Analysing mutation schemes for real-parameter genetic algorithms

Kalyanmoy Deb*

Department of Electrical and Computer Engineering,
Michigan State University,
428 S. Shaw Lane, 2120 EB, East Lansing, MI 48824, USA
Fax: 517 353 1980 E-mail: kdeb@egr.msu.edu
*Corresponding author

Debayan Deb

Department of Computer Science and Engineering,
Michigan State University,
East Lansing, MI 48824, USA
E-mail: debdebay@msu.edu

Abstract: Mutation is an important operator in genetic algorithms (GAs), as it ensures maintenance of diversity in evolving populations of GAs. Real-parameter GAs (RGAs) handle real-valued variables directly without going to a binary string representation of variables. Although RGAs were first suggested in early '90s, the mutation operator is still implemented variable-wise – in a manner that is independent to each variable. In this paper, we investigate the effect of five different mutation schemes for RGAs using two different mutation operators – polynomial and Gaussian mutation operators. 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 for both mutation operators. Moreover, parametric studies with their associated parameters reveal suitable working ranges of the parameters. Interestingly, both mutation operators with their respective optimal parameter settings are found to possess a similar inherent probability of offspring creation, a matter that is believed to be the reason for their superior working. This study signifies that the long suggested mutation clock operator should be considered as a valuable mutation operator for RGAs.

Keywords: genetic algorithms; mutation operator; mutation clock; polynomial mutation; real-parameter optimisation; Gaussian mutation.

Reference to this paper should be made as follows: Deb, K. and Deb, D. (2014) 'Analysing mutation schemes for real-parameter genetic algorithms', *Int. J. Artificial Intelligence and Soft Computing*, Vol. 4, No. 1, pp.1–28.

Biographical notes: Kalyanmoy Deb is a Professor and Koenig Endowed Chair of Electrical and Computer Engineering at Michigan State University, East Lansing, USA and has been working in soft computing area for the past 25 years. He has published more than 330 research papers, two textbooks, and 18 edited books. Citation count of his publications on Google Scholar is more than 40,000 with a h-index of 72. He has received Infosys prize, Shanti Swarup Bhatnagar prize, Fredrich Wilhelm Bessel research award, Cajastur Mamdai prize for soft computing and many other recognitions, including the Distinguished Alumnus award from his alma mater IIT Kharagpur.

Debayan Deb is an undergraduate student of Computer Science and Engineering at Michigan State University. He has a keen interest in soft computing approaches. While at school, he has participated in many computer programming contests.

1 Introduction

Mutation operator in a genetic algorithm (GA) is used primarily as a mechanism for maintaining diversity in the population (Goldberg, 1989; Holland, 1975). In contrast to a recombination operator, a mutation operator operates on only one evolving population member at a time and modifies it independent to the rest of the population members. Although mutation operator alone may not constitute an efficient search, along with a suitable recombination operator, mutation operator plays an important role in making the overall search efficient (Goldberg, 1989).

Early researchers on GAs used binary strings to represent a real-valued variable. For an ℓ-bit string mapped between the specified variable bounds [a, b], GA's search is limited to a discrete set of 2^{ℓ} equi-spaced points within the bounds. In such a binary-coded GA (BGA), usually a one-point or two-point crossover operators were used. For mutation, a bit-wise mutation operator which attempted to mutate every bit (alter the bit to its complement) with a probability p_m independently to the outcome of mutation to other bits. Parametric studies (Schaffer et al., 1989) have shown that a mutation probability of $p_m = 1 / L$ (where L is the total number of bits used to represent all n variables) performed the best.

Researchers have realised that there are certain shortcomings of using a BGA to solve real-parameter optimisation problems. One of the reasons was the artificial discreteness associated with the coding mechanism and the second was the biasness of mutated solutions towards certain parts of the search space. Starting early 1990s, real-parameter GAs (RGAs) (Eshelman and Schaffer, 1993; Voigt et al., 1995; Deb and Agrawal, 1995; Higuchi et al., 2000; Kita et al., 1999) were suggested. However, all of these studies concentrated in developing a recombination operator that would take two real values (parents) and recombine (or blend) to create two new real values as offsprings. The recombination operations were performed variable-wise or vector-wise. For RGAs, several mutation operators were also suggested – random mutation (Michalewicz, 1992), Gaussian mutation (Schwefel, 1987), polynomial mutation (Deb and Agrawal, 1999; Deb, 2001), and others. The effect is to perturb the current variable value (parent) to a neighbouring value (offspring). While operated on a multi-variable population member, one common strategy has been to mutate each variable with a pre-specified probability p_m at a time using one of the above mutation operators. Not much attention has been made so far in analysing whether this one-at-a-time mutation scheme is efficient or there exists other mutation schemes that may be more efficient in RGAs.

In this paper, we consider two different and commonly-used mutation operators (polynomial mutation and Gaussian mutation) and investigate the performance of five different mutation schemes including the usual one-at-a-time scheme. The mutation schemes are compared based on computational time and the required number of function evaluations. Thereafter, a parametric study of the best-found mutation scheme is performed to not only suggest an efficient mutation scheme but also an adequate range of

its associated parameter values. After finding the optimal parameter settings, the corresponding probability distributions for creating offspring solutions are compared to reveal interesting observations.

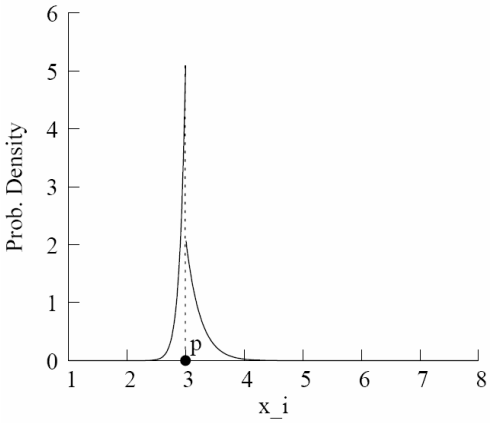
In the remainder of this paper, we briefly describe two mutation operators used in this study in detail. Thereafter, we present five different mutation schemes. Simulation results of a RGA with identical selection and recombination operators but different mutation schemes are compared against each other and against a no-mutation scheme. Interesting observations are made. The effect of mutation strength and mutation probability for both mutation operators are investigated for the winning mutation scheme. Finally, both mutation schemes are compared based on their inherent probability distributions. Conclusions of the study are finally made.

2 Polynomial Mutation in RGAs

Deb and Agrawal (1999) 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:

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

Figure 1 Probability density function of creating a mutated child solution using polynomial mutation operator



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

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

$$\bar{\delta}_R = 1 - \left(2(1-u)\right)1/(1+\eta m), \quad \text{for } u > 0.5. \tag{3}$$

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 $\eta_m = 20$.

3 Gaussian mutation in RGAs

Let $x_i \in [a_i, b_i]$ be a real variable. Then the truncated Gaussian mutation operator changes x_i to a neighbouring value x_i' using the following probability distribution:

$$p(x_i'; x_i, \sigma_i) = \begin{cases} \dfrac{\frac{1}{\sigma_i}\phi\left(\frac{x_i' - x_i}{\sigma_i}\right)}{\Phi\left(\frac{b_i - x_i}{\sigma_i}\right) - \Phi\left(\frac{a_i - x_i}{\sigma_i}\right)} & \text{if } a_i \le x_i' \le b_i, \\ 0, & \text{otherwise,} \end{cases} \tag{4}$$

where $\phi(\zeta) = \dfrac{1}{\sqrt{2\pi}} \exp\left(-\dfrac{1}{2}\zeta^2\right)$ is the probability distribution of the standard normal distribution and $\Phi(\cdot)$ is the cumulative distribution function. This mutation operator has a mutation strength parameter σ_i for every variable, which should be related to the bounds a_i and b_i. We use $\sigma = \sigma_i / (b_i - a_i)$ as a fixed non-dimensionalised parameter for all n variables.

