Investigation of Mutation Schemes in Real-Parameter Genetic Algorithms

Debayan Deb[1] and Kalyanmoy Deb[2]

[1] Department of Computer Science, Michigan State University,
East Lansing, MI 48824, USA
ronny3050@gmail.com
[2] Department of Mechanical Engineering, Indian Institute of Technology Kanpur,
208016, India
deb@iitk.ac.in

Abstract. In this paper, we investigate the effect of five different mutation schemes for real-parameter genetic algorithms (RGAs). Based on extensive simulation studies, it is observed that a mutation clock implementation is computationally quick and also efficient in finding a solution close to the optimum on four different problems used in this study. Moreover, parametric studies on the polynomial mutation operator identify a working range of values of these parameters. This study signifies that the long-suggested mutation clock operator should be considered as a valuable mutation operator for RGAs.

1 Introduction

Real-parameter evolutionary optimization has received a lot of attention over the years [1–4]. In a real-parameter genetic algorithm (RGA), the mutation operator is used primarily as a mechanism for maintaining diversity in the population [5, 6]. In contrast to a recombination operator, a mutation operator operates on only one individual at a time. In RGAs, several mutation operators are suggested – random mutation [7], Gaussian mutation [8], polynomial mutation [9, 10], and others. The effect is to perturb the current variable value to a neighboring value. While operated on multi-variable individual, one common strategy is to mutate each variable with a pre-specified probability.

Despite the existence of different mutation operators for perturbing a variable, the procedure of applying mutation operator to GA population members can be applied in many different ways. The effect of different mutation schemes on the performance of a real-parameter GA (RGA) is not investigated adequately yet. In this paper, we suggest and compare five different mutation schemes based on their effects on four different standard test problems. In all cases, we use the polynomial mutation for perturbing a variable. Similar studies can also be performed with other mutation operators.

In the remainder of the paper, we briefly describe the polynomial mutation operator. Thereafter, we present five different mutation schemes suggested here.

B.K. Panigrahi et al. (Eds.): SEMCCO 2012, LNCS 7677, pp. 1–8, 2012.

Simulation results of a real-parameter GA with identical selection and recombination operators but different mutation schemes are compared against each other and against no mutation. Interesting observations are made. Finally, the effect of polynomial mutation strength and probability are investigated for the winning mutation scheme. Conclusions of the study are then made.

2 Polynomial Mutation in Real-Parameter GAs

Deb and Agrawal [9] suggested a polynomial mutation operator with a user-defined index parameter (η_m). Based on a theoretical study, they concluded that η_m induces an effect of a perturbation of $O((b-a)/\eta_m)$ in a variable, where a and b are lower and upper bounds of the variable. They also found that a value $\eta_m \in [20, 100]$ is adequate in most problems that they tried. In this operator, a polynomial probability distribution is used to perturb a solution in a parent's vicinity. The probability distribution in both left and right of a variable value is adjusted so that no value outside the specified range $[a, b]$ is created by the mutation operator. For a given parent solution $p \in [a, b]$, the mutated solution p' for a particular variable is created for a random number u created within $[0, 1]$, as follows:

$$p' = \begin{cases} p + \bar{\delta}_L(p - x_i^{(L)}), \text{ for } u \leq 0.5, \\ p + \bar{\delta}_R(x_i^{(U)} - p), \text{ for } u > 0.5. \end{cases} \quad (1)$$

Then, either of the two parameters ($\bar{\delta}_L$ or $\bar{\delta}_R$) is calculated, as follows:

$$\bar{\delta}_L = (2u)^{1/(1+\eta_m)} - 1, \quad \text{for } u \leq 0.5, \quad (2)$$

$$\bar{\delta}_R = 1 - (2(1-u))^{1/(1+\eta_m)}, \quad \text{for } u > 0.5. \quad (3)$$

To illustrate, Figure 1 shows the probability density of creating a mutated child point from a parent point $p = 3.0$ in a bounded range of [1,8] with $\eta_m = 20$.

