Genetic Algorithm for the Traveling Salesman Problem using Sequential Constructive Crossover Operator

Article *in* CiiT International Journal of Biometrics and Bioinformatics · March 2010

DOI: 10.14569/IJACSA.2020.0110275 · Source: DOAJ

1 author:

Zakir Hussain Ahmed
Imam Muhammad bin Saud Islamic University
50 PUBLICATIONS **680** CITATIONS

Genetic Algorithm for the Traveling Salesman Problem using Sequential Constructive Crossover Operator

Zakir H. Ahmed zhahmed@gmail.com
Department of Computer Science,
Al-Imam Muhammad Ibn Saud Islamic University,
P.O. Box No. 5701, Riyadh-11432
Kingdom of Saudi Arabia

Abstract

This paper develops a new crossover operator, Sequential Constructive crossover (SCX), for a genetic algorithm that generates high quality solutions to the Traveling Salesman Problem (TSP). The sequential constructive crossover operator constructs an offspring from a pair of parents using better edges on the basis of their values that may be present in the parents' structure maintaining the sequence of nodes in the parent chromosomes. The efficiency of the SCX is compared as against some existing crossover operators; namely, edge recombination crossover (ERX) and generalized N-point crossover (GNX) for some benchmark TSPLIB instances. Experimental results show that the new crossover operator is better than the ERX and GNX.

Keywords: Traveling salesman problem, NP-complete, Genetic algorithm, Sequential constructive crossover.

1. INTRODUCTION

The Traveling Salesman problem (TSP) is one of the benchmark and old problems in Computer Science and Operations Research. It can be stated as:

A network with 'n' nodes (or cities), with 'node 1' as 'headquarters' and a travel cost (or distance, or travel time etc.,) matrix C= [c_{ij}] of order n associated with ordered node pairs (i, j) is given. The problem is to find a least cost Hamiltonian cycle.

On the basis of the structure of the cost matrix, the TSPs are classified into two groups – symmetric and asymmetric. The TSP is symmetric if $c_{ij} = c_{ji}$, $\forall$ i, j and asymmetric otherwise. For an n-city asymmetric TSP, there are $(n-1)!$ possible solutions, one or more of which gives the minimum cost. For an n-city symmetric TSP, there are $\frac{(n-1)!}{2}$ possible solutions along with their reverse cyclic permutations having the same total cost. In either case the number of solutions becomes extremely large for even moderately large n so that an exhaustive search is impracticable.

There are mainly three reasons why TSP has been attracted the attention of many researcher's and remains an active research area. First, a large number of real-world problems can be modeled by TSP. Second, it was proved to be NP-Complete problem [1]. Third, NP-Complete problems are intractable in the sense that no one has found any really efficient way of solving them for large

problem size. Also, NP-complete problems are known to be more or less equivalent to each other; if one knew how to solve one of them one could solve the lot.

The TSP finds application in a variety of situations such as automatic drilling of printed circuit boards and threading of scan cells in a testable VLSI circuit [2], X-ray crystallography [3], etc.

The methods that provide the exact optimal solution to the problem are called exact methods. An implicit way of solving the TSP is simply to list all the feasible solutions, evaluate their objective function values and pick out the best. However it is obvious that this "exhaustive search" is grossly inefficient and impracticable because of vast number of possible solutions to the TSP even for problem of moderate size. Since practical applications require solving larger problems, hence emphasis has shifted from the aim of finding exactly optimal solutions to TSP, to the aim of getting, heuristically, 'good solutions' in reasonable time and 'establishing the degree of goodness'. Genetic algorithm (GA) is one of the best heuristic algorithms that have been used widely to solve the TSP instances.

