Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

Effects of crossover operator combined with mutation operator in genetic algorithms for the generalized travelling salesman problem

Zakir Hussain Ahmed[a*], Md. Taizuddin Choudhary[b] and Ibrahim Al-Dayel[a]

[a]Department of Mathematics and Statistics, College of Science, Imam Mohammad Ibn Saud Islamic University (IMSIU), Riyadh 11432, Kingdom of Saudi Arabia
[b]Department of Mathematics, B. R. A. Bihar University, Muzaffarpur, Bihar, India, 84200

C H R O N I C L E

Article history:
Received January 22 2024
Received in Revised Format April 30 2024
Accepted May 25 2024
Available online May 25 2024

Keywords:
Generalized travelling salesman problem
Genetic algorithms
Crossover operator
Mutation operator
Sequential constructive crossover
Insertion mutation

A B S T R A C T

Here, we consider the generalized travelling salesman problem (GTSP), which is a generalization of the travelling salesman problem (TSP). This problem has several real-life applications. Since the problem is complex and NP-hard, solving this problem by exact methods is very difficult. Therefore, researchers have applied several heuristic algorithms to solve this problem. We propose the application of genetic algorithms (GAs) to obtain a solution. In the GA, three operators—selection, crossover, and mutation—are successively applied to a group of chromosomes to obtain a solution to an optimization problem. The crossover operator is applied to create better offspring and thus to converge the population, and the mutation operator is applied to explore the areas that cannot be explored by the crossover operator and thus to diversify the search space. All the crossover and mutation operators developed for the TSP can be used for the GTSP with some modifications. A better combination of these two operators can create a very good GA to obtain optimal solutions to the GTSP instances. Therefore, four crossover and three mutation operators are used here to develop GAs for solving the GTSP. Then, GAs is compared on several benchmark GTSPLIB instances. Our experiment shows the effectiveness of the sequential constructive crossover operator combined with the insertion mutation operator for this problem.

1. Introduction

The travelling salesman problem (TSP) is defined as a network of n cities (or nodes) with a distance matrix denoted by $D=[d_{ij}]$. The problem is to find a minimum distance tour for the salesman who visits every city only once. It is among the most researched combinatorial optimization problems (COPs). It has several applications in logistics, planning, clustering of data arrays, manufacturing of microchips, DNA sequencing, machine scheduling problems, and crystallographic X-rays (Hu & Raidl, 2008). In the literature, numerous exact and heuristic procedures are available for obtaining solutions to this problem. However, there are conditions in real-life problems for which additional constraints must be imposed on the basic TSP. This leads to variations in the TSP, for example, TSP with time windows, TSP with backhauls, multiple TSP, and generalized TSP, etc. We propose to study the generalized TSP (GTSP), which was first introduced by Henry-Labordere (1969). The GTSP is an extension of the traditional TSP. In the GTSP, cities are partitioned into m clusters, represented by $C_1, C_2, \dots, C_m$. The aim and objective of the problem is to find a minimum distance tour (Hamiltonian cycle) for the salesman who visits exactly one city in each cluster. The GTSP becomes the traditional TSP when the number of clusters is the same as the number of cities and when each cluster contains exactly one city. If the distance matrix D is asymmetric, then the GTSP is an asymmetric GTSP; otherwise, it is a symmetric GTSP. The clustered TSP (CTSP) is a variation of the GTSP that finds a minimum distance tour by visiting all cities, with the restriction that all cities in a cluster must be visited consecutively before leaving the cluster (Ahmed, 2013a). The GTSP is known to be an NtP-hard problem (Henry-Labordere, 1969). It has many interesting real-life applications. For example, if a salesman wants to distribute his product among all his dealers in the country, he could visit all the local dealers in only one out of many possible cities in each state, so he can minimize the costs of his trip (Noon and Bean,

* Corresponding author
E-mail zaahmed@imamu.edu.sa (Z. H. Ahmed)
ISSN 1923-2934 (Online) - ISSN 1923-2926 (Print)
2024 Growing Science Ltd.
doi: 10.5267/j.ijiec.2024.5.004

1993). Furthermore, the GTSP has applications in mail delivery (Laporte et al., 1996), airplane routing (Grefenstette, 1987), etc. To solve the TSP and its variations, several procedures have been suggested in the literature: one exact procedure and another heuristic/metaheuristic procedure. Dynamic programming (Henry-Labordere, 1969), branch and bound (Dimitrijevic et al., 1996), branch and cut (Fischetti, 1997), and lexisearch algorithm (Ahmed, 2011) are exact methods that provide exact solutions. However, as the problem size increases, finding exact solutions using these methods becomes very difficult. There is no good exact algorithm for solving this problem exactly in polynomial time, and for small-sized problem instances, very long computation times are needed. Thus, one must apply heuristic algorithms to solve the problem. However, heuristic methods do not ensure exact solutions but rather give near exact solutions rapidly. The most recent methods for solving several COPs are called metaheuristics. Several metaheuristics have been suggested for solving the usual TSP and its variations because of the difficulty of solving these problems. Some of these algorithms include ant colony optimization (Yang et al., 2008), genetic algorithms (Huang et al., 2005), particle swarm optimization (Tasgetiren et al., 2007), and variable neighborhood method (Hu & Raidl, 2008). However, the genetic algorithm (GA) is recognized as one of the leading heuristic procedures for solving such problems.

John Holland proposed GAs, which correspond to the survival-of-the-fittest principle among variant communities created using random differences in the chromosome configuration in the biological sciences. They are recognized as effective because of their easy, flexible, and simple ability to code solutions as chromosomes. In GAs, several chromosomes, together called the initial population, are passed through three main processes (operators), namely, selection, crossover, and mutation, to obtain an appropriate solution to a given problem. In the selection process, better chromosomes are selected probabilistically. In the crossover process, two-parent chromosomes mate to create better offspring chromosome(s), whereas the mutation process randomly alters some of the genetic material in the chromosomes. The crossover process is highly important among these processes for designing and implementing GAs (Goldberg, 1989). One very positive thing is that any crossover technique constructed for the TSP can be used for the GTSP with some modifications. Therefore, we applied four crossover and three mutation operators to develop GAs and studied their effectiveness on GTSPLIB problem instances (Fischetti et al., 1997) of different sizes. The computational experiment proves the success of the sequential constructive crossover operator combined with the insertion mutation operator. Furthermore, we aim to establish a relative rank of these four crossover techniques for the GTSP.

This manuscript is arranged as follows: Section 2 briefly reviews the current literature on the problem. Section 3 develops different GAs to solve the GTSP. Computational results for several GAs using different crossover and mutation procedures on asymmetric GTSPLIB instances are reported in Section 4. Finally, Section 5 presents future work and concluding remarks.

2. Literature review

The GTSP was first introduced by Henry-Labordere (1969) for real-life applications involving routing clients through welfare agencies and recording balancing problems encountered in computer design. To solve the GTSP, the author developed a dynamic programming (DP) method, which is obtained from DP approaches for the usual TSP, in which a state is defined by the clusters that were already visited. However, in this method, as the number of clusters increases, the number of states increases exponentially. Based on integer linear programming, a method was proposed by Laporte & Norbert (1983) for solving the problem and applied to both Euclidean and non-Euclidean instances provided that the distance matrix is symmetric. The proposed method could solve the largest instance of size 50 cities with 10 clusters. Based on the minimal rooted tree as a relaxation, a branch-and-bound algorithm was proposed by Dimitrijevic et al. (1996) for solving the problem. The proposed method could solve the largest instance of size 52 cities with 13 clusters. A branch-and-cut method was suggested by Fischetti et al. (1997) for finding the solution to the problem that involves a combination of branch-and-bound and cutting plane methods. The method could solve the largest instance of size 442 cities with 89 clusters.

A composite heuristic of generalized initialization, insertion, and improvement was proposed by Renaud et al. (1996) for solving the TSP, which was modified by Renaud & Boctor (1998) to solve the GTSP. The algorithm has three steps: initialization, insertion, and improvement. In the initialization step, a partial tour is created, whereas in the insertion step, the tour is completed by adding the cities with the shortest distances from clusters that are not included. In the improvement step, the tour is improved using 2-opt, 3-opt, and 4-opt heuristics. A random-key GA was proposed by Snyder & Daskin (2006) for solving the GTSP. Furthermore, two local search methods, the swap method and the 2-opt method, were combined to develop a memetic algorithm. Different GAs were developed by Silberholz & Golden (2007) using different genetic operators to solve the GTSP. Furthermore, two local improvement methods, the 2-opt heuristic and swap methods, were used to enhance the performance of the GAs. A memetic algorithm was developed by Bontoux et al. (2010) for the GTSP by combining a GA with local search methods. Helsgaun (2015) developed a heuristic algorithm by combining a transformation of the GTSP into the usual TSP proposed by Noon & Bean (1993) with the Lin-Kernighan-Helsgaun TSP solver to solve the GTSP instances. As reported, the algorithm could improve the solution quality for the GTSPLIB instances over the formerly best available algorithms. A heuristic approach was suggested by Smith & Imeson (2007) for solving the GTSP using an adaptive large neighborhood search. The algorithm eliminates and includes cities over and over into the created tour. Furthermore, nearest, farthest, and random insertion methods are used to improve the tour. Three GTSP neighborhoods were described and incorporated into an efficient iterated local search method by Schmidt & Irnich (2022). Additionally, cluster optimization was used.

3. Our Genetic Algorithms

We aim to demonstrate the performances of different crossover operators combined with mutation operators to solve the GTSP. Briefly, in GAs, different genetic operators have different impacts on the solutions to an optimization problem; following initial population generation, selection operators, crossover operators, and mutation operators are considered for designing various GAs for the GTSP.

