The Role of Crossover in Genetic Algorithms to Solve Optimization of a Function Problem

Falih Hassan

ABSTRACT

The genetic algorithm is an adaptive search method that has the ability for a smart search to find the best solution and to reduce the number of trials and time required for obtaining the optimal solution.

The aim of this paper was to study the behavior of different types of crossover operators in the performance of GA. We have also studied the effects of the parameters and variables (crossover probability, mutation rate, population size and number of generation) for controls the algorithm. This work accumulated some types of crossover operators to be a reference to all researchers; it was implemented on Optimization of a function. We investigate to explore the role of crossover in GAs with respect to this problem, by using a comparison study of the iteration results obtained from change the parameters values (crossover probability, mutation rate, population size and number of generation).

The experimental results reported will show more light into how crossover effects the GAs search power in the context of optimization problems.

The work explains the role of crossover operators in GAs and it shows the iteration results obtained with implementation in Delphi version 6.0 visual programming language exploiting the object oriented tools of this language.

INTRODUCTION

The goals of creating **Artificial Intelligence** (AI) and artificial life can be traced back to the very beginnings of the computer age. The earliest computer scientists (Alan Turing, John von Neumann, Norbert Wiener, and others), were motivated in large part by visions of imbuing computer programs with intelligence, with the life-like ability to self-replicate, and with adaptive capability to learn and to control their environments. These early pioneers of computer science were as much interested in biology and psychology as in electronics, and they looked to natural systems as guiding metaphors for how to achieve their visions. These biologically motivated computing activities have waxed and waned over the years, but since the early 1980s they have all undergone a resurgence in the computation research community. The first has grown into the field of neural networks, the second into machine learning, and the third into what is now called "**evolutionary computation**", of which genetic algorithms are the most prominent example [1].

Genetic algorithms (GAs) are a part of evolutionary computing, which is a rapidly growing area of artificial intelligence. GAs are inspired by Darwin's theory about evolution. Simply said, solution to a problem solved by genetic algorithms is evolved.
GAs were first suggested by John Holland and developed by him and his students and colleagues in the seventies. This lead to Holland's book "Adoption in Natural and Artificial Systems" published in 1975 [2].Over the last twenty years, it has been used to solve a wide range of search, optimization and machine learning problems. Thus, the GA is an iteration procedure, which maintains a constant size population of candidate solution [2]. In 1992 John Koza has used GA to evolve programs to perform certain tasks. He called his method "**genetic programming**" (GP) [3].

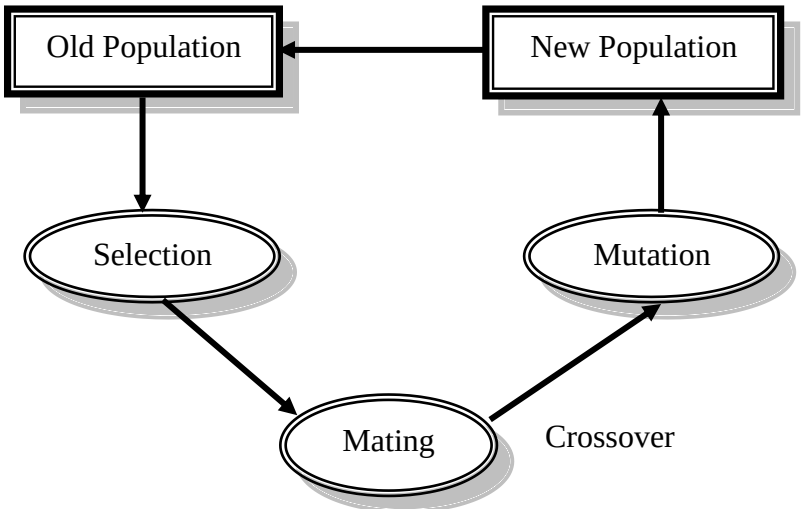
GENETIC ALGORITHM

Genetic Algorithms (GA's) are search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string

structures with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search. In every generation, a new set of artificial creatures (strings) is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure. While randomized, GA's are no simple random walk. They efficiently exploit historical information to speculate on new search point with expected improved performance [4].

GA's attempt to identify optimal solution by applying the techniques of natural selection to a population of solutions, the solutions are evaluated, the bad solutions are killed, and the remaining solutions are recombined (mate) to form a new generation of solution. The general principle is that of Darwinism the good traits will survive overtime as they are less likely to be on solutions that are killed a generation. Over a number generation in an overall increase in the quality of solution in the population.

Thus, a GA is an iterative procedure, which maintains a constant size population of candidate solution. During each iteration step (Generation) the structures in the current population are evaluated, and, on the basic of those evaluations, a new population of candidate solutions formed. The basic GA cycle based on the three processes (selection, mating and mutation) as shown in figure(1) [5].



