New Mechanism of Combination Crossover Operators in Genetic Algorithm for Solving the Traveling Salesman Problem

Pham Dinh Thanh, Huynh Thi Thanh Binh, and Bui Thu Lam

Abstract. Traveling salesman problem (*TSP*) is a well-known in computing field. There are many researches to improve the genetic algorithm for solving *TSP*. In this paper, we propose two new crossover operators and new mechanism of combination crossover operators in genetic algorithm for solving *TSP*. We experimented on *TSP* instances from *TSP*-Lib and compared the results of proposed algorithm with genetic algorithm (*GA*), which used *MSCX*. Experimental results show that, our proposed algorithm is better than the *GA* using *MSCX* on the min, mean cost values.

Keywords: Traveling Salesman Problem, Genetic Algorithm, Modified Sequential Constructive Crossover.

1 Introduction

The traveling salesman problem is an important problem in computing fields and has many applications in the daily life such as scheduling, vehicle routing, *VLSI* layout design, etc. The problem was first formulated in 1930 and it has been one of the most intensively studied problems in optimization techniques. Until now, researchers have obtained numerous significant results for this problem.

  TSP is defined as following: Let 1, 2, ..., n is the labels of the n cities and $C = [c_{i,j}]$ be an $n \times n$ cost matrix where $c_{i,j}$ denotes the cost of traveling from city i to city j. *TSP* is the problem of finding the n-city closed tour having the minimum cost such that each city is visited exactly once. The total cost A of a tour is.

Pham Dinh Thanh
Tay Bac University, Son La, Vietnam

Huynh Thi Thanh Binh
Hanoi University of Science and Technology, Hanoi, Vietnam

Bui Thu Lam
Le Quy Don Technical University, Hanoi, Vietnam

$$A(n) = \sum_{i=1}^{n-1} c_{i,i+1} + c_{n,1} \qquad (1)$$

TSP is formulated as finding a permutation of *n* cities, which has the minimum cost. This problem is known to be *NP*-hard [1, 2] but it can be applied in many real world applications [13] so a good solution would be useful.

Many algorithms have been suggested for solving *TSP*. *GA* is an approximate algorithm based on natural evolution, which applied to many different types of the combinatorial optimization. *GA* can be used to find approximate solutions for *TSP*.

There are a lot of improvements in *GA* that have been developed to increase the performance in solving the *TSP* such as: optimizing creating initial population [3], improving mutation operator [17], creating new crossover operator [12, 20, 21, 22, 23, 24, 25], combining with local search [4, 6, 7, 8, 18].

In this paper, we introduce two new crossover operators: *MSCX_Radius* and *RX*. We propose new mechanism of combination proposed crossover operators and *MSCX* [25] in *GA* to solve *TSP*. This combination is expected to adapt the changing of population. We experimented on *TSP* instances from *TSP*-Lib and compared the results of proposed algorithm with *GA* which used *MSCX*. Experimental results show that, our proposed algorithm is better than the *GA* using *MSCX* on the min, mean cost values.

The rest of this paper is organized as follows. In section 2, we will present related works. Section 3 and 4 contain the description of our new crossovers and the proposed algorithm for solving *TSP* respectively. The details of our experiments and the computational and comparative results are given in section 5. The paper concludes with section 6 with some discussions on the future extension of this work.

2   Related Work

TSP is *NP*-hard problems. There are two approaches for solving *TSP*: exact and approximate. Exact approaches are based on Dynamic Programming [14], Branch and Bound [2], Integer Linear Programming [21], etc. Exact approaches used to give the optimal solutions for *TSP*. However, these algorithms have exponential running time, therefore they only solved small instances. As M. Held and R. M. Karp [14] pointed out Dynamic Programming takes $O(n^2 \cdot 2^n)$ running time, so that it only solves *TSP* with a small number of the vertices.

In recent years, approximation approaches for solving *TSP* are interested by researchers. These approaches can solve large instances and give approximate solutions near to the optimal solution (sometime optimal). Approximation approaches for solving *TSP* are 2-opt, 3-opt [1], simulated annealing [7, 16], tabu search [7, 16]; nature based optimization algorithms and population based optimization algorithms: genetic algorithm [3, 6, 7, 8, 10, 11, 12, 13, 16, 17, 19, 22, 25], neural networks [15]; swarm optimization algorithms: ant colony optimization [7, 23], bee colony optimization [18].

