Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.049704

ARTICLE

Appropriate Combination of Crossover Operator and Mutation Operator in Genetic Algorithms for the Travelling Salesman Problem

Zakir Hussain Ahmed[1,*], Habibollah Haron[2] and Abdullah Al-Tameem[3]

[1]Department of Mathematics and Statistics, College of Science, Imam Mohammad Ibn Saud Islamic University (IMSIU), Riyadh, 11432, Kingdom of Saudi Arabia

[2]Faculty of Computing, Universiti Teknologi Malaysia, Johor Bahru, 81310, Malaysia

[3]College of Computer and Information Sciences, Imam Mohammad Ibn Saud Islamic University (IMSIU), Riyadh, 11432, Kingdom of Saudi Arabia

*Corresponding Author: Zakir Hussain Ahmed. Email: zaahmed@imamu.edu.sa

Received: 16 January 2024 Accepted: 28 March 2024 Published: 15 May 2024

ABSTRACT

Genetic algorithms (GAs) are very good metaheuristic algorithms that are suitable for solving NP-hard combinatorial optimization problems. A simple GA begins with a set of solutions represented by a population of chromosomes and then uses the idea of survival of the fittest in the selection process to select some fitter chromosomes. It uses a crossover operator to create better offspring chromosomes and thus, converges the population. Also, it uses a mutation operator to explore the unexplored areas by the crossover operator, and thus, diversifies the GA search space. A combination of crossover and mutation operators makes the GA search strong enough to reach the optimal solution. However, appropriate selection and combination of crossover operator and mutation operator can lead to a very good GA for solving an optimization problem. In this present paper, we aim to study the benchmark traveling salesman problem (TSP). We developed several genetic algorithms using seven crossover operators and six mutation operators for the TSP and then compared them to some benchmark TSPLIB instances. The experimental studies show the effectiveness of the combination of a comprehensive sequential constructive crossover operator and insertion mutation operator for the problem. The GA using the comprehensive sequential constructive crossover with insertion mutation could find average solutions whose average percentage of excesses from the best-known solutions are between 0.22 and 14.94 for our experimented problem instances.

KEYWORDS

Travelling salesman problem; genetic algorithms; crossover operator; mutation operator; comprehensive sequential constructive crossover; insertion mutation

1 Introduction

The travelling salesman problem (TSP) is an important combinatorial optimization problem (COP) that is proven as NP-hard [1]. It is a very difficult COP that was first documented in 1759 by Euler as the Knights' tour problem. Later on, in 1932, in a German book, the term 'travelling salesman' was first used. Furthermore, in 1948, the RAND Corporation formally introduced the problem. The

problem may be stated as below: There is a set of n nodes (cities) including the depot 'node 1'. For every pair of nodes (i, j), a distance (or travel time or travel cost) matrix, $D = [d_{ij}]$, is given. The aim is to find a minimum distance (or cost) Hamiltonian cycle. There are two types of TSP-asymmetric TSP and symmetric TSP. The TSP is symmetric if $d_{ij} = d_{ji}$, for all node pairs i, j; otherwise, asymmetric.

The TSP has several practical applications in very-large-scale integrated circuits, vehicle routing, automatic drilling of printed circuit boards and circuits, X-ray crystallography, computer wiring, DNA sequencing, movement of people, machine scheduling problems, packet routing in the global system for mobile communications, logistics service [2], etc. There are probably (n–1)! solutions for the asymmetric TSP and probably $\frac{(n-1)!}{2}$ solutions for the symmetric TSP, out of those at least one gives the minimum cost. The problem is very difficult as the probable number of solutions is large in both TSP types. As the problem is very difficult and can be used to model several other difficult problems, many researchers developed many exact and metaheuristic/heuristic algorithms for solving the problem. Exact algorithms include branch-and-bound [3], branch-and-cut [4], integer programming [5], and lexisearch [6]. In general, instances of size $n > 100$ cannot be solved optimally by an exact algorithm in a reasonable time. As many real-world problem instances are of size bigger than 100, thus, to solve these problem instances one must use heuristic algorithms. Heuristic algorithms do not ensure the optimality of the obtained solution; however, they obtain a nearly exact optimal solution within a short computational effort. Metaheuristic algorithms are very recent heuristic algorithms that can be developed for various COPs. Examples of such algorithms for the TSP are ant colony optimization [7], genetic algorithm [8], simulated annealing [9], tabu search [10], particle swarm optimization [11], etc. Amongst these metaheuristics, the genetic algorithm is the best algorithm for the TSP as well as other COPs.

Genetic algorithm (GA) is a very robust and effective metaheuristic method for solving large-sized COPs. It is based on the evolutionary process of natural biology. Each legitimate solution to a particular problem is represented by a chromosome whose fitness is assessed by its objective function. In a simple GA, a set (or population) of chromosomes is produced randomly at an initial stage, and then probably three operators are applied to create new and probably better populations in succeeding iterations (or generations). Selection is the first operator that removes and copies probabilistically some chromosomes of a generation and passes them to the next generation. The second operator, crossover, arbitrarily selects a pair of parent chromosomes and then mates them to produce new and probably better offspring chromosome(s). Mutation is the third operator that arbitrarily changes some genes (or position values) of a chromosome. Amongst them, crossover is the highly valuable operator that compresses the search space, while mutation expands the search space. As a result, the probability of utilizing the mutation operator is fixed very low, while the probability of utilizing the crossover operator is fixed very high. These operators are repeatedly applied until the optimal solution is obtained, or the predefined maximum number of iterations (generations) is reached [12].

Since crossover is a highly valuable operator, various crossovers are developed and enhanced to obtain a quality solution to the TSP. In this study, seven crossover and six mutation operators are considered in GAs on TSP, and then a comparative study amongst them has been conducted on some standard TSPLIB instances [13]. A comparative study shows that the comprehensive sequential constructive crossover [14] is the best crossover operator and insertion mutation [15] is the best mutation operator, and their combination is one of the best combinations. It should be mentioned that the purpose of this study is only to examine the efficiency of the crossover operator combined with the mutation operator, not to obtain the optimal solution to the problem.

The rest of the paper is arranged as follows: a brief review of some crossover operators and mutation operators for the TSP is shown in Section 2. Seven crossover operators have been briefly discussed in Section 3, whereas Section 4 discusses six mutation operators for our simple GAs. Section 5 reports computational experiments for simple genetic algorithms using various crossover and mutation operators. Finally, concluding remarks and future works are presented in Section 6.

2 A Review of Crossover and Mutation Operators for the TSP

In GA search for the solution, new chromosomes are created from the old ones using crossover and mutation operators. A pair of chromosomes is selected randomly from the mating pool using the crossover operator. Then a crossover site along their shared length is chosen randomly, and the data after the site of the chromosomes are exchanged, thus producing two new offspring chromosomes. However, this simple crossover process may not produce valid chromosomes for the TSP. Numerous crossover operators were suggested for the TSP. Distance-based crossover and blind crossover are two major categories of crossover operators available in the literature. Offspring chromosomes are produced using distances between nodes in a distance-based crossover operator, while offspring chromosomes do not require any data on the problem in a blind crossover operator but rather require knowing only about the problem's limitations, if any. The heuristic crossover (HX) [16], greedy crossover (GX) [17], sequential constructive crossover (SCX) [8], distance-preserving crossover (DPX) [18], etc., are some well-known distance-based crossover operators, and ordered crossover (OX) [17], partially mapped crossover (PMX) [19], position based operator (PBX) and order based crossover (OBX) [20], cycle crossover (CX) [21], alternating edges crossover (AEX) [22], generalized N-point crossover (GNX) [23], edge recombination crossover (ERX) [24], etc., are some well-known blind crossover operators.

In the traditional mutation process, a gene (or position) in a chromosome is selected randomly under some prefixed probability, and the gene value is changed, thus, the chromosome is modified. The purpose of the mutation operator is to prevent the early convergence of the GA to non-optimal solutions by bringing back the lost genetic material or inserting new information into the population. As the inferior chromosomes are dropped in each generation, some genetic characteristics might be missing forever. By performing random modifications in chromosomes, GA ensures that a new search space is touched, and that selection and crossover might not fully ensure. This way, the mutation ensures that no valuable characteristics are missing early, and thus, it preserves population diversity. The traditional mutation process may not produce valid chromosomes for the TSP. Several mutation operators were suggested for the TSP. Insertion mutation [14], exchange mutation [25], inversion mutation [26], displacement mutation [27], heuristic mutation [28], greedy swap mutation [29], adaptive mutation [30], etc., are some well-known mutation operators. In [31], an efficient mutation approach using transpose, shift-and-insert, and swap neighborhood operators, is suggested to produce the best solutions for the TSP. However, it is a local search method.

3 Crossover Operators for Our GAs

For the TSP, solutions are represented by chromosomes which are defined as a permutation of nodes. The nodes are labeled as {1, 2, 3, . . . , n}, where n is the size of the problem (that is, the total number of nodes in the network). For a 7-node problem instance, the tour {1→6→7→3→2→4→5→1} is represented by (1, 6, 7, 3, 2, 4, 5). The objective function is defined as the total distance of the tour and the fitness function is defined as the multiplicative inverse of the objective function. Consequently, a set of chromosomes, called the initial population, of prefixed size, P_s, is produced randomly. The stochastic

remainder selection method is used to produce a mating pool. The following seven crossover operators are used for our comparative study. Each crossover operator is applied under the crossover probability, P_c rule.

3.1 Sequential Constructive Crossover Operator

In [8], the SCX is suggested for the TSP that produces a good quality solution for both symmetric and asymmetric TSPLIB instances, which is compared with the other two crossover operators and found to be the best. It builds an offspring chromosome using better edges based on the edge values appearing in the parent chromosome. It also uses good-quality edges which are not present in either parent. It searches sequentially both parents and accepts the first unvisited node that is found after the present node. If an unvisited node is not seen in both parents, it searches sequentially from the start of the chromosome and picks the first unvisited node. A comparative study is reported in [32], which reports that SCX is the best among eight crossovers. The algorithm for the SCX [8] is presented in Algorithm 1.

Algorithm 1: Sequential constructive crossover algorithm

Input: n, Distance matrix $D = [d_{ij}]$, Crossover probability P_c, Pair of chromosomes.
Output: Offspring chromosome.
Generate random number $r \in [0,1]$.
if $(r \leq P_c)$ then **do**
  Set $p = 1$.
  The offspring chromosome contains only 'node 1'.
  for $i = 2$ **to** n **do**
    In each chromosome consider the first 'legitimate node' that appeared after 'node p'.
    if 'legitimate node' is not available in a chromosome, **then**
      Search from the start of the chromosome and consider the first 'legitimate node' that appeared after 'node p'.
    end if
    Suppose 'node α' and 'node β' are found in the 1st and 2nd chromosomes respectively.
    if $(d_{p\alpha} < d_{p\beta})$ then **do**
      Add 'node α' to the offspring chromosome.
    Else
      Add 'node β' to the offspring chromosome.
    end if
    Rename the present node as 'node p' and continue.
  end for
end if
Return the offspring chromosome

We consider the 10-node problem given as a distance matrix in Table 1. Let P1: (1, 6, 3, 9, 4, 5, 7, 8, 2, 10) and P2: (1, 7, 9, 3, 2, 4, 8, 5, 10, 6) be a pair of chosen parent chromosomes with distances 447 and 558 respectively. We use these chromosomes for applying to all crossover operators. We set the headquarters (first gene) as 'node 1', and so, the procedures are started from the 'node 1'. Applying the SCX on the parent chromosomes, one can obtain the offspring O: (1, 6, 7, 9, 4, 5, 8, 10, 3, 2) with the distance 502 (see Table 2).

