Available online at www.sciencedirect.com

ScienceDirect

Applied Mathematics and Computation 188 (2007) 895–911

APPLIED MATHEMATICS AND COMPUTATION

www.elsevier.com/locate/amc

A new crossover operator for real coded genetic algorithms

Kusum Deep, Manoj Thakur [*]

Department of Mathematics, Indian Institute of Technology Roorkee, Roorkee 247 667, India

Abstract

In this paper, a new real coded crossover operator, called the Laplace Crossover (LX) is proposed. LX is used in conjunction with two well known mutation operators namely the Makinen, Periaux and Toivanen Mutation (MPTM) and Non-Uniform Mutation (NUM) to define two new generational genetic algorithms LX–MPTM and LX–NUM respectively. These two genetic algorithms are compared with two existing genetic algorithms (HX–MPTM and HX–NUM) which comprise of Heuristic Crossover operator and same two mutation operators. A set of 20 test problems available in the global optimization literature is used to test the performance of these four genetic algorithms. To judge the performance of the LX operator, two kinds of analysis is performed. Firstly a pair wise comparison is performed between LX–MPTM and HX–MPTM, and then between LX–NUM and HX–NUM. Secondly the overall comparison of performances of all the four genetic algorithms is carried out based on a performance index (PI). The comparative study shows that Laplace crossover (LX) performs quite well and one of the genetic algorithms defined (LX–MPTM) outperforms other genetic algorithms.
© 2006 Elsevier Inc. All rights reserved.

Keywords: Genetic algorithms; Global optimization; Real coded crossover operators

1. Introduction

A general nonlinear global optimization problem could be stated as

$$\text{Max/Min } f(x), \quad \text{where } f : S \subseteq R^n \to R, \ S \neq \phi.$$

The set S is called the feasible region and f is called the objective function.

Since a maximization problem can always be converted into a minimization problem, without loss of generality we can talk about minimization problems only.

A point $x^* \in S$ is called a local minima of f if $f(x^*) \leqslant f(x)\ \forall x \in N_\varepsilon(x^*) \cap S$ where $N_\varepsilon(x^*) = \{x \mid \|x - x^*\| < \varepsilon, \varepsilon > 0\}$ is a small neighborhood of the point x^*. If $f(x^*) \leqslant f(x)\ \forall x \in S$ then x^* is said to be the global minima of f. In the present work we are concerned with the problem having box-constraints only. i.e.

$$S = \{x \in R^n | x_i^{\min} \leqslant x_i \leqslant x_i^{\max},\ i = 1, 2, \ldots, n\}.$$

[*] Corresponding author.
E-mail address: manojdma@iitr.ernet.in (M. Thakur).

0096-3003/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2006.10.047

A variety of application problems in engineering, science and technology can be formulated as nonlinear global optimization problems having local as well as global optima. Mostly, the user is interested in determining the global minima. However it is difficult to determine the global minima rather than local minima. The optimization techniques available in literature for solving a general nonlinear optimization problem could be classified into two groups:

- Deterministic optimization techniques.
- Stochastic optimization techniques.

Deterministic optimization techniques work with a single solution (point by point algorithms) and use deterministic transition rules to guide the search towards the optimum solution. Deterministic optimization techniques are local minimizer in nature because they begin the search procedure with a guess solution (often chosen randomly in the search space) and if this guess solution is not chosen close enough to the global minimum solution they are likely to be trapped in the local minimum solution. Furthermore, another drawback of the deterministic optimization technique is that they are not generic in nature. Most of them are designed to solve a particular class of optimization problem. For instance, conjugate gradient method works well for quadratic optimization problems and has theoretical proof but is not appropriate for solving multimodal problems [1].

To prevail over these obstacles described above many stochastic optimization techniques like Evolutionary Algorithms, Simulated Annealing etc. have been developed which rely heavily on computational power. Among these, Evolutionary Algorithms are found to be very promising global optimizers. Evolutionary Algorithms mainly comprise of three population based heuristic methodologies: Genetic Algorithms, Evolutionary Programming, and Evolutionary Strategies [2]. Genetic Algorithms (GA) are perhaps the most popular Evolutionary Algorithms [3].

GAs are general purpose population based search techniques which mimic the principle of natural selection and natural genetics laid by Charles Darwin. The concept of GAs was given by Holland [4]. This was first used to solve optimization problem by De-Jong's [5]. A detailed implementation of GA could be found in Goldberg [6]. In GA, a population of potential solutions, termed as chromosomes/individuals, is evolved over successive generations using a set of genetic operators called selection, crossover and mutation. First of all, based on some criteria every chromosome is assigned a fitness value, and then the selection operator is applied to choose relatively 'fit' chromosomes to be part of reproduction process. In reproduction process new individuals are created through crossover and mutation operators. Crossover operator blends the genetic information between chromosomes to explore the search space, whereas mutation operator is used to maintain adequate diversity in the population of chromosomes to avoid premature convergence.

In initial implementation of GAs, the chromosomes were represented by strings of binary alphabets. Binary GAs are found to be robust search technique to avoid local minima but the computational cost is usually very high as compared to deterministic optimization technique. A major drawback of Binary GAs is that they face difficulties when applied to problems having large search space and seeking high precision. In Binary Coded GAs string length must be chosen a priori to get a desired degree of precision in the final solution. Larger string length results in more precise solution. Goldberg et al. [7] has shown that for large string lengths, larger population size is required which in turn increase the computational cost. Another drawback in binary coded GA is the "Hamming-Cliff" problem which arises when the hamming distance (in binary coding) between two adjacent integers (in decimal code) is very large. A large number of bits are to be altered to change an integer to the adjacent one which causes the reduction in the efficiency of Binary GA. The Gray-coded GAs overcome the hamming-cliff problem of Binary Coded GAs. However, in Jing and Yang [8] it is found that it still require a large amount of computations.

To overcome these difficulties related to binary encoding of continuous parameter optimization problems, real-encoding of chromosomes is used. GAs which make use of the real-encoding of chromosomes are termed as Real Coded GAs. This representation seems quite natural to deal with problems having continuous search space. Earlier work on Real Coded GAs can be found in [9–13].

In Real Coded GAs, crossover has always been considered to be the fundamental search operator. In the recent past, a lot of effort has been put into the development of sophisticated real coded crossover operators to improve the performances of Real Coded GAs for function optimization.

In this paper, we propose a new crossover operator based on Laplace distribution. We call it the Laplace crossover (LX). Two new generational GAs are designed using LX operator and their performance is compared with two existing GAs [14,15]. To have a fare comparison of our algorithms with the existing generational GAs, a set of 20 test problems of various level of difficulty is used. Two approaches are used to compare the algorithms. In the first approach, a pair wise comparison of the proposed GAs with the existing GAs is performed and in the second approach the overall comparison is performed.

The paper is organized as follows: in Section 2 literature review on real coded operators is given, in Section 3 the proposed Laplace crossover is defined. In Section 4 the other genetic operators used in this study are explained. In Section 5 two new GAs based on Laplace crossover are discussed. The name and formulations of test problems is given in Section 6. In Section 7, the experimental setup of all the GAs is explained. The discussions on the performance of various GAs is given in Section 8. Finally in Section 9, the conclusions are drawn.