Since the crossover operator plays a vital role in GA, so many crossover operators have been proposed for the TSP. Goldberg and Lingle [4] defined an operator called PMX (partially mapped crossover), which used two crossover points. The section between these points defines an interchange mapping. This PMX operator was the first attempt to apply GAs to the TSP, in which they found near-optimal solutions to a well-known 33-node problem. The OX (ordered crossover) operator developed by Davis [5] builds offspring by choosing a subsequence of a tour from one parent and preserving the relative order of nodes from the other parent. Another crossover operator, named CX (cycle crossover) operator was proposed by Oliver et al. [6], where offspring are built in such a way that each node (and its position) comes from one of the parents. Whitley et al. [7] proposed edge recombination crossover (ERX) operator that uses an 'edge map' to construct an offspring that inherits as much information as possible from the parent structures. This edge map stores all the connections from the two parents that lead into and out of a node. A crossover operator based on the conventional N-point crossover operator, named as generalized N-point crossover (GNX), was proposed by Radcliffe and Surry [8]. Poona and Carter [9] developed a tie break crossover (TBX), which was then modified by Choi et al. [10] by combining PMX and TBX operators. Moon et al. [11] proposed a new crossover operator named Moon Crossover (MX), which mimics the changes of the moon such as waxing moon → half moon → gibbous → full moon. As reported, performance of MX operator and OX operator is almost same, but OX never reached an optimal solution for all trials.

In this paper, a new crossover operator named sequential constructive crossover (SCX) is developed and accordingly a genetic algorithm based on SCX is developed for solving the TSP.

This paper is organized as follows: Section 2 develops a genetic algorithm based on SCX for the TSP. Section 3 describes computational experiments for three crossover operators. Finally, Section 4 presents comments and concluding remarks.

2. GENETIC ALGORITHMS

Genetic algorithms (GAs) are based essentially on mimicking the survival of the fittest among the species generated by random changes in the gene-structure of the chromosomes in the evolutionary biology [12]. In order to solve any real life problem by GA, two main requirements are to be satisfied:

(a) a string can represent a solution of the solution space, and

(b) an objective function and hence a fitness function which measures the goodness of a solution can be constructed / defined.

A simple GA works by randomly generating an initial population of strings, which is referred as gene pool and then applying (possibly three) operators to create new, and hopefully, better populations as successive generations. The first operator is reproduction where strings are copied to the next generation with some probability based on their objective function value. The second operator is crossover where randomly selected pairs of strings are mated, creating new strings. The third operator, mutation, is the occasional random alteration of the value at a string position. The crossover operator together with reproduction is the most powerful process in the GA search. Mutation

diversifies the search space and protects from loss of genetic material that can be caused by reproduction and crossover. So, the probability of applying mutation is set very low, whereas the probability of crossover is set very high.

2.1. Genetic coding

To apply GA for any optimization problem, one has to think a way for encoding solutions as feasible chromosomes so that the crossovers of feasible chromosomes result in feasible chromosomes. The techniques for encoding solutions vary by problem and, involve a certain amount of art. For the TSP, solution is typically represented by chromosome of length as the number of nodes in the problem. Each gene of a chromosome takes a label of node such that no node can appear twice in the same chromosome. There are mainly two representation methods for representing tour of the TSP – adjacency representation and path representation. We consider the path representation for a tour, which simply lists the label of nodes. For example, let {1, 2, 3, 4, 5} be the labels of nodes in a 5 node instance, then a tour {1→ 3→4→ 2→ 5 →1} may be represented as (1, 3, 4, 2, 5).

2.2. Fitness function

The GAs are used for maximization problem. For the maximization problem the fitness function is same as the objective function. But, for minimization problem, one way of defining a 'fitness function' is as $F(x) = \frac{1}{f(x)}$, where *f*(x) is the objective function. Since, TSP is a minimization problem; we consider this fitness function, where f(x) calculates cost (or value) of the tour represented by a chromosome.

