A Comparative Study of Adaptive Mutation Operators for Genetic Algorithms

Imtiaz Korejo, Shengxiang Yang, and ChangheLi

Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, UK
(e-mail: {iak5, s.yang, cl160}@mcs.le.ac.uk)

Abstract

Genetic algorithms (GAs) are a class of stochastic optimization methods inspired by the principles of natural evolution. Adaptation of strategy parameters and genetic operators has become an important and promising research area in GAs. Many researchers are applying adaptive techniques to guide the search of GAs toward optimum solutions. Mutation is a key component of GAs. It is a variation operator to create diversity for GAs. This paper investigates several adaptive mutation operators, including population level adaptive mutation operators and gene level adaptive mutation operators, for GAs and compares their performance based on a set of uni-modal and multi-modal benchmark problems. The experimental results show that the gene level adaptive mutation operators are usually more efficient than the population level adaptive mutation operators for GAs.

1 Introduction

Genetic algorithms (GAs) are powerful search methods. They are inspired by the Darwin's theory of survival of the fittest. GAs were first introduced by John Holland in 1960s in USA. Nowadays, GAs have been successfully applied for solving many optimization problems due to the properties of easy-to-use and robustness for finding good solutions to difficult problems [6]. The efficiency of GAs depends on many parameters, such as the initial population, the representation of individuals, the selection strategy, and the recombination (crossover and mutation) operators. Mutation is used to maintain the diversity of the entire population by changing individuals bit by bit with a small probability $pm \in [0, 1]$. Usually, the mutation probability has a significant effect on the performance of GAs.

Many researchers have suggested different static mutation probabilities for GAs. These static mutation probabilities are derived from experience or by trial-and-error. De Jong proposed $pm = 0.001$ in [9]. Grefenstette proposed $pm = 0.01$ [7]. According to Schaffer, pm should be set to $[0.001, 0.005]$ [13]. In [2], Bäck suggested $pm = 1.75/(N * L^{1/2})$, where N means the population size and L denotes the length of individuals. This equation is based on Schaffer's results [13]. In [11], it is suggested that $pm = 1/L$ should be generally "optimal". It is very difficult, though not impossible, to find an appropriate parameter setting for pm for the optimal performance.

The operator adaptation techniques in GAs can be classified into three categories, i.e., population level, individual level, and component level adaptation [1]. Operator adaptation depends on how operators are updated. At the population level, parameters are adapted globally by using the feedback information from the current population. Individual level adaptation changes parameters for each individual in the population. Component level adaptation is done separately on some components or genes of an individual in the population.

This paper focuses on the comparative analysis of different population-level and gene-level adaptive mutation operators for GAs based on a set of benchmark optimization problems. The experimental results show that the performance of different adaptive mutation operators depends on the test problem and that the gene level adaptive mutation operators are usually more efficient than the population level adaptive mutation operators for GAs.

The rest of this paper is organized as follows. Section 2 briefly reviews the population level adaptation and gene level adaptation mutation operators in the literature. Section 3 presents the experimental study of comparing the performance of several GAs with different gene level and population level adaptive mutation operators and a particle swarm optimization (PSO) algorithm with a population level adaptive mutation operator. Finally, some conclusions are given in Section 4.

2 Adaptation in Mutation Operators

Adaptation of strategy parameters and genetic operators has become an important and promising area of research on GAs. Many researchers are focusing on solving optimization problems by using adaptive techniques, e.g., probability matching, adaptive pursuit method, numerical optimization, and graph coloring algorithms [16, 17, 12]. The value of parameters and genetic operators are adjusted in GAs. Parameter setting and adaptation in mutation was first introduced in evolutionary strategies [13]. The classification of parameter settings has been introduced differently by the researchers [5, 4, 15].

Basically, there are two main type of parameter settings: parameter tuning and parameter control. Parameter tuning means to set the suitable parameters before the run of algorithms and the parameters remain constant during the execution of algorithms. Parameter control means to assign initial values to parameters and then these values adaptively change during the execution of algorithms. According to [5], parameters are adapted according to one of three methods: deterministic adaptation adjusts the values of parameters according to some deterministic rule without using any feedback information from the search space; adaptive adaptation modifies the parameters using the feedback information from the search space; and self-adaptive adaptation adapts the parameters by the GA itself.