To implement the above concept, we use the following equation to compute the offspring x_i':

$$x_i' = x_i + \sqrt{2}\sigma(b_i - a_i)\,\mathrm{erf}^{-1}(u_i'), \tag{5}$$

where u_i' and an approximated value of $\mathrm{erf}^{-1}(u_i')$ are calculated as follows (with $u_i \in [0, 1]$ a random number):

$$u_i' = \begin{cases} 2u_L(1 - 2u_i), & \text{if } u_i \le 0.5, \\ 2u_R(2u_i - 1), & \text{otherwise,} \end{cases}$$

$$\mathrm{erf}^{-1}(u_i') \approx \mathrm{sign}(u_i') \left(\sqrt{\left(\frac{2}{\pi\alpha} + \frac{\ln(1 - u_i'^2)}{2} \right)^2 - \frac{\ln(1 - u_i'^2)}{\alpha}} - \frac{2}{\pi\alpha} + \frac{\ln(1 - u_i'^2)}{2} \right)^{\frac{1}{2}},$$

where $\alpha = \dfrac{8(\pi - 3)}{3\pi(4 - \pi)} \approx 0.140012$ and $\mathrm{sign}(u_i')$ is –1 if $u_i' < 0$ and is +1 if $u_i' \ge 0$. Also, u_L and u_R are calculated as follows:

$$u_L = 0.5\left(\mathrm{erf}\left(\frac{a_i - x_i}{\sqrt{2}(b_i - a_i)\sigma} \right) + 1 \right),$$

$$u_R = 0.5\left(\text{erf}\left(\frac{b_i - x_i}{\sqrt{2}(b_i - a_i)\sigma}\right)+1\right),$$

where erf() function is the standard C-library error function:

$$\text{erf}(y) = \frac{2}{\sqrt{\pi}}\int_0^y \exp(-t^2)\,dt.$$

The above mutated offspring x_i' will always lie within $[a_i, b_i]$ for any random number $u_i \in [0, 1]$. Note that for $u_i \le u_L$, $x_i' = a_i$ will be the outcome and for $u_i \ge u_R$, $x_i' = b_i$ will be automatically created.

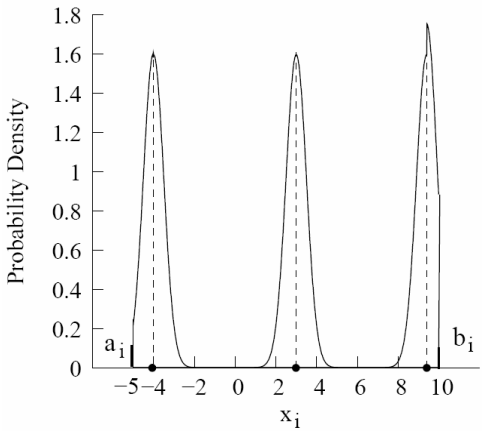
Thus, the Gaussian mutation procedure for mutating i^{th} variable x_i is as follows:

Step 1 Create a random number $u_i \in [0, 1]$.

Step 2 Use equation (5) to create offspring x_i' from parent x_i.

Figure 2 shows the probability density functions for three parents $x_i = -4.5$, 3.0 and 9.4 with bounds [–5, 10] and $\sigma = 1/30$. Note that the probabilities of creating an offspring $x_i' < -5$ and $x_i' > 10$ in the respective cases are zero. Also, for two extreme parents (close to respective boundaries), the Gaussian distribution is truncated to make zero probabilities of creating offsprings outside the specified lower and upper bounds.

Figure 2 Probability density functions for creating mutated offspring solutions from three parents using the Gaussian mutation operator



4 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.

4.1 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 (Goldberg, 1989; Deb, 2001). 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.

4.2 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 (1989) 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:

$$p(t) = \lambda \exp(-\lambda t), \tag{6}$$

where λ is the inverse of the average occurrence of mutations. We implement here the mutation clock operator, for the first time, to RGAs. 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 follows:

$$l = \frac{1}{p_m} \log(1-u). \tag{7}$$

If k^{th} variable on i^{th} individual in the population is currently mutated, the next variable to be mutated is $((k + l) \bmod n)^{\text{th}}$ variable of the $((k + l) / n)^{\text{th}}$ 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.

4.3 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.

4.4 Scheme 4: fixed strategy 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 fashion in every generation and then variables are mutated using the polynomial mutation operator according to the sorted order of variables from first to last population member. If n is smaller than the population size, after n variables are mutated for the n top population members, the same order is followed from $(n + 1)^{\text{th}}$ population member. Again, exactly one mutation is performed per individual. If n is larger than population size, the above scheme is implemented for multiple

generations so as to provide an equal chance of mutation to each variable over a number of consecutive generations.

4.5 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 as a probability distribution, $\bar{\lambda}$ is used by finding the root of the following equation for a fixed n:

$$\lambda\exp(-n\lambda) - \exp(-\lambda) - \lambda + 1 = 0. \tag{8}$$

For $n = 15$, $\bar{\lambda} = 0.169$ is found. We provide the respective $\bar{\lambda}$ value for a few n (number of variables) in the following table:

n	$\bar{\lambda}$
5	0.364
10	0.226
15	0.169
20	0.136
30	0.101
50	0.068
100	0.039

Thereafter, for a random number $u \in [0, 1]$, the variable $(l + 1)$ that should be mutated is found as follows:

$$l = \frac{1}{\bar{\lambda}}\log\left(1 - u\left(1 - \exp\left(-n\bar{\lambda}\right)\right)\right). \tag{9}$$

This scheme makes one mutation per individual.

5 Results with polynomial mutation

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 ($n = 15$), initialised and bounded within a skewed range: $x_i \in [-5, 10]$. These bounds do not cause the optimum to lie exactly at the middle of the chosen search space, a feature that may be otherwise exploited by an algorithm to quickly converge to the optimum. We use the following GA parameter settings:

1 no. of real variables, $n = 15$

2 population size = 150

3 SBX operator probability, $p_c = 0.9$

4 SBX operator index, $\eta_c = 2$

5 polynomial mutation prob., $p_m = 1 / n$

6 polynomial mutation index, $\eta_m = 20$

7 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 51 initial populations are used for all mutation schemes. Mutation schemes are then compared with each other and with the zero mutation scheme.

Three performance measures are used for comparing different schemes. All schemes are coded in C programming language and efforts have been made to make the codes as efficient as possible. Moreover, each code is run on the same computer. Therefore, run time (CPU time of execution of a run, in seconds) is an indicator of how fast an algorithm runs. Total number of generations to reach a particular function value is another indicator of function evaluations (since population size is kept the same for all schemes) needed to solve a problem by a scheme. Finally, total number of mutations needed from the first generation till the end is an indicator of mutations needed to find the target solution. Smaller is each of these numbers for a scheme, the better is the performance of the scheme.

5.1 Ellipsoidal function

This function is unimodal and additively separable having $f_{ell}(\mathbf{x}^*) = 0$:

$$f_{ell}(x) = \sum_{i=1}^{D} i x_i^2. \tag{10}$$

The performance of different mutation schemes on the problem is shown in Table 1. While all mutation schemes perform successfully on all 51 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.

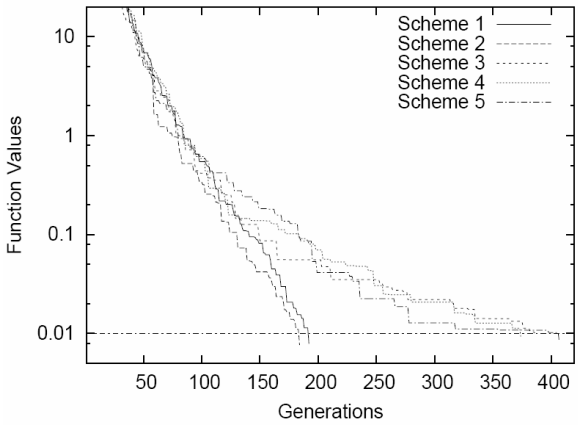
Figure 3 shows how the objective value is reduced with generation for all five mutation schemes. Clearly, Schemes 1 and 2 perform the best. Table 1 shows that despite the use of a clock in Scheme 2, it takes more or less identical number of mutations. Scheme 2 takes about $O(1 / p_m)$ or $O(n)$ less number of random numbers, thereby making it faster than Scheme 1. Thus, it can be concluded that Scheme 2 (mutation clock) is found to be most efficient mutation scheme for the ellipsoidal problem.

