© International Journal of Combinatorial Optimization Problems and Informatics, Vol. 3, No. 1, Jan-April 2012, pp. 38-60. ISSN: 2007-1558.

Variant of partially mapped crossover for the Travelling Salesman problems

Kusum Deep, Hadush Mebrahtu
Department of Mathematics, Indian Institute of Technology, Roorkee, India
kusumfma@iitr.ernet.in, hhadumk@gmail.com

Abstract. In this paper a variant of partially mapped crossover (VPMX) is designed using cut point positions and is tested for its performance with the existing partially mapped crossover (PMX).In order to test the performance ,two mutation operators are used. These mutation operators are inverted displacement and inversion mutations. Partially mapped crossover (PMX) with inversion and with inverted displacement and a variant of partially mapped crossover (VPMX) with inversion and with inverted displacement are programmed in C++ and implemented on a set of ten benchmark problems taken from the Travelling salesman problem library (TSPLIB). The results indicate that the designed variant of PMX is superior by showing a better performance in eight instances in combination with the inverted displacement mutation. In two instances PMX has obtained a better result. One is PMX with inversion mutation and the other is PMX with inverted displacement mutation.

Keywords: Genetic Algorithm, partially mapped crossover, Travelling salesman problem and variant of partially mapped crossover.

1 Introduction

The traveling salesman problem (TSP) is a significant representative of combinatorial optimization problems. Many practical and real life problems, such as scheduling [1,2], manufacturing control system [3] and bioinformatics [4] can be transformed into the TSP.This makes TSP a popular problem.

Two basic and different classes of TSP can be identified by the properties of a cost matrix. These are symmetric and asymmetric TSP. In symmetric TSP $c_{ij} = c_{jj}$, $\forall$i, j, otherwise this set of problems are referred as asymmetric TSP. If the cities lie in a metric space, satisfying the triangle inequality, the problem is referred as metric TSP. Assuming that each city in a tour is marked by its position (x_i,y_i) in the plane, and the cost matrix C contains the Euclidean distances between the ith and jth city [12]:

$$c_{ij} = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2} \tag{1}$$

Then the problem is both symmetric and metric.

Following Sur-Kolay et al. [5], TSP can be stated mathematically as follows:

If a travelling salesman were to visit a given finite set of cities with cost (c_{ij}) of travelling from city *i* to *j* exactly once and return to its original city, then which tour would obtain the minimum cost? Formally let {1, 2, 3, …, *n*} be labels of the *n* cities, then the total cost TC of a TSP tour is:

Received Aug 10, 2011 / Accepted Sep 20, 2011

$$TC(n) = \sum_{i=1}^{n-1} C_{i,i+1} + C_{n,1} \qquad (2)$$

Or if we have a calculated data in the form of a matrix it can be stated as follows:

Let distances, costs or times taken between *n* cities are stored in a matrix D with elements d_{ij} where *i*; *j* = 1, 2, …, *n* and the diagonal elements d_{ii} are zero. If a tour is represented by a cyclic permutation π of {1, 2, …, *n*} where π (i) represents the city that follows city *i* on the tour, then the traveling Salesman problem is the optimization problem to find a permutation π that minimizes the distance, cost or time of the tour denoted by

$$\sum_{i=1}^{n} d_{i\pi(i)_i}, \quad \pi(n) = 1 \qquad (3)$$

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 large number of possible solutions to the TSP even for problem of moderate size. For practical applications require solving larger problems, 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’. Some of the new heuristic approaches that are being adapted to solve the NP-hard TSP are Particle swarm [6], Ant Colony [7], Simulated Annealing [8] and Genetic algorithm [9].

Genetic algorithms (GAs) [10] are population based search techniques which mimics the principles of natural selection and natural genetics laid by Charles Darwin. In Genetic algorithm, a population of potential solutions termed as chromosomes/individuals are evolved over successive generations using a set of genetic operators called selection, crossover and mutation. First of all, based on some criteria, every chromosome is assigned a fitness value (in our case the above formulation of TSP) and then the selection operator is applied to choose relatively fit chromosomes to be part of the reproduction process. In reproduction process new individuals are created through application of operators. Large number of operators has been developed for improving the performance of GA, because the performance of algorithm depends on the ability of these operators. One of the operators, Crossover operator, blends the genetic information between chromosomes to explore the search space, where as mutation operator is used to maintain adequate diversity in the population of chromosomes and avoid premature convergence.

One of the operators that play a critical role in solving TSP by performing an exchange of information between individuals during generation and helps in obtaining global optimal solution is crossover operator. In the last 20-30 years several crossovers for TSP have been proposed. One of them is the partially mapped crossover [11]. TSP tours can have different representations, such as path, adjacency ordinal and binary matrix representation. From the representations the path representation is the best natural representation of a tour and we will focus on it. For example a tour

2 – 8 – 3 – 7 – 4 – 1 – 9 –6 – 5 is represented simply as
(2 8 3 7 4 1 9 6 5)

In this paper a new variant of partially mapped crossover operator is proposed and its performance over the existing partially mapped crossover is checked. This paper is organized as follows: Section 2 is a literature review on crossover operators for TSP. Section 3 explains the existing and the proposed variant of partially mapped crossover operator. Section 4 describes the mutations used here. In section 5, we will have experimental results for comparison. Finally conclusion will be presented in section 6.

2 Literature Review

The main idea of GAs is to enhance candidate solutions by simulating the mechanisms of natural evolution, such as crossover, selection, and mutation. The operation of crossover is subject to chromosome representation, which can be integer, binary, real, or order representation. Order (or permutation) representation is the most common representation of chromosomes with combinatorial optimization problems that GAs has to tackle. Even though order representation facilitates GAs to handle combinatorial optimization problems, it also causes an intrinsic constraint in the operation of chromosomes, because no duplicate numbers/cities or nodes/ are allowed in a chromosome. Therefore, the crossover for binary-coded GAs, such as one-point, two-point, and uniform crossovers, cannot be directly applied to order-based GAs.

To address the issue of the legality of an order representation, several crossover operators for order-based GAs are proposed. Several heuristic and exact algorithms have been developed in the field of optimization to solve TSP. The exact algorithms [12] are designed to find the optimal solution, that is, the tour of minimal length. They are computationally expensive since they consider all solutions in order to identify the optimum.

From these Branch and bound algorithms are commonly used to find an optimal solution to TSP which are good for less than or equal 70 cities. Running an exact algorithm for hours on a powerful computer is not cost- effective if a solution within a few percent of the optimal can be found quickly on small computers. Accordingly heuristic algorithms are often preferred to exact algorithms to solve TSP. Generally, TSP heuristics can be classified as tour construction procedures [13], tour improvement procedures [14] and composite procedures [15]. In this research we used evolutionary algorithms especially genetic algorithm to solve the travelling salesman problem (TSP).

Evolutionary algorithms are randomized search techniques aimed at stimulating the natural evolution of asexual species according to Fogel et al. [16]. In this model individuals were created via random mutation to the existing individuals. Holland and his students extended this model by allowing “sexual reproduction” i.e. the recombination or crossover of genetic materials from two parents to create a new offspring. These algorithms were Genetic algorithms but they were not designed to solve combinatorial optimization problems like TSP. Consequently improving these algorithms were needed. Off course fitness function with penalty terms and repair operators to transform infeasible solutions to feasible ones were proposed by Richardson et al. [17] to alleviate these problems. However the approaches were designed for very specific application domains and are not relevant in a TSP context. The TSP as opposed to most problems tackled by GAs is a pure ordering problem. Accordingly specialized permutation operators like crossover must be developed for this problem. In Grefenstette et al. [18] a coding scheme for one point crossover was developed and generated feasible offspring, but the sequencing information in the two parent chromosomes is not well transferred to off springs and the resulting search becomes close to a random search. Cycle crossover which focuses on subsets of cities that occupy the same subset of positions in both parents was also proposed [19]. In this crossover a subset of cities are copied from the first parent to the offspring (at the same positions), and the remaining positions are filled with the cities of the second parent. In this way, the position of each city is inherited from one of the two parents. However, many edges can be broken in the process, because the initial subset of cities is not necessarily located at consecutive positions in the parent tours. In relation to order preserving crossover operators the following were proposed.

A modified crossover which is an extension of the one- point crossover for permutation problems is proposed in Davis [20]. By Davis [20] and Oliver et al. [19] an order crossover had been proposed by extending the modified crossover by two cut points and builds offspring by choosing a subsequence of a tour from one parent and preserving the relative order of chromosome from the other parent.