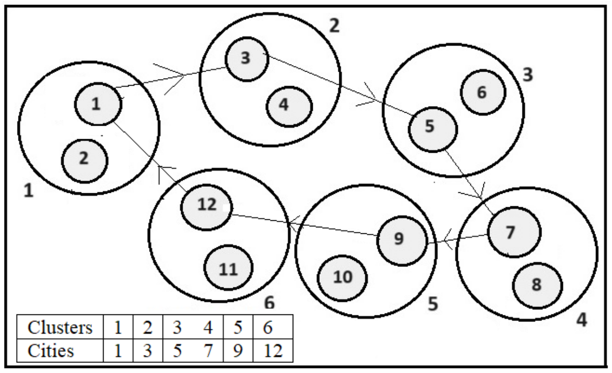
3.1 Initial Population

To solve any optimization problem using a GA, initially, a representation of a solution should be defined as a chromosome. The simplest way to represent the solution path is through the cities listed in order. Here, the local-global method proposed by Potvin (1996) is used for the GTSP that distinguishes between local connections (connections among cities from different clusters) and global connections (connections among clusters). An individual is represented as a list of clusters ($C_1, C_2, ..., C_m$) that represents the global tour (Hamiltonian cycle) $\{C_1-C_2-...-C_m\}$. Given a global tour ($C_1, C_2, ..., C_m$), one of the tours can be found as $\{v_1, v_2..., v_m)$ with the property that city $v_i \in C_i$ for all $i \in \{1, 2..., m\}$, which is a local Hamiltonian cycle (tour). For example, the individual (1, 5, 3, 7, 4, 2, 6) represents the global tour that passes the clusters in the order $C_1-C_5-C_3-C_7-C_4-C_2-C_6-C_1$.

Suppose that n = 12 and m = 6, and the clusters are as follows: $C_1 = \{1, 2\}$, $C_2 = \{3, 4\}$, $C_3 = \{5, 6\}$, $C_4 = \{7, 8\}$, $C_5 = \{9, 10\}$ and $C_6 = \{11, 12\}$. If the global path is (1, 2, 3, 4, 5, 6), then one of the local paths may be (1, 3, 5, 7, 9, 12), which can be shown together as in Fig. 1 and Fig. 2. In Fig. 1, the first row includes integers from 1 to 6, which shows the chain of clusters that were visited by the salesperson, and the second row includes the cities selected from the corresponding clusters that were visited by the salesperson in the chain.

Clusters	1	2	3	4	5	6
Cities	1	3	5	7	9	12

Fig. 1. A chromosome representation



Clusters	1	2	3	4	5	6
Cities	1	3	5	7	9	12

Fig. 2. A tour is represented as a chromosome

This local path is an example of a chromosome whose objective function value is the total distance of the cities in the path. Since our problem is a minimization problem, the fitness function is the inverted objective function. Therefore, a group of chromosomes called an initial population of size P_s is generated arbitrarily.

3.2 Selection Operator

In this selection operator, several stronger chromosomes are selected, and weaker chromosomes are discarded from the current population to construct the next generation population. The selection operator improves the performance of GAs, excluding the GA, which can produce variant results throughout generations, similar to what random sampling can produce. The algorithm maintains stability between the exploitation and exploration of the GA search space. There are numerous selection methods available in literature. The roulette wheel selection (RWS) (Goldberg, 1989) method using the fitness proportional rule is applied for our GAs.

3.3 Crossover Operators

In GAs, crossover is a very important operation that is executed on two chromosomes to reproduce offspring chromosome(s). The selection operator and the crossover operator can accelerate the convergence of solutions. The usual crossover operators may not be able to reproduce legal offspring(s) for the GTSP. However, the TSP-based crossover operators can be modified to implement the GTSP.

3.3.1 Partially Mapped Crossover Operator

The partially mapped crossover (PMX) method was developed by Goldberg and Lingle (1985) and defines an exchange mapping in subchromosomes between two arbitrary points in a pair of chromosomes. This crossover was designed for the usual TSP. We modify this crossover operator and then apply it to the GTSP as follows. Let P1 and P2 be two parent chromosomes with distances of 128 and 163, respectively, concerning the distances given in Table 1 and the same clusters (n = 12 and m = 6) as mentioned in Section 3.1. We use these same parent chromosomes to illustrate all crossover operators here.

P1

1	3	5	4	6	2
1	6	10	7	12	4

P2

2	4	5	6	3	1
4	7	9	11	5	1

Table 1
Distance Matrix

	Cluster	**1**		**2**		**3**		**4**		**5**		**6**	
Cluster	**Node**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
1	**1**	999	3	5	48	48	8	8	5	5	3	3	0
	2	3	999	3	48	48	8	8	5	5	0	0	5
2	**3**	5	3	999	72	72	48	48	24	24	3	3	5
	4	48	48	74	999	0	6	6	12	12	48	48	48
3	**5**	48	48	74	0	999	6	6	12	12	48	48	48
	6	8	8	50	6	6	999	0	8	8	8	8	8
4	**7**	8	8	50	6	6	0	999	8	8	8	8	8
	8	5	5	26	12	12	8	8	999	0	5	5	5
5	**9**	5	5	26	12	12	8	8	0	999	5	5	5
	10	3	0	3	48	48	8	8	5	5	999	0	3
6	**11**	3	0	3	48	48	8	8	5	5	0	999	3
	12	0	3	5	48	48	8	8	5	5	3	3	999

The arbitrarily chosen cutoff points were assumed to be between the 3rd and 4th genes and between the 5th and 6th genes, and the area between them is shown by shading.

P1

1	3	5	4	6	2
1	6	10	7	12	4

P2

2	4	5	6	3	1
4	7	9	11	5	1

The mapping segments of the clusters between these points are 4↔6 and 6↔3. These segments are reproduced in offspring as follows:

O1

*	*	*	4	6	*
			7	12	

O2

*	*	*	6	3	*
			11	5	

We now add other clusters along with the cities from the alternative parent chromosome that do not result in an infeasible offspring chromosome:

O1

2	*	5	4	6	1
4		9	7	12	1

O2

1	*	5	6	3	2
1		10	11	5	4

The * in O1 would be cluster 4, which is from P2 but is present in O1; thus, we added cluster 3 along with city 5 by using the cluster map 4↔6 and 6↔3. Therefore, complete O1 offspring is produced at a distance of 76. Similarly, the * in O2 is cluster 3, which is from P1 but is present in O2; thus, we added cluster 4 along city 7 by using the cluster map 4↔6 and 6↔3. Therefore, complete O2 offspring is produced at a distance of 112.

O1

2	3	5	4	6	1
4	5	9	7	12	1

O2

1	4	5	6	3	2
1	7	10	11	5	4

3.3.2. Ordered Crossover Operator

The ordered crossover (OX) method was developed by Davis (19985) to construct offspring by selecting a section of a chromosome from one parent chromosome and preserving the relative chain of clusters along with the cities from another chromosome. This operator was designed for the usual TSP. We modify this operator and then apply it to the GTSP as follows.

Suppose the randomly chosen cutoff points are between the 3rd and 4th genes and between the 5th and 6th genes, and the area between them is shown by shading.

P1	1	3	5	4	6	2	P2	2	4	5	6	3	1
	1	6	10	7	12	4		4	7	9	11	5	1

First, the sub-chromosomes between the cutoff points are replicated in the offspring chromosomes as follows:

O1	*	*	*	4	6	*	O2	*	*	*	6	3	*
				7	12						11	5	

Next, beginning from the 2nd cutoff point of one parent chromosome, clusters along with the corresponding cities from the other parent, excluding present clusters in the offspring, are replicated by maintaining the order. The order of the clusters in P2 from the 2nd cutoff point is {1, 2, 4, 5, 6, 3}. After eliminating the present clusters {4, 6}, the order becomes {1, 2, 5, 3}; that is, these clusters are placed along with the corresponding cities in O1 beginning from the 2nd cutoff point. Next, wrap around the starting of the offspring chromosome when it goes to the end to build a complete offspring chromosome. Similarly, the second offspring chromosome is also created. Therefore, complete O1 and O2 offspring are produced at distances of 86 and 115, respectively.

O1	2	5	3	4	6	1	O2	1	5	4	6	3	2
	4	9	5	7	12	1		1	10	7	11	5	4

3.3.3. Cycle Crossover Operator

Cycle crossover (CX) was developed by Oliver et al. (1987), where offspring chromosomes are created such that each cluster (and its place) and the corresponding city originate from one of the parent chromosomes. It was designed for the usual TSP. We modify this term and apply it to the GTSP as follows.

The O1 and O2 offspring chromosomes are generated by considering the 1st clusters and their corresponding cities from P1 and P2, respectively.

O1	1	*	*	*	*	*	O2	2	*	*	*	*	*
	1							4					

Since every cluster in the offspring chromosome must be included from one of its parent chromosomes (from the same place), in O1, the next cluster must be cluster 2, as this cluster is from P2, which is in the same place as the present cluster 1. In P1, this cluster is at place ‘6’. Thus, cluster 2, along with city 4, is copied in 6th place in O1. Similarly, in O2, the next cluster must be cluster 1, as this cluster is from P1, which is in the same place as the present cluster 2. In P2, this cluster is at place ‘6’. Thus, cluster 1, along with city 1, is copied in 6th place in O2.

O1	1	*	*	*	*	2	O2	2	*	*	*	*	1
	1					4		4					1

Next, in O1, we must select cluster 1, as this cluster is from P2, which is in the same place as the present cluster 2, but cluster 1 is already present in O1. Thus, a cycle is completed. Similarly, in O2, we must select cluster 2, as this cluster is from P1, which is in the same place as the present cluster 1, but cluster 2 is already present in O2. Thus, a cycle is completed. Therefore, in both O1 and O2, the remaining clusters along with their corresponding cities are chosen and filled from the other parent chromosomes. Therefore, complete O1 and O2 offspring are produced at distances of 117 and 78, respectively.

O1	1	4	5	6	3	2	O2	2	3	5	4	6	1
	1	7	9	11	5	4		4	6	10	7	12	1

3.3.4. Sequential constructive crossover operator

Sequential constructive crossover (SCX) was suggested by Ahmed (2010) for the usual TSP method, which generates only one offspring chromosome. The tool sequentially examines parent chromosomes and selects the 1st legitimate (untouched) gene that is discovered after the present gene on both parent chromosomes. In a parent chromosome, if no legitimate gene is found, it sequentially examines from the beginning of that parent chromosome and selects the legitimate gene. After that, the distance of each gene from the gene of interest was calculated, and the better gene was added to the offspring chromosome. The SCX algorithm was successfully applied to the TSP with variations (Ahmed, 2013b, 2013c, 2014, 2020). We modified this expression and then applied it to the GTSP as in Algorithm 1.