There are two main groups of adaptive mutation operators, one group are the population-level adaptive mutation (PLAM) operators and the other are the gene-level adaptive mutation (GLAM) operators.

2.1 Population-Level Adaptive Mutation Operators

In [10], we designed a mutation operator that can adaptively select the most suitable mutation operator for particle swarm optimization (PSO) for a specific problem. It is difficult to find the

best result by using only a single mutation operator, so various mutation operators may be used at different levels on a single problem to achieve the best result. The PSO proposed in [10] uses a population-level adaptive mutation operator, which will be denoted PLAM_PSO in this paper.

PLAM_PSO uses three mutation operators, the Cauchy mutation operator, the Gaussian mutation operator, and the Levy mutation operator. All mutation operators have an equal initial selection ratio of 1/3. Each mutation operator is applied according to its selection ratio and its offspring fitness is evaluated. Gradually, the most suitable mutation operator will be chosen automatically and control all the mutation behavior in the whole PSO. In order to explain an updating equation for the adaptive mutation operator in PSO, $prog_i(t)$ of operator i at generation t is defined as follows:

$$prog_i(t) = \sum_{j=1}^{M_i} f(p_j^i(t)) - \min\left(f(p_j^i(t)), f(c_j^i(t))\right), \tag{1}$$

where $p_j^i(t)$ and $c_j^i(t)$ denote a parent and its child produced by mutation operator i at generation t respectively, and M_i is the number of particles that select mutation operator i to mutate.

The reward value $reward_i(t)$ of operator i at generation t is defined as follows:

$$reward_i(t) = exp(\frac{prog_i(t)}{\sum_{j=1}^{N} prog_j(t)}\alpha + \frac{s_i}{M_i}(1-\alpha)) + \ c_i p_i(t) - 1 \tag{2}$$

where s_i is the number of particles whose children have a better fitness than themselves after being mutated by mutation operator i, $p_i(t)$ is the selection ratio of mutation operator i at generation t, α is a random weight between $(0, 1)$, N is the number of mutation operators, and c_i is a penalty factor for mutation operator i, which is defined as follows:

$$c_i = \begin{cases} 0.9, & \text{if } s_i = 0 \text{ and } p_i(t) = \max_{j=1}^{N}\left(p_j(t)\right) \\ 1, & \text{otherwise} \end{cases} \tag{3}$$

If the previous best operator has no contribution at the current generation, then the selection ratio of the current best operator will decrease.

With the above definitions, the selection ratio of mutation operator i is updated according to the following equation:

$$p_i(t+1) = \frac{reward_i(t)}{\sum_{j=1}^{N} reward_j(t)}(1 - N * \gamma) + \gamma, \tag{4}$$

where γ is the minimum selection ratio for each mutation operator, which is set 0.01 for all the experiments in this paper. This selection ratio update equation considers four factors: the progress value, the ratio of successful mutations, previous selection ratio, and the minimum selection ratio. Another important parameter for the adaptive mutation operator is the frequency of updating the selection ratios of mutation operators. That is, the selection ratio of each mutation operator can be updated at a fixed frequency, e.g., every U_f generations, instead of every generation.

In [8], a GA with a population based adaptive mutation operator, denoted PLAM_GA in this paper, was proposed. This algorithm uses four mutation operators (M1–M4). The M1 operator inverts the bit value 0 to 1 and 1 to 0, the M2 operator swaps any two bits in a single individual, the third one reverses the interval order of bits in an individual, and the last one (M4) just changes one

bit in an individual. These four mutation operators are used adaptively in the GA. All mutation ratios of these operators are assigned initial values, e.g, 0.1. Each mutation operator is applied by its mutation ratio. After the mutation operation,the progress value is calculated using the following equation

$$progress_i(t) = \sum_{j=1}^{M_i} \max\left(f(p_j^i(t)), f(c_j^i(t)) - f(p_j^i(t))\right) \tag{5}$$

