ISSN: 2229-6956 (ONLINE)
DOI: 10.21917/ijsc.2015.0150

CROSSOVER OPERATORS IN GENETIC ALGORITHMS: A REVIEW

A.J. Umbarkar[1] and P.D. Sheth[2]
[1]*Department of Information Technology, Walchand College of Engineering, India*
E-mail: anantumbarkar@rediffmail.com
[2]*Department of Master of Computer Applications, Government College of Engineering, Karad, India*
E-mail: pranalisheth@gmail.com

Abstract

The performance of Genetic Algorithm (GA) depends on various operators. Crossover operator is one of them. Crossover operators are mainly classified as application dependent crossover operators and application independent crossover operators. Effect of crossover operators in GA is application as well as encoding dependent. This paper will help researchers in selecting appropriate crossover operator for better results. The paper contains description about classical standard crossover operators, binary crossover operators, and application dependant crossover operators. Each crossover operator has its own advantages and disadvantages under various circumstances. This paper reviews the crossover operators proposed and experimented by various researchers.

Keywords:

Evolutionary Algorithm, Genetic Algorithm, Crossover, Genetic Operators

1. INTRODUCTION

Genetic algorithm is a method of searching. It searches a result equal to or close to the answer of a given problem. New generation of solutions is created from solutions in previous generation. Basic strategy used in GA to create the best solutions/offspring is to crossover the parent genes. Various crossover techniques are built to get the optimum solution as early as possible in minimum generations. The selection of crossover operator has more impact on the performance of GA. The premature convergence [38] in GA can be avoided by selecting appropriate breeding operators. In this paper, the crossover operators are classified in three categories such as standard crossovers, binary crossovers and real/tree crossover s which are application dependant.

The Section 2 explains standard crossovers, which are application independent. Section 3 explains the binary crossovers with some modified crossovers to improve performance of GA. Section 4 explains the application dependant crossovers (real/tress). Section 5 discusses the findings of this review work.

2. STANDARD CROSSOVERS

2.1 1-POINT CROSSOVER

It is one of the simple crossover technique used for random GA applications. This crossover uses the single point fragmentation of the parents and then combines the parents at the crossover point to create the offspring or child.

1-Point crossover first selects two parents used for crossover and then randomly selects any crossover point p_i ($i = 0$ to $n-1$). Two offspring are created by combining the parents at crossover point. A simple example is shown below which performs one point crossover and creates two parents.

Parent 1:	1 0 1 0 | 1 0 0 1 0
Parent 2:	1 0 1 1 | 1 0 1 1 0
Offspring 1:	1 0 1 0 | 1 0 1 1 0
Offspring 2:	1 0 1 1 | 1 0 0 1 0

In above example, point between 4th and 5th gene is selected as crossover point.

2.2 K-POINT CROSSOVER

It uses the random crossover point to combine the parents same as per 1-Point crossover. To provide the great combination of parents it selects more than one crossover points to create the offspring or child [1].

K-Point Crossover first selects the two parents used for crossover and then randomly select K crossover points P_{1i} to P_{k-1i} ($i = 0$ to $n - 1$). Two offspring are created by combining the parents at crossover point. A simple example is shown below which performs one point crossover and creates two parents.

Parent 1:	1 0 | 1 0 | 1 0 0 | 1 0
Parent 2:	1 1 | 0 0 | 1 0 1 | 1 0
Offspring 1:	1 0 | 0 0 | 1 0 0 | 1 0
Offspring 2:	1 1 | 1 0 | 1 0 1 | 1 0

In above example, the points between 2nd and 3rd, 4th and 5th and 7th and 8th gene are selected as crossover points.

2.3 SHUFFLE CROSSOVER

Shuffle Crossover helps in creation of offspring which have independent of crossover point in their parents. It uses the same 1-Point Crossover technique in addition to shuffle.

Shuffle Crossover selects the two parents for crossover. It firstly randomly shuffles the genes in the both parents but in the same way. Then it applies the 1-Point crossover technique by randomly selecting a point as crossover point and then combines both parents to create two offspring. After performing 1-point crossover the genes in offspring are then unshuffled in same way as they have been shuffled.

Select Shuffle Points

Parent 1:	1 1 1 0 1 0 0 1 0
Parent 2:	1 0 0 0 1 0 1 1 0

Shuffle genes as Shuffle Points

Parent 1:	0 1 0 1 1 0 1 1 0
Parent 2:	0 0 1 1 1 0 0 1 0

Select 1- Point Crossover Point

Parent 1:	0 1 0 1 | 1 0 1 1 0
Parent 2:	0 0 1 1 | 1 0 0 1 0

Perform 1-Point Crossover Point

Offspring 1:	0 1 0 1 | 1 0 0 1 0
Offspring 2:	0 0 1 1 | 1 0 1 1 0

Select unshuffled points same as shuffled points

Offspring 1:	0 1 0 1 1 0 0 1 0
Offspring 2:	0 0 1 1 1 0 1 1 0

Unshuffled the genes in Offspring

Offspring 1:	1 1 0 0 1 0 0 1 0
Offspring 2:	1 0 1 0 1 0 1 1 0

2.4 REDUCED SURROGATE CROSSOVER

Reduced Surrogate Crossover minimizes the unwanted crossover operations in case of the parents having same genes. In these cases the Reduced Surrogate Crossover first checks for the individual genes in the parents. It creates list of all possible crossover points where the genes of the both parents are different.

After performing this check, if no crossover point is there then no action is taken. But in case, if parents are differing in more than 1 gene then it keeps the list of all crossover points. It then randomly selects one crossover point from the list and performs 1-point crossover to create the offspring.

2.5 UNIFORM CROSSOVER

Uniform crossover provides the uniformity in combining the bits of both parents. It performs this operation of swapping bits in the parents to be included in the offspring by choosing a uniform random real number *u* (between 0 to 1).

Uniform crossover selects the two parents for crossover. It creates two child offspring of n genes selected from both of the parents uniformly. The random real number decides whether the first child select the *i*th genes from first or second parent [1].

Parent 1:	1 1 1 0 1 0 0 1 0
Parent 2:	1 0 0 0 1 0 1 1 0
Offspring 1:	1 1 0 0 1 0 1 1 0
Offspring 2:	1 0 1 0 1 0 0 1 0

2.6 AVERAGE CROSSOVER (AX)

Average Crossover is the value based crossover technique. It uses two parents to perform crossover and creates only one offspring [1]. Average Crossover creates one offspring from taking average of the two parents. It selects two parents as X and Y and generate the child Z as follows: each gene in a child is taken by averaging genes from both parents.