2.3. Reproduction operator

In reproduction/selection process, chromosomes are copied into next generation mating pool with a probability associated with their fitness value. By assigning to next generation a higher portion of the highly fit chromosomes, reproduction mimics the Darwinian survival-of-the-fittest in the natural world. In natural population, fitness is determined by a creature's ability to survive predators, pestilence, and other obstacles to adulthood and subsequent reproduction. In this phase no new chromosome is produced. The commonly used reproduction operator is the proportionate reproduction operator, where a string is selected for the mating pool with a probability proportional to its fitness value. We have considered the stochastic remainder selection method [13] for our genetic algorithms.

2.4. Sequential constructive crossover operator (SCX)

The search of the solution space is done by creating new chromosomes from old ones. The most important search process is crossover. Firstly, a pair of parents is randomly selected from the mating pool. Secondly, a point, called crossover site, along their common length is randomly selected, and the information after the crossover site of the two parent strings are swapped, thus creating two new children. Of course, this basic crossover method does not support for the TSP.

The sequential constructive crossover (SCX) operator constructs an offspring using better edges on the basis of their values present in the parents' structure. It also uses the better edges, which are present neither in the parents' structure. As the ERX and GNX, the SCX does not depend only on the parents' structure; it sometimes introduces new, but good, edges to the offspring, which are not even present in the present population. Hence, the chances of producing a better offspring are more than those of ERX and GNX. A preliminary version of the operator is reported as local improvement technique [14, 15]. The algorithm for the SCX is as follows:

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

Step 2: - Sequentially search both of the parent chromosomes and consider the first 'legitimate node' (the node that is not yet visited) appeared after 'node p' in each parent. If no 'legitimate node' after 'node p' is present in any of the parent, search sequentially the nodes {2, 3, …, n} and consider the first 'legitimate' node, and go to Step 3.

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

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

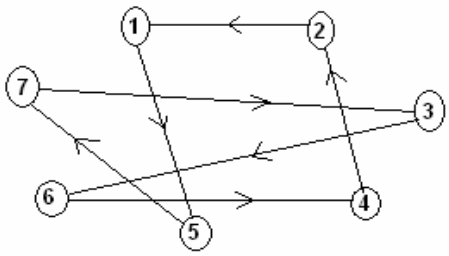
Let us illustrate the SCX through the example given as cost matrix in Table 1. Let a pair of selected chromosomes be P_1: (1, 5, 7, 3, 6, 4, 2) and P_2: (1, 6, 2, 4, 3, 5, 7) with values 312 and 331 respectively.

Node	1	2	3	4	5	6	7
1	999	75	99	9	35	63	8
2	51	999	86	46	88	29	20
3	100	5	999	16	28	35	28
4	20	45	11	999	59	53	49
5	86	63	33	65	999	76	72
6	36	53	89	31	21	999	52
7	58	31	43	67	52	60	999

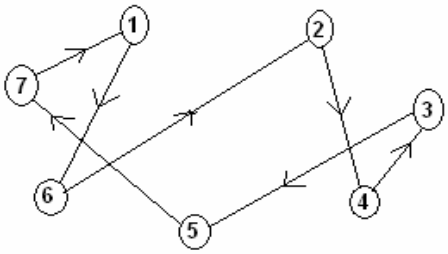
TABLE 1: The cost matrix.