where $progress_i(t)$ is the progress value of operator i at generation t, f is the fitness of an individual, $p_j^i(t)$ and $c_j^i(t)$ are the parent and its offspring produced by mutation operator i at generation(t), and M_i represent the total number of individuals that select the mutation operator i to mutate. The mutation ratio of operator i is updated according to their average progress value at generation(t), according to the following equation:

$$p_i(t+1) = \frac{progress_i(t)}{\sum_{j=1}^{N} progress_j(t)}(P_{mutation} - N * \delta) + \delta \tag{6}$$

where $p_i(t)$ is the mutation ratio of mutation operator i at generation t, N is the total number of mutation operators, $\delta = 0.01$ is the minimum mutation ratio for each mutation operator and $P_{mutation}$ means the initial mutation probability. The key idea behind PLAM is to apply more than one mutation operator on different stages to achieve the best result for a specific problem, at the same time the mutation ratio of the operator is updated by using the above formula.

2.2 Gene-Level Adaptation Mutation Operators

In [20], a statistics-based adaptive non-uniform mutation (SANUM) was proposed for GAs, which is a gene level adaptive mutation operator. SANUM calculates the frequency of ones for each locus in the current population to adapt the mutation probability for that locus during the execution of the GA. If the amount of ones in alleles for a gene locus is increased (or decreased) over the population, that gene locus is called 1-inclined (or 0-inclined). A gene locus is called non-inclined if there is no trend of increasing or decreasing of 1's in the gene locus. The probability of mutation for each locus i at generation t is adjusted by using following equation.

$$pm(i,t) = P_{max} - 2 * |f_1(i,t) - 0.5| * (P_{max} - P_{min}) \tag{7}$$

where $f_1(i,t)$ represent the frequency of 1's in the locus i over the population at generation t, $|x|$ returns the absolute value of x, P_{max} and P_{min} are the maximum and minimum value of the mutation probability for a locus.

In paper [19], the authors used an unparallel adaptive technique on each locus of a chromosome, called Gene Based Adaptive Mutation (GBAM). In GBAM, each gene locus has two different mutation probabilities: pm^1 is used for those loci that have the value of 1 and pm^0 is used for those loci that have the value of 0. Initially, all mutation probabilities are assigned to a value, e.g, 0.02. The probabilities of pm^1 and pm^0 are automatically updated based on the feedback information from the search space, according to the relative success or failure of those chromosomes having a "1" or "0" at that locus for each generation. The new mutation probability for each locus i at generation $t+1$ is updated using the following equations in the case of a maximization problem.

$$pm^0(i,t+1) = \begin{cases} pm^0(i,t) + \gamma, & \text{if } G_{avg}^1(i,t) > P_{avg} \\ pm^0(i,t) - \gamma, & \text{otherwise} \end{cases} \tag{8}$$

$$pm^1(i,t+1) = \begin{cases} pm^1(i,t) - \gamma, & \text{if } G^1_{avg}(i,t) > P_{avg} \\ pm^1(i,t) + \gamma, & \text{otherwise} \end{cases} \quad (9)$$

where γ is the updated value for the mutation rate, $G^1_{avg}(i,t)$ is the average fitness of individuals with allele "1" for locus i at generation t, and $P_{avg}(t)$ is the average fitness of the population at generation t. The above update mechanism is used for each locus separately.

Another gene based adaptive mutation method, called GBAM_FAD, was proposed by Yang and Uyar [21]. This method constructs probabilities of each gene locus with the combination information of fitness and allele distribution. GBAM_FAD also uses two different mutation probabilities for each gene locus, just as in GBAM. The probabilities of each gene locus are adaptively updated based on the correlated feedback information from the search process, according to the relative success or failure of individuals. The new mutation probabilities for each locus i at generation $t+1$ are updated using the following equations in the case of maximization problems.

$$pm^0(i,t+1) = pm^0(i,t) + \gamma * sgn((G^1_{avg}(i,t) - P_{avg}(t))(f_1(i,t) - 0.5)) \quad (10)$$

$$pm^1(i,t+1) = pm^1(i,t) - \gamma * sgn((G^1_{avg}(i,t) - P_{avg}(t))(f_1(i,t) - 0.5)) \quad (11)$$

where $G^1_{avg}(i,t)$ and $P_{avg}(t)$ have the same meaning as in Eqs. (8) and (9), $f_1(i,t)$ is calculated frequency of ones in the alleles in the locus i over the population at generation t and the method $sgn(x)$ returns the value 1, 0, or -1 if $x > 0$, $x = 0$, and $x < 0$, respectively. The GBAM_FAD algorithm efficiently solves deception problems.