Order based and position based crossover operators were also proposed [21]. For order based crossover, first a subset of cities is selected in the first parent. In the offspring, these cities appear in the same order as in the first parent, but at positions taken from the second parent. Then, the remaining positions are filled with the cities of the second parent. While in position based crossover a subset of positions is selected in the first parent. Then, the cities found at these positions are copied to the offspring (at the same positions). The other positions are filled with the remaining cities, in the same relative order as in the second parent.

Studies Oliver et al. [19] and Starkweather et al. [22] demonstrated that order preserving crossover were mostly superior to operators preserving absolute position. Similarly Alternate edge and heuristic crossover [23], Edge recombination crossover [24] for adjacency representation that uses an edge to construct an offspring that inherits as much information as possible from the parent structures and matrix based crossover [25] as matrix representation have been studied. Moon proposed a new crossover operator named Moon Crossover (MX), which mimics the changes of the moon and the Performance of MX operator and OX (order crossover) operator is reported to be almost same.

Partially mapped crossover (PMX) [11] is one of the most popular and effective crossovers for order-based GAs to deal with combinatorial optimization problems, especially the TSP. In view of the operation, PMX can be regarded as a modification of two-point crossover but additionally uses a mapping relationship to legalize offspring that have duplicate numbers.

Goldberg and Lingle [11] developed partially matched crossover (PMX) that preserves absolute position using two cut points in parents. This operator first randomly selects two cut points on both parents. In order to create an offspring, the substring between the two cut points in the first parent replaces the corresponding substring in the second parent. Then, the inverse replacement is applied outside of the cut points, in order to eliminate duplicates and recover all cities.

Multi parent extension of partially mapped crossover was proposed by Chuan-Kang et al. [26] .In this extension the mapping list and legalization of PMX are modified to deal with the issues that arise from the increase of parents in PMX. Genetic Algorithm for Traveling Salesman Problem using Modified Partially-Mapped Crossover Operator [27] has also tried to reduce the premature convergence that can occur when using PMX only.

3 The partially mapped crossover and its variant

Goldberg and Lingle [11] proposed partially mapped crossover where the cut points of the randomly selected substrings on both parents are at the same position. But here the question is why at the same position? So different positions (as a variant) are taken and the results obtained indicated a better optimal value than the existing partially mapped crossover.

3.1 Existing Partially Mapped Crossover (PMX)

Algorithm

► Substring selection: Cut two substrings of equal size on each parent at the same positions.

► Substring exchange: Exchange the two selected substrings to produce proto-child.

► mapping list determination: Determine the mapping relationship based on the selected substrings.

► Offspring legalization: Legalize proto-child with the mapping relationship.

Example: Let P1 and P2 be parents

P_1: 1 2 5 6 4 3 8 7

P_2: 1 4 2 3 6 5 7 8

Suppose the randomly chosen cut points are as follows

P_1: 1 2 │ 5 6 4 │ 3 8 7

P_2: 1 4 │ 2 3 6 │ 5 7 8

The proto child would be

1 4 │ 5 6 4 │ 5 7 8

This has duplicates 4 and 5.To repair this we use the mapping b/w the substrings inside the cut points as follows

5 $\leftrightarrow$ 2 which implies 2 replaces the duplicate 5 outside the cut points.

6 $\leftrightarrow$ 4 and 3 $\leftrightarrow$ 4 which means 4 can be replaced by 6 or 3, but 6 already exist. Hence after 4 is replaced by 3 the first offspring becomes

O_{11}: 1 3 5 6 4 2 7 8

The second child is created similarly.

3.2 Variant of Partially Mapped Crossover (VPMX)

Algorithm

► Substring selection: Cut two substrings of equal size on each parent at randomly chosen positions.

► Substring exchange: Exchange the two selected substrings to produce proto-child.

► mapping list determination: Determine the mapping relationship based on the selected substrings.

► Offspring legalization: Legalize proto-child with the mapping relationship.

There is no reason to take the cut points at the same positions in both parents. So we will take at different positions in both parents as follows.

Example: Consider the same parents P_1 and P_2 as above.

P_1: 1 2 5 6 4 3 8 7

P_2: 1 4 2 3 6 5 7 8

Using different cut points randomly in both parents for instance

P_1: 1 2 │ 5 6 4 │ 3 8 7

P_2: 1 │ 4 2 3 │ 6 5 7 8

Applying the above procedure/algorithm the first child we get becomes

O_{12}: 1 5 6 4 2 3 7 8

This is different from the above child. The second child can be produced similarly.

Observe that the child (O_{11}) obtained by partially Mapped Crossover and the child (O_{12}) obtained by its variant are completely different.

4 Mutation Operators used

In order to accomplish the experimental results succefully, two mutation operators are used. One is the known Inversion mutation with both PMX and VPMX. The second mutation is the combination of Inversion and Displacement mutations known as Inverted Displacement with PMX and VPMX.

► The Inversion mutation

Inversion Mutation selects two positions within a chromosome/tour at random and then inverts the cities in the substring between these two positions.

Example: Consider the following tour/chromosome (P)

P = (7 6 5 4 3 8 2 9 1)

If a sub tour 5 4 3 is selected at random using two positions

P = (7 6 │5 4 3 │ 8 2 9 1) and inverting the sub tour 4 5 6 the mutated tour (O) will be

O = (7 6 3 4 5 8 2 9 1)

► The Displacement mutation

Displacement Mutation selects a sub tour at random and inserts it at a random position outside the sub tour .Insertion can be viewed as a special case of displacement in which the substring contains only one city.

Example: Consider the following tour/chromosome (P)

P = (9 5 7 3 6 1 2 8 4)

If the randomly selected sub tour is 5 7 3 and the randomly selected insertion position is between 8 and 4,that is

P = (9 5 7 3 6 1 2 8 │ 4) then the mutated tour (O) will be

O= (9 6 1 2 8 5 7 3 4)

► The Inverted Displacement mutation

▪The first step is to select two positions within a chromosome/tour at random and then invert the cities in the substring between these two positions.

▪In the second step the substring obtained by the first step is inserted at a random position outside the substring.

Example: Consider the following tour/chromosome (P)

P = (4 3 9 6 5 2 1 8 7)

In the first step if the randomly selected two positions are between 9 and 6 and 1 and 8

P = (4 3 9 │ 6 5 2 1 │ 8 7) and inverting the substring 6 5 2 1 then we have a chromosome (Q)

Q = (4 3 9 | 1 2 5 6 | 8 7)

In the second step if the randomly selected position for displacement is say between 4 and 3 then the mutated tour (O) would be

O = (4 1 2 5 6 3 9 8 7)

5 Experimental Setup and Results

5.1 Experimental Setup

Parameter tuning [28] is Getting an appropriate combination of parameters occurring in GA and is considered to be highly important and perhaps the difficult work. In case of permutation coded GA parameter tuning is generally more difficult as compared to binary coded GA due to the simple reason that the number of tunable parameters occurring in a permutation coded GA are usually more than that occurring in binary GA. This difficulty also increases in both the permutation and binary GA as we take larger and larger test problems into consideration. In order to overcome this problem an extensive experiment has been carried out for all the four GAs. These four GAs are partially mapped crossover with Inversion mutation (PMXI), partially mapped crossover with Inverted Displacement mutation, variant of partially mapped crossover with Inversion mutation (VPMXI) and variant of partially mapped crossover with Inverted Displacement mutation (VPMXID) .These Genetic Algorithms are tested using ten TSPLIB [29] bench mark test problems. The problems are gr24, swiss42, eil51, eil76, kroA100, eil101, lin105, kroB150, rat195, and gil262. The final parameter values for all the GAs are 0.9 as probability of crossover and 0.01 as probability of mutation. We do not claim that these parameter values are the best for any problem in general. But these values are selected and recommended since they are found to repeatedly giving good results for most of the problems and hence they are appropriate values to choose if we talk about the overall performance of the algorithms in general. Initial populations in all algorithms are obtained by nearest neighbor initialization strategy. The selection is done with elitism of size one i.e. if the best individual of the parent population is better than the best individual in offspring population then it replaces the best individual of offspring population.

For a particular problem and a particular algorithm, a run is said to be successful run here if the best objective function value found in that run lies within 5% accuracy of the best known objective function value of that problem. For each GA 20 independent runs with different seed numbers are taken and execution time of successful runs, the average number of function evaluations, mean and standard deviations are recorded. The maximum number of generations is also fixed to be 30000 for all the GAs. All the algorithms are implemented in C++ and the experiments for all computations are performed on a 1.66 GHz processor PC with 2 GB of RAM.

By calculating the percentage of excess/error/ above the optimal value reported in TSPLIB [29], we measured the quality of solution using the formula given by:

$$\text{Percentage of Error (\%)} = \frac{\text{Value obtained - optimal value}}{\text{Optimal value}} \times 100$$

5.1 Experimental Results

The Experimental results of the GAs are compared and discussed in this section. Analysis on the performance of four algorithms using ten problems from TSPLIB [29] is summarized by graphs and tables below.