Figure(1) The basic cycle of GA.

An abstract view of the GA is:

```
Generation=0;
Initialize G(P); {G=Generation ; P=Population}
Evaluate G(P);
While (GA has not converged or terminated)
        Generation = Generation + 1;
        Select G(P) from G(P-1);
        Crossover G(P);
        Mutate G(P);
        Evaluate G(P);
End (While)
   Terminate the GA [2].
```

CODING SCHEME

The first step in writing a GA is to create a coding scheme. In this step, domain knowledge is embedded in the abstract representation. Thus, a coding scheme is a method for expressing a solution in a string called chromosome or organism.

An organism of length m is a vector of the form $x_1,x_2,\ldots,x_m$, where each x_i is an allele or gene. The domain of values from which xi is chosen is called alphabet of the problem, and the number of different characters in the alphabet is called order [6].

Many successful type of coding scheme have been discovered, such as [7]:

1. Binary Encoding

Binary encoding is the most common, mainly because first works about GA used this type of encoding. In binary encoding every chromosome is a string of bits, 0 or 1.

Chromosome A	101100101100101011100
Chromosome B	111110000111000101011

Binary coding gives many possible chromosomes even with a small number of all. On the other hand, this encoding is often not natural for many problems and sometimes corrections must be made after crossover and/or mutation, for example the Knapsack problem.

2. Permutation (Integer) Encoding

Permutation encoding is only useful for ordering problems. Even for this problems some types of crossover and mutation corrections must be made to leave the chromosome, consistent i.e. have real sequence in it.

Chromosome A	1 5 3 2 6 4 7 9 8
Chromosome B	8 5 6 7 2 3 1 4 9

3. Value (Real) Encoding

Direct value encoding can be used in problems, where some complicated value, such as real numbers are used. Use of binary encoding for this type of problems would be very difficult. In value encoding, every chromosome is a string of some values. Values can be connected to problem, form numbers, real numbers or chars to some complicated objects.

Chromosome A	1.2345 5.3243 0.4556 2.4545
Chromosome B	A B C D E F I J E F B D

Value encoding is very good for some special problems, such as finding weights for neural network.

4. Tree Encoding

Tree encoding is used mainly for evolving programs or expressions, for Genetic Programming (GP).

In tree encoding every chromosome is a tree of some objects, such as functions or commands in programming language.

Chromosome A	(+ x (/ 5 y))
Chromosome B	(do until step wall)

THE TYPES OF CROSSOVER OPERATORS

The mating operation for GA creates variation by producing new offspring that consists of part taken from each parent [3]. This operation uses the mating pool as parents of the next generation. It walks through the following algorithm: [8]

Select two chromosomes, by the some selection method used for reproduction operation, the two chromosomes are A and B:

A = a_1 a_2 a_3 … a_n

B = b_1 b_2 b_3 … b_n

Select the appropriate mating method for these two chromosomes (one-point, two-point, PMX, CX, …, etc).

The result is two new chromosomes.

These are several numbers mating operation depends on the method of mating, and each one is appropriate for a particular application, the familiar operator is crossover.

Crossover operator defined probability, the higher crossover probability, the more quickly new individuals are introduced into the population. The low crossover probability causes search stagnation [7].

Depending on the representation of variables of the individuals the following algorithms can be applied. The binary representation traditionally used in GA's and its offers the maximum number of schemata per bit of information of any coding and consequently the bit string representation of solutions has dominated genetic algorithm research.

One (Single) Point Crossover

In single-point crossover select randomly one point in the range [1,2,...,N-1], where N is the length of the string. The first offspring is produced by deleting the crossover fragment of the first parent from the first and inserting the crossover fragment of the second parent at the crossover point of the first point. The second offspring is produced in a symmetric manner (see Figure (2)) [3,9].

For example :

If we have the following strings:

Individual1 = a_1 a_2 a_3 ... a_n

Individual2 = b_1 b_2 b_3 ... b_n

And we gain randomly, *pos*=r, then:

Offspring1 = a_1 a_2 a_3 ... a_r b_{r+1} ... b_n

Offspring2 = b_1 b_2 b_3 ... b_r a_{r+1} ... a_n

Where *pos* denotes the position, and r integer number in the range length of offspring.