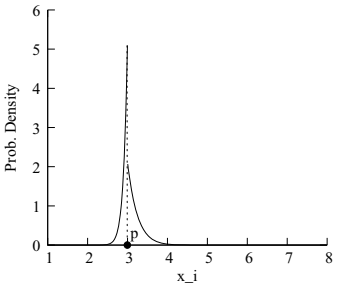

Fig. 1. Probability density function of creating a mutated child solution

3 Five Mutation Schemes

We discuss five different mutation schemes used in this study. We also compare all five schemes with Scheme 0 in which no mutation operator is used.

Scheme 1: Usual Mutation
This mutation scheme follows the usual method of mutating each and every variable one at a time with a pre-defined mutation probability p_m [5, 10]. Usually, $p_m = 1/n$ is used, so that on an average, one variable gets mutated per individual. A random number $u \in [0, 1]$ is created for every variable for an individual and if $u \leq p_m$ the variable is mutated using the polynomial mutation operator.

Scheme 2: Mutation Clock
The usual mutation scheme described above requires n random numbers to be created per individual. To reduce the computational complexity, Goldberg [5] suggested a *mutation clock* scheme for binary-coded GAs, in which once a bit is mutated, the next bit to be mutated (in the same or in a different individual) is determined by using an exponential probability distribution: $p(t) = \lambda \exp(-\lambda t)$, where λ is the inverse of the average occurrence of mutations. We implement here mutation clock, for the first time, to real-parameter GAs. With a mutation probability of p_m, on an average, one mutation will occur in $1/p_m$ variables. Thus, $\lambda = p_m$. For a mutation event, a random number $u \in [0, 1]$ is first chosen. Then, equating $u = \int_{t=0}^{l} p_m \exp(-p_m t)dt$, we obtain the next occurrence (l) of mutation as:

$$l = \frac{1}{p_m} \log(1 - u). \tag{4}$$

If k-th variable on i-th individual in the population is currently mutated, the next variable to be mutated is $((k + l) \bmod n)$-th variable of the $((k + l)/n)$-th individual from current individual in the population. At every generation, the initial variable to be mutated is calculated using $i = k = 1$. This operator should, on an average, require n times less number of random numbers than that required in Scheme 1.

Scheme 3: One Mutation per Solution
Here, we choose a random variable $i \in [1, n]$ and x_i is mutated using the polynomial mutation. Exactly, one mutation is performed per individual.

Scheme 4: Fixed Mutation
This mutation scheme is similar to Scheme 3, except that every variable is given an equal chance, thereby implementing a less noisy implementation of Scheme 3. For this purpose, variables are ordered in a random order in every generation and then variables are mutated using the polynomial mutation according to the sorted order of variables from first to last population member. After n variables are mutated in n top population members, the same order is followed from $(n+1)$-th population member. Again, exactly one mutation is performed per individual.

Scheme 5: Diversity based Mutation
In this mutation scheme, we put more probability for mutation to a variable that has less population-wise diversity. To implement, first the variance of values of each variable across the population members is computed and variables are sorted in ascending order of variance. Thereafter, an exponential probability distribution ($p(i) = \lambda \exp(-\lambda i)$ for $i \in [0, n - 1]$ is used. To make the above a probability distribution, λ is used by finding the root of the following equation for a fixed n: $\lambda \exp(-n\lambda) - \exp(-\lambda) - \lambda + 1 = 0$. Thereafter, for a random number $u \in [0, 1]$, the variable $(l + 1)$ that should be mutated is given:

$$l = \frac{1}{\bar{\lambda}} \log\left(1 - u(1 - \exp(-n\bar{\lambda}))\right). \tag{5}$$

For $n = 15$, $\bar{\lambda} = 0.168$ is found. This scheme makes one mutation per individual.