Algorithm 1: Sequential constructive crossover algorithm

```
Input: Distance matrix D, Crossover probability $P_c$, Pair of parent chromosomes.
Output: Offspring chromosome.
Generate random number $r \in [0,1]$.
if ($r \leq P_c$) then do
    Set t = 1st cluster and p = 1st city from 1st parent chromosome.
    The offspring chromosome contains ‘cluster t’ along with ‘city p’.
    for i = 2 to m do
        In each chromosome consider the first ‘legitimate cluster' that appeared after 'cluster t’.
        if a 'legitimate cluster' is not available in a chromosome, then
            Search from the starting of the chromosome (wrap around) and consider the first ‘legitimate
            cluster' that appeared after 'cluster t’.
        end if
        Suppose ‘cluster x’ along with 'city α' and ‘cluster y’ along with 'city β' are found in 1st and 2nd parent
        chromosomes, respectively.
        if ($d_{p\alpha} < d_{p\beta}$) then do
            Add ‘cluster x’ along with 'city α' to the offspring chromosome.
        Else
            Add ‘cluster y’ along with 'city β' to the offspring chromosome.
        end if
        Rename the present cluster as 'cluster t' and the present city as ‘city p’ and continue.
    end for
end if
Return the offspring chromosome
```

To illustrate the operation of the SCX, we consider the same parent chromosomes. The offspring chromosome O is generated by considering the 1st cluster 1 and its corresponding city 1 from P1.

O

1	*	*	*	*	*
1					

The legitimate clusters after cluster 1 in P1 and P2 are cluster 3 with city 6 and cluster 2 (after wrapped around) with city 4, respectively. The distances from city 1 to city 3 and from city 1 to city 2 are 8 and 48, respectively. Since cluster 3 with city 6, is cheaper, we added this information to the offspring chromosome.

O

1	3	*	*	*	*
1	6				

The legitimate clusters after cluster 3 in P1 and P2 are cluster 5 with city 10 and cluster 2 (after wrapped around) with city 4, respectively. The distances from city 6 to city 10 and from city 6 to city 4 are 8 and 6, respectively. Since cluster 2 with city 4 is cheaper, we added this information to the offspring chromosome.

O

1	3	2	*	*	*
1	6	4			

The legitimate clusters after cluster 2 in P1 and P2 are cluster 5 (after wrapped around) with city 10 and cluster 4 with city 7, respectively. Now, the distances from city 4 to city 10 and from city 4 to city 7 are 48 and 6, respectively. Since cluster 4 with city 7, is cheaper, we added this information to the offspring chromosome.

O	1	3	2	4	*	*
	1	6	4	7		

The legitimate clusters after cluster 4 in P1 and P2 are cluster 6 with city 12 and cluster 5 with city 9, respectively. The distances from city 7 to city 12 and from city 7 to city 9 are both 8. So, the first cluster, i.e., cluster 6 with city 12, is cheaper, we added this information to the offspring chromosome.

O	1	3	2	4	6	*
	1	6	4	7	12	

The legitimate clusters after cluster 6 in P1 and P2 are cluster 5 (after wrapped around) with city 10 and cluster 5 with city 9, respectively. Now, the distances from city 12 to city 10 and from city 12 to city 9 are 3 and 5, respectively. Since cluster 5 with city 10, is cheaper, we added this information to the offspring chromosome. Therefore, complete offspring O is produced at a distance of 34.

O	1	3	2	4	6	5
	1	6	4	7	12	10

3.4. Mutation Operators

By introducing arbitrary changes in the GA population, the variety in the population can be increased through the use of a mutation operator. In this operator, some gene(s) in a chromosome are randomly chosen and then changed; thus, the information is changed. We use the following three mutation operators with mutation probability, P_{mt}, and then compare them. All the mutation operators are illustrated through the following chromosome P.

P	1	3	2	4	6	5
	1	6	4	7	12	10

3.4.1. Swap Mutation

The swap mutation (SWPM) operator arbitrarily chooses two places in a chromosome and then swaps the genes in these places (Banzhaf, 1990). For example, if 2nd and 4th places are arbitrarily chosen and cluster 3 with city 6 and cluster 4 with city 7 are swapped in their places, then the mutated chromosome P will be as follows.

Muted P	1	4	2	3	6	5
	1	7	4	6	12	10

3.4.2. Insertion Mutation

The insertion mutation (INSM) operator arbitrarily chooses one place in a chromosome, takes the corresponding gene, and then inserts it at any random place (Fogel, 1988). For example, if 3rd place is arbitrarily chosen and cluster 2 with city 4 is inserted between 5th and 6th places, then the mutated chromosome P will be as follows.

Muted P	1	3	4	6	2	5
	1	6	7	12	4	10

3.4.3. Inversion Mutation

The inversion mutation (INVM) operator arbitrarily chooses two places in a chromosome and then inverts the subchromosome between these places (Fogel, 1990). For example, if the 3rd and 5th places are arbitrarily chosen and the subchromosome between them is inverted, then the mutated chromosome P will be as follows.

Muted P	1	3	6	4	2	5
	1	6	12	7	4	10

Our simple GAs are nonhybrid and simply apply the usual GA modules and GA processes. Our GAs initialize an arbitrary population, choose fitter chromosomes using the roulette wheel selection procedure, and apply a chosen crossover procedure and a mutation procedure. One of the simple GAs is described in Algorithm 2.

Algorithm 2: *Simple genetic algorithm*

```
Input: n, m, D, $P_s$, $P_c$, $P_{mt}$.
Output: Best chromosome with its value.
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 a selection operator.

    Perform a crossover operator with crossover probability $P_c$.

    Perform a mutation operator with mutation probability $P_{mt}$.

    Evaluate the population.

end while
Return the best chromosome with its value.
```

4. Computational Experiments

Several simple GAs with four crossover operators-PMX, OX, CX, and SCX-and three mutation-SWPM, INSM, and INVM-were coded in Visual C++ and run on asymmetric GTSPLIB instances (Fischetti et al., 1997) on a laptop with an Intel(R) Core(TM) i7-1065G7 CPU@1.30 GHz and 8.00GB RAM under MS Windows 11. Each GA was run 20 times for each problem instance. There are 88 symmetric and asymmetric instances with up to 1084 cities, and 217 clusters are present in the GTSPLIB. These instances are obtained from the benchmark TSPLIB (Reinelt, 1991). It is found that asymmetric instances are relatively harder than symmetric instances are; therefore, we considered only 19 asymmetric instances. We used a random population for running all algorithms to avoid biases in a particular algorithm. The best solution (BS), average solution (AS), and percentage of the excess of average solution (AE(%)) over the best-known solution (BKS) (Ben-Arieh et al., 2003), standard deviation (SD) of the solutions, and average time (AT) (in seconds) to discover the best solution for the first time among the 20 executions were reported for every instance by all the algorithms. The AE (%) of the average solution was calculated by the formula: AE (%) = 100×(AS/BKS - 1).

The GAs are verified to be dominated by the size of the population (P_s), probability of crossover (P_c), probability of mutation (P_{mt}), and stopping condition. We set $P_s = 200$ and $P_c = 1.0$ to check the absolute functioning of the crossover methods and almost equal computing time as a stopping condition. First, we look at the functioning of the crossover methods on these asymmetric instances. A comparative investigation among four GAs using four distinct crossover operators with no mutation method is shown in Table 2. In this table, the instance name and its BKS inside the brackets are reported in the first column, the data names are reported in the second column, and the results obtained by distinct crossovers are reported in the next four columns. Additionally, the best results are indicated by the **boldface** symbols.

In Table 2, investigating the **boldfaces**, the GA using SCX with no mutation is revealed to be the best among all the other GAs using other crossovers. Looking at the best solution, the GA using PMX could hit (at least once in 20 runs) the BKS for four instances-4br17, 7ftv33, 9p43 and 9ftv44; the GA using OX could hit the BKS for seven instances-4br17, 7ftv33, 8ftv35, 8ftv38, 9p43, 9ftv44 and 12ftv55; the GA using CX could hit the BKS for two instances-only 4br17 and 7ftv33; and the GA using SCX could hit the BKS for eight instances-4br17, 7ftv33, 8ftv35, 8ftv38, 9p43, 9ftv44, 10ftv47, and 12ftv55. From this observation, it can be claimed that the GA using SCX is the leading GA, the GA using OX is the next best, the GA using PMX is the third most common, and the GA using CX is the least common.

Table 2
Comparison of GAs using four crossovers with no mutation for antisymmetric GTSPLIB instances