Parent 1:	5 3 3 2 3 9 7 6 5
Parent 2:	5 4 7 6 5 2 6 1 3
Offspring 1:	5 3 5 4 4 5 6 3 4

2.7 DISCRETE CROSSOVER (DC)

Discrete Crossover uses the random real number to create one child from two parents.

Unlike the uniform crossover only one child is generated in Discrete Crossover. It selects two parents as X and Y and generate the child Z such that it select genes of both the parents uniformly. The random real number decides from which parent to take the genes for child. [1]

Parent 1:	1 1 1 0 1 0 0 1 0
Parent 2:	1 0 0 0 1 0 1 1 0
Offspring 1:	1 1 0 0 1 0 1 1 0

2.8 FLAT CROSSOVER (FC)

Flat Crossover uses the random real number to create one child from two parents.

Same as of Discrete Crossover it selects the genes from parent based on uniform random real number. But the selected random real number should be a subset of set having the minimum and maximum of the genes of the both parents. It selects two parents as X and Y and generate the child Z such that it selects random real number which is either min or max from genes in both parents and then assign this real number in child gene.

2.9 HEURISTIC CROSSOVER/INTERMEDIATE CROSSOVER (HC/IC)

Heuristic Crossover creates one child offspring from two parents. It uses the $\alpha\ \varepsilon < 0, 1 >$ assuming $x_i <= y_i$. For each gene in the child it select the uniform random real number α. And the child gene is calculated from above equation.

$$x_i\,(t+1) = x_i(t) + \alpha\,(y_i(t) - x_i(t))$$

Parameter α may be of constant value equal to 0.5 or may be selected by a draw from interval <0,1 (row: 5).

2.10 STATISTICS-BASED ADAPTIVE NON-UNIFORM CROSSOVER (SANUX)

It is based on the concepts of intrinsic attribute and extrinsic tendency of valuing allele for a gene locus. In optimal solution (encoded in binary string) of a given problem, for a gene locus if its allele is 1 it is called 1-intrinsic, if its allele is 0 it is called 0-intrinsic, otherwise if its allele either 0 or 1 it is called neutral. During the running of a GA, for a gene locus, if the frequency of 1's in its alleles over, the population tends to increase with time (generation), it called 1-inclined; if the frequency of 1's tends to decrease, it is called 0-inclined; otherwise it is called non-inclined. [13]

Usually and hopefully as the GA progresses, the gene loci which are 1-intrinsic will appear to be 1-inclined. SANUX makes use of this convergence information as feedback information to direct the crossover by adjusting the swapping probability of each locus.

Now during the evolution of the GA, after generation of new population the distribution of 1's $f_1(i, t)$ from each locus over the population is calculated. Then SANUX operation is performed. After applying SANUX, the mask is generated bit by flipping a coin biased to generate “1” with probability $p_s(i, t)$. Finally, the generated mask is used to guide the crossover in the same way it guides the traditional uniform crossover.

1’s Frq. in loci:	0.4	0.2	0.6	0.9	0.9	0.2
Calculating:	|	|	|	|	|	|
Swapping prob:	0.4	0.2	0.4	0.1	0.1	0.2
biased flipping:	|	|	|	|	|	|
Created mask:	1	0	1	0	0	0
Applying mask:	|		|			
Parent P1:	0	1	0	1	1	1
Parent P2:	1	1	1	1	1	0
Swapping:	|		|			
Child C1:	1	1	1	1	1	1
Child C2:	0	1	0	1	1	0

SANUX

3. BINARY CROSSOVERS

3.1 RANDOM RESPECTFUL CROSSOVER (RRC)

It selects two parents for crossover and offspring is generated based on the similarity vector of the parents. It first creates similarity vector $S_{ab} = (S_{1ab},\ldots,S_{nab})$ such that if both genes of parents have the same values then the similarity vector contains the values of the parent else similarity vector contain null value for that gene [1].

After creation of similarity vector S, two children are created according to the values of the similarity vector. If similarity vector contains 1 then gene of both children is set to 1 and if it contains 9 then genes of both children are set to 0. Apart from this if similarity vector contains null value for any gene then the child gene is selected by taking a uniform random real number. If this number is < 0.5 then 1 is stored otherwise 0 is stored.

Parent 1:	1 1 1 0 1 0 0 1 0
Parent 2:	1 0 0 0 1 0 1 1 0
Offspring 1:	1 1 0 0 1 0 1 1 0
Offspring 1:	1 0 0 0 1 0 0 1 0

This algorithm duplicates genes of parents in an offspring at every position wherever they are identical.

3.2 MASKED CROSSOVER (MX)

The MX operator uses a mask vector to determine which bits of which parent are inherited by the offspring. The first step is the duplication of the bits of the parents. The bits of the first parent are copied to the first offspring and, accordingly, of the second parent to the second offspring. In the second step, the offspring exchange bits among each other at those positions where the mask vectors of the parent were equal to 1, indicated domination of that parent at that position and the mask vectors of the other parent were equal to 0.

The mask vectors are initiated in $P(0)$ randomly. During every iteration of GA, the mask vectors are inherited by each offspring from its parent. Then the mask vectors of the offspring as well as the parents undergo modification. The modification process is based on comparison of fitness of offspring and the parents. If good offspring are created, the masks of the parents do not need to be modified and the masks of the offspring may be very similar to those of the parents. In a situation where bad offspring were created the masks of the parents as well as of the offspring need to be modified.

3.3 1- BIT ADAPTATION CROSSOVER (1BX)

In the 1 BX method, the last bit of the solution vector is reserved for the code of one of the two of the applied crossover operators. Assuming “0” corresponds to Uniform Crossover (UX) operator and “1” corresponds to 2-Point Crossover (2-PX) operator, the choice of one of them is made according to the rule: if the last bit of the parents is off the same value then choose the operator indicated by this bit. Otherwise chooses the operator through the selection by a draw i.e. select the uniform random real number from 0 to 1. If this value is <0.5, then Uniform Crossover is performed otherwise 2-Point Crossover is used.

Application of the described crossover scheme combines the choice of the operator with the solution vector. Moreover, this choice is carried out separately for each parent pair; hence this scheme is called local adaptation. Global adaptation version has also presented, but as it was emphasized by the author, significantly worse results were obtained by its application.

3.4 MULTIVARIATE CROSSOVER (MC)

MC divides the whole parent string into the q substrings. The crossover is performed based on random value selected for each substring. If this value is < P_c then crossover is performed other only parent genes are copied into child offspring. It performs the standard 1-Point Crossover for the substring when the condition is satisfied [1].

