International Journal of Advances in Mathematics
Volume 2018, Number 1, Pages 1-14, 2018
eISSN 2456-6098
© adv-math.com

Optimization through Genetic Algorithm with a New and Efficient Crossover Operator

Abid Hussain[1,*], Yousaf Shad Muhammad[1] and Asim Nawaz[1]
[1] Department of Statistics, Quaid-i-Azam University, Islamabad, Pakistan.
E-mail: abid0100@gmail.com

ABSTRACT. Selection criteria, crossover and mutation are three major operators involved in the genetic algorithm's performance. A lot of work has been done on these operators but the crossover has a significant role in the operation of genetic algorithms. A number of crossover operators have been introduced in literature and all have different impact on genetic algorithm's performance. In this study, we proposed a new crossover operator to improve the performance of genetic algorithms. Proposed operator is applied along with some traditional crossover operators on seven benchmark problems. Simulation studies show a remarkable performance of the proposed crossover scheme which obtained a fast convergence and better results than existing or traditional crossover operators.

1 Introduction

Genetic Algorithms (GAs) are used a vocabulary borrowed from natural genetics developed by John Holland [1]. After Holland's work, his students made development in his idea with some new directions and today it is a powerful tool to solving search and optimization problems. A study about gas pipelines by Goldberg, proved that GAs has practically uses and applications in engineering [2]. GAs operate in cycles called generations that produce successive population by survival-of-the-fittest selection followed by genetic operators and other die-off. With the help of these operators, GAs consumes less memory than other optimization algorithms and work efficiently.

* Corresponding Author.
Received September 26, 2017; revised November 02, 2017; accepted November 09, 2017.
2010 Mathematics Subject Classification: 03E99, 03B99.
Key words and phrases: Genetic algorithms, crossover operators, benchmark functions, comparison.
This is an open access article under the CC BY license http://creativecommons.org/licenses/by/3.0/.

GAs operate with populations of strings (chromosomes), which are coded to represent some underlying parameters set. Selection procedure, crossover and mutation operators are applied to successive strings population to create new strings population. These operators don't involve nothing complex structure as the random number generation, strings copying and partially exchange and impressiveness. The two most commonly applied genetic search operators are crossover and mutation. To take information from two parents to produce offspring and continuously introduce a small variation to the next generation are crossover and mutation operators respectively. The flow chart that how GAs work is given in Figure 1.

The main objective of this article is to present the performance of crossover operators that have major impact in GAs process. The purpose of crossover operator is to vary the individuals quality by combining the desired characteristics from both parents. A comprehensive studied about various crossover operators and some suggestions to selecting among these operators have been introduced and reviewed [3]. Some of the traditional crossover operators are: one-point, two-point, multi-point, uniform, discrete, heuristic and arithmetic etc. An affected study of crossover operators on GAs performance which showed that with large search space, to using uniform crossover outperforms than a one-point crossover, which in turn outperforms than two-point crossover operator [4]. In general, however their theory also suggested that GAs with crossover should outperform than with out crossover operator(i.e., mutation only).