Table 1: The distance matrix

Node	1	2	3	4	5	6	7	8	9	10
1	999	46	54	16	53	12	60	29	54	35
2	46	999	100	56	79	45	75	65	81	32
3	54	100	999	47	65	58	82	48	63	82
4	16	56	47	999	38	12	47	41	38	51
5	53	79	65	38	999	43	18	78	4	85
6	12	45	58	12	43	999	49	41	44	42
7	60	75	82	47	18	49	999	88	21	88
8	29	65	48	41	78	41	88	999	77	38
9	54	81	63	38	4	44	21	77	999	86
10	35	32	82	51	85	42	88	38	86	999

Table 2: Offspring chromosome by SCX

Present 'node p'	Legitimate node after 'node p' in		Accepted node	Partial offspring with distance D
	P1 with distance	P2 with distance		
1	Node 6 with $d_{16} = 12$	Node 7 with $d_{17} = 60$	Node 6	(1, 6) with D = 12
6	Node 3 with $d_{63} = 58$	Node 7 with $d_{67} = 49$	Node 7	(1, 6, 7) with D = 61
7	Node 8 with $d_{78} = 88$	Node 9 with $d_{79} = 21$	Node 9	(1, 6, 7, 9) with D = 82
9	Node 4 with $d_{94} = 38$	Node 3 with $d_{93} = 63$	Node 4	(1, 6, 7, 9, 4) with D = 120
4	Node 5 with $d_{45} = 38$	Node 8 with $d_{48} = 41$	Node 5	(1, 6, 7, 9, 4, 5) with D = 158
5	Node 8 with $d_{58} = 78$	Node 10 with $d_{5,10} = 85$	Node 8	(1, 6, 7, 9, 4, 5, 8) with D = 236
8	Node 2 with $d_{82} = 65$	Node 10 with $d_{8,10} = 38$	Node 10	(1, 6, 7, 9, 4, 5, 8, 10) with D = 274
10	Node 3 with $d_{10,3} = 82$	Node 3 with $d_{10,3} = 82$	Node 3	(1, 6, 7, 9, 4, 5, 8, 10, 3) with D = 356
3	Node 2 with $d_{32} = 100$	Node 2 with $d_{32} = 100$	Node 6	(1, 6, 7, 9, 4, 5, 8, 10, 3, 2) with D = 456
			Completed offspring with distance = 502	

3.2 Adaptive Sequential Constructive Crossover Operator

An adaptive SCX (ASCX) is developed in [33], which produces an offspring chromosome adaptively by searching in the forward/backward/mixed direction depending on distances to the next nodes. A comparative study amongst eight separate crossover operators shows that the operator ASCX is the best one. Algorithm 2 shows the pseudocode of the ASCX operator for the TSP.

Algorithm 2: Adaptive sequential constructive crossover algorithm

Input: n, D, P_c, Pair of parent chromosomes
Output: Offspring chromosome.
Generate a random number r $\in$ [0,1].
if (r $\leq P_c$) then **do**
Set p = 1, i = 1, q = 1, j = n + 1.
for k = 2 to n **do**
Forward (right) direction: In each parent chromosome consider the first 'un-visited' node found after 'node p'.
if no 'un-visited' node is found in a parent, **then**
Examine from the starting of the parent (wrap around) and choose the first 'un-visited' node found before 'node p'.
end if
Assume that 'node α' and 'node β' are selected from 1st and 2nd parents respectively.
Backward (left) direction: In each chromosome consider the first 'un-visited' node found left of 'node p'.
if no 'un-visited' node is found in a parent, **then**
Examine from the end of the parent (wrap around) and choose the first 'un-visited' node found after 'node p'.
end if
Assume that 'node γ' and 'node δ' are selected from 1st and 2nd parents respectively.
Now, suppose among four nodes, 'node u' is the closest with distance s = min. {$d_{p\alpha}$, $d_{p\beta}$, $d_{p\gamma}$, $d_{p\delta}$}.
Backward (left) direction: In each chromosome consider the first 'un-visited' node found left of 'node q'.
if no 'un-visited' node is found in a parent, **then**
Examine from the end of the parent (wrap around) and choose the first 'un-visited' node found after 'node q'.
end if
Assume that 'node w' and 'node x' are selected from 1st and 2nd parents respectively.
Forward (right) direction: In each chromosome consider the first 'un-visited' node found after 'node q'.
if no 'un-visited' node is found in a parent, **then**
Examine from the starting of the parent (wrap around) and choose the first 'un-visited' node found before 'node q'.
end if
Assume that 'node y' and 'node z' are selected from 1st and 2nd parents respectively.
Now, suppose among four nodes, 'node v' is the closest with distance t = min. {d_{wq}, d_{xq}, d_{yq}, d_{zq}}.
If s $\leq$ t, then set i = i + 1 and add 'node u' in position 'i' in the partially constructed offspring chromosome and set p = u. Otherwise, set j = j - 1 and add 'node v' in position 'j' in the partially constructed offspring chromosome and set q = v.
end for
end if
Return the offspring chromosome

Applying the ASCX on the parent chromosomes, one can obtain the offspring O: (1, 6, 7, 9, 4, 5, 8, 10, 3, 2) with the distance 406 (see Table 3).

Table 3: Offspring chromosome by ASCX

Node p	Legitimate node after 'node p' (in both directions) in		Node q	Legitimate node before 'node q' (in both directions) in		Accepted node	Partial offspring with distance D
	P1 (distance)	P2 (distance)		P1 (distance)	P2 (distance)		
1	6 ($d_{16} = 12$) 10 ($d_{1,10} = 35$)	7 ($d_{17} = 60$) 6 ($d_{16} = 12$)	1	10 ($d_{10,1} = 35$) 6 ($d_{61} = 12$)	6 ($d_{61} = 12$) 7 ($d_{71} = 60$)	Node 6	(1, 6, ∗, ∗, ∗, ∗, ∗, ∗, ∗, ∗) with D = 12
6	3 ($d_{63} = 58$) 10 ($d_{6,10} = 42$)	7 ($d_{67} = 49$) 10 ($d_{6,10} = 42$)	1	10 ($d_{10,1} = 35$) 3 ($d_{31} = 54$)	10 ($d_{10,1} = 35$) 7 ($d_{71} = 60$)	Node 10	(1, 6, ∗, ∗, ∗, ∗, ∗, ∗, ∗, 10) with D = 47
6	3 ($d_{63} = 58$) 2 ($d_{62} = 45$)	7 ($d_{67} = 49$) 5 ($d_{65} = 43$)	10	2 ($d_{2,10} = 32$) 3 ($d_{3,10} = 82$)	5 ($d_{5,10} = 85$) 7 ($d_{7,10} = 88$	Node 2	(1, 6, ∗, ∗, ∗, ∗, ∗, ∗, 2, 10) with D = 79
6	3 ($d_{63} = 58$) 8 ($d_{68} = 41$)	7 ($d_{67} = 49$) 5 ($d_{65} = 43$)	2	8 ($d_{82} = 65$) 3 ($d_{32} = 100$)	3 ($d_{32} = 100$) 4 ($d_{42} = 56$)	Node 8	(1, 6, 8, ∗, ∗, ∗, ∗, ∗, 2, 10) with D = 120
8	3 ($d_{83} = 48$) 7 ($d_{87} = 88$)	5 ($d_{85} = 78$) 4 ($d_{84} = 41$)	2	7 ($d_{72} = 75$) 3 ($d_{32} = 100$)	3 ($d_{32} = 100$) 4 ($d_{42} = 56$)	Node 4	(1, 6, 8, 4, ∗, ∗, ∗, ∗, 2, 10) with D = 161
4	5 ($d_{45} = 38$) 9 ($d_{49} = 38$)	5 ($d_{45} = 38$) 3 ($d_{43} = 47$)	2	7 ($d_{72} = 75$) 3 ($d_{32} = 100$)	3 ($d_{32} = 100$) 5 ($d_{52} = 79$)	Node 5	(1, 6, 8, 4, 5, ∗, ∗, ∗, 2, 10) with D = 199
5	7 ($d_{57} = 18$) 9 ($d_{59} = 4$)	7 ($d_{57} = 18$) 3 ($d_{53} = 65$)	2	7 ($d_{72} = 75$) 3 ($d_{32} = 100$)	3 ($d_{32} = 100$) 7 ($d_{72} = 75$)	Node 9	(1, 6, 8, 4, 5, 9, ∗, ∗, 2, 10) with D = 203
9	7 ($d_{97} = 21$) 3 ($d_{93} = 63$)	3 ($d_{93} = 63$) 7 ($d_{97} = 21$)	2	7 ($d_{72} = 75$) 3 ($d_{32} = 100$)	3 ($d_{32} = 100$) 7 ($d_{72} = 75$)	Node 7	(1, 6, 8, 4, 5, 9, 7, ∗, 2, 10) with D = 224
7	3 ($d_{73} = 82$) 3 ($d_{73} = 82$)	3 ($d_{73} = 82$) 3 ($d_{73} = 82$)	2	3 ($d_{32} = 100$) 3 ($d_{32} = 100$)	3 ($d_{32} = 100$) 3 ($d_{32} = 100$)	Node 3	(1, 6, 7, 9, 4, 5, 8, 10, 3, 2) with D = 406
			Completed offspring (1, 6, 7, 9, 4, 5, 8, 10, 3, 2) with distance = 406				

3.3 Greedy Sequential Constructive Crossover Operator

Furthermore, Ahmed [34] proposed another crossover named greedy SCX (GSCX) that produces an offspring chromosome using the greedy method adaptively by searching in the forward direction. Comparative studies among five different crossover operators show that the GSCX is the best one. Algorithm 3 shows the pseudocode of the GSCX operator for the TSP.

Algorithm 3: Greedy sequential constructive crossover algorithm

Input: n, D, P_c, Pair of parent chromosomes.
Output: Offspring chromosome.
Generate random number r ∈ [0,1].
if ($r \leq P_c$) then **do**
  Set p = 1.
  The offspring chromosome contains only 'node 1'.
  for i = 2 **to** n **do**
    In each parent chromosome consider the first un-visited' node found after 'node p'.
    if no 'un-visited' node is found in a parent, **then**

(Continued)

Algorithm 3 (continued)

Consider the closest 'un-visited' node from the group of remaining 'unvisited' nodes and concatenate it to the partially constructed offspring present node as 'node p' chromosome'. Rename the and continue to the next iteration.

end if

Suppose 'node α' and 'node β' are selected in 1st and 2nd chromosomes respectively.

if ($d_{\alpha p} < d_{\beta p}$) then **do**

Add 'node α' to the offspring chromosome.

else

Add 'node β' to the offspring chromosome.

end if

Rename the present node as 'node p' and continue.

end for

end if

Return the offspring chromosome

Applying the GSCX on the parent chromosomes, one can obtain the offspring O: (1, 6, 4, 5, 7, 9, 3, 8, 10, 2) with the distance 328 (see Table 4).

