On Genetic Operators for Unconstrained Optimization Problems

A. K. Bhunia[a] *, P. Pal[b] *, S. Chattopadhyay[c], B. K. Medya[d]

[a]*Department of Mathematics, Burdwan University, Burdwan-713104 , India*

[b]*Department of Computer Science & Engineering, Hooghly Engineering & Technology College , Hooghly-712103, West Bengal, India.*

[c]*Department of Information Technology, Jadavpur University, India*

[d]*Department of Information Technology, N.I.T., Kolkata, India*

Abstract

In this paper, a set of real coded genetic algorithms has been considered for solving optimization problems with continuous variables to study the performance of different evolutionary and genetic operators like selection, crossover and mutation operators. For these purposes, these algorithms have been tested with two benchmark functions and the computational results have been compared on generation size, objective function value, function evaluation and CPU time.

Keywords **:** Genetic Algorithm, Selection, Crossover, Mutation.

1. Introduction

Algorithms for solving optimization problems are of increasing importance in many practical situations with respect to global perspective. However, in case of modern engineering design and system operations, solving optimization problems is a formidable task in the context of global issue. Generally, these types of problems are non-convex (or non-concave), multimodal and high dimensional with several local optima. To overcome these difficulties, several stochastic search and optimization methods based on natural evolution and genetics have been suggested to obtain the global optimum. These are usually heuristic in nature. Among these, Genetic algorithm (GA) has become very popular. This algorithm (GA) was developed by Holland (1975). According to Goldberg (1989), Michalewicz (1996), Sakawa (2002), Mitchell (1996), Gen and Cheng (1997), GA is a computerized stochastic search and optimization algorithm based on the mechanics of natural evolution and genetics. Initially some research works have been started for the development of initialization process, chromosome representation and GA operators (like selection / reproduction, crossover and mutation operators). In the development of this subject different types of operators was developed by various researchers.

*** AMO – Advanced Modeling and Optimization. ISSN: 1841-4311**

Corresponding Author: p5_pal@yahoo.co.in (P. Pal)

In this context, one may refer to the works of Baker (1985), Deb (2000), Miettinen (2003), Chelouah and Siary (2000), Yiu et al. (2004), Jana and Biswal (2004), Deb et al. (2002). In this paper, we have developed a set of GAs with real coding for solving optimization problems with continuous variables combining different evolutionary and genetic operators to test the performance of those operators. Then these algorithms have been tested with two benchmarks functions and compared the computational results with respect to GA characteristics like average generation size, objective function value, function evaluation and CPU time.

2. Problem definition

Let us consider a bound constrained minimization problem

$$\textit{Minimize } f(x)$$
$$\textit{subject to } l \le x \le u$$

where $x = (x_1, x_2, ..., x_i, ...x_n)$ is a variable vector in R^n, $f(x)$ is the objective function and $l = (l_1, l_2, ..., l_i, ..., l_n)$ and $u = (u_1, u_2, ..., u_i, ..., u_n)$. We denote the domain of x_i by $[l_i, u_i]$ and the feasible solution space by $[l, u]$.

3. Genetic Algorithm

Genetic Algorithm (GA) is an exhaustive search algorithm based on the mechanics of natural evolution and genetics. It has been successfully applied to different types of optimization problems, for its several advantages over conventional optimization methods. Holland [cf. Holland (1975)] was inspired by the well known Darwin's theory about natural evolution and constructed an evolutionary algorithm based on the fundamental principle of the Darwin's theory: 'Survival of the fittest'. This algorithm is known as genetic algorithm. The theoretical basis for the GA is the Schema Theorem, which states that individual chromosomes with short, low-order, highly fit schemata or building blocks receive an exponentially increasing number of trials in successive generations. From the natural genetics, we know that chromosomes are the main carriers of hereditary information from parent to offspring and those genes, which represent hereditary factors, are lined up on chromosomes. At the time of reproduction, crossover and mutation take place among the parent chromosomes. In this way, hereditary factors of parents are mixed-up and carried to their offspring. Again according to Darwinian principle, only the fittest animals can survive in nature. So a pair of fittest parent normally reproduces a better offspring. The same phenomenon is followed to create a genetic algorithm for solving an optimization problem. In this algorithm, potential solutions of the problem are analogous with the chromosomes and chromosome of better offspring with the better solution of the problem. Different genetic operators viz., crossover and mutation happen among a set of potential solutions to get a new set of solutions and it continues until a termination criterion is satisfied.