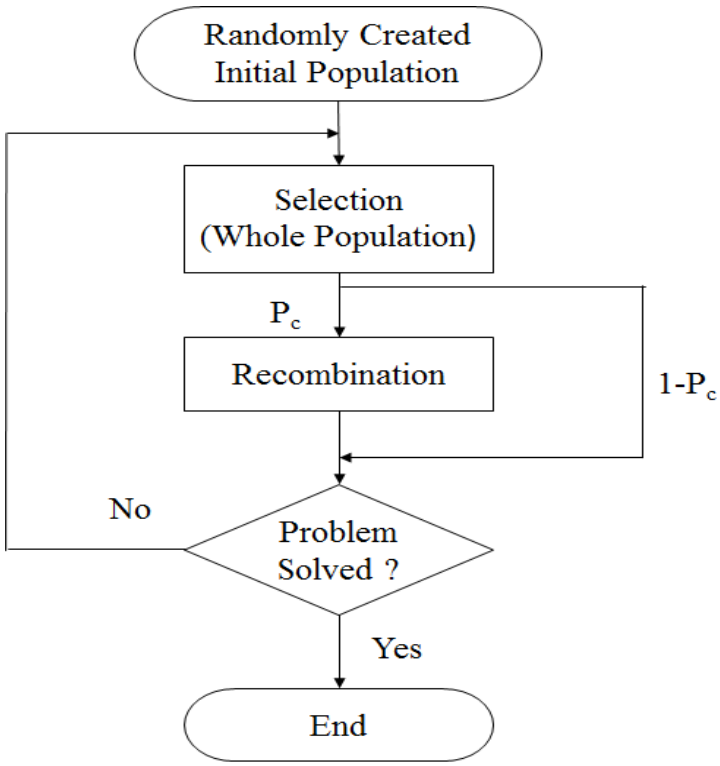

Figure 1: Flowchart of the Genetic Algorithm

A large number of crossover operators used in GAs with binary representation and results confirmed the high efficiency of uniform and two-point crossover operators [5]. GAs applied on the dimension optimization

problems to compare one-point, two-point and uniform crossover operators and results suggested that two-point crossover operator are best [6, 7, 8].

A comparison studied among the one-point, two-point, three-point and four-point crossover operators by Wu and Chow [9]. A suggested theory that multi-point crossover operator in term of fast progress becomes very slow when compares with one-point crossover [10]. A comparative study of different crossover operators with varying the population size and suggested that two-point crossover is performed better when population size is large and uniform crossover outperformed with small population size [11]. Syswerda showed in his study that uniform crossover operator is more efficient when compared with two-point crossover [12]. A study about the effect of different behavior of crossover operators and found improved results with combining the two crossover operators [13, 14].

To present the theoretical background about traditional crossover operators in Section 2, proposed approach in Section 3, benchmark functions in Section 4, experimental results and discussion in Section 5 and conclusion in Section 6.

2 Some Traditional Crossover Operators

There are a lot of crossover operators in GAs literature but most commonly used crossover operators as follows:

2.1 One-point Crossover Operator

The simplest crossover technique for GAs is one-point crossover operator. Using this technique, first select two parents and then randomly selects any one crossover point from the interval [1, *l*-1], where *l* is the length of string. All bits are interchanged between parents after selected crossover point to make offspring [15]. Given two parents and their resulting offspring with the crossover site between 5th and 6th bits have the following form:

Parent a	0	0	1	1	0	1	0	0
Parent b	1	0	0	1	0	0	0	1
Offspring a	0	0	1	1	0	0	0	1
Offspring b	1	0	0	1	0	1	0	0

2.2 Two-point Crossover Operator

In this approach, select any two points from the interval [1, *l*-1], where *l* is the length of string. After selected points, both strings are interchange between fixed points to make new offspring [15]. Given two parents and their resulting offspring with the crossover sites are between 3rd and 4th and 6th and 7th bits have the following form:

Parent a	0	0	1	1	0	1	0	0
Parent b	1	0	0	1	0	0	0	1
Offspring a	0	0	1	1	0	0	0	0
Offspring b	1	0	0	1	0	1	0	1

2.3 Uniform Crossover Operator

This is quite different from previous crossover operators and the idea was first used Syswerda [12]. After selected parents, introduce a new randomly generated binary crossover mask according to same length of the string. If it is a 0 in the binary crossover mask, the bit is copied from the second parent otherwise from the first parent. Given two parents and their resulting offspring have the following form:

Parent a	0	0	1	1	0	1	0	0
Parent b	1	0	0	1	0	0	0	1
Mask	1	0	0	1	1	0	1	1
Offspring a	0	0	0	1	0	0	0	0
Offspring b	1	0	1	1	0	1	0	1

2.4 Discrete Crossover Operator

Discrete crossover operator used random numbers to create one offspring from two parents. In this approach offspring selects genes from two parents according to uniform basis and random number decides that which parent's genes to take for offspring [16]. Given two parents and their resulting offspring has the following form:

Parent a	1	1	1	0	1	0	0	1
Parent b	1	0	0	0	1	0	1	1
Offspring	1	1	0	0	1	0	1	1

2.5 Arithmetic Crossover Operator

To produce offspring in arithmetic crossover operator is such a way that every bit of a offspring is convex combination of bits from its two parents. For two parents x_1 and x_2, the collection of all convex combinations is called a convex hull and define the offspring [17]. Given two parents x_1 and x_2 and their resulting offspring y_1 and y_2 have the form:

$$y_1 = \lambda_1 x_1 + \lambda_2 x_2$$
$$y_2 = \lambda_2 x_1 + \lambda_1 x_2$$

where,

$$\lambda_1 + \lambda_2 = 1$$

A special case when $\lambda_1 = \lambda_2 = 0.5$, called average crossover [18] and intermediate crossover [19]