Select 'node 1' as the 1st gene. The 'legitimate' nodes after 'node 1' in P_1 and P_2 are 'node 5' and 'node 6' respectively with c_{15}=35 and c_{16}=63. Since $c_{15} < c_{16}$, we accept 'node 5'. So, the partially constructed chromosome will be (1, 5). The 'legitimate' node after 'node 5' in both P_1 and P_2 is 'node 7'. So, we accept the 'node 7', and the partially constructed chromosome will be (1, 5, 7). The 'legitimate' node after 'node 7' in P_1 is 'node 3', but none in P_2. So, for P_2, we consider the first 'legitimate' node in the set {2, 3, 4, 5, 6, 7}, that is, 'node 2'. Since $c_{72} = 31 < 43 = c_{73}$, we accept 'node 2'. Thus, the partially constructed chromosome will be (1, 5, 7, 2). Again, the 'legitimate' node after 'node 2' in P_1 is none, but in P_2 is 'node 4'. So, for P_1, we consider the first 'legitimate' node in the set {2, 3, 4, 5, 6, 7}, that is, 'node 3'. Since $c_{24} = 46 < 86 = c_{23}$, we accept 'node 4'. So, the partially constructed chromosome will be (1, 5, 7, 2, 4). The 'legitimate' node after 'node 4' in P_1 is none, but in P_2 is 'node 3'. So, for P_1, we consider the first 'legitimate' node in the set {2, 3, 4, 5, 6, 7}, that is, 'node 3'. We accept 'node 3', which will lead to the partial chromosome (1, 5, 7, 2, 4, 3). The 'legitimate' node after 'node 3' in P_1 is 'node 6', but none in P_2. So, for P_2, we consider the first 'legitimate' node in the set {2, 3, 4, 5, 6, 7}, that is, 'node 6'. We accept the 'node 6'. Thus the complete offspring chromosome will be (1, 5, 7, 2, 4, 3, 6) with value 266 which is less than value of both the parent chromosomes. The crossover is shown in Figure 1. The parents are showing as (a) and (b), while (c) is a possible offspring.

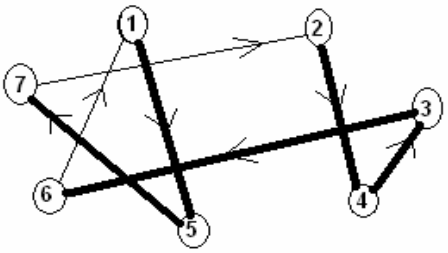
Parents' characteristics are inherited mainly by crossover operator. The operator that preserves good characteristics in the offspring is said to be good operator. The SCX is excellent in preserving good characteristics of the parents in offspring. In Figure 1(c), bold edges are the edges which are present either in first parent or in second parent. Out of seven edges five edges are selected from either of the parents. That is, 71.4 % of edges are selected from parents. The edge (5, 7) is common in both the parents, the edges (1, 5) and (3, 6) are selected from the first parent, while the edges (2, 4) and (4, 3) are from the second parent. The edges (1, 5) and (3, 6) are the 2nd and 3rd minimum edges in the first parent, while the edges (4, 3) and (2, 4) are the 1st and 3rd minimum in the second parent. Also, the new edges (6, 2) and (7, 1) have lesser values. In addition, the SCX can generate a wide variety of offspring.



(a) P1: (1, 5, 7, 3, 6, 4, 2)



(b) P2: (1, 6, 2, 4, 3, 5, 7)



(c) Offspring: (1, 5, 7, 2, 4, 3, 6)

FIGURE 1: Example of Sequential Constructive crossover operator.

2.5. Offspring by two other crossover operators

We consider here two more crossover operators – edge recombination crossover (ERX) and generalized N-point crossover (GNX) for producing offspring using the same pair of parents P_1 and P_2 in section 2.4.

For ERX, the edge table of the example is shown in Table 2. The new offspring is initialized with 'node 1'. The candidates for the next node are 5, 2, 6 and 7. The nodes 5, 2 and 7 have two edges: initial three minus 'node 1', and the 'node 6' has three edges: initial four minus 'node 1'. Node 6 has three edges and thus is not considered. Assume 'node 5' is randomly chosen. Node 5 now has edges to nodes 7 and 3, so 'node 7 is chosen next. Node 7 only has an edge to node 3, so 'node 3' is chosen next. Node 3 has edges to nodes 6 and 4, both of which have two edges left. Suppose the 'node 4' is randomly chosen; then the 'node 4' has edges to the nodes 6 and 2, both of which have one edge left. Next, randomly choose 'node 6', which has an edge to 'node 2'; of course this is the last node to be selected, so 'node 2' is chosen next. The resulting offspring may be (1, 5, 7, 3, 4, 6, 2) with value 323, which is more than one of the parents' value. Here also, five edges are selected from either of the parents, but with higher values.