2. Literature review on real coded crossover operators

In earlier implementations of Real Coded GAs, Wright [9] suggested Heuristic crossover which was applied to produce one offspring from a pair of parents with a bias towards the better one. Radcliffe [16] presented a Flat Crossover in which an offspring was generated randomly between the genes of the parents. Michalewicz [12] developed Arithimatical Crossover and its two variants: uniform arithmetical crossover and non-uniform arithmetical crossover. Eshelman and Schaffer [13] introduced the concept of interval schemata for real-coded GAs and suggested a blend crossover (BLX-α) operator for two parents. Muhlebein and Voosen [17] introduced Extended Line Crossover and Extended Intermediate Crossover which is a special case of Blend-α crossover for $\alpha = 0.25$. Voigt et al. [18] presented a fuzzy recombination operator.

Deb and Agrawal [19] developed the Simulated Binary Crossover (SBX), which simulates the working principle of the single-point crossover operator on binary strings in continuous domain. SBX works on a pair of parents and produces two offsprings.

Ono and Kobayashi [20] suggested a unimodal normally distributed crossover operator (UNDX), where three parent solutions are used to create two or more children solutions. UNDX was implemented in a steady state GA and was used to solve three difficult large dimensional problems. Ono et al. [21] improved the performance of the UNDX with the help of uniform crossover operator but only three problems were reported in the paper. Kita et al. [22] extended the work of Ono et al. and proposed a multi-parental UNDX operator with more than three parents. However it was seen that the performance of the original GA incorporating UNDX was more or less similar to the extended multi-parent UNDX operator. Also the number of problems used for analysis were too less (three) to conclude anything significant.

To overcome the difficulty of premature convergence, Herrera and Lozano [23] introduced real coded crossover based on fuzzy connectives. Some extensions of fuzzy connective based crossover operator are suggested in Herrera et al. [24]. The performance of real coded GAs based on fuzzy connectives is found to be satisfactory but again on a relatively small and special class of problems.

Tsutsui et al. [25] proposed a new crossover called Simplex Crossover and concluded that the simplex crossover works well on functions having multimodality and/or epistasis with a medium number of parents: three parents on a low (10) dimensional function and four parents on a high (20) dimensional function.

Deb et al. [26] proposed another real coded crossover called parent centric crossover operator (PCX). PCX was successfully implemented in a real coded GA and found lowest ever value of some of the test problems available in the literature than any other optimization technique. But performance of PCX was reported only on a small set of three problems. Recently, a modified version of PCX operator is proposed in Sinha et al. [27]. The performance of PCX was significantly improved but the modified version of PCX depends heavily on a priori knowledge of the problem under consideration.

Recombination operators such as unimodal normal distribution crossover (UNDX), simplex crossover (SPX), and blend crossover (BLX) are mean-centric approaches, whereas the simulated binary crossover (SBX), fuzzy recombination and parent centric crossover (PCX) are parent-centric approaches.

3. The proposed Laplace crossover operator

In this section, we propose a new crossover which uses Laplace distribution. It is a parent centric operator. We have named this crossover operator as Laplace Crossover (LX) operator. The density function of Laplace distribution is given by

$$f(x) = \frac{1}{2b}\exp\left(-\frac{|x-a|}{b}\right), \quad -\infty < x < \infty \tag{1}$$

and distribution function of Laplace distribution is given by

$$F(x) = \begin{cases} \frac{1}{2}\exp\left(\frac{|x-a|}{b}\right), & x \leqslant a, \\ 1-\frac{1}{2}\exp\left(-\frac{|x-a|}{b}\right), & x > a, \end{cases} \tag{2}$$

where $a \in R$ is called the location parameter and $b > 0$ is termed as scale parameter. The density function of Laplace distribution when $a = 0$ for both $b = 0.5$ and $b = 1$ is simultaneously shown in Fig. 1. For $b = 0.5$, the probability of creating offsprings near the parents is higher and for $b = 1$, distant points are likely to be selected as offsprings.

Using LX, two offsprings $y^{(1)} = (y_1^{(1)}, y_2^{(1)}, \ldots, y_n^{(1)})$ and $y^{(2)} = (y_1^{(2)}, y_2^{(2)}, \ldots, y_n^{(2)})$ are generated from a pair of parents $x^{(1)} = (x_1^{(1)}, x_2^{(1)}, \ldots, x_n^{(1)})$ and $x^{(2)} = (x_1^{(2)}, x_2^{(2)}, \ldots, x_n^{(2)})$ in the following way.

First, a uniformly distributed random number $u \in [0, 1]$ is generated. Then, a random number β is generated which follows the Laplace distribution by simply inverting the distribution function of Laplace distribution as follows:

$$\beta = \begin{cases} a - b\log_e(u), & u \leqslant \frac{1}{2}, \\ a + b\log_e(u), & u > \frac{1}{2}. \end{cases} \tag{3}$$

The offsprings are given by the equations

$$y_i^{(1)} = x_i^{(1)} + \beta\left|x_i^{(1)} - x_i^{(2)}\right|, \tag{4}$$

$$y_i^{(2)} = x_i^{(2)} + \beta\left|x_i^{(1)} - x_i^{(2)}\right|. \tag{5}$$

From Eqs. (4) and (5) it is clear that both the offsprings are placed symmetrically with respect to the position of the parents. For smaller values of b, offsprings are likely to be produce near the parents and for larger values of b offsprings are expected to be produced far from the parents. For fixed values of a and b, LX dispenses offsprings proportional to the spread of parents i.e. if the parents are near to each other the offsprings are expected to be near to each other and if the parents are far from each other then the offsprings are likely

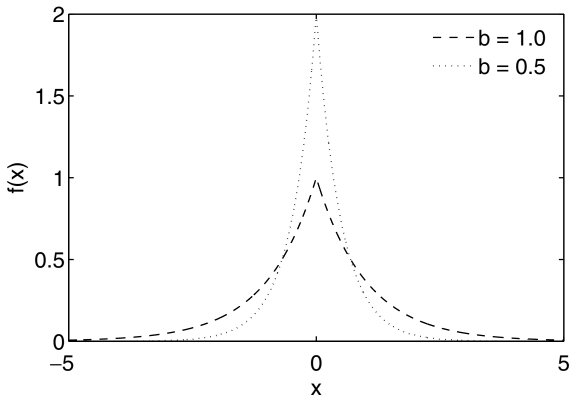

Fig. 1. Density function of Laplace distribution ($a = 0$, $b = 0.5$ and $b = 1$).

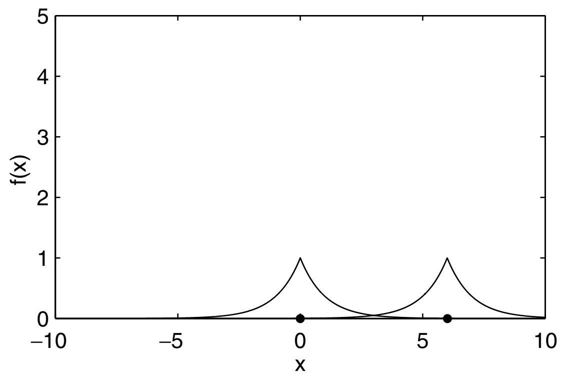

Fig. 2. Spread of the offsprings when the parents are far (for a fixed value of b and $a = 0$).