Notations of crossover operators used here with corresponding mutations are given below: PMXI - Partially mapped crossover with inversion mutation, PMXID - Partially mapped crossover with Inverted Displacement mutation, VPMXI - Variant of Partially mapped crossover with inversion mutation, VPMXID - Variant of Partially mapped crossover with Inverted Displacement mutation.

Table 1. Experimental Result of gr24 obtained for four algorithms before or at 30000 generations

crossover Operators	**Mutation operators**	**Cost and time /sec**	**Generations**					**Best and %**
			1000	**5000**	**10000**	**20000**	**30000**	
PMX	Inversion	min	1438	1348	1348	1348	1348	1348 (5.09)
		max	2522	2522	2522	2522	2522	
		time	3	10	21	35	53	
	InverDisp	Min	1357	1345	1345	1345	1345	1345 (5.08)
		max	2178	2178	2350	2391	2391	
		time	2	11	18	34	51	
VPMX	Inversion	min	1397	1397	1397	1376	1376	1376 (8)
		max	2833	2833	2833	2833	2833	
		time	2	11	22	39	61	
	InverDisp	min	1337	1317	1317	1317	1317	**1317 (3.5)**
		max	2807	3015	3015	3015	3015	
		time	2	11	21	39	59	

In Table 1 above it is observed that the Variant of Partially mapped crossover with Inverted Displacement mutation (VPMXID) has relatively a better performance than especially the partially mapped crossover with both mutation operators. Its percentage of error is 3.5% from the known optimal value.

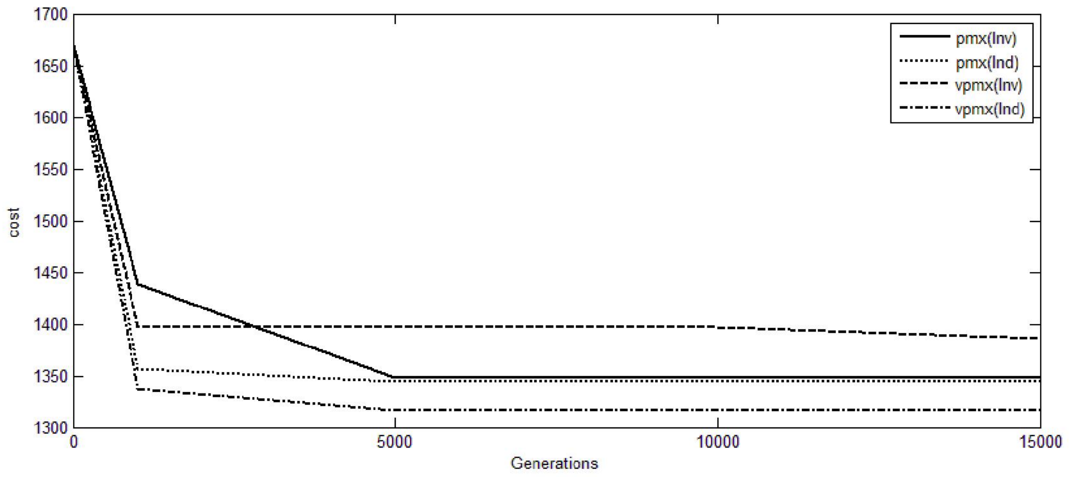

Fig. 1 Comparative convergence of the algorithms for gr24.

For gr24 the Comparative convergence of the four algorithms can be clearly seen in Fig.1 above. After few generations the graph of VPMXID dominates the remaining three algorithms.

Table 2. Experimental Result of swiss42 obtained for four algorithms before or at 30000 generations.

crossover Operators	Mutation operators	Cost and time /sec	Generations					Best and %
			1000	5000	10000	20000	30000	
PMX	Inversion	min	1425	1345	1345	1345	1345	1345 (5.7)
		max	2900	2900	2900	2900	2900	
		time	3	18	34	68	103	
	InverDisp	min	1384	1336	1335	1335	1335	1335 (4.8)
		max	2238	2331	2331	2331	2331	
		time	4	20	36	70	107	
VPMX	Inversion	min	1432	1383	1383	1383	1383	1383 (8.6)
		max	2702	2702	2702	2702	2788	
		time	4	23	43	84	136	
	InverDisp	min	1334	1295	1286	1273	1273	**1273 (0)**
		max	2600	2824	3066	3066	3066	
		time	5	24	45	89	134	

In Table 1 it has been seen the performance of VPMXID to be quite better than the remaining three algorithms. Similarly here in Table 2 VPMXID has obtained the known optimal value which can be found in TSPLIB [29] with 0% error. In the two problems gr24 and swiss42 VPMX has a better performance than the existing PMX.

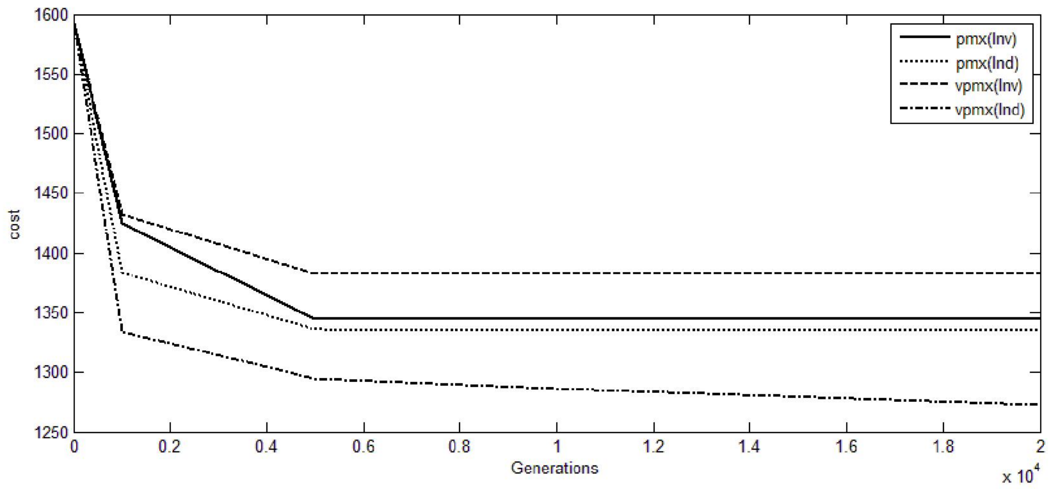

Fig. 2 Comparative convergence of the algorithms for swiss42.

Fig. 2 above shows convergence comparison of the algorithms for swiss42.Here the graph of VPMXID is seen to have a clear superiority over all the remaining algorithms.

Table 3. Experimental Result ofeil51 obtained for four algorithms before or at 30000 generations.

crossover Operators	Mutation operators	Cost and time /sec	Generations					Best and %
			1000	5000	10000	20000	30000	
PMX	Inversion	min	485	452.1	452.1	450.4	448.3	448.3 (5.2)
		max	707.3	707.3	707.3	707.3	707.3	
		time	13	62	124	254	371	
	InverDisp	min	478.8	442.2	442.2	442.2	437.7	437.7 (2.7)
		max	748.1	748.1	771.5	771.5	771.5	
		time	12	60	120	236	359	
VPMX	Inversion	min	491.9	439.3	439.3	439.3	439.3	439.3 (3.1)
		max	996.1	996.1	996.1	996.1	996.1	
		time	14	67	132	264	400	
	InverDisp	min	486.8	446.7	446.7	446.7	436.5	**436.5 (2.4)**
		max	901.5	966.4	966.4	966.4	966.4	
		time	15	72	140	287	403	

Table 3 indicates that a relatively better near optimal value is obtained using VPMXID. VPMX has shown a good performance than PMX upto now for the problems gr24, swiss42 and eil51.