A GA for a particular problem must have the following components.

- **(a)** GA parameters
- **(b)** Chromosome representation for potential solutions to the problem
- **(c)** Initialization of chromosomes
- **(d)** Evaluation of fitness function
- **(e)** Selection process.
- **(f)** Genetic operators – crossover and mutation.

There are different parameters used in the genetic algorithm, viz. population size (*p_size*), maximum number of generations (*m_gen*), crossover rate/probability of crossover (*p_cross*) and mutation rate/probability of mutation (*p_mute*). There is no hard and fast rule for choosing the population size for GA. However, if the population size is considered to be large, storing of data in the intermediate steps of GA may create some problems at the time of computation with the help of computer. On the other hand, for very small population size, some genetic operations cannot be implemented. Particularly, mutation operator does not work properly as the mutation rate is very low. Regarding the maximum number of generations, there is no clear indication for considering this value. Generally, it depends upon the number of variables, number of constraints and the size of search space of the optimization problems. Again, from the natural genetics, it is well known that the crossover rate is always greater than that of mutation rate. Usually, the crossover rate varies from 0.6 to 0.95 whereas the mutation rate as 0.1 to 0.15. Sometimes, the mutation rate is considered as $1/n$ where n is the number of genes of the individuals.

For proper and successful functioning of GA, the designing of an appropriate chromosome of solutions to the problem is an important task. In the initial implementation of GAs, the chromosomes were represented by the strings of binary numbers. These GAs are called binary GAs. These are found to be robust search technique to avoid the local optima but the computational cost is very high as compared to the deterministic optimization technique. However, for the problems having large search space and seeking high precession, there arises a number of difficulties in these methods. To overcome these difficulties, real numbers are used to represent the chromosomes in GAs. In this case, a chromosome is coded in the form of vector/matrix of integer/floating point numbers and every component of that chromosome represents a decision variable of the problem. This type of Genetic Algorithm is known as real coded genetic algorithm (RCGA).

After the selection of chromosome representation, the next step is to initialize the chromosomes which will take part in the artificial genetic operations. This procedure produces population size number of chromosomes in which every component is randomly generated within the bounds of the corresponding decision variable. However, in constrained optimization problems, especially for the non-linear constrained optimization problems, the domains of the variables cannot be obtained precisely. So, each chromosome vector is created randomly within its domain

until all constraints are satisfied by applying Repair algorithm (RA). The RA varies from problem to problem. Alternatively, constrained optimization problem can be solved converting it into unconstrained optimization problem using penalty function technique.

After initialization of chromosome representation, the next step is to check the characteristics of each chromosome whether they are highly fitted or not. For this purpose, the fitness value for each chromosome is calculated. Chromosomes with higher fitness will receive larger probabilities of inheritance in subsequent generations, while chromosomes with lower fitness will more likely be eliminated. The selection of a good and accurate fitness function is thus a key to the success of solving any problem quickly. In most of the cases, the objective function of the problem is considered as the fitness function.

3.1. Selection

The selection process is one of the most important factors in the genetic algorithm. This process is stochastic and biased towards the best solution. It depends on the evolutionary principle "Survival of the fittest." The primary objective of this operator is to emphasize on the above average solutions and eliminate the below average solutions from the population for the next generation. This is achieved by performing the following tasks:

- Identify good (usually above-average) solutions in a population.
- Make multiple copies of good solutions.
- Eliminate bad solutions from the population so that multiple copies of good solutions can be placed in the population.

There are several selection schemes, such as roulette wheel selection, ranking selection, elitist preserving selection, stochastic universal sampling selection, truncation selection, tournament selection, etc. Here we shall discuss some of these schemes only.