GA is one of computational model inspired by evolution, which has been applied to a large number of real world problems. *GA* can be used to get approximate

solutions for *TSP*. High adaptability and the generalizing feature of *GA* help to execute the traveling salesman problem by a simple structure.

M. Yagiura and T. Ibaraki [16] proposed *GA* for three permutation problems including *TSP*; and *GA* solving *TSP* uses *DP* in its crossover operator. The experiments are executed on 15 randomized Euclidean instances (5 instances for each $n = 100$, 200, 500). The proposed algorithm [16] could get better solutions than Multi-Local, Genetic-Local and Or-opt when sufficient computational time was allowed. However, the experimental results have been pointed out that, their proposed algorithm is ineffective to compare some heuristics specially designed to the given *TSP*, such as Lin-Kernighan method [1, 16].

In [5], the authors used local search and *GA* for solving *TSP*. The experiments are executed in kroA100, kroB100 and kroC100 instances. The experiments results show that the combination of two genetic operators, *IVM* and *POS*, and 2-opt have better cost for solving *TSP* problem. However, the algorithm took more time to converge to the global optimum than using 3-opt.

In 1997, Bernd Freisleben, Peter Merz [7] proposed Genetic Local Search for the *TSP*. This algorithm used idea of hill climber to develop local search in *GA*. The experiment shows that the best solutions are better than the one in [24] on running time and better on min cost range from $0.46\% \rightarrow 0.21\%$.

Crossover operator is one of the most important component in *GA*, which generates new individual(s) by combining genetic material from two parents but preserving gene from the parents. The researchers have studied many different optimal crossover operators like creating new crossover operators [22, 24], modifying exist crossover operators [20, 21, 23, 25], and hybridizing crossover operators [10].

Sehrawat, M. et al. [20] modified Order Crossover (*OX*). They selected the first crossover point which is the first node of the minimum edge from second chromosome. The experiment was executed on five sample data. The modifying order crossover (*MOX*) could get better solutions than *OX* on two sample data but number of the best solutions is found by *MOX* more than *OX*.

The new genetic algorithm (called *FRAG_GA*) was developed by Shubhra, Sanghamitra and Sankar [21]. There were two new operators: nearest fragment (*NF*) and modified order crossover (*MOC*). The *NF* is used for optimizing initial population. In the *MOC*, the authors performed two changes: length of a substring for performing order crossover is $y = \max\{2, \alpha\}$, where $n/9 \leq \alpha \leq n/7$ (n is the total number of cities) and the length of substring is predefined at any times performing crossover. The experiments are executed in Grtschels24, kroA100, d198, ts225, pcb442 and rat783 instances. The authors compared *FRAG_GA* with *SWAPGATSP* [12] and *OXSIM* (standard *GA* with order crossover and simple inversion mutation) [13]. The experiment results showed that the best result, the average result and computation time of *FRAG_GA* are better than one of *SWAPGATSP*, *OXSIM*.

In [22], the authors proposed an improving *GA* (*IGA*) with a new crossover operator (Swapped Inverted Crossover - *SIC*) and a new operation called Rearrangement. *SIC* creates 12 children from 2 parents then select 10 for applying multi mutation. Finally select 2 best individuals. Rearrangement Operation is applied to all individuals in population. It finds the maximum cost of two adjacent cities then swap one

city with three other cities. The experiments are executed 10 times for each instances (KroA100, D198, Pcb442 and Rat783). The experiments show that, performance of *IGA* is better than the three compared *GAs*.

Kusum and Hadush [23] modified the *OX*. In these proposing crossovers, the positions of cut points or the length of the substrings in both parents are different. The experimented on six Euclidean instances derived from *TSP*-lib (eil51, eil76, kroA100, eil101, lin105 and rat195). Crossover rate is 0.9 and mutation rate is 0.01. The experimental results show that results of one modifying crossover are better than *OX* for six *TSP* instances.

In [24], the authors proposed new crossover operator, Sequential Constructive crossover (*SCX*). The main idea of *SCX* is selecting the edges having less value based on maintaining the sequence of cities in the parents. The experiments are performed in 27 *TSPLIB* instances. Results of experiment show that *SCX* is better than the *ERX* and *GNX* on quality of solutions and solution times.