In binary encoding, consider the following two individuals with 11 binary variables each:

Individual1 0 1 1 1 0 0 1 1 0 1 0

Individual2 1 0 1 0 1 1 0 0 1 0 1

The chosen crossover position 5. After crossover the new individuals are created:

Offspring1 0 1 1 1 0 | 1 0 0 1 0 1

Offspring2 1 0 1 0 1 | 0 1 1 0 1 0

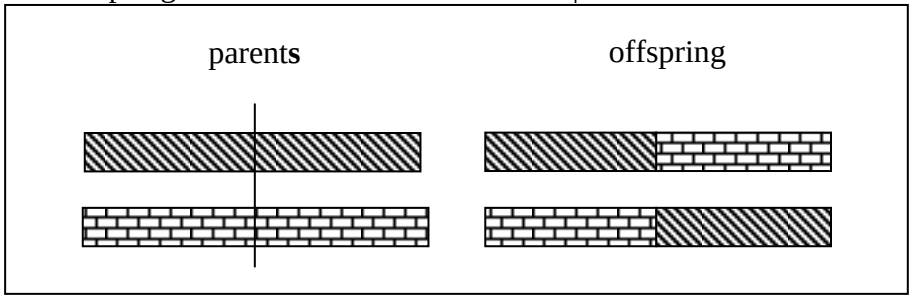


Figure (2) Single-point crossover.

Holland used this operator so much because it was very fast; but it was poor when the chromosomes are symmetric [8].

Two Point Crossover

Here we need two random points P1 and P2 (see Figure (3)) [8].

For example:

Consider the following two individuals with 11 binary variables each:

Individual1 0 1 1 1 0 0 1 1 0 1 0

Individual2 1 0 1 0 1 1 0 0 1 0 1

The chosen crossover position is:

And we gain, randomly P1=i, and P2=j, then:

Offspring1 = a_1 a_2 a_3 … a_{i-1} b_i ... b_j a_{j+1} ... a_n

Offspring2 = b_1 b_2 b_3 … b_{i-1} a_i ... a_j b_{j+1} ... b_n

In Binary encoding two point crossover

crossover position are i=3, j = 7

After crossover the new individuals are created:

Offspring1 0 1 1 | 1 0 1 0 | 0 1 0 1

Offspring2 1 0 1 | 0 1 0 1 | 1 0 1 0

This operator is used with long chromosomes, where such modern genetic algorithm using it because it was fast and efficient [9].

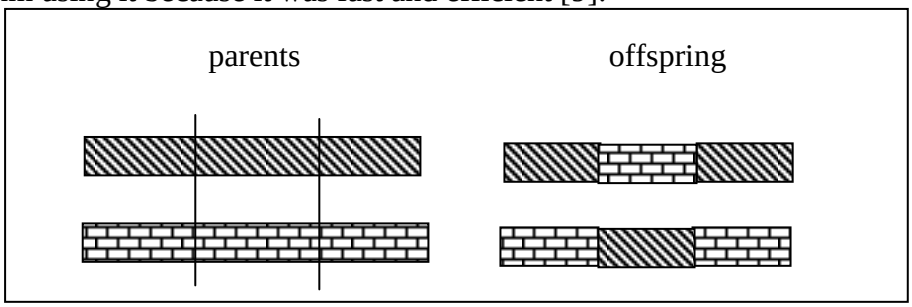


Figure (3) Two-point crossover

Multi-Point Crossover

For multi-point crossover, m crossover positions k[1,2,...,N-1], i=1:m N: number of variables of an individual, are chosen at random with no duplicates and sorted in ascending order. Then, the variables between successive crossover points are exchanged between the two parents to produce two new offspring. The section between the first variable and the first crossover point is not exchanged between individuals [9]. Figure (4) illustrates this process.

Consider the following two individuals with 11 binary variables each:

Individual1 0 1 1 1 0 0 1 1 0 1 0

Individual2 1 0 1 0 1 1 0 0 1 0 1

The chosen crossover positions are:

cross pos. (m=3) 2 6 10

After crossover the new individuals are created:

Offspring1 0 1 | 1 0 1 1 | 0 1 1 1 | 1

Offspring2 1 0 | 1 1 0 0 | 0 0 1 0 | 0

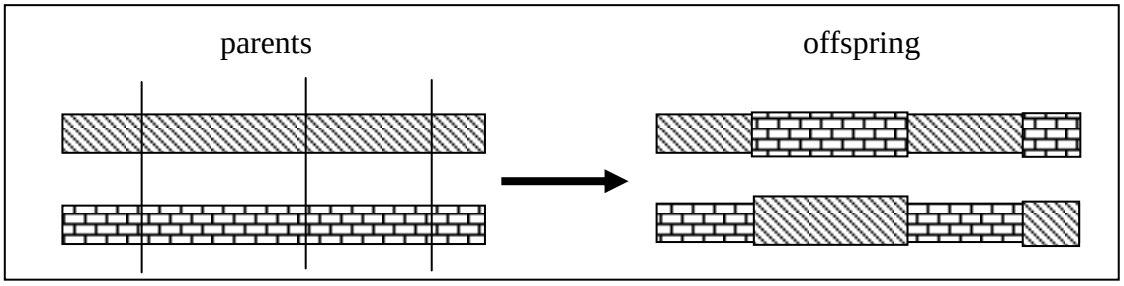


Figure (4) Multi-point crossover.