3.1.1. Ranking selection

In this selection, the ranking order of the individuals with respect to their corresponding fitness value determines the probability of the selection within the current population. The population is actually sorted from the best to worst fashion. The selection probability of each individual is determined according to the ranking rather than their fitness value. There are various methods for assigning the selection probability for each individual on the basis of ranking, including linear and nonlinear ranking methods.

The linear ranking method was proposed by Baker (1985) where each individual in the population is assigned a rank in the increasing order of fitness and the selection probability of i-th individual in the population is calculated by following formula:

$$p_i = \frac{1}{p\_size}\left(f_{\max} - (f_{\max} - f_{\min})\frac{i-1}{p\_size-1}) \right)$$

where $f_{\max}$ and $f_{\min}$ denote the maximum and minimum fitness values of the chromosomes of the population respectively.

On the otherhand, Michalewicz (1996) proposed a non-linear ranking method, also known as exponential ranking method with selection probability p_i of the individual with rank i obtained as

$$p_i = c(1-c)^{i-1} \text{ where } c \in [0,1].$$

3.1.2. Tournament selection

In this selection, a group of chromosomes (individuals) from the population is chosen randomly and the best individual in this group is selected as parents for the next generation which takes part in the genetic operations like crossover and mutation. This process will be repeated until the population size numbers of chromosomes selected. The parameter for the tournament selection is the tournament size *Tour*. *Tour* takes the values ranging from 2 to *p_size*. This selection method with size two is based on the following assumptions:

For unconstrained optimization problems:

(i) Among two selected chromosomes, the chromosome with better fitness value is selected.

For constrained optimization problems:

(i) When two chromosomes (individuals) are feasible then the individual with better fitness value is selected.

(ii) When one chromosome (individual) is feasible and another is infeasible then the feasible one is selected.

(iii) When two chromosomes (individuals) are infeasible with unequal constraints violation, then the chromosome with less constraint violation is selected.

(iv) When two chromosomes (individuals) are infeasible with equal constraints violation, then any one chromosome (individual) is selected.

3.2. Crossover

After the selection process, the resulting chromosomes will take part in the crossover operation to produce the better offspring to improve the current population. This operation operates on two or more parent solutions at a time and produces offspring by combining the features of the parent solutions. In this operation, expected [*p_cross*p_size*] (* denotes the product and [ ] denotes the integral value) number of chromosomes will take part. Hence in order to perform this operation, [*p_cross*p_size*] numbers of chromosomes are selected. The selection process is as follows:

(i) Select two or more chromosomes randomly from the current population.

(ii) Produce offspring by crossover technique.

(iii) Repeat the steps (i) and (ii) for appropriate times as per applied crossover technique.

There are several types of crossover operators, viz. uniform crossover, arithmetic crossover, whole arithmetical crossover, Blend crossover, Laplace crossover etc. Here, we shall discuss some of these operators only.

3.2.1. Whole arithmetical crossover

In this crossover, two different linear combinations of two parent chromosomes (vectors) are considered to produce the offspring.

At t-th. generation, if the parent chromosomes S_v^t and S_w^t are selected randomly from the population for crossover operation, then the produced offspring will be as follows:

$$S_v^{t+1} = a * S_w^t + (1-a) * S_v^t \text{ and } S_w^{t+1} = a * S_v^t + (1-a) * S_w^t$$

where a is a proper fraction which can be chosen randomly.
Alternatively, the proper fraction a can be chosen in the following way:

$$a = p_{max} / (p_{max} + p_{min})$$

where $p_{max} = max\{p_j, j=1, 2\ldots, p\_size\}$,

$p_{min} = min\{p_j, j=1,2,\ldots, p\_size\}$.

and p_j denotes the fitness value of j-th chromosome.

Here, this operator uses a simple static parameter $a \in [0, 1]$, as it always guarantees closure property. If a is constant, then the crossover is called uniform arithmetical crossover. Otherwise, it is known as non-uniform arithmetical crossover.