In 2012, Sabry, Abdel-Moetty and Asmaa [25] proposed new crossover operator, Modified Sequential Constructive crossover (*MSCX*), which is an improvement of the *SCX* [24]. The *MSCX* create an offspring and description as follows:

Step 1: Start from 'First Node' of the parent 1 (i.e., current node p = parent1(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 from parent 1 and parent 2 (the first 'legitimate node' that is not yet visited from parent1 and parent2), 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.

Although a lot of crossovers were developed for solving *TSP*, but each operator has its property, so, in this paper, we propose two new crossover operators and mechanism of combination them with *MSCX* crossover [25]. This scheme is expected to adapt the changing and convergence of population and improve the effectiveness in terms of cost of tour. The proposed algorithm will be presented in the next section.

3   Proposed Crossover Operators

This section introduces two new crossover operators: *MSCX_Radius*, *RX*, which are developed for improving the best solutions and increase the diversity of the population.

3.1 MSCX_Radius Crossover

MSCX_Radius modify the step two of *MSCX* [25]. In *MSCX_Radius*, if no 'legitimate node' after current node, find sequentially r nodes, which are not visited from the parents. Then select the node having the smallest distance to current node. r is parameter of *MSCX_Radius*.
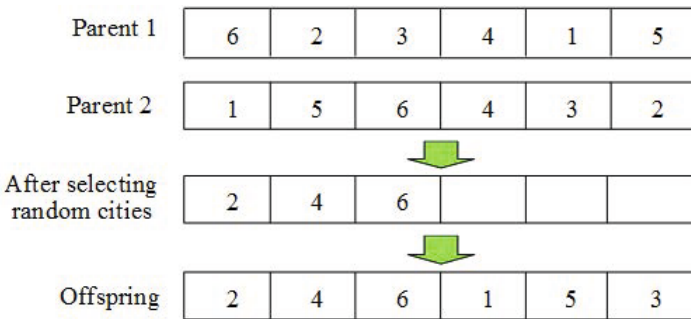
3.2 RX Crossover

This crossover operator is described as following:

Step 1: Randomly select *pr%* cities from the first parent to the offspring.

Step 2: Copy the remaining unused cities into the offspring in the order they appear in the second parent.

Step 3: Create the second offspring in an analogous manner, with the parent roles reverse.

Figure 1 show an example of *RX* crossover operator.

Parent 1	6	2	3	4	1	5
Parent 2	1	5	6	4	3	2
After selecting random cities	2	4	6			
Offspring	2	4	6	1	5	3

Fig. 1 Illustration of the RX crossover operator, pr% = 20%

4 Proposed Mechanism of Combination Two New Crossovers and MSCX

This section proposes new mechanism of combination two propose crossover operators *MSCX_Radius* and *RX* with *MSCX*. We then use apply this mechanism in an improving genetic algorithm (*CXGA*) for solving *TSP*.

The workflow of *CXGA* is described in Fig.2.

The workflow of *HRX* module is shown in Fig.3.

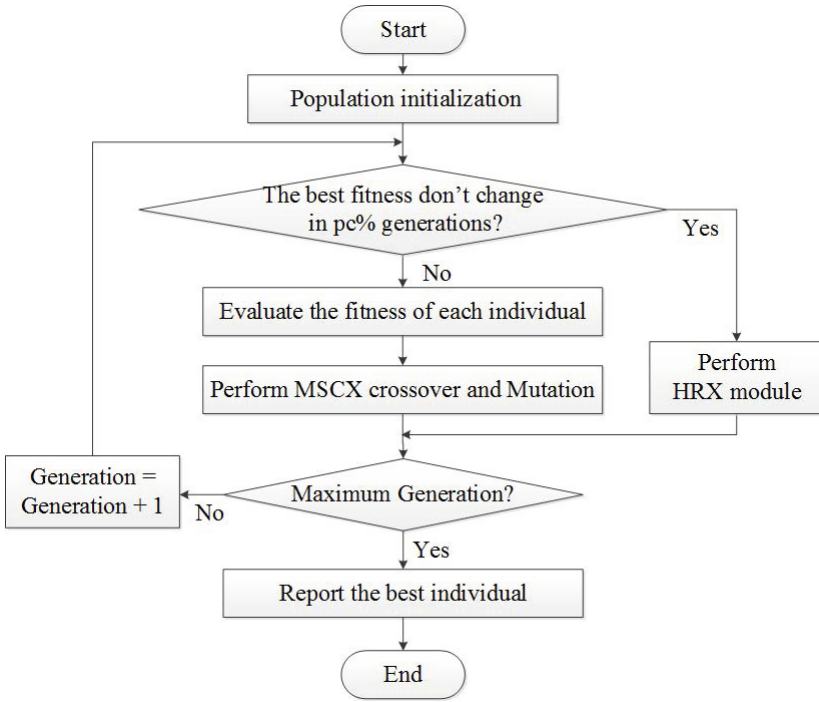In the first part, *prx%* of individuals will be chosen for *RX* crossover and the rest for *MSCX_Radius*