Node	Edge list	Node	Edge list
1	5, 2, 6, 7	**5**	1, 7, 3
2	4, 1, 6	**6**	3, 4, 1, 2
3	7, 6, 4, 5	**7**	5, 3, 1
4	6, 2, 3		

TABLE 2: The edge table for the parents P_1 and P_2.

For GNX, suppose N=2, consider same parents P_1: (1, 5, 7, **3, 6**, 4, 2) and P_2: (**1, 6, 2**, 4, 3, **5, 7**) and G2X with cross points 3 and 5, where the bold nodes are the ones that would normally be chosen by N-point crossover. Suppose the order in which the segments are tested is (2, 3, 1). Then the 2nd

segment of P_1 will be inserted whole, giving the proto-child (x, x, x, 3, 6, x, x). Nodes in the 3rd segment from P_2 will then be tested in a random order. Both the city 5 and 7 will be accepted, giving the proto-child (x, x, x, 3, 6, 5, 7). The 1st segment of P_2 is then tested, and 1 and 2 will be accepted, giving final proto-child at the end of the 1st phase as (1, x, 2, 3, 6, 5, 7). The untested segments are the visited in random order. Only the 1st segment for P_1 is relevant here. All the nodes are rejected. So, the proto-child at the end of 2nd phase is as (1, x, 2, 3, 6, 5, 7). Since this child is still incomplete, it must be randomly filled up. In this case however, only one legal chromosome has the required node pattern, so the final child may be given by (1, 4, 2, 3, 6, 5, 7) with value 326, which is more than one of the parents' value. Here, four edges are selected from either of the parents with higher values.

From the above analysis, we can draw the conclusion that our SCX gives is better than ERX and GNX.

2.6. Survivor selection

After performing crossover operation survivor selection method is used for selecting next generation population. Traditionally, the survivor selection of GA considers only the fitter chromosomes. The survivor selection of GA considers two kinds of chromosomes for the next generation: (1) parents in current population of size m, and (2) offspring that are generated by crossover of size m. We consider the $(\mu+\lambda)$ survivor selection method that combines chromosomes in (1) and (2), sorts them in ascending order according to their fitness, and considers the first m chromosomes for the next generation. In worst case, all the μ parents in the present generation will survive into the next generation.

2.7. Mutation operator

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

2.8. Control parameters

These are the parameters that govern the GA search process. Some of them are:

(a) Population size: - It determines how many chromosomes and thereafter, how much genetic material is available for use during the search. If there is too little, the search has no chance to adequately cover the space. If there is too much, the GA wastes time evaluating chromosomes.

(b) Crossover probability: - It specifies the probability of crossover occurring between two chromosomes.

(c) Mutation probability: - It specifies the probability of doing bit-wise mutation.

(d) Termination criteria: - It specifies when to terminate the genetic search.

2.9. Structure of genetic algorithms

GAs may be summarized as follows:

```
GA( )
{  Initialize random population;
   Evaluate the population;
   Generation = 0;
   While termination criterion is not satisfied
```

```
  { Generation = Generation + 1;
    Select good chromosomes by reproduction procedure;
    Perform crossover with probability of crossover (Pc);
    Select fitter chromosomes by survivor selection procedure;
    Perform mutation with probability of mutation (Pm);
    Evaluate the population;
  }
}
```

3. COMPUTATIONAL EXPERIMENTS

For comparing the efficiency of the different crossover operators, genetic algorithms using SCX, ERX and GNX have been encoded in Visual C++ on a Pentium 4 personal computer with speed 3 GHz and 448 MB RAM under MS Windows XP, and for some TSPLIB instances. Initial population is generated randomly. The following common parameters are selected for the algorithms: population size is 200, probability of crossover is 1.0 (i.e., 100%), probability of mutation is 0.01 (i.e., 1%), and maximum of 10,000 generations as the terminating condition. The experiments were performed 10 times for each instance. The solution quality is measured by the percentage of excess above the optimal solution value reported in TSPLIB website, as given by the formula

$$Excess(\%) = \frac{Solution\ Value - Optimal\ Solution\ Value}{Optimal\ Solution\ Value}\ x\ 100\,.$$

We report percentage of excess of best solution value and average solution value over the optimal solution value of 10 runs. The table also reports the average time of convergence (in second) by the algorithms.