The most fundamental difference between the MC operator and other operators using variable-to-variable recombination is that the answer to the question “whether to crossover” is checked in the MC method separately for each substring. As for other operators, the answer to that question refers to parent vector as a whole.

3.5 HOMOLOGOUS CROSSOVER (HX)

The HX operator is based on the standard K-Point Crossover operator. Introduced modification relies on the fact that only strings of bits which are at least of a certain length or of an admissible degree of similarity are allowed to participate in crossover. Determination of the degree of similarity is based on the XOR operator.

This strategy is aimed at transferring strings with specified parameters to the next generation. In HX, the value of o and r is determined a priori as constant or dynamically changed in the GA run.

3.6 COUNT PRESERVING CROSSOVER (CPC)

The CPC operator carries out its task by assuming that the number of bits equal to “1” in every chromosome in the initial population $P(0)$ is the same.

CPC may guarantee the preservation of the constant number of bits equal to "1" due to application of two lists noting the differences between the parents. List L_{up} includes positions of those bits, on which there are differences between the parents, but the first parent at a given position holds a bit equal to "1" and the second equal to "0". List L down similarly notes the positions of differences, but the first parent at a given position holds a bit equal to "0" and the second equal to "1". The offspring creation process which makes use of those lists is based on the exchange of bits between the offspring at those positions which, are indicated by subsequent element pairs from lists L_{up} and L_{down}.

Number of elements in L_{up} and in L_{down} is the same, which is a direct result of the assumption, that the number of bits equal to "1" is constant for all chromosomes in P(0).

3.7 ELITIST CROSSOVER (EX)

In the standard genetic algorithm, the selection process is always preceded by the crossover process. In the EX method, both processes are integrated. During the first step of the entire population is randomly shuffled. Then from each successive pair of parental vectors, two new vectors are created by crossover. From a 'family' created, two best vectors are singled out and implemented as offspring to the next population.

Application of elitist selection in the traditional way that is on the level of the entire population may often be the reason for the premature convergence of the algorithm. An EX elitist selection applied on the "family" level eliminates this danger according to the authors.

3.8 SCANNING CROSSOVER (SCAN), UNIFORM SCANNING CROSSOVER (U-SCAN), OCCURRENCE BASED SCANNING CROSSOVER (OB-SCAN) FITNESS BASED SCANNING CROSSOVER (FB-SCAN)

Depending on the heuristics applied to scanning crossover, three variations are: uniform scanning crossover (U-Scan), occurrence based scanning crossover (OB-Scan), and fitness based scanning crossover (FB-Scan).

Increasing the number of parents in OB-Scan leads to a higher probability for the major alleles in the population to exist in every offspring, which will cause the population to concentrate on a certain region and thus lose the diversity rapidly [18], [19]. This situation is called premature convergence, a result of unbalanced exploitation and exploration.

3.9 SELF-ADAPTIVE SIMULATED BINARY CROSSOVER (SBX)

A self-adaptive procedure for updating the distribution index used in the simulated binary crossover or SBX operator which is a commonly-used real-parameter recombination operator. This crossover is also good for multi-objective optimization problem.

3.10 OTHER BINARY CROSSOVER

Some binary crossover is proposed by researchers are - Circ1e-ring crossover, Sufficient Exchanging crossover [12], Adaptive crossovers [12], Diagonal crossover [21, 22], Best combinatorial crossover (BCX) and Hybridization crossover (HX) [14].

4. APPLICATION DEPEDANT CROSSOVERS (REAL/TREE)

4.1 CROSSOVER FOR TSP PROBLEMS

4.1.1 Order Based Crossover (OBX):

The order based crossover operator selects at random several positions in one of the parent tours, and the order of the cities in the selected positions of this parent is imposed on the other parent to produce one child. The other child is generated in an analogous manner for the other parent [1].

4.1.2 Modified Order Crossover (MOC):

A randomly chosen crossover point divides the parent strings in left and right substrings. The right substrings of the parent s1 and s2 are selected. After selection of cities the process is the same as the order crossover. Only difference is that instead of selecting random several positions in a parent tour all the positions to the right of the randomly chosen crossover point are selected [1].

For example with the following parents and crossover point

s1 = (1 2 3 4 | 6 9 8 5 7) and

s2 = (2 1 9 8 |5 6 3 7 4)

After position selection

s1 = (1 2 * * * 9 8 * *) and

s2 = (2 1 * * * * 3 * 4)

Now obtain the generated pair of children as

b1 = (1 2 5 6 3 9 8 7 4) and

b2 = (2 1 6 9 8 5 3 7 4)

Clearly this method allows only the generation of valid strings.

4.1.3 Partially-Mapped Crossover (PMX):

It transmits ordering and values information from the parent strings to the offspring. A portion of one parent string is mapped onto a portion of the other parent string and the remaining information is exchanged. Consider, for example, the following two parents: (1 2 3 4 5 6 7 8) and (3 7 5 1 6 8 2 4). The PMX operator creates an offspring in the following way. It begins by selecting uniformly at random two cut points along the strings, which represent the parents. Suppose, for example, that the first cut point is selected between the third and the fourth string element, and the second one between the sixth and the seventh string element. Hence, (1 2 3 | 4 5 6 | 7 8) and (3 7 5 | 1 6 8 I 2 4). The substrings between the cut points are called the mapping sections. In our example, they define the mappings 4 <-> 1, 5 <-> 6, and 6 <-> 8. Now the mapping section of the first parent is copied into the second offspring, and the mapping section of the second parent is copied into the first offspring: offspring 1: (x x x | l 6 8 | x x) and offspring 2: (x x x | 4 5 6 | x x). Then offspring *i* (*i* = 1, 2) is filled up by copying the elements of the *i*th parent. In case, a number is already present in the offspring it is replaced according to the mappings [2].

For example, the first element of offspring 1 would be a 1, like the first element of the first parent. However, there is already a 1 present in offspring 1. Hence, because of the mapping 1 <-> 4 choose the first element of offspring 1 to be a 4. The second, third and seventh elements of offspring 1 can be taken from the first parent. However, the last element of offspring 1 would be an 8, which is already present. Because of the mappings 8 <-> 6, and 6 <-> 5, it is chosen to be a 5. Hence, offspring 1: (4 2 3 | 1 6 8 | 7 5). Analogously, we find offspring 2: (3 7 8 | 4 5 6 | 2 1). The absolute positions of some elements of both parents are preserved.