3.2.2. Blend Crossover

Eshelman and Schaffer (1993) introduced the solution of interval schemata which is similar to the principle to the virtual alphabets. They suggested a crossover operator for real-parameter GAs known as single parameter Blend crossover $(BLX-\alpha)$. Blend crossover creates offspring randomly within a hyper-rectangular box defined by the parent points. In case of *i*-th component of parent chromosomes S_i^1 and S_i^2, the $BLX-\alpha$ randomly chooses a solution in the range $\left(S_i^1 - \alpha D_i,\ S_i^2 + \alpha D_i\right)$ where $D_i = S_i^2 - S_i^1$ and it is assumed that $S_i^2 < S_i^1$. Thus the resulting offspring can be alternatively represented as follows:

$$S_i' = (1-\mu_i) S_i^1 + \mu_i S_i^2 \qquad \text{(A)}$$

where $\mu_i = (1+2\alpha) r_i - \alpha$ and $r_i (\in [0,1])$ is a random number.

From the investigation with some test problems previously, it is observed that $BLX-\alpha$ performs better with $\alpha = 0.5$. If α is zero, this crossover produces a random solution in the range $\left(S_i^1,\ S_i^2\right)$.

In this crossover, the location of the offspring directly depends on the difference of the parent chromosomes. This conception can easily be understood if we rewrite the equation (A) in the following manner.

$$S_i' - S_i^1 = \mu_i \left(S_i^2 - S_i^1 \right)$$

i.e., if the diversity in the parent solution is large, then a large diversity in an offspring solution is expected and vice-versa.

On the otherhand, in case of another type of crossover known as two parameters Blend crossover ($BLX - \alpha - \beta$), an offspring is created by choosing a solution randomly from the range $\left(S_i^1 - \alpha D_i,\ S_i^2 + \beta D_i \right)$, where α and β are the two positive real numbers. Fig.-1 shows one-dimensional case of Blend crossover.

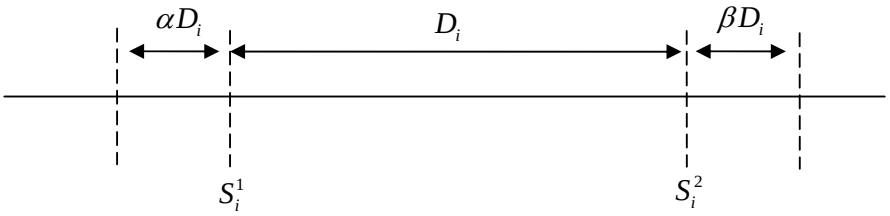

Fig.-1: Blend crossover in one-dimensional case

3.2.3. Laplace crossover

Deep and Thakur (2007) first proposed this operator based on Laplace distribution of probability theory. The density and distribution functions of Laplace distribution are given by

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

$$F(x) = \left\{ \frac{1}{2} \exp\left(\frac{|x-a|}{b} \right), \quad x \le a \right.$$

where $a \in R$ and $b > 0$. Here, a and b are known as location and scale parameters respectively.

In this operation, randomly selected two parent chromosomes

$$S_v = \{ v_1, v_2, \cdots, v_n \} \text{ and } S_w = \{ w_1, w_2, \cdots, w_n \}$$

produce two offspring as follows:

$$S_v' = \{ v_1', v_2', ..., v_n' \} \text{ and } S_v' = \{ w_1', w_2', ..., w_n' \}$$

with $\quad v_i' = v_i + \alpha |v_i - w_i|$

and $w_i' = v_i + \alpha |v_i - w_i|,\ i = 1, 2, ..., n$

where α, a random number, follows the Laplace distribution and it is generated by inverting the distribution function of Laplace distribution as follows:

$$\alpha = \begin{cases} a - b\log_e^r, & r \le 0.5 \\ a + b\log_e^{(r)}, & r > 0.5 \end{cases}$$

where, $r \in (0,1)$ is a uniformly distributed random number.

3.3. Mutation

The main objective of the mutation operator is to introduce the genetic diversity of the population. This operator is used to enhance the fine-tuning capabilities of the system. This is implemented to a single chromosome only with lower probability.

The general algorithm of mutation is as follows:

(i) Calculate the integral value of *p*_mute * *p*_size and store it in *N*.

(ii) Select a chromosome randomly from population and then select a gene of that chromosome in random.

(iii) Produce a new gene corresponding to the selected gene by mutation operation.

(iv) Repeat the steps (ii) to (iii) for *N* times.