2.6 Heuristic Crossover Operator

The heuristic operator [20] is different from other techniques because of the determining the direction of the search by using the values of objective function. It produces only one offspring and some time produce no offspring at

all. The scheme gives only offspring O_1 from two parents P_1 and P_2 according to the following rule:

$$O_1 = r(P_2 - P_1) + P_2$$

where r is a randomly generated number between 0 and 1, and P_2 is not worse than P_1.

It may be possible that this operator produces an infeasible offspring. In such situation, another random value of r is generated for new offspring. If there is no new solution meeting the constraints is found after t attempts, then the operator gives up and produces no offspring. It seems that this scheme contributes to the precision of the solution found; its major tasks are find the local tuning and searching in the most promising direction.

3 Proposed Crossover Scheme

In this section of the study, we present one newly developed crossover operator for GAs according to the concept of positive and negative criteria which are used in multiplication operations. Binary string represents with two digits only, zeros and ones so using this approach we consider zeros as negative and ones as positive resultant signs respectively or vice versa. Given two parents and their resulting offspring according to the multiplication operations have the following form:

Parent a	+	-	+	+	-	-	+	-
Parent b	-	+	-	+	+	-	+	+
Offspring	-	-	-	+	-	+	+	-

This approach produces only one offspring O from two parents P_a and P_b according to the following rule:

$$P_a = (a_1, a_2, ..., a_n)$$

$$P_b = (b_1, b_2, ..., b_n)$$

$$O = (o_1, o_2, ..., o_n)$$

Where,

$$o_i = \begin{cases} 1 & ; a_i = b_i \\ 0 & ; a_i \neq b_i \end{cases}$$

The following result representing in binary form of the approach is:

Parent a	1	0	1	1	0	0	1	0
Parent b	0	1	0	1	1	0	1	1
Offspring	0	0	0	1	0	1	1	0

We see later in results that this approach gives a fast convergence on some benchmark functions. To apply this crossover operator, we made a MATLAB function "xoverplusminus" for genetic algorithms tool as crossover operator for custom used. The code of our function to verify its fast convergence is given in the Algorithm 1:

```
function xoverkids = xoverplusminus...
(parents,option,GenomeLength,FitnessFcn,Unused,ThisPopulation)
nkids = length(parents)/2;
xoverkids = zeros(nkids,GenomeLength);
index = 1;
for i=1:nKids
r1 = parents(index);
index = index + 1;
r2 = parents(index);
index = index + 1;
xoverkids(i,:) = not (xor(ThisPopulation(r1,:) , ThisPopulation(r2,:)));
end
end
```

Algorithm 1: MATLAB code for xoverplusminus function

4 The Benchmark Functions

In this study, we are using seven multi-model benchmark functions that are most popular and used in several studies. The necessary information about these functions are as follows:

The first benchmark function (The Rosenbrock) is one of five DeJong's standard functions [21]. It is a classic optimization problem also known as a banana function because of its distinctive shape in a contour plot. It is stated as follows:

$$f(x) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 \tag{1}$$

subject to

$$-2.048 \leq x_1, x_2 \leq 2.048;$$

The global optimum lies inside a narrow, long, parabolic shaped flat valley and to find a minimum point is trivial. To find out the global optimum value of this function is difficult, that's why this problem has been continuously used as a benchmark to test the performance of new optimization techniques or algorithms. The function has "0" global optimum value at (1, 1).

The second benchmark function (The Rastrigin) takes from DeJong's standard functions [21] with the addition of cosine modulation in order to produce frequent local minima. This is highly multi-modal and difficult to find optimum due to regularly distributed of its minima locations. The optimization problem is stated as follows:

$$f(x) = 20 + x_1^2 - 10cos(2\pi x_1) + x_2^2 - 10cos(2\pi x_2) \tag{2}$$

subject to

$$-5.12 \leq x_1, x_2 \leq 5.12;$$

The function has a number of local optima and the global minimum value of the function is "0" at (0, 0).