Instance	Results	PMX	OX	CX	SCX	Instance	Results	PMX	OX	CX	SCX
4br17	BS	**31**	**31**	**31**	**31**	13ftv64	BS	793	763	896	735
(31)	AS	31.00	31.00	31.00	31.00	(708)	AS	867.05	821.75	1132.20	777.05
	AE(%)	**0.00**	**0.00**	**0.00**	**0.00**		AE(%)	22.46	16.07	59.92	**9.75**
	SD	0.00	0.00	0.00	0.00		SD	81.41	35.66	99.86	26.14
	AT	0.00	0.00	0.00	0.00		AT	0.00	0.00	0.00	0.00
7ftv33	BS	**476**	**476**	**476**	**476**	14ft70	BS	8202	7978	8819	8074
(476)	AS	534.30	501.00	543.75	480.85	(7707)	AS	8721.15	8260.50	9508.30	8271.10
	AE(%)	12.25	5.25	14.23	**1.02**		AE(%)	13.16	7.18	23.37	**7.32**
	SD	48.86	25.11	40.43	14.59		SD	128.96	145.76	301.81	130.56
	AT	0.00	0.00	0.00	0.00		AT	0.02	0.01	0.00	0.00
8ftv35	BS	545	**525**	579	**525**	15ftv70	BS	687	625	947	613
(525)	AS	582.25	561.25	645.25	556.55	(594)	AS	759.85	726.05	1157.05	652.10
	AE(%)	10.90	6.90	22.90	**6.01**		AE(%)	27.92	22.23	94.79	**9.78**
	SD	37.97	27.11	40.01	25.88		SD	97.67	50.49	116.54	24.44
	AT	0.00	0.00	0.00	0.00		AT	0.00	0.00	0.00	0.00
8ftv38	BS	554	**511**	569	**511**	20kro124p	BS	13808	13549	21667	11999
(511)	AS	599.10	535.75	628.30	516.55	(11203)	AS	15975.80	14963.10	23278.80	12753.25
	AE(%)	17.24	4.84	22.95	**1.09**		AE(%)	42.60	33.56	107.79	**13.84**
	SD	25.88	22.79	28.36	6.10		SD	1109.32	876.64	1391.15	380.09
	AT	0.00	0.00	0.00	0.00		AT	0.01	0.01	0.00	0.00
9p43	BS	**5563**	**5563**	5564	**5563**	35ftv170	BS	1515	1532	3377	1359
(5563)	AS	5570.65	5564.60	5572.30	5565.65	(1205)	AS	1773.75	1743.30	2772.60	1463.35
	AE(%)	0.14	**0.03**	0.17	0.05		AE(%)	47.20	44.67	130.09	**21.44**
	SD	3.56	1.39	7.92	3.13		SD	262.69	130.62	219.11	52.15
	AT	0.00	0.01	0.00	0.00		AT	0.03	0.06	0.01	0.01
9ftv44	BS	**510**	**510**	567	**510**	65rbg323	BS	673	637	950	548
(510)	AS	562.40	552.45	678.10	522.80	(471)	AS	765.25	702.55	1014.80	578.65
	AE(%)	10.27	8.32	32.96	**2.51**		AE(%)	62.47	49.16	115.46	**22.86**
	SD	40.05	17.09	54.14	15.71		SD	34.00	28.90	39.38	16.27
	AT	0.00	0.00	0.00	0.00		AT	0.13	1.05	0.02	0.07
10ftv47	BS	594	589	697	**569**	72rbg358	BS	1033	860	1105	803
(569)	AS	672.75	607.90	783.05	606.90	(693)	AS	1090.90	935.20	1215.85	849.55
	AE(%)	18.23	6.84	37.62	**6.66**		AE(%)	57.42	34.95	75.45	**22.59**
	SD	32.14	20.63	40.13	17.15		SD	35.56	27.79	38.26	21.46
	AT	0.00	0.00	0.00	0.00		AT	0.10	1.79	0.02	0.09
10ry48p	BS	6547	6339	6987	6320	81rbg403	BS	1235	1244	1491	1290
(6284)	AS	6905.90	6583.05	7468.35	6481.95	(1170)	AS	1567.25	1302.40	1531.90	1319.10
	AE(%)	9.90	4.76	18.85	**3.15**		AE(%)	33.95	**11.32**	30.93	12.74
	SD	405.97	117.51	285.53	132.88		SD	79.30	36.03	23.00	17.78
	AT	0.00	0.00	0.00	0.00		AT	0.14	3.04	0.05	0.12
11ft53	BS	2711	2654	2850	2656	89rbg443	BS	920	976	1356	748
(2648)	AS	2914.75	2886.50	3318.90	2712.80	(632)	AS	1187.35	1038.75	1443.6	799.7
	AE(%)	10.07	9.01	25.34	**2.45**		AE(%)	87.87	64.36	128.42	**26.53**
	SD	121.16	107.31	172.68	54.68		SD	35.89	25.97	49.20	27.14
	AT	0.00	0.00	0.00	0.00		AT	0.34	3.80	0.05	0.20
12ftv55	BS	751	**689**	817	**689**						
(689)	AS	784.80	714.75	953.55	690.20						
	AE(%)	13.90	3.74	38.40	**0.17**						
	SD	76.78	33.53	73.26	3.82						
	AT	0.00	0.00	0.00	0.00						

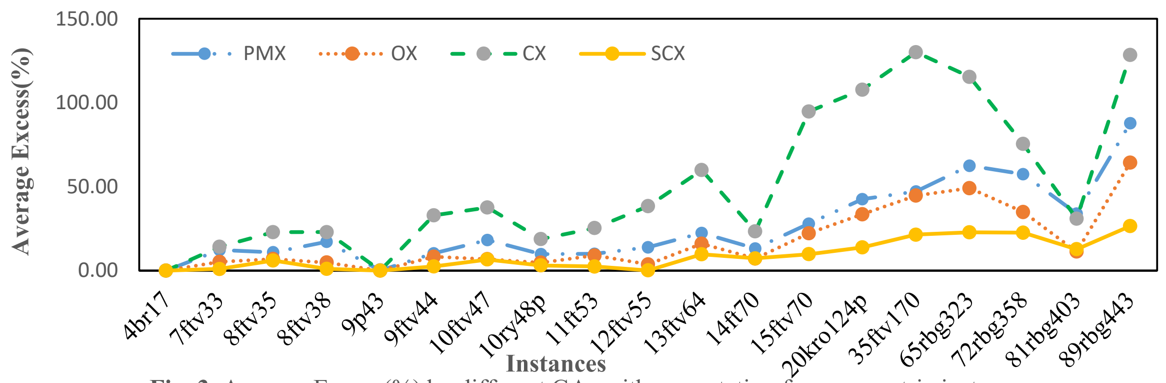


Fig. 3. Average Excess(%) by different GAs with no mutation for asymmetric instances

Considering the average solution in Table 2, the same conclusion can be drawn. That is, the GA using the SCX operator without the mutation operator is found to be best, the GA using the OX operator without the mutation operator is found to be the second best, and the GA using the CX operator without the mutation operator is found to be the worst. Fig. 3 shows the results that confirm the usefulness of GA using SCX. We now execute different GAs using four crossovers and three mutations using the same mutation probability. Certainly, fixing the same probability of mutation for all mutations for every instance is not an easy task. After executing several mutation probabilities, we fix $P_{mt} = 0.20$ and a maximum of *10n* generations as the stopping condition. A comparative investigation among the four GAs using four different crossovers with the SWPM is shown in Table 3.

Table 3
Comparison of GAs using four crossovers with SWPM for antisymmetric GTSPLIB instances.

Instance	Results	PMX	OX	CX	SCX	Instance	Results	PMX	OX	CX	SCX
4br17	BS	**31**	**31**	**31**	**31**	13ftv64	BS	736	726	746	**708**
(31)	AS	31.00	31.00	31.00	31.00	(708)	AS	817.95	801.95	839.15	737.95
	AE(%)	**0.00**	**0.00**	**0.00**	**0.00**		AE(%)	15.53	13.27	18.52	**4.23**
	SD	0.00	0.00	0.00	0.00		SD	33.45	29.53	69.93	21.79
	AT	0.00	0.00	0.00	0.00		AT	0.01	0.01	0.01	0.00
7ftv33	BS	**476**	**476**	**476**	**476**	14ft70	BS	8060	7930	8200	7830
(476)	AS	489.02	487.95	500.50	476.00	(7707)	AS	8345.50	8296.05	8362.95	8050.90
	AE(%)	2.74	2.51	5.15	**0.00**		AE(%)	8.28	7.64	8.51	**4.46**
	SD	25.32	24.66	30.10	0.00		SD	157.08	142.32	163.13	128.01
	AT	0.00	0.00	0.00	0.00		AT	0.01	0.01	0.01	0.01
8ftv35	BS	**525**	**525**	**525**	**525**	15ftv70	BS	635	625	706	596
(525)	AS	565.25	549.20	590.95	533.30	(594)	AS	711.50	661.50	749.55	610.00
	AE(%)	7.67	4.61	12.56	**1.58**		AE(%)	19.78	11.36	26.19	**2.69**
	SD	32.42	25.23	30.52	24.54		SD	61.33	57.32	70.12	18.56
	AT	0.00	0.00	0.00	0.00		AT	0.02	0.02	0.01	0.00
8ftv38	BS	513	**511**	516	**511**	20kro124p	BS	13611	13211	13442	11817
(511)	AS	531.55	526.80	548.55	512.45	(11203)	AS	15921.20	14721.20	13382.80	12732.75
	AE(%)	4.02	3.09	7.35	**0.28**		AE(%)	42.12	31.40	19.46	**13.65**
	SD	20.75	24.82	22.75	11.09		SD	578.52	578.52	713.73	376.50
	AT	0.00	0.00	0.00	0.00		AT	0.05	0.05	0.06	0.01
9p43	BS	**5563**	**5563**	**5563**	**5563**	35ftv170	BS	1515	1522	1424	1271
(5563)	AS	5564.05	5563.90	5564.05	5563.00	(1205)	AS	1601.25	1501.25	1686.65	1378.75
	AE(%)	0.02	0.02	0.02	**0.00**		AE(%)	32.88	24.59	39.97	**14.42**
	SD	1.90	2.77	1.80	0.00		SD	103.99	95.36	200.56	63.45
	AT	0.00	0.00	0.00	0.00		AT	0.27	0.27	0.44	0.26
9ftv44	BS	**510**	**510**	543	**510**	65rbg323	BS	673	625	659	517
(510)	AS	547.65	542.65	581.50	515.60	(471)	AS	762.95	675.32	681.50	543.45
	AE(%)	7.38	6.40	14.02	**1.10**		AE(%)	61.99	43.38	44.69	**15.38**
	SD	15.16	14.56	40.18	20.91		SD	16.38	15.46	13.90	16.18
	AT	0.00	0.00	0.00	0.00		AT	1.14	1.14	1.86	3.62
10ftv47	BS	583	580	593	**569**	72rbg358	BS	1033	860	859	732
(569)	AS	589.75	584.75	614.70	580.10	(693)	AS	1049.25	918.25	882.40	774.40
	AE(%)	3.65	2.77	8.03	**1.95**		AE(%)	51.41	32.50	27.33	**11.75**
	SD	10.66	9.23	37.83	9.22		SD	11.25	10.23	17.89	22.69
	AT	0.00	0.00	0.00	0.00		AT	2.00	2.01	3.10	5.94
10ry48p	BS	6369	6349	**6284**	**6284**	81rbg403	BS	1220	1225	1303	1272
(6284)	AS	6518.60	6479.65	6464.95	6351.65	(1170)	AS	1450.10	1290.10	1335.15	1311.85
	AE(%)	3.73	3.11	2.88	**1.08**		AE(%)	23.94	**10.26**	14.12	12.12
	SD	125.08	115.03	175.10	87.89		SD	16.85	15.78	17.75	19.78
	AT	0.00	0.00	0.00	0.00		AT	1.75	1.73	3.01	9.83
11ft53	BS	2702	2662	2672	**2648**	89rbg443	BS	920	921	873	656
(2648)	AS	2805.25	2756.53	2791.50	2659.20	(632)	AS	1157.95	1009.45	960.60	727.00
	AE(%)	5.94	4.10	5.42	**0.42**		AE(%)	83.22	59.72	51.99	**15.03**
	SD	67.18	59.12	116.70	26.97		SD	17.47	17.47	15.61	29.34
	AT	0.00	0.00	0.00	0.00		AT	2.82	2.82	3.44	8.49
12ftv55	BS	**689**	**689**	**689**	**689**						
(689)	AS	696.00	694.00	703.60	689.00						
	AE(%)	1.02	0.73	2.12	**0.00**						
	SD	12.68	13.02	27.64	0.00						
	AT	0.01	0.01	0.01	0.00						