Simplex Crossover

Renders and Bersini [10] experimented with simplex crossover for numerical optimization problems: this crossover involves computing centered of group of parents and moving from the worst individuals beyond the centroids point.

For example:

Individual1	1 0 1 1 0 0 1
Individual2	0 0 1 0 1 0 0
The worst	0 0 1 1 0 1 1
Offspring	1 0 1 0 1 0 0

Uniform Crossover

A further generalization of one point, two point, and multi point crossover is the uniform crossover [11,12], for each bit in the first offspring it decides (with some probability p) which parent will contribute its value in that position. The second offspring would receive the bit from the other parent.

For example

For p=0.5 (0.5- uniform crossover), the strings

Individual1	0 0 1 0 0 0 1 1 0 1 0 0 1
Individual2	1 1 1 0 1 1 0 0 0 1 0 0 0

May produce the following pair offspring

Offspring1	01 01 12 01 12 12 02 11 01 12 01 01 02
Offspring2	12 12 11 02 01 01 11 02 02 12 02 02 12

Where subscripts (1 and 2) indicate the parent (Individual1 and Individual2 respectively).

For a given bit, if p=0.1(0.1- uniform crossover), two strings Individual1 and Individual2 may produce:

Offspring1	0 0 1 0 1 0 1 1 0 1 0 0 1
Offspring2	1 1 1 0 0 1 0 0 0 1 0 0 0

Since the uniform crossover exchanges bits rather than relative location. For some problems [12] this ability out weights the disadvantage destroying building blocks. However, for the problems the uniform crossover was inferior to two-point crossover.

Shuffle Crossover

Shuffle crossover [13] is related to uniform crossover. A single crossover position (as in single-point crossover) is selected. But before the variables are exchanged, they are randomly shuffled in both parents. After recombination, the variables in the offspring are unshffled. This removes positional bias as the variables are randomly reassigned each time crossover is performed.

For example :

Individual1	0 1 0 1 0 1 0 1 0 1
Individual2	1 1 1 1 0 0 0 0 1 1

Shuffle: (1 to 3, 3 to 6, 6 to 2, 2 to 9 , 9 to 10 ,10 to 1)

Offspring1	1 1 0 1 0 0 0 1 1 0
Offspring2	1 0 1 1 0 1 0 0 1 1

After applied the crossover operator, the two offspring's will be:

Offspring1	1 1 0 1 0 1 0 0 1 1
Offspring2	1 0 1 1 0 0 0 1 1 0

Finally, When taking unshuffled step we get:

Offspring1	0 1 1 1 0 1 0 0 1 1
Offspring2	1 1 0 1 0 0 0 1 0 1

THE ROLE OF CROSSOVER IN GA

In this paper we will try to study the effect of different types of crossover operators on certain known problem which where solved using genetic algorithm, such as optimization of a function. This problem was chosen according to different factors such as representation of the problem (which have a great influence on genetic algorithm) can be applied more efficiently. Furthermore, this problem chosen since they own a high complexity (the size and the shape of the search space), which its, cannot be solved using traditional known searches, like exhaustive search method.

Problem Definition

There is a large class of interesting problems for which no reasonably fast algorithms have been developed. Given such a hard optimization problem it is often possible to find an efficient algorithm whose solution is approximately optimal. We discuss the basic features of a GA for optimization of a simple function.

$$f(x_1,x_2)= 21.5 + x_1 . \sin(4\pi x_1) + x_2 . \sin(20\pi x_2) \quad \ldots(1)$$

where $-3.0 \le x_1 \le 12.1$ and $4.1 \le x_2 \le 5.8$.

Since x_1, x_2 are real numbers, this implies that the search space can be huge and that traditional methods can fail to find the optimal solution [14,15].

Problem Representation

To apply the GA for maximizing $f(x)$, a genetic representation of solution to the problem must be appropriately chosen first. The Simple GA uses the binary representation in which each point is described by a chromosome vector coded as a binary string. We use a binary vector as a chromosome to represent real values of the variable x, the length of the vector depends on the required precision, which in this example, is six places after the decimal point.

The domain of the variable x_1 has length 15.1; the precision requirement implies that the range [-3.0,12.1] should divided into at least 15.1×10000 equal size ranges. This means that 18 bits are required as the first part of the chromosome:

$2^{17} < 151000 < 2^{18}$

The domain of the variable x_2 has length 1.7; the precision requirement implies that the range [4.1,5.8] should divided into at least 1.7×10000 equal size ranges. This means that 15 bits are required as the second part of the chromosome:

$2^{14} < 17000 < 2^{15}$

The total length of a chromosome (solution vector) is then m=18+15=33 bits; the first 18 bits code x_1 and remaining 15 bits (19–33) code x_2.