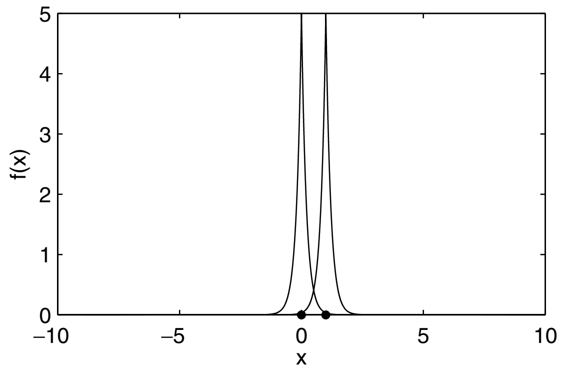

Fig. 3. Spread of the offsprings when the parents are near (for a fixed value of b and $a = 0$).

to be far from each other. A realization of the above idea can be had from Figs. 2 and 3 (the parents are placed at $(0,0)$, $(6,0)$ and $(0,0)$, $(1,0)$, respectively). This way the proposed crossover operator exhibits self adaptive behavior.

4. The other operators used herein

In this section, we define the other operators used in this paper i.e. Heuristic crossover, Makinen, Periaux and Toivanen Mutation and Non-Uniform Mutation.

4.1. Heuristic crossover

Heuristic crossover (HX) is introduced in Wright [9]. Later, Michalewicz et al. [28] use HX to solve some linearly constrained problems and incorporate HX in GENOCOP system. Subsequently, Michalewicz [29] apply HX to solve nonlinear constrained optimization problems. Some more experiments with HX are reported in Michalewicz et al. [30]. In recent studies [14,15], HX is applied on constrained as well as unconstrained optimization problems having various levels of difficulty.

From a pair of parents $x^{(1)} = (x_1^{(1)}, x_2^{(1)}, \ldots, x_n^{(1)})$ and $x^{(2)} = (x_1^{(2)}, x_2^{(2)}, \ldots, x_n^{(2)})$ an offspring $y = (y_1, y_2, \ldots, y_n)$ is generated in the following manner:

$$y_i = u(x_i^{(2)} - x_i^{(1)}) + x_i^{(2)}, \tag{6}$$

where u is a uniformly distributed random number in the interval $[0,1]$ and the parent $x^{(2)}$ has fitness value not worse than that of parent $x^{(1)}$. If the offspring so generated lie outside the feasible region, a new random number u is generated to produce another offspring using Eq. (5). If required, this process is repeated up to k times. There are some interesting features which make HX different from other crossover operators available in real coded GAs literature.

(i) HX produces at most one offspring from a given pair of parents.
(ii) Unlike other real coded crossover operators, HX makes use of fitness function values of parents in producing offspring.

4.2. Makinen, periaux and toivanen mutation (MPTM)

This mutation operator was proposed by Makinen et al. [31] in which they use it in a GA to solve some multidisciplinary shape optimization problems in aerodynamics and electromagnetics. In [14] it is used in a GA to solve a large set of constrained optimization problem. Although this operator is not given any specific name in literature, we have named it MPT mutation on the originators of this operator (Makinen, Periaux and Toivanen).

From a point $x = (x_1, x_2, \ldots, x_n)$ the mutated point $\hat{x} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n)$ is created as follows. Let r be a uniformly distributed random number such that $r \in [0,1]$. Then the muted solution is given by

$$\hat{x}_i = (1 - \hat{t})x_i^{\mathrm{l}} + \hat{t}x_i^{\mathrm{u}}, \tag{7}$$

where

$$\hat{t} = \begin{cases} t - t\left(\frac{t-r}{t}\right)^b & \text{if } r < t, \\ t & \text{if } r = t, \\ t + (1-t)\left(\frac{r-t}{1-t}\right)^b & \text{if } r > t, \end{cases} \tag{8}$$

and

$$t = \frac{x - x_i^{\mathrm{l}}}{x_i^{\mathrm{u}} - x}, \tag{9}$$

where x_i^{l} and x_i^{u} are lower and upper bounds of the ith decision variable, respectively.

Unlike Non-Uniform Mutation operator, the strength of MPT does not decrease as the generation increases.

4.3. Non-uniform mutation (NUM)

Michalewicz's Non-Uniform Mutation is one of the widely used mutation operators in real coded GAs [12,28]. From a point $x^t = (x_1^t, x_2^t, \ldots, x_n^t)$ the muted point $x^{t+1} = (x_1^{t+1}, x_2^{t+1}, \ldots, x_n^{t+1})$ is created as follows:

$$x_i^{t+1} = \begin{cases} x_i^t + \Delta(t, x_i^{\mathrm{u}} - x_i^t) & \text{if } r \leqslant 0.5, \\ x_i^t - \Delta(t, x_i^t - x_i^{\mathrm{l}}) & \text{otherwise}, \end{cases} \tag{10}$$

where t is the current generation number and r is a uniformly distributed random number between 0 and 1. x_i^{l} and x_i^{u} are lower and upper bounds of the ith component of the decision vector, respectively. The function $\Delta(t,y)$ given below takes value in the interval $[0,y]$

$$\Delta(t,y) = y\left(1 - u^{\left(1-\frac{t}{T}\right)}\right)^b, \tag{11}$$

where u is a uniformly distributed random number in the interval $[0,1]$, T is the maximum number of generations and b is a parameter, determining the strength of the mutation operator. In the initial generations Non-Uniform Mutation tends to search the space uniformly and in the later generations it tends to search the space locally i.e. closer to its descendants [12].

5. Proposed new GAs

The objective of this study is to introduce a new crossover operator LX and to evaluate the performance of LX in comparison to other crossover operators existing in literature. For this we have chosen two real coded GAs defined in [14,15]. These two genetic algorithms make use of Makinen, Periaux and Toivanen Mutation (MPTM) and Non-Uniform Mutation (NUM), respectively. The Heuristic Crossover (HX) is used as recombination operator in both the GAs we name these two GAs as HX–MPTM and HX–NUM, respectively.

Using the LX operator defined in Section 3; we propose two new GAs by combining LX with the MPTM and NUM defined in Section 4. Both the GAs are generational GAs. We name these GAs as LX–MPTM and LX–NUM, respectively.

In all four the genetic algorithms tournament selection is used as the selection operator. The structure of all the genetic algorithms defined above is very much similar to the Simple GA of Goldberg.

The pseudo code of LX–MPTM and LX–NUM is given below.

```
begin
  m = 0
  initialize P(m)
  evaluate fitness of P(m)
  While (termination criterion is not satisfied) do
    m = m + 1;
    select P(m) from P(m − 1)
    apply LX on P(m)
    apply mutation (MPTM/NUM) Operator
  end do
end begin
```

If LX produces an offspring which violates a box-constraint i.e. $x_i < x_i^{\min}$ or $x_i > x_i^{\max}$ for some i, then x_i is assigned a random value in the interval $[x_i^{\min}, x_i^{\max}]$.

Table 1 summarizes the operators used in the four GAs used in this study. The values of various parameters occurring in selection, crossover and mutation operators are given in Section 7.