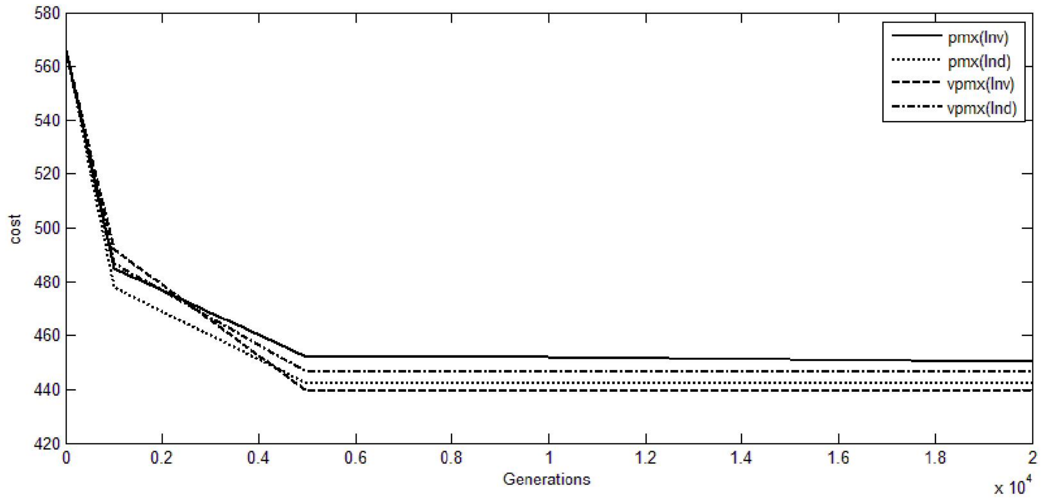

Fig. 3 Comparative convergence of the algorithms for eil51.

Fig. 3 above shows a clear and better convergence of VPMX with Inverted Displacement than PMX with both mutation operators.

Table 4. Experimental Result ofeil76 obtained for four algorithms before or at 30000 generations.

crossover Operators	Mutation operators	Cost and time /sec	Generations					Best and %
			1000	5000	10000	20000	30000	
PMX	Inversion	min	590.8	581.5	581.5	581.5	581.5	581.5 (8)
		max	831.5	831.5	831.5	831.5	831.5	
		time	26	129	256	516	765	
	InverDisp	min	605.2	580.5	568.6	568.6	562.4	562.4 (4.5)
		max	851.7	851.7	851.7	877.7	877.7	
		time	26	129	258	514	775	
VPMX	Inversion	min	599.1	576	576	576	576	576 (7)
		max	1148.2	1148.2	1148.2	1148.2	1148.2	
		time	29	143	286	581	859	
	InverDisp	min	598.7	584.6	584.6	582.5	561.5	**561.5 (4.3)**
		max	1105.5	1105.5	1258.3	1258.3	1258.3	
		time	32	168	279	560	845	

As can be seen in Table 4 VPMXID did not obtain the known optimal value, however its performance is encouraging. For eil76 PMXID has also got a good result having a very small difference with that of VPMXID.But still the better near optimal values of gr24 , swiss42, eil51and eil76 are obtained by Variant of PMX than PMX.

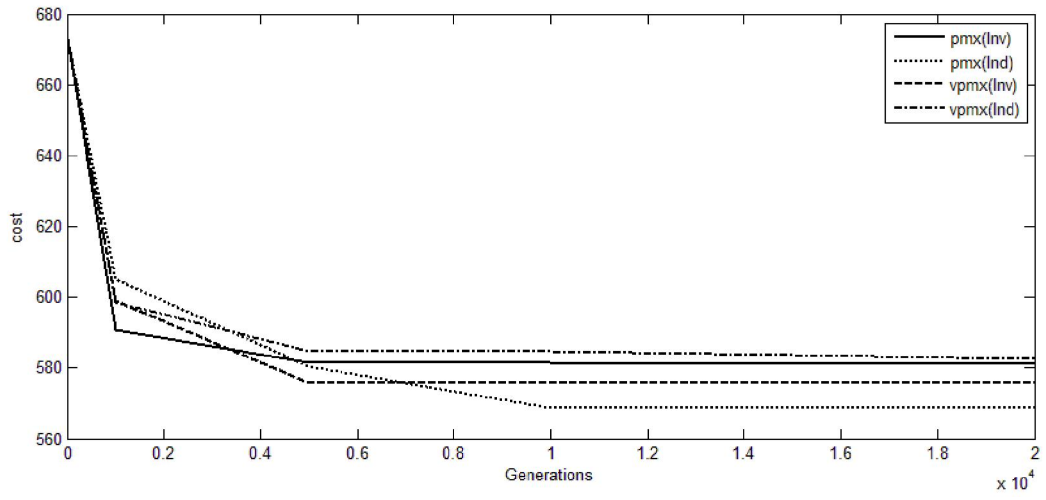

Fig. 4 Comparative convergence of the algorithms for eil76.

Fig. 4 shows Comparative convergence of the algorithms for eil76 up to 20000 generations, however around 30000 generations' graphs of VPMXID and PMXID looks coinciding.

Table 5. Experimental Result of kroA100 obtained for four algorithms before or at 30000 generations.

crossover Operators	Mutation operators	Cost and time /sec	Generations					Best and %
			1000	5000	10000	20000	30000	
PMX	Inversion	min	24698.5	24099.7	23479.5	23427.7	23427.7	23427.7 (10)
		max	38892.3	38892.3	38892.3	38892.3	38892.3	
		time	47	230	477	890	1340	
	InverDisp	min	23652.9	22290.7	21987.5	21987.5	21987.5	21987.5 (3.3)
		max	39778.4	39778.4	39778.4	39778.4	39778.4	
		time	49	233	504	986	1337	
VPMX	Inversion	min	24698.5	24025.9	22969.4	22337	22337	22337 (4.9)
		max	55373.7	55373.7	55373.5	52712.4	52712.4	
		time	38	188	376	998	1487	
	InverDisp	min	24252.1	22328.8	22174.2	21965.8	21965.8	**21965.8 (3.2)**
		max	52298.3	52298.3	52298.3	52298.3	58089	
		time	49	243	483	972	1776	

In Table 5 the performance of VPMXID is still better than the remaining three algorithms. Up to now better results for the problems gr24, swiss42, eil51, eil76 and kroA100 are obtained by Variant of PMX than PMX. For kroA100 PMXID has also got a good result having a small difference with that of VPMXID.

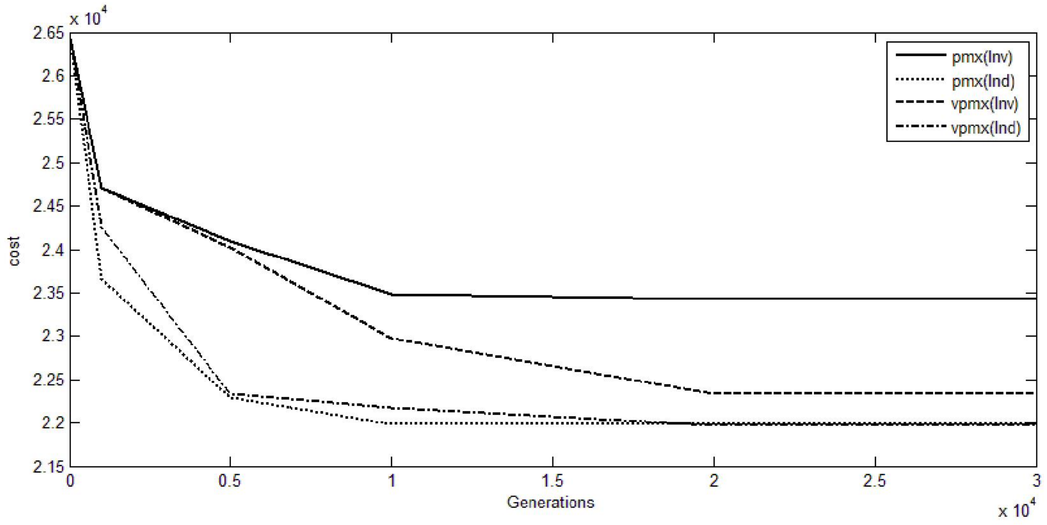

Fig. 5 Comparative convergence of the algorithms for kroA100.

Fig. 5 clearly shows the comparative convergence of the algorithms. After 20000 generations graphs of VPMXID and PMXID looks coinciding.

Table 6. Experimental Result of eil101 obtained for four algorithms before or at 30000 generations.

crossover Operators	Mutation operators	Cost and time /sec	Generations					Best and %
			1000	5000	10000	20000	30000	
PMX	Inversion	min	728.2	705.6	704.1	697.2	686.7	686.7 (9.1)
		max	1017	1017	1017	1017	1017	
		time	47	220	461	919	1310	
	InverDisp	min	720	697.1	676	671.8	671.8	671.8 (6.8)
		max	984.4	984.4	984.4	1040	1040	
		time	47	239	474	953	1459	
VPMX	Inversion	min	721.8	712	696.1	696.1	696.1	696.1 (10.6)
		max	1125	1125	1125	1143.2	1143.2	
		time	48	242	484	994	1495	
	InverDisp	min	724.6	703.5	684.8	680.4	670.4	670.4 (6.5)
		max	1307.9	1307.9	1307.9	1307.9	1307.9	
		time	49	243	488	982	1510	

In the above Table 6 even though the result is not that much encouraging but VPMXID has still shown a relatively better performance than the remaining three algorithms.Here we observe that the relative near optimal values of the problems gr24, swiss42, eil51, eil76, kroA100 and eil101 are obtained using Variant of PMX.