4 Results

We consider four different problems to investigate the effect of five suggested mutation schemes. The objective functions have minimum at $x_i = 0$ $(i = 1, \ldots, n)$ for the first three problems, and $x_i = 1$ $(i = 1, \ldots, n)$ for the fourth problem. However, in each case, we consider 15 variables $(n = 15)$, initialized and bounded within $x_i \in [-5, 10]$. This does not make the optimum exactly at the middle of the search space. We use the following GA parameter settings: (i) No. of real variables, $n = 15$, (ii) Population size = 150, (iii) SBX operator probability, $p_c = 0.9$, (iv) SBX operator index, $\eta_c = 2$, (v) Polynomial mutation operator probability, $p_m = 1/\text{n}$, (vi) Polynomial mutation operator index, $\eta_m = 20$, and (vii) Termination parameter $\epsilon_T = 0.01$. For each mutation scheme, we make 51 different runs starting from different initial populations, however the same set of initial populations are used for all mutation schemes. Mutation schemes are then compared with each other and with the zero mutation scheme.

4.1 Ellipsoidal Function

This function is unimodal and additively separable having $f_{ell}(\mathbf{x}^*) = 0$: $f_{ell}(x) = \sum_{i=1}^{D} i x_i^2$. The performance of different mutation schemes on the problem is shown in Table 1. While all mutation schemes perform successfully on 100% runs, GAs without mutation have failed to find the required solution in more than 50% of the runs. This amply suggests the importance of using a mutation scheme in RGAs. Figure 2 shows how the objective value is reduced with generation for all five mutation schemes. Clearly, Schemes 1 and 2 perform the best.

4.2 Schwefel's Function

The function is as follows: $f_{sch}(x) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j \right)$. The performance of different mutation schemes on this problem is shown in Table 2. Figure 3 shows the

Table 1. Performance of different mutation schemes on ellipsoidal problem

		Scheme 0	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5
Run time	min.	**0.05**	0.07	0.06	0.11	0.10	0.15
	avg.	2.96	**0.08**	**0.08**	0.15	0.14	0.20
	med.	3.56	0.09	**0.08**	0.15	0.15	0.19
	max.	4.49	0.10	**0.09**	0.20	0.20	0.25
Generations	min.	**122.00**	157.00	149.00	277.00	247.00	311.00
	avg.	7,701.12	190.27	**188.84**	385.00	370.11	410.60
	med.	10,000.00	193.00	**185.00**	389.00	375.00	408.00
	max.	10,000.00	232.00	**226.00**	500.00	502.00	539.00
Mutations	min.	0.00	15,737.00	**15,321.00**	27,700.00	24,700.00	31,100.00
	avg.	0.00	**19,113.27**	19,629.78	38,500.00	37,011.76	41,060.78
	med.	0.00	19,259.00	**19,196.00**	38,900.00	37,500.00	40,800.00
	max.	0.00	23,256.00	**23,232.00**	50,000.00	50,200.00	53,900.00

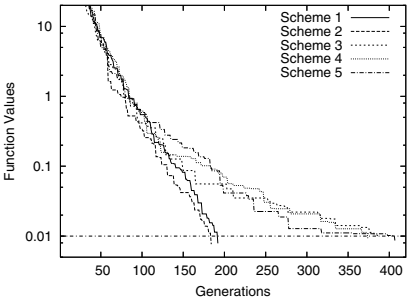

Fig. 2. Variation of number of function evaluations with generation for ellipsoidal problem

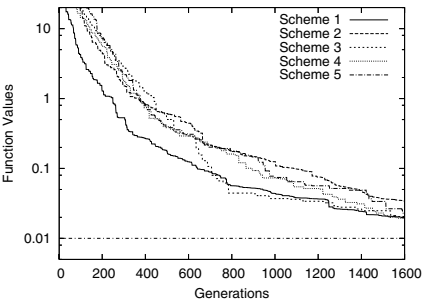

Fig. 3. Variation of number of function evaluations with generation for Schwefel's function