Let us consider an example chromosome: (010001001011010000111110010100010) corresponds to (x_1,x_2) = (1.052426,5.755330). The fitness value for this chromosome is:

$$f(1.052426,5.755330) = 20.252640$$

Initial Population

To optimize the function f using GA, we create a population of pop_size chromosomes. All 33 bits in all chromosomes are initialized randomly.

Evaluation function

Evaluation function for binary vector v is equivalent to, the function f:

$$eval(v) = f(x),$$

where the chromosome v represents the real value x.

During the evaluation phase we decode each chromosome and calculate the fitness s function from(x_1,x_2) values just decode.
For example, three chromosomes:

v_1 = (10011010000000111111110100110111111),
v_2 = (11100010010011011100101010011010),
v_3 = (00001000011001000000101011101101),

correspond to values x_1 and x_2 respectively. Consequently, the evaluation function would rate them as follows:

$eval(v_1) = f$(6.084492,5.652242) = 26.0196001
$eval(v_2) = f$(10.348434,4.380264) = 27.580015
$eval(v_3) = f$(-2.516603,4.390381) = 19.526329

Clearly, the chromosome v_1 is the best of the three chromosomes, since its evaluation returns the highest value.

Genetic Operators

1. **Selection Operator**

   In this selection part of the GA, the survivor set is chosen so as to keep not only the fittest points but also some unavoidable points. Roulette wheel sums up the fitness' of all individuals and calculates each individual percentage of the total fitness. The percentage of the total fitness is then used as the probabilities to select N individuals from the set population and copy them into the set selected-parents.

2. **The Mating Crossover Operator**

   Individuals from the set selected-parents are mated to generate offspring's for the next generation. The two parents generate two offspring's using a crossover operation. For this example, to illustrate the crossover operator on chromosome with a crossover with probability Pc, we generate a random integer number pos from the range 1..32. The number *pos* indicate the position of the crossing point. The pair of chromosomes is:

   v_1 = (10011010000000111111110100110111111),
   v_3 = (00001000011001000000101011101101),

and the generated number *pos*=9. These chromosomes are cut after the 9th bit and replaced by a pair of the offspring:
the two resulting offspring are:

v_1'= (10011010001100100000101011101101),
v_3'= (00001000000000111111110100110111111),

these offspring evaluate to

$f(v_1')$=29.741, $f(v_3')$=36.22971.

3. **Mutation Operator**

   Mutation is a random change of one or more genes (positions in a chromosome) with a probability equal to the mutation rate Pm a gene is changed/swapped, i.e 0◊1 and 1◊0. The probability for a mutation is usually kept small. Assume that the fifth gene from the v3 chromosome was selected for a mutation. Since the fifth gene in this chromosome is 1, it would be flipped into 0. So the chromosome v_3 after this mutation would be:

   v_3'= (00000000000000111111110100110111111).

4. **Genetic Parameters**

   For this particular problem, Michalewicz [16] used the following parameters: population size pop_size=20, probability of crossover Pc=0.25, probability of mutation Pm =0.01.

Experimental Results

In table (1) we provide the generation number for which we noted improvement in the evaluation function, together with the value of the function. The best chromosome after (200) generations was:

v_{max} = (11111010100111111111110101001111)

Which corresponds to a value (x_1,x_2)=(12.002307,5.712009), and $f(v_{max})$ is slightly larger than 38.629.

Table (1) Results of 200 generations for optimization of a function problem.

Generation number	Evolution function
0	36.290817
1	37.759151
6	37.338592
15	37.758549
37	37.973587
54	38.048171
73	38.208092
91	38.279362
114	38.607517
192	38.629047

For this problem, a simulation has been constructed in order to apply the GA, using the crossover parameters mentioned above, the following results are be obtained:

v_{max} = (01101111111111111110110111111111111)

Which corresponds to a value (x_1,x_2)=(12.0995,5.7995), and $f(v_{max})$=38.761.

THE EFFECT OF DIFFERENT TYPES OF CROSSOVER

In this part, we will try to study the effect of applying different types of crossover on the reported algorithms, on their performance, speed, and ability to find the solution.

To see the effect of using different types of crossover operators on this problem, Michalewicz [16] used the One-Point crossover depending on the following parameters: Pc=0.25, Pm=0.01, Pop_size=20 NG=1000. Table (2) describes the comparison study of the iterations results between the above crossover and the other kinds which are implemented on this problem. In addition, the table shows the average of iterations and finesses results of (10) runs.

Table (2) comparison study of One-Point crossover and other kinds.