Table 1 Performance of different mutation schemes on ellipsoidal problem using polynomial mutation operator

		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

Notes: Scheme 0: no mutation, Scheme 1: usual mutation, Scheme 2: mutation clock, Scheme 3: one mutation per solution, Scheme 4: fixed strategy mutation, and Scheme 5: diversity-based mutation.

Figure 3 Variation of number of function evaluations with generation for ellipsoidal problem using polynomial mutation operator



5.2 Schwefel's function

The function is as follows:

$$f_{sch}(x) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j \right). \tag{11}$$

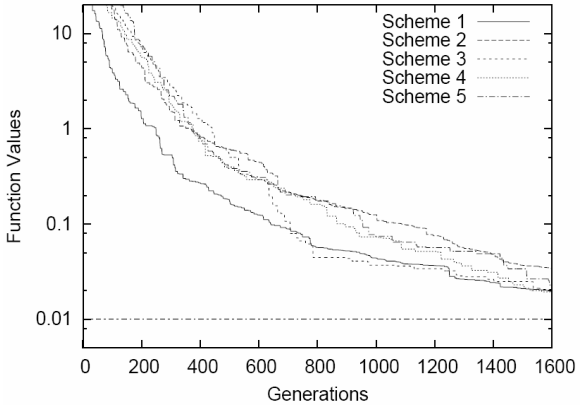
The performance of different mutation schemes on this problem is shown in Table 2. Figure 4 shows the reduction in population-best objective value with generation. All five schemes perform almost equally well for this problem. Table 2 shows that to achieve the same function value, Scheme 5 requires least number of generations and hence least number of mutations compared to other mutation schemes. Since Schwefel's function links all variables together, a variance-based mutation (Scheme 5) helps to identify the less-converged variables to be mutated more often, thereby making the performance of Scheme 5 better. However, the performance of other four mutation schemes are also comparable, although RGAs without any mutation did not perform well.

Table 2 Performance of different mutation schemes on Schwefel's problem using polynomial mutation operator

		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*

Notes: Scheme 0: no mutation, Scheme 1: usual mutation, Scheme 2: mutation clock, Scheme 3: one mutation per solution, Scheme 4: fixed strategy mutation, and Scheme 5: diversity-based mutation.

Figure 4 Variation of number of function evaluations with generation for Schwefel's function using polynomial mutation operator



5.3 Ackley's function

Next, we consider Ackley's function:

$$f_{ack}(x) = 20 + e - 20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right). \tag{12}$$

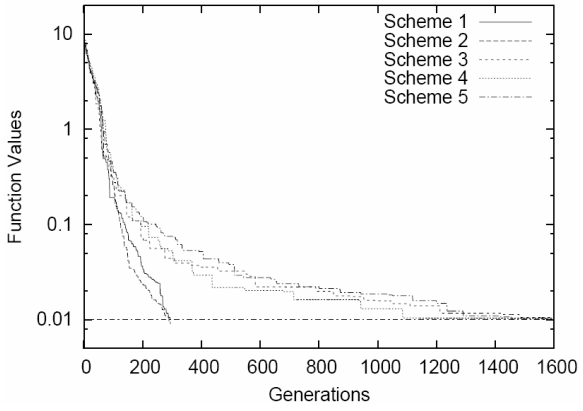
The performance of different mutation schemes on this problem is shown in Table 3. The table shows that Schemes 1 and 2 perform much better than other three mutation schemes. Again, RGAs without any mutation operator do not perform well. Figure 5 shows the variation of objective value with generation. It is clear that Schemes 1 and 2 perform better than other three schemes.

Table 3 Performance of different mutation schemes on Ackley's function using polynomial mutation operator

		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

Notes: Scheme 0: no mutation, Scheme 1: usual mutation, Scheme 2: mutation clock, Scheme 3: one mutation per solution, Scheme 4: fixed strategy mutation, and Scheme 5: diversity-based mutation.

Figure 5 Variation of *f* with generation for Ackley's function using polynomial mutation operator



5.4 Rosenbrock's function

Rosenbrock's function is as follows:

$$f_{ros}(x) = \sum_{i=1}^{D-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(x_1 - 1\right)^2 \right]. \tag{13}$$

Since this is a difficult problem to solve, as reported in the literature (Deb et al., 2012), we use $\epsilon_T = 15$ for this problem. The performance of different mutation schemes on this problem is shown in Table 4. The table shows that Schemes 1 and 2 perform the best. Scheme 2 (mutation clock) takes slightly smaller number of generations and computational time to find a similar solution. Clearly, RGAs without the mutation operator do not perform well, as usual.

Table 4 Performance of different mutation schemes on Rosenbrock's problem using polynomial mutation operator

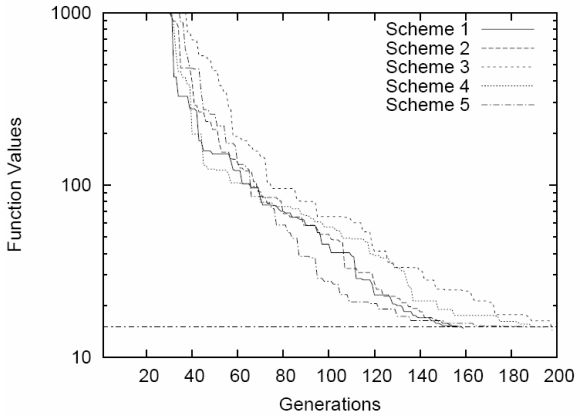
		Scheme 0	*Scheme 1*	*Scheme 2*	*Scheme 3*	*Scheme 4*	*Scheme 5*
Run time	Min.	0.08	*0.05*	*0.05*	*0.05*	*0.05*	0.06
	Avg.	4.63	0.24	*0.14*	0.29	0.27	0.30
	Med.	5.14	0.08	*0.07*	0.09	0.08	0.10
	Max.	5.51	3.09	*0.99*	2.45	2.77	2.21
Generations	Min.	165.00	*102.00*	106.00	110.00	126.00	110.00
	Avg.	9433.04	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

Notes: Scheme 0: no mutation, Scheme 1: usual mutation, Scheme 2: mutation clock, Scheme 3: one mutation per solution, Scheme 4: fixed strategy mutation, and Scheme 5: diversity-based mutation.

Figure 6 reiterates the fact that Scheme 2 performs slightly better than Scheme 1 and other mutation schemes considered in this study.

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

Figure 6 Variation of f with generation for Rosenbrock's function using polynomial mutation operator



6 Parametric study on polynomial mutation

Having established the superiority of mutation clock scheme, we now perform a parametric study of the distribution index η_m and mutation probability p_m of polynomial mutation operator for Scheme 2 (mutation clock) only.

6.1 Parametric study for distribution index

In the previous section, $\eta_m = 20$ was used. Here, all other RGA parameters are kept fixed, except that η_m is changed in different simulations. Fifty-one runs are made for each case.