Table 2. Performance of different mutation schemes on Schwefel's problem

		Scheme 0	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5
Run time	min.	3.99	0.52	0.69	0.59	**0.47**	0.68
	avg.	4.72	1.27	1.26	**0.91**	0.94	1.04
	med.	4.96	1.22	1.23	**0.88**	0.94	1.03
	max.	5.38	2.60	1.84	1.85	**1.53**	1.66
Generations	min.	10,000.00	1,065.00	1,522.00	1,335.00	**1,060.00**	1,301.00
	avg.	10,000.00	2,559.57	2,788.29	2,053.10	2,146.77	**1,997.37**
	med.	10,000.00	2,490.00	2,718.00	1,979.00	2,122.00	**1,971.00**
	max.	10,000.00	5,307.00	4,076.00	4,210.00	3,470.00	**3,161.00**
Mutations	min.	0.00	106,891.00	159,104.00	133,500.00	**106,000.00**	130,100.00
	avg.	0.00	257,230.11	289,855.70	205,309.80	214,676.47	**199,737.25**
	med.	0.00	250,244.00	282,629.00	197,900.00	212,200.00	**197,100.00**
	max.	0.00	533,128.00	423,452.00	421,000.00	347,000.00	**316,100.00**

reduction in objective value with generation. All five schemes performs almost equally well for this problem.

4.3 Ackley's Function

Next, we consider Ackley's function: $f_{ack}(x) = 20 + e - 20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right)$. The performance of different mutation schemes on this problem is shown in Table 3. Figure 4 shows the variation of objective value with generation. It is clear that Schemes 1 and 2 perform better than other three schemes, including the zero-mutation scheme.

4.4 Rosenbrock's Function

Rosenbrock's function is as follows: $f_{ros}(x) = \sum_{i=1}^{D-1}\left[100\left(x_{i+1} - x_i^2\right)^2 + (x_1 - 1)^2\right]$. Since this is a difficult problem to solve, we use $\epsilon_T = 15$. The performance of

Table 3. Performance of different mutation schemes on Ackley's function

		Scheme 0	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5
Run time	min.	**0.09**	0.11	0.10	0.53	0.41	0.58
	avg.	3.72	**0.14**	**0.14**	0.71	0.68	0.86
	med.	3.93	0.14	**0.13**	0.72	0.69	0.86
	max.	4.08	**0.18**	0.19	1.02	0.87	1.56
Generations	min.	**207.00**	224.00	225.00	1,220.00	969.00	1,132.00
	avg.	9,427.43	**291.88**	303.31	1,641.90	1,563.51	1,682.78
	med.	10,000.00	**291.00**	296.00	1,652.00	1,598.00	1681.00
	max.	10,000.00	**372.00**	429.00	2,364.00	2,031.00	3,066.00
Mutations	min.	0.00	**22,392.00**	23,523.00	122,000.00	96,900.00	113,200.00
	avg.	0.00	**29,349.04**	31,527.67	164,190.19	156,350.98	168,278.43
	med.	0.00	**29,068.00**	30,891.00	165,200.00	159800.00	168,100.00
	max.	0.00	**37,258.00**	44,327.00	236,400.00	203,100.00	306,600.00

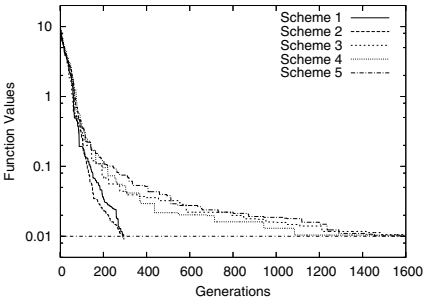

Fig. 4. Variation of f with generation for Ackley's function

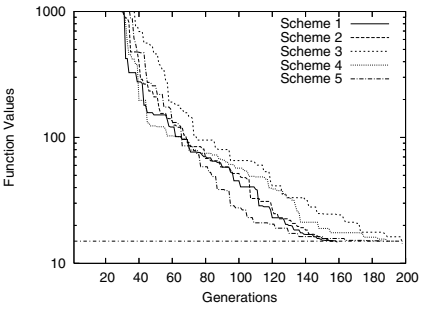

Fig. 5. Variation of f with generation for Rosenbrock's function

different mutation schemes on this problem is shown in Table 4. Figure 5 indicates that Schemes 1 and 2 perform slightly better.

On four different problems, it is observed that in terms of function evaluations, Schemes 1 and 2 perform better than the other three mutation schemes including the zero mutation scheme. Tables show that Scheme 2 requires much smaller number of computational time compared to Scheme 1. Hence, we recommend the use of mutation clock as an efficient mutation scheme for real-parameter GAs.

5 Parametric Studies with Mutation Index and Mutation Probability

Having established the superiority of mutation clock, we now perform a parametric study of the distribution index η_m for Scheme 2. Figure 6 shows that $\eta_m \approx 100$ performs the best on the ellipsoidal problem. Figure 7 shows $\eta_m \in [100, 150]$ produce best results for Schwefel's function. Similar plots for Ackley's and Rosenbrock's functions are found but for brevity we do present them here.

Table 4. Performance of different mutation schemes on Rosenbrock's problem

		Scheme 0	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5
Run time	min.	0.08	**0.05**	**0.05**	**0.05**	**0.05**	0.06
	avg.	4.63	0.24	**0.14**	0.29	0.27	0.30
	med.	5.14	0.08	**0.07**	0.09	0.08	0.10
	max.	5.51	3.09	**0.99**	2.45	2.77	2.21
Generations	min.	165.00	**102.00**	106.00	110.00	126.00	110.00
	avg.	9433.04	473.75	**309.12**	632.88	598.29	561.21
	med.	10,000.00	160.00	**159.00**	199.00	196.00	188.00
	max.	10,000.00	6,284.00	**2,163.00**	5,481.00	6,227.00	4,207.00
Mutations	min.	0.00	**10,148.00**	11,034.00	11,000.00	12,600.00	11,000.00
	avg.	0.00	47,593.15	**32,159.07**	63,288.23	59,829.41	56,121.57
	med.	0.00	**15,827.00**	16,544.0	19,900.00	19,600.00	18,800.00
	max.	0.00	631,346.00	**225,044.00**	548,100.00	622,700.00	420,700.00

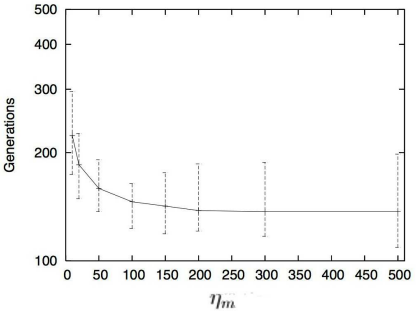

Fig. 6. Parametric study of η_m for Scheme 2 on ellipsoidal function

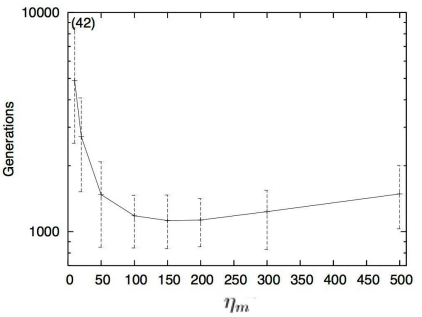

Fig. 7. Parametric study of η_m for Scheme 2 on Schwefel's function

Next, we perform a parametric study of mutation probability p_m with Scheme 2 and fix the mutation index as $\eta_m = 100$ (which is found to be near-optimal above). We use $p_m = k/n$ with $k = 0.01$, 0.1, 0.5, 0.75, 1, 1.5, 2, and 5. The rest of the GA parameters are kept the same as before. Figure 8 shows the number of generations needed for 51 runs for the ellipsoidal problem. Mutation probabilities of $0.01/n$ and $0.1/n$ are not found to produce a reasonable result. The figure shows that $0.5/n$ to $1.5/n$ perform the best. Thus, the usual practice of $p_m = 1/n$ is justified by our study. Similar results are found for other two functions as well. Thus, based on the above simulation results, we suggest the following parameter values for the polynomial mutation operator which is to be applied with SBX recombination operator with $\eta_c = 2$ and the binary tournament selection operator:

Mutation scheme: = Mutation clock [5],
Mutation index, η_m [9]: = $[100, 150]$,
Mutation probability, p_m:= $[0.5/n, 1.5/n]$, where n is the number of variables.

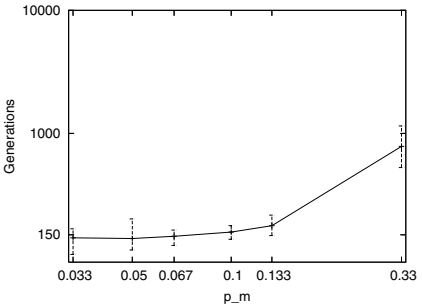

Fig. 8. Parametric study of p_m for Scheme 2 on the ellipsoidal function

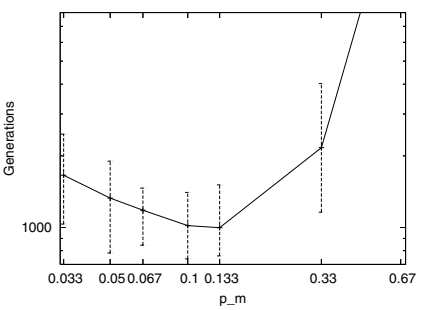

Fig. 9. Parametric study of p_m for Scheme 2 on Schwefel's function

6 Conclusions

Mutation operators are used for maintaining diversity in a GA. In this paper, we have suggested five different mutation schemes for real-parameter GAs. Based on their performance, we conclude the following: (i) any of the five mutation operators is better than not performing any mutation, (ii) the mutation clock operator which was suggested in eighties has been long forgotten and has been found here to be the fastest in terms of its execution time and best in terms of its performance. Parametric studies have also found a range of mutation index for the polynomial mutation ($\eta_m \in [100, 150]$) and mutation probability $p_m \in [0.5/n, 1.5/n]$ (where n is the number of variables) for its best performance. Similar studies can be performed with other real-parameter mutation operators.

References

1. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation Journal 9(2), 159–195 (2000)
2. Price, K.V., Storn, R., Lampinen, J.: Differential Evolution: A Practical Approach to Global Optimization. Springer (2005)
3. Kennedy, J., Eberhart, R.C., Shi, Y.: Swarm intelligence. Morgan Kaufmann (2001)
4. Deb, K., Agrawal, R.B.: Simulated binary crossover for continuous search space. Complex Systems 9(2), 115–148 (1995)
5. Goldberg, D.: Genetic algorthims in search, optimization, and machine learning. Addison-Wesley, Reading (1989)
6. Holland, J.H.: Adaptation in Natural and Artificial Systems. MIT Press, Ann Arbor (1975)
7. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer, Berlin (1992)
8. Schwefel, H.-P.: Collective phenomena in evolutionary systems. In: Checkland, P., Kiss, I. (eds.) Problems of Constancy and Change – the Complementarity of Systems Approaches to Complexity, pp. 1025–1033. Intl. Soc. for General Sys. Res, Budapest (1987)
9. Deb, K., Agrawal, S.: A niched-penalty approach for constraint handling in genetic algorithms. In: Proceedings of the Intl. Conf. on ANN and GAs, pp. 235–243. Springer (1999)
10. Deb, K.: Multi-objective optimization using evolutionary algorithms. Wiley, Chichester (2001)