Our third benchmark function (The Colville) has four dimensions and highly irregular pattern to convergence taken from [20]. It is stated as follows:

$$\begin{aligned} f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + \\ 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1) \end{aligned} \tag{3}$$

subject to

$$-10 \leq x_1, x_2, x_3, x_4 \leq 10;$$

This function is highly multi-modal and not easy to locate its optimum points because of its more dimensions. The function has a global minimum value of "0" at (1, 1, 1, 1).

The 2-D six-hump camel back function is our fourth global optimization test function. Within the bounded region are six local minima, two of them are global minima. This function is also a highly irregular pattern to convergence taken from [20]. It is stated as follows:

$$f(x) = (4 - 2.1x_1^2 + 1/3x_1^4)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2 \tag{4}$$

subject to

$$-3 \leq x_1 \leq 3 \quad and \quad -2 \leq x_2 \leq 2;$$

The function has a global minimum value of "-1.0316" at two different points (-0.0898, 0.7126) and (0.0898, -0.7126).

The fifth benchmark function (The Gold-Price) has taken from [22]. It is stated as follows:

$$\begin{aligned} f(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1^2 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \\ .[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)] \end{aligned} \tag{5}$$

subject to

$$-2 \leq x_1, x_2 \leq 2;$$

The function has a global minimum value of "3" at (0, -1).

The Easom test is our sixth benchmark function which has the global minimum fall in a small area relative to the search space taken from [23]. It is stated as follows:

$$f(x) = -cos(x_1)cos(x_2)exp[-(x_1-\pi)^2-(x_2-\pi)^2] \tag{6}$$

subject to

$$-100 \leq x_1, x_2 \leq 100;$$

The function has a global minimum at (π, π) with "-1".

Our last test function (The Matyas) is taken from [24]. It is stated as follows:

$$f(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2 \tag{7}$$

subject to

$$-10 \leq x_1, x_2 \leq 10;$$

. The global minimum is located at (0, 0) with "0".

5 Experimental Results and Discussion

We use genetic algorithm (GA) tool in MATLAB software to compare the proposed crossover operator with some of the traditional crossovers. One of the main difficulties in building a practical GA is in choosing suitable values for parameters such as population size, scaling function, selection criteria, elite count, probability of crossover (P_c), mutation function and function tolerance. The selections of parameter values are vary depending on the problem to be solved. For this empirical study we fixed some parameters of genetic algorithm. All fixed parameters in our study are given in the Table 1. The GA runs independently 20 time with 100, 500 and 1000 generations. In Table 2, we applied four traditional and one proposed crossover operator on seven benchmark problems and divide our experimental results in best, worst and average performances.

Table 1: Fixed Parameters for GA Tool

Population Size	20
Scaling Function	Proportional
Selection Criteria	Roulette Wheel
Elite Count	2.0
Crossover Fraction	0.8
Mutation Function	Constraint Dependent
Function Tolerance	1e-6

Table 2: Comparison of results among different crossover operators

Function	Generation	Result	One-point	Two-point	Arithmetic	Intermediate	Plus-minus
		Best	0.0175	0.0134	0.0010	0.0039	0.0000
	100	Worst	0.2307	0.3463	0.3036	0.4843	0.0000
		Average	0.0942	0.1222	0.0991	0.2101	0.0000
		Best	0.0000	0.0020	0.0000	0.0019	0.0000
Rosenbrock	500	Worst	0.1766	0.2510	0.2673	0.1748	0.0000
		Average	0.0722	0.1019	0.0822	0.0791	0.0000
		Best	0.0000	0.0000	0.0000	0.0017	0.0000
	1000	Worst	0.0167	0.0334	0.0461	0.0506	0.0000
		Average	0.0028	0.0092	0.0102	0.0204	0.0000
		Best	0.9942	0.9942	0.9942	0.0030	0.0000
	100	Worst	3.9767	4.9708	3.9766	1.8683	1.9937
		Average	1.8927	2.0342	1.5672	1.0521	1.0043
		Best	0.9942	0.9942	0.9942	0.0000	0.0000
Rastrigin	500	Worst	1.9983	3.9766	3.9766	1.9893	1.9893
		Average	1.4331	1.9782	1.3765	0.7891	0.8872
		Best	0.9942	0.9942	0.0000	0.0000	0.0000
	1000	Worst	1.9883	1.9883	1.9883	0.9942	1.9883
		Average	1.0921	1.5435	0.9164	0.4344	0.5455
		Best	0.0776	0.0617	0.1391	0.3034	0.0000
	100	Worst	20.543	13.272	9.2036	23.991	0.0000
		Average	9.9833	7.1210	6.7634	10.768	0.0000
		Best	0.0645	0.0413	0.0026	0.0923	0.0000
Colville	500	Worst	7.9683	9.1621	8.7313	5.4289	0.0000
		Average	3.0864	4.0238	3.8723	3.7641	0.0000
		Best	0.0228	0.0218	0.0012	0.0907	0.0000
	1000	Worst	6.1222	3.4365	6.0414	4.4014	0.0000
		Average	1.7694	1.5831	1.6328	1.9895	0.0000
		Best	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316
	100	Worst	-0.2155	-1.0316	-1.0316	-1.0314	-0.9631
		Average	-0.8287	-1.0316	-1.0316	-1.0316	-1.0043
		Best	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316
Six-hump	500	Worst	-1.0316	-1.0316	-1.0316	-1.0283	-1.0283
		Average	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316