6. Test problems

The performances of LX–MPTM and LX–NUM defined in Section 5 are compared against the two existing GAs HX–MPTM and HX–NUM. For this purpose we have collected a set of 20 well known benchmark test problems of different levels of complexity and multimodality from the global optimization literature. This set contains unimodal as well multimodal problems. All the problems are scalable i.e. the number of decision variables can be increased/decreased as per users choice. The complete formulation of the test problems is given below

1. *Ackley's problem*

$$\min_x f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e,$$

$$-30 \leqslant x_i \leqslant 30,\ \ x^* = (0,0,\ldots,0)\ \text{and}\ f(x^*) = 0.$$

Table 1
Summary of operators used in the four GAs

Operators used	HX–MPTM	HX–NUM	LX–MPTM	LX–NUM
Selection	Tournament	Tournament	Tournament	Tournament
Crossover	HX	HX	LX	LX
Mutation	MPTM	NUM	MPTM	NUM

2. *Cosine mixture problem*

$$\min_x f(x) = \sum_{i=1}^{n} x_i^2 - 0.1 \sum_{i=1}^{n} \cos(5\pi x_i), \quad -1 \leqslant x_i \leqslant 1,\ x^* = (0,0,\ldots,0) \text{ and } f(x^*) = 0.1n.$$

3. *Exponential problem*

$$\min_x f(x) = \exp\left(-0.5\sum_{i=1}^{n} x_i^2\right), \quad -1 \leqslant x_i \leqslant 1,\ x^* = (0,0,\ldots,0) \text{ and } f(x^*) = -1.$$

4. *Griewank problem*

$$\min_x f(x) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right), \quad -600 \leqslant x_i \leqslant 600,\ x^* = (0,0,\ldots,0) \text{ and } f(x^*) = 0.$$

5. *Levy and Montalvo problem 1*

$$\min_x f(x) = \frac{\pi}{n}\left(10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right](y_n - 1)^2\right), \quad \text{where}$$

$$y_i = 1 + \frac{1}{4}(x_i + 1),\ -10 \leqslant x_i \leqslant 10,\ x^* = (0,0,\ldots,0) \text{ and } f(x^*) = 0.$$

6. *Levy and Montalvo problem 2*

$$\min_x f(x) = 0.1\left(\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2\left[1 + \sin^2(3\pi x_{i+1})\right] + (x_n - 1)^2\left[1 + \sin^2(2\pi x_n)\right]\right),$$
$$-5 \leqslant x_i \leqslant 5,\ x^* = (0,0,\ldots,0) \text{ and } f(x^*) = 0.$$

7. *Paviani problem*

$$\min_x f(x) = \sum_{i=1}^{10}\left[(\ln(x_i - 2))^2 + (\ln(10 - x_i))^2\right] - \left(\prod_{i=1}^{10} x_i\right)^{0.2}, \quad 2 \leqslant x_i \leqslant 10,\ f(x^*) \approx -997807.705158.$$

8. *Rastrigin problem*

$$\min_x f(x) = 10n + \sum_{i=1}^{n}\left[x_i^2 - 10\cos(2\pi x_i)\right], \quad -5.12 \leqslant x_i \leqslant 5.12,\ x^* = (0,0,\ldots,0) \text{ and } f(x^*) = 0.$$

9. *Rosenbrock problem*

$$\min_x f(x) = \sum_{i=1}^{n-1}\left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right], \quad -30 \leqslant x_i \leqslant 30,\ x^* = (0,0,\ldots,0) \text{ and } f(x^*) = 0.$$

10. *Schwefel problem*

$$\min_x f(x) = -\sum_{i=1}^{n} x_i \sin\left(\sqrt{|x_i|}\right), \quad -500 \leqslant x_i \leqslant 500,\ x^* = (420.97, 420.97, \ldots, 420.97) \text{ and}$$
$$f(x^*) = -418.9829 * n.$$

11. *Sinusoidal problem*

$$\min_x f(x) = -\left[2.5\prod_{i=1}^{n} \sin(x_i - z) + \prod_{i=1}^{n} \sin(5(x_i - 30))\right],$$
$$0 \leqslant x_i \leqslant 180,\ x^* = (120, 120, \ldots, 120) \text{ and } f(x^*) = -3.5.$$

12. *Zakharov's function*

$$\min_x f(x) = \sum_{i=1}^n x_i^2 + \left(\sum_{i=1}^n \frac{i}{2}x_i\right)^2 + \left(\sum_{i=1}^n \frac{i}{2}x_i\right)^4, \quad -5.12 \leqslant x_i \leqslant 5.12,\ x^* = (0,0,\ldots,0) \text{ and } f(x^*) = 0.$$

13. *Sphere function*

$$\min_x f(x) = \sum_{i=1}^n x_i^2, \quad -5.12 \leqslant x_i \leqslant 5.12,\ x^* = (0,0,\ldots,0) \text{ and } f(x^*) = 0.$$

14. *Axis parallel hyper ellipsoid*

$$\min_x f(x) = \sum_{i=1}^n ix_i^2, \quad -5.12 \leqslant x_i \leqslant 5.12,\ x^* = (0,0,\ldots,0) \text{ and } f(x^*) = 0.$$

15. *Schewefel problem 3*

$$\min_x f(x) = \sum_{i=1}^n |x_i| + \prod_{i=1}^n x_i, \quad -10 \leqslant x_i \leqslant 10,\ x^* = (0,0,\ldots,0) \text{ and } f(x^*) = 0.$$

16. *Schewefel problem 4*

$$\min_x f(x) = \max_i \{|x_i|, 1 \leqslant i \leqslant n\}, \quad -100 \leqslant x_i \leqslant 100,\ x^* = (0,0,\ldots,0) \text{ and } f(x^*) = 0.$$

17. *De-Jong's function with noise*

$$\min_x f(x) = \sum_{i=1}^n (x_i^4 + \text{rand}(0,1)), \quad -10 \leqslant x_i \leqslant 10,\ x^* = (0,0,\ldots,0) \text{ and } f(x^*) = 0.$$

18. *Ellipsoidal function*

$$\min_x f(x) = \sum_{i=1}^n (x_i - i)^2, \quad -n \leqslant x_i \leqslant n,\ x^* = (1,2,\ldots,n) \text{ and } f(x^*) = 0.$$

19. *Generalized penalized function 1*

$$\min_x f(x) = \frac{\pi}{n}\left(10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\right)$$
$$+ \sum_{i=1}^n u(x_i, 10, 100, 4), \quad \text{where } y_i = 1 + \frac{1}{4}(x_i + 1),\ -10 \leqslant x_i \leqslant 10,\ x^* = (0,0,\ldots,0) \text{ and } f(x^*) = 0.$$

20. *Generalized penalized function 2*

$$\min_x f(x) = 0.1\left(\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + \sin^2(3\pi x_{i+1})] + (x_n - 1)^2[1 + \sin^2(2\pi x_n)]\right)$$
$$+ \sum_{i=1}^n u(x_i, 10, 100, 4), \quad -5 \leqslant x_i \leqslant 5,\ x^* = (0,0,\ldots,0) \text{ and } f(x^*) = 0.$$

In problem number 19 and 20, the value of penalty function u is given by the following expression:

$$u(x,a,k,m) = \begin{cases} k * pow((x-a),m) & \text{if } x > a, \\ -k * pow((x-a),m) & \text{if } x < -a, \\ 0 & \text{otherwise.} \end{cases}$$

Table 2
Parameter values for all the four GAs

Name of GA	Crossover probability	Mutation probability	Crossover index	Mutation index	Tournament size
HX–MPTM	0.70	0.020	–	4	3
LX–MPTM	0.50	0.005	0.20	4	2
HX–NUM	0.70	0.010	–	4	3
LX–NUM	0.50	0.005	0.15	4	2

7. Experimental setup

Finding the most appropriate combination of parameters occurring in a GA is termed as parameter tuning and is considered to be the most important and perhaps most difficult task. In case of real coded GA parameter tuning is generally more difficult as compared to binary coded GA due to the simple reason that the numbers of tunable parameters occurring in a real coded GA are usually more than that occurring in binary GA. This results in more possible combinations of parameter settings in real coded GAs than binary GAs. This difficulty also increases in both the real and binary GA as we take larger and larger test suit into consideration because due to multimodality and nonlinearity of different kind of objective functions, it becomes very challenging to suggest common fixed values of various parameters for the entire test suit.