Crossover	NG	Fitness
1-P	361	38.724
2-P	455	38.663
M-P	395	38.716
Un	455	38.649
SIMP	351	38.750
SHUF	310	38.761

From table (2), the average iterations results shows that the Shuffle, (followed by Simplex and One-Point) crossover operator is the best, because it makes a randomly shuffle in both parents before they are exchange. The Uniform and Two-Point

crossover operators are the worst, because there is a destruction in the building blocks (good genes).

THE EFFECT OF DIFFERENT PARAMETERS ON CROSSOVER

The crossover is an extremely important component of a genetic algorithm. Many GA practitioners believe that if we delete the crossover operators from a GA the result is no longer a GA. In fact; many GA practitioners believe that the use of a crossover operator distinguishes GA from all other optimization algorithms. This section will tries to study the effect of different genetic parameters on the performance of the proposed algorithms.

1. **Studying the Effect of Probability of Crossover (Pc) on it**

   This operator owns a major role in GA, so specifying the probability of crossover, not done randomly, but it must depend on many runs of the simulation this problem, in order to tune this operator to obtain the fine probability of crossover. We will apply this operator with different values and so other operators.

   In this problem, table (3) shows that the population size, the number of generation and the mutation rate are all fixed, while the crossover probability takes the values 0.0, 0.5 and 0.8.

   Table(3) Crossover probability effect when NG=750,popsize=50,Pm=0.05.

Crossover	Pc	Iteration	Max. Fit.	Min. Error
ONE_POINT	0.0	695	38.750	0.303
	0.5	487	38.734	0.303
	0.8	119	38.740	0.121
TWO_POINT	0.0	741	38.688	0.545
	0.5	207	38.714	0.424
	0.8	482	38.725	0.364
MULTI_POINT	0.0	744	38.570	0.242
	0.5	130	38.732	0.182
	0.8	605	38.732	0.303
UNIFORM	0.0	737	38.730	0.242
	0.5	552	38.758	0.242
	0.8	524	38.714	0.364
SIMPLEX	0.0	736	38.758	0.242
	0.5	225	38.718	0.364
	0.8	666	38.755	0.303
SHUFFLE	0.0	688	38.712	0.182
	0.5	144	38.761	0.061
	0.8	427	38.746	0.242

From table (3) we notes the following analytic aspects:

a. The worst iteration results (high iteration levels) are observed when the effect of the crossover probability is eliminated (when Pc=0.0), which means the search way as good as random.
b. Notice that, when Pc=0.0, the NG gets the highest value among the other values of NG from other values of Pc different from 0.0.
c. When the One-Point crossover be used with higher crossover probability than the one used before, the iteration results is be improved.

d. In the most kinds of the used crossovers, the iteration results are be improved when using Pc ≥ 0.5.
e. The results appeared that the Shuffle and Simplex crossover operators are better than other kinds of crossover operators, which are used to solve this problem.

2. **Studying the Mutation Rate (Pm) Effect on Crossover**

This operator plays a dual role in genetic algorithm, it provides and maintains diversity in a population, so that other operators can continue to work and it can work as a search operator in its own right. We will apply this operator with different numbers of mutation rate and so other operators.

In this problem, table (4) shows that the population size, the number of generation and the crossover probability are all fixed, while the mutation rate takes the values 0.01, 0.05 and 0.08.

Table (4) Mutation rate effect when NG= 750, popsize = 50, Pc = 0.5.

Crossover	Pm	Iteration	Max. Fit.	Min. Error
ONE_POINT	0.01	648	38.684	0.485
	0.05	278	38.728	0.364
	0.08	405	38.754	0.364
TWO_POINT	0.01	282	38.595	0.303
	0.05	278	38.631	0.364
	0.08	583	38.760	0.121
MULTI_POINT	0.01	210	38.585	0.303
	0.05	507	38.620	0.424
	0.08	543	38.745	0.364
UNIFORM	0.01	113	38.477	0.485
	0.05	258	38.741	0.121
	0.08	574	38.758	0.242
SIMPLEX	0.01	212	38.735	0.303
	0.05	191	38.738	0.182
	0.08	666	38.753	0.242
SHUFFLE	0.01	475	38.750	0.242
	0.05	121	38.754	0.242
	0.08	249	38.754	0.364

From table (4) we notes the following analytic aspects:

a. In the most kinds of the used crossovers, including the One-Point crossover, the iteration results is be improved when using Pc=0.5.
b. Its be certain from the results that Pm prefer to be minimum as possible (Pm≤0.1).

3. **Studying the Population Size (Popsize) Effect on Crossover**

The operation of determines the population size is depending on the nature of the problem, that we require solving it. When increasing the complexity of search space then it needs to a large population. In general, we can not estimate the real size but we could given the domain of it. In this operator we use different populations with other operators.

In this problem, table (5) shows that the number of generation, the mutation rate and the crossover probability are all fixed, while the population size takes the values 20, 50 and 90.