4.1.4 *Modified Partially-Mapped Crossover (MPMX):*

Modified PMX (MPMX) crossover operator was proposed (independently) by Brown [39] in the late 80's. The MPMX operator initially partitions the parents' solution strings and the offspring strings into three sections (left, middle and right). These sections are randomly created through the selection of two random crossover points that will be used for both the parents and offspring for this instance of the crossover in the GA. Stage two provides the offspring with the middle section of its solution string. This is the donated middle section of parent 1. The third stage, is the insertion of elements into the left and right sections of the offspring. This is accomplished using parent 2 as the donator. Corresponding positions in the parents donate elements to the offspring, provided they have not already been donated by parent 1. The final stage is to complete the offspring using a random permutation of the elements not yet allocated to the offspring over the previous stages. The following example illustrates the:

Parent 1 - (0 8 | 4 5 6 7 | 1 2 3 9)

Parent 2 - (6 7 | 1 2 4 8 | 3 5 9 0)

Offspring stage1- (- - | - - - - | - - - -)

Offspring stage2- (- - | 4 5 6 7 | - - - -)

Offspring stage3- (- - | 4 5 6 7 | 3 - 9 0)

Offspring stage4- (8 1 4 5 6 7 3 2 9 0)

4.1.5 *Cycle Crossover (CX) [3]:*

It attempts to create an offspring from the parents where every position is occupied by a corresponding element from one of the parents. For example, consider again the parents (1 2 3 4 5 6 7 8) and (2 4 6 8 7 5 3 1). Now choose the first element of the offspring equal to either the first element of the first parent string or the first element of the second parent string. Hence, the first element of the offspring has to be a 1 or a 2. Suppose choose it to be 1, (1 * * * * * * *). Now consider the last element of the offspring. Since this element has to be chosen from one of the parents, it can only be an 8 or a 1. If a 1 were selected, the offspring would not represent a legal individual. Therefore, an 8 is chosen, (1 * * * * * * 8). It finds that the fourth and the second element of the offspring also have to be selected from the first parent, which results in (1 2 * 4 * * * 8). The positions of the elements chosen up to now are said to be a cycle. Now consider the third element of the offspring. This element it may choose from any of the parents. Suppose that we select it to be from parent 2. This implies that the fifth, sixth and seventh elements of the offspring also have to be chosen from the second parent, as they form another cycle. Thus, we find the following offspring: (1 2 6 4 7 5 3 8). The absolute positions, of on average half the elements of both parents are preserved.

4.1.6 *Order Crossover Operator (0X1):*

It constructs an offspring by choosing a substring of one parent and preserving the relative order of the elements of the other parent. For example, consider the following two parent strings: (1 2 3 4 5 6 7 8) and (2 4 6 8 7 5 3 l), and suppose that we select a first cut point between the second and the third bit and a second one between the fifth and the sixth bit. Hence, (1 2 | 3 4 5 | 6 7 8) and (2 4 | 6 8 7 | 5 3 1). The offspring are created in the following way. Firstly, the string segments between the cut point are copied into the offspring, which give (* * | 3 4 5 | * * *) and (* * | 6 8 7 | * * *). Next, starting from the second cut point of one parent, the rest of the elements are copied in the order in which they appear in the other parent, also starting from the second cut point and omitting the elements that are already present. When the end of the parent string is reached, we continue from its first position. In our example, this gives the following children: (8 7 | 3 4 5 | 1 2 6) and (4 5 | 6 8 7 | 1 2 3) [4].

4.1.7 *Order-based Crossover (OX2):*

OX2 was suggested in connection with schedule problems, is a modification of the OX1 operator. The OX2 operator selects at random several positions in a parent string, and the order of the elements in the selected positions of this parent is imposed on the other parent. For example, consider again the parents (1 2 3 4 5 6 7 8) and (2 4 6 8 7 5 3 I), and suppose that in the second parent in the second, third and sixth positions are selected. The elements in these positions are 4, 6 and 5 respectively. In the first parent, these elements are present at the fourth, fifth and sixth positions. Now the offspring are equal to parent 1 except in the fourth, fifth and sixth positions: (1 2 3 * * * 7 8). We add the missing elements to the offspring in the same order in which they appear in the second parent. This results in (1 2 3 4 6 5 7 8). Exchanging the role of the first parent and the second parent gives, using the same selected positions, (2 4 3 8 7 5 6 1) [5].

The position-based crossover operator (POS), which was also suggested in connection with schedule problems, is a second modification of the OX1 operator. It also starts with selecting a random set of positions in the parent strings. However, this operator imposes the position of the selected elements on the corresponding elements of the other parent. For example, consider the parents (1 2 3 4 5 6 7 8) and (2 4 6 8 7 5 3 l), and suppose that the second, third and sixth positions are selected. This leads to the following offspring: (1 4 6 2 3 5 7 8) and (4 2 3 8 7 6 5 1).

4.1.8 *Voting Recombination Crossover operator (VR):*

It can be seen as a P-sexual crossover operator, where *p* is a natural number greater than, or equal to, 2. It starts by defining a threshold, which is a natural number smaller than, or equal to *p*. Next, for every; $i \in \{l, 2, . . .N\}$ the set of i^{th} elements of all the parents is considered. If in this set an element occurs at least the threshold number of times, it is copied into the offspring. For example, if we consider the parents (p = 4) (1 4 3 5 2 6), (1 2 4 3 5 6), (3 2 1 5 4 6), (1 2 3 4 5 6) and we define the threshold to be equal to 3 we find (1 2 x x x 6). The remaining positions of the offspring are filled with mutations. Hence, our example might result in (1 2 4 5 3 6) [6].

4.1.9 Maximal Preservation crossover (MPX):

The MPX operator was developed by Gorges-Schleuter and Mülhelenbein [42] in 1988 specifically for the TSP. It is closely related to the PMX crossover operator [48]. MPX operates by initially selecting a random substring (the TSP this is a subtler) from the first parent (called the donor). This subtour is usually defined as being a tour with string length less than or equal to the TSP problem size *n* divided by 2. A minimum subtour length is also set, typically at 10 elements (unless the TSP problem size is very small), as substrings that are very short are ineffective and substrings that are too large do not allow for meaningful variation. Selecting appropriate sized substrings provides a suitable means for parents to transmit significant loci information to the offspring. The second stage of MPX is to remove the elements currently in the offspring from the second parent. Then the remaining elements are inserted into the offspring, the first parent's substring having been placed at the start of the offspring and the remaining free elements of the offspring being filled by the clean parent 2 strings. This three stage operation of MPX is illustrated in following example:

Parent 1 - (1 4 3 5 2 6)

Parent 2 - (1 2 4 3 5 6)

Offspring (1 4 3 x x x)

Cleaned Parent 2 - (- 2 - - 5 6)

Offspring (1 4 3 2 5 6)

With regard to the MPX and its application to the TSP, although the MPX prevents invalid tour generation in the offspring, they are liable to be produced with few building blocks being inherited from both parents due to the cleaning of the second parent's string prior to completing the offspring strings.