Table 4: Offspring chromosome by GSCX

Present 'node p'	Legitimate node after 'node p' in		Accepted node	Partial offspring with distance D
	P1 with distance	P2 with distance		
1	Node 6 with $d_{16} = 12$	Node 7 with $d_{17} = 60$	Node 6	(1, 6) with D = 12
6	Node 3 with $d_{63} = 58$	Node 4 with $d_{64} = 12$	Node 4	(1, 6, 4) with D = 24
4	Node 5 with $d_{45} = 38$	Node 8 with $d_{48} = 41$	Node 5	(1, 6, 4, 5) with D = 62
5	Node 7 with $d_{57} = 18$	Node 10 with $d_{5,10} = 85$	Node 7	(1, 6, 4, 5, 7) with D = 80
7	Node 8 with $d_{78} = 88$	Node 9 with $d_{79} = 21$	Node 9	(1, 6, 4, 5, 7, 9) with D = 101
9	Node 8 with $d_{98} = 77$	Node 3 with $d_{93} = 63$	Node 3	(1, 6, 4, 5, 7, 9, 3) with D = 164
3	Node 8 with $d_{38} = 48$	Node 2 with $d_{32} = 100$	Node 8	(1, 6, 4, 5, 7, 9, 3, 8) with D = 212
8	Node 2 with $d_{82} = 65$	Node 10 with $d_{8,10} = 38$	Node 10	(1, 6, 4, 5, 7, 9, 3, 8, 10) with D = 250
10	Node 2 with $d_{10,2} = 32$	Node 2 with $d_{10,2} = 32$	Node 2	(1, 6, 4, 5, 7, 9, 3, 8, 10, 2) with D = 282
		Completed offspring (1, 6, 4, 5, 7, 9, 3, 8, 10, 2) with distance = 328		

3.4 Reverse Greedy Sequential Constructive Crossover Operator

Ahmed [14] proposed another crossover named reverse greedy SCX (RGSCX) by applying the GSCX in the reverse direction that produces an offspring in the reverse direction, that is, from the last node (gene) of the offspring back to the first node (gene) of the same. Algorithm 4 shows the pseudocode of the RGSCX operator for the TSP.

Algorithm 4: Reverse greedy sequential constructive crossover algorithm

Input: n, D, P_c, Pair of parent chromosomes.
Output: Offspring chromosome.
Generate random number r ∈ [0,1].
if ($r \leq P_c$) then **do**
Suppose the 'node α' and the 'node β' are the last nodes in the 1st and 2nd parent respectively. Then for selecting the last node, we check whether $d_{\alpha 1} < d_{\beta 1}$. If yes, then select 'node α', otherwise, 'node β' as the last node of the partially constructed offspring chromosome. Then rename this present node as 'node p'.
for $i = n-1$ **down to** 2 **do**
In each parent chromosome consider the first un-visited' node found before 'node p'.
if no 'un-visited' node is found in a parent, **then**
Consider the closest 'un-visited' node from the group of remaining 'un-visited' nodes and concatenate it to the partially constructed offspring chromosome'. Rename the present node as 'node p' and continue to the next iteration.
end if
Suppose 'node α' and 'node β' are selected in 1st and 2nd chromosomes respectively.
if ($d_{\alpha p} < d_{\beta p}$) then **do**
Add 'node α' to the offspring chromosome.
else
Add 'node β' to the offspring chromosome.
end if
Rename the present node as 'node p' and continue.
end for
end if
Return the offspring chromosome

Applying the RGSCX on the parent chromosomes, one can obtain the offspring O: (1, 2, 10, 8, 3, 5, 7, 9, 4, 6, 1) with the distance 330 (see Table 5).

Table 5: Offspring chromosome by RGSCX

Present 'node p'	Legitimate node before 'node p' in		Accepted node	Partial offspring with distance D
	P1 with distance	P2 with distance		
1	Node 10 with $d_{10,1} = 35$	Node 6 with $d_{61} = 12$	Node 6	(1, ∗, ∗, ∗, ∗, ∗, ∗, ∗, ∗, 6, 1) with D = 12
6	Node 10 with $d_{10,6} = 42$	Node 4 with $d_{46} = 12$	Node 4	(1, ∗, ∗, ∗, ∗, ∗, ∗, ∗, 4, 6, 1) with D = 24
4	Node 9 with $d_{94} = 38$	Node 2 with $d_{24} = 56$	Node 9	(1, ∗, ∗, ∗, ∗, ∗, ∗, 9, 4, 6, 1) with D = 62
9	Node 3 with $d_{39} = 63$	Node 7 with $d_{79} = 21$	Node 7	(1, ∗, ∗, ∗, ∗, ∗, 7, 9, 4, 6, 1) with D = 83
7	Node 5 with $d_{57} = 18$	Node 5 with $d_{57} = 18$	Node 5	(1, ∗, ∗, ∗, ∗, 5, 7, 9, 4, 6, 1) with D = 101
5	Node 3 with $d_{35} = 65$	Node 8 with $d_{85} = 78$	Node 3	(1, ∗, ∗, ∗, 3, 5, 7, 9, 4, 6, 1) with D = 166
3	Node 8 with $d_{83} = 48$	Node 8 with $d_{83} = 48$	Node 8	(1, ∗, ∗, 8, 3, 5, 7, 9, 4, 6, 1) with D = 214
8	Node 10 with $d_{10,8} = 38$	Node 2 with $d_{28} = 65$	Node 10	(1, ∗, 10, 8, 3, 5, 7, 9, 4, 6, 1) with D = 252
10	Node 2 with $d_{2,10} = 32$	Node 2 with $d_{2,10} = 32$	Node 2	(1, 2, 10, 8, 3, 5, 7, 9, 4, 6, 1) with D = 284
		Completed offspring (1, 2, 10, 8, 3, 5, 7, 9, 4, 6, 1) with distance = 330		

3.5 Comprehensive Sequential Constructive Crossover Operators

Ahmed [14] proposed a comprehensive SCX (CSCX) by combining GSCX and RGSCX that produces two offspring. Comparative studies among six different crossover operators show that the CSCX is the best one. We have considered a total of three comprehensive SCX operators for our study.

3.5.1 CSCX1

For a pair of parent chromosomes, the first offspring is generated using SCX and the second using RGSCX. Applying the CSCX1 to the parent chromosomes, one can obtain the following offspring:

O1: (1, 6, 7, 9, 4, 5, 8, 10, 3, 2) with the distance 502.

O2: (1, 2, 10, 8, 3, 5, 7, 9, 4, 6) with the distance 330.

3.5.2 CSCX2

For a pair of parent chromosomes, the first offspring is generated using GSCX and the second using RGSCX. Applying the CSCX2 to the parent chromosomes, one can obtain the following offspring:

O1: (1, 6, 4, 5, 7, 9, 3, 8, 10, 2) with the distance 328

O2: (1, 2, 10, 8, 3, 5, 7, 9, 4, 6) with the distance 330.

3.5.3 CSCX3

For a pair of parent chromosomes, the first offspring is generated using ASCX and the second using RGSCX. Applying the CSCX3 to the parent chromosomes, one can obtain the following offspring:

O1: (1, 6, 7, 9, 4, 5, 8, 10, 3, 2) with the distance 406.

O2: (1, 2, 10, 8, 3, 5, 7, 9, 4, 6) with the distance 330.

4 Mutation Operators for Our GAs

In GAs, diversity in the population is increased by introducing random variation in the population which is done in the mutation process. In this process, a gene in a chromosome is selected randomly and the corresponding gene is changed, thus, the information is modified. The following six mutation operators are used for our comparative study. Each operator is applied under the mutation probability, P_m, rule. We discuss all mutation operators through the chromosome P: (1, 6, 7, 9, 4, 5, 8, 10, 3, 2).

4.1 Exchange Mutation

In the exchange mutation (EXCH) process, two positions are selected randomly, and then the genes on these positions are exchanged [25]. For example, if positions 3 and 7 are selected randomly, genes 7 and 10 are exchanged with their positions, then the mutated chromosome will be O: (1, 6, **10**, 9, 4, 5, 8, **7**, 3, 2).

4.2 3-Exchange Mutation

In the 3-exchange mutation (3-EXCH) process, three different positions are randomly selected, say, r_1, r_2, and r_3, then the genes on these positions are exchanged as follows: $P(r_1) \leftrightarrow P(r_2)$ and then $P(r_2) \leftrightarrow P(r_3)$ [35]. For example, if positions 2, 6, and 9 are randomly selected, then genes 6 and 5

are exchanged that leads to the chromosome (1, **5**, 7, 9, 4, **6**, 8, 10, 3, 2), and then genes 6 and 3 are exchanged that leads to the mutated chromosome as O: (1, **5**, 7, 9, 4, **3**, 8, 10, **6**, 2).

4.3 Displacement Mutation

In the displacement mutation (DISP) process, a subchromosome is selected randomly and inserted at any random position outside the subchromosome [27]. For example, if the subchromosome (6, 7, 9, 4, 5) is selected randomly and the position between 8 and 9 is selected randomly for insertion, then the mutated chromosome will be O: (1, 8, 10, **6, 7, 9, 4, 5**, 3, 2).

4.4 Insertion Mutation

In the insertion mutation (INST) process, a gene is chosen randomly and is inserted at any random location in the chromosome [15]. For example, if gene 3 is selected randomly and the position between 4 and 5 is selected randomly, then the mutated chromosome will be O: (1, 6, 7, 9, **3**, 4, 5, 8, 10, 2).

4.5 Inversion Mutation

In the inversion mutation (INVS) process, two positions are selected randomly, and the genes between these positions are inverted [26]. For example, if two positions 4 and 8 are selected randomly, then the subchromosome (9, 4, 5, 8, 10) is inverted leading to the mutated chromosome as O: (1, 6, 7, **10, 8, 5, 4, 9**, 3, 2).

4.6 Adaptive Mutation

In the adaptive mutation (ADAP) process, data from all chromosomes of the current population are gathered to detect a pattern among them. If the mutation is to be performed, chromosomes that do not like the pattern will be muted. The algorithm is as follows [30]:

Step 1: Consider all chromosomes in the current population.

Step 2: Construct a one-dimensional array of size n (size of the problem), suppose A, by storing a gene that appears the minimum number of times in the current position (except the 1st position) of all chromosomes.

Step 3: If the mutation is allowed, select two genes randomly such that they are not the same in the corresponding positions of array A, and swap them.

For example, suppose A = [1, 5, 2, 3, 6, 2, 5, 7, 10, 8] is the array and 4th and 8th positions are selected randomly. The 4th position's gene 9 and the 8th position's gene 10 do not match the array elements in the corresponding positions, so they are exchanged. So, the mutated chromosome will be O: (1, 6, 7, 10, 4, 5, 8, 9, 3, 2).