Instance	n	Opt. Sol.	ERX			GNX			SCX		
			Best (%)	Avg (%)	Avg Time	Best (%)	Avg (%)	Avg Time	Best (%)	Avg (%)	Avg Time
br17	17	39	0.00	0.00	2.10	0.00	0.00	0.26	0.00	0.00	0.11
ftv33	34	1286	1.94	4.98	70.22	15.47	19.49	7.64	0.00	3.58	2.25
ftv35	36	1473	3.19	5.20	76.39	17.58	18.56	1.48	0.00	0.59	9.69
ftv38	39	1530	4.38	5.45	160.87	6.99	13.18	4.03	0.24	0.46	6.89
p43	43	5620	1.44	1.89	213.99	2.46	2.58	22.33	0.05	0.10	22.98
ftv44	45	1613	5.46	6.39	157.23	14.01	15.71	18.46	0.62	0.93	19.22
ftv47	48	1776	5.97	8.48	200.70	20.10	20.38	42.67	0.51	1.73	25.99
ry48p	48	14422	2.04	2.31	185.59	15.18	18.13	39.33	0.59	0.60	25.73
ft53	53	6905	18.03	19.51	122.75	18.29	23.33	29.35	1.03	1.77	36.73
ftv55	56	1608	13.18	14.41	328.59	22.51	24.71	23.74	0.62	1.45	35.11
ftv64	65	1839	25.24	27.48	326.95	25.23	29.87	91.39	0.49	1.54	76.56
ft70	70	38673	10.40	10.56	561.14	6.53	7.81	90.13	0.70	0.86	74.19
ftv70	71	1950	30.41	34.56	432.31	20.72	23.04	135.97	2.15	2.75	58.69
kro124p	100	36230	30.96	37.15	542.57	25.72	28.58	178.64	4.24	4.93	142.02
ftv170	171	2755	62.45	66.15	526.46	51.00	60.69	483.21	6.13	8.93	259.60

TABLE 3: Summary of the results by the crossover operators for asymmetric TSPLIB instances.

Table 3 gives the result for fifteen asymmetric TSPLIB instances of size from 17 to 171. The solution quality of the algorithms is insensitive to the number of runs. Only one instance, br17 of size 17, could be solved exactly by ERX and GNX, whereas three instances, br17, ftv33 and ftv35, could be solved exactly, at least once in ten runs, by SCX. Between ERX and GNX, on the basis of quality of best solution value and average solution value, for the instances from ftv33 to ftv64, ERX is found to be better; but for four instances ft70, ftv70, kro124p and ftv170, GNX is found to be better. That means, as size of the problem increases GNX is found to be better than ERX. It is to be noted that we have

implemented only the original versions of ERX and GNX for the comparative study. On the basis of quality of the solution, as a whole, for all the instances SCX is found to be the best one. On the basis of time of convergence, GNX is found to be better than ERX, and SCX is the best one.

Figure 2 shows performance of different crossover operators for the instance ftv170 (considering only 1000 generations). All crossover operators have some randomized factors, which make them more efficient when trying to copy an allele. The more randomized these operators are, the more possibilities of progress should have. Among them GNX operator has wide range of variations, but it is not the best. Also, ERX operator has some variations, but is the worst. On the other hand, SCX provides us best results. But it has limited range of variations and gets stuck in local minimums quickly.