Table 2: Continued.

Function	Generation	Result	One-point	Two-point	Arithmetic	Intermediate	Plus-minus
		Best	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316
	1000	Worst	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316
		Average	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316
		Best	3.0000	3.0000	3.0000	3.0000	3.0000
	100	Worst	30.009	33.158	37.436	30.000	31.089
		Average	6.8793	7.4472	8.8790	11.988	10.010
		Best	3.0000	3.0000	3.0000	3.0000	3.0000
Gold-Price	500	Worst	3.0000	30.000	30.007	10.131	20.059
		Average	3.0000	4.3811	4.9976	5.3110	3.8573
		Best	3.0000	3.0000	3.0000	3.0000	3.0000
	1000	Worst	3.0000	3.0000	30.000	3.0000	6.4342
		Average	3.0000	3.0000	3.5641	3.0000	3.2103
		Best	-0.9999	-0.9999	-0.9999	-0.9999	-0.9999
	100	Worst	-0.9997	-0.9997	-0.9997	-0.9997	-0.9999
		Average	-0.9998	-0.9998	-0.9998	-0.9998	-0.9999
		Best	-0.9999	-0.9999	-0.9999	-0.9999	-0.9999
Easom	500	Worst	-0.9999	-0.9999	-0.9999	-0.9999	-0.9999
		Average	-0.9999	-0.9999	-0.9999	-0.9999	-0.9999
		Best	-0.9999	-0.9999	-0.9999	-0.9999	-0.9999
	1000	Worst	-0.9999	-0.9999	-0.9999	-0.9999	-0.9999
		Average	-0.9999	-0.9999	-0.9999	-0.9999	-0.9999
		Best	0.0029	0.0041	0.0033	0.0023	0.0031
	100	Worst	0.0324	0.1624	0.0449	0.0871	0.0377
		Average	0.0196	0.0734	0.0297	0.0581	0.0203
		Best	0.0014	0.0034	0.0029	0.0019	0.0014
Matyas	500	Worst	0.0297	0.0877	0.0341	0.0614	0.0221
		Average	0.0134	0.0490	0.0179	0.0376	0.0129
		Best	0.0008	0.0019	0.0013	0.0010	0.0006
	1000	Worst	0.0190	0.0341	0.0246	0.0202	0.0177
		Average	0.0059	0.0178	0.0101	0.0096	0.0044