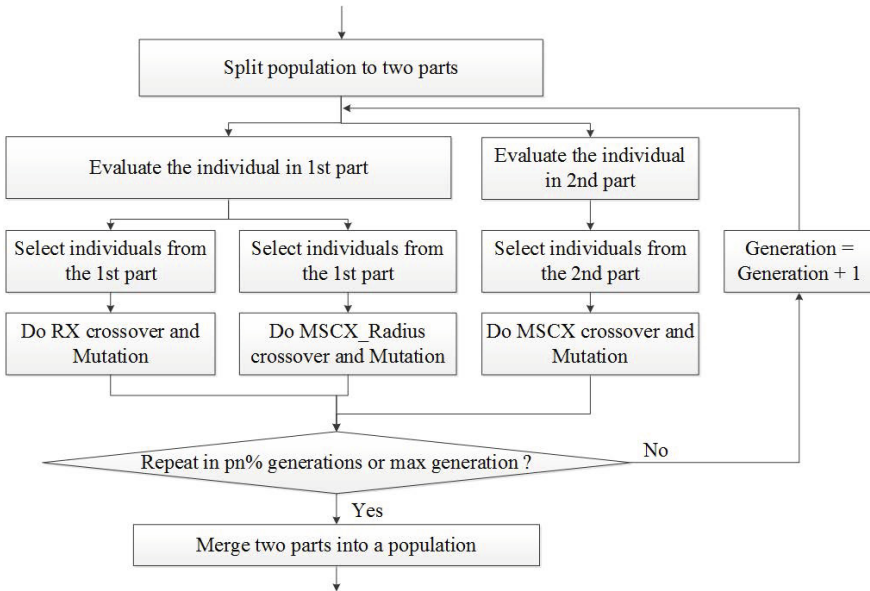**Fig. 2** Structure of improved genetic algorithm

Fig. 3 Structure of HRX module in CXGA

Sketch of the *HRX* module is presented as below:

```
Procedure: HRX (P, prx, pr, r)
Input:  The population P
        r: parameter of MSCX_Radius
        prx: percent of individuals from first part use RX
        pr: percent of number of cities is gotten random in RX
Output: The optimization population P'

Begin
  Split P into two parts: P1 and P2;
  i ← 0; FPᵢ ← P2; SPᵢ ← P1; numInRX ← (prx * |P1|)/100;
  ng ← number of generations perform HRX module;
  While i < ng do
    For j := 1 to numInRX/2 do
      Select random individuals from SPᵢ;
      Do RX(pr)crossover, mutation;
      Add offsprings to SPᵢ₊₁;
    End for
    For j :=1 to |P1| - numInRX do
      Select random individuals from SPᵢ;
      Do MSCX_Radius(r) crossover, mutation;
      Add offspring to SPᵢ₊₁;
    End for
    For j := 1 to |P2| do
      Select random individuals from FPᵢ ;
      Do MSCX crossover, mutation;
      Add offspring to FPᵢ₊₁;
    End for
    i ← i + 1;
  End while
  Merge FPi, SPi into P';
  Return P'
End;
```

5 Computational Results

5.1 Problem Instances

The results are reported for the symmetric *TSP* by extracting benchmark instances from the *TSP*-Lib [9]. The instances chosen for our experiments are eil51.tsp, Pr76.tsp, Rat99.tsp, KroA100, Lin105.tsp, Bier127.tsp, Ts225.tsp, Gil262.tsp, A280.tsp, Lin318.tsp, Pr439.tsp and Rat575.tsp. The number of vertices: 51, 76, 99, 100, 105, 127, 225, 262, 280, 318, 439, 575. Their weights are Euclidean distance in 2-D.

5.2   System Setting

In the experiment, the system was run 10 times for each problem instance. All the programs were run on a machine with Intel Pentium Duo E2180 2.0GHz, 1GB RAM, and were installed by C# language.