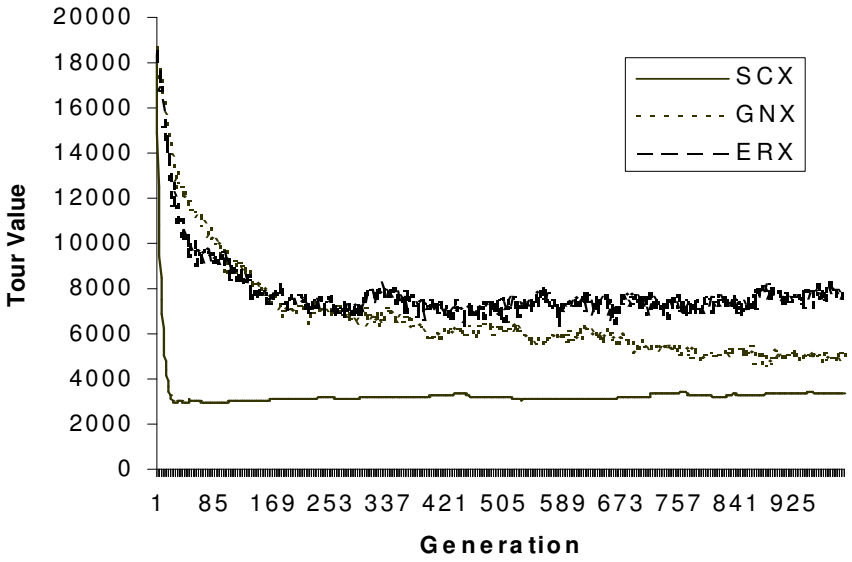


FIGURE 2: Performance of different crossover operators on the instance ftv170.

Instance	n	Opt. Sol.	ERX			GNX			SCX		
			Best (%)	Avg (%)	Avg Time	Best (%)	Avg (%)	Avg Time	Best (%)	Avg (%)	Avg Time
bayg29	29	1610	0.00	0.25	18.11	9.25	10.62	0.65	0.00	0.00	2.19
eil51	51	426	1.41	2.03	157.32	18.78	20.11	20.44	0.00	0.63	4.59
berlin52	52	7542	0.00	3.19	78.57	19.32	22.24	40.11	0.00	0.24	5.10
eil76	76	538	5.20	5.95	286.90	20.07	20.76	141.96	0.00	0.87	128.94
pr76	76	108159	9.08	9.75	230.67	25.13	26.36	142.89	0.11	1.43	131.10
kroA100	100	21282	27.43	32.60	583.55	54.93	68.47	110.55	4.04	4.37	48.75
kroC100	100	20749	40.22	42.25	222.51	54.69	59.02	201.09	1.80	2.77	123.76
eil101	101	629	27.03	27.72	531.80	32.59	32.80	219.37	0.75	1.12	226.42
lin105	105	14379	30.05	33.31	728.44	48.13	50.30	264.80	2.52	2.67	185.90
brg180	180	1950	65.77	74.76	706.00	58.46	59.66	516.69	0.00	0.51	636.67
d198	198	15780	69.04	78.92	870.77	65.71	73.35	304.23	4.09	4.56	542.23

TABLE 4: Summary of the results by the crossover operators for symmetric TSPLIB instances.

We continue this study for some symmetric TSPLIB instances, which is reported in Table 4. The table gives the result for eleven symmetric TSPLIB instances of size from 29 to 198. For these instances also, the solution quality of the algorithms is insensitive to the number of runs. Only one instance, bayg29 of size 29, could be solved exactly by ERX, and none could be solved exactly by GNX,

whereas five instances, bayg29, eil51, berlin52, eil76 and brg180, could be solved exactly, at least once in ten runs, by SCX. Between ERX and GNX, on the basis of quality of best solution value and average solution value, for the instances from bayg29 to lin105, ERX is found to be better; but for two instances brg180 and d198, GNX is found to be better. For these symmetric instances also, as size of the problem increases GNX is found to be better than ERX. On the basis of quality of the solution, for all of these instances also, SCX is found to be the best one.