In Table 3, investigating the **boldfaces**, the GA using SCX with SWPM is found to be the best among all the other GAs using other crossovers. Looking at the best solution, the GA using PMX with SWPM could hit (at least once in 20 runs) the BKS for six instances-4br17, 7ftv33, 8ftv35, 9p43, 9ftv44 and 12ftv55; the GA using OX with SWPM could hit the BKS for seven instances-4br17, 7ftv33, 8ftv35, 8ftv38, 9p43, 9ftv44 and 12ftv55; the GA using CX with SWPM could hit the BKS for six instances-4br17, 7ftv33, 8ftv35, 9p43, 10ry48p and 12ftv55; and the GA using SCX with SWPM could hit the BKS for eleven instances-4br17, 7ftv33, 8ftv35, 8ftv38, 9p43, 9ftv44, 10ftv47, 10ry48p, 11ft53, 12ftv55 and 13ftv64. From this observation, one can say that the GA using SCX with SWPM is in the top position, the GA using OX with SWPM is in the next position, the GA using PMX with SWPM and the GA using CX with SWPM are competing for the third position.

Considering the average solution in Table 3, one can reach a clearer conclusion. The GA using SCX with SWPM was the best, the GA using OX with SWPM was the second best, the GA using CX with SWPM was the third best, and the GA using PMX with SWPM was the worst. Fig. 4 shows the results that confirm the usefulness of GA using SCX with SWPM.

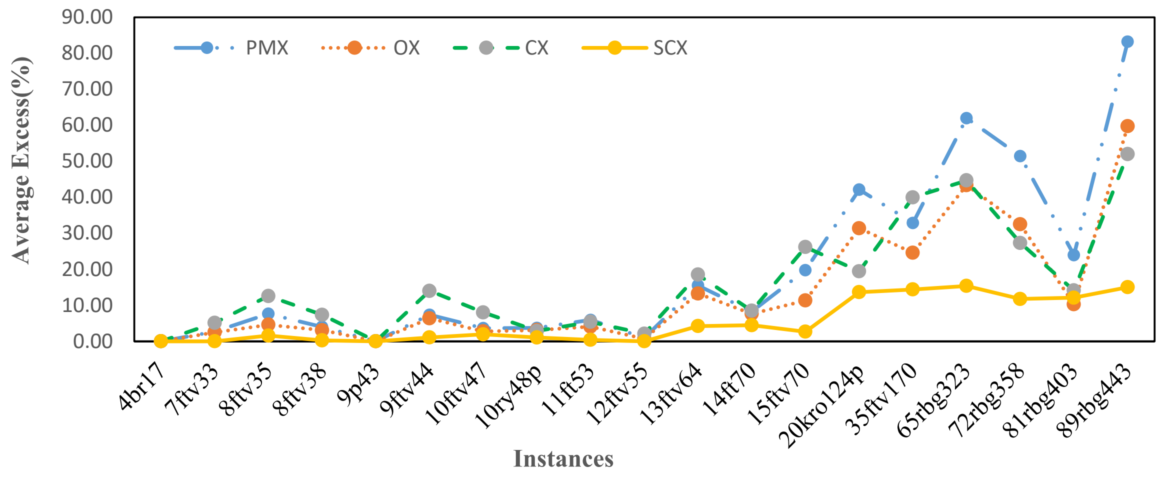


Fig. 4. Average Excess(%) by different GAs with SWPM for asymmetric instances

Table 4
Comparison of GAs using four crossovers with INSM for antisymmetric GTSPLIB instances

Instance	Results	PMX	OX	CX	SCX	Instance	Results	PMX	OX	CX	SCX
4br17	BS	**31**	**31**	**31**	**31**	13ftv64	BS	756	735	751	**708**
(31)	AS	31.00	31.00	31.00	31.00	(708)	AS	820.20	800.20	829.65	733.45
	AE(%)	**0.00**	**0.00**	**0.00**	**0.00**		AE(%)	15.85	13.02	17.18	**3.59**
	SD	0.00	0.00	0.00	0.00		SD	39.95	33.54	45.14	34.39
	AT	0.00	0.00	0.00	0.00		AT	0.01	0.01	0.01	0.00
7ftv33	BS	**476**	**476**	**476**	**476**	14ft70	BS	7980	7960	7965	7922
(476)	AS	488.12	486.20	511.50	476.00	(7707)	AS	8283.20	8233.20	8226.40	8047.25
	AE(%)	2.55	2.14	7.46	**0.00**		AE(%)	7.48	6.83	6.74	**4.41**
	SD	32.47	24.82	30.16	0.00		SD	151.94	145.23	145.62	119.09
	AT	0.00	0.00	0.00	0.00		AT	0.01	0.01	0.01	0.01
8ftv35	BS	**525**	**525**	**525**	**525**	15ftv70	BS	646	636	647	**594**
(525)	AS	560.30	554.20	565.30	534.80	(594)	AS	693.55	643.55	676.30	604.40
	AE(%)	6.72	5.56	7.68	**1.87**		AE(%)	16.76	8.34	13.86	**1.75**
	SD	32.58	25.93	32.58	24.42		SD	44.99	39.78	81.00	26.06
	AT	0.00	0.00	0.00	0.00		AT	0.01	0.01	0.01	0.00
8ftv38	BS	513	**511**	516	**511**	20kro124p	BS	13564	13564	13572	12108
(511)	AS	531.00	518.60	541.00	512.45	(11203)	AS	15867.85	14767.85	14493.50	12779.50
	AE(%)	3.91	1.49	5.87	**0.28**		AE(%)	41.64	31.82	29.37	**14.07**
	SD	25.69	19.21	25.69	11.09		SD	635.78	635.78	532.55	325.72
	AT	0.00	0.00	0.00	0.00		AT	0.04	0.04	0.05	0.01
9p43	BS	**5563**	**5563**	**5563**	**5563**	35ftv170	BS	1502	1414	1636	1281
(5563)	AS	5563.80	5564.65	5563.90	5563.00	(1205)	AS	1595.32	1485.65	1807.85	1370.55
	AE(%)	0.01	0.03	0.02	**0.00**		AE(%)	32.39	23.29	50.03	**13.74**
	SD	2.64	3.24	2.64	0.00		SD	96.54	102.32	124.28	61.68
	AT	0.00	0.00	0.00	0.00		AT	0.31	0.31	0.33	0.38
9ftv44	BS	**510**	**510**	543	**510**	65rbg323	BS	695	625	636	500
(510)	AS	539.20	534.20	550.65	517.10	(471)	AS	756.70	683.32	678.10	531.10
	AE(%)	5.73	4.75	7.97	**1.39**		AE(%)	60.66	45.08	43.97	**12.76**
	SD	18.18	15.23	20.15	21.51		SD	14.57	1325.00	25.80	18.16
	AT	0.00	0.00	0.00	0.00		AT	1.03	1.03	1.88	3.78
10ftv47	BS	578	573	575	572	72rbg358	BS	987	878	869	708
(569)	AS	589.20	583.70	602.25	580.60	(693)	AS	1037.55	908.63	900.60	749.80
	AE(%)	3.55	2.58	5.84	**2.04**		AE(%)	49.72	31.12	29.96	**8.20**
	SD	18.23	17.52	21.18	8.52		SD	21.76	20.13	18.18	23.15
	AT	0.00	0.00	0.00	0.00		AT	1.77	1.59	2.65	7.11
10ry48p	BS	6319	6309	6309	**6284**	81rbg403	BS	1357	1267	1317	1271
(6284)	AS	6484.95	6435.45	6450.75	6309.95	(1170)	AS	1388.40	1298.40	1349.55	1300.40
	AE(%)	3.20	2.41	2.65	**0.41**		AE(%)	18.67	**10.97**	15.35	11.15
	SD	109.87	99.85	196.03	85.87		SD	16.89	15.32	19.93	14.74
	AT	0.01	0.01	0.00	0.00		AT	2.25	2.15	3.77	9.34
11ft53	BS	2683	2673	2674	2651	89rbg443	BS	893	897	910	**632**
(2648)	AS	2780.75	2729.55	2734.35	2668.65	(632)	AS	1134.75	986.24	987.85	688.60
	AE(%)	5.01	3.08	3.26	**0.78**		AE(%)	79.55	56.05	56.31	**8.96**
	SD	64.91	61.32	87.32	42.34		SD	18.38	18.38	16.21	21.71
	AT	0.01	0.01	0.01	0.00		AT	2.79	2.79	3.89	6.48
12ftv55	BS	**689**	**689**	**689**	**689**						
(689)	AS	698.55	693.55	715.05	689.00						
	AE(%)	1.39	0.66	3.78	**0.00**						
	SD	16.41	15.75	35.20	0.00						
	AT	0.00	0.00	0.01	0.00						

We now execute different GAs using four crossovers with the INSM using the same mutation probability. A comparative investigation among these GAs is shown in Table 4. In Table 4, investigating the **boldfaces**, the GA using SCX with INSM is in the first position among all the other GAs using other crossovers. Looking at the best solution, the GA using PMX with INSM could hit (at least once in 20 runs) the BKS for six instances-4br17, 7ftv33, 8ftv35, 9p43, 9ftv44 and 12ftv55; the GA using OX with INSM could hit the BKS for seven instances-4br17, 7ftv33, 8ftv35, 8ftv38, 9p43, 9ftv44 and 12ftv55; the GA using CX with INSM could hit the BKS for five instances-4br17, 7ftv33, 8ftv35, 9p43, and 12ftv55; and the GA using SCX with INSM could hit the BKS for eleven instances-4br17, 7ftv33, 8ftv35, 8ftv38, 9p43, 9ftv44, 10ry48p, 12ftv55, 13ftv64, 15ftv70 and 89rbg443. From this observation, one can say that the GA using SCX with INSM is in the top position, the GA using OX with INSM is in the next position, the GA using PMX with INSM is in third position, and the GA using CX with INSM is in the last position. Considering the average solution in Table 4, one can conclude that the GA using SCX with INSM is in the first position, the GA using OX with INSM is in the second position, the GA using CX with INSM is in the third position, and the GA using PMX with INSM is in the last position. Fig. 5 shows the results that confirm the usefulness of GA using SCX with the INSM.