To achieve this goal we have carried out an extensive experiment for all the four GAs. An empirical study of various possible combinations of crossover probability, mutation probability and the indexes occurring in the expressions of crossovers and mutations is done. The final parameter values for all the GAs are given in Table 2. We do not claim that these parameter settings are the best for any problem in general, but these values are selected and recommended since they are found to be repeatedly giving good results for most of the problems and hence they are appropriate values to choose if we talk about the overall performance of the algorithms in general.

For a uniform testing environment of all the algorithms, the population size is taken to be ten times the number of decision variables. All the algorithms use tournament selection (with size 3 for HX–MPTM and HX–NUM and 2 for LX–MPTM and LX–NUM). In addition all the GAs are elite preserving GAs with elitism size one i.e. if the best individual of the parent population is better that the best individual in offspring population then it replaces the best individual of offspring population. The value of a for LX is fixed to be zero and for HX, k is fixed to be 4 (as in [14]). If HX fails to produce a feasible offspring, like LX operator, a random point in feasible region is chosen. The number of variables in the entire problem set is fixed to be 30. For each GA 30 independent runs with different initial populations are taken and the objective function values, average number of function evaluations and execution time of successful runs are recorded. For a particular problem and a particular algorithm, a run is said to be a successful run if the best objective function value found in that run lies within 1% accuracy of the best know objective function value of that problem. The maximum numbers of generations are also fixed to be 5000 for all the GAs. All the algorithms are implemented in C++ and the experiments are done on a PIV 2.8 GHz machine with 512 MB RAM under WINXP platform.

8. Results and discussion

In this section the results of GA are compared and discussed. We have analyzed the performance in two ways as explained below.

8.1. First method of analysis

Since our main contribution in the present study is the introduction of LX operator; therefore we are interested in analyzing the effect of our new crossover operator on the performance of GAs. So first of all, we study and compare the four GAs in two pair viz. HX–MPTM vs. LX–MPTM and then HX–NUM vs. LX–NUM. This type of comparison looks quite logical in view of the fact that the GAs in these two pairs are structurally similar. In both the pairs only the crossover operators are different and the remaining operators are same. So comparison of HX–MPTM vs. LX–MPTM and HX–NUM vs. LX–NUM will reflect the effect of crossover operator on the performance of GAs.

8.1.1. *HX–MPTM vs. LX–MPTM*

The number of successful runs (out of 30 runs), average number of function evaluations and average execution time of successful runs are listed in Table 3. From this table it is observed that except the Rosenbrock's function (problem no. 9); both HX–MPTM and LX–MPTM solve all problems in all the 30 runs. In 15 test cases the average functions evaluations and average execution time of successful runs required by LX–MPTM are less than that of HX–MPTM whereas in four cases LX–MPTM takes more function evaluations and execution time than HX–MPTM.

In order to compare the quality of the solutions found, we list the mean and standard deviation of the best objective function value found after a fixed number of generations for each test case (see Table 4). In 16 problems the mean objective function value and standard deviation found by LX–MPTM is less than that of HX–MPTM and in four instances the mean objective function value and standard deviation found by HX–MPTM is less than that of LX–MPTM.

Further to ascertain the significance of the difference of mean of objective function values, we apply *t*-test at a 0.05 level of significance as proposed in GAs literature [32]. A '$\sim$' sign indicates that there is no significant difference in the means and a '+' ('$-$') sign indicates that the mean objective function value obtained by HX–MPTM (LX–MPTM) is significantly large than the mean objective function value obtained LX–MPTM (HX–MPTM). This is shown in the last column of Table 4.

In three cases there is no significant difference in the mean objective function values, in 15 cases the mean function value by HX–MPTM are significantly greater than LX–MPTM and there are only two cases where the mean function value by LX–MPTM are significantly greater than HX–MPTM.

In view of the above discussions we conclude that LX–MPTM outperforms HX–MPTM both in terms of average number of function evaluations of successful runs as well as the average execution time required. Also, in most of the cases the mean objective function values obtained by LX–MPTM are better than HX–MPTM with less standard deviation. This means that LX–MPTM repeatedly converges to near global optimum in lesser number of generations and execution time.

Table 3
No. of successful runs, average no. of function evaluations and execution time of successful runs for HX–MPTM and LX–MPTM

Problem number	Number of successful runs (out of 30 runs)		Average number of function evaluations of successful runs		Average time of execution (in s)	
	HX–MPTM	LX–MPTM	HX–MPTM	LX–MPTM	HX–MPTM	LX–MPTM
1	30	30	196,401	116,731	3.564567	2.242733
2	30	30	63,691	46,601	1.076533	0.847400
3	30	30	44,271	31,221	0.496867	0.388533
4	30	30	253,931	143,371	4.485933	2.722933
5	30	30	51,741	36,361	1.230733	0.963033
6	30	30	68,001	51,091	1.600533	1.272900
7	30	30	75,751	153,161	1.451567	3.017200
8	30	30	245,281	350,541	4.219267	6.515100
9	0	0	–	–	–	–
10	30	30	86,421	236,411	1.339600	4.081267
11	30	30	79,071	57,381	2.091667	1.541167
12	30	30	94,701	65,711	2.243233	1.642167
13	30	30	87,031	59,861	0.944267	0.723967
14	30	30	117,911	78,571	1.277600	0.944267
15	30	30	158,411	105,721	2.098967	1.529667
16	30	30	679	720	0.007300	0.007800
17	30	30	811	771	0.016667	0.015100
18	30	30	143,231	101,331	1.530733	1.219767
19	30	30	117,031	90,121	3.073433	2.594800
20	30	30	157,521	119,521	4.150500	3.293233

Table 4
Mean and standard deviation of objective function values and *t*-test results for mean objective function values for HX–MPTM and LX–MPTM