Figure 7 Parametric study of η_m for Scheme 2 on ellipsoidal function using polynomial mutation operator

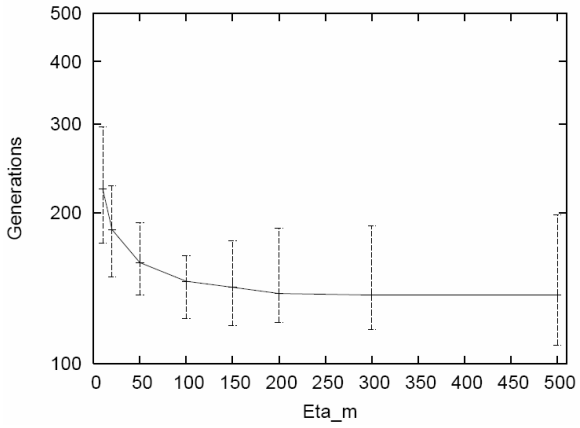


Figure 7 shows that $\eta_m \in [100, 200]$ performs the best on the ellipsoidal problem. For smaller η_m ($<< 100$) values, the perturbance due to mutation is large and hence the respective RGA finds it difficult to find a near-optimal solution quickly. On the other hand, for larger η_m values ($>> 200$), the perturbance is too small to create a near-optimal solution quickly. The values $\eta_m \in [100, 200]$ provide RGAs right perturbance in parent solutions to converge near to the optimal solution. This working range for η_m is wide, thereby demonstrating the robustness of the polynomial mutation operator with the proposed mutation clock scheme.

Figure 8 shows that $\eta_m \in [100, 150]$ produces best results for Schwefel's function, which is a similar observation as that made for the ellipsoidal function. The number in bracket indicates the number of times (out of 51) for which RGAs terminated with $f(\mathbf{x}) \leq \epsilon_T$ condition.

Figure 8 Parametric study of η_m for Scheme 2 on Schwefel's function using polynomial mutation operator

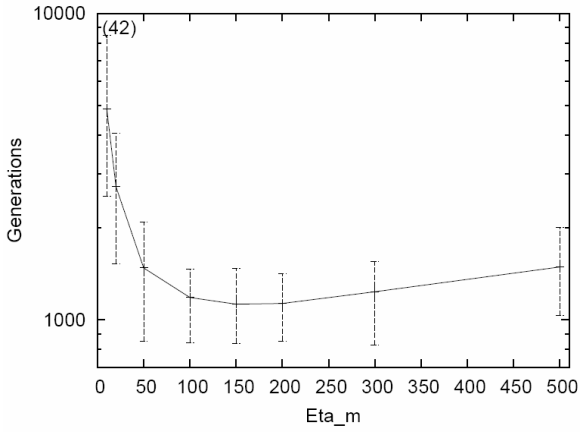


Figure 9 Parametric study of η_m for Scheme 2 on Ackley's function using polynomial mutation operator

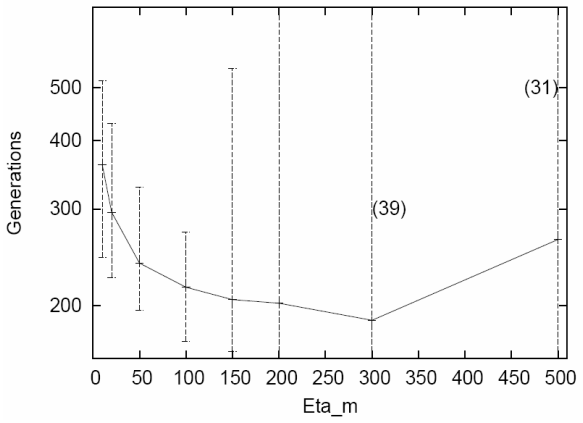
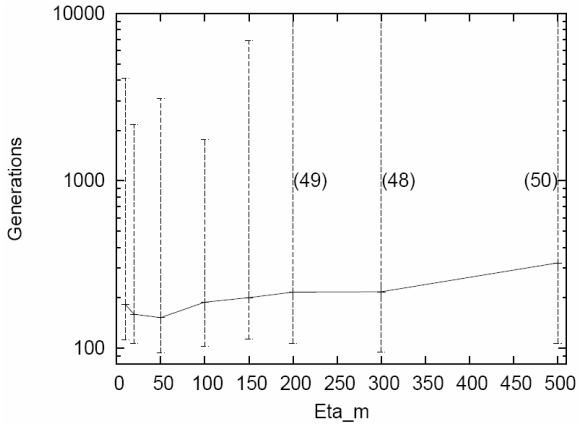


Figure 9 shows $\eta_m \in [100, 150]$ produce best results for Ackley's function. Although for $\eta_m \in [200, 300]$ some runs require smaller number of generations, the variance of the performance over 51 runs is too wide for it to be recommended for practice.

Figure 10 shows $\eta_m \in [20, 100]$ produce best results for Rosenbrock's function.

Figure 10 Parametric study of η_m for Scheme 2 on Rosenbrock's function using polynomial mutation operator



6.2 Parametric study for mutation probability

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 in the previous subsection). 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 RGA parameters are kept the same as before.

Figure 11 Parametric study of p_m for Scheme 2 on the ellipsoidal function using polynomial mutation operator

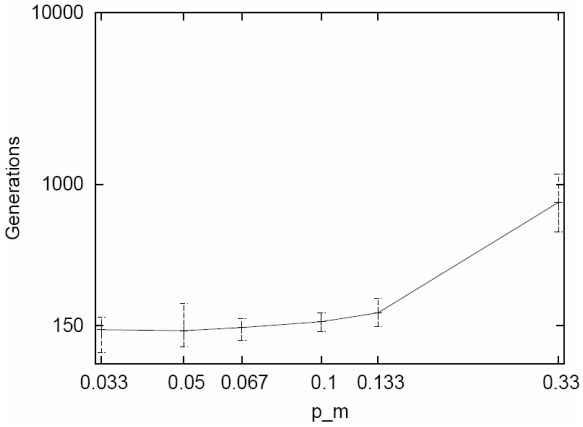
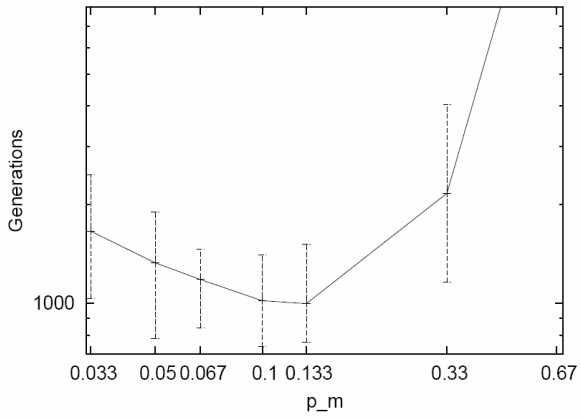


Figure 11 shows the number of generations needed for 51 runs for the ellipsoidal problem. Mutation probabilities of 0.01 / n and 0.1 / n are not found to produce a reasonable result. The figure shows that p_m values of 0.5 / n to 1.5 / n perform the best. Thus, the usual practice of using $p_m = 1 / n$ is justified by our study.

Figure 12 shows that $p_m \in [0.75, 2.0] / n$ performs better for the Schwefel's function. Too low or too high values of p_m cause too little changes or too frequent changes in variables of the population members and hence are not productive.

Figure 12 Parametric study of p_m for Scheme 2 on Schwefel's function using polynomial mutation operator



For Ackley's function, $p_m \in [0.75, 1.5] / n$ performs better, with $p_m = 1 / n$ clearly performing the best. For the Rosenbrock's function, although most p_m values find similar high performance, the least variance in the results come with $p_m = 1 / n$.