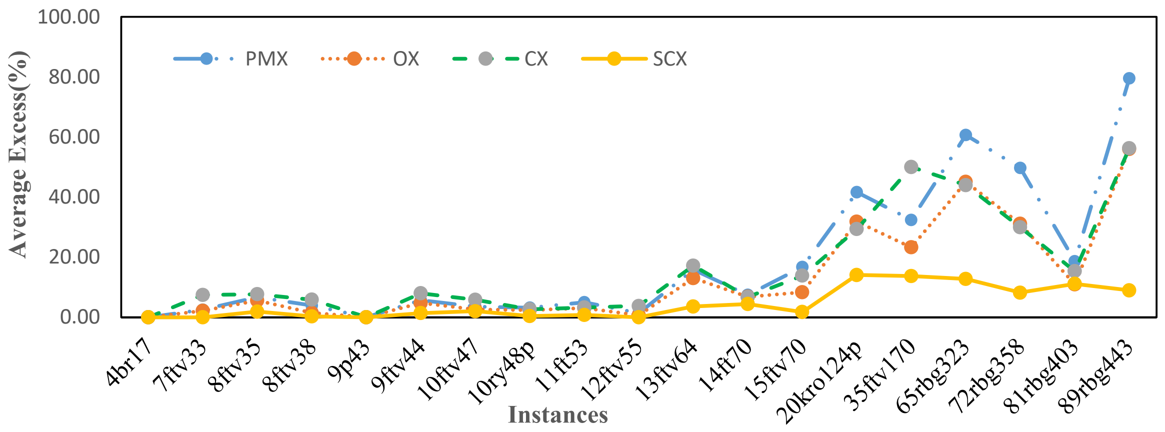


Fig. 5. Average Excess(%) by different GAs with INSM for asymmetric instances

We now execute different GAs using four crossovers with the INVM using the same mutation probability. A comparative investigation among these GAs is shown in Table 5. In Table 5, investigating the **boldfaces**, the GA using SCX with INVM is in the top position among all the other GAs using other crossovers. Looking at the best solution, the GA using PMX with INVM could hit (at least once in 20 runs) the BKS for seven instances-4br17, 7ftv33, 8ftv35, 8ftv38, 9p43, 9ftv44 and 12ftv55; the GA using OX with INVM could hit the BKS for seven instances-4br17, 7ftv33, 8ftv35, 8ftv38, 9p43, 9ftv44 and 12ftv55; the GA using CX with INVM could hit the BKS for six instances-4br17, 7ftv33, 8ftv35, 8ftv38, 9p43 and 12ftv55; and the GA using SCX with INVM could hit the BKS for eleven instances-4br17, 7ftv33, 8ftv35, 8ftv38, 9p43, 9ftv44, 10ftv47, 10ry48p, 11ft53, 12ftv55 and 15ftv70. From this observation, one can say that the GA using SCX with INVM is in the top position, the GA using PMX with INVM and the GA using OX with INVM are competing for the second position, and the GA using CX with INVM is in the last position. Considering the average solution in Table 5, one can conclude that the GA using SCX with INVM is in the first position, the GA using OX with INVM is in the second position, the GA using CX with INVM is in the third position, and the GA using PMX with INVM is in the last position. Fig. 6 shows the results that confirm the usefulness of GA using SCX with INVM.

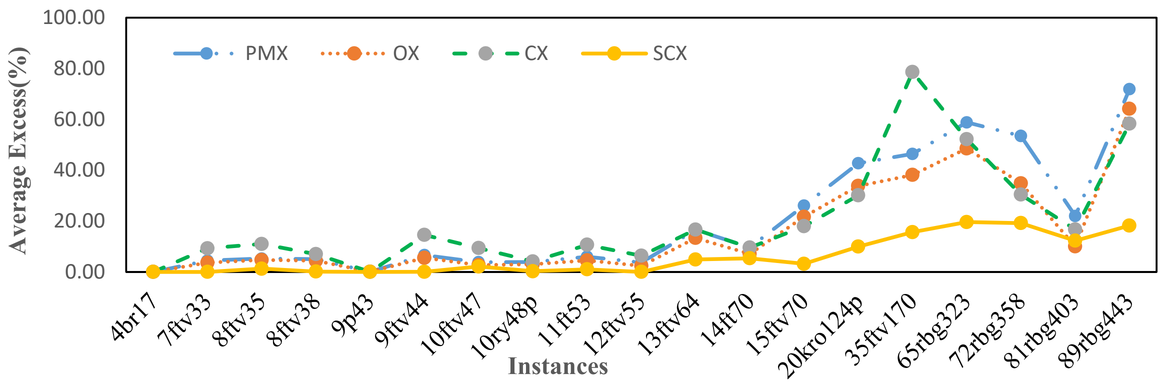


Fig. 6. Average Excess(%) by different GAs with INVM for asymmetric instances

Table 5
Comparison of GAs using four crossovers with the INVM for antisymmetric GTSPLIB instances

Instance	Results	PMX	OX	CX	SCX	Instance	Results	PMX	OX	CX	SCX
4br17	BS	**31**	**31**	**31**	**31**	13ftv64	BS	763	751	745	712
(31)	AS	31.00	31.00	31.00	31.00	(708)	AS	825.00	802.50	825.75	742.60
	AE(%)	**0.00**	**0.00**	**0.00**	**0.00**		AE(%)	16.53	13.35	16.63	**4.89**
	SD	0.00	0.00	0.00	0.00		SD	29.97	23.78	55.03	19.27
	AT	0.00	0.00	0.00	0.00		AT	0.01	0.01	0.01	0.00
7ftv33	BS	**476**	**476**	**476**	**476**	14ft70	BS	8238	8155	8264	8000
(476)	AS	497.03	493.40	520.25	476.00	(7707)	AS	8451.40	8255.14	8443.95	8119.45
	AE(%)	4.42	3.66	9.30	**0.00**		AE(%)	9.66	7.11	9.56	**5.35**
	SD	35.45	25.15	34.00	0.00		SD	143.70	154.32	149.55	113.93
	AT	0.00	0.00	0.00	0.00		AT	0.01	0.01	0.02	0.01
8ftv35	BS	**525**	**525**	**525**	**525**	15ftv70	BS	735	705	678	**594**
(525)	AS	552.70	549.80	582.70	531.75	(594)	AS	748.80	722.80	700.70	612.85
	AE(%)	5.28	4.72	10.99	**1.29**		AE(%)	26.06	21.68	17.96	**3.17**
	SD	29.42	28.92	29.42	26.84		SD	42.54	43.02	60.95	19.05
	AT	0.00	0.00	0.00	0.00		AT	0.02	0.02	0.01	0.00
8ftv38	BS	**511**	**511**	**511**	**511**	20kro124p	BS	14472	14472	14128	11504
(511)	AS	536.70	533.35	546.70	511.65	(11203)	AS	15988.90	14988.90	14577.50	12320.50
	AE(%)	5.03	4.37	6.99	**0.13**		AE(%)	42.72	33.79	30.12	**9.98**
	SD	28.63	20.13	28.63	4.73		SD	666.82	666.82	651.07	443.11
	AT	0.00	0.00	0.00	0.00		AT	0.04	0.04	0.07	0.01
9p43	BS	**5563**	**5563**	**5563**	**5563**	35ftv170	BS	1601	1425	1704	1309
(5563)	AS	5564.75	5565.75	5564.80	5563.00	(1205)	AS	1764.60	1664.60	2151.80	1393.20
	AE(%)	0.03	0.05	0.03	**0.00**		AE(%)	46.44	38.14	78.57	**15.62**
	SD	2.34	3.52	2.34	0.00		SD	82.49	98.32	112.74	67.15
	AT	0.00	0.00	0.00	0.00		AT	0.29	0.29	0.40	0.29
9ftv44	BS	**510**	**510**	515	**510**	65rbg323	BS	685	645	678	532
(510)	AS	543.70	538.23	584.05	510.15	(471)	AS	748.10	699.63	716.90	563.32
	AE(%)	6.61	5.54	14.52	**0.03**		AE(%)	58.83	48.54	52.21	**19.60**
	SD	14.60	13.24	39.28	16.62		SD	15.65	13.48	26.54	17.52
	AT	0.00	0.00	0.00	0.00		AT	1.03	1.03	1.87	3.69
10ftv47	BS	582	573	589	**569**	72rbg358	BS	1025	905	889	792
(569)	AS	591.00	584.32	622.30	581.05	(693)	AS	1063.65	934.53	903.75	826.15
	AE(%)	3.87	2.69	9.37	**2.12**		AE(%)	53.48	34.85	30.41	**19.21**
	SD	13.94	12.87	29.73	7.14		SD	13.96	12.56	18.08	23.32
	AT	0.00	0.00	0.00	0.00		AT	1.48	1.49	2.41	4.58
10ry48p	BS	6332	6312	6303	**6284**	81rbg403	BS	1403	1243	1341	1253
(6284)	AS	6528.40	6474.45	6543.40	6300.85	(1170)	AS	1427.30	1287.30	1364.95	1313.75
	AE(%)	3.89	3.03	4.13	**0.27**		AE(%)	21.99	**10.03**	16.66	12.29
	SD	128.34	121.24	218.23	118.46		SD	14.49	13.89	14.57	18.54
	AT	0.01	0.01	0.00	0.01		AT	1.85	1.86	3.37	7.08
11ft53	BS	2757	2643	2659	**2648**	89rbg443	BS	1245	952	908	681
(2648)	AS	2805.55	2768.45	2930.80	2674.55	(632)	AS	1086.05	1037.32	1000.45	747.25
	AE(%)	5.95	4.55	10.68	**1.00**		AE(%)	71.84	64.13	58.30	**18.24**
	SD	70.97	65.49	89.31	35.22		SD	15.24	15.24	23.99	32.20
	AT	0.01	0.01	0.01	0.00		AT	2.52	2.52	3.10	10.01
12ftv55	BS	**689**	**689**	**689**	**689**						
(689)	AS	714.70	705.45	732.80	689.00						
	AE(%)	3.73	2.39	6.36	**0.00**						
	SD	26.56	25.63	48.41	0.00						
	AT	0.01	0.01	0.01	0.00						

From the above observation, it is very clear that with or without any mutation operator, SCX can produce the best solutions among the crossover operators. However, it is not clear which mutation operator, SCX can produce the best solutions among the mutation operators. For that reason, we summarize the above results in Table 6. In this table, the instance name and mutation name are reported in the first and second columns, respectively; however, AEs (%) caused by distinct crossovers with distinct mutations are reported in the next four columns; and the grand average (GAV) using all crossovers with a particular mutation operator is reported in the last column. Furthermore, a partial average solution (PAV) using a crossover operation with all the mutations is reported for each instance. The best results are shown in **boldface**.