Problem number	Mean		Standard deviation		*t*-Test results
	HX–MPTM	LX–MPTM	HX–MPTM	LX–MPTM	
1	3.17E−04	2.18E−07	1.46E−04	6.94E−08	+
2	−2.99E+00	−3.00E+00	4.47E−13	0.00E+00	+
3	0.99E+00	1.00E+00	5.45E−05	1.74E−06	+
4	1.52E−03	1.14E−03	1.18E−03	1.18E−03	~
5	5.34E−09	9.94E−14	3.42E−09	6.45E−14	+
6	6.58E−10	8.56E−16	5.24E−10	1.29E−15	+
7	−9.98E+05	−9.93E+05	4.81E+01	1.08E+03	–
8	9.22E−12	1.23E−12	1.53E−11	4.89E−12	+
9	1.61E+01	1.85E+01	1.81E+01	6.38E−01	~
10	−1.26E+04	−1.26E+04	2.77E−07	9.32E−05	~
11	−3.50E+00	−3.50E+00	6.31E−04	4.29E−06	+
12	7.86E−09	8.04E−15	1.29E+08	1.20E−14	+
13	5.48E−05	1.32E−07	3.32E−05	7.44E−08	+
14	8.83E−05	−3.94E−08	4.32E−05	2.31E−08	+
15	2.88E−03	4.28E−05	1.26E+03	1.40E−05	+
16	1.19E−22	5.35E−06	0.00E+00	1.55E−05	–
17	3.17E−04	1.63E−04	3.31E−04	1.30E−04	+
18	5.02E−04	1.11E−06	2.93E−04	7.05E−07	+
19	1.30E−12	1.75E−22	1.29E−12	0.00E+00	+
20	1.30E−13	2.55E−27	2.12E−13	0.00E+00	+

8.1.2. LX–NUM vs. HX–NUM

Table 5 consists of the number of successful runs and average number of function evaluations and execution time of successful runs required by HX–NUM and LX–NUM. HX–NUM is able to reach near the global minimum of Rosenbrock's in 12 runs while LX–NUM failed completely to solve Rastringin's and

Table 5
No. of successful runs, average no. of function evaluations and execution time of successful runs for HX–NUM and LX–NUM

Problem number	Number of successful runs (out of 30 runs)		Average number of function evaluations of successful runs		Average time of execution (in s)	
	HX–NUM	LX–NUM	HX–NUM	LX–NUM	HX–NUM	LX–NUM
1	30	30	232,901	130,041	4.133833	2.473433
2	27	27	171,823	54,723	2.932259	0.996556
3	30	30	60,651	35,141	0.681233	0.436967
4	30	30	252,571	159,971	4.456767	3.034900
5	24	30	67,676	46,761	1.612583	1.244800
6	30	30	92,131	68,721	2.172367	1.727100
7	30	30	31,461	122,071	0.608333	2.408833
8	28	0	349,962	–	6.030107	–
9	12	0	1,171,276	–	12.506333	–
10	30	29	66,201	245,918	1.023967	4.240276
11	29	30	95,939	61,951	2.541483	1.665600
12	28	30	118,683	78,681	2.818607	1.968733
13	30	30	107,581	67,661	1.164567	0.818767
14	30	30	140,451	89,391	1.521367	1.075000
15	30	30	148,251	118,541	1.965633	1.714067
16	30	30	719	690	0.007800	0.007300
17	30	30	811	911	0.016667	0.017700
18	24	30	143,776	104,281	1.539667	1.253667
19	30	29	230,711	111,322	6.134900	3.205828
20	30	30	267,741	140,751	7.128633	3.886967

Rosenbrock's problems (problem nos. 8 and 9). In two cases (problem nos. 5 and 18) LX–NUM gives better success rate than HX–NUM. In four cases (problem nos. 10, 11, 12 and 19) there is a small difference between the number of successful runs of HX–NUM and LX–NUM. In 18 problems both HX–NUM and LX–NUM have positive success rate. In 15 out of these 18 problems, LX–NUM requires lesser function evaluation and execution time than HX–NUM.

The overall success rate of HX–NUM is better than LX–NUM. Though the sum of average function evaluations required by LX–NUM is less than HX–NUM but it is important to notice that the sum of average number of function evaluations is quite higher due to positive success rate of HX–NUM over Rastringin's and Rosebnbrock's function (problem nos. 8 and 9).

In Table 6, the mean and standard deviation of the best objective function value found after a fixed number of generations for each test case is reported. In 15 cases the mean objective function value found by HX–NUM is less than that of LX–NUM whereas in five cases the mean objective function value found by LX–NUM is less than that of HX–NUM. In 16 cases the standard deviation of the objective function values for LX–NUM is less than HX–NUM. The *t*-test (at a 0.05 level of significance) results are summarized in last column of Table 6. In six cases there is no significant difference in the mean objective function values, in 11 cases the mean function values by HX–NUM are more than LX–NUM and in only three cases mean function values by LX–NUM are more than HX–NUM.

From above discussion we can conclude that the HX–NUM gives slightly more success than LX–NUM but in most of those problems where both of the algorithms give positive success, LX–NUM is more efficient than HX–NUM.

8.2. Second method of analysis

After analyzing the behavior of Laplace crossover our next goal is to compare the relative performance of all the four GAs simeltaneously. To accomplish this task we have used the performance index (PI) defined by Bharti [33]. This was used for comparing the relative performance of some versions of controlled random search technique (RST). Later, Mohan and Nguyen [34] used this PI for comparing some hybrid version of RST. The relative performance of an algorithm using this PI is calculated in the following manner:

Table 6
Mean and standard deviation of objective function values and *t*-test results for mean objective function values for HX–NUM and LX–NUM

Problem number	Mean		Standard deviation		*t*-Test results
	HX–NUM	LX–NUM	HX–NUM	LX–NUM	
1	1.28E−03	6.66E−07	1.39E−03	2.80E−07	+
2	−2.98E+00	−2.99E+00	5.11E−02	4.51E−02	~
3	−9.99E−01	−1.00E+00	3.86E−04	7.13E−06	+
4	3.68E−03	8.57E−04	3.73E−03	5.36E−04	+
5	3.11E−02	3.34E−09	8.67E−02	1.83E−08	+
6	1.43E−08	3.91E−09	7.38E−08	2.14E−08	~
7	−9.97E+05	−9.97E+05	1.86E+02	5.33E+02	+
8	3.11E−13	7.30E+00	5.01E−13	3.08E+00	−
9	1.25E+01	1.85E+01	4.38E+00	1.80E−01	−
10	−1.26E+04	−1.25E+04	7.49E−09	9.41E+01	−
11	−3.41E+00	−3.50E+00	4.56E−01	2.54E−05	~
12	2.86E−01	3.32E−10	1.55E+00	1.82E−09	~
13	3.64E−04	8.92E−07	2.69E−04	5.48E−07	+
14	4.67E−04	3.43E−07	3.28E−04	1.34E−07	+
15	1.73E−03	1.32E−04	1.24E−03	4.09E−05	+
16	2.88E−04	1.23E−05	1.89E−04	3.51E−05	+
17	8.20E−05	1.38E−04	2.91E−04	1.20E−04	~
18	3.20E+00	1.98E−06	6.25E+00	1.02E−06	+
19	7.09E−11	3.46E−03	1.57E−10	1.89E−02	~
20	5.66E−14	5.75E−26	9.31E−14	0.00E+00	+

$$\mathrm{PI} = \frac{1}{N_\mathrm{p}} \sum_{i=1}^{N_\mathrm{p}} (k_1\alpha_1^i + k_2\alpha_2^i + k_3\alpha_3^i),$$

where $\alpha_1^i = \frac{\mathrm{Sr}^i}{\mathrm{Tr}^i}$,

$$\alpha_2^i = \begin{cases} \frac{\mathrm{Mt}^i}{\mathrm{At}^i} & \text{if } \mathrm{Sr}^i > 0, \\ 0 & \text{if } \mathrm{Sr}^i = 0, \end{cases} \quad \text{and}$$

$$\alpha_3^i = \begin{cases} \frac{\mathrm{Mf}^i}{\mathrm{Af}^i} & \text{if } \mathrm{Sr}^i > 0, \\ 0 & \text{if } \mathrm{Sr}^i = 0. \end{cases}$$

$i = 1, 2, \ldots, N_\mathrm{p}$

Sr^i	number of successful runs of ith problem
Tr^i	total number of runs of ith problem
At^i	average time used by an algorithm in obtaining the solution of ith problem
Mt^i	minimum of average time used by all the algorithms in obtaining the solution of ith problem
Af^i	average number of function evaluations of successful runs used by an algorithm in obtaining the solution of ith problem
Mf^i	minimum of average number of function evaluations of successful runs used all algorithms in obtaining the solution of ith problem
N_p	total number of problems analyzed

k_1, k_2 and k_3 ($k_1 + k_2 + k_3 = 1$ and $0 \leqslant k_1, k_2, k_3 \leqslant 1$) are the weights assigned to the percentage of success, the execution time and the average number of function evaluations of successful runs, respectively.