Figure 13 Parametric study of p_m for Scheme 2 on the Ackley's function using polynomial mutation operator

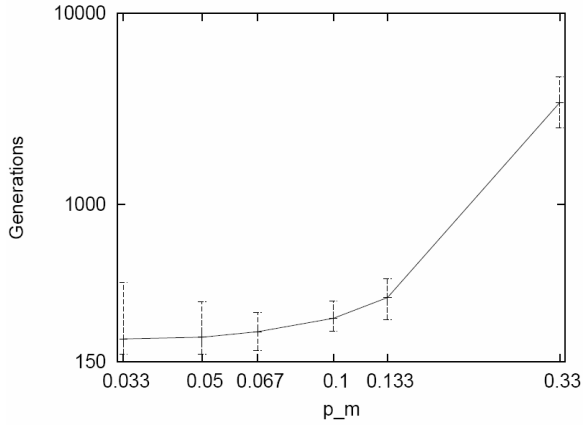
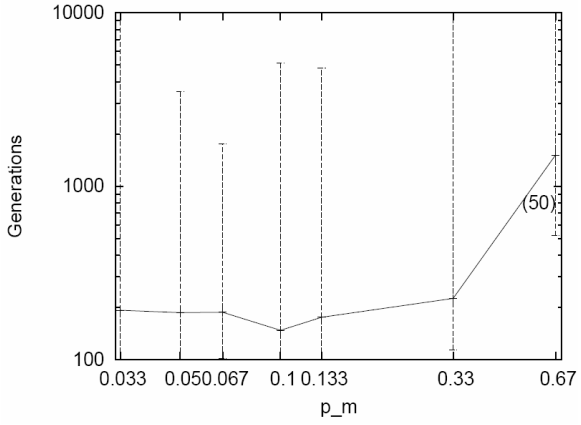


Figure 14 Parametric study of p_m for Scheme 2 on Rosenbrock's function using polynomial mutation operator



Thus, based on the above extensive simulation study, we recommend the following parameter values for the polynomial mutation operator applied with the SBX recombination operator having $\eta_c = 2$ and with the binary tournament selection operator:

- mutation scheme = mutation clock (Goldberg, 1989)
- mutation index, $\eta_m = [100, 150]$ (Deb and Agrawal, 1999)
- mutation probability, $p_m = [0.75, 1.5] / n$, where n is the number of variables.

Keeping in mind the no free lunch (NFL) theorem (Wolpert and Macready, 1997), we do not expect one particular mutation scheme to be most efficient for all problems, but our study on four mostly unimodal and lightly multimodal problems indicate that mutation clock may be a good compromise for solving such problems.

7 Results with Gaussian mutation

We now perform a similar study for the Gaussian mutation operator, which is another commonly-used mutation operator used in real-parameter optimisation. For this mutation, we use the following RGA parameter settings:

1 no. of real variables, $n = 15$

2 population size = 150

3 SBX operator probability, $p_c = 0.9$

4 SBX operator index, $\eta_c = 2$,

5 Gaussian mutation prob. $p_m = 0.067$,

6 Gaussian mutation strength $\sigma = 1/30$

7 termination parameter $\epsilon_T = 0.01$.

Like before, for each mutation scheme, we make 51 different runs starting from different initial populations, however the same set of 51 initial populations are used for all mutation schemes. Mutation schemes are then compared with each other and with the zero mutation scheme using three performance measures.

7.1 Ellipsoidal function

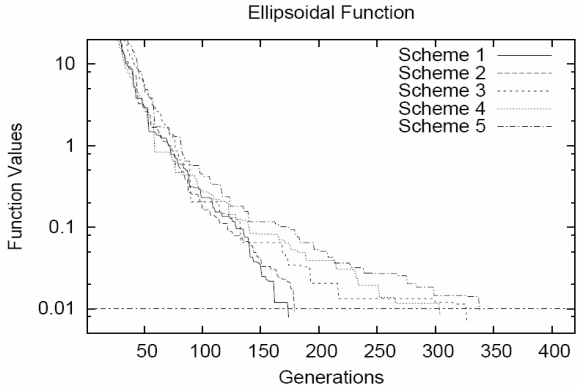
The performance of different mutation schemes on the problem is shown in Table 5. All five mutation schemes perform successfully on 100% runs, however, RGAs without mutation have failed to find the required solution in more than 50% of the runs. This amply suggests the importance of using a mutation scheme in RGAs. Figure 15 shows how the objective function value is reduced with generation for all five mutation schemes.

Table 5 Performance of different mutation schemes on ellipsoidal problem using Gaussian mutation operator

		Scheme 0	*Scheme 1*	*Scheme 2*	*Scheme 3*	*Scheme 4*	*Scheme 5*
Run time	Min.	*0.06*	0.07	*0.06*	0.09	0.08	0.11
	Avg.	3.10	*0.08*	*0.08*	0.14	0.13	0.17
	Med.	3.81	*0.08*	*0.08*	0.14	0.13	0.17
	Max.	4.55	0.10	*0.09*	0.18	0.17	0.24
Generations	Min.	*122.00*	140.00	142.00	215.00	174.00	232.00
	Avg.	7,701.12	*175.00*	178.78	327.96	308.08	344.08
	Med.	10,000.00	*175.00*	180.00	328.00	305.00	339.00
	Max.	10,000.00	222.00	214.00	428.00	405.00	497.00
Mutations	Min.	0.00	*14,054.00*	14,591.00	21,500.00	17,400.00	23,200.00
	Avg.	0.00	*17,573.03*	18,594.12	32,796.08	30,807.84	34,407.84
	Med.	0.00	*17,631.00*	18,714.0	32,800.00	30,500.00	33,900.0
	Max.	0.00	22,377.00	*22,113.00*	42,800.00	40,500.00	49,700.00

Notes: Scheme 0: no mutation, Scheme 1: usual mutation, Scheme 2: mutation clock, Scheme 3: one mutation per solution, Scheme 4: fixed strategy mutation, and Scheme 5: diversity-based mutation.

Figure 15 Variation of number of function evaluations with generation for ellipsoidal problem using Gaussian mutation operator



Clearly, Schemes 1 and 2 perform the better than the other three schemes.

7.2 Schwefel's function

The performance of different mutation schemes on this problem is shown in Table 6. The performance of all five mutation schemes are similar, with Scheme 3 having a slight edge.

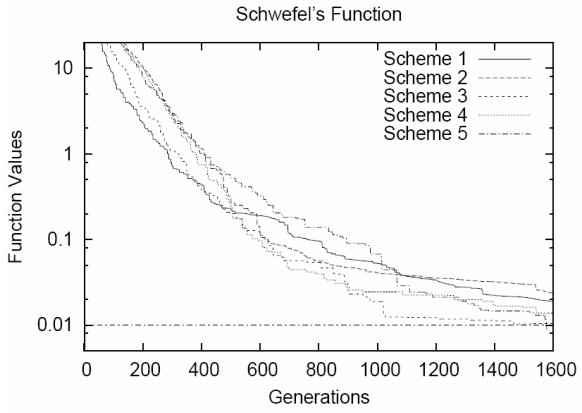
Table 6 Performance of different mutation schemes on Schwefel's problem using Gaussian mutation operator

		Scheme 0	*Scheme 1*	*Scheme 2*	*Scheme 3*	*Scheme 4*	*Scheme 5*
Run time	Min.	4.05	0.51	0.42	*0.40*	0.52	0.49
	Avg.	4.60	1.06	1.03	*0.75*	0.80	0.88
	Med.	4.73	1.00	1.01	0.75	0.81	0.86
	Max.	5.25	1.75	2.23	1.49	*1.26*	1.46
Generations	Min.	10,000.00	983.00	*861.00*	872.00	1,107.00	904.00
	Avg.	10,000.00	2046.02	2,147.80	1,622.35	1,717.68	*1,597.78*
	Med.	10,000.00	1,940.00	2,094.00	1,633.00	1,742.00	*1,578.0*
	Max.	10,000.00	3,343.00	4,700.00	3,217.00	*2,593.00*	2,672.00
Mutations	Min.	0.00	98,937.00	89,239.00	*87,200.00*	110,700.00	90,400.00
	Avg.	0.00	205,714.96	223,275.14	162,235.29	171,768.63	*159,778.43*
	Med.	0.00	195,130.00	217,533.00	163,300.00	174,200.00	*157,800.00*
	Max.	0.00	336,173.00	488,678.00	321,700.00	*259,300.00*	267,200.00

Notes: Scheme 0: no mutation, Scheme 1: usual mutation, Scheme 2: mutation clock, Scheme 3: one mutation per solution, Scheme 4: fixed strategy mutation, and Scheme 5: diversity-based mutation.