Table (5) Population size effect when NG = 750, Pm = 0.05, Pc = 0.5.

Crossover	Popsize	Iteration	Max. Fit.	Min. Error
ONE_POINT	20	420	38.630	0.424
	50	262	38.727	0.303
	90	451	38.759	0.242
TWO_POINT	20	674	38.689	0.424
	50	244	38.746	0.303
	90	146	38.737	0.242
MULTI_POINT	20	289	37.144	0.424
	50	143	38.744	0.303
	90	228	38.734	0.364
UNIFORM	20	244	38.365	0.303
	50	117	38.719	0.182
	90	182	38.751	0.242
SIMPLEX	20	112	38.699	0.364
	50	206	38.738	0.424
	90	322	38.743	0.424
SHUFFLE	20	456	38.725	0.485
	50	102	38.751	0.242
	90	420	38.749	0.242

From table (5) we notes the following analytic aspects:

a. All kinds of the used crossover operators, the best results obtained when popsize=50.
b. The best two crossover operators are the Shuffle and Simplex crossover operators.

4. **Studying the Number of Generation (NG) Effect on Crossover**

NG operator considered as one of the important operators of GA, since its different from the other operators that its may finds a solution in its high level of iterations. In the same time, this operator can be considered as criteria of the crossover efficiency, or in general, the algorithm efficiency in solving the problem. Table (6) shows that the popsize, the Pm and the Pc are all fixed, while the NG takes the values 500,750 and 1000.

Table(6) Number of generation effect when popsize=50, Pm=0.05, Pc=0.5.

Crossover	NG	Iteration	Max. Fit.	Min. Error
ONE_POINT	500	281	38.734	0.364
	750	230	38.762	0.121
	1000	518	38.743	0.424
TWO_POINT	500	221	38.670	0.182
	750	190	38.754	0.242
	1000	313	38.757	0.242
MULTI_POINT	500	229	38.731	0.303
	750	192	38.686	0.485
	1000	320	38.739	0.303
UNIFORM	500	209	38.697	0.303
	750	278	38.686	0.485
	1000	72	38.690	0.364
SIMPLEX	500	167	38.740	0.303
	750	127	38.748	0.364
	1000	192	38.736	0.242
SHUFFLE	500	474	38.752	0.182
	750	107	38.728	0.424
	1000	310	38.760	0.182

From table (6) we notes the following analytic aspects:

a. Its important to mention that, there is a relation between the execution time and the control parameters (number of generation and population size).
b. In optimization of a function problem, its not necessary to obtain results in the high levels of generations.

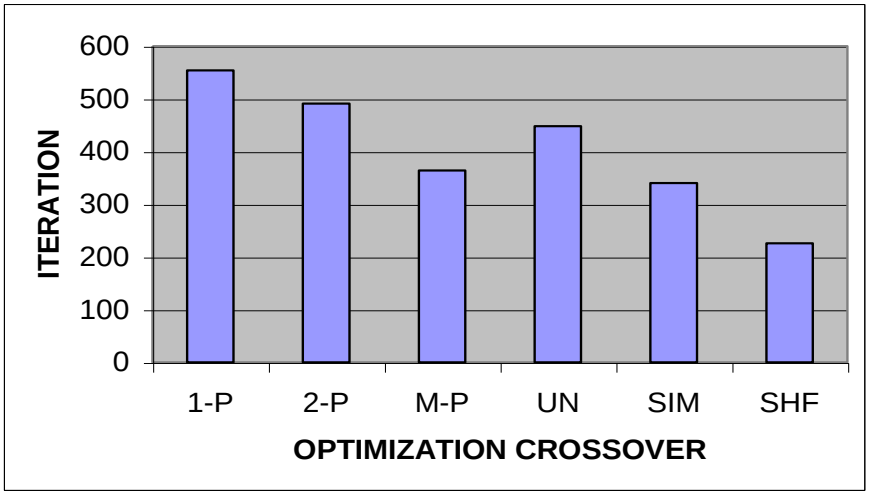
PARAMETRIC STUDY FOR ALL CROSSOVER OPERATORS

From our experiences and experimental results, the best parameters values for all crossover kinds are be chosen to make a comparison study for each kind of crossover for each problem for several runs. The minimum iteration results are tabled in tables with statistical diagram to illustrate these results.

For this problem, the best values are: NG = 750, popsize = 50, Pm=0.05 and Pc=0.5. Table (7) shows the comparison study of iterations results for 10 runs and Figure (5) illustrates the statistical diagram of the comparison study results.

Table (7) Comparison study of crossover iterations results for optimization of a function problem.

Crossover	NG	Fitness
1-P	551	38.599
2-P	489	38.644
M-P	364	38.727
Un	430	38.702
SIMP	326	38.736
SHUF	288	38.761



Figure(5) Statistical diagram of the comparison study results for optimization of a function problem.