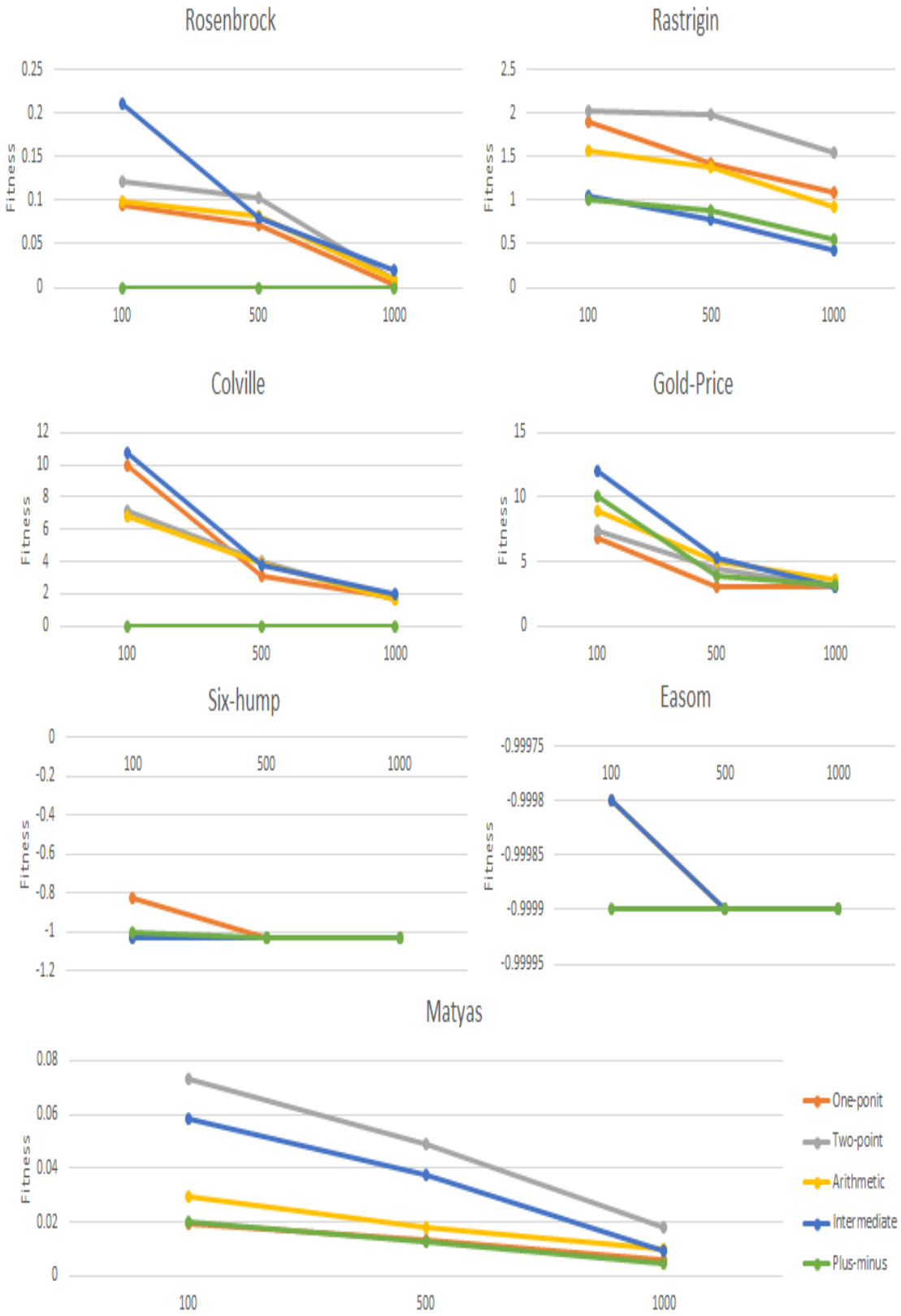

Figure 2: Graphical representation among crossover operators

For the first benchmark function (The Rosenbrock), the proposed operator gives optimum result for all generations in all 20 runs. For second function (The Rastrigin), the intermediate operator performs slightly better than proposed scheme but for all other operators, it is much better for both best and average results. Novel scheme is performing again with optimum result for all generations in all 20 runs, but other crossover operators not only a single time for our third function (The Colville). For our fourth benchmark function (The Six-hump), all the operators are working well and in most runs they gave an exact values. Our fifth function (The Gold-Price) is more sensitive for less generations but the performance is improved when the generations will enhanced with all operators. For the sixth benchmark function (The Easom), all operators are working on similar pattern and gave almost same and exact value most of the runs. For the last benchmark function (The Matyas), the proposed scheme again perform better than all existing operators. Our discussion is ended that the GA is sensitive for number of generations and not for number of runs.

For fast comparisons, we also display a graphical behaviors of the empirical results of average values among all five crossover operators for all benchmark problems in Figure 2. We see that the novel approach is how much fast to converge among all the schemes.