4.1.10 Masked crossover (MkX):

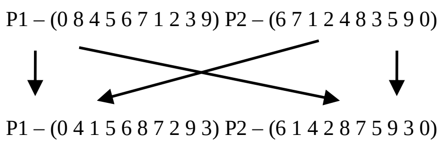
The Masked Crossover (MkX) technique was first proposed by Louis and Rawlins in 1991 [43] as a crossover operator which would efficiently operate in the combinatorial logic design problem area rather than as a combinatorial optimization technique. MkX [48] attempts to impart loci information from parent to offspring in a more effective manner than previous crossover methods. Louis and Rawlins state that MkX tries to preserve schemas identified by the masks and they identify this as one of their key goals [43]. The MkX operator assigns each parent a mask that biases crossover. Once these masks have been positioned then the operation is as following:

1. Copy Parent1 to Offspring1 and Parent2 to Offspring2
2. For (*i* from 1 to string-length)
   if Mask2*i* = 1 and Mask1*i* = 0
3. Copy the i^{th} bit from Parent2 to Offspring1
   if Mask1*i* = 1 and Mask2*i* = 0
4. Copy the i^{th} bit from Parent1 to Offspring2

The offspring of MkX also require masks, should they be selected to be parents in another generation. The masks are normally provided to the offspring by the parents. Typically the parent that is designated the dominant parent is called Parent1 the dominant parent with respect to Offspring1 as Offspring1 inherits Parent1's bits unless Parent2 feels strongly (Mask2*i* = 1) and Parent1 does not (Mask1*i* = 0). A number of mask rules are also defined by Louis and Rawlins. Two of which are used when the simple rule of assigning masks from dominant parent to offspring don't apply. Chan [47] notes that the MkX is an ineffective crossover operator for the TSP as it fails to preserve the ordering of the solutions. Validity of solution is problematic and (in conjunction with the selected mutation operator) typically involves a repair or penalty function.

4.1.11 Position crossover (PX):

The Position Crossover (PX) operator was developed by Syswerda in 1991 [5], [48]. PX was later evaluated by Barbulescu [44] where she examined and compared PX's operation to similar operators for scheduling problems. This crossover technique is closely related to OX and OX2 crossover techniques. PX operates by selecting several random locations along the parent strings. The elements are then inherited by the offspring in the order that they occur in the first parent (P). The remaining elements required to complete the offspring (O) are donated by the second parent (with the elements donated by the first parent omitted) in the order that they appear in the second parent. Each step of the operation is illustrated as follows:


P1 – (0 8 4 5 6 7 1 2 3 9) P2 – (6 7 1 2 4 8 3 5 9 0)
P1 – (0 4 1 5 6 8 7 2 9 3) P2 – (6 1 4 2 8 7 5 9 3 0)

4.1.12 Complete Subtour Exchange Crossover:

The complete subtour exchange crossover (CSEX) operator is designed to operate with the path representation. CSEX was proposed by Katayama and Narihisa [45] to be used specifically for permutation problems (such as the TSP). The philosophy behind CSEX [48] is to encourage the offspring to inherit as many good traits (substance) from the parents as possible. CSEX enumerates substance that have the same direction (or reversed direction) on two permutations as common substance.

Parent 1 - (0 1 2 3 4 5 6 7 8 9)

Parent 2 - (4 9 7 6 5 0 8 2 1 3)

Offspring 1- (0 2 1 3 4 5 6 7 8 9)

Offspring 2- (0 1 2 3 4 7 6 5 8 9)

Offspring 3- (0 2 1 3 4 7 6 5 8 9)

Offspring 4- (4 9 5 6 7 0 8 2 1 3)

Offspring 5- (4 9 7 6 5 0 8 1 2 3)

Offspring 6- (4 9 5 6 7 0 8 1 2 3)

In above see an example of CSEX in operation. The common subtours are 12 and 5 6 7 in parent 1, and 7 6 5 and 2 1 in parent 2. It should be noted that CSEX does not include the subtour 1 2 3 from parent 1 and 2 1 3 from parent 2 as they are not the same or symmetrical. CSEX by allowing only the same (or symmetrical) subtours can enumerate all the common subtours with $O(n)$ time. Having selected the common subtours, the offspring are produced by inverting the common subtours from the parent. In the example, parent 1 produces offspring 1, 2 and 3 by inverting a common subtour for each offspring. This is then repeated for parent 2 which produces offspring 4, 5 and 6. Once all the offspring are produced they are evaluated for fitness and the two fittest offspring survive to the next generation.

4.1.13 Heuristic crossover (HX):

It is worth noting that the previous crossover operators did not exploit the distances between the cities (i.e. the length of the edges). In fact, it is a characteristic of the genetic approach to avoid any heuristic information about a specific application domain, apart from the overall evaluation or fitness of each chromosome. This characteristic explains the robustness of the genetic search and its wide applicability [9] [10] [48].

However, some researchers departed from this line of thinking and introduced domain-dependent heuristics into the genetic search, to create "hybrid" genetic algorithms. They have sacrificed robustness over a wide class of problems, for better performance on a specific problem. The heuristic crossover HX is an example of this approach and can be described as follows:

1. Pick a random starting city at one of the two parents.
2. Compare the edges leaving the current city in both parents and select the shorter edge.
3. If the shorter parental edge introduces a cycle in the partial tour, then extend the tour with a random edge that does not introduce a cycle.
4. Repeat Steps 2 and 3 until all cities are included in the tour.

4.1.14 Edge Recombination crossover (ER):

Quite often, the alternate edge operator introduces many random edges in the offspring, particularly the last edges, when the choices for extending the tour are limited. Since the offspring must inherit as many edges as possible from the parents, the introduction of random edges should be minimized. The edge recombination operator reduces the myopic behavior of the alternate edge approach with a special data structure called the "edge map".

Basically, the edge map maintains the list of edges that are incident to each city in the parent tours, and lead to cities not yet included in the offspring. Hence, these edges are still available for extending the tour, and are said to be active. The strategy is to extend the tour by selecting the edge that leads to the city with the minimum number of active edges. In case of equality between two or more cities, one of these cities is selected at random. With this strategy, the approach is less likely to get trapped in a "dead end", namely, a city with no remaining active edges that require the selection of a random edge [8] [48].