The aforementioned GLAM operators have already been investigated on simple uni-modal functions (OneMax and Royal Road), multi-modal functions (Deceptive and 4-Peak problems), and random L-SAT functions [19, 20, 21]. The PLAM operators have also been investigated on various multi-dimensional problems [8, 10]. These operators are implemented on different benchmark optimization problems. It is very difficult to say that which operator is more suitable for which problem. In order to better understand these operators, we compare their performance on a set of benchmark problems in this paper. The experimental study is described below.

3 Experimental Study

3.1 Design of Experiments

Experiments were carried out to compare the performance of several GAs with adaptive mutation operators. They are the PLAM_GA and the three GAs with SANUM, GBAM, and GBAM_FAD respectively, which are described in Section 2. We also tested the PLAM_PSO algorithm. The experiments were conducted on 12 different benchmark optimization functions with two or more dimensions. Some of these functions are unimodal and some are multi-modal. Functions f_1 to f_6 are maximization problems and functions f_7 to f_{12} are minimization problems. These functions are widely used in the literature for comparison, analysis and assessment of various algorithms. They are shown in Table 1.

For the presentation of GAs and PLAM_PSO, we use the gray encoding scheme and real coding, respectively. The parameters of PLAM_PSO were set as suggested in [3] as follows: the acceleration constants $\eta 1 = \eta 2 = 1.496180$ and the inertia weight $\omega = 0.729844$. For all the GAs, the genetic

Table 1: The test functions, where n and D are the number of variables (dimensions) and the domain of a problem ($D \in R^n$) respectively, f_1–f_6 are maximization functions, and f_7–f_{12} are minimization functions.

Test problem	n	D
$f_1(x) = \sum_{i=1}^{n} x_i^3$	30	$[0.0, 16.384]$
$f_2(x) = \sum_{i=1}^{n} x_i^4 \lvert \sin(\pi x_i) \rvert$	30	$[0.0,\ 16]$
$f_3(x) = x_1 - x_2 + x_3$	3	$[0.0, 1023.0]$
$f_4(x) = x_1 * x_2 + x_3$	3	$[0.0, 1023]$
$f_5(x) = x_1/(x_2 + 1) + x_3$	3	$[0.0, 1023]$
$f_6(x) = x_1 * x_2 * x_3 - 100x_1 * x_2$	3	$[-512, 512]$
$f_7(x) = \sum_{i=1}^{n} x_i^2$	30	$[-5.12, 5.12]$
$f_8(x) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$	2	$[-2.048, 2.048]$
$f_9(x) = \sum_{i=1}^{n} int(x_i)$	30	$[-5.12, 5.12]$
$f_{10}(x) = \sum_{i=1}^{n} x_i^4 + Gauss(0, 1)$	30	$[-1.28, 1.28]$
$f_{11}(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{\lvert x_i \rvert})$	30	$[-512, 512]$
$f_{12}(x) = \sum_{i=1}^{n} (x_i)^2/4000 - \prod_{i=1}^{n} \cos(x_i/\sqrt{i}) + 1$	30	$[-512, 512]$

operators were set as follows: the tournament selection with tournament size of 2, elitism of size 1, 2-point crossover with a probability 1.0 and the population size $N = 250$. The initial selection ratio was 1/3 for each adaptive mutation operator and the minimum selection ratio γ was set to 0.001 for each adaptive mutation operator, the update frequency U_f was set to 5 and T in [10] was set to 10 for PLAM_PSO. For PLAM_GA, the initial probability was set to $P_{mutation} = 0.1$ and $\delta = 0.01$. For SANUM, the parameters were fixed as: $(\alpha, \beta) = (0.05, 0.04)$ and $P_{min} = 0.0001$ (i.e, $pm(i,t) \in [0.0001, 0.05]$ for each locus i). For GBAM and GBAM_FAD, the following parameters were used: $\gamma = 0.001$, $[P_{min}, P_{max}] = [0.0001, 0.2]$, and initially $pm^1(i,0) = pm^0(i,0) = 0.01$ for each gene i.