6 Conclusion

Various crossover operators have been presented for GAs that are used with different applications. In this article, we also introduced a novel and efficient crossover scheme for GAs. The proposed approach provides the better convergence rate compared to other crossover methods. The performance of each crossover method in terms of best, average and worst individual fitness is weighed through the implementation of a program in MATLAB. A set of experiments on a selected set of multi-modal testing functions of varying difficulty was described. Experimental results and comparative study provided the evidence to improved performance of the proposed technique than other crossover methods of GAs as a whole. The novel scheme was successfully applied to the optimization problems of a set of well-known benchmark functions, which encourages further improvements of this idea. After the results of benchmark functions in our study, we suggest that proposed crossover may be a good candidate for a crossover operator to get fast and accurate results in which researchers can have more confidence to apply it.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

Acknowledgments

The authors are thankful to the associate editor for some useful comments that led to an improved version of the article.

References

[1] J. H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence,University of Michigan Press,Oxford, UK, (1975).

[2] D. E. Golberg, Genetic algorithms in search, optimization and machine learning, Addison-Wesley Publishing Company, (1989).

[3] L. J. Eshelman, R. A. Caruana, and J. D. Schaffer, Biases in the crossover landscape. In Proceedings of the third international conference on Genetic algorithms, Morgan Kaufmann Publishers Inc. 10-19 (1989).

[4] W. M. Spears and V. Anand, A study of crossover operators in genetic programming. In International Symposium on Methodologies for Intelligent Systems, Springer,409-418 (1991).

[5] S. Picek and M. Golub, Comparison of a crossover operator in binary-coded genetic algorithms. WSEAS transactions on computers, 9, 1064-1073 (2010).

[6] H. Adeli and N. T. Cheng, Integrated genetic algorithm for optimization of space structures. Journal of Aerospace Engineering, 6(4), 315-328 (1993).

[7] H. Adeli and N. T. Cheng, Augmented lagrangian genetic algorithm for structural optimization. Journal of Aerospace Engineering, 7(1), 104-118 (1994).

[8] H. Adeli and N. T. Cheng, Concurrent genetic algorithms for optimization of large structures. Journal of Aerospace Engineering, 7(3), 276-296 (1994).

[9] S. J. Wu and P. T. Chow, Steady-state genetic algorithms for discrete optimization of trusses. Computers and Structures, 56(6), 979-991 (1995).

[10] W. M. Jenkins, On the application of natural algorithms to structural design optimization. Engineering structures, 19(4), 302-308 (1997).

[11] K. A. DeJong and W. M. Spears, An analysis of the interacting roles of population size and crossover in genetic algorithms. In International Conference on Parallel Problem Solving from Nature, Springer, 38-47 (1990).

[12] G. Syswerda, Uniform crossover in genetic algorithms. In Proc. Third International Conference of Genetic Algorithms, Morgan Kaufmann, 2-9 (1989).

[13] O. Hasancebi and F. Erbatur, Evaluation of crossover techniques in genetic algorithm based optimum structural design. Computers and Structures, 78(1), 435-448 (2000).

[14] D. Zaharie, Influence of crossover on the behavior of differential evolution algorithms. Applied Soft Computing, 9(3), 1126-1138 (2009).

[15] C. R. Reeves and J. E. Rowe, Genetic Algorithms Principles and Perspectives,. Kluwer Academic Publishers, (2003).

[16] A. J. Umbarkar and P. D. Sheth, Crossover operators in genetic algorithms: a review. ICTACT Journal on Soft Computing, 6(1), 1083-1092 (2015).

[17] M. Gen and R. Cheng. Genetic algorithm and engineering optimization. John Wily and Sons, New York, (2000).

[18] L. Davis, Handbook of genetic algorithms. Van Nostrand Reinhold, New York, (1991).

[19] H. P. Schwefel, Numerical optimization of computer models. John Wiley and Sons, Inc., (1981).

[20] Z. Michalewicz, Genetic Algorithms+ Data Structures= Evolution Programs. Springer, 3rd edition, (1996).

[21] K. A. DeJong, An analysis of the behavior of a class of genetic adaptive systems. Ph. D. Thesis, University of Michigan, (1975).

[22] A. A. Goldstein and J. F. Price, On descent from local minima. Mathematics of Computation, 25(115):569-574, (1971).

[23] E. E. Easom, A survey of global optimization techniques. PhD thesis, University of Louisville, (1990).

[24] X. S. Yang, Nature-inspired optimization algorithms. Elsevier, (2014).