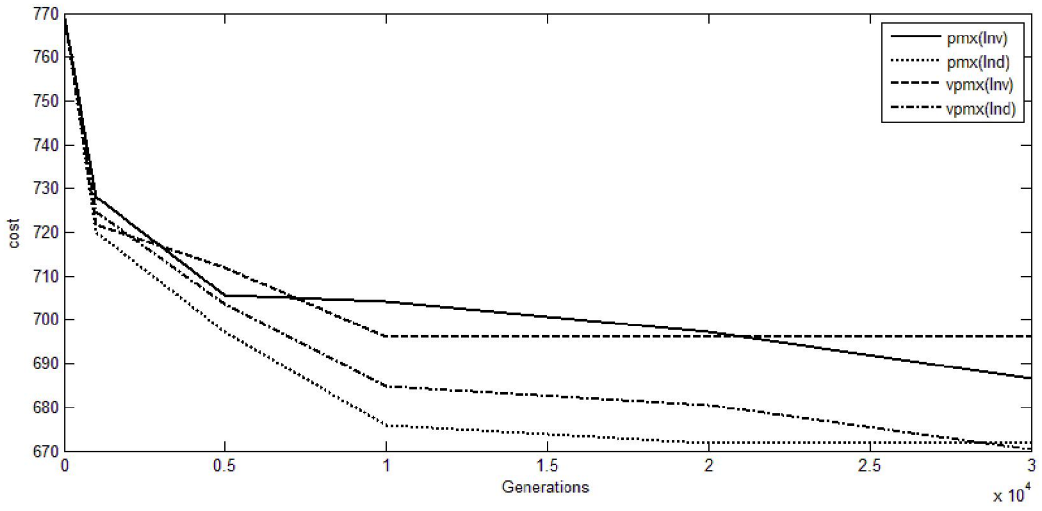

Fig. 6 Comparative convergence of the algorithms for eil101.

Fig. 6 shows the comparative convergence of the algorithms.The computation between VPMXID and PMXID is high after 25000 generations.

Table 7. Experimental Result of lin105 obtained for four algorithms before or at 30000 generations.

crossover Operators	Mutation operators	Cost and time /sec	Generations					Best and %
			1000	5000	10000	20000	30000	
PMX	Inversion	min	16403.3	16285.7	15920.4	15878.5	15878.5	15878.5 (10.4)
		max	26294.5	26294.5	26294.5	26294.5	27171.7	
		time	43	218	435	865	1338	
	InverDisp	min	15613.3	15529.5	15106.1	15089.7	15089.7	15089.7 (4.9)
		max	27065.1	27065.1	27065.1	27065.1	27065.1	
		time	42	211	421	843	1406	
VPMX	Inversion	min	16706.4	16323.5	16317.3	16124.9	16124.9	16124.9 (12.1)
		max	31673.8	35335.9	37180.7	37180.7	37180.7	
		time	48	244	486	976	1457	
	InverDisp	min	16317	15725.1	15254.5	15237	14993.2	**14993.2 (4.2)**
		max	40257.4	40257.4	40257.4	40257.4	40257.4	
		time	48	239	475	952	1438	

As the experimental results of lin105 in Table 7 indicate, the Variant of PMX is still dominant with percentage error of 4.2 from the known optimal value.Here VPMX with Inversion mutation and PMX with Inversion mutation obtained wrost results with percentage error of 12.1 and 10.4 respectively.Any way for the problems gr24, swiss42, eil51, eil76, kroA100, eil101and lin105 VPMXID gave a better value than PMXI and PMXID.

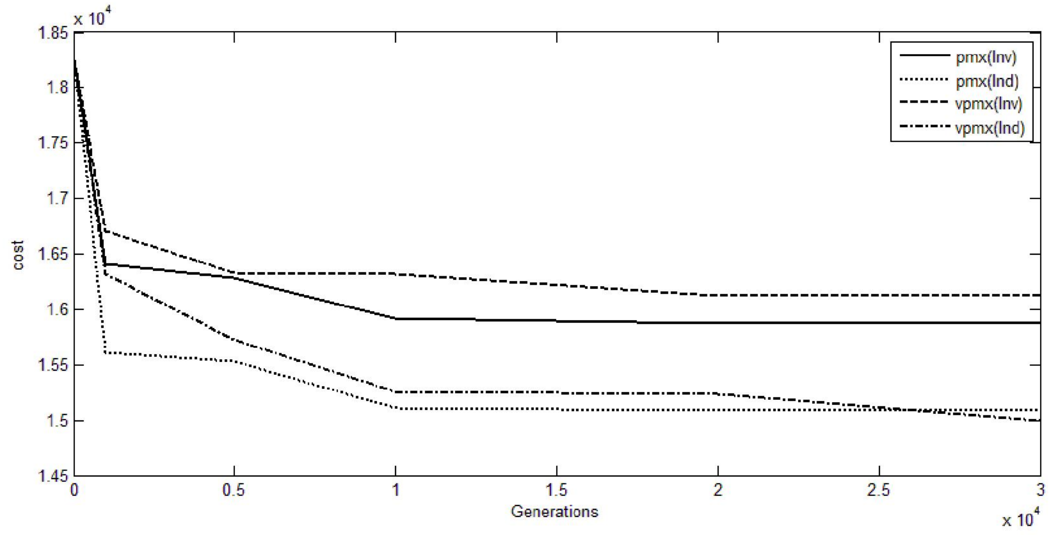

Fig. 7 Comparative convergence of the algorithms for lin105.

In Fig.7 the contest between VPMXID and PMXID indicates that VPMXID dominates PMXID after 25000 generations.

Table 8. Experimental Result of kroB150 obtained for four algorithms before or at 30000 generations.

crossover Operators	Mutation operators	Cost and time /sec	Generations					Best and %
			1000	5000	10000	20000	30000	
PMX	Inversion	min	30087.6	28936.5	28488.5	28327.6	28327.6	28327.6 (8.4)
		max	46935.2	46935.2	46935.2	46935.2	46935.2	
		time	68	337	686	1384	1993	
	InverDisp	min	28888.6	28137.1	27817.8	27639.9	27265.1	27265.1 (4.3)
		max	45014.3	45014.3	45014.3	45014.3	45014.3	
		time	66	329	657	1309	1975	
VPMX	Inversion	min	30878.7	29522.1	28989.6	28545.9	28430	28430 (8.8)
		max	51652.2	51652.2	51652.2	51652.2	51652.2	
		time	74	367	734	1483	2444	
	InverDisp	min	30958.6	29481	29096	27825.4	27181.4	**27181.4 (4)**
		max	63982	63982	63982	63982	63982	
		time	74	368	740	1475	2330	

In Table 8 above it can be clearly observed that VPMXID the best performer in relative to the remaining three algorithms. The percentage of error is 4%.The results of both PMX and VPMX with Inversion mutation are not that much encouraging. The better results for the problems gr24, swiss42, eil51, eil76, kroA100, eil101, lin105 and kroB150 are obtained by VPMXID with Inverted Displacement mutation.

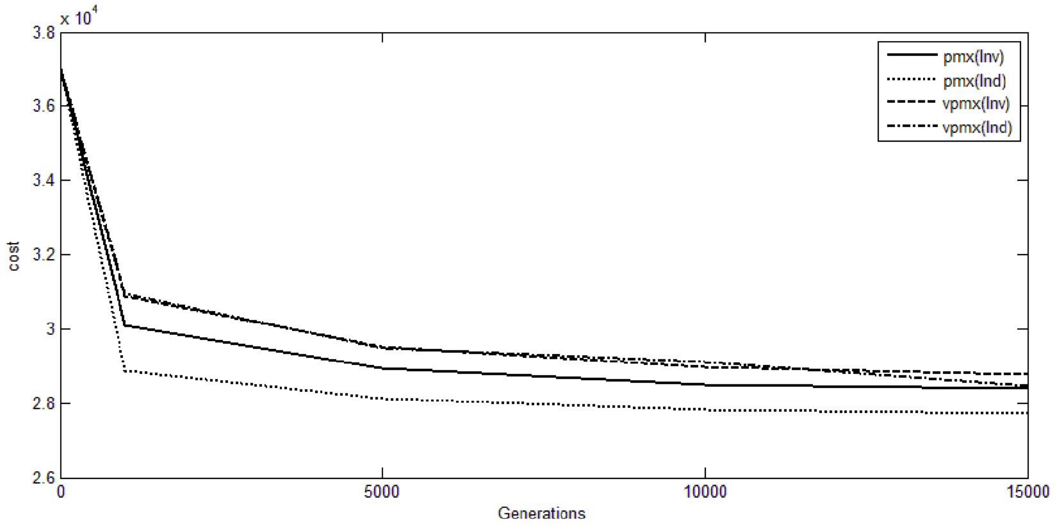

Fig. 8 Comparative convergence of the algorithms for kroB150.