Our GA is a simple, non-hybrid that uses traditional GA operators and processes. In our GA, initiating with a random chromosome population, the better chromosomes are chosen by the stochastic remainder selection method, and then the population passes through the selected one crossover operator and one mutation operator. Our simple GA may be designed as follows:

SimpleGA()

{ Initialize a random population of size P_s.

Evaluate the population.

Generation = 0.

```
While the stopping condition is not satisfied
{ Generation = Generation + 1.
    Select fitter chromosomes by selection operator.
    Select a crossover operator and do a crossover with crossover probability P_c.
    Select a mutation operator and do a mutation with mutation probability P_m.
    Evaluate the population.
}
}
```

5 Computational Experiments

We have encoded different simple GAs using various crossover and mutation processes in Visual C++ and executed some TSPLIB instances [13] on a Laptop with Intel(R) Core(TM) i7-1065G7 CPU @ 1.30 GHz and 8.00 GB RAM under MS Windows 11. For each instance, the experiments have been done 50 times. We measured the solution quality by the percentage of excess (EX(%)) of the obtained best solution (OBS) from the best-known solution (BKS) informed on the TSPLIB website, using the formula: EX (%) = 100 × (OBS/BKS-1). We report the best solution (BS), average solution (AS), percentage of excess of the best solution (BEX(%)), and percentage of excess of average solution (AEX(%)) over the BKS among 50 executions, standard deviation (SD) of the solutions in 50 executions, average computational time (AT) (in seconds) to find the best solution for the first time for each instance using each algorithm. To avoid preference for any distinct algorithm, the same randomly generated population is used for executing the proposed algorithms. Notice that GAs are controlled by parameters-population size (P_s), crossover probability (P_c), mutation probability (P_m), and termination condition.

We set $P_s = 50$, $P_c = 1.0$ (i.e., 100%) as the crossover probability to see the real behavior of the crossover operators and almost the same computational time as a termination criterion. First, we see the functioning of the crossover operators on fourteen asymmetric TSPLIB instances. Table 6 shows the comparative study amongst GAs using seven crossover operators without using any mutation operator. In this table, the first column reports an instance name and its BKS within brackets. The second column reports the data names and the remaining columns report the corresponding results by different crossover operators mentioned in the first row. Furthermore, the boldfaces in the results indicate that the result by a particular algorithm is the best one amongst all algorithms for a particular instance.

Table 6: Comparison of GAs using various crossover operators without any mutation operator for some antisymmetric TSPLIB instances

Instance	Results	SCX	ASCX	GSCX	RGSCX	CSCX1	CSCX2	CSCX3
ftv33	BS	1472	1378	1359	1412	1345	1362	1378
(1286)	AS	1608.20	1412.68	1479.70	1535.18	1435.88	1411.02	1400.66
	BEX (%)	14.46	7.15	5.68	9.80	**4.59**	5.91	7.15
	AEX (%)	25.05	9.85	15.06	19.38	11.65	9.72	**8.92**

(Continued)

Table 6 (continued)

Instance	Results	SCX	ASCX	GSCX	RGSCX	CSCX1	CSCX2	CSCX3
	SD	77.26	44.42	58.52	61.33	46.69	28.33	19.72
	AT	0.00	0.00	0.01	0.01	0.01	0.01	0.01
ftv35	BS	1626	1594	1536	1606	1515	1515	1575
(1473)	AS	1779.32	1697.26	1657.20	1758.90	1623.22	1602.50	1704.70
	BEX (%)	10.39	8.21	4.28	9.03	**2.85**	**2.85**	6.92
	AEX (%)	20.80	15.22	12.51	19.41	10.20	**8.79**	15.73
	SD	100.67	48.59	56.32	79.76	52.65	53.27	51.22
	AT	0.01	0.01	0.01	0.00	0.01	0.01	0.01
ftv38	BS	1695	1697	1639	1693	1584	1560	1701
(1530)	AS	1872.80	1777.50	1735.66	1845.96	1709.74	1665.64	1785.62
	BEX (%)	10.78	10.92	7.12	10.65	3.53	**1.96**	11.18
	AEX (%)	22.41	16.18	13.44	20.65	11.75	**8.87**	16.71
	SD	107.61	32.79	65.95	73.21	65.86	42.27	24.84
	AT	0.01	0.01	0.01	0.01	0.01	0.01	0.02
p43	BS	5660	5645	5642	5627	5627	5629	5637
(5620)	AS	5699.96	5652.68	5699.26	5687.80	5650.42	5656.30	5647.30
	BEX (%)	0.71	0.44	0.39	**0.12**	**0.12**	0.16	0.30
	AEX (%)	1.42	0.58	1.41	1.21	0.54	0.65	**0.49**
	SD	27.49	5.89	55.32	50.28	13.28	12.00	3.31
	AT	0.01	0.01	0.01	0.01	0.01	0.01	0.02
ftv44	BS	1911	1702	1773	1761	1613	1613	1647
(1613)	AS	2084.50	1868.82	1919.80	1953.20	1778.84	1745.14	1858.68
	BEX (%)	18.47	5.52	9.92	9.18	**0.00**	**0.00**	2.11
	AEX (%)	29.23	15.86	19.02	21.09	10.28	**8.19**	15.23
	SD	103.22	55.83	67.90	100.50	62.63	69.42	58.44
	AT	0.01	0.01	0.01	0.01	0.01	0.01	0.01
ftv47	BS	2091	2032	2002	2079	1856	1867	2042
(1776)	AS	2389.26	2150.94	2193.00	2223.44	2049.60	2019.20	2154.16
	BEX (%)	17.74	14.41	12.73	17.06	**4.50**	5.12	14.98
	AEX (%)	34.53	21.11	23.48	25.19	15.41	**13.69**	21.29
	SD	133.18	68.97	92.19	84.16	72.38	67.49	46.82
	AT	0.00	0.01	0.00	0.01	0.01	0.01	0.01
ry48p	BS	16554	15308	15868	15294	14877	15164	15176
(14422)	AS	17829.18	15702.24	16631.86	16154.42	15688.90	15728.48	15696.58
	BEX (%)	14.78	6.14	10.03	6.05	**3.15**	5.14	5.23
	AEX (%)	23.62	8.88	15.32	12.01	**8.78**	9.06	8.84
	SD	835.66	318.98	461.11	508.60	276.07	285.06	296.04
	AT	0.00	0.01	0.01	0.01	0.02	0.02	0.01
ft53	BS	8578	8188	8379	8186	7718	7776	8497
(6905)	AS	10206.82	8826.44	9522.54	9461.52	8277.22	8355.32	9040.86
	BEX (%)	24.23	18.58	21.35	18.55	**11.77**	12.61	23.06

(Continued)

Table 6 (continued)

Instance	Results	SCX	ASCX	GSCX	RGSCX	CSCX1	CSCX2	CSCX3
	AEX (%)	47.82	27.83	37.91	37.02	**19.87**	21.00	30.93
	SD	768.09	239.55	698.79	440.60	307.68	280.81	279.28
	AT	0.01	0.01	0.01	0.01	0.01	0.01	0.01
ftv55	BS	1903	1770	1746	1768	1694	1691	1732
(1608)	AS	2150.04	1848.64	1957.40	1903.26	1792.88	1782.66	1819.18
	BEX (%)	18.35	10.07	8.58	9.95	5.35	**5.16**	7.71
	AEX (%)	33.71	14.97	21.73	18.36	11.50	**10.86**	13.13
	SD	127.59	50.34	92.73	99.11	57.24	59.07	34.07
	AT	0.01	0.01	0.01	0.01	0.01	0.01	0.02
ftv64	BS	2281	2043	2097	1933	1890	1899	2169
(1839)	AS	2592.24	2288.10	2288.26	2350.82	2102.16	2026.88	2287.78
	BEX (%)	24.03	11.09	14.03	5.11	**2.77**	3.26	17.94
	AEX (%)	40.96	24.42	24.43	27.83	14.31	**10.22**	24.40
	SD	176.58	80.11	98.34	144.51	94.14	78.94	70.14
	AT	0.02	0.03	0.02	0.02	0.03	0.02	0.06
ft70	BS	42062	41966	41598	42100	40486	40054	41636
(38673)	AS	44689.54	43184.36	43323.32	44348.98	41834.48	41651.24	42953.32
	BEX (%)	8.76	8.51	7.56	8.86	4.69	**3.57**	7.66
	AEX (%)	15.56	11.67	12.02	14.68	8.17	**7.70**	11.07
	SD	1192.44	601.91	932.88	870.70	583.34	518.87	594.43
	AT	0.02	0.03	0.01	0.01	0.02	0.02	0.03
ftv70	BS	2414	2163	2209	2261	2005	2033	2233
(1950)	AS	2776.58	2382.94	2398.32	2493.64	2228.44	2165.70	2393.82
	BEX (%)	23.79	10.92	13.28	15.95	**2.82**	4.26	14.51
	AEX (%)	42.39	22.20	22.99	27.88	14.28	**11.06**	22.76
	SD	193.48	82.73	114.10	113.90	151.74	74.61	74.40
	AT	0.03	0.05	0.03	0.03	0.05	0.04	0.09
kro124p	BS	44329	40771	41661	40694	39205	39257	40472
(36230)	AS	49789.36	42156.86	43816.00	44234.68	41934.00	41275.08	42270.92
	BEX (%)	22.35	12.53	14.99	12.32	**8.21**	8.35	11.71
	AEX (%)	37.43	16.36	20.94	22.09	15.74	**13.93**	16.67
	SD	2382.39	598.17	1147.27	1460.59	1229.21	927.58	692.57
	AT	0.03	0.07	0.06	0.08	0.10	0.09	0.10
ftv170	BS	4457	3343	3637	3604	3268	3170	3308
(2755)	AS	5149.26	3562.72	4115.64	4125.86	3631.02	3466.18	3511.44
	BEX (%)	61.78	21.34	32.01	30.82	18.62	**15.06**	20.07
	AEX (%)	86.91	29.32	49.39	49.76	31.80	**25.81**	27.46
	SD	379.77	99.72	246.02	305.92	196.85	149.82	90.02
	AT	0.09	0.13	0.09	0.08	0.17	0.14	0.14

Looking at the boldfaces in Table 6, the comprehensive crossover operators are observed better than the other crossover operators. Operator CSCX1 could find the best solution for ten instances–ftv33, ftv35, p43, ftv44, ftv47, ry48p, ft53, ftv64, ftv70 and kro124p, whereas CSCX2 could find for six instances–ftv35, ftv38, ftv44, ftv55, ft70 and ftv170. Furthermore, CSCX1 could find the best average solution for two instances–ry48p and ft53; whereas CSCX2 could find for ten instances–ftv35, ftv38, ftv44, ftv47, ftv55, ftv64, ft70, ftv70, kro124p and ftv170; and CSCX3 for two instances–ftv33 and p43. It is seen that two operators–CSCX1 and CSCX2 are competing. However, CSCX2 is observed better than CSCX1, and CSCX1 is found to be better than CSCX3. So, CSCX2 is the best crossover operator among all crossover operators without using any mutation operator. Among other crossover operators, ASCX, GSCX, and RGSCX could find the best solutions for three, six, and five instances, respectively. Further, ASCX, GSCX, and GSCX could find the best average solution for twelve, two, and zero instances, respectively. Two operators–ASCX and GSCX are competing. However, looking at the average solutions and standard deviations, we can conclude that ASCX is better than GSCX. Looking at the average computational times by the GAs using these crossover operators, almost all algorithms are taking almost the same time.