From the above definition it is clear that PI is a function of k_1, k_2 and k_3. Since $k_1 + k_2 + k_3 = 1$, one of k_i, $i = 1, 2, 3$ could be eliminated to reduce the number of dependent variables from the expression of PI. But it is still difficult to analyze the behavior of PI, because the surface plots of PI for all four GAs are overlapping and it is difficult to visualize them.

We adopt the same methodology as given in Mohan and Nguyen [34] i.e. equal weights are assigned to two terms at a time in the PI expression. This way PI becomes a function of one variable. The resultant cases are as follows:

(i) $k_1 = w,\ k_2 = k_3 = \frac{1-w}{2},\ 0 \leqslant w \leqslant 1,$
(ii) $k_2 = w,\ k_1 = k_3 = \frac{1-w}{2},\ 0 \leqslant w \leqslant 1,$
(iii) $k_3 = w,\ k_1 = k_2 = \frac{1-w}{2},\ 0 \leqslant w \leqslant 1.$

The graphs corresponding to each of the cases (i), (ii) and (iii) are shown in Figs. 4–6, respectively. In these figures the horizontal axis represents the weight w and the vertical axis represents the performance index PI.

In case (i), the execution time and average number of function evaluations of successful runs are given equal weights. PI's of all four algorithms are superimposed in the Fig. 4 for comparison and to get a ranking of the performance of the four algorithms. It is observed that for LX–MPTM the value of PI is more than all the remaining three GAs. For the remaining three algorithms we divide the entire range into three sub-intervals $0 \leqslant w < 0.157$, $0.157 \leqslant w < 0.701$ and $0.701 \leqslant w \leqslant 1$. In these intervals the PI rankings are in the order LX–NUM > HX–MPTM > HX–NUM, HX–MPTM > LX–NUM > HX–NUM and HX–MPTM > HX–NUM > LX–NUM, respectively.

In case (ii), equal weights are assigned to the percentage of success and average number of function evaluations of successful. From Fig. 5, it is clear the in the entire interval ($0 \leqslant w \leqslant 1$) the ranking of PI follows the following order LX–MPTM > HX–MPTM > LX–NUM > HX–NUM.

In case (iii), the PI of LX–MPTM is more than all other GAs (refer Fig. 6). For $0 \leqslant w < 0.526$ and $0.526 \leqslant w < 1$ the ranking of rest of the GAs are HX–MPTM > LX–NUM > HX–NUM and LX–NUM > HX–MPTM > HX–NUM, respectively.

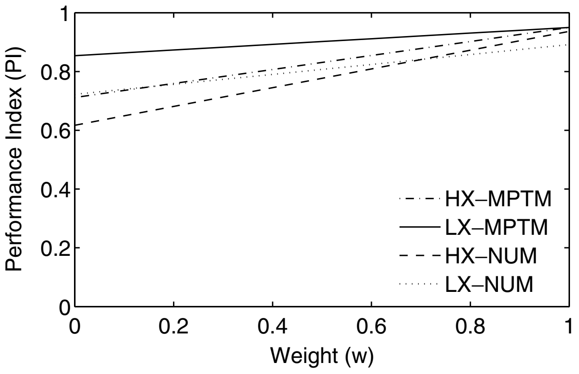

Fig. 4. Performance index of all GAs when $k_1 = w$ and $k_2 = k_3 = (\frac{1-w}{2})$.

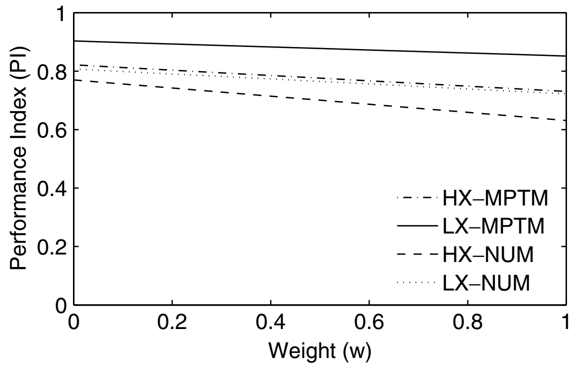

Fig. 5. Performance Index of all GAs when $k_2 = w$ and $k_1 = k_3 = (\frac{1-w}{2})$.

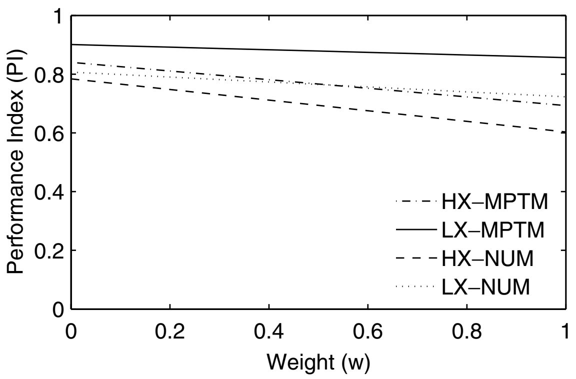

Fig. 6. Performance Index of all GAs when $k_3 = w$ and $k_1 = k_2 = (\frac{1-w}{2})$.

From the above discussion we can conclude that LX–MPTM outperforms all other remaining GAs however for the other GAs the value of the PI depends upon the weights assigned to the terms appearing in the expression of PI.

9. Conclusions

In this paper a new real coded crossover operator, called the Laplace Crossover operator (LX), is introduced. LX is a parent centric crossover operator and found to exhibit self adaptive behavior. By adjoining LX with two well known mutation operators, namely MPTM and NUM, two new GAs called LX–MPTM and LX–NUM, are proposed. For an analogical comparison they are compared with two existing GAs based on Heuristic Crossover operator (HX–MPTM and HX–NUM). This is meaningful, so that the strength and weakness of LX can be measured over Heuristic Crossover HX.

The experiments are performed on a set of 20 benchmark problems available in global optimization literature. These problems are of particular interest since their dimension can be controlled (increased /decreased) by the user.

Two methodologies are used to assess the performance the two new GAs. Firstly, the two new GAs namely LX–MPTM and LX–NUM are compared with HX–MPTM and HX–NUM, respectively. In this method the comparison is made with respect to percentage of success, the average number of function evaluations and execution time of successful runs. Also, to compare the quality of solution obtained by the algorithms, *t*-test is performed on the mean of the objective function value. It is concluded that LX–MPTM, which uses the newly defined crossover operator LX, outperforms HX–MPTM. The same type of analysis is performed for the second pair of algorithms namely LX–NUM and HX–NUM. It is observed that the overall success rate of HX–NUM is slightly better than LX–NUM but in most of those cases where both of them give positive success, the average number of function evaluations and average execution time required by LX–NUM, is less than that of HX–NUM.