Figure 16 shows the reduction in objective function value with generation. All five schemes perform almost equally well on this problem.

Figure 16 Variation of number of function evaluations with generation for Schwefel's function using Gaussian mutation operator



7.3 Ackley's function

The performance of different mutation schemes on this problem is shown in Table 7. It is clear from the table that Schemes 1 and 2 are almost an order of magnitude better than the rest of the schemes.

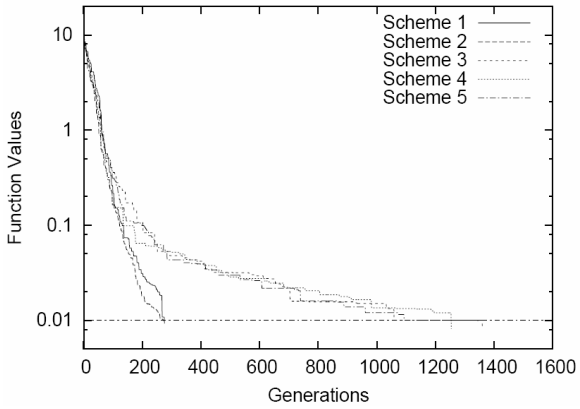
Table 7 Performance of different mutation schemes on Ackley's function using Gaussian mutation operator

		Scheme 0	*Scheme 1*	*Scheme 2*	*Scheme 3*	*Scheme 4*	*Scheme 5*
Run time	Min.	*0.10*	*0.10*	*0.10*	0.31	0.37	0.58
	Avg.	4.18	0.14	*0.13*	0.64	0.63	0.90
	Med.	4.55	0.14	*0.13*	0.64	0.61	0.90
	Max.	5.12	0.19	0.20	0.91	0.95	1.35
Generations	Min.	207.00	*201.00*	202.00	690.00	807.00	831.00
	Avg.	9,427.43	*275.55*	278.21	1,375.31	1,301.33	1,403.84
	Med.	10,000.00	277.00	*274.00*	1,361.00	1,255.00	1,346.00
	Max.	10,000.00	*366.00*	415.00	2,009.00	2,089.00	2,131.00
Mutations	Min.	0.00	*20,258.00*	21,079.00	69,000.00	80,700.00	83,100.00
	Avg.	0.00	*27,701.18*	28,928.57	137,531.37	130,133.33	1403,84.31
	Med.	0.00	*27,982.00*	28,479.00	136,100.00	125,500.00	134,600.00
	Max.	0.00	*36,914.00*	43,290.00	200,900.00	208,900.00	213,100.00

Notes: Scheme 0: no mutation, Scheme 1: usual mutation, Scheme 2: mutation clock, Scheme 3: one mutation per solution, Scheme 4: fixed strategy mutation, and Scheme 5: diversity-based mutation.

Figure 17 shows the variation of objective function value with generation. It is also clear from the figure that Schemes 1 and 2 perform better than other three schemes.

Figure 17 Variation of f with generation for Ackley's function using Gaussian mutation operator



7.4 Rosenbrock's function

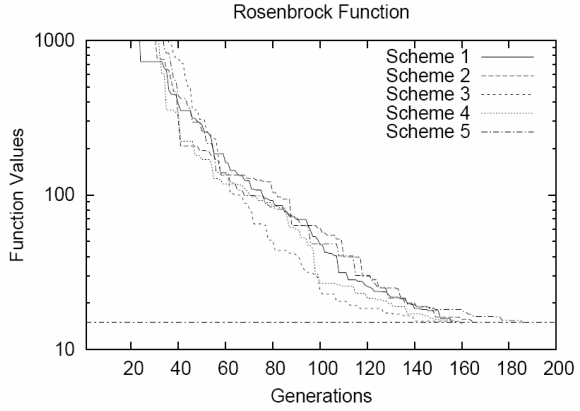
Like before, we use ϵ_T = 15. The performance of different mutation schemes on this problem is shown in Table 8. Figure 18 indicates that all schemes perform more or less in a similar manner.

Table 8 Performance of different mutation schemes on Rosenbrock's problem using Gaussian mutation operator

		Scheme 0	*Scheme 1*	*Scheme 2*	*Scheme 3*	*Scheme 4*	*Scheme 5*
Run time	Min.	0.08	0.05	0.05	0.05	*0.04*	0.06
	Avg.	4.23	1.64	0.20	*0.17*	0.21	0.32
	Med.	4.58	0.08	0.08	0.08	*0.07*	0.10
	Max.	5.07	0.23	1.76	1.35	2.11	5.40
Generations	Min.	165.00	89.00	95.00	104.00	*86.00*	122.00
	Avg.	9433.04	451.76	487.02	*360.11*	462.96	585.80
	Med.	10,000.00	*157.00*	166.00	161.00	159.00	187.00
	Max.	10,000.00	3,181.00	3,708.00	*2,918.00*	4,579.00	10,000.00
Mutations	Min.	0.00	8,839.00	9,823.00	10,400.00	*8,600.00*	12,200.00
	Avg.	0.00	45,438.71	50,614.08	*36,011.76*	46,296.08	58,580.39
	Med.	0.00	15,853.00	17,315.00	16,100.00	*15,900.00*	18,700.00
	Max.	0.00	319,117.00	385,371.00	291,800.00	457,900.00	*100,000.00*

Notes: Scheme 0: no mutation, Scheme 1: usual mutation, Scheme 2: mutation clock, Scheme 3: one mutation per solution, Scheme 4: fixed strategy mutation, and Scheme 5: diversity-based mutation.

Figure 18 Variation of f with generation for Rosenbrock's function using Gaussian mutation operator



On four different problems, it is observed that in terms of function evaluations, Schemes 1 and 2 perform better than or similar to other three mutation schemes. All proposed mutation schemes are better than RGAs without any mutation. Tables show that Scheme 2 requires much smaller computational time compared to Scheme 1. Hence, we

recommend the use of the mutation clock operator as an efficient mutation scheme for real parameter GAs with the Gaussian mutation operator as well.

8 Parametric studies on Gaussian mutation operator

We now perform a parametric study of the mutation strength σ and mutation probability p_m for Scheme 2 (mutation clock) alone using the Gaussian mutation. This is due to the fact that mutation clock performed better, in general, on the chosen test problems.

8.1 Parametric study for mutation strength

Figure 19 shows that $\sigma \approx 0.25/15 = 1/60$ performs the best on the ellipsoidal problem. As the mutation strength is increased, the performance gets worse.

Figure 19 Parametric study of σ for Scheme 2 on ellipsoidal function using Gaussian mutation operator

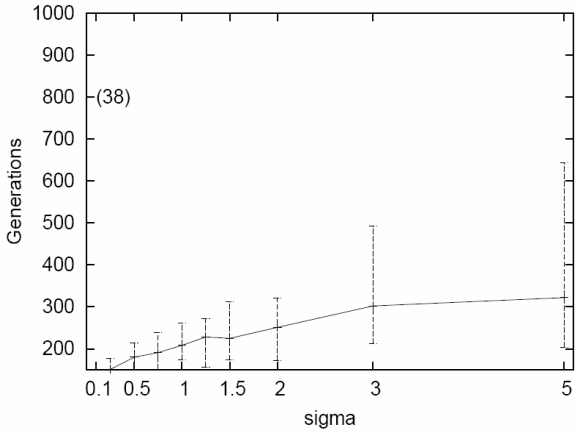


Figure 20 Parametric study of σ for Scheme 2 on Schwefel's function using Gaussian mutation operator

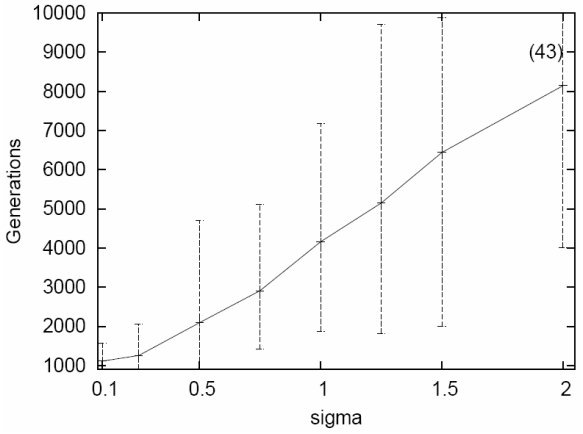


Figure 21 Parametric study of σ for Scheme 2 on Ackley's function using Gaussian mutation operator