Accordingly, CSCX2 produces the best results, while CSCX1 is the second best, ASCX and CSCX3 are competing for the third best, and SCX is the worst one. However, it is confirmed that ASCX, GSCX, and RSCX are the improvements of SCX. The results are also depicted in Fig. 1, which also demonstrates the usefulness of CSCX2 and CSCX1. Further, Fig. 1 shows that for most of the instances, SCX finds the worst solution quality, and other algorithms find improved solutions over the solutions by SCX. The operators ASCX and CSCX3 are competing for the third position.

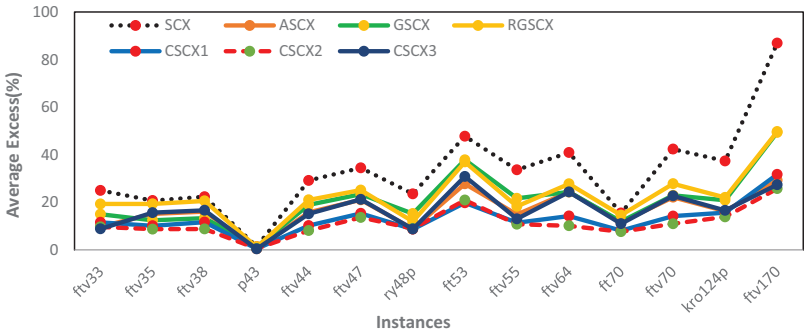

Figure 1: Average Excess (%) by different crossovers for asymmetric instances

Furthermore, to justify the above observations, the two-tailed Student *t*-test was conducted with a 5% significant level between CSCX2 and other crossover operators for the instances, and the results are summarized in Table 7. As expected, CSCX2 is the best-ranked crossover and CSCX1 is the second-best. Between CSCX3 and ASCX, there is no significant difference, as expected, they are in the third rank.

We now implement different mutation operators with the probability of mutation as 0.10 (10%) without any crossover operator on the same TSPLIB instances. Table 8 shows the comparative study among GAs using six mutation operators without any crossover operator.

Table 7: Results of *t*-test on antisymmetric TSPLIB instances

Crossover	Inferior crossovers
CSCX2	CSCX1, CSCX3, ASCX, GSCX, RGSCX, SCX
CSCX1	CSCX3, ASCX, GSCX, RGSCX, SCX
CSCX3	GSCX, RGSCX, SCX
ASCX	GSCX, RGSCX, SCX
GSCX	RGSCX, SCX
RGSCX	SCX

Table 8: Comparison of GAs using various mutation operators without any crossover operator for some antisymmetric TSPLIB instances

Instance	Results	EXCH	3-EXCH	DISP	INST	INVS	ADAP
ftv33	BS	1640	1988	1643	1455	1631	2102
(1286)	AS	1904.46	2161.08	1832.20	1650.82	1764.80	2341.36
	BEX (%)	27.53	54.59	27.76	**13.14**	26.83	63.45
	AEX (%)	48.09	68.05	42.47	**28.37**	37.23	82.07
	SD	90.01	83.01	90.15	73.51	60.18	74.01
	AT	0.03	0.03	0.04	0.04	0.03	0.03
ftv35	BS	2024	2278	1977	1816	1887	2537
(1473)	AS	2332.06	2589.66	2227.98	2007.08	2161.72	2673.48
	BEX (%)	37.41	54.65	34.22	**23.29**	28.11	72.23
	AEX (%)	58.32	75.81	51.25	**36.26**	46.76	81.50
	SD	113.05	98.95	83.63	89.07	67.48	65.05
	AT	0.02	0.02	0.03	0.02	0.02	0.02
ftv38	BS	2307	2621	2201	1982	2097	2729
(1530)	AS	2539.66	2840.20	2426.14	2178.80	2346.98	2910.18
	BEX (%)	50.78	71.31	43.86	**29.54**	37.06	78.37
	AEX (%)	65.99	85.63	58.57	**42.41**	53.40	90.21
	SD	91.50	93.02	85.22	74.7	81.35	67.61
	AT	0.02	0.02	0.03	0.02	0.02	0.02
p43	BS	5932	5938	5912	5896	5847	5906
(5620)	AS	5985.66	5999.68	6017.88	5955.14	5943.92	5993.66
	BEX (%)	5.55	5.66	5.20	4.91	**4.04**	5.09
	AEX (%)	6.51	6.76	7.08	5.96	**5.76**	6.65
	SD	23.66	28.36	48.82	30.33	34.05	27.3
	AT	0.02	0.02	0.02	0.02	0.02	0.02
ftv44	BS	2883	3415	2986	2730	2770	3326
(1613)	AS	3294.16	3624.12	3164.00	2892.98	2985.84	3503.22
	BEX (%)	78.74	111.72	85.12	**69.25**	71.73	106.20
	AEX (%)	104.23	124.68	96.16	**79.35**	85.11	117.19

(Continued)

Table 8 (continued)

Instance	Results	EXCH	3-EXCH	DISP	INST	INVS	ADAP
	SD	124.39	108.05	82.13	85.36	75.41	73.20
	AT	0.02	0.03	0.03	0.03	0.03	0.03
ftv47	BS	3487	3829	3343	2920	3226	3604
(1776)	AS	3724.02	4080.52	3553.12	3228.52	3419.64	3932.98
	BEX (%)	96.34	115.60	88.23	**64.41**	81.64	102.93
	AEX (%)	109.69	129.76	100.06	**81.79**	92.55	121.45
	SD	101.02	112.76	100.21	101.26	83.85	86.40
	AT	0.03	0.02	0.04	0.03	0.03	0.03
ry48p	BS	24146	26793	25101	21408	21004	25226
(14422)	AS	26409.04	28929.30	26227.22	23551.38	23375.90	27719.80
	BEX (%)	67.42	85.78	74.05	48.44	**45.64**	74.91
	AEX (%)	83.12	100.59	81.86	63.30	**62.09**	92.20
	SD	958.36	931.11	662.94	843.20	673.27	738.48
	AT	0.03	0.03	0.03	0.04	0.03	0.03
ft53	BS	13629	14692	13005	11760	12309	13964
(6905)	AS	14553.34	15673.16	13975.50	12853.78	13312.40	14794.78
	BEX (%)	97.38	112.77	88.34	**70.31**	78.26	102.23
	AEX (%)	110.77	126.98	102.40	**86.15**	92.79	114.26
	SD	361.25	370.59	381.64	478.29	375.76	328.91
	AT	0.03	0.03	0.04	0.04	0.03	0.04
ftv55	BS	4016	4432	3803	3138	3480	3931
(1608)	AS	4232.66	4624.50	4022.02	3705.64	3788.26	4242.64
	BEX (%)	149.75	175.62	136.50	**95.15**	116.42	144.47
	AEX (%)	163.23	187.59	150.13	**130.45**	135.59	163.85
	SD	105.99	87.70	90.45	144.68	90.65	109.52
	AT	0.03	0.03	0.05	0.04	0.03	0.04
ftv64	BS	5077	5481	4774	4589	4506	4954
(1839)	AS	5410.1	5816.48	5198.08	4867.76	4898.00	5208.36
	BEX (%)	176.07	198.04	159.60	149.54	**145.02**	169.39
	AEX (%)	194.19	216.28	182.66	**164.70**	166.34	183.22
	SD	106.83	110.53	104.32	119.72	129.43	94.78
	AT	0.03	0.03	0.04	0.04	0.04	0.05
ft70	BS	55299	57613	54493	53105	53563	53530
(38673)	AS	56449.24	58471.60	55922.24	54444.74	54927.62	55105.24
	BEX (%)	42.99	48.97	40.91	**37.32**	38.50	38.42
	AEX (%)	45.97	51.19	44.60	**40.78**	42.03	42.49
	SD	637.71	463.18	432.45	493.46	493.04	575.33
	AT	0.04	0.04	0.05	0.05	0.04	0.06
ftv70	BS	5893	6277	5474	5283	5396	5564
(1950)	AS	6183.82	6593.22	5917.32	5567.48	5629.42	5851.40
	BEX (%)	202.21	221.90	180.72	**170.92**	176.72	185.33

(Continued)

Table 8 (continued)

Instance	Results	EXCH	3-EXCH	DISP	INST	INVS	ADAP
	AEX (%)	217.12	238.11	203.45	**185.51**	188.69	200.07
	SD	130.99	114.43	124.48	117.14	110.93	115.85
	AT	0.03	0.03	0.05	0.04	0.04	0.04
kro124p	BS	119031	125155	117060	108655	105305	104426
(36230)	AS	124887.30	132340.80	121685.40	114223.30	112477.60	111656.30
	BEX (%)	228.54	245.45	223.10	199.90	190.66	**188.23**
	AEX (%)	244.71	265.28	235.87	215.27	210.45	**208.19**
	SD	2336.44	2639.20	1930.04	2035.75	2017.32	2005.09
	AT	0.05	0.04	0.07	0.06	0.06	0.07
ftv170	BS	19406	20309	18924	18278	17877	15902
(2755)	AS	19839.92	20617.48	19343.58	18791.04	18455.06	16511.78
	BEX (%)	604.39	637.17	586.90	563.45	548.89	**477.21**
	AEX (%)	620.14	648.37	602.13	582.07	569.88	**499.34**
	SD	192.21	167.63	157.24	194.85	228.75	240.48
	AT	0.12	0.13	0.22	0.2	0.15	0.16

Looking at the boldfaces in Table 8, the mutation operators EXCH, 3-EXCH, and DISP could not find the best solution for any instance. The mutation operator could find the best solution for nine instances–ftv33, ftv35, ftv38, ftv44, ftv47, ft53, ftv55, ft70, and ftv70; whereas INVS could find it for three instances–p43, ry48p, and ftv64; and ADAP could find for two instances kro124p and ftv170. Furthermore, EXCH, 3-EXCH, and DISP could not find the best average solution for any instance, whereas INST could find for ten instances–ftv33, ftv35, ftv38, ftv44, ftv47, ft53, ftv55, ftv64, ft70, and ftv70; INVS could find for two instances–p43 and ry48p; and ADAP could find for two instances–kro124p and ftv170. It is clear that the INST is the best one, and the two operators–INVS and ADAP are competing for the second best. The operators EXCH, 3-EXCH, and DISP could neither find the best solution nor find the best average solution for any instance, hence, they are not good enough for the TSP in comparison to the other three operators. Looking at the average computational times by the GAs using these mutation operators, almost all algorithms are taking almost the same time.