The second methodology is to compare all the four algorithms simultaneously. For this, a performance index given in literature is used. Based on this performance index it is concluded that among all the four algorithms, the LX–MPTM has a definite edge over the other remaining three algorithms considered in this paper.

The objective of the present study is to make a comparison of LX with HX. Certainly LX could be compared with any other crossover operator existing in literature. The comparison of LX with SBX is being reported in another paper. Based on this study, it is expected that in future, the Laplace Crossover operator shows a great potential for further research.

Acknowledgements

The authors are thankful to the anonymous referee for their valuable comments and suggestions. The second author also acknowledges Council for Scientific and Industrial Research (CSIR), New Delhi, India, for providing the financial support vide grant number 9/143 (439) 2003-EMR-I.

References

[1] K. Deb, Muiti-objective Optimization using Evolutionary Algorithms, Wiley, Chichester, 2001.
[2] P.J. Bentley, Evolutionary Design by Computers, Morgan Kaufmann, San-Francisco, 1999.
[3] T. Back, H.-P. Schwefel, An Overview of Evolutionary Algorithms for Parameter Optimization, Evolutionary Computation 1 (1) (1993) 1–23.
[4] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan press, Ann Arbor, 1975.
[5] K.A. De-Jong, An analysis of the behavior of a class of genetic adaptive systems, Doctoral Dissertation, University of Michigan, 1975.
[6] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, New York, 1989.
[7] D.E. Goldberg, Real-coded genetic algorithms, virtual alphabets, and blocking, Complex Systems 5 (2) (1991) 139–168.
[8] J.X. Jing, J.D. Yang, An improved simple genetic algorithm—accelerating genetic algorithm, Systems Practice 21 (4) (2001) 8–13.
[9] A.H. Wright, Genetic algorithms for real parameter optimization, in: G.J.E. Rawlins (Ed.), Foundations of Genetic Algorithms I, Morgan Kaufmann, San Mateo, 1991, pp. 205–218.
[10] L. Davis, Handbook of Genetic Algorithms, Van Nostrand Reinhold, 1989.
[11] C.Z. Janikow, Z. Michalewicz, An experimental comparison of binary and floating point representation in genetic algorithms, in: Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, San Francisco, 1991, pp. 31–36.
[12] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, New York, 1992.
[13] L.J. Eshelman, J.D. Schaffer, Real-coded genetic algorithms and interval schemata, in: D.L. Whitley (Ed.), Foundation of Genetic Algorithms II, Morgan Kaufmann, San Mateo, CA, 1993, pp. 187–202.

[14] K. Meittinen, M.M. Makela, J. Toivanen, Numerical comparision of some penalty based constraint handling techniques in genetic algorithms, Journal of Global Optimization 27 (2003) 427–446.
[15] H. Maaranen, K. Meittinen, M.M. Makela, Quasi random initial population for genetic algorithms, Computer and Mathematics with Applications 47 (2004) 1885–1895.
[16] N.J. Radcliffe, Equivalence class analysis of genetic algorithms, Complex Systems 2 (5) (1991) 183–205.
[17] H. Muhlebein, D. Schlierkamp-Voosen, Predictive models for breeder genetic algorithms in continuous parameter optimization, Evolutionary Computation 1 (1) (1993) 25–49.
[18] H.M. Voigt, H. Muhlenbein, D. Cvetkovic, Fuzzy recombination for the breeder genetic algorithms, in: L.J. Eshelmen (Ed.), Proceedings of the 6th International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 1995, pp. 104–111.
[19] K. Deb, R.B. Agrawal, Simulated binary crossover for continuous search space, Complex Systems 9 (1995) 115–148.
[20] I. Ono, S. Kobayashi, A real-coded genetic algorithm for function optimization using unimodal normal distribution crossover, in: T. Back (Ed.), Proceedings of the Seventh International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 1997, pp. 246–253.
[21] I. Ono, H. Kita, S. Kobayashi, A robust real-coded genetic algorithm using unimodal normal distribution crossover augmented by uniform crossover: effects of self-adaptation of crossover probabilities, in: W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honavar, M. Jakiela, R. Smith (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1 1999), Morgan Kaufmann, San Mateo, CA, 1999, pp. 496–503.
[22] H. Kita, I. Ono, S. Kobayashi, The multi-parent unimodal normal distribution crossover for real-coded genetic algorithms, Proceedings of the 1999 Congress on Evolutionary Computation, vol. 2, IEEE, 1999, pp. 1588–1595.
[23] F. Herrera, M Lozano, Adaptation of genetic algorithm parameters based on fuzzy logic controllers, in: Genetic Algorithms and Soft Computing, 1996.
[24] F. Herrera, M. Lozano, J.L. Verdegay, Dynamic and heuristic fuzzy connectives based crossover operators for controlling the diversity and convergence of real-coded genetic algorithms, International Journal of Intelligent System 2 (1999) 1013–1041.
[25] S. Tsutsui, M. Yamamura, T. Higuchi, Multi-parent recombination with simplex crossover in real-coded genetic algorithms. in: W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honavar, M. Jakiela, R. Smith (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1 1999), 1999, pp. 657–664.
[26] K. Deb, A. Anand, D. Joshi, A computationally efficient evolutionary algorithm for real-parameter evolution, Evolutionary Computation Journal 10 (4) (2002) 371–395.
[27] A. Sinha, S. Tiwari, K. Deb, A population based steady state procedure for real parameter optimization, KANGAL Technical Report no. 2005004, 2005.
[28] Z. Michalewicz, T. Logan, S. Swaminathan, Evolutionary operators for continuous convex parameter space, in: A.V. Sebald, L.J. Fogel (Eds.), Proceeding of 3rd Annual Conference on Evolutionary Programming, World Scientific, River Edge, NJ, 1994, pp. 84–97.
[29] Z. Michalewicz, Genetic algorithms, numerical optimization and constraints, in: L.J. Eshelmen (Ed.), Proceedings of the 6th International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 1995, pp. 151–158.
[30] Z. Michalewicz, D. Dasgupta, R.G. Le Riche, M. Schoenauer, Evolutionary algorithms for constrained engineering problems, Computers & Industrial Engineering Journal 30 (4) (1996) 851–870.
[31] R.A.E. Makinen, J. Periaux, J. Toivanen, Multidisciplinary shape optimization in aerodynamics and electromagnetic using genetic algorithms, International Journal for Numerical Methods in Fluids 30 (2) (1999) 149–159.
[32] F. Herrera, M. Lozano, Two-loop real-coded genetic algorithms with adaptive control of mutation step size, Applied Intelligence 13 (2000) 187–2004.
[33] Bharti, Controlled random search technique and their applications, Ph.D. Thesis. Department of Mathematics, University of Roorkee, Roorkee, India, 1994.
[34] C. Mohan, H.T. Nguyen, A controlled random search technique incorporating the simulating annealing concept for solving integer and mixed integer global optimization problems, Computational Optimization and Applications 14 (1999) 103–132.