For tours of 13564287 and 14236578 (path representation), the initial edge map is shown below:

City 1 has edges to: 3 4 7 8
City 2 has edges to: 3 4 8
City 3 has edges to: 1 2 5 6
City 4 has edges to: 1 2 6
City 5 has edges to: 3 6 7
City 6 has edges to: 3 4 5
City 7 has edges to: 1 5 8
City 8 has edges to: 1 2 7

Fig.1. Edge Map

Let us assume that city 1 is selected as the starting city. Accordingly, all edges incident to city 1 must be deleted from the initial edge map. From city 1, we can go to city 3, 4, 7 or 8. City 3 has three active edges, while cities 4, 7 and 8 have two active edges, as shown by the edge map (a) in Fig.1. Hence, a random choice is made between cities 4, 7 and 8. We assume that city 8 is selected. At 8, we can go to cities 2 and 7. As indicated in the edge map (b), city 2 has two active edges and city 7 only one, so the latter is selected. From 7, there is no choice, but to go to city 5. From this point, edge map (d) offers a choice between cities 3 and 6 with two active edges. Let us assume that city 6 is randomly selected. From city 6, we can go to cities 3 and 4, and edge map (e) indicates that both cities have one active edge. We assume that city 4 is randomly selected. Finally, from city 4 we can only go to city 2, and from city 2 we must go to the city 3.

Some modified edge recombination crossover are Edge Exchange Crossover (EXX) and Edge Assembly Crossover (EAX) [31].

4.1.15 Alternate Edges Crossover:

It is a good introduction to other edge-preserving operators. Here, a starting edge (*i*, *j*) is selected at random in one parent. Then, the tour is extended by selecting the edge (*j*, *k*) in the other parent. The tour is progressively extended in this way by alternatively selecting edges from the two parents. When an edge introduces a cycle, the new edge is selected at random (and is not inherited from the parents) [9] [48].

In following example, an offspring is generated from two parent chromosomes that encode the tours 13564287 and 14236578, respectively, using the adjacency representation. Here, edge (1, 4) is first selected in parent 2, and city 4 in position 1 of parent 2 is copied at the same position in the offspring. Then, the edges (4, 2) in parent 1, (2, 3) in parent 2, (3, 5) in parent 1 and (5, 7) in parent 2 are selected and inserted in the offspring. Then, edge (7, 1) is selected in parent 1. However, this edge introduces a cycle and a new edge incident to 7 and to a city not yet visited is selected at random. Let us assume that (7, 6) is chosen. Then, edge (6, 5) is selected in parent 2, but it also introduces a cycle. At this point, (6, 8) is the only selection that does not introduce a cycle. Finally, the tour is completed with edge (8, 1).

parent 1: 3 8 5 2 6 4 1 7

parent 2: 4 3 6 2 7 5 8 1

offspring: 4 3 5 2 7 8 6 1

The Alternate Edges Crossover

The final offspring encodes the tour 14235768, and all edges in the offspring are inherited from the parents, apart from the edges (7, 6) and (6, 8). In the above description, an implicit orientation of the parent tours is assumed. For symmetric problems, the two edges that are incident to a given city can be considered. In the above example, when we get to city 7 and select the next edge in parent 1, edges

(7, 1) and (7, 8) can both be considered. Since (7, 1) introduces a cycle, edge (7, 8) is selected. Finally, edges (8, 6) and (6, 1) complete the tour.

parent 1: 3 8 5 2 6 4 1 7

parent 2: 4 3 6 2 7 5 8 1

offspring: 4 3 5 2 7 1 8 6

It is alternate edges crossover.

4.1.16 Greedy Subtour Crossover:

New crossover operator named `Greedy Subtour Crossover (GSX)' that acquires the longest possible sequence of parents' subtours. Using GSX the solution can pop up from local minima more effectively than by using simulated annealing (SA) methods.

In the GSX, we use the path representation for a genetic coding. For example, chromosome $g = (D, H, B, A, C, F, G, E)$ means that the salesperson visits towns D, H, B, A,.., E, successively, and returns to town D.

Suppose that chromosomes of parents are $ga = (D, H, B, A, C, F, G, E)$ and $gb = (B, C, D, G, H, F, E, A)$. First, choose one town at random. In this example, town C is chosen. Then, $x = 4$ and $y = 1$ because $a_4 = \mathrm{C}$ and $b_1 = \mathrm{C}$ respectively. Now the child g is (C).

Next, pick up towns from the parents alternately. Begin with a3 (town A) because x <- 4 <- 1 = 3, and next is b2 (town D) because y <- 1<- 1 = 2. The child becomes $g = (A, C, D)$.

In the same way, add a_2 (town B), b_3 (town G), a_1 (town H), and the child becomes $g = (H, B, A, C, D, G)$. Now the next town is $b_4 = \mathrm{H}$ and town H has already appeared in the child (remember the salesperson may not visit the same town twice), so we can't add any more towns from parent g_b. Therefore we add towns from parent g_a. The next town is $a_0 = D$, but D is already used. Thus we can't add towns from parent g_a, either. Then, we add the rest of the towns, i.e., E and F, to the child in the random order. Finally the child is $g = (H, B, A, C, D, G, F, E)$ [11] [48].

4.1.17 Edge Assembly Crossover:

EAX [48] has two important features—preserving parents' edges using a novel technique and adding new edges by a greedy method, analogous to a minimal spanning tree. Several issues, including the selection mechanism and heuristic methods, which affect the performance of EAX have been considered.

4.2 CROSSOVER FOR OBJECT CLASSIFICATION PROBLEMS

- **Object classification problems:** Looseness control crossover (LCC) and Headless chicken crossover (HCC) [32]
- **Crossover for Sudoku problem:** Product Geometric Crossover (PMX) [33]
- **Crossover for grouping, graph, sequence and glude space applications:** Quotient Geometric Crossovers [34] and Merge Crossover (MX) [46].
- **Crossover for Graph Partitioning Problems:** Geometric Crossover (GX) & Geometric Crossover Labeling-independent (LI) GX Crossover and Landscape of Labeling-independent Crossover [36].
- **Crossover for Graph Colouring Problem (for Parallel GA):** Conflict elimination crossover (CEX), Greedy partition crossover (GPX), Union Independent set crossover (UISX) and Sum product crossover (SPPX) [37].
- **Multi-Parent Crossover/ Multi-Parent feature Crossover (MFX), Multicut crossover (MX) and Seed crossover (SX) [14],[15]and[16]:** The increase of parents brings about a more comprehensive survey for determining the offspring genes and leads to a stronger tendency towards exploitation or exploration or both [18], [19], [20].
- **Center of mass crossover (CMX):** These multi-parent crossovers can lead to better performance although the performance is problem-dependent [14, 15].
- **Unimodal normal distribution crossover (UNDX):** Multiple parents into unimodal normal distribution crossover (UNDX) to enhance the diversity of offspring. This multi-parent extension of UNDX exhibits its improvement in search ability on highly epistatic problems [24].
- **Simplex crossover (SPX):** SPX performs well with three or four parents for multimodal and epistatic problems [17].