Now, we shall discuss some commonly used mutation operator like, Non-uniform mutation and Exponentiation mutation.

3.3.1. Non-uniform Mutation

In this operation, if $S_v = \{v_1, v_2, \cdots, v_n\}$ be a selected chromosome and $v_k \in [l_k, u_k]$, where l_k and u_k are the lower and upper bounds of v_k respectively, be the element to be mutated in t-th generation, the resulting chromosome can be represented as $S_v' = \{v_1, v_2, \cdots, v_k', \cdots, v_n\}$ with $1 \le k \le n$ and

$$v_k' = \begin{cases} v_k + \Delta(t, u_k - v_k) & \text{if a random digit is 0} \\ v_k - \Delta(t, v_k - l_k) & \text{if a random digit is 1} \end{cases}$$

where the function $\Delta(t, y)$ returns a value in the range [0, 1] such that the probability of $\Delta(t, y)$ being close to 0 (zero) as *t* increases. Initially, this property causes this operator to search the space uniformly (when *t* is small) and very locally at later stage. The function $\Delta(t, y)$ is considered as follows either $\Delta(t, y) = y\left(1 - r^{(1-r/T)^b}\right)$

or, $\Delta(t, y) = y \times r(1 - t/T)^b$

where *r* is a random number from [0, 1], *T*, the maximum generation number, *t*, the current generation number and *b*, the system parameter determining the degree of non-uniformity.

3.3.2. Exponential Mutation

Makinen et. al. (1999) proposed a mutation operator to solve some multidisciplinary shape optimization problems using GA in aerodynamics and electromagnetics. Meittinen et. al. (2003) used this operator in a GA to solve the large set of constrained optimization problem. Deep and Thakur (2007) in their work used this operator and named it Makinen, Periaux and Toivanen Mutation (MPTM). In this mutation, if the element (component) v_k of a chromosome $S_v = \{v_1, v_2, v_k, ..., v_n\}$ is selected for this mutation (domain of v_k is $[l_k, u_k]$) the resulting chromosome will be a vector $S_v' = \{v_1, v_2, ..., v_k', ..., v_n\}$ with $1 \le k \le n$ and $v_k' = (1-\hat{t})l_k + \hat{t}u_k$,

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

and

$$t = \frac{x - l_k}{u_k - x},$$

where *r* be a uniformly distributed random number between 0 and 1.

4. Algorithm

The working procedure of GA is given in the following algorithm:

Step-1: Initialize the parameters of Genetic Algorithm, bounds of variables.

Step-2: $t = 0$ [t represents the number of current generation.]

Step-3: Initialize $P(t)$ [$P(t)$ represents the population at t-th generation.]

Step-4: Evaluate the fitness function for each chromosome of $P(t)$.

Step-5: Find the best found result from $P(t)$ with respect to the fitness value of a chromosome.

Step-6: t is increased by unity.

Step-7: If ($t >$ maximum generation number) go to Step-14.

Step-8: Select $P(t)$ from $P(t-1)$ by any selection process like Rowlett wheel selection, ranking selection, tournament selection, etc.

Step-9: Alter $P(t)$ by crossover and mutation operations.

Step-10: Evaluate the fitness function for each chromosome of $P(t)$.

Step-11: Find best found result from $P(t)$ with respect to the fitness value of a chromosome.

.Step-12: Compare the results of $P(t)$ and $P(t-1)$ and store better one.

Step-13: Go to step-6.

Step-14: Print the best found result.

Step-15: Stop.

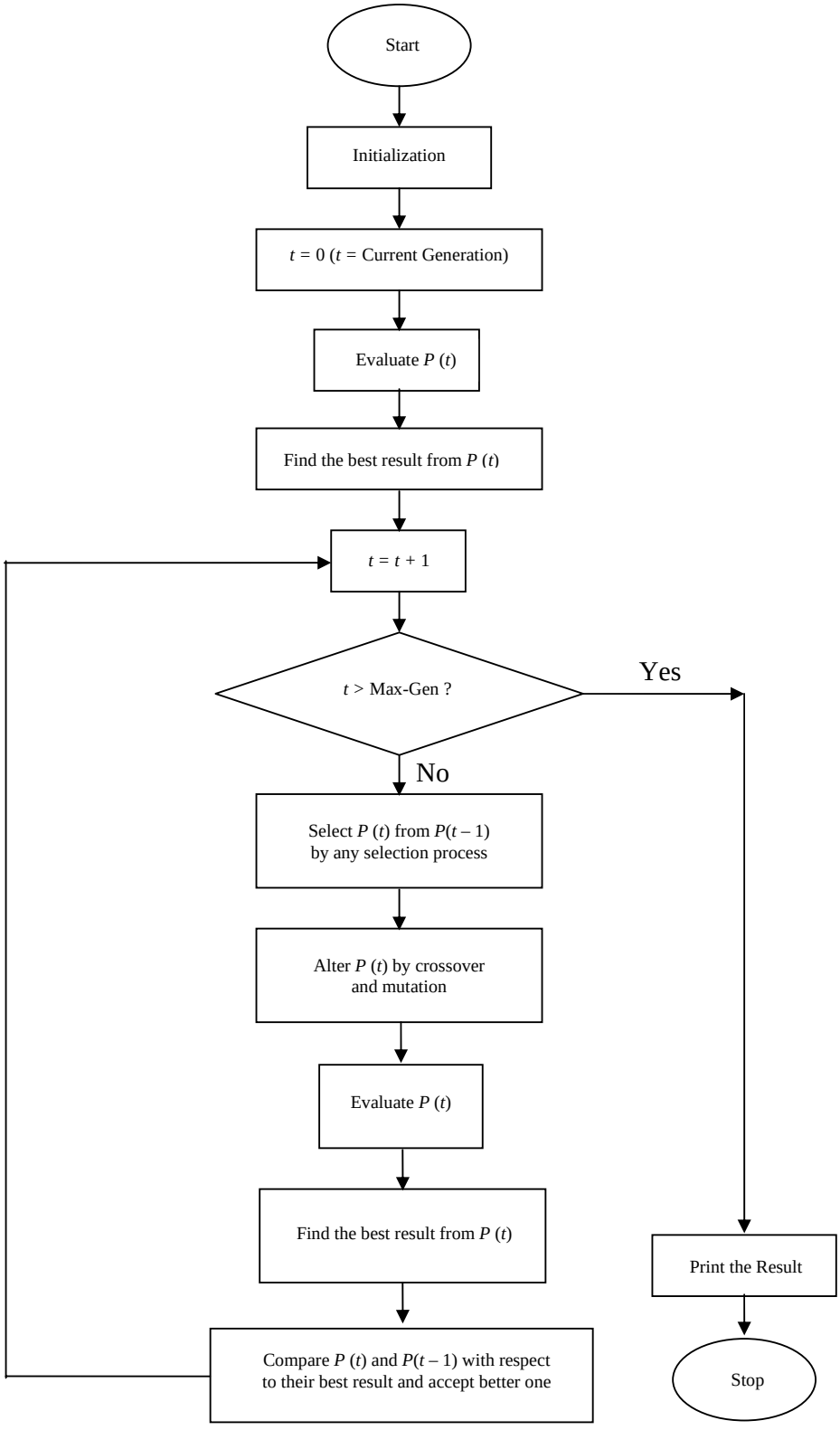

Fig.-2: Flow-chart of simple GA

Fig.-2 shows the flow-chart of simple Genetic Algorithm. In our work, we have considered a set of genetic algorithms GA_{ABC} where A, B and C represent the selection, crossover and mutation operators used respectively. In these algorithms, we have used two selection operators, viz. Ranking selection (R) and Tournament selection (T), three crossover operators, viz. Whole-arithmetical crossover (W), Blend crossover (B) and Laplace crossover (L) and two mutation operators, viz. Non-uniform mutation (N) and Exponential mutation (E).

5. Numerical Example

To compare the performance of different GA operators, we have proposed different GAs with two benchmark test problems having dimensions 10 and 5 which has been coded in C/C++ and tested those on a Pentium IV (3.0 GHz processor, 1 GB RAM) PC under LINUX environment. For this purpose, we have considered the following GA parameter values:

p_size=100, *p_cross*=0.8, *p_mute*=0.15 for both non-uniform mutation and exponential mutation, $b = 5$

Here, the first termination criteria "The best individual does not improve over specified generations" has been used with the specified generations as 10.

In this work, 20 independent runs have been performed by each proposed GA to obtain the best, mean and worst function value, average generation, average function evaluation, CPU time and also the standard deviation of objective function value which have been displayed in the following Table-1 and Table-2 .

1. *De Jong function (5 variables):*

$$F3(x_1, x_2, x_3, x_4, x_5) = \sum_{i=1}^{n} x_i^2$$

$$search\ domain: \ -5.12 \le x_i \le 5.12,\ i = 1,2,3,4,5.$$

2. *Rosenbrock (R_{10}) function (10 variables):*

$$R_{10}(x) = \sum_{j=1}^{4} [100(x_j^2 - x_{j+1}^2)^2 + (x_j - 1)^2]$$

$$search\ domain: -5 < x_j < 10,\ j = 1,2,3,4,5,6,7,8,9,10.$$

Table-1: Result of minimization of Rosenbrock's (R_{10}) function

GAs	Average Generation	Average Objective Function	Best objective function value	Worst objective function value	Standard Deviation (Objective function)	Average function evaluation	Average CPU time
GA_{RWN}	21204	9.5×10^{-8}	9.0×10^{-8}	1.0×10^{-7}	5.12989×10^{-9}	4241170	26.596
GA_{RLN}	21185	9.7×10^{-8}	9.0×10^{-8}	1.0×10^{-7}	4.70162×10^{-9}	4237370	38.750
GA_{RBN}	21228	9.75×10^{-8}	9.0×10^{-8}	1.0×10^{-7}	5.5012×10^{-9}	4245800	50.832
GA_{RWE}	6320	9.9×10^{-8}	9.0×10^{-8}	1.0×10^{-7}	3.07794×10^{-9}	1264280	8.408
GA_{RLE}	4979	0.1×10^{-8}	0.1×10^{-8}	1.0×10^{-7}	0.0	996110	13.393
GA_{RBE}	5523	9.95×10^{-8}	9.0×10^{-8}	1.0×10^{-7}	2.23607×10^{-9}	1104780	17.758
GA_{TWN}	21585	9.6×10^{-8}	9.0×10^{-8}	1.0×10^{-7}	5.02625×10^{-9}	4317310	60.880
GA_{TLN}	21493	0.1×10^{-8}	0.1×10^{-8}	1.0×10^{-7}	0.0	4298830	119.244
GA_{TBN}	21423	9.75×10^{-8}	9.0×10^{-8}	1.0×10^{-7}	4.44262×10^{-9}	4284930	127.668
GA_{TWE}	9271	1.0×10^{-7}	1.0×10^{-7}	1.0×10^{-7}	0.0	1854430	42.199
GA_{TLE}	7888	9.9×10^{-8}	9.0×10^{-8}	1.0×10^{-7}	3.07794×10^{-9}	1577920	43.709
GA_{TBE}	7008	9.85×10^{-8}	9.0×10^{-8}	1.0×10^{-7}	3.66348×10^{-9}	1401810	45.844

Table-2: Result of minimization of De Jong's function

GAs	Average Generation	Average Objective Function value	Best objective function value	Worst objective function value	Standard Deviation (Objective function)	Average function evaluation	Average CPU time
GA_{RWN}	1667	7.05×10^{-8}	1.0×10^{-8}	9.0×10^{-8}	2.74293×10^{-8}	166825	0.366
GA_{RLN}	1100	2.4×10^{-8}	0.0	9.0×10^{-8}	3.13553×10^{-8}	110165	0.269
GA_{RBN}	906	3.4×10^{-8}	0.0	9.0×10^{-8}	3.64764×10^{-8}	90715	0.366
GA_{RWE}	157	3.55×10^{-8}	0.0	8.0×10^{-8}	3.72014×10^{-8}	15810	0.038
GA_{RLE}	106	2.35×10^{-8}	0.0	7.0×10^{-8}	3.06551×10^{-8}	10740	0.046
GA_{RBE}	91	3.2×10^{-8}	0.0	7.0×10^{-8}	2.44088×10^{-8}	9275	0.051
GA_{TWN}	1417	2.45×10^{-8}	0.0	6.0×10^{-8}	3.42552×10^{-8}	141860	0.353
GA_{TLN}	762	3.1×10^{-8}	0.0	9.0×10^{-8}	4.05099×10^{-8}	76330	0.363
GA_{TBN}	1215	2.2×10^{-8}	0.0	9.0×10^{-8}	3.01924×10^{-8}	121690	0.630
GA_{TWE}	201	5.0×10^{-8}	0.0	9.0×10^{-8}	3.26061×10^{-8}	20155	0.132
GA_{TLE}	151	3.65×10^{-8}	0.0	8.0×10^{-8}	2.79614×10^{-8}	15220	0.130
GA_{TBE}	113	4.15×10^{-8}	0.0	8.0×10^{-8}	2.53969×10^{-8}	11385	0.118

From Table-1 and Table-2, it is observed that GA_{RWE} (i.e., Genetic Algorithm with ranking selection, whole-arithmetic crossover and exponential mutation) performs best over others with respect to the objective function value, function evaluation, average generation size and CPU time. However, other two GA versions like, GA_{RLE} and GA_{RBE} produce the quite same results.

5. Conclusion

In this paper, we have shown the performance analysis of various GA versions using different GA operators with two benchmark test problems. From this analysis, it is clear that GA_{RWE} is the best GA version whereas other two

versions GA_{RLE} and GA_{RBE} are moderate. So, one may use any one of these three algorithms alternatively in every operational aspect for solving realistic optimization problems.

References

[1] J.H. Holland, *Adaptation of Natural and Artificial system*, University of Michigan Press, Ann Arbor, 1975.

[2] D. E. Goldberg, *Genetic Algorithms: Search, Optimization and Machine Learning*; Addison Wesley, Reading: MA, 1989.

[3] Z. Michalawicz, *Genetic Algorithms + Data Structure= Evaluation Programs*; Springer Verlag, Berlin, 1996.

[4] M. Sakawa, *Genetic Algorithms and fuzzy multiobjective optimization*; Kluwer Academic Publishers, USA, 2002.

[5] M. Mitchell, *Introduction to Genetic Algorithms*, PHI, New Delhi, 1996.

[6] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*, New York : Wiley, 1997.

[7] J.E. Baker, "Adapted selection methods for genetic algorithms," *Proceedings of the first international conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 101-111,1985

[8] K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, pp. 311-338, 2000.

[9] K. Meittinen, M.M. Makela and J. Toivanen, "Numerical comparison of some penalty based constraint handling techniques in genetic algorithms," *Journal of Global Optimization*, vol. 27, pp. 427- 446, 2003.

[10] R. Chelouah and P. Siarry, "A continuous genetic algorithm designed for the global optimization of multimodal functions," *J. Heuristics*, vol. 6, pp. 191- 214, 2000.

[11] K.F.C. Yiu, Y. Liu and K.L. Teo, "A hybrid descent Method for Global Optimization," *Journal of Global Optimization*, vol. 28, pp. 229-238, 2004.

[12] R.K. Jana and M.P. Biswal, "Stochastic simulation based genetic algorithm for chance constraint programming problems with continuous variable," *International Journal of Computer Mathematics*, vol. 81, pp. 1069-1076, 2004.

[13] K. Deb, A. Anand and D. Joshi, "A computationally efficient evolutionary algorithm for real parameters evolution," *Evolutionary Computation Journal*, vol. 10, no. 4, pp. 371-395, 2002.

[14] K. Deep and M. Thakur, "A new crossover operator for real coded genetic algorithms," *Applied Mathematics and Computation*, vol. 188, no. 1, pp. 895-911, 2007.

[15] R.A.E. Makinen, J. Periaux and J. Toivanen, "Multidisciplinary shape optimization in aerodynamics and electromagnetic using genetic algorithm," *International Journal for Numerical Methods in Fluids*, vol. 30, no. 2, pp. 149-159, 1999.