5.3   Experimental Setup

This paper implemented two sets of experiments. In the first, we run *GA* using *MSCX_Radius* (named *GA1*), *GA* using *RX* (named *GA2*) and compare with *GA* using *MSCX* [25] (named *GA3*). In the second, we compare the performance of *CXGA* with *GA3*.

When execute the *HRX* module, the population is split into two part. The first one includes the best solutions which uses a combination of *MSCX_Radius* and *RX* crossover; the second includes the rest solutions of population, which uses *MSCX* crossover.

The parameters for experiments are:
Population size: $p_s = 100$
Number of evaluation: 1000000
Mutation rate: $p_m = 1$/number_of_city (chromo length)
Crossover rate: $p_c = 0.9$

5.4   Experimental Resultstitle

The experiments were implemented in order to compare *GA1*, *GA2*, *CXGA* with *GA3* in term of the min, mean, standard deviation values and running times.

For comparing effects of two new crossover operations: *MSCX_Radius* and *RX*. We tested *GA1*, *GA2* with different values of *r*, *pr* parameters. The best results obtain by *GA1*, *GA2* are compared with the ones obtain by *GA3*.

Figure 4 summarizes the mean cost of *GA1* when *r* = 2, 3 and 5 respectively. With *r* = 2, the results found by *GA1* are the best.

The Fig. 5 illustrates the mean cost of *GA2* when *pr%* = 10%, 30% and 50% respectively. The diagrams show that, the mean cost of *GA2* when *pr%* = 10% are better than ones when *pr%* = 30%, 50%.

Experiment results on Fig. 4, Fig. 5 show that *r* = 2 and *pr%* = 10% are the best parameters for *GA1* and *GA2* and they will be selected for comparison with *GA3*.

HRX module was implemented in differences parameters to find the best parameter. The size of the first part is 90%, *pc* =15%, *r* = 5, *pr* = 30%, *pn* = 5%, *prx* = 40%.

In order to select the best value of pc in *HRX* module, we analyzed the correlative of the best solution obtaining from *CXGA* with different values of *pc* parameter (*pc%* = 5%, 10%, 15%, 20%, 30% and 50%).
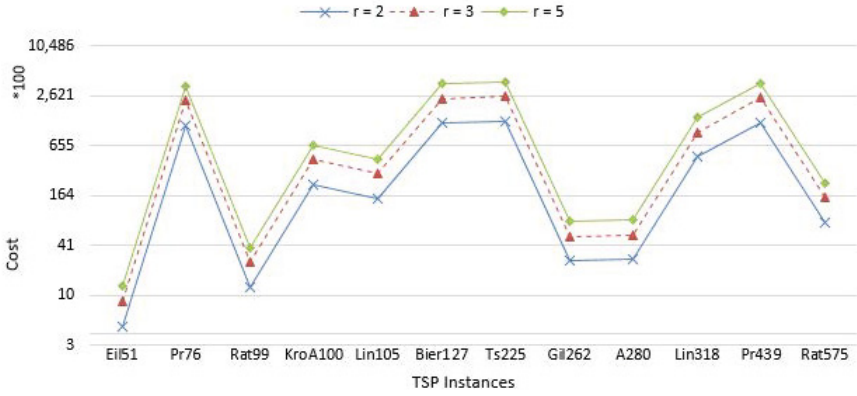
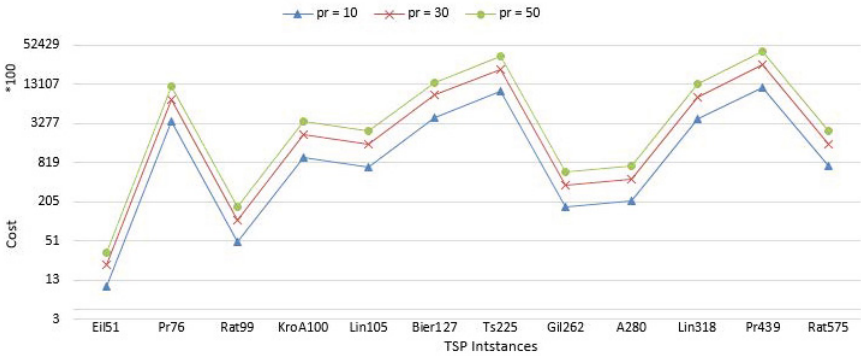**Fig. 4** The mean cost on TSP instances of GA1 when r = 2, 3 and 5