Fig. 8 shows the comparative convergence of the graphs up to 15000 generations because of the gap in results of the graphs. But after 25000 generations VPMXID dominates the remaining and PMXID is second best performer.

Table 9. Experimental Result of rat195 obtained for four algorithms before or at 30000 generations.

crossover Operators	**Mutation operators**	**Cost and time /sec**	**Generations**					**Best and %**
			1000	**5000**	**10000**	**20000**	**30000**	
PMX	Inversion	min	2550.8	2453	2433.2	2415.8	2415.8	**2415.8 (3.9)**
		max	3452	3452	3452	3545.6	3545.6	
		time	68	337	686	1384	1993	
	InverDisp	min	2541	2520.4	2490.9	2460.3	2455.6	2455.6 (5.7)
		max	3728.3	3728.3	3728.3	3763.1	3763.1	
		time	87	438	864	1719	2514	
VPMX	Inversion	min	2564.8	2502.2	2447.1	2447.1	2423.6	2423.6 (4.3)
		max	4261.7	4261.7	4920.9	4920.9	4920.9	
		time	95	476	997	2049	3303	
	InverDisp	min	2628.5	2589.4	2541	2493	2493	2493 (7.3)
		max	4700.4	5057.7	5297.2	5297.2	5297.2	
		time	94	471	934	1881	2853	

What makes the experimental of rat195 different from the above eight experimental Results is that the relatively better near optimal value is obtained by PMX with Inversion mutation.VPMX with Inversion mutation is second best having a percentage error of 4.3.

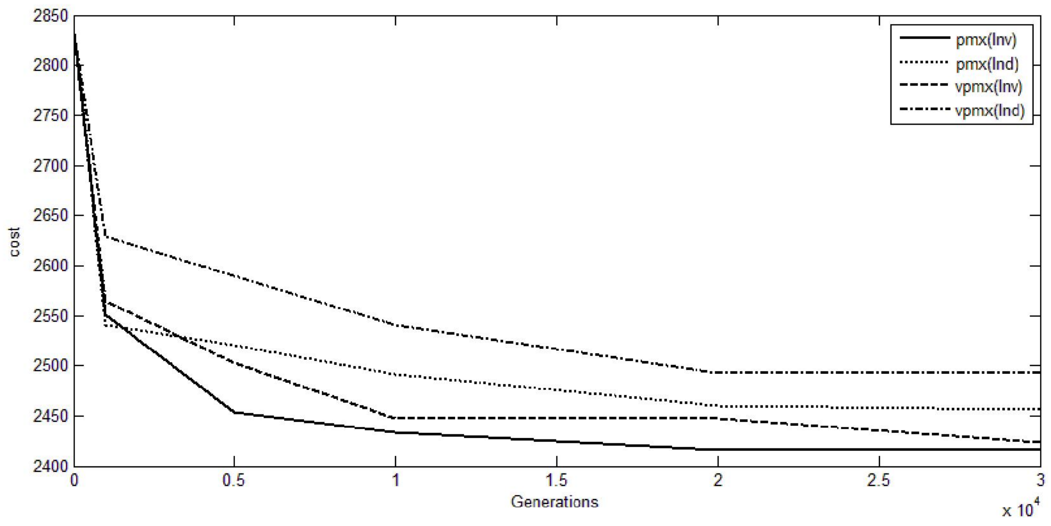

Fig. 9 Comparative convergence of the algorithms for rat195.

Fig. 9 clearly shows a better comparative convergence of PMXI and VPMXI starting from number of generations less than 5000.Here the result of VPMXID is not encouraging.

Table 10. Experimental Result of gil262 obtained for four algorithms before or at 30000 generations.

crossover Operators	Mutation operators	Cost and time /sec	Generations					Best and %
			1000	5000	10000	20000	30000	
PMX	Inversion	min	2877.6	2837.0	2784.4	2697.9	2645.2	2645.2 (11.2)
		max	3899.4	3899.4	3899.4	3899.4	3899.4	
		time	118	576	1140	2277	3402	
	InverDisp	min	2887	2812.3	2752.5	2659	2610.7	**2610.7 (9.7)**
		max	3971.8	3971.8	3971.8	3971.8	3971.8	
		time	115	610	1132	2263	3404	
VPMX	Inversion	min	2887	2883.3	2798.1	2761.3	2715.8	2715.8 (14.1)
		max	5305.8	5305.8	5305.8	5305.8	5305.8	
		time	128	633	1260	2522	3842	
	InverDisp	min	2887	2854.3	2808.6	2771.3	2742.4	2742.4 (15.3)
		max	6196.7	6196.7	6196.7	6196.7	6196.7	
		time	128	640	1270	2557	3850	

In Table 10 the results obtained are different from what we have seen up to now. Even VPMXID has obtained a relatively worst value than the remaining three algorithms. The performance of all algorithms here is not that much encouraging. The smallest percentage of error is 9.7 by PMXID which is a better performer than the remaining algorithms.

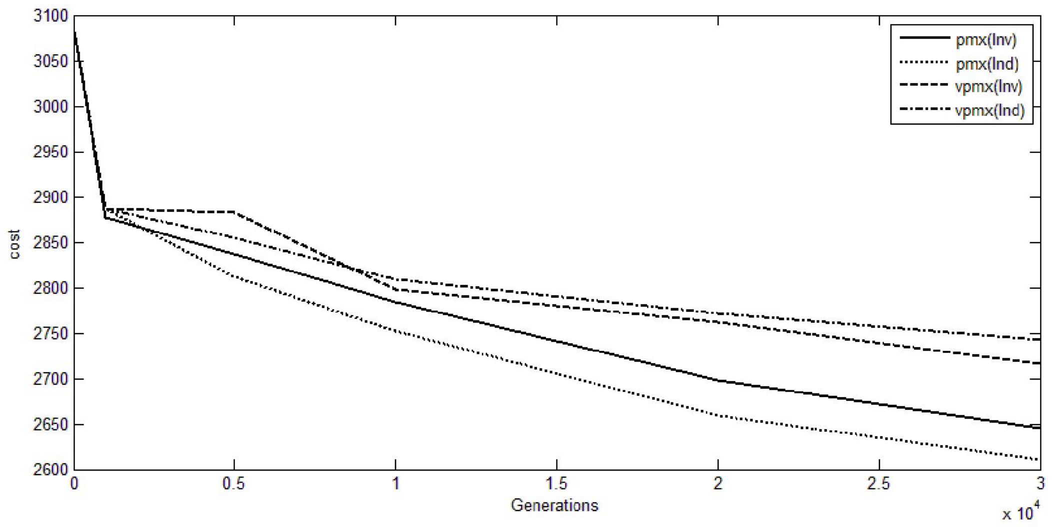

Fig. 10 Comparative convergence of the algorithms for gil262.

In Fig. 10 it can be observed that up to 30000 generations, values obtained by all algorithms are above 2600 which is far from the known optimal value for gil262.The gap in the graphs shows how far their values are different.

In the Tables below Average No. of function evaluation of successful runs, the numbers of successful runs, Average time of execution of successful runs in seconds are summarized. Mean of objective function values of successful runs and Standard Deviation of successful runs out of 20 runs are also reported.

According to Table 11 PMXI and VPMXID have shown 20 successful runs out of 20 runs in four and three instances respectively. In Fig. 11 this can be observed.

For clear comparison of the results the box plots of the number of successful runs, Average No. of function evaluation of successful runs, Average time of execution of successful runs in seconds are also presented.