The results are also depicted in Fig. 2, which also demonstrates the usefulness of INST, INVS, DISP, and ADAP mutation operators. It is very clear from the figure that INST mutation is the best and INVS is the second best for the problem instances of size less than 100, and ADAP is the best, and INVS and INST are competing for the second best, for the problem instances of size more than or equal to 100.

So, we have conducted the two-tailed Student t-test with a 5% significant level between INST and other mutation operators for the instances. However, the results are not reported here. As expected, INST is the best-ranked mutation and INVS is the second best. However, DISP is placed in third place and ADAP is placed in fourth place.

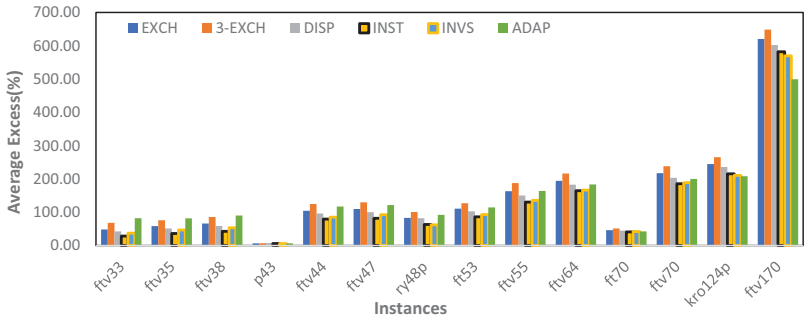

Figure 2: Average Excess (%) by different mutations for asymmetric instances

We now implement different mutation operators with the crossover operators using the common mutation probability. However, it is not easy to fix mutation probability for all mutation operators for all instances. After doing extensive experiments, we fix $P_m = 0.10$ and a maximum of 2000 generations for the termination condition. Table 9 shows the comparative study among simple GAs using all seven crossover operators without any mutation operator and with six mutation operators on fourteen asymmetric TSPLIB instances. In the first column, we report the instance name, and its best-known solution value within the brackets informed on the TSPLIB website. The second column reports the name of the mutation operator, the second last column reports the grand average of the average excess (%) (GAVG) using a particular mutation operator and all crossover operators, and the last column reports the average percentage of improvement (IMPV) by GAs using a particular mutation operator over GAs without any mutation operator. The remaining columns report the average excess (%) by a crossover operator without using any mutation and using all six mutation operators as mentioned therein.

Table 9: Comparison of the crossover and mutation operators for some TSPLIB instances

Instance	Mutation	SCX	ASCX	GSCX	RGSCX	CSCX1	CSCX2	CSCX3	GAVG	IMPV
ftv33	No Mute	25.05	9.85	15.06	19.38	11.65	9.72	8.92	14.23	—-
(1286)	EXCH	15.20	8.50	13.88	12.54	6.44	6.15	8.36	10.15	28.65
	3-EXCH	16.36	8.53	11.65	12.27	6.63	6.07	8.35	9.98	29.87
	DISP	14.29	8.42	8.40	13.95	6.94	6.43	8.40	9.55	32.91
	INST	15.83	8.40	7.96	7.61	**5.61**[a]	5.92[b]	7.58	**8.41**	**40.87**
	INVS	16.18	8.48	13.21	12.43	6.91	6.64	8.37	10.31	27.51
	ADAP	15.75	8.44	13.55	11.00	6.42	6.45	8.35	9.99	29.77
	PAVG	16.95	8.66	11.96	12.74	7.23	**6.77**	8.33	—-	—-
ftv35	No Mute	20.80	15.22	12.51	19.41	10.20	8.79	15.73	14.67	—-
(1473)	EXCH	11.64	11.00	10.81	10.92	2.66	3.07	9.45	8.51	42.01
	3-EXCH	11.86	11.50	10.49	11.76	3.43	3.13	10.49	8.95	38.98
	DISP	10.37	8.77	8.71	9.89	4.17	3.11	8.03	7.58	48.36
	INST	9.89	8.85	9.05	9.50	**2.43**[a]	2.75	7.91	**7.20**	**50.94**
	INVS	12.00	11.55	10.84	10.32	2.66	2.67[b]	9.47	8.50	42.05
	ADAP	10.29	9.17	8.70	10.24	2.65	3.35	8.98	7.63	48.01
	PAVG	12.41	10.87	10.16	11.72	4.03	**3.84**	10.01	—-	—-

(Continued)

Table 9 (continued)

Instance	Mutation	SCX	ASCX	GSCX	RGSCX	CSCX1	CSCX2	CSCX3	GAVG	IMPV
ftv38	No Mute	22.41	16.18	13.44	20.65	11.75	8.87	16.71	15.72	—-
(1530)	EXCH	10.35	9.75	8.64	9.22	5.57	3.63	8.70	7.98	49.23
	3-EXCH	11.45	11.05	11.01	11.32	6.32	3.85	8.93	9.13	41.90
	DISP	10.43	9.41	8.95	9.48	5.96	3.38	8.51	8.02	49.00
	INST	10.14	8.89	8.09	9.95	4.57	**2.81**[b]	8.75	7.60	51.65
	INVS	10.83	10.29	8.90	9.92	4.17	4.20	9.23	8.22	47.71
	ADAP	10.00	7.86	9.00	9.45	3.70[a]	4.08	8.27	**7.48**	**52.41**
	PAVG	12.23	10.49	9.72	11.43	6.01	**4.40**	9.87	—-	—-
p43	No Mute	1.42	0.58	1.41	1.21	0.54	0.65	0.49	0.90	—-
(5620)	EXCH	0.67	0.28	0.57	0.40	**0.20**[a]	0.34	0.25	0.39	56.98
	3-EXCH	0.60	0.21	0.37	0.22	**0.20**[a]	0.37	0.23	**0.31**	**65.08**
	DISP	0.63	0.31	0.50	0.44	0.23	0.31	0.25	0.38	57.62
	INST	0.66	0.28	0.57	0.42	0.22	0.33	0.26	0.39	56.51
	INVS	0.40	0.36	0.51	0.39	0.24	0.30[b]	0.26	0.35	60.95
	ADAP	0.74	0.27	0.67	0.52	0.35	0.53	0.26	0.48	46.98
	PAVG	0.73	0.33	0.66	0.51	**0.28**	0.40	0.29	—-	—-
ftv44	No Mute	29.23	15.86	19.02	21.09	10.28	8.19	15.23	16.99	—-
(1613)	EXCH	14.45	11.74	13.83	9.50	3.48	1.22	11.76	9.43	44.52
	3-EXCH	15.31	11.97	14.88	9.92	2.00	**0.74**[b]	6.68	**8.79**	**48.29**
	DISP	16.35	12.26	14.40	12.17	3.08	1.45	11.54	10.18	40.10
	INST	16.21	13.57	14.59	8.80	3.87	1.19	11.67	9.98	41.24
	INVS	15.13	13.84	14.40	8.79	0.78[a]	0.91	9.05	8.99	47.11
	ADAP	13.32	11.65	13.09	9.34	3.81	1.91	13.01	9.45	44.39
	PAVG	17.14	12.98	14.89	11.37	3.90	**2.23**	11.28	—-	—-
ftv47	No Mute	34.53	21.11	23.48	25.19	15.41	13.69	21.29	22.10	—-
(1776)	EXCH	18.19	16.98	16.74	17.75	4.93	5.14[b]	12.39	**13.16**	**40.46**
	3-EXCH	19.00	18.27	16.02	18.83	5.54	5.37	10.26	13.33	39.70
	DISP	17.76	17.29	15.60	16.95	4.49	5.44	16.49	13.43	39.22
	INST	18.51	18.46	15.51	18.85	**4.30**[a]	6.26	13.01	13.56	38.66
	INVS	19.25	18.49	16.17	19.16	6.84	7.61	16.90	14.92	32.50
	ADAP	18.43	18.16	17.90	17.90	6.07	7.30	15.57	14.48	34.50
	PAVG	20.81	18.40	17.35	19.23	**6.80**	7.26	15.13	—-	—-
ry48p	No Mute	23.62	8.88	15.32	12.01	8.78	9.06	8.84	12.36	—-
(14422)	EXCH	11.65	8.15	11.51	7.86	4.71	4.64	7.90	8.06	34.79
	3-EXCH	12.31	8.61	11.73	7.70	4.41	4.25	6.62	7.95	35.70
	DISP	11.36	8.82	11.18	7.96	4.20[a]	4.52	8.52	8.08	34.62
	INST	10.91	8.74	9.61	8.05	6.04	5.10	7.50	7.99	35.33
	INVS	10.60	8.39	10.55	7.23	4.25	**4.07**[b]	8.40	**7.64**	**38.20**
	ADAP	11.54	8.32	11.00	8.27	5.79	5.72	8.41	8.44	31.74
	PAVG	13.14	8.56	11.56	8.44	5.45	**5.34**	8.03	—-	—-

(Continued)

Table 9 (continued)

Instance	Mutation	SCX	ASCX	GSCX	RGSCX	CSCX1	CSCX2	CSCX3	GAVG	IMPV
ft53	No Mute	47.82	27.83	37.91	37.02	19.87	21.00	30.93	31.77	—-
(6905)	EXCH	23.30	21.71	17.24	22.59	11.04	12.87	16.95	17.96	43.48
	3-EXCH	24.23	17.94	18.15	17.47	11.31	13.47	18.83	**17.34**	**45.41**
	DISP	23.56	23.38	23.10	23.13	9.75	11.09[b]	23.29	19.62	38.25
	INST	23.44	23.20	23.36	21.49	**9.65**[a]	11.68	23.14	19.42	38.87
	INVS	23.63	22.60	23.03	22.69	13.32	14.54	22.72	20.36	35.91
	ADAP	23.48	22.63	22.56	22.38	15.95	16.39	20.55	20.56	35.28
	PAVG	27.07	22.76	23.62	23.82	**12.98**	14.44	22.34	—-	—-
ftv55	No Mute	33.71	14.97	21.73	18.36	11.50	10.86	13.13	17.75	—-
(1608)	EXCH	14.87	11.33	12.20	11.60	3.76	**3.43**[b]	10.81	9.71	45.28
	3-EXCH	15.16	10.89	12.24	10.52	3.94	3.63	8.35	9.25	47.91
	DISP	13.64	11.60	12.94	11.74	3.72	3.65	10.08	9.62	45.79
	INST	14.84	11.46	13.30	10.14	3.79	3.81	10.51	9.69	45.39
	INVS	13.92	12.06	12.61	10.95	4.17	4.58	9.86	9.74	45.15
	ADAP	14.18	11.59	11.96	9.05	3.61[a]	4.16	9.49	**9.15**	**48.44**
	PAVG	17.19	11.98	13.85	11.76	4.93	**4.88**	10.32	—-	—-
ftv64	No Mute	40.96	24.42	24.43	27.83	14.31	10.22	24.40	23.80	—-
(1839)	EXCH	20.12	19.51	18.83	19.19	2.96	2.84	15.03	14.07	40.89
	3-EXCH	20.23	19.09	15.86	16.62	3.33	2.87	12.16	12.88	45.88
	DISP	21.29	19.79	17.98	17.00	2.99	**2.55**[b]	14.25	13.69	42.47
	INST	20.30	19.90	14.48	14.49	2.80	2.65	14.61	**12.75**	**46.44**
	INVS	20.91	16.90	19.54	18.51	4.26	3.90	13.91	13.99	41.22
	ADAP	20.01	19.48	15.12	17.13	2.75[a]	3.22	14.59	13.19	44.59
	PAVG	23.40	19.87	18.03	18.68	4.77	**4.04**	15.57	—-	—-
ft70	No Mute	15.56	11.67	12.02	14.68	8.17	7.70	11.07	11.55	—-
(38673)	EXCH	8.16	7.57	8.11	7.76	5.32	6.11	7.56	7.23	37.43
	3-EXCH	8.24	7.66	8.56	8.08	5.06	6.10	7.53	7.32	36.64
	DISP	7.96	7.01	6.19	6.59	**4.63**[a]	5.22[b]	6.43	**6.29**	**45.53**
	INST	8.10	7.38	7.42	7.09	5.90	6.28	7.29	7.07	38.81
	INVS	8.43	7.20	7.48	7.37	6.04	6.40	7.65	7.23	37.44
	ADAP	7.28	7.15	6.34	6.11	5.08	6.14	6.66	6.39	44.65
	PAVG	9.10	7.95	8.02	8.24	**5.74**	6.28	7.74	—-	—-
ftv70	No Mute	42.39	22.20	22.99	27.88	14.28	11.06	22.76	23.37	—-
(1950)	EXCH	20.22	18.32	16.23	19.52	2.54	3.02	14.61	13.49	42.26
	3-EXCH	18.80	15.11	15.19	17.37	2.62	3.04	12.11	**12.03**	**48.51**
	DISP	20.48	19.76	15.69	16.28	2.55	2.50[b]	15.53	13.26	43.28
	INST	20.14	18.65	13.07	13.69	2.21	3.45	15.02	12.32	47.29
	INVS	17.44	17.07	14.03	16.87	3.99	5.06	13.36	12.55	46.31
	ADAP	18.65	18.19	15.01	14.84	**2.19**[a]	4.62	12.94	12.35	47.15
	PAVG	22.59	18.47	16.03	18.06	**4.34**	4.68	15.19	—-	—-