4. CONCLUSION & FUTURE WORK

We have proposed a new crossover operator named sequential constructive crossover (SCX) for a genetic algorithm for the Traveling Salesman Problem (TSP). We presented a comparative study among SCX, ERX and GNX for some benchmark TSPLIB instances. In terms of quality of the solution, for the less sized instances, ERX is found to be better than GNX. But, as the size increases GNX is found to be better than ERX. Among all the operators, experimental results show that our proposed crossover operator (SCX) is better than the ERX and GNX, in terms of quality of solutions as well as solution times.

In this present study, we only consider the original version of ERX and GNX. Our aim was only to compare the quality of the solutions by different crossover operators. Our aim was not to improve the solution quality by any of the operators. That is why; we do not use any local search technique to improve the solution quality. We do not set high population size and do not consider parallel version of algorithms to obtain exact solution as was done by Whitley et al. [7]. Also, we set here highest probability of crossover to show the exact nature of crossover operators. Mutation with lowest probability is applied just not to get stuck in local minima quickly.

it is very difficult to say that what moderate sized instance is unsolvable exactly by our crossover operator, because, for example, the instance brg180 of size 180 could be solved exactly, at least one in ten runs, whereas ftv38 of size 39 could not be solved within ten runs. So an incorporation of good local search technique to the algorithm may solve exactly the other instances, which is under our investigation.

Acknowledgements

This research was supported by Deanery of Academic Research, Al-Imam Muhammad Ibn Saud Islamic University, Saudi Arabia vide Grant No. 280904.

5. REFERENCES

[1] C.H. Papadimitriou and K. Steglitz. *"Combinatorial Optimization: Algorithms and Complexity"*. Prentice Hall of India Private Limited, India, 1997.

[2] C.P. Ravikumar. *"Solving Large-scale Travelling Salesperson Problems on Parallel Machines"*. Microprocessors and Microsystems 16(3), pp. 149-158, 1992.

[3] R.G. Bland and D.F. Shallcross. *"Large Travelling Salesman Problems arising form Experiments in X-ray Crystallography: A Preliminary Report on Computation"*. Operations Research Letters 8, pp. 125-128, 1989.

[4] D.E. Goldberg and R. Lingle. *"Alleles, Loci and the Travelling Salesman Problem"*. In J.J. Grefenstette (ed.) Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications. Lawrence Erlbaum Associates, Hilladale, NJ, 1985.

[5] L. Davis. *"Job-shop Scheduling with Genetic Algorithms"*. Proceedings of an International Conference on Genetic Algorithms and Their Applications, pp. 136-140, 1985.

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

[7] D. Whitley, T. Starkweather and D. Shaner. *"The Traveling Salesman and Sequence Scheduling: Quality Solutions using Genetic Edge Recombination".* In L. Davis (Ed.) Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York, pp. 350-372, 1991.

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

[9] P. Poon and J. Carter. *"Genetic algorithm crossover operations for ordering applications"*. Computers and Operations Research 22, pp. 135–47, 1995.

[10] I. Choi, S. Kim and H. Kim. *"A genetic algorithm with a mixed region search for the asymmetric traveling salesman problem"*. Computers & Operations Research 30, pp. 773 – 786, 2003.

[11] C. Moon, J. Kim, G. Choi and Y. Seo. *"An efficient genetic algorithm for the traveling salesman problem with precedence constraints".* European Journal of Operational Research 140, pp. 606-617, 2002.

[12] D.E. Goldberg. *"Genetic Algorithms in Search, Optimization, and Machine Learning".* Addison-Wesley, New York, 1989.

[13] K. Deb. *"Optimization For Engineering Design: Algorithms And Examples".* Prentice Hall Of India Pvt. Ltd., New Delhi, India, 1995.

[14] Z.H. Ahmed. "A sequential Constructive Sampling and Related approaches to Combinatorial Optimization". PhD Thesis, Tezpur University, India, 2000.

[15] Z.H. Ahmed and S.N.N. Pandit. *"The travelling salesman problem with precedence constraints".* Opsearch 38, pp. 299-318, 2001.

[16] TSPLIB, http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/