Irrespective of any mutation, as shown in Table 6, by observing at PAV (in **boldface**), except for instance 4br17, the crossover operators PMX and CX could not obtain the best average for any problem instance; OX was the best for only one instance-81rbg403, whereas SCX was the best for the remaining seventeen instances. The PAVs that are reported in Table 6 are shown in Fig. 7. Hence, SCX is in the first position, OX is in the second position, and PMX is in the third position.

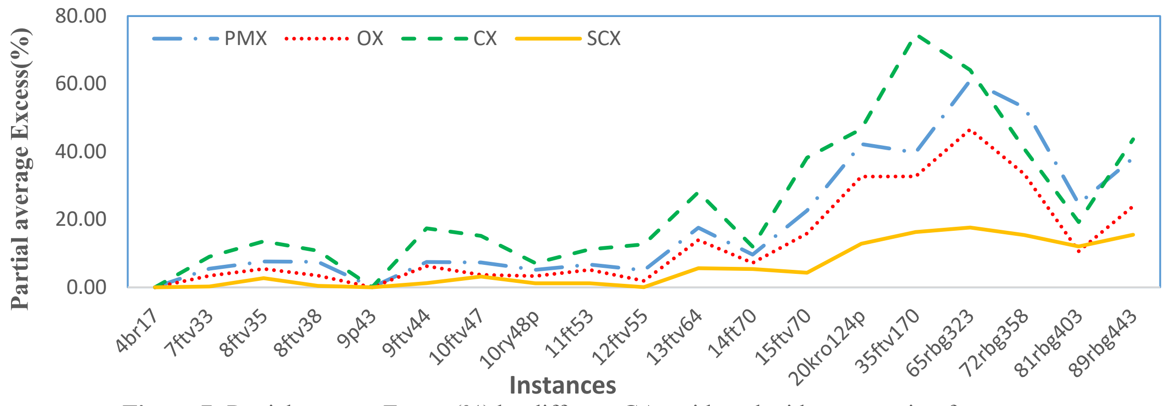


Figure 7: Partial average Excess(%) by different GAs with and without mutation for asymmetric instances

Table 6
Comparison of the crossovers and mutations for several GTSPLIB instances

Inst	Mute	PMX	OX	CX	SCX	GAV	Inst	Mute	PMX	OX	CX	SCX	GAV
4br17	NO	0.00	0.00	0.00	0.00	0.00	13ftv64	NO	22.46	16.07	59.92	9.75	27.05
	SWPM	0.00	0.00	0.00	0.00	0.00		SWPM	15.53	13.27	18.52	4.23	12.89
	INSM	0.00	0.00	0.00	0.00	0.00		INSM	15.85	13.02	17.18	3.59*	**12.41**
	INVM	0.00	0.00	0.00	0.00	0.00		INVM	16.53	13.35	16.63	4.89	12.85
	PAV	0.00	0.00	0.00	0.00			**PAV**	17.59	13.93	28.06	**5.62**	
7ftv33	NO	12.25	5.25	14.23	1.02	8.19	14ft70	NO	13.16	7.18	23.37	7.32	12.76
	SWPM	2.74	2.51	5.15	0.00	**2.60**		SWPM	8.28	7.64	8.51	4.46	7.23
	INSM	2.55	2.14	7.46	0.00	3.04		INSM	7.48	6.83	6.74	4.41*	**6.36**
	INVM	4.42	3.66	9.30	0.00	4.34		INVM	9.66	7.11	9.56	5.35	7.92
	PAV	5.49	3.39	9.03	**0.25**			**PAV**	9.64	7.19	12.05	**5.39**	
8ftv35	NO	10.90	6.90	22.90	6.01	11.68	15ftv70	NO	27.92	22.23	94.79	9.78	38.68
	SWPM	7.67	4.61	12.56	1.58	6.60		SWPM	19.78	11.36	26.19	2.69	15.01
	INSM	6.72	5.56	7.68	1.87	**5.46**		INSM	16.76	8.34	13.86	1.75*	**10.18**
	INVM	5.28	4.72	10.99	1.29*	5.57		INVM	26.06	21.68	17.96	3.17	17.22
	PAV	7.64	5.45	13.53	**2.69**			**PAV**	22.63	15.90	38.20	**4.35**	
8ftv38	NO	17.24	4.84	22.95	1.09	11.53	20kro124p	NO	42.60	33.56	107.79	13.84	49.45
	SWPM	4.02	3.09	7.35	0.28	3.69		SWPM	42.12	31.40	19.46	13.65	**26.66**
	INSM	3.91	1.49	5.87	0.28	**2.89**		INSM	41.64	31.82	29.37	14.07	29.23
	INVM	5.03	4.37	6.99	0.13*	4.13		INVM	42.72	33.79	30.12	9.98*	29.15
	PAV	7.55	3.45	10.79	**0.45**			**PAV**	42.27	32.65	46.69	**12.88**	
9p43	NO	0.14	0.03	0.17	0.05	0.10	35ftv170	NO	47.20	44.67	130.09	21.44	60.85
	SWPM	0.02	0.02	0.02	0.00	**0.01**		SWPM	32.88	24.59	39.97	14.42	**27.96**
	INSM	0.01	0.03	0.02	0.00	0.02		INSM	32.39	23.29	50.03	13.74*	29.86
	INVM	0.03	0.05	0.03	0.00	0.03		INVM	46.44	38.14	78.57	15.62	44.69
	PAV	0.05	0.03	0.06	**0.01**			**PAV**	39.73	32.67	74.67	**16.30**	
9ftv44	NO	10.27	8.32	32.96	2.51	13.52	65rbg323	NO	62.47	49.16	115.46	22.86	62.49
	SWPM	7.38	6.40	14.02	1.10	7.23		SWPM	61.99	43.38	44.69	15.38	41.36
	INSM	5.73	4.75	7.97	1.39	**4.96**		INSM	60.66	45.08	43.97	12.76*	**40.62**
	INVM	6.61	5.54	14.52	0.03*	6.67		INVM	58.83	48.54	52.21	19.60	44.80
	PAV	7.50	6.25	17.37	**1.26**			**PAV**	60.99	46.54	64.08	**17.65**	
10ftv47	NO	18.23	6.84	37.62	6.66	17.34	72rbg358	NO	57.42	34.95	75.45	22.59	47.60
	SWPM	3.65	2.77	8.03	1.95*	4.10		SWPM	51.41	32.50	27.33	11.75	30.75
	INSM	3.55	2.58	5.84	2.04	**3.50**		INSM	49.72	31.12	29.96	8.20*	**29.75**
	INVM	3.87	2.69	9.37	2.12	4.51		INVM	53.48	34.85	30.41	19.21	34.49
	PAV	7.32	3.72	15.22	**3.19**			**PAV**	53.01	33.36	40.79	**15.44**	
10ry48p	NO	9.90	4.76	18.85	3.15	9.16	81rbg403	NO	33.95	11.32	30.93	12.74	22.24
	SWPM	3.73	3.11	2.88	1.08	2.70		SWPM	23.94	10.26	14.12	12.12	15.11
	INSM	3.20	2.41	2.65	0.41	**2.17**		INSM	18.67	10.97	15.35	11.15*	**14.03**
	INVM	3.89	3.03	4.13	0.27*	2.83		INVM	21.99	10.03	16.66	12.29	15.24
	PAV	5.18	3.33	7.13	**1.23**			**PAV**	24.64	**10.65**	19.26	12.07	
11ft53	NO	10.07	9.01	25.34	2.45	11.72	89rbg443	NO	87.87	64.36	128.42	26.53	76.80
	SWPM	5.94	4.10	5.42	0.42*	3.97		SWPM	23.94	10.26	14.12	12.12	15.11
	INSM	5.01	3.08	3.26	0.78	**3.03**		INSM	18.67	10.97	15.35	11.15*	**14.03**
	INVM	5.95	4.55	10.68	1.00	5.55		INVM	21.99	10.03	16.66	12.29	15.24
	PAV	6.74	5.18	11.17	**1.16**			**PAV**	38.12	23.91	43.64	**15.52**	
12ftv55	NO	13.90	3.74	38.40	0.17	14.05							
	SWPM	1.02	0.73	2.12	0.00	**0.97**							
	INSM	1.39	0.66	3.78	0.00	1.46							
	INVM	3.73	2.39	6.36	0.00	3.12							
	PAV	5.01	1.88	12.66	**0.04**								

Irrespective of any crossover, as shown in Table 6, by observing at GAV (in **boldface**), except for instance 4br17, the mutation operator INVM could not obtain the best average for any problem instance; SWPM was the best for five instances-7ftv33, 9p43, 12ftv55, 20kro124p, and 35ftv170, whereas INSM was the best for the remaining thirteen instances. The GAV that is reported in Table 6 is shown in Fig. 8. Although SWPM and INSM compete, INSM is in the first position, SWPM is in the second position, and INVM is in the last position. Furthermore, GAs generated from mutations exhibited significant improvements compared with GAs without mutations.