(Continued)

Table 9 (continued)

Instance	Mutation	SCX	ASCX	GSCX	RGSCX	CSCX1	CSCX2	CSCX3	GAVG	IMPV
kro124p	No Mute	37.43	16.36	20.94	22.09	15.74	13.93	16.67	20.45	—-
(36230)	EXCH	19.91	16.23	18.36	18.50	12.57	11.20	15.93	16.10	21.27
	3-EXCH	19.34	15.84	15.75	15.45	10.48	11.81	13.34	**14.57**	**28.73**
	DISP	21.50	16.56	16.67	17.53	**8.36**[a]	9.31[b]	16.33	15.18	25.77
	INST	20.32	16.17	16.32	17.54	13.39	9.67	15.94	15.62	23.61
	INVS	17.47	16.21	16.74	16.63	11.35	12.31	16.64	15.33	25.01
	ADAP	18.65	16.23	16.71	17.04	11.18	10.66	16.06	15.22	25.58
	PAVG	22.09	16.23	17.36	17.83	11.87	**11.27**	15.85	—-	—-
ftv170	No Mute	86.91	29.32	49.39	49.76	31.80	25.81	27.46	42.92	—-
(2755)	EXCH	34.37	27.23	28.00	26.11	17.70	16.73	22.01	24.59	42.70
	3-EXCH	32.83	27.12	25.33	25.00	19.44	18.98	22.18	24.41	43.12
	DISP	38.24	29.98	26.79	25.80	14.39	14.01	23.68	24.70	42.46
	INST	40.31	27.19	26.70	24.98	16.77	14.94	23.51	24.92	41.95
	INVS	48.71	28.98	34.05	31.12	30.79	25.03	25.18	31.98	25.49
	ADAP	40.45	28.06	25.90	24.17	12.61[a]	**11.81**[b]	26.78	**24.25**	**43.49**
	PAVG	45.97	28.27	30.88	29.56	20.50	**18.19**	24.40	—-	—-

By looking at the partial average of average excess (%) (PAVG) (in **boldface**), regardless of any mutation operator, in Table 9, operators SCX, ASCX, GSCX, RGSCX, and CSCX3 could not find best average for any instance, CSCX1 is found to be the best for five instances, namely, p43, ftv47, ft53, ft70 and ftv70; CSCX2 is found to be best for nine instances–ftv33, ftv35, ftv38, ftv44, ry48p, ftv55, ftv64, kro124p and ftv170. Hence, CSCX2 is observed as the best one, CSCX1 is the second best, CSCX3 is the third best, and ASCX is the fourth best. CSCX1 and CSCX2 are competing and are very close to each other. Looking at the last column, IMPV, it is confirmed that using the mutation operator in GA has a great impact on the solution quality of the problem instances. The PAVG that is shown in Table 9 is also depicted in Fig. 3. Looking at the figure, it is very clear that SCX with mutation operators is the worst one and all other crossover operators with mutation operators have some improvement over the SCX with mutation operators. Though CSCX1 and CSCX2 both with mutations are competing with each other, however, CSCX2 with mutations is the best one, CSCX1 with the mutations is the second best, and CSCX3 with mutations is the third best.

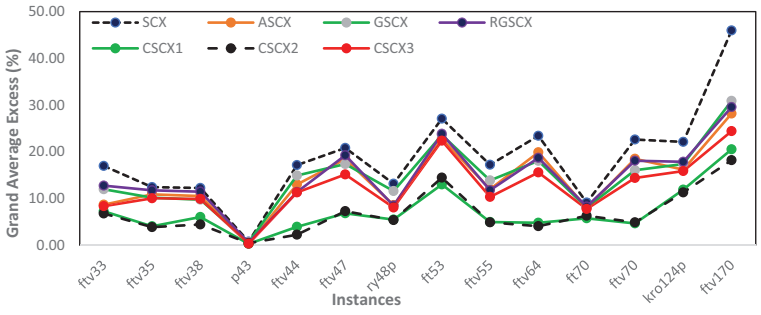

Figure 3: Comparative study of seven different crossover operators

By looking at the grand average of the average excess (%) (GAVG) (in **boldface**), regardless of any crossover operator, in Table 9, EXCH is found to be the best for only one instance–ftv47; 3-EXCH is the best for five instances–p43, ftv44, ft53, ftv64, and kro124p; DISP is best for only one instance–ft70; INST is best for three instances–ftv33, ftv35 and ftv64; INVS is found best for one instance–ry48p; and ADAP is best for three instances–ftv38, ftv55 and ftv170. Hence, it is observed that 3-EXCH is the best one, then INST and ADAP are the second best, and then EXCH, DISP, and INVS are the third best. Further, looking at the last column, which shows the average improvement using crossover operators with different mutation operators over GAs using only crossover operators without using any mutation, it confirmed our observation. It is noted that all GAs using mutation have significant improvements over GAs without any mutation operator. The GAVG that is shown in Table 9 is also depicted in Fig. 4, which further confirms our observation, and shows clear improvement of GAs with crossover and mutation operators over GAs with crossover operators and without any mutation operator.

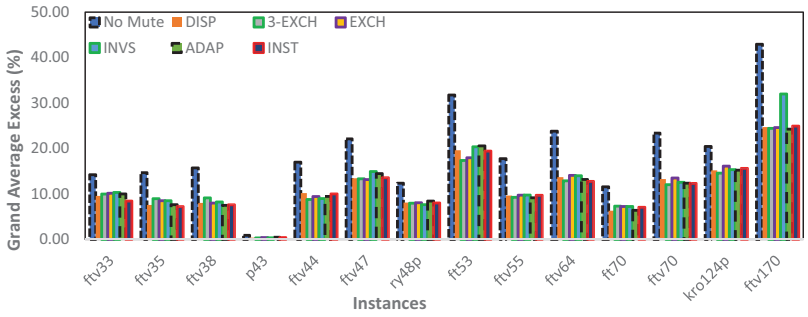

Figure 4: Comparative study of different mutation operators

Since CSCX2 with mutation operators is the best one and CSCX1 with mutation operators is the second best, we now look at the combinations of CSCX1 with any mutation and CSCX2 with any mutation, in Table 9. We mark by superscript[a] for the best average excess (%) using CSCX1 with any mutation operator for instance, and we mark by superscript[b] for the best average excess (%) using CSCX2 with any mutation operator for instance.

By looking at the average excess (%) (marked by superscript[a]), in Table 9, the combination of CSCX1 and EXCH is found to be best for only one instance–p43; the combination of CSCX1 and 3-EXCH is best for only one instance–p43; the combination of CSCX1 and DISP is best for three instances–ry48p, ft70 and kro124p; combination of CSCX1 and INST is found to be best for four instances–ftv33, ftv35, ftv47 and ft53; combination of CSCX1 and INVS is best for only one instance-ftv44; combination of CSCX1 and ADAP is best for five instances–ftv38, ftv55, ftv64, ftv70 and ftv170. Hence, the combination of CSCX1 and ADAP is the best one, and the combination of CSCX1 and INST is the second best, and the combination of CSCX1 and DISP is the third best.

By looking at the average excess (%) (marked by superscript[b]), in Table 9, the combination of CSCX2 and EXCH is found to be best for two instances–ftv47 and ftv55; the combination of CSCX2 and 3-EXCH is best for only one instance–ftv44; the combination of CSCX2 and DISP is best for five instances–ft53, ftv64, ft70, ftv70, and kro124p; the combination of CSCX2 and INST is found to be best for two instances–ftv33 and ftv38; the combination of CSCX2 and INVS is best for three instances–ftv35, p43, and ry48p; the combination of CSCX2 and ADAP is best for only one instance-ftv170. Hence, the combination of CSCX2 and DISP is the best one, the combination of CSCX2 and

INVS is the second best, and the combination of CSCX2 and INST, and CSCX2 and EXCH are the third best.

Finally, regardless of the combination of any crossover and any mutation operator, by looking at the average excess (%) (marked by **boldface** superscript[a] or superscript[b]), in Table 9, the combination of CSCX1 and EXCH, and, the combination of CSCX1 and 3-EXCH is found to be best for only one instance–p43; the combination of CSCX1 and DISP is found to be best for two instances–ft70 and kro124p; the combination of CSCX1 and INST is found to be best for four instances–ftv33, ftv35, ftv47 and ft53. The remaining combinations of CSCX1 and ADAP, CSCX2 and EXCH, CSCX2 and 3-EXCH, CSCX2 and DISP, CSCX2 and INST, CSCX2 and INVS, CSCX2 and ADAP, are observed as the best for only one instance each. Hence, among all combinations, the combination of CSCX1 and INST is observed as the best one, and the combination of CSCX1 and DISP is the second best. Looking at Table 9, one can see that GA using CSCX1 and INST could find average solutions whose average percentage of excesses from the best-known solutions are 5.61, 2.43, 4.57, 0.22, 3.87, 4.30, 6.04, 9.65, 3.79, 2.80, 5.90, 2.21, 9.67 and 14.94 for ftv33, ftv35, ftv38, p43, ftv44, ftv47, ry48p, ft53, ftv55, ftv64, ft70, ftv70, kro124p and ftv170, respectively.