4.3 CROSSOVER FOR SUDOKU PROBLEM

Knowledge-Based Nonuniform Crossover [26], Strong Context Preserving Crossover (SCPC) Weak Context Preserving Crossover (SCPC) [27], Hierarchical Crossover [28], Selective crossover [29], Rank & proximity Based Crossover (RPC) [30], Depth-Dependent Crossover (DDX) [35], Alternating-Position Crossover (AP) [7], Circle Ring Crossover [12] and Sufficient Exchanging [12].

5. CONCLUSION

Many crossover operators are present in GA that are used in applications. The encoding type used in GA is the major criteria for selecting the crossover. The global convergence and search space must be considered while selecting the crossover operators. Effect of crossover operators in GA is application as well as encoding dependent. Many researchers say that the value of crossover probability is in between 0.6 and 1.0 and it also depends on the type of crossover used. Increasing crossover probability increases the opportunity for recombination but also may disrupt in good combination. Depending on encoding, standard crossover can have high chance to produce illegal offspring (it is application dependent e.g. TSP). The application to be solved must consider for all the crossover operators available along with possible encoding methods to get good results.

Many new applications are solved by existing crossover operators by considering their effectiveness in related applications. Most of time new crossover operators use old crossover operators along with additional changes. To get the proper crossover operators for a new problem it is recommended that, to see similar types of problems solved using GA along with various crossover operators used. It is recommended that for solving any problem it is important to overview the search space with modality extremes. Continuity (nature of search space), study existing crossover operators and then select or create a new crossover (by mixing).

ACKNOWLEDGMENTS

Our special thanks to Dr. Tomasz Dominik Gwiazda for his e-book "Genetic Reference Volume–I Crossover for Single objective numerical optimization" and Dr. George G. Mitchell for his Ph.D Thesis "Evolutionary Computation Applied to

Combinatorial Optimization Problems" for inspiring to write review paper on crossover operator.

REFERENCES

[1] Tomasz Dominik Gwiazda, "*Genetic Algorithms Reference*", Volume –I, Poland: Tomasz Gwiazda, 2006

[2] David E. Goldberg and Robert Lingle Jr., "Alleles, loci and the traveling salesman problem", *Proceedings of the 1st International Conference on Genetic Algorithms*, pp. 154-159, 1985.

[3] I. M. Oliver, D. J. Smith, and J. R. C. Holland, "A study of permutation crossover operators on the TSP", *Proceedings of the 2nd International Conference on Genetic Algorithms on Genetic Algorithms and their Application*, pp. 224-230, 1987.

[4] Lawrence Davis, "Applying adaptive algorithms to epistatic domains", *Proceedings of the 9th international joint conference on Artificial Intelligence*, Vol. 1, pp. 162-164, 1985.

[5] Gilbert Syswerda, "Schedule optimization using genetic algorithms", *Handbook of Genetic Algorithms*, pp. 332-349, New York: Van Nostrand Reinhold, 1991.

[6] H. Muhlenbein, "Parallel genetic algorithms, population genetics and combinatorial optimization", *Proceedings of Workshop on Parallel Processing: Logic, Organization and Technology*, pp. 398-406, 1991.

[7] P. Larranaga, C. M. H. Kuijpers, M. Poza, and R. H. Murga, "Optimal Decomposition of Bayesian Networks by Genetic Algorithms", Internal Report, EHU-KZAA-IKT-3-94, Department of Computer Science and Artificial Intelligence, University of the Basque Country, 1994.

[8] L. Darrell Whitley, Timothy Starkweather and D'Ann Fuquay, "Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator", *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 133-140, 1989.

[9] John J. Grefenstette, Rajeev Gopal, Brian J. Rosmaita and Dirk Van Gucht, "Genetic Algorithms for the Traveling Salesman Problem", *Proceedings of the 1st International Conference on Genetic Algorithms*, pp. 160-168, 1985.

[10] J. Grefenstette, "*Incorporating Problem Specific Knowledge into Genetic Algorithms*", *In L. Davis (ed.)* "*Genetic Algorithms and Simulated Annealing*", Morgan Kaufmann, pp. 42-60, 1987.

[11] Sengoku, H., Yoshihara, I., "A Fast TSP Solution using Genetic Algorithm (Japanese)", *Proceedings of 46th National Convention of Information Processing Society of Japan*, 1993.

[12] Liang-Jie Zhang, Mao Zhi-Hong and Li Yan-Da, "Mathematical Analysis of Crossover Operator in Genetic Algorithms and its Improved Strategy", *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 412-417, 1995.

[13] Shengxiang Yang, "Adaptive Non-Uniform Crossover Based on Statistics for Genetic Algorithms", *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 650-657, 2002.

[14] Yoshida Junichi, Miki Mitsunori, Hiryasu Tomoyuki and Sakata Yoshinobu, "New Crossover Scheme for Parallel Distributed Genetic Algorithms " *Proceedings of the IASTED International Conference on Parallel and Distributed Computing And Systems*, Vol. 1, No. 6-9, pp. 145-150, 2000.

[15] S. Shigeyoshi Tsutsui, "Multi-parent recombination in genetic algorithms with search space boundary extension by mirroring", *Parallel Problem Solving from Nature – PPSN V, Lecture Notes in Computer Science*, Vol. 1498, pp. 428-437, 2006.

[16] S. Tsutsui and A. Ghosh, "A study on the effect of multi-parent recombination in real coded genetic algorithms", *Proceedings of IEEE World Congress on Computational Intelligence*, pp. 828-833, 1998.

[17] Chuan-Kang Ting and Chun-Cheng Chen, "The Effects of Supermajority on Multi-Parent Crossover", *IEEE Congress on Evolutionary Computation*, pp. 4524-4530, 2007.

[18] S. Tsutsui, M. Yamamura and T. Higuchi. "Multi-parent recombination with simplex crossover in real coded genetic algorithms", *Proceedings of the Genetic and Evolutionary Computation Conference*, Vol. 1, pp. 657-664, 1999.