Fig. 5 The mean cost on TSP instances of GA2 when pr% = 10%, 30% and 50%
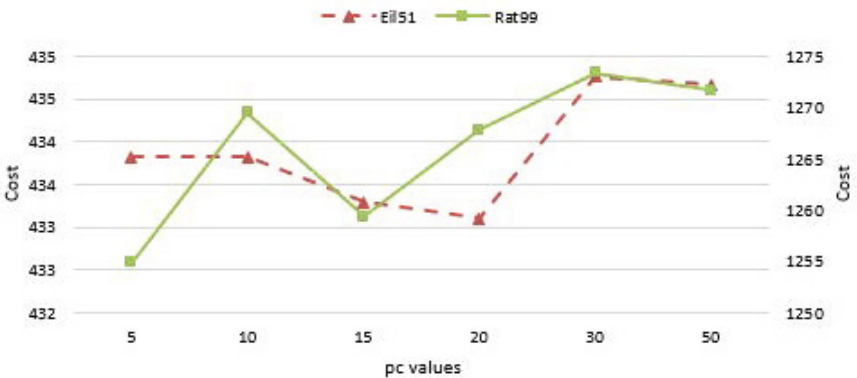

Fig. 6 The relationship between the pc values, mean cost found by 10 running times of CXGA on Eil51, Rat99 instance

Table 1 The results of CXGA found by 10 running times on Eil51, Pr76, Rat99, KroA100, Lin105, Bier127, Ts225, Gil262, A280 when r = 2,3,5,7,10

	Eil51		Pr76		Rat99		KroA100		Lin105		Bier127		Ts225		Gil262		A280	
	Min	Mean	Min	Mean	Min	Mean	Min	Mean	Min	Mean	Min	Mean	Min	Mean	Min	Mean	Min	Mean
r = 2	429.0	432.9	109695.8	111698.7	1242.4	1256.9	21505.5	22036.3	14614.4	14804.5	120634.3	122241.3	127677.2	128737.8	2527.8	2569.2	2672.2	2732.0
r = 3	430.6	439.4	101165.3	111987.0	1239.9	1261.4	21779.4	22119.6	14645.9	14793.3	120303.0	121845.6	127619.0	129039.7	2534.7	2566.8	2664.3	2726.5
r = 5	429.0	432.6	110651.4	111688.5	1239.1	1256.5	21753.5	22014.8	14593.5	14795.9	120819.0	122073.9	128159.5	128985.9	2484.8	2552.4	2666.1	2731.3
r = 7	430.6	433.1	109695.8	111918.3	1241.7	1259.3	21694.8	22068.8	14612.0	14820.2	120846.0	122113.4	127848.3	128992.4	2484.9	2562.7	2659.0	2732.5
r = 10	430.6	434.1	109695.8	111832.7	1235.4	1256.3	21753.5	22133.6	14673.7	14803.7	120846.0	122211.1	128573.6	129016.1	2473.6	2556.8	2672.2	2728.8