3.2 Experimental Results and Analysis

This section presents the average result of 50 independent runs of each algorithm on the test functions. For each run of an algorithm on a function, 100 generations were allowed. The experimental results are shown in Table 2.

From Table 2, several results can be seen. Firstly, the performance of PLAM_PSO is better than other all mutation algorithms on all optimization benchmark functions. Especially the efficiency of PLAM_PSO, GBAM_FAD and GBAM is much better than other mutation algorithms on f_1, and f_2 (see Fig. 1). On function f_3, GBAM and both PLAM operators are better than other two mutation algorithms. The performacne of PLAM_PSO algorithm is good, GBAM and GBAM_FAD obtain the same results on function f_4. The GBAM algorithm gets the optimum result on function f_5, GBAM_FAD and PLAM_PSO gets closer result to GBAM on the same function. It can be seen that among the four algorithms with adaptive mutation operators, PLAM_PSO, GBAM, and PLAM_GA obtain the same results but better than GBAM_FAD and SANUM on function f_6.

Table 2: Average result over 50 independent runs of algorithms on the test functions.

Test function	PLAM_PSO	GBAM_FAD	GBAM	SANUM	PLAM_GA
f_1	131710	130454	131451	92664	108900
f_2	1.73e+06	1.717e+06	1.720e+06	1.076e+06	1.264e+06
f_3	2045.95	2045.83	2045.99	2033.77	2045.97
f_4	1.048e+06	1.047e+06	1.047e+06	1.045e+06	1.047e+06
f_5	2034.84	2043.77	2045.87	1695.16	2027.99
f_6	1.604e+8	1.603e+08	1.604e+08	1.574e+08	1.604e+08
f_7	0.03144	0.104391	0.082496	0.629343	0.226689
f_8	0.00092	0.0110618	0.0118705	0.150079	0.0373568
f_9	0	0	0	0	0
f_{10}	0.000678	0.00407724	0.00509537	0.0661653	0.0143548
f_{11}	-12305.6	-10286.2	-10270	-8728.76	-10052.6
f_{12}	0.476	5.33026	5.32015	51.6891	13.8288

Secondly, on minimum optimization functions, the performance of PLAM_PSO, GBAM and GBAM_FAD is also better than other algorithms with adaptive mutation operators in all tested functions. For function f_7, the performance of adaptive mutation operators is ranked in the following sequence: PLAM_PSO, GBAM, GBAM_FAD, PLAM_GA, and SANUM. The result of PLAM_PSO and GBAM_FAD are better than the other three adaptive mutation algorithms on function f_8. All five mutation algorithms obtain the global minimum optimum result after a few generations on f_9. PLAM_PSO is more efficient than the other four adaptive mutation operators on f_{10}. PLAM_PSO, GBAM_FAD and GBAM are more efficient than the other two adaptive mutation operators on f_{11} and f_{12} (see Fig. 1).

Thirdly, statistical analysis of five mutation operators in two groups (population-level adaptive mutation and Gene-level adaptive mutation operator) is carried out using the two-tailed t-test with a 98 degree of freedom at a 0.05 level of significance. Table 3 shows the t-test results for pairs of algorithms, where the result is shown as "+", "−", or "∼" if the first algorithm in a pair is significantly better than, significantly worse than, or statistically equivalent to the second algorithm, respectively. The PLAM_PSO, GBAM_FAD and GBAM algorithms are statistically better than other two adaptive approaches for finding the optimum value.

The performances of the five adaptive mutation algorithms are reasonably good except SANUM. Generally speaking, PLAM_PSO, GBAM_FAD and GBAM are the most efficient on both the minimum and maximum optimization problems.

4 Conclusions

This paper presents a comparative study of a population-level adaptive mutation operator with a gene-level adaptive mutation operator on multi-dimensional benchmark functions. The performance of different adaptive mutation operators varies on different functions. From the experimental results, it can be concluded that PLAM_PSO, GBAM_FAD and GBAM mutation algorithms perform well