In this particular problem, from table (7) and Figure (5), we notes that the best operators is the Shuffle crossover because it makes a randomly shuffle in both parents before they are exchange.

CONCLUSIONS

This work concludes the following aspects:

1. This paper can be considered as a reference, which contains a description on all types of crossover, which can rarely found in the literature.
2. This research found that for the optimization of a function, the Shuffle crossover was the best to be applied.
3. For the parametric study, the research concludes:
   a. The worst iteration results (high iteration levels) are be obtained when the effect of the crossover probability is eliminated (when Pc=0.0), since the search in this state is closed to the random search.
   b. For the population size parameters, the result reveal that due to the founder effect, the GA's can not always locate the peaks of the fitness landscape, even with higher crossover rate.
   c. Although, that the mutation rate parameter preferred to be chosen as minimum as possible ($Pm \leq 0.1$), but its still related to the problem which be discussed.
   d. It's naturally that, there is a relation between the number of generation and population size. This parameter specification is related to the problem which be discussed.
4. In this study, the correlation between the binary crossover operators and the real crossover operators has been observed.

FUTURE WORKS

We present here, some of the future works which can be implemented in the future:

1. We have been concentrating on the study of the role of number of crossover operators, our future work will extend to the study of other kinds of crossover operators.
2. Since the results obtained for crossover strategies are encouraging. Another research line is to investigate other plausible adaptive crossover schemes; in order to develop more robust GA's for optimization task.
3. In future work we can replace the types of binary representation by real representation applications.
4. In future work we can study the several selection methods for GA's such as ranking, tournament and proportional on effectiveness of results of role crossover.
5. More research on the combination of the types and levels of adaptation needs to be done as this could lead to significant improvements to finding good solutions on the speed of finding them.
6. In the future work we can apply the crossover operators on the problems that require a large complexity.

REFERENCES

[1]. Mitchell M., "**An Introduction of Genetic Algorithms**", Abroad Book 1998.

[2]. Syswerda, G., "**Uniform Crossover in Genetic Algorithms**", Proceedings of the 3th International Conference on Genetic Algorithms, San Mateo pp. 2-9, 1989.

[3]. Vose, Michael D, "**The Simple Genetic Algorithm: Foundations and Theory**", MIT Press, Cambridge, MA, 1999.

[4]. Spears, W.M., and De Jong, K. A., "**On the Virtues of Parameterized Uniform Crossover**", Proceedings of the 4th International Conference on Genetic Algorithms, San Mateo pp. 230-236, 1991.

[5]. Grefenstette J. J., "**Optimization of Control Parameters of Genetic Algorithm**" Genetic Algorithm; IEEE Computer Society, 1992.

[6]. Obitko M.,"**Genetic Algorithm**",http://www.obitko.email%c/source.html, 1998.

[7]. Oliver, J. M., Smith, D. J., and Holland, J. R.C., "**A Study of Permutation Crossover Operators on the Traveling Salesman Problem**", Proceedings of the 2nd International Conference on Genetic Algorithms pp. 224-230, 1987.

[8]. Bies, Robert R. and et al, "**A Genetic Algorithm-Based, Hybrid Machine Learning Approach to Model Selection**". Journal of Pharmacokinetics and Pharmacodynamics (Netherlands:Springer):196–221, 2006.

[9]. Goldberg, David E, "**The Design of Innovation: Lessons from and for Competent Genetic Algorithms**", Addison-Wesley, Reading, MA, 2002.

[10]. Fogel, David B, "**Evolutionary Computation: Toward a New Philosophy of Machine Intelligence**", IEEE Press, Piscataway, NJ. Third Edition, 2006.

[11]. Michalewicz, Zbigniew, "**Genetic Algorithms + Data Structures = Evolution Programs**", Springer-Verlag, 1999.

[12]. Poli, R., Langdon, W. B., McPhee, N. F., "**A Field Guide to Genetic Programming**", Lulu.com, freely available from the internet. ISBN 978-1-4092-0073-4, 2008.

[13]. Schmitt, Lothar M, "**Theory of Genetic Algorithms**", Theoretical Computer Science 259: 1-61, 2001.

[14]. Wright A.H., "**Genetic Algorithms for Real Parameters Optimization**", Foundation of Genetic Algorithms 1st Workshop on the Foundation of Genetic Algorithms and Classifier Systems, pp. 205-218, 1991.

[15]. Syswerda, G., "**Schedule Optimization Using Genetic Algorithms**", pp. 332-349, 1991.

[16]. Schmitt, Loather M, "**Theory of Genetic Algorithms II**", models for genetic operators over the string-tensor representation of populations and convergence to global optima for arbitrary fitness function under scaling, Theoretical Computer Science 310: 181-231, 2004.