Min: Minimum cost; Mean: Mean cost

Table 2 The Results found by GA1, GA2, GA3 and CXGA algorithm for TSP intstances

Instances	NCity	GA1				GA2				GA USING MSCX				CXGA			
		Min	Mean	Std	R_Time	Min	Mean	Std	R_Time	Min	Mean	Std	R_Time	Min	Mean	Std	R_Time
Eil51	51	432.1	435.1	2.9	1.0	925.6	1013.9	50.2	0.4	431.5	435.4	3.7	0.9	429.0	432.6	2.7	1.0
Pr76	76	110919.0	115001.6	2097.7	1.6	315495.9	349169.1	16502.1	0.5	112022.0	113556.0	986.5	1.5	110651.4	111688.5	804.4	1.5
Rat99	99	1244.3	1284.1	20.5	2.2	4555.6	4880.2	205.4	0.6	1249.5	1271.9	20.6	2.1	1239.1	1256.5	15.2	2.1
KroA100	100	21978.1	22329.4	325.8	2.2	94681.5	99235.6	3029.9	0.6	21868.2	22388.6	421.6	2.2	21753.5	22014.8	258.0	2.2
Lin105	105	14751.6	15193.1	440.9	2.4	66635.6	70254.7	2676.8	0.6	14689.9	14966.3	215.3	2.4	14593.5	14795.9	153.9	2.3
Bier127	127	121263.6	122873.1	800.8	3.2	373473.6	395735.0	14681.4	0.6	121610.5	123742.7	985.6	3.1	120819.0	122073.9	622.4	3.0
Ts225	225	129210.9	129925.2	499.4	8.1	977739.8	1025709.4	38644.3	0.8	128282.7	129313.1	524.1	8.1	128159.5	128985.9	501.5	7.6
Gil262	262	2604.3	2653.8	36.4	10.3	16309.9	17031.1	503.5	1.0	2555.2	2615.9	28.1	10.4	2484.8	2552.4	43.2	9.8
A280	280	2745.4	2779.6	25.2	11.6	19353.8	21027.9	710.1	1.0	2669.6	2737.2	45.8	11.7	2666.1	2731.3	55.4	11.0
Lin318	318	47113.3	48039.3	558.1	14.3	362670.9	374143.0	7991.4	1.1	46057.6	46927.8	704.5	14.6	45280.5	46609.7	984.2	13.6
Pr439	439	119595.3	123512.6	1782.9	25.4	1120733.8	1171935.4	44592.6	1.4	116681.5	120078.9	2718.7	26.4	114158.6	117905.1	3265.9	24.3
Rat575	575	7720.2	7858.5	100.3	41.8	70666.9	72422.8	1120.0	1.9	7699.8	7782.0	76.7	44.2	7527.8	7659.5	50.5	39.9

NCity: Number of city; Min: Minimum cost; Mean: Mean cost; Std: Standard Deviation of minimum cost; R_Time: Running time (minutes)

The Fig. 6 shows the dependence between the *pc* values, mean cost values found on 10 running times of *CXGA*. According to the experiments in the Fig. 6, *pc%* = 15% is quite reasonable in our algorithm.

In *MSCX_Radius* crossover, the bigger the *r* parameter is, the more increasing the running times is. In addition, according to the results in the Table 1, the results of *CXGA* when *r* = 5 are better than the ones when *r* = 2, 3, 7 and 10 in most instances on mean and min values (values in bold). So, we chose 5 in all experiments for *r* value.

Table 2 summarizes the results found by *GA3*, *CXGA* and the best results of *GA1*, *GA2* for 12 *TSP* instances of size from 51 to 575.

Mean, min cost value found by *GA1* are worse than *GA3* on 8/12 and 7/12 instances. Standard deviation values found by *GA1* worse than *GA3* on 3 instances. The running time of *GA1* are lower than *GA3* on 3/12 instances. The running time of *GA2* are faster than *GA3* on all instances. Min, mean and standard deviation values found by *GA2* are greater than *GA3* about three times on all instances.

The mean cost values found by *CXGA* algorithm are better than the ones found by *GA3* from 0.2% to 2.4%. The min cost found by *CXGA* are better than the one found by *GA3* from 0.1% to 2.8%. The running time of *CXGA* are faster than the ones found by *GA3* on 11/12 instances. The standard deviation values found by *CXGA* are better than *GA3* on 7/12 instances (values in bold).

6 Conclusion

In this paper, we propose two new crossover operators, called *RX* and *MSCX_Radius*, and new mechanism of combination in *GA* to adapt the convergence of the population for solving *TSP*. We experimented on 12 Euclidean instances derived from *TSP*-lib with the number of vertices from 51 to 575. Experiment results show that, the proposed combination crossover operators in *GA* is effective for *TSP*.

In the future, we are planning to apply propose mechanism of combination to another optimization problem.

References

[1] Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling salesman problem. Operations Research 21, 498–516 (1973)
[2] Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing Natural Computing. Series, 1st edn. Springer (2003)
[3] Snyder, L.V., Daskin, M.S.: A random-key genetic algorithm for the generalized traveling salesman problem. European Journal of Operational Research 174, 38–53 (2006)
[4] Paquete, L., Stützle, T.: A two-phase local search for the biobjective traveling salesman problem. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Deb, K., Thiele, L. (eds.) EMO 2003. LNCS, vol. 2632, pp. 479–493. Springer, Heidelberg (2003)
[5] Neissi, N.A., Mazloom, M.: GLS Optimization Algorithm for Solving Travelling Salesman Problem. In: Second Int. Conf. on Computer and Electrical Engineering, vol. 1, pp. 291–294. IEEE Press (2009)