Table 11. Number of successful runs

problem	**Number of *successful runs* (out of 20 runs)**			
	PMXI	**PMXID**	**VPMXI**	**VPMXID**
Gr24	20	12	16	20
Swiss42	19	12	15	19
Eil51	20	19	19	20
Eil76	19	20	19	20
kroA100	18	14	17	17
Eil101	19	12	13	16
Lin105	20	16	13	14
kroB150	12	13	14	15
Rat195	20	16	18	16
Gil262	19	11	15	12

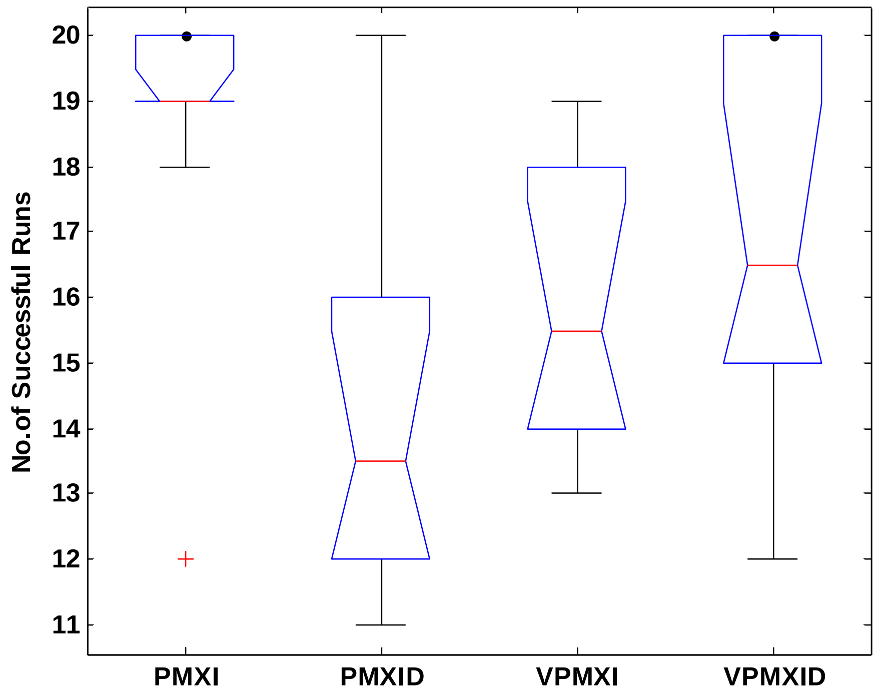

Fig. 11 Performance analysis of the algorithms PMXI, PMXID, VPMXI and VPMXID for successful runs.

In Table 12 it is observed that the average number of function evaluation of successful runs when PMXID is applied it has shown less number of function evaluations in four instances. When VPMXI is applied it has also shown less number of function evaluations in four instances. While PMXI and VPMXID have shown less number of function evaluations in one instance each.

But this doesn't imply that the optimal values obtained by PMXID and VPMXI are better than that of PMXI and VPMXID as observed in Tables 1-10.

In Fig. 12 the box plot clearly shows the average number of function evaluation of successful runs of ten problems.

Table 12. Average number of function evaluation of successful runs

problem	**Average No. of *function evaluation* of successful runs**			
	PMXI	**PMXID**	**VPMXI**	**VPMXID**
Gr24	877505.1	876926.9	876207.4	878173.3
Swiss42	1559411	1557623	1560019	1558227
Eil51	2830538	2830236	2831817	2832523
Eil76	4542566	4542294	4542559	4543608
kroA100	6400052	6400717	6400072	6400627
Eil101	6420379	6420120	6421501	6420468
Lin105	6502424	6502694	6502301	6501409
kroB150	7650299	7650486	7650257	7650302
Rat195	9203167	9203279	9203101	9203999
Gil262	2263808	2263892	2263709	2263987

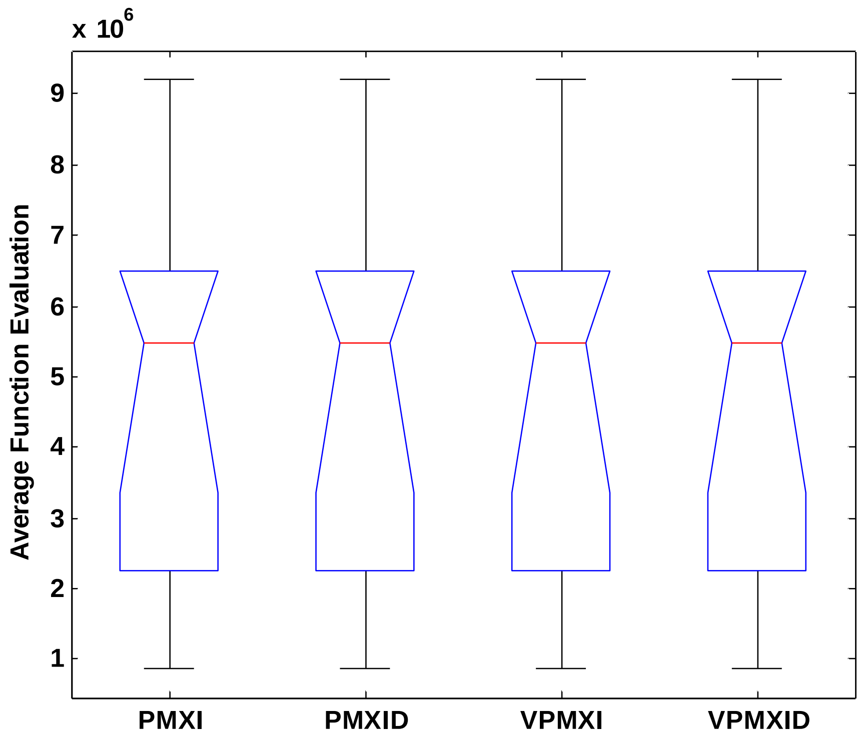

Fig.12 Performance analysis of the algorithms PMXI, PMXID, VPMXI and VPMXID for Average function evaluation.

In Table 13 the genetic algorithm that used VPMXID performed well by giving the smallest Average time of execution of successful runs in six instances. PMXI performed well by giving the smallest Average time of execution of successful runs in two instances. It is also described by the plot box in Fig. 13.

Table 13. Average time of execution of successful runs

problem	**Average *time* of execution of successful runs in seconds**			
	PMXI	**PMXID**	**VPMXI**	**VPMXID**
Gr24	49.1	55.3	50.8	48.8
Swiss42	105.7	108	104.5	102
Eil51	375	368.1	372.3	373.7
Eil76	768.6	770.9	769.5	760.2
kroA100	1345.4	1350.1	1341	1335.5
Eil101	1326.6	1342.5	1314.4	1309.9
Lin105	1342.1	1339.4	1348.5	1337
kroB150	1992.3	2001.1	1994.2	1992.4
Rat195	2677	2680.3	2674.5	2681.2
Gil262	3399.5	3401.8	3407.2	3402.5

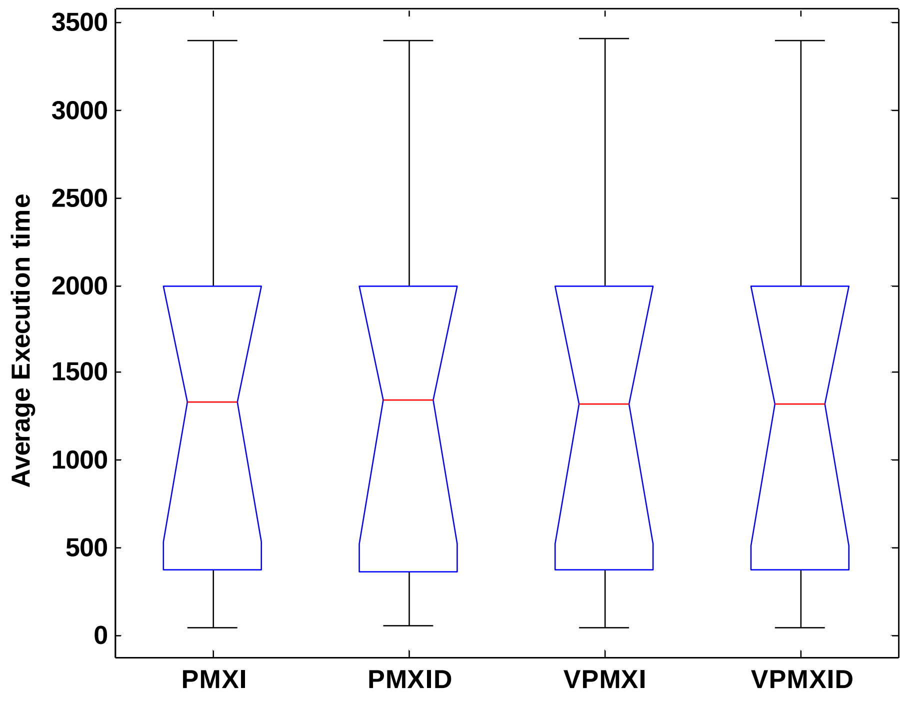

Fig. 13 Performance analysis of the algorithms PMXI, PMXID, VPMXI and VPMXID for Average execution time of successful runs.

In the following Tables 14-15 the mean and standard deviation of objective function values of successful runs are presented.