[19] Chuan-Kang Ting, "On the convergence of multi-parent genetic algorithms", *IEEE Congress on Evolutionary Computation*, Vol. 1, pp. 396-403, 2005.

[20] Chuan-Kang Ting, "On the mean convergence time of multi-parent genetic algorithms without selection", *Proceedings of the 8th European Conference on Advances in Artificial Life*, Vol. 3630, pp. 403-412, 2005.

[21] A. E. Eiben, "*Multiparent Recombination*", Handbook of Evolutionary Computation, Institute of Physics Publishing and Oxford University Press, 1997.

[22] A. E. Eiben and C.H.M. van Kemenade, "Diagonal crossover in genetic algorithms for numerical optimization", *Journal of Control and Cybernetics*, Vol. 26, No. 3, pp. 447-465, 1997.

[23] A.E. Eiben, C.H.M. van Kemenade and J.N. Kok. "Orgy in the Computer: Multi-parent reproduction in genetic algorithms", *Proceedings of the 3rd European Conference on Artificial Life*, pp. 934-945, 1995.

[24] Kalyan Deb, S. Karthik and Tatsuya Okabe, "Self-Adaptive Simulated Binary Crossover for Real-Parameter Optimization", *Genetic and Evolutionary Computation Conference*, pp. 1187-1194, 2007.

[25] H. Kita, I. Ono and S. Kobayashi, "Multi-parental extension of the unimodal normal distribution crossover for real-coded genetic algorithms", *Proceedings of the Congress on Evolutionary Computation*, Vol. 2, pp. 1581-1588, 1999.

[26] Inki Hong, Andrew B. Kahng and Byung Ro Moon, "Exploiting Synergies of Multiple Crossovers: Initial Studies", *Proceedings of IEEE International Conference on Evolutionary Computation*, Vol. 1, 1995.

[27] Harpal Maini, Kishan Mehrotra, Chilukuri Mohan and Sanjay Ranka, "Knowledge-Based Nonuniform

Crossover", *Proceedings of the 1st IEEE Conference on World Congress on Computational Intelligence Evolutionary Computation*, pp. 22-271, 1994.

[28] Patrik D'haeseleer, "Context Preserving Crossover in Genetic Programming", *Proceedings of the 1st IEEE Conference on World Congress on Computational Intelligence Evolutionary Computation*, Vol. 1, pp. 256-261, 1994.

[29] P. J. Bentley and J. P. Wakefield, "Hierarchical Crossover in Genetic Algorithms", *Proceedings of the 1st On-line Workshop on Soft Computing*, 1996.

[30] Chilkuri K. Mohan, "Selective crossover: towards fitter offspring", *Proceedings of the ACM symposium on Applied Computing*, pp. 374-378, 1998.

[31] Goutam Chakraborty and Basabi Chakraborty, "Rank and Proximity Based Crossover (RPC) to Improve Convergence in Genetic Search", *IEEE Congress on Evolutionary Computation*, Vol. 2, pp. 1311-1316, 2005.

[32] Yuichi Nagata, "Criteria for designing crossovers for TSP**",** *IEEE Congress on Evolutionary Computation,* Vol. 2, pp. 1465-1472, 2004.

[33] Mengjie Zhang, Xiaoying Gao and Weijun Lou, "Looseness Controlled Crossover in GP for Object Recognition", *IEEE Congress on Evolutionary Computation*, pp.1285-1292, 2006.

[34] Alberto Moraglio, Julian Togelius and Simon Lucas, "Product Geometric Crossover for the Sudoku Puzzle", *IEEE Congress on Evolutionary Computation*, pp. 470-476, 2006.

[35] Yourim Yoon, YongHyuk Kim, Alberto Moraglio and Byung Ro Moon, "Quotient Geometric Crossovers", Technical Report, CSM-467, Department of Computer Science, University of Essex, 2007.

[36] Takuya Ito, Hitoshi Iba and Satoshi Sato, "Depth-Dependent Crossover for Genetic Programming", Proceedings *IEEE World Congress on Computational Intelligence Evolutionary Computation*, pp. 775-780, 1995.

[37] Alberto Moraglio, Yong-Hyuk Kim, Yourim Yoon and Byung-Ro Moon, "Geometric Crossovers for Multiway Graph Partitioning", *Evolutionary Computation*, Vol. 15, No. 4, pp. 445-474, 2007.

[38] Zbigniew Kokosiński, Marcin Kołodziej and Krzysztof Kwarciany, "Parallel Genetic Algorithm for Graph Coloring Problem", *Computational Science - ICCS 2004*, Vol. 3036, pp. 215-222, 2004.

[39] Jong De Jong and Kenneth Alan, "An Analysis of the Behavior of a class of Genetic Adaptive Systems", PhD Thesis, University of Michigan, 1975.

[40] Donald E. Brown, Christopher L. Huntley and Andrew R. Spillane, "A Parallel Genetic Heuristic for the Quadratic Assignment Problem", *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 406-415, 1989.

[41] Andrew L. Tuson, "The Implementation of a Genetic Algorithm for the Scheduling and Topology Optimisation of Chemical Flowshops", Technical Report TRGA94-01, Physical Chemistry Laboratory, Oxford University, 1994.

[42] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer, "Evolution Algorithms in Combinatorial Optimization", *Parallel Computing*, Vol. 7, No. 1, pp. 65-85, 1988.

[43] Sushil. J. Louis and Gregory J. E. Rawlins, "Designer Genetic Algorithms: Genetic Algorithms in Structure Design", *Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 53-60, 1991.

[44] Laura Barbulescu, Adele E. Howe, L. Darrell Whitley and Mark Roberts, "Understanding Algorithm Performance on an Oversubscribed Scheduling Application", *Journal of Artificial Intelligence Research*, Vol. 27, pp. 577-615, 2006.

[45] Kengo Katayama and Hiroyuki Narihisa, "An Efficient Hybrid Genetic Algorithm for the Traveling Salesman Problem", *Electronics and Communications in Japan*, Vol. 84, No. 2, pp. 76-83, 2001.

[46] Sushil J. Louis, Xiangying Yin and Zhen Ya Yuan, "Multiple Vehicle Routing With Time Windows Using Genetic Algorithms", *Proceedings of the Congress on Evolutionary Computation*, 1999.

[47] Yam Ling Chan, "A Genetic Algorithm Shell for Iterative Timetabling", M.S. Thesis, Department of Computer Science. RMIT University, 1994.

[48] George G. Mitchell, "Evolutionary Computation Applied to Combinatorial Optimisation Problems," Ph.D. Thesis, School of Electronic Engineering, Dublin City University, 2007.