Since the GAs using SCX with any mutation, the GAs using SWPM with any crossover, and the GAs using INSM with any crossover are better combinations, we now look at the GAs using SCX combined with any mutation, as shown in Table 6. We mark the best average excess (%) with an asterisk*. By observing the average excess (%), except for four instances 4br17, 7ftv33, 9p43, and 12ftv55, the GA combining SCX and SWPM is the best for only two instances, the GA combining SCX and INSM is the best for six instances, and the GA combining SCX and INVM is the best for five instances. Hence, the GA using SCX combined with INSM is in the first position, and the GA using SCX combined with INVM is in the second position.

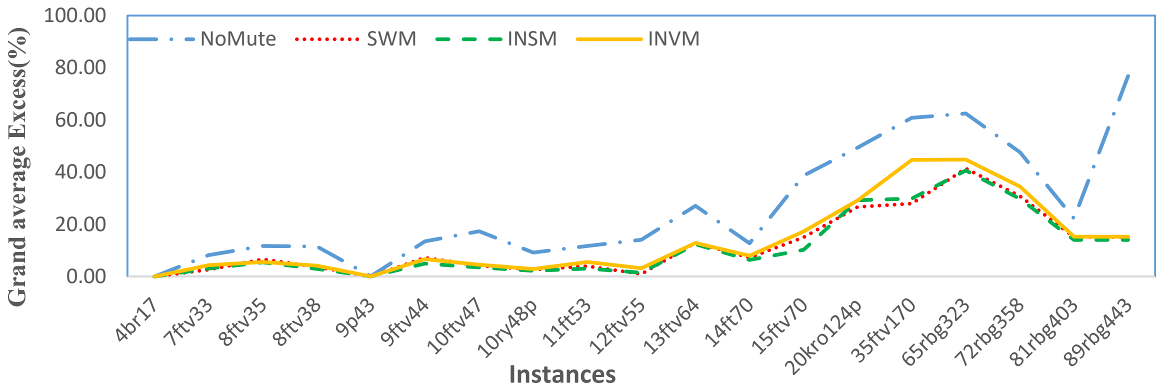


Fig. 8. Grand average Excess(%) by different GAs with and without mutation for asymmetric instances

5. Conclusions and future work

We suggest several simple genetic algorithms using four crossover and three mutation operators for solving the generalized travelling salesman problem (GTSP). We first illustrated the crossover and mutation operators using several examples. We then implemented the algorithms using Visual C++ and ran them on asymmetric GTSPLIB instances. To evaluate the performance of the crossovers, algorithms were run using only crossovers with a crossover probability of 1.00 and with no mutation on the problem instances. It was observed that sequential constructive crossover is highly effective. Next, to evaluate the performance of the combination of crossover and mutation operators, we run the algorithms using four crossovers with a probability of 1.00 and three mutations with a probability of 0.20 on asymmetric instances. From our present study, it appears that the genetic algorithm using sequential constructive crossover combined with insertion mutation is the top algorithm, and the genetic algorithm using sequential constructive crossover combined with inversion mutation is the second best.

In this research, we aimed to investigate different combinations of crossovers and mutations in genetic algorithms to find a solution to the GTSP and determine the best combination. We did not aim to find the best quality solution; hence, no local search algorithm was incorporated into the genetic algorithms. Even though the sequential constructive crossover combined with the insertion mutation was the topmost method, it was not able to find the exact optimal solution for some problem instances. Therefore, we plan to incorporate several local search algorithms and/or heuristic algorithms into simple genetic algorithms to obtain quality solutions for the problem instances.

Acknowledgment

This work was supported and funded by the Deanship of Scientific Research at Imam Mohammad Ibn Saud Islamic University (IMSIU) (grant number IMSIU-RPP2023104).

Availability of Data

The datasets generated and/or analyzed during the current study are available in the GTSPLIB repository, https://www.cs.rhul.ac.uk/home/zvero/GTSPLIB/.

Conflicts of interest

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

References

Ahmed, Z. H. (2010). Genetic Algorithm for the Traveling Salesman Problem using Sequential Constructive Crossover Operator. *International Journal of Biometrics & Bioinformatics. 3(6)*, 96–105.

Ahmed, Z. H. (2011). A data-guided lexisearch algorithm for the asymmetric traveling salesman problem. *Mathematical Problems in Engineering, 2011,* 750968.

Ahmed, Z. H. (2013a). An exact algorithm for the clustered travelling salesman problem. *Opsearch, 50(2),* 215-228.

Ahmed, Z. H. (2013b). A hybrid genetic algorithm for the bottleneck traveling salesman problem. *ACM Transactions on Embedded Computing Systems (TECS), 12(1),* 1–10.

Ahmed, Z. H. (2013c). An experimental study of a hybrid genetic algorithm for the maximum traveling salesman problem. *Mathematical Sciences, 7(6),* 1–7.

Ahmed, Z. H. (2014). The ordered clustered travelling salesman problem: A hybrid genetic algorithm. *The Scientific World Journal*, *2014*, 258207.

Ahmed, Z. H. (2020). A comparative study of eight crossover operators for the maximum scatter travelling salesman problem. *International Journal of Advanced Computer Science and Applications(IJACSA), 11(6),* 317-329.

Banzhaf, W. (1990). The molecular traveling salesman. *Biological Cybernetics, 64,* 7–14.

Ben-Arieh, D., Gutin, G., Penn, M., Yeo, A., & Zverovitch, A. (2003). Transformations of generalized ATSP into ATSP. *Operations Research Letters, 31(5)*, 357-365.

Bontoux, B., Artigues, C., & Feillet, D. (2010). A memetic algorithm with a large neighborhood crossover operator for the generalized traveling salesman problem. *Computers & Operations Research, 37(11),* 1844–1852.

Davis, L. (1985). Job-shop scheduling with genetic algorithms. In *Proc. ICGA*, Pittsburgh, PA, USA, 136-140.

Dimitrijevic, V., Milosavljevic, M., & Markovic, M. (1996). A branch and bound algorithm for solving a generalized traveling salesman problem. *Publikacije Elektrotehnic̆kog fakulteta, Serija Matematika, 7*, 31–35.

Fischetti, M., Salazar-Gonzales, J. J., & Toth, P. (1997). A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research, 45(3)*, 378–394. [https://www.cs.rhul.ac.uk/home/zvero/GTSPLIB/] [last accessed on 25th February 2024].

Fogel, D. B. (1988). An evolutionary approach to the travelling salesman problem. *Biological Cybernetics, 60(2),* 139-144.

Fogel, D. B. (1990). A parallel processing approach to a multiple travelling salesman problem using evolutionary programming. In *Proc. Fourth annual Symposium on Parallel Processing*, Fullerton, California, 318–326.

Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. Addison-Wesley Longman Publishing Co., New York.

Goldberg, D. E., & Lingle, R. (1985). Alleles, loci and the travelling salesman problem. In *Proc. ICGA*, Pittsburgh, PA, USA, 154–159.

Grefenstette, J. J. (1987). Incorporating problem specific knowledge into genetic algorithms. *Genetic algorithms and simulated annealing, 4*, 42–60.

Helsgaun, K. (2015). Solving the equality generalized traveling salesman problem using the Lin-Kernighan-Helsgaun algorithm. *Mathematical Programming Computation, 7(3)*, 269–287.

Henry-Labordere, A. L. (1969). The record balancing problem: A dynamic programming solution of a generalized salesman problem. *RIRO, B-2,* 43–49.

Hu, B., & Raidl, G. R. (2008). Effective neighborhood structures for the generalized traveling salesman problem. *Lecture Notes in Computer Science, 4972,* 36–47.

Huang, H., Yang, X., Hao, Z., Wu, C., Liang, Y., & Zhao, X. (2005). Hybrid chromosome genetic algorithm for generalized traveling salesman problems. *Lecture Notes in Computer Science*, *3612,* 137–140.

Laporte, G., & Norbert, Y. (1983). Generalized travelling salesman problem through n sets of nodes: An integer programming approach. *INFOR, 2(1),* 61–75.

Laporte, G., Asef-Vaziri, A., & Sriskandarajah, C. (1996). Some applications of the generalized travelling salesman problem. *Journal of the Operational Research Society, 47(12),* 1461–1467.

Noon, C. E., & Bean, J. C. (1993). An efficient transformation of the generalized traveling salesman problem. *INFOR, 31(1),* 39–44.

Oliver, I. M., Smith, D. J., & Holland, J. R. C. (1987). A Study of Permutation Crossover Operators on the Travelling Salesman Problem. In Grefenstette, J. J. (ed.) *Genetic Algorithms and Their Applications*, Proceedings of the 2nd International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hilladale, NJ, 224-230.

Potvin, J.-Y. (1996). Genetic Algorithms. *Annals of Operations Research, 63*, 339–370.

Reinelt, G. (1991). TSPLIB—A traveling salesman problem library. *ORSA Journal on Computing, 3(4),* 376-384.

Renaud, J., & Boctor, F. F. (1998). An efficient composite heuristic for the symmetric generalized traveling salesman problem. *European Journal of Operational Research, 108(3),* 571–584.

Renaud, J., Boctor, F. F., & Laporte, G. (1996). A fast composite heuristic for the symmetric traveling salesman problem. *INFORMS Journal on Computing, 8*, 134–143.

Schmidt, J., & Irnich, S. (2022). New neighborhoods and an iterated local search algorithm for the generalized traveling salesman problem. *EURO Journal on Computational Optimization, 10*, 100029.

Silberholz, J., & Golden, B. (2007). The generalized traveling salesman problem: A new genetic algorithm approach. In E. K. Baker, A . Joseph, A . Mehrotra, & M. A. Trick (Eds.), *Extending the horizons: Advances in computing, optimization, and decision technologies*. In Operations Research/Computer Science Interfaces Series, 37, 165–181.

Smith, S. L., & Imeson, F. (2017). GLNS: An effective large neighborhood search heuristic for the generalized traveling salesman problem. *Computers & Operations Research*, *87*, 1–19.

Snyder, L. V., & Daskin, M. S. (2006). A random-key genetic algorithm for the generalized traveling salesman problem. *European Journal of Operational Research, 74*, 38–53.

Tasgetiren, M. F., Suganthan, P. N., & Pan, Q. K. (2007). A discrete particle swarm optimization algorithm for the generalized traveling salesman problem. in *Proc. GECCO 2007*, 158–167.

Yang, J., Shi, X., Marchese, M., & Liang, Y. (2008). An ant colony optimization method for generalized TSP problem. *Progress in Natural Science, 18*, 1417–1422.