[6] Bernd, F., Peter, M.: New Genetic Local Search Operators Traveling Salesman Problem. In: Ebeling, W., Rechenberg, I., Voigt, H.-M., Schwefel, H.-P. (eds.) PPSN 1996. LNCS, vol. 1141, pp. 890–899. Springer, Heidelberg (1996)

[7] Freisleben, B., Merz, P.: New Genetic Local Search for the TSP: New Results. In: Int. Conf. on Evolutionary Computation, pp. 159–164. IEEE Press (1997)

[8] Freisleben, B., Merz, P.: A Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman Problems. In: Int. Conf. on Evolutionary Computation, pp. 616–621. IEEE Press (1996)

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

[10] Renders, J.M., Bersini, H.: Hybridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways. In: IEEE World Congress on Computational Intelligence, vol. 1, pp. 312–317. IEEE Press (1994)

[11] Jih, W.-R., Hsu, J.Y.-J.: Dynamic vehicle routing using hybrid genetic algorithms. In: Int. Conf. on Robotics & Automation, vol. 1, pp. 453–458. IEEE Press (1999)

[12] Ray, S.S., Bandyopadhyay, S., Pal, S.K.: New operators of genetic algorithms for traveling salesman problem. In: ICPR 2004, vol. 2, pp. 497–500. Cambridge, UK (2004)

[13] Larranaga, P., Kuijpers, C., Murga, R., Inza, I., Dizdarevic, S.: Genetic algorithms for the traveling salesman problem: A review of representations and operators. In: Artificial Intelligence, vol. 13, pp. 129–170. Kluwer Academic Publishers (1999)

[14] Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. Journal of the Society for Industrial and Applied Mathematics 10, 196–210 (1962)

[15] Haykin, S.: Neural Networks: A Comprehensive Foundation, 2nd edn. Prentice-Hall (1999)

[16] Yagiura, M., Ibaraki, T.: The Use of Dynamic Programming in Genetic Algorithms for Permutation Problems. European Journal of Operational Research 92, 387–401 (1996)

[17] Murat, A., Novruz, A.: Development a new mutation operator to solve the Traveling Salesman Problem by aid of Genetic Algorithms. In: Expert Systems with Applications, vol. 38, pp. 1313–1320. ScienceDirect (2011)

[18] Mersmann, O., Bischl, B., Bossek, J., Trautmann, H., Wagner, M., Neumann, F.: Local search and the traveling salesman problem: A feature-based characterization of problem hardness. In: Hamadi, Y., Schoenauer, M. (eds.) LION 2012. LNCS, vol. 7219, pp. 115–129. Springer, Heidelberg (2012)

[19] Sourav, S., Anwesha, D., Satrughna, S.: Solution of traveling salesman problem on scx based selection with performance analysis using Genetic Algorithm. International Journal of Engineering Science and Technology (IJEST) 3, 6622–6629 (2011)

[20] Sehrawat, M., Singh, S.: Modified Order Crossover (OX) Operator. International Journal on Computer Science & Engineering 3, 2014–2019 (2011)

[21] Ray, S.S., Bandyopadhyay, S., Pal, S.K.: New genetic operators for solving TSP: Application to microarray gene ordering. In: Pal, S.K., Bandyopadhyay, S., Biswas, S. (eds.) PReMI 2005. LNCS, vol. 3776, pp. 617–622. Springer, Heidelberg (2005)

[22] Sallabi, O.M., El-Haddad, Y.: An Improved Genetic Algorithm to Solve the Traveling Salesman Problem. Proceedings of World Academy of Science: Engineering & Technology 52, 530–533 (2009)

[23] Kusum, D., Hadush, M.: New Variations of Order Crossover for Travelling Salesman Problem. Int. Journal of Combinatorial Optimization Problems and Informatics 2, 2–13 (2011)

[24] Zakir, H.A.: Genetic Algorithm for the Traveling Salesman Problem using Sequential Constructive Crossover Operator. Int. Journal of Biometric and Bioinformatics 3, 96–106 (2010)

[25] Abdel-Moetty, S.M., Heakil, A.O.: Enhanced Traveling Salesman Problem Solving using Genetic Algorithm Technique with modified Sequential Constructive Crossover Operator. Int. Journal of Computer Science and Network Security 12, 134–139 (2012)