Table 14. Mean of objective function values of successful runs

problem	*Mean* of objective function values of successful runs			
	PMXI	**PMXID**	**VPMXI**	**VPMXID**
Gr24	1349.1	1348	1340.2	1299.7
Swiss42	1345.5	1339.1	1348.4	1299
Eil51	441.3	438.7	439	436.1
Eil76	576.1	559.8	566	557.2
kroA100	22874.2	21988.1	22006.6	21961.4
Eil101	676.7	669.3	685	664.2
Lin105	15846.2	15058.6	16100.9	14797.1
kroB150	28036.5	27143.3	28229.4	26992.8
Rat195	2403.3	2434.7	2412.5	2471.5
Gil262	2625.6	2589.8	2703.1	2691.4

Table 15. Standard deviation of objective function values of successful runs

problem	*Standard Deviation* of successful runs			
	PMXI	**PMXID**	**VPMXI**	**VPMXID**
Gr24	34.1	33.7	30.4	26.8
Swiss42	79.2	75.5	69.8	26.4
Eil51	5.5	4.6	4.4	2.9
Eil76	3.8	2.4	3.1	1.9
kroA100	174.4	159.1	137.3	112.8
Eil101	9.6	2.4	3.6	1.1
Lin105	67.7	50.1	78.2	41.6
kroB150	199.8	180.1	167.5	127.3
Rat195	23.4	20.9	10.7	9.2
Gil262	13.7	12.9	19.7	15.6

6 Conclusion

The main and basic objective of this paper is try to check the performance of a designed variant of partially mapped crossover and compare its performance with the existing partially mapped crossover by solving known TSPLIB [29] problems. Ten TSPLIB [29] are tested to check the performance of partially mapped crossover and variant of partially mapped crossover. In order to make sure the experimental are fair two mutation operators which are tested to be superior in are used. These are Inversion and Inverted Displacement mutations.

Here we are simply comparing the existing partially mapped crossover and its variant only using Inversion and Inverted Displacement mutations.

As the results of testing four algorithms in ten problems indicate, the new variant of partially mapped crossover with Inverted Displacement mutation [31] outperforms the existing partially mapped crossover. This is checked by winning in eight instances out of ten problems. It seems promising and anyone who had been using PMX can now use its variant VPMX.As can be seen in [30] the performance of partially mapped crossover or its variant are not comparable with the new variations of order crossover

This is because the relative best results of all six TSPLIB[29] problems which are common to both are obtained by the new variations of order crossover .Finally even if the performance of the variant of partially mapped crossover is better than the existing partially mapped crossover in the experimental results we see, in the future it has to be checked by moderate and large size instances of TSPLIB [29]. In addition to this the two points cut positions can be extended to four cut point positions as the mapping relation remains like the existing partially mapped crossover operator.

7 References

[1] Ho, W. and Ji, P.: An integrated scheduling problem of PCB components on sequential pick-and-place machines: Mathematical models and heuristic solutions. Expert Systems with Applications, Vol. 36, No. 3P2 (2009) 7002-7010.

[2] Pan, J. and Huang, H.: A hybrid genetic algorithm for no-wait job shop scheduling problems. Expert Systems with Applications Vol. 36, No. 3P2 (2009) 5800-5806.

[3] Skliarova, L. and Ferrari, A.: FPGA-based implementation of genetic algorithm for the traveling salesman problem and its industrial application, Lecture notes in computer science Vol. 2358, Springer (2002) 77-87.

[4] Ezziane, Z.: Applications of artificial intelligence in bioinformatics: A review. Expert Systems with Applications Vol. 30, No. 1 (2006) 2-10.

[5] Sur-Kolay, S., Banerjee, S. and Murthy, C.A.: Flavours of Travelling Salesman problem In VLSI Design. 1st Indian international conference on Artificial Intelligence, Hyderabad (2003).

[6] Kennedy, J. and Eberhart, R.C.: Swarm Intelligence. Morgan Kauffman publishers, San Francisco (2001).

[7] Dorigo, M. and Gambardella, L.M.: Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1 (1997) 53-66.

[8] Laarhoven, P.V. and Aarts, E.H.L.: Simulated Annealing: Theory and Applications, Kluwer Academic (1987)

[9] Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution programs, Berlin: Springer-Verlag (1992).

[10] Goldberg, D.E.: Genetic Algorithms in Search. Optimization and Machine Learning. Addison-Wesley (1989).

[11] Goldberg, D.E. and Lingle, R.: Alleles, Loci and the Traveling Salesman Problem. Proceedings of the First International Conference on Genetic Algorithms (1985) 154-159.

[12] Laporte, G.: The Traveling Salesman Problem: An Overview of Exact and Approximate Algorithms. European Journal of Operational Research, Vol. 59, No. 2 (1992) 231-247.

[13] Rosenkrantz, D.J., Sterns, R.E. and Lewis, P.M.: An Analysis of several Heuristics for the Traveling Salesman Problem. SIAM Journal on Computing, Vol. 6, No. 3 (1977) 563-581.

[14] Lin, S. and Kernighan, B.: An Effective Heuristic Algorithm for the Traveling Salesman Problem. Operations Research, Vol. 21, No. 2 (1973) 498-516.

[15] Gendreau, M., Hertz, A. and Laporte, G.: New Insertion and Post-Optimization Procedures for the Traveling Salesman Problem. Operations Research, Vol. 40, No. 6 (1992) 1086-1094.

[16] Fogel, L.J., Owens, A.J. and Walsh, M.J.: Artificial Intelligence through Simulated Evolution. John Wiley (1966).

[17] Richardson, J.T., Palmer, M., Liepins, G.E. and Hilliard, M.: Some Guidelines for Genetic Algorithms with Penalty Functions. Proceedings of the Third International Conference on Genetic Algorithms (1989) 191-197.

[18] Grefenstette, J., Gopal, R., Rosmaita, B.J. and Gucht, D.V.: Genetic Algorithms for the Traveling Salesman Problem. Proceedings of the First International Conference on Genetic Algorithms (1985) 160-168.

[19] Oliver, I.M., Smith, D.J and Holland, J.R.C.: A Study of Permutation Crossover Operators on the Traveling Salesman Problem. Proceedings of the Second International Conference on Genetic Algorithms (1987) 224-230.

[20] Davis, L.: Applying Adaptive Algorithms to Epistactic Domains. Proceedings of the International Joint Conference on Artificial Intelligence – IJCAI85, Vol. 1 (1985) 162-164.

[21] Syswerda, G.: Schedule Optimization using Genetic Algorithms. Handbook of Genetic Algorithms, L. Davis Eds, Van Nostrand Reinhold (1990) 332-349.

[22] Starkweather, T., McDaniel, S., Mathias, K., Whitley, D. and Whitley, C.: A Comparison of Genetic Sequencing Operators. Proceedings of the Fourth International Conference on Genetic Algorithms (1991) 69-76.

[23] Grefenstette, J.: Incorporating Problem Specific Knowledge into Genetic Algorithms. Genetic Algorithms and Simulated Annealing, L. Davis Eds, Morgan Kaufmann (1987) 42-60.

[24] Whitley, L.D.: The Genitor Algorithm and Selection pressure: Why Rank-Based Allocation of Reproductive Trials is Best. Proceedings of the Third International Conference on Genetic Algorithms (1989) 116-121.

[25] Homaifar, A., Guan, S. and Liepins, G.: A New Approach on the Traveling Salesman Problem by Genetic Algorithms. Proceedings of the Fifth International Conference on Genetic Algorithms (1993) 460-466.

[26] Chuan-Kang, T., Chien-Hao, S. and Chung-Nan L.: Multi-parent extension of partially mapped crossover for combinatorial optimization problems, Expert Systems with Applications Vol. 37, No. 3 (2010) 1879-1886.

[27] Singh, V. and Choudhary, S.: Genetic Algorithm for Traveling Salesman Problem: Using Modified Partially-Mapped Crossover Operator. Multimedia, Signal Processing and Communication Technologies (2009) 20-23.

[28] Kusum Deep and Manoj Thakur: A Real Coded Multi Parent Genetic Algorithm for Function Optimization. Journal of Hybrid Computing Research, Vol. 1, No. 2 (2008) 67-83.

[29] TSPLIB: (http://www.iwr.uniheidelberg.de/groups/comopt/software/TSPLIB95/)

[30] Deep Kusum and Hadush Mebrahtu. New Variations of Order Crossover for Travelling Salesman Problem. International Journal of Combinatorial Optimization problems and Informatics Vol. 2, No. 1 (2011) 2-13.

[31] Kusum Deep and Hadush Mebrahtu. Combined Mutation Operators of Genetic Algorithm for the Travelling Salesman problem. International Journal of Combinatorial Optimization problems and Informatics Vol. 2, No. 3, (2011) 1-23.