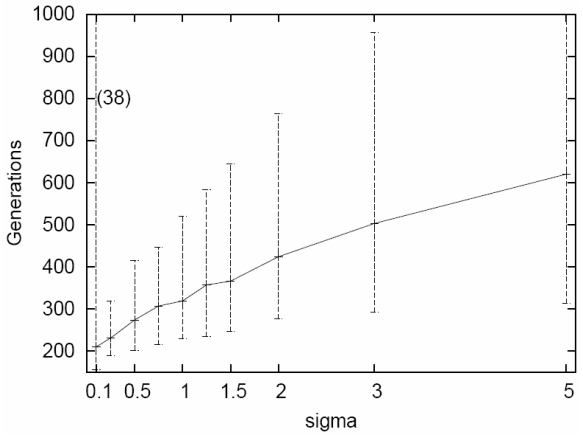


Figure 22 Parametric study of σ for Scheme 2 on Rosenbrock's function using Gaussian mutation operator

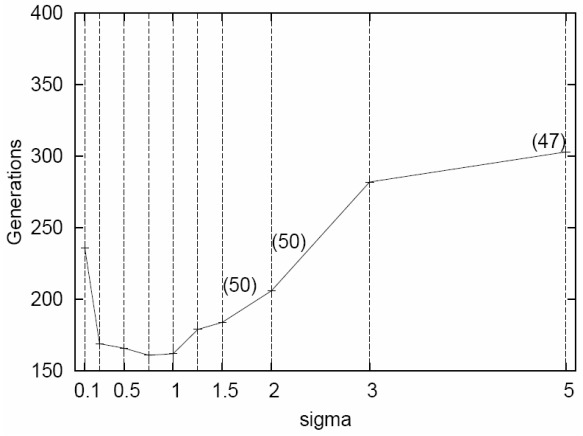


Figure 20 shows $\sigma \approx 0.1/15 = 1/150$ produce best results for Schwefel's function. Figure 21 shows $\sigma \approx 0.25/15 = 1/60$ produce best results for the Ackley's function. A smaller value makes 38 out of 51 runs successful. Figure 22 shows although median performance is better for $\sigma \in [0.25, 1.0]/15$, there is a large variance in the performance over 51 runs for the Rosenbrock's function. Nevertheless, the above simulations suggest that $\sigma = 0.25/15 = 1/60$ performs better than other values.

8.2 Parametric study with mutation probability

Next, we perform a parametric study of the mutation probability p_m with Scheme 2 and fix the mutation index as $\sigma = 1/60$ (which is found to be near-optimal in the previous subsection). We use $p_m = k/n$ with $k = 0.1$, 0.5, 0.75, 1, 1.5, 2, and 5. The rest of the RGA parameters are kept the same as before.

Figure 23 Parametric study of p_m for Scheme 2 on the ellipsoidal function using Gaussian mutation operator

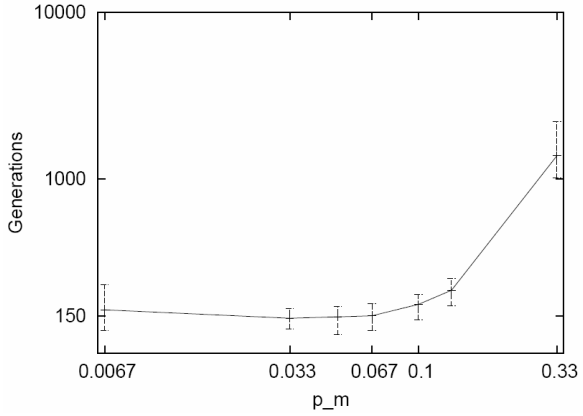


Figure 24 Parametric study of p_m for Scheme 2 on Schwefel's function using Gaussian mutation operator

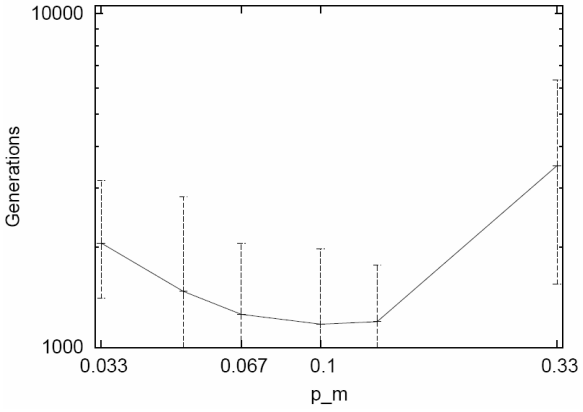


Figure 25 Parametric study of p_m for Scheme 2 on the Ackley's function using Gaussian mutation operator

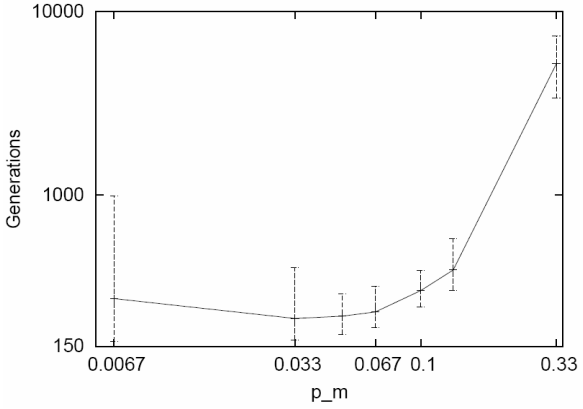


Figure 26 Parametric study of p_m for Scheme 2 on Rosenbrock's function using Gaussian mutation operator

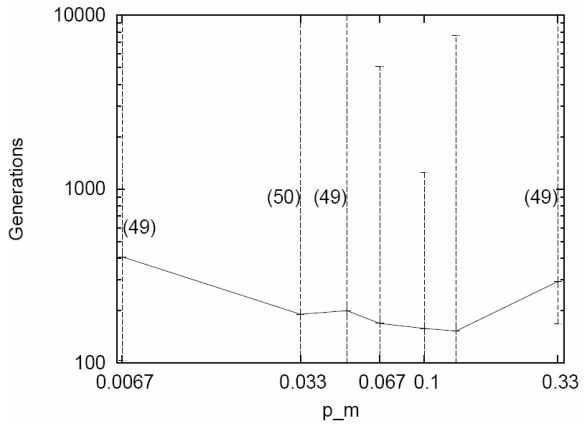


Figure 23 shows the number of generations needed for 51 runs for the ellipsoidal problem. Mutation probabilities in the range [0.5, 1.5] / n are found to produce the best performance. On Schwefel's function (Figure 24), $p_m \in [0.75, 2.0] / n$ is found to produce better results. For Ackley's function (Figure 25), $p_m \in [0.5, 1.0] / n$ produces the best and for Rosenbrock's function (Figure 26), $p_m \in [0.75, 2.0] / n$ produces the best performance.

Thus, based on the above simulation results, we recommend the following parameter values for the Gaussian mutation operator applied with the SBX recombination operator having $\eta_c = 2$ and with the binary tournament selection operator:

- mutation scheme = mutation clock (Goldberg, 1989)
- standard deviation, $\sigma = [1/60, 1/30]$
- mutation probability, $p_m = [0.75, 1.0] / n$, where n is the number of variables.