6 Conclusions and Future Work

Simple genetic algorithms using seven crossover operators and six mutation operators are suggested for the well-known travelling salesman problem (TSP). First, the crossover and mutation operators were illustrated through some example chromosomes. Then, experimental studies have been carried out on the benchmark TSPLIB instances. To observe the real behavior of different crossover operators, we implemented simple genetic algorithms using seven different crossover operators with a cent percentage of crossover probability and without any mutation operator on the benchmark instances. It can be seen from our study that comprehensive sequential constructive crossovers are very effective, especially, since CSCX2 is the best one and CSCX1 is the second best. Next, to observe the behavior of different mutation operators, we implemented simple genetic algorithms using six different mutation operators with ten percentage of mutation probability and without any crossover operator on the benchmark instances. It can be seen from our study that insertion mutation is the best one, inversion mutation is the second best, and displacement mutation is the third best. To observe the behavior of the combination of different mutation and crossover operators, we implemented simple genetic algorithms using seven crossover and six different mutation operators on the benchmark instances. By looking at the percentage of average solution excess, one can see from our study that the combination of CSCX-1 and insertion mutation is observed as the best one, and the combination of CSCX-1 and displacement mutation is the second best. The GA using CSCX1 with insertion mutation could find average solutions whose average percentage of excesses from the best-known solutions are between 0.22 and 14.94 for our experimented problem instances.

Our aim in this study was only to compare several crossover operators and several mutation operators regarding the solution quality. Our aim was not to improve the solution quality, so, we did not incorporate any local search method to enhance the solution quality. Although the combination of comprehensive sequential constructive crossover and insertion mutation was observed as the best, it still could not obtain the exact solution to many experimented problem instances. Of course, a thoroughly fine-tuned parameter set, that is, crossover probability, population size, mutation probability, etc., may find better quality solutions. Some literature combines metaheuristic and exact algorithms, and uses sub-tour division [36], to find better solutions. We also plan to combine an exact method and a metaheuristic algorithm to find better-quality solutions to the problem instances.

Acknowledgement: The authors are very much thankful to the anonymous honorable reviewers for their careful reading of the paper and their many insightful comments and constructive suggestions which helped the authors to improve the paper. The authors further are very much thankful to the Deanship of Scientific Research at Imam Mohammad Ibn Saud Islamic University (IMSIU) for its financial support.

Funding Statement: This work was supported and funded by the Deanship of Scientific Research at Imam Mohammad Ibn Saud Islamic University (IMSIU) (Grant Number IMSIU-RP23030).

Author Contributions: The authors confirm their contribution to the paper as follows: study conception and design: Z. H. Ahmed; data collection and implementation: H. Haron, A. Al-Tameem; analysis and interpretation of results: Z. Ahmed, H. Haron, A. Al-Tameem; draft manuscript preparation: H. Haron; final manuscript preparation and revision: Z. H. Ahmed, A. Al-Tameem. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are openly available in the TSPLIB repository at http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] R. M. Karp, *Reducibility among Combinatorial Problems*. Boston, MA: Springer, 1972, pp. 85–103.
[2] A. Larni-Fooeik, N. Ghasemi, and E. Mohammadi, "Insights into the application of the traveling salesman problem to logistics without considering financial risk: A bibliometric study," *Manage. Sci. Lett.*, vol. 14, no. 3, pp. 189–200, 2024. doi: 10.5267/j.msl.2023.11.002.
[3] W. Zhang, J. J. Sauppe, and S. H. Jacobson, "Results for the close-enough traveling salesman problem with a branch-and-bound algorithm," *Comput. Optim. Appl.*, vol. 85, no. 2, pp. 369–407, 2023. doi: 10.1007/s10589-023-00474-3.
[4] T. Q. T. Vo, M. Baiou, and V. H. Nguyen, "A branch-and-cut algorithm for the balanced traveling salesman problem," *J. Comb. Optim.*, vol. 47, no. 4, pp. 515, 2024. doi: 10.1007/s10878-023-01097-4.
[5] R. Carr, R. Ravi, and N. Simonetti, "A new integer programming formulation of the graphical traveling salesman problem," *Math. Program.*, vol. 197, no. 2, pp. 877–902, 2023. doi: 10.1007/s10107-022-01849-w.
[6] Z. H. Ahmed and I. Al-Dayel, "An exact algorithm for the single-depot multiple travelling salesman problem," *IJCSNS Int. J. Comput. Sci. Netw. Secur.*, vol. 20, no. 9, pp. 65–75, 2020.
[7] Y. Wang and Z. Han, "Ant colony optimization for traveling salesman problem based on parameters optimization," *Appl. Soft Comput.*, vol. 107, no. 2, pp. 107439, 2021. doi: 10.1016/j.asoc.2021.107439.
[8] Z. H. Ahmed, "Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator," *Int. J. Biomet. Bioinform.*, vol. 3, no. 6, pp. 96–105, 2010.
[9] İ. İlhan and G. Gökmen, "A list-based simulated annealing algorithm with crossover operator for the traveling salesman problem," *Neural Comput. Appl.*, vol. 34, no. 10, pp. 7627–7652, 2022. doi: 10.1007/s00521-021-06883-x.
[10] M. Gendreau, G. Laporte, and F. Semet, "A tabu search heuristic for the undirected selective travelling salesman problem," *Eur. J. Oper. Res.*, vol. 106, no. 2–3, pp. 539–545, 1998. doi: 10.1016/S0377-2217(97)00289-0.
[11] Y. Cui, J. Zhong, F. Yang, S. Li, and P. Li, "Multi-subdomain grouping-based particle swarm optimization for the traveling salesman problem," *IEEE Access*, vol. 8, pp. 227497–227510, 2020. doi: 10.1109/ACCESS.2020.3045765.

[12] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York: Addison-Wesley, 1989.
[13] G. Reinelt, "TSPLIB—A traveling salesman problem library," *ORSA J. Comput.*, vol. 3, no. 4, pp. 376–384, 1991. Accessed: Nov. 20, 2023. [Online]. Available: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
[14] Z. H. Ahmed, "Genetic algorithm with comprehensive sequential constructive crossover for the travelling salesman problem," *Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 5, pp. 245–254, 2020. doi: 10.14569/IJACSA.2020.0110533.
[15] D. B. Fogel, "An evolutionary approach to the travelling salesman problem," *Biol. Cybern.*, vol. 60, no. 2, pp. 139–144, 1988. doi: 10.1007/BF00202901.
[16] J. J. Grefenstette, "Incorporating problem specific knowledge into genetic algorithms," in L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing*, London, UK: Pitman/Pearson, 1987, pp. 42–60.
[17] L. Davis, "Job-shop scheduling with genetic algorithms," in *Proc. Int. Conf. Genetic Algor. Appl.*, 1985, pp. 136–140.
[18] B. Freisleben and P. Merz, "A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems," in *Proc. 1996 IEEE Int. Conf. Evol. Comput.*, Nagoya, Japan, 1996, pp. 616–621.
[19] D. E. Goldberg and R. Lingle, "Alleles, loci and the travelling salesman problem," in J. J. Grefenstette (Ed.), *Proc. First Int. Conf. Genetic Algor. Appl.*, Hilladale, NJ, Lawrence Erlbaum Associates, 1985.
[20] G. Syswerda, "Schedule optimization using genetic algorithms," in L. Davis (Ed.), *Handbook of Genetic Algorithms*, New York: Van Nostrand Reinhold, 1991, pp. 332–349.
[21] 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.), *Proc. First Int. Conf. Genetic Algor. Appl.*, Hilladale, NJ, Lawrence Erlbaum Associates, 1987.
[22] J. Grefenstette, R. Gopal, B. Rosmaita, and D. Gucht, "Genetic algorithms for the traveling salesman problem," in J. J. Grefenstette (Ed.), *Proc. First Int. Conf. Genetic Algor. Appl.*, Mahwah, NJ, Lawrence Erlbaum Associates, 1985, pp. 160–168.
[23] N. J. Radcliffe and P. D. Surry, "Formae and variance of fitness," in D. Whitley, M. Vose (Eds.), *Foundations of Genetic Algorithms 3*, San Mateo, CA: Morgan Kaufmann, 1995, pp. 51–72.
[24] D. Whitley, T. Starkweather, and D. Shaner, "The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination," in L. Davis (Ed.), *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, 1991, pp. 350–372.
[25] W. Banzhaf, "The molecular traveling salesman," *Biol. Cybern.*, vol. 64, no. 1, pp. 7–14, 1990. doi: 10.1007/BF00203625.
[26] D. Fogel, "A parallel processing approach to a multiple travelling salesman problem using evolutionary programming," in *Proc. Fourth Annual Symp. Parallel Process.*, Fullerton, California, 1990, pp. 318–326.
[27] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution programs*. Berlin: Springer-Verlag, 1992.
[28] M. Gen and R. Cheng, *Genetic Algorithm and Engineering Design*. New York: John Wiley and Sons, 1997, pp. 118–127.
[29] S. J. Louis and R. Tang, "Interactive genetic algorithms for the traveling salesman problem," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, 1999, pp. 385–392.
[30] Z. H. Ahmed, "An improved genetic algorithm using adaptive mutation operator for the quadratic assignment problem," in *Proc. 37th Int. Conf. Telecommun. Signal Process (TSP 2014)*, Berlin, Germany, 2014, pp. 616–620.
[31] A. B. Doumi, B. A. Mahafzah, and H. Hiary, "Solving traveling salesman problem using genetic algorithm based on efficient mutation operator," *J. Theor. Appl. Inf. Technol.*, vol. 99, no. 15, pp. 3768–3781, 2021.
[32] I. H. Khan, "Assessing different crossover operators for travelling salesman problem," *IJISA Int. J. Intell. Syst. Appl.*, vol. 7, no. 11, pp. 19–25, 2015. doi: 10.5815/ijisa.2015.11.03.
[33] Z. H. Ahmed, "Adaptive sequential constructive crossover operator in a genetic algorithm for solving the traveling salesman problem," *IJACSA Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 2, pp. 593–605, 2020. doi: 10.14569/issn.2156-5570.

[34] Z. H. Ahmed, "Solving the traveling salesman problem using greedy sequential constructive crossover in a genetic algorithm," *IJCSNS Int. J. Comput. Sci. Netw. Secur.*, vol. 20, no. 2, pp. 99–112, 2020.
[35] Z. H. Ahmed, "Experimental analysis of crossover and mutation operators for the quadratic assignment problem," *Ann. Oper. Res.*, vol. 247, no. 2, pp. 833–851, 2016. doi: 10.1007/s10479-015-1848-y.
[36] R. Jain *et al.*, "Application of proposed hybrid active genetic algorithm for optimization of traveling salesman problem," *Soft Comput.*, vol. 27, no. 8, pp. 4975–4985, 2023. doi: 10.1007/s00500-022-07581-z.