction

We consider the benchmark travelling salesman problem (TSP) for our study that finds a least cost Hamiltonian cycle in a network. The problem can be formally stated as follows: A network with 'n' nodes, with 'node 1' as 'depot' and a travel cost (or distance, or travel time etc.,) matrix C= [cij] of order n associated with ordered pairs (i, j) of nodes is given. The problem is to find a least cost Hamiltonian cycle. There are two types of TSP - symmetric and asymmetric. The TSP is symmetric if cij = cji, for all i, j, else, asymmetric. If there are n nodes in a network, then for asymmetric TSP, there will be $(n-1)!$ possible solutions out of which at least one of them provides the minimum cost, and for asymmetric TSP, there will be $\frac{(n-1)!}{2}$ possible solutions along with same valued opposite cyclic permutations. In both types, the number of possible solutions is very large; so, a complete search is very hard, if it is not impossible. The problem is proved to be NP-Hard [1] and it has many real-life applications [2].

Several exact and heuristic/metaheuristic algorithms have been developed and reported in the literature for solving the TSP. Though exact algorithms obtain exact optimal solution to the problem, but the computational time increases exponentially as the problem size increases. On the other hand, heuristic algorithms do not guarantee the optimality of the solution, but they obtain near optimal solution in a very short time. So, the researchers give more importance to find good quality heuristic solutions in reasonable time, rather to find exact solution after lot of computational time. Some of most effective metaheuristic algorithms are ant colony optimization, genetic algorithm, simulated annealing, state transition algorithm, tabu search, artificial neural network, artificial bee colony, black hole, and particle swarm optimization. Out of these algorithms genetic algorithm is very popular and one of the best metaheuristic algorithms for solving the TSP. In simple GA, selection, crossover and mutation are three operators, and among them crossover is very important operator. So, several crossover operators have been developed for solving the TSP as well as other combinatorial optimization problems ([3], [4]).

In this paper, we propose a modified version of sequential constructive crossover (SCX), named greedy SCX (GSCX) operator that produces better offspring than SCX. We then compare the efficiency of the proposed crossover operator with greedy crossover, SCX and bidirectional circular SCX operators for solving the TSP on some benchmark problem instances.

This paper is organized as follows: The literature review is given in Section 2. Next Section 3 discusses existing and proposed crossover operators for developing variant genetic algorithms for the TSP, while Section 4 describes computational experiments for four crossover operators. Finally, Section 5 presents comments and concluding remarks.

2. Literature Review

Since the crossover operator plays a vital role in GA, so many crossover operators have been proposed for the TSP. Goldberg and Lingle [5] 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. As reported, the authors found near-optimal solution to a well-known 33-node problem instance.

Manuscript received February 5, 2020
Manuscript revised February 20, 2020

The ordered crossover (OX) is developed in [6] which builds offspring by choosing a subsequence of a tour from one parent and preserving the relative order of nodes from the other parent. However, it was applied to the job-shop scheduling problem. Syswerda [7] proposed order based crossover (OBX) that selects several positions randomly in one parent tour and the order of the nodes in the selected positions of this parent is imposed on the other parent. Position based operator (PBX) is also proposed in [7] that selects a set of positions randomly in the parent tours. But it imposes the position of the selected nodes on the corresponding nodes of the other parent.

Grefenstette et al. [8] proposed alternating edges crossover (AEX) operator that assumes a chromosome as a directed cycle of arcs. The offspring is built by selecting alternative arcs from both parents, with some additional random selections in case of infeasibility. The authors also proposed the greedy crossover (GX) operator for the TSP.

Oliver et al. [9] developed cycle crossover (CX) that builds an offspring where every node and its corresponding position originated from one of the parents.

The edge recombination crossover (ERX) is proposed by Whitley et al. [10] which follows an edge map to builds an offspring by inheriting as many edges as possible from the parents, and the common edges of both parents have high priority. The generalized N-point crossover (GNX) [11] is developed based on the traditional N-point crossover operator.

A crossover operator named sequential constructive crossover (SCX) is developed in [12] for solving the TSP and compared with ERX and GNX on symmetric and asymmetric TSPLIB instances. As reported, SCX is better than ERX and GNX. The SCX is then modified in [13] by considering that if no legitimate node is present in any of the parents after current node, search continues from the beginning of the parent chromosome and select the first legitimate node as the next node.

Bidirectional circular sequential constructive crossover (BCSCX) is developed in [14] by modifying SCX, which 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.

3. Genetic Algorithms for the TSP

Genetic algorithm (GA) is a search process, proposed by John Holland in 1970s, inspired by natural biological evolution process. It starts initially with a population of strings, called chromosomes, that encode solutions to a problem, and operates possibly three operators, namely selection, crossover and mutation, to produce new and possibly better populations in successive generations. Crossover along with selection operator is the main leading procedure in genetic algorithms. Mutation expands search space and defends from loss of any genetic substance due to selection and crossover operators. Although GA is among the best algorithms, but its performance depends on initial population along with three operators and four parameters, which are discussed here.

3.1. Chromosome Representation and Selection Operator

There are many methods for representing solutions as chromosomes for the TSP. We consider path representation which makes a list of node labels so that no node is repeated 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→8→6→3→7 → 4→2→9→ 5 →1} may be represented as (1, 8, 6, 3, 7, 4, 2, 9, 5). The objective function is the sum of the costs of all edges in the tour.

A fitness function can be defined using objective function, $F(x) = \frac{1}{f(x)}$ and we define the fitness function as , where f(x) is the objective function. Usually GA starts with a set of chromosomes called an initial population of chromosomes. We have considered randomly generated initial population and then apply selection operator on the population. In selection operator, old chromosomes are copied into mating pool, usually with a probability related to their fitness value. In this operator no new chromosome is created. We have considered the stochastic remainder selection process [15] for our GAs.

3.2. Existing Crossover Operators

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 is presented in [16], and found that SCX outperformed other operators in achieving good quality solution for the TSP. The SCX is also effectively applied to other combinatorial optimization problems ([17]-[24]). So, we are going to consider and discuss GX, SCX and BCSCX operators.

3.2.1. Greedy Crossover Operator

The greedy crossover (GX) is proposed in [8] 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 1. Let P1: (1, 2, 3, 4, 6, 9, 5, 7, 8) and P2: (1, 3, 5, 7, 8, 9, 4, 2, 6) be a pair of selected parent chromosomes with costs 83 and 75 respectively. We consider these same chromosomes for illustrating all other crossover operators. We fix headquarters (first gene) as 'node 1', and so, we start the procedure from the 'node 1'.

Table 1: 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 node 1 with their costs 7 and 15 respectively. The node 2 is cheaper, so, it is added to the incomplete offspring that becomes: (1, 2).
Next, the nodes 3, 1, 6 and 4 are neighbours of node 2 with costs 8, 11, 3 and 7 respectively. The node 6 is the cheapest, so, it is added to the incomplete offspring that becomes: (1, 2, 6).
Next, the nodes 9, 4, 1 and 2 are neighbours of node 6 with costs 5, 11, 6 and 13 respectively. The node 9 is the cheapest so, it is added to the incomplete offspring that becomes: (1, 2, 6, 9).
Next, the nodes 5, 6, 4 and 8 are neighbours of node 9 with costs 10, 7, 9 and 10 respectively. The node 6 is the cheapest, but it exists in the offspring, so, node 3 is selected randomly and added to the incomplete offspring that becomes: (1, 2, 6, 9, 3).
Next, the nodes 4, 2, 5 and 1 are neighbours of node 3 with costs 16, 5, 12 and 15 respectively. The node 2 is the cheapest, but it exists in the offspring, so, node 4 is selected randomly and added to the incomplete offspring that becomes: (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.

3.2.2. Sequential Constructive Crossover Operator

The sequential constructive crossover (SCX) operator is proposed in [12] and then modified in [13] that builds an offspring using better arcs based on their cost present in the parents' structure. Moreover, it uses the better arcs that are not present in both parents' structure. It 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 starting of the chromosome and then compares their associated cost to decide the next node of the child chromosome. The algorithm for the SCX [13] is given below:

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 cpα < cpβ, 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 P1 and P2 are 2 and 3 respectively with c12=7 and c13=15. Since c12<c13, we accept node 2, and the partially constructed chromosome becomes (1, 2).
The legitimate nodes after node in P1 and P2 are 3 and 6 respectively with c23=8 and c26=3. Since c26<c23, we accept node 6, and the partially constructed chromosome becomes (1, 2, 6).
The legitimate node after node 6 in P1 is 9 with c69=5, but none in P2. So, for P2, we sequentially search from the beginning of the chromosome and find the first legitimate node 3 with c63=8. Since c69<c63, we accept node 9, and the partially constructed chromosome becomes (1, 2, 6, 9).
The legitimate nodes after node 9 in P1 and P2 are 5 and 4 respectively with c95=10 and c94=9. Since c94<c95, we accept node 4, and the partially constructed chromosome becomes (1, 2, 6, 9, 4).
The legitimate node after node 4 in P1 is 5 with c45=9, but none in P2. So, for P2, we sequentially search from the beginning of the chromosome and find the first legitimate node 3 with c43=11. Since c45<c43, we accept node 5, and the partially constructed chromosome becomes (1, 2, 6, 9, 4, 5).
Continuing this way, we obtain the complete offspring chromosome as: (1, 2, 6, 9, 4, 5, 7, 8, 3) with cost 72.

3.2.3. Bidirectional Circular Sequential Constructive Crossover Operator

The bidirectional circular sequential constructive crossover (BCSCX) operator is proposed in [14] by modifying SCX that searches for next neighbors in both left and right directions in both parents. So, four neighbor genes are considered. Also, during searching for the next neighbor gene, if it reaches to the end or to the start of any parent, it will wrap around.

We illustrate the operator through the same example mentioned above. Select 'node 1' as the 1st gene and search for the next node. The legitimate nodes after node 1 in both directions in P1 are 2 and 8 (after wrapping around) with costs 7 and 9 respectively, and in P2 are 3 and 6 (after wrapping around) with costs 15 and 6 respectively. We accept node 6 as it is the cheapest among four nodes, and so, the partially constructed chromosome becomes (1, 6).

The legitimate nodes after node 6 in both directions in P1 are 9 and 4 with costs 5 and 11 respectively, and in P2 are 3 (after wrapping around) and 2 with costs 8 and 13 respectively. We accept node 9 as it is the cheapest, and so, the partially constructed chromosome becomes (1, 6, 9).

The legitimate nodes after node 9 in both directions in P1 are 5 and 4 with costs 10 and 9 respectively, and in P2 are 4 and 8 with costs 9 and 10 respectively. We accept node 4 as it is the cheapest, and so, the partially constructed chromosome becomes (1, 6, 9, 4).

The legitimate nodes after node 4 in both directions in P1 are 5 and 3 with costs 9 and 11 respectively, and in P2 are 2 and 8 with costs 5 and 4 respectively. We accept node 8 as it is the cheapest, and so, the partially constructed chromosome becomes (1, 6, 9, 4, 8).

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

3.3. Proposed Crossover Operator: Greedy Sequential Constructive Crossover

We are going to modify SCX operator and name it greedy SCX that works like as SCX. In the step 2 of SCX, if no legitimate node is found in any parent, then search continued sequentially from the starting of the parent and consider the first legitimate node. Though as reported, the method is better, however, there is room for modifying this rule. In this present study, we introduce greedy method in this step. So, we define the algorithm for GSCX 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 cpα < cpβ, 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 operator through the same example given above. Select 'node 1' as the 1st gene. The legitimate nodes after node 1 in P1 and P2 are 2 and 3 respectively with c12=7 and c13=15. Since c12<c13, we accept node 2, and so, the partially constructed chromosome becomes (1, 2).

The legitimate nodes after node 2 in P1 and P2 are 3 and 6 respectively with c23=8 and c26=3. Since c26<c23, we accept node 6, and so, the partially constructed chromosome becomes (1, 2, 6).

The legitimate node after node 6 in P1 is 9, but none in P2. So, we search and find the cheapest legitimate node as 8, and we accept node 8, and so, the partially constructed chromosome becomes (1, 2, 6, 8).

There is no legitimate node after node 8 in P1, so, we search and find the cheapest legitimate node as 5, and we accept node 5. So, the partially constructed chromosome becomes (1, 2, 6, 8, 5).

The legitimate nodes after node 5 in both P1 and P2 are same node 7. So, we accept node 7, and the partially constructed chromosome becomes (1, 2, 6, 8, 5, 7).

There is no legitimate node after node 7 in P1, so, we search and find the cheapest legitimate node as 3, and we accept node 3. So, the partially constructed chromosome becomes (1, 2, 6, 8, 5, 7, 3).

Continuing this way, we obtain the complete offspring chromosome as: (1, 2, 6, 8, 5, 7, 3, 9, 4) with cost 42.

Through the above manual experiment, we found that our proposed crossover operator GSCX is the best among the four crossover operators considered above.

3.4. Mutation Operator

In GAs, the mutation operator is applied after crossover operator. It usually selects randomly a gene (position) in the chromosome and changes the corresponding allele (value of the gene), and thus modifies the information. Since the less fit members of successive generations are discarded in previous operators and thereby some good characteristics of genetic materials might be lost forever, hence the mutation operator is required to recover them. By doing infrequent

random changes in chromosomes, GAs confirm that new portions of the search area are visited, which reproduction and crossover couldn't ensure fully. So, mutation guarantees that no significant elements are early lost, thus it maintains variety in mating pool. Generally, one can expect that mutation can help the crossover operators to come out from local optima and find better quality solution. The traditional mutation operator does not work for the TSP. For our GAs, we have used the reciprocal exchange mutation that randomly selects two nodes and exchanges them.

3.5. Genetic Parameters

The GA search procedure is governed mainly by four parameters. The first one is population size that determines number of chromosomes in a population during the search. If the number is very less, the search has no possibility to effectively cover the whole search space. On the other hand, if it is large, the search process wastes computational time. The second one is crossover probability that specifies the probability of doing crossover on parent chromosomes. The third one is mutation probability that sets the probability of performing bit-wise mutation. The last one is the stopping condition that states the condition for stopping the search process.

3.6. Variants of Our Genetic Algorithm

We propose to execute, and test different variants of a simple GA as shown below. All variants follow the same GA structure except selection of a crossover operator from the following four crossover operators: GX, SCX, BCSCX and GSCX.

```
VariantGA ( )
{   Initialize random population of size Ps;
    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 Pc;
            Do bit-wise mutation with mutation
    probability Pm;
            Evaluate the population;
    }
}
```

Within one selection, a single crossover operator is implemented. Through this we are going to measure efficiency of the crossover operators, and to find their relative ranking. Each variant GA is non-hybrid, simple, which uses basic GA processes and operators, but does not incorporate any other heuristic algorithm.

4. Computational Experiments

Variant GAs using different crossovers have been encoded in Visual C++. In order to compare the efficiency of the different crossover operators, variant GAs are applied on twenty eight benchmark TSPLIB instances [25] and run on a Laptop with i3-3217U CPU@1.80 GHz and 4 GB RAM under MS Windows 7. Among the twenty eight problem instances, the instances ftv33, ftv35, ftv38, p43, ftv44, ftv47, ry48p, ft53, ftv55, ftv64, ft70, ftv70, kro124p, ftv170, rbg323, rbg358, rbg403 and rbg443 are asymmetric, and gr21, fri26, bayg29, dantzig42, eil51, berlin52, pr76, lin105, d198 and a280 are symmetric TSPs. For all variant GAs, the parameters are set as follows: population size is 50, crossover probability is 1.0 (i.e., 100%), mutation probability is 0.20 (i.e., 20%), and maximum of 1,000 generations as the stopping condition. For each instance, the experiments have been repeated 50 times. Figures 1 present results for the instance ftv64 (considering only 50 generations) by all GA variants. Each graph corresponds to a crossover operator, and it shows how the current solution improves depending on the number of generations. In the figure, the labels on the left margin denote the solution cost, while the labels on the right margin denote the percentage of excess to the best known solution (Excess (%)). Figure 1 shows that SCX has some variations, but it is not the best. Though BCSCX and GSCX have less variations and are competing each other, still GSCX provides us best results. But GSCX has limited variation range and gets stuck in local minimums very quickly.

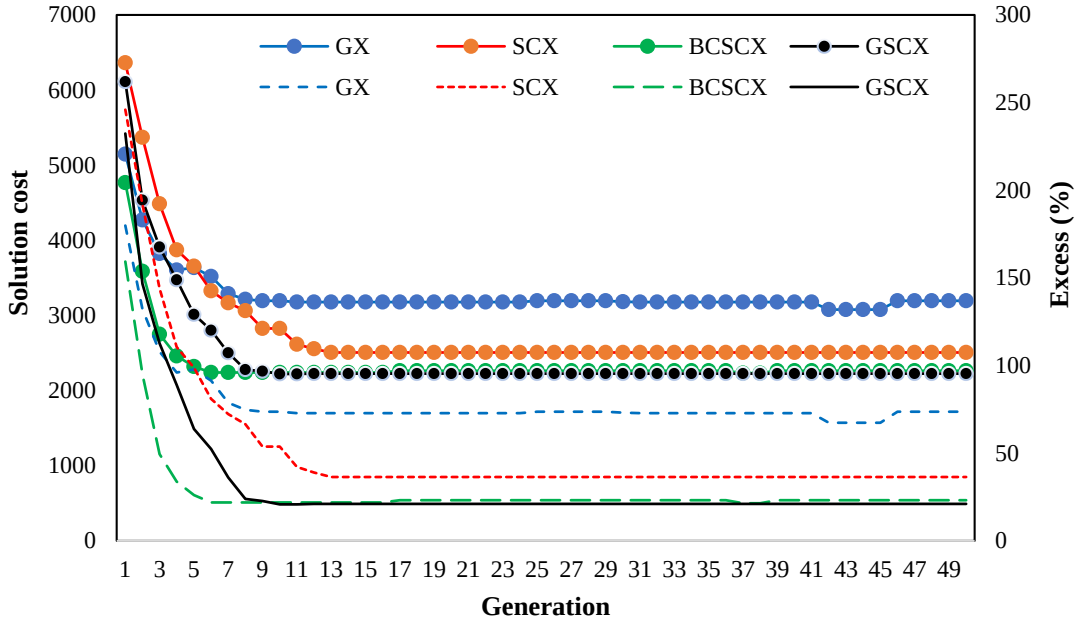

Fig. 1 Performance of four crossover operators for the instance ftv64

The results of experiments by the four GA variants are summarized in Tables 2(a & b) and 4. 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 a GA variant considered by a certain selection of crossover operator. The result is described by its best cost, average 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 following formula.

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

The Tables 2(a & b) report results by the GA variants for the asymmetric TSPLIB instances.

Table 2 (a): Summary of the results by the variant GAs for asymmetric TSPLIB instances

Instance	n	Results	GX	SCX	BCSCX	GSCX
ftv33 (1286)	34	Best Sol	1510	1371	1405	1380
		Avg. Sol	1679.9	1474.48	1478.94	**1458.48**
		Avg. Exc (%)	30.63	14.66	15.00	13.41
		S.D.	52.31	49.02	42.04	47.24
		Avg. Time	0.07	0.04	0.00	0.05
ftv35 (1473)	36	Best Sol	1667	1535	1586	1531
		Avg. Sol	1850.08	1636.56	1657.08	**1631.32**
		Avg. Exc (%)	25.60	11.10	12.50	10.75
		S.D.	53.92	59.86	31.98	47.09
		Avg. Time	0.11	0.13	0.07	0.05
ftv38 (1530)	39	Best Sol	1746	1630	1619	1613
		Avg. Sol	1911.48	1705.68	1712.56	**1690.50**
		Avg. Exc (%)	24.93	11.48	11.93	10.49
		S.D.	43.37	30.78	20.85	39.65
		Avg. Time	0.13	0.10	0.07	0.05
p43 (5620)	43	Best Sol	5788	5632	5654	5631
		Avg. Sol	5851.62	5642.86	5680.70	**5641.20**
		Avg. Exc (%)	4.12	0.41	1.08	0.38
		S.D.	23.28	7.26	11.69	6.97
		Avg. Time	0.34	0.15	0.15	0.17
ftv44		Best Sol	1899	1748	1758	1706

(1613)	45	Avg. Sol	2093.12	1867.96	1869.12	**1853.28**
		Avg. Exc (%)	29.77	15.81	15.88	14.90
		S.D.	64.72	58.88	51.53	60.57
		Avg. Time	0.18	0.22	0.06	0.12
ftv47 (1776)	48	Best Sol	2350	1880	2002	1864
		Avg. Sol	2607.24	2048.06	2140.06	**2021.72**
		Avg. Exc (%)	46.80	15.32	20.50	13.84
		S.D.	103.41	90.37	69.21	70.92
		Avg. Time	0.3	0.34	0.26	0.26
ry48p (14422)	48	Best Sol	20792	15425	15764	15469
		Avg. Sol	24152.6	16211.96	16266.92	**16150.78**
		Avg. Exc (%)	67.47	12.41	12.79	11.99
		S.D.	1790.95	331.80	224.72	278.65
		Avg. Time	0.23	0.32	0.09	0.17
ft53 (6905)	53	Best Sol	10109	7678	7848	7882
		Avg. Sol	11144.14	**8494.82**	8524.50	8614.86
		Avg. Exc (%)	61.39	23.02	23.45	24.76
		S.D.	455.60	246.93	183.91	277.13
		Avg. Time	0.25	0.26	0.31	0.35
ftv55 (1608)	56	Best Sol	2106	1717	1777	1723
		Avg. Sol	2305.6	1871.44	1865.32	**1841.82**
		Avg. Exc (%)	43.38	16.38	16.00	14.54
		S.D.	80.58	85.30	65.52	50.89
		Avg. Time	0.32	0.47	0.44	0.36
ftv64 (1839)	65	Best Sol	2545	2022	2105	1990
		Avg. Sol	2836.6	2224.24	2216.32	**2140.28**
		Avg. Exc (%)	54.25	20.95	20.52	16.38
		S.D.	141.67	104.31	55.18	76.32
		Avg. Time	0.54	0.64	0.33	0.29
ft70	70	Best Sol	43629	40782	40638	41129
(38673)		Avg. Sol	45288.32	42406.22	**41761.26**	42185.60
		Avg. Exc (%)	17.11	9.65	7.99	9.08
		S.D.	1002.38	579.69	384.31	571.46
		Avg. Time	0.86	0.79	1.38	0.64

Table 2(b): Summary of the results by the variant GAs for asymmetric TSPLIB instances

Instance	**n**	**Results**	**GX**	**SCX**	**BCSCX**	**GSCX**
ftv70 (1950)	71	Best Sol	2467	2155	2176	2118
		Avg. Sol	2734.46	2392.74	2309.22	**2296.32**
		Avg. Exc (%)	40.23	22.70	18.42	17.76
		S.D.	107.33	109.02	55.30	69.83
		Avg. Time	0.78	0.79	0.67	0.48
kro124p (36230)	100	Best Sol	81824	41331	41668	41251
		Avg. Sol	89253.80	43674.54	**42544.46**	42829.16
		Avg. Exc (%)	146.35	20.55	17.43	18.21
		S.D.	2557.47	638.43	566.01	780.6
		Avg. Time	0.57	1.25	0.13	0.42
ftv170 (2755)	171	Best Sol	4667	3285	3257	3656
		Avg. Sol	4817.32	**3523.74**	3608.40	3799.50
		Avg. Exc (%)	74.86	27.90	30.98	37.91
		S.D.	71.52	113.55	92.49	130.15
		Avg. Time	4.17	3.77	3.23	1.74
rbg323 (1326)	323	Best Sol	2102	1658	1660	1597
		Avg. Sol	2192.08	1718.76	1725.52	**1677.12**
		Avg. Exc (%)	65.32	29.62	30.13	26.48
		S.D.	31.70	22.19	20.63	34.19
		Avg. Time	15.51	12.37	24.79	15.76
rbg358 (1163)	358	Best Sol	2054	1524	1582	1522
		Avg. Sol	2203.12	1699.20	1711.30	**1591.04**
		Avg. Exc (%)	89.43	46.10	47.15	36.80
		S.D.	52.02	29.22	23.23	36.93

		Avg. Time	17.28	16.71	30.14	18.16
rbg403 (2465)	403	Best Sol	3760	3314	3229	3149
		Avg. Sol	3828.34	3401.18	3479.62	**3214.22**
		Avg. Exc (%)	55.31	37.98	41.16	30.39
		S.D.	39.16	26.48	24.06	42.23
		Avg. Time	21.43	19.98	33.89	28.96
rbg443 (2720)	443	Best Sol	3705	3705	3710	3545
		Avg. Sol	3742.88	3882.52	3872.66	**3644.00**
		Avg. Exc (%)	37.61	42.74	42.38	33.97
		S.D.	22.11	26.78	25.53	54.96
		Avg. Time	37.87	26.82	42.64	35.31

The crossover GX could not obtain lowest average cost for any asymmetric instance. The crossover SCX obtains lowest average costs with lowest S.D. for the instances ft53 and ftv170, whereas BCSCX obtains lowest average costs with lowest S.D. for ft70 and kro124p respectively. The proposed crossover GSCX obtains lowest average costs with lower S.D. for remaining fourteen instances, namely, ftv33, ftv35, ftv38, p43, ftv44, ftv47, ry48p, ftv55, ftv64, ftv70, rbg323, rbg358, rbg403 and rbg443. So, the proposed crossover GSCX is found to be the best. The results are depicted in Figure 2, which also shows the effectiveness of our proposed crossover operator GSCX. The crossovers SCX and BCSCX are competing, and GX is the worst.

In order to decide if GSCX-based GA average is significantly different from 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 [26]:

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

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 $\bar{X}_2$ and SD_2 values are obtained by the GSCX-based GA, while $\bar{X}_1$ and SD_1 values are obtained by other GA variants.

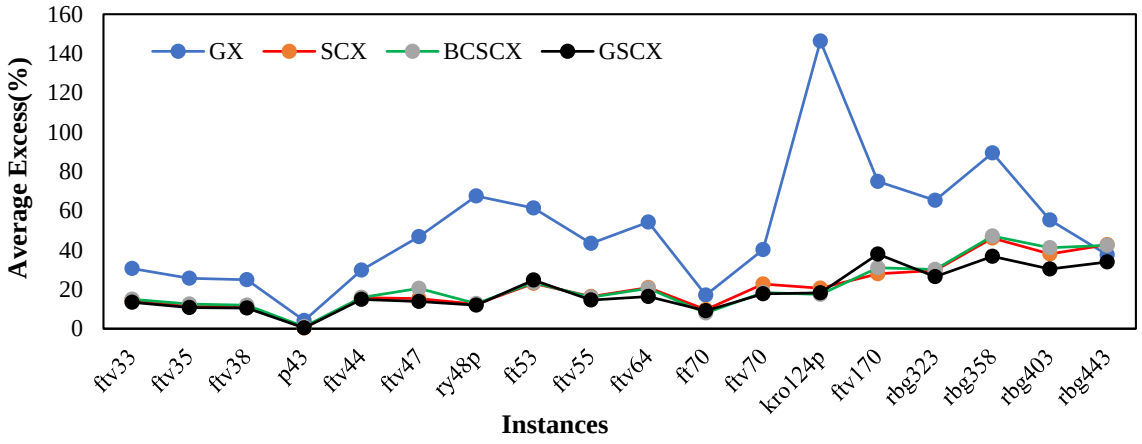

Fig. 2 Average Excess(%) by GA variants on asymmetric TSPLIB instances

Table 3 reports the calculated values of the t-statistic. The t values can be positive or negative. The positive value shows that the GSCX finds better solution than the competitive GA variant. The negative value shows that the competitive GA

finds better solution. The confidence interval at 95% confidence level (t0.05 = 1.96) is used. When t-value is greater than 1.96, there is a significant difference between the two values, so, GSCX finds better solution if t-value is positive, and if t-value is negative then the competitive GA finds better solution. When t-value is less than 1.96, then there is no significant difference between the observed values. The information about the GA variants that obtained significantly better solutions is also reported in the table.

For three instances there is no statistically significant difference between GSCX and BCSCX. On twelve instances GSCX is found better than BCSCX. There is no statistically significant difference between GSCX and SCX for seven instances. GSCX is found better than SCX on nine instances. Next, GSCX found better than GX on all eighteen instances. Hence, GSCX is found to be the best one.

Table 3: The calculated t-values by variant GAs against GSCX on asymmetric TSPLIB instances and the information about significantly better variant GAs

Instance	**GX**	**SCX**	**BCSCX**	**Instance**	**GX**	**SCX**	**BCSCX**
ftv33	21.9900	1.6452	2.2648	ftv64	30.2899	4.54 72	5.6518
Better	GSCX	-----	GSCX	Better	GSCX	GSCX	GSCX
ftv35	21.3908	0.4816	3.1678	ft70	18.8234	1.89 72	-4.3132
Better	GSCX	-----	GSCX	Better	GSCX	-----	**BCSCX**
ftv38	26.3237	2.1169	3.4470	Ftv70	23.9520	5.21 32	1.0138
Better	GSCX	GSCX	GSCX	Better	GSCX	GSCX	-----
p43	60.6123	1.1546	20.3157	kro124p	121.5329	5.86 82	-2.0669
Better	GSCX	-----	GSCX	Better	GSCX	GSCX	**BCSCX**
ftv44	18.9400	1.2165	1.3943	ftv170	47.9760	-11.1759	-8.3781
Better	GSCX	-----	-----	Better	GSCX	**SCX**	**BCSCX**
ftv47	32.6865	1.6050	8.3595	rbg323	77.3138	7.15 12	8.4845
Better	GSCX	-----	GSCX	Better	GSCX	GSCX	GSCX
ry48p	30.9036	0.9884	2.2711	rbg358	67.1605	16.0775	19.2951
Better	GSCX	-----	GSCX	Better	GSCX	GSCX	GSCX
ft53	33.2010	-2.2638	-1.9017	rbg403	74.6426	26.2556	38.2239
Better	GSCX	**SCX**	-----	Better	GSCX	GSCX	GSCX
ftv55	34.0641	2.0874	1.9828	rbg443	11.6839	27.3097	26.4128
Better	GSCX	GSCX	GSCX	Better	GSCX	GSCX	GSCX

The Table 4 reports results by the GA variants for the symmetric TSPLIB instances. The crossover GX obtains lowest average cost with lowest S.D. for only one instance, dantzig42. The crossover SCX could not obtain lowest average cost for any symmetric instance, however, it obtains best solution at least once in 50 runs for the instances gr21 and fri26. The BCSCX obtains lowest average costs with lowest S.D. for four instances, fri26, eil51, lin105 and a280. The proposed crossover GSCX obtains lowest average costs with lower S.D. for remaining five instances, namely, gr21, bayg29, berlin52, pr76 and d198. So, the proposed crossover GSCX is found to be the best. The results are depicted in Figure 3, which also shows the effectiveness of our proposed crossover operator GSCX.

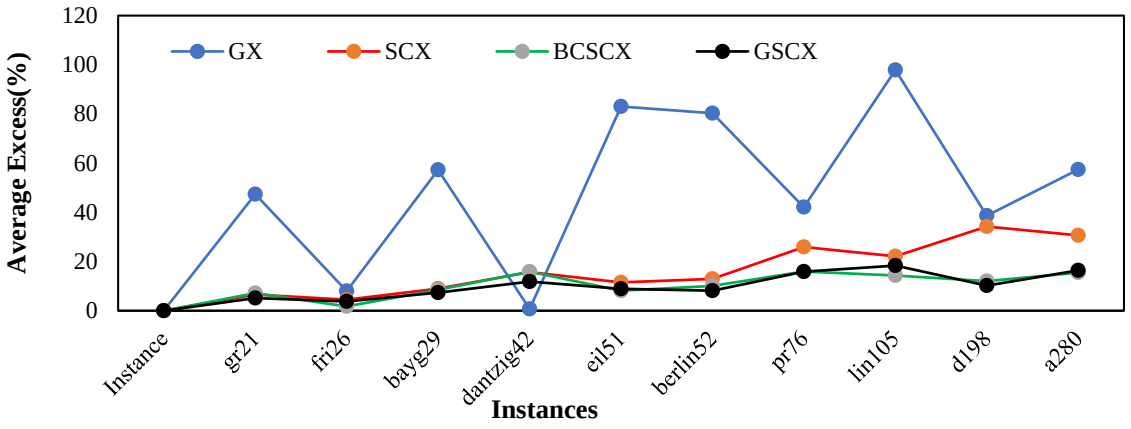

Fig. 3 Average Excess(%) by GA variants on symmetric TSPLIB instances

Table 4: Summary of the results by the variant GAs for symmetric TSPLIB instances

Instance	**n**	**Results**	**GX**	**SCX**	**BCSCX**	**GSCX**
gr21 (2707)	21	Best Sol	3614	2707	2707	2707
		Avg. Sol	3988.84	2885.36	2900.82	**2845.28**
		Avg. Exc(%)	47.35	6.59	7.16	5.11
		S.D.	169.99	104.16	86.26	86.33
		Avg. Time	0.05	0.03	0.05	0.01
fri26 (937)	26	Best Sol	955	937	937	937
		Avg. Sol	1012.08	979.04	**953.44**	972.62
		Avg. Exc(%)	8.01	4.49	1.75	3.80
		S.D.	5.51	24.54	9.37	17.72
		Avg. Time	0.12	0.04	0.02	0.02
bayg29 (1610)	29	Best Sol	2310	1634	1656	1634
		Avg. Sol	2531.36	1754.78	1747.58	**1727.94**
		Avg. Exc(%)	57.23	8.99	8.55	7.33
		S.D.	93.28	73.42	52.01	61.98
		Avg. Time	0.11	0.06	0.02	0.02
dantzig42 (699)	42	Best Sol	699	723	725	723
		Avg. Sol	**704.18**	808.72	810.08	781.80
		Avg. Exc(%)	0.74	15.70	15.89	11.85
		S.D.	3.64	30.50	28.83	26.27
		Avg. Time	0.04	0.06	0.01	0.08
eil51 (426)	51	Best Sol	725	441	444	436
		Avg. Sol	779.66	475.02	**460.66**	463.94
		Avg. Exc(%)	83.02	11.51	8.14	8.91
		S.D.	18.55	16.51	6.76	12.01
		Avg. Time	0.44	0.33	0.79	0.38
berlin52 (7542)	52	Best Sol	12677	7908	7919	7926
		Avg. Sol	13598.98	8517.42	8300.94	**8156.70**
		Avg. Exc(%)	80.31	12.93	10.06	8.15
		S.D.	542.4	331.35	205.66	189.50
		Avg. Time	0.34	0.40	0.11	0.22
pr76 (108159)	76	Best Sol	149635	122955	118596	116844
		Avg. Sol	153701.9	136178.38	125392.00	**125293.76**

		Avg. Exc(%)	42.11	25.91	15.93	15.84
		S.D.	2130.97	6419.40	3220.70	4273.80
		Avg. Time	0.18	0.74	0.20	0.48
lin105	105	Best Sol	26929	17254	15637	15921
(14379)		Avg. Sol	28445.84	17558.94	**16441.34**	17018.08
		Avg. Exc(%)	97.83	22.12	14.34	18.35
		S.D.	395.12	1075.52	374.69	557.25
		Avg. Time	2.04	1.46	0.11	1.10
d198	198	Best Sol	21212	19633	17441	17244
(15780)		Avg. Sol	21872.64	21184.38	17682.28	**17388.94**
		Avg. Exc(%)	38.61	34.25	12.06	10.20
		S.D.	334.17	806.58	131.52	490.07
		Avg. Time	7.3	1.72	5.46	2.82
a280	280	Best Sol	4905	3134	2820	2980
2579		Avg. Sol	4059.62	3368.60	**2979.00**	3001.00
		Avg. Exc(%)	57.41	30.62	15.51	16.36
		S.D.	72.46	153.46	57.45	99.19
		Avg. Time	4.07	0.75	9.33	3.34

For the symmetric instances also, to decide if GSCX based GA average is significantly different than the averages obtained by other GA variants, we perform Student's t-test and the calculated t-values are reported in the Table 5.

Table 5: The calculated t-values by variant GAs against GSCX on symmetric TSPLIB instances and the information about significantly better variant GAs

Instance	**GX**	**SCX**	**BCSCX**	**Instance**	**GX**	**SCX**	**BCSCX**
gr21	41.9863	2.07378	3.1857	berlin52	66.3057	6.61511	3.6105
Better	GSCX	GSCX	GSCX	Better	GSCX	GSCX	GSCX
fri26	14.88850	1.48847	-6.6980	pr76	41.64028	9.87988	0.1285
Better	GSCX	------	**BCSCX**	Better	GSCX	GSCX	------
bayg29	50.2164	1.95554	1.6992	lin105	117.1021	3.1256	-6.0121
Better	GSCX	------	------	Better	GSCX	GSCX	**BCSCX**
dantzig42	-20.4872	4.68133	5.0754	d198	52.9131	28.1504	4.0468
Better	**GX**	GSCX	GSCX	Better	GSCX	GSCX	GSCX
eil51	100.0087	3.79989	-1.6660	a280	60.3263	14.0823	-1.3435
Better	GSCX	GSCX	------	Better	GSCX	GSCX	------

There is no statistically significant difference between GSCX and BCSCX for four instances. On four instances, gr21, dantzig42, berlin52 and d198, GSCX is better than BCSCX, whereas BCSCX is better than GSCX on two instances, fri26 and lin105. For two instances there is no statistically significant difference between GSCX and SCX, and GSCX is better than SCX for remaining eight instances. On only one instance, dantzig42, GX is better than GSCX, and on remaining nine instances GSCX is better than GX.

Based on the above study it very clear that the proposed crossover operator GSCX is the best. To decide better operator between BCSCX and SCX, we made statistical t-test between them on all twenty eight instances (not reported in any table) and found that there is no significant difference between them for twelve instances. Out of remaining sixteen instances, BCSCX is found better on ten instances, whereas SCX is better on six instances. Hence BCSCX is better than SCX, and GX is the worst.

Table 6: Comparative study between HX and our proposed GSCX

Results	Instance	HX	GSCX	Instance	HX	GSCX	Instance	HX	GSCX
Best Sol		2707	2707		10464	7926		12255	2980
Avg. Sol	gr21	**2807.92**	2845.28	berlin52	13006.74	**8156.70**	a280	13011.00	**3001.00**
Avg. Exc(%)		3.73	5.11		72.46	8.15		404.50	16.36
Best Sol		937	937		11102	7882		5275	1597
Avg. Sol	fri26	**972.26**	972.62	ft53	12530.46	**8614.86**	rbg323	5372.22	**1677.12**
Avg. Exc(%)		3.76	3.80		81.47	24.76		305.14	26.48
Best Sol		1447	1380		113960	41251		5880	1522
Avg. Sol	ftv33	1618.12	**1458.48**	kro124p	120524.24	**42829.16**	rbg358	5999.44	**1591.04**
Avg. Exc(%)		25.83	13.41		232.66	18.21		415.86	36.80
Best Sol		1740	1613		43612	15921		6659	3149
Avg. Sol	ftv38	2000.46	**1690.50**	lin105	48589.64	**17018.08**	rbg403	6743.24	**3214.22**
Avg. Exc(%)		30.75	10.49		237.92	18.35		173.56	30.39
Best Sol		699	723		13215	3656		7218	3545
Avg. Sol	dantzig42	821.98	**781.80**	ftv170	13878.00	**3799.50**	rbg443	7318.74	**3644.00**
Avg. Exc(%)		17.59	11.85		403.74	37.91		169.07	33.97

Recently, a comparative study among eleven crossover operators for the TSP has been carried out and found that heuristic crossover (HX) [27] is the best [28]. In HX, a greedy heuristic is applied to create an offspring from two parent chromosomes. So, GA using HX is implemented and run on only fifteen selected instances under same GA settings for comparing with our proposed GSCX. The Table 6 reports a comparative study between HX and GSCX. In terms of best solution cost, except for the instance dantzig42, GSCX is found better than HX, and in terms of worst and average solution costs, except for the instances gr21 and fri26, GSCX is found better than HX. From this study it is very clear that our proposed crossover GSCX is far better than HX.

5. Conclusion & Discussions

For solving the benchmark TSP using GAs, several crossover operators have been developed by different researchers. We have also proposed a new crossover operator, GSCX, for solving the TSP. This proposed crossover operator is a modification of the SCX for improving the quality of offspring. We considered three existing crossover operators, namely, GX, SCX and BCSCX to compare with our proposed crossover GSCX. We first applied these operators in manual experiment on two parent chromosomes to produce an offspring, for each crossover operator, and found that our crossover GSCX is the best among the four crossover operators. We then developed four variant GAs using four different crossover operators and carried out comparative study of the GAs on eighteen asymmetric and ten symmetric TSPLIB instances. In terms of solution quality, our comparative study showed that our proposed crossover operator GSCX is the best, BCSCX is the second-best, SCX is the third best and GX is the worst. This observation is verified by Student's t-test at 95% confidence level. Further we carried out a comparative study between HX and GSCX, found that GSCX is far better than HX. Thus, GSCX might be used for other related combinatorial optimization problems.

In this present study, the objective was to compare the quality of the solutions obtained by the existing three and proposed one crossover operators. We did not look to improve the solution quality by the operators. So, we considered the original version of the operators. Also, we did not use any local search technique to design the most competitive algorithm for the TSP. So, we have used simple and pure GA process. Also, we set highest crossover probability to show the exact characteristics of crossover operators. Mutation was used with lowest probability just not to get stuck in local minima quickly. However, good local search procedure can be incorporated to hybridize the algorithm to solve problem instances more accurately, which is under our investigation.

Conflicts of Interest

The author declares that there is no conflict of interest regarding the publication of this paper.

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 45 (5) (1998) 753–782.
[2] C.P. Ravikumar, Solving large-scale travelling salesperson problems on parallel machines, Microprocessors and Microsystems 16(3) (1992) 149-158.
[3] D.E. Goldberg, Genetic algorithms in search, optimization, and machine learning, Addison-Wesley, New York, 1989.
[4] Z.H. Ahmed, Algorithms for the quadratic assignment problem, LAP LAMBERT Academic Publishing, Mauritius, 2019, 104 pages.
[5] D.E. Goldberg, 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.
[6] L. Davis, Job-shop scheduling with genetic algorithms, Proceedings of an International Conference on Genetic Algorithms and Their Applications, pp. 136-140, 1985.
[7] G. Syswerda, Schedule optimization using genetic algorithms, In Davis, L. (ed.) Handbook of Genetic Algorithms, New York: Van Nostrand Reinhold, 332–349, 1991.
[8] J. Grefenstette, R. Gopal, B. Rosmaita, 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, 160–168, 1985.
[9] I.M. Oliver, D. J. Smith, 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.
[10] D. Whitley, T. Starkweather, 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, 350-372, 1991.
[11] N.J. Radcliffe, 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, 51-72, 1995.
[12] Z.H. Ahmed, Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator, International Journal of Biometrics & Bioinformatics 3 (2010) 96-105.
[13] Z.H. Ahmed, Improved genetic algorithms for the traveling salesman problem, International Journal of Process Management and Benchmarking 4(1) (2014) 109-124.
[14] S. Kang, S.-S. Kim, J.-H. Won, Y.-M. Kang, Bidirectional constructive crossover for evolutionary approach to travelling salesman problem, 2015 5th IEEE International Conference on IT Convergence and Security (ICITCS) (2015) 1-4.
[15] K. Deb, Optimization for engineering design: algorithms and examples, Prentice Hall of India Pvt. Ltd., New Delhi, India, 1995.
[16] I. H. Khan, Assessing different crossover operators for travelling salesman problem, International Journal of Intelligent Systems and Applications (IJISA) 7(11) (2015) 19-25.
[17] Z.H. Ahmed, A hybrid genetic algorithm for the bottleneck traveling salesman problem, ACM Transactions on Embedded Computing Systems 12 (2013) Art. No. 9.
[18] Z.H. Ahmed, An experimental study of a hybrid genetic algorithm for the maximum travelling salesman problem, Mathematical Sciences 7 (2013) 1-7.
[19] Z.H. Ahmed, The ordered clustered travelling salesman problem: A hybrid genetic algorithm, The Scientific World Journal, Art ID 258207 (2014) 13 pages.
[20] Z.H. Ahmed, A simple genetic algorithm using sequential constructive crossover for the quadratic assignment problem, Journal of Scientific and Industrial Research 73 (2014) 763-766.
[21] Z.H. Ahmed, Experimental analysis of crossover and mutation operators for the quadratic assignment problem, Annals of Operations Research 247 (2016) 833-851.
[22] Z.H. Ahmed, The minimum latency problem: a hybrid genetic algorithm, IJCSNS International Journal of Computer Science and Network Security 18(11) (2018) 153-158.
[23] Z.H. Ahmed, Performance analysis of hybrid genetic algorithms for the generalized assignment problem, IJCSNS International Journal of Computer Science and Network Security 19(9) (2019) 216-222.
[24] M.A. Al-Omeer, Z.H. Ahmed, Comparative study of crossover operators for the MTSP, 2019 International Conference on Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, 3-4 April 2019, 1-6.
[25] G. Reinelt, TSPLIB, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
[26] M. Nikolić, D. Teodorović, Empirical study of the bee colony optimization (BCO) algorithm, Expert Systems with Applications 40 (2013) 4609–4620.
[27] J. J. Grefenstette, Incorporating problem specific knowledge into genetic algorithms. In L. Davis (Ed.), Genetic algorithms and simulated annealing, London, UK: Pitman / Pearson, (1987) 42–60.
[28] T. Weise, Y. Jiang, Q. Qi, W. Liu, A branch-and-bound-based crossover operator for the traveling salesman problem. International Journal of Cognitive Informatics and Natural Intelligence (IJCINI) 13(3) (2019), 1-18.

Zakir Hussain Ahmed is a Full Professor in the Department of Mathematics and Statistics at Al Imam Mohammad Ibn Saud Islamic University, Riyadh, Kingdom of Saudi Arabia. Till the end of 2019, he was in the Department of Computer Science at the same University. He obtained MSc in Mathematics (Gold Medalist), Diploma in Computer Application, MTech in Information Technology and PhD in Mathematical Sciences (Artificial Intelligence/Combinatorial Optimization) from Tezpur University (Central), Assam, India. Before joining the current

position, he served in Tezpur University, Sikkim Manipal Institute of Technology, Asansol Engineering College and Jaypee Institute of Engineering and Technology, India. His research interests include artificial intelligence, combinatorial optimization, digital image processing and pattern recognition. He has several publications in the fields of artificial intelligence, combinatorial optimization and image processing.