8.3 Comparison of polynomial and Gaussian mutation operators

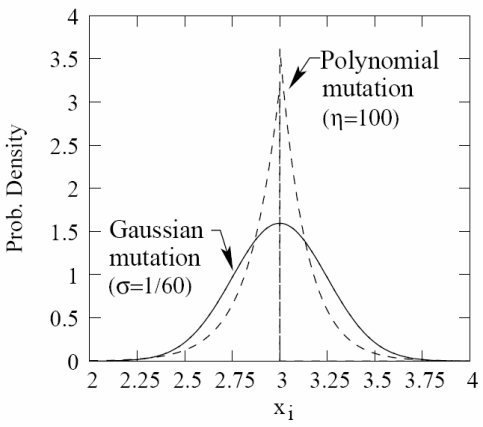
Independent parametric studies revealed that for all other parameters being equal, $\eta_m = 100$ for polynomial mutation and $\sigma = 0.25/15 = 1/60$ for Gaussian mutation perform the best. We tabulate the performance of RGAs with Scheme 2 (mutation clock) for these two parameter settings in Table 9 with $p_m = 1 / n$. It is clear that the performance of these two mutation operators with respective parameter settings are similar, except in the number of overall number of mutations for Rosenbrock's function. Compared to the wide range of these performance metric values observed with other mutation schemes and parameter values, as shown in earlier tables of this paper, the difference in the performance metric values between the two mutation operators for their respective parameter values is small.

Table 9 Performance of polynomial mutation (with $\eta_m = 100$) and Gaussian mutation (with $\sigma = 1/60$) on all four problems

		Mut. op.	*Ellipsoidal*	*Schwefel*	*Ackley*	*Rosenbrock*
Run time (s)	Min.	Poly.	0.050	0.398	0.079	0.047
		Gauss.	0.050	0.339	0.089	0.046
	Avg.	Poly.	0.059	0.556	0.100	0.172
		Gauss.	0.064	0.611	0.148	0.235
	Med.	Poly.	0.059	0.553	0.099	0.087
		Gauss.	0.063	0.587	0.108	0.081
	Max.	Poly.	0.067	0.688	0.125	0.815
		Gauss.	0.074	0.951	0.110	2.407
Generations	Min.	Poly.	123.00	843.00	172.00	102.00
		Gauss.	124.00	726.00	189.00	96.00
	Avg.	Poly.	144.74	1,184.25	272.00	373.59
		Gauss.	153.22	1,312.14	235.80	491.63
	Med.	Poly.	146.00	1,182.00	216.00	188.00
		Gauss.	151.00	1,257.00	231.00	169.00
	Max.	Poly.	164.00	1,466.00	274.00	1,760.00
		Gauss.	178.00	2,051.00	319.00	5,076.00
Mutations	Min.	Poly.	12,799.00	87,651.00	18,005.00	10,451.00
		Gauss.	12,876.00	75,713.00	19,840.00	9,932.00
	Avg.	Poly.	15,054.80	123,105.71	22,634.31	38,818.88
		Gauss.	15,935.00	136,389.57	24,516.37	51,127.71
	Med.	Poly.	15,173.00	122,801.00	22,415.00	19,773.00
		Gauss.	15,681.00	130,351.00	23,992.00	17,430.00
	Max.	Poly.	17,192.00	152,027.00	28,127.00	182,379.00
		Gauss.	18,687.00	212,520.00	33,100.00	527,745.00

To investigate why these two parameter settings cause a similar performance of the two mutation operators, we plot the probability density function of two mutation events on a parent $x_i = 3.0$ with bounds [–5, 10]. The polynomial mutation distribution is used with $\eta_m = 100$ and the Gaussian mutation is used with $\sigma = 1/60$. Figure 27 shows that both have a similar probability distribution of creating offspring solutions around the parent. It is interesting that an inherent equivalent probability distribution for creating offspring solutions make both mutation operators to exhibit a similar performance. This amply reveals the fact that instead of the detail of implementation of this or that mutation operator being important as often reported in the literature, it is the inherent probability of offspring creation that dictates the efficacy of a mutation operator.

Figure 27 Comparison of polynomial and Gaussian mutation for a parent $x_i = 3.0$ in [–5, 10]



9 Conclusions

Mutation operators are used for maintaining diversity in a GA. In this paper, we have suggested five different mutation schemes for RGAs and tested their performance with two commonly-used mutation operators – polynomial mutation and Gaussian mutation – on four different test problems. Based on their performance, we conclude the following:

- any of the five mutation operators is better than not performing any mutation at all
- the mutation clock operator, which was suggested in '80s, has been long ignored and has been found here to be the fastest in terms of its execution time and better in terms of its performance
- the above conclusions are valid for both polynomial and Gaussian mutation operators.

Parametric studies have also found a range of mutation parameters for both mutation operators. For the polynomial mutation, the distribution index $\eta_m \in [100, 150]$ and mutation probability $p_m \in [0.5, 1.5] / n$ (where n is the number of variables) have been found to perform the best. For Gaussian mutation operator, its mutation strength $\sigma \in [0.25, 0.5] / (b_i - a_i)$ (where a_i and b_i are the lower and upper bounds of i^{th} variable) and mutation probability $p_m \in [0.75, 1.0] / n$ have been found to perform better.

This study is extensive in performing simulation studies with two different mutation operators which are analysed with five different mutation schemes. Superior performance of mutation clock scheme with both mutation operators provide us confidence to recommend this scheme in practice. Moreover, a more or less similar optimal mutation probability of around $1 / n$ observed for both mutation operators also gives us confidence of its usage in other problems. Interestingly, the inherent probability distributions of creating offspring for polynomial and Gaussian mutation operators have been found to be similar and instrumental in the superior performance of both methods with the mutation clock scheme. Furthermore, we believe that this study should resurrect the use of mutation clock in RGA studies and encourage its use in the coming years.

Acknowledgements

The first author acknowledges the Academy of Finland Grant No. 133387.

References

Deb, K. (2001) *Multi-objective Optimization using Evolutionary Algorithms*, Wiley, Chichester, UK.

Deb, K. and Agrawal, R.B. (1995) 'Simulated binary crossover for continuous search space', *Complex Systems*, Vol. 9, No. 2, pp.115–148.

Deb, K. and Agrawal, S. (1999) 'A niched-penalty approach for constraint handling in genetic algorithms', *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA-99)*, Springer-Verlag, pp.235–243.

Deb, K., Dhebar, Y. and Pavan, N.V.R. (2012) *Non-uniform Mapping in Binary-coded Genetic Algorithms*, Technical Report KanGAL Report No. 2012011, IIT Kanpur, India.

Eshelman, L.J. and Schaffer, J.D. (1993) 'Real-coded genetic algorithms and interval-schemata', *Foundations of Genetic Algorithms 2 (FOGA-2)*, pp.187–202.

Goldberg, D. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA.

Higuchi, T., Tsutsui, S. and Yamamura, M. (2000) 'Theoretical analysis of simplex crossover for real-coded genetic algorithms', *Parallel Problem Solving from Nature (PPSN-VI)*, pp.365–374.

Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*, MIT Press, Ann Arbor, MI.

Kita, H., Ono, I. and Kobayashi, S. (1999) 'Multi-parental extension of the unimodal normal distribution crossover for real-coded genetic algorithms', *Proceedings of the 1999 Congress on Evolutionary Computation*, pp.1581–1587.

Michalewicz, Z. (1992) *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin.

Schaffer, J.D., Caruana, R.A., Eshelman, L.J. and Das, R. (1989) 'A study of control parameters affecting online performance of genetic algorithms', *Proceedings of the International Conference on Genetic Algorithms*, pp.51–60.

Schwefel, H-P. (1987) 'Collective phenomena in evolutionary systems', in Checkland, P. and Kiss, I. (Eds.): *Problems of Constancy and Change – the Complementarity of Systems Approaches to Complexity*, pp.1025–1033, International Society for General System Research, Budapest.

Voigt, H-M., Mühlenbein, H. and Cvetković, D. (1995) 'Fuzzy recombination for the breeder genetic algorithm', *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp.104–111.

Wolpert, D.H. and Macready, W.G. (1997) 'No free lunch theorems for optimization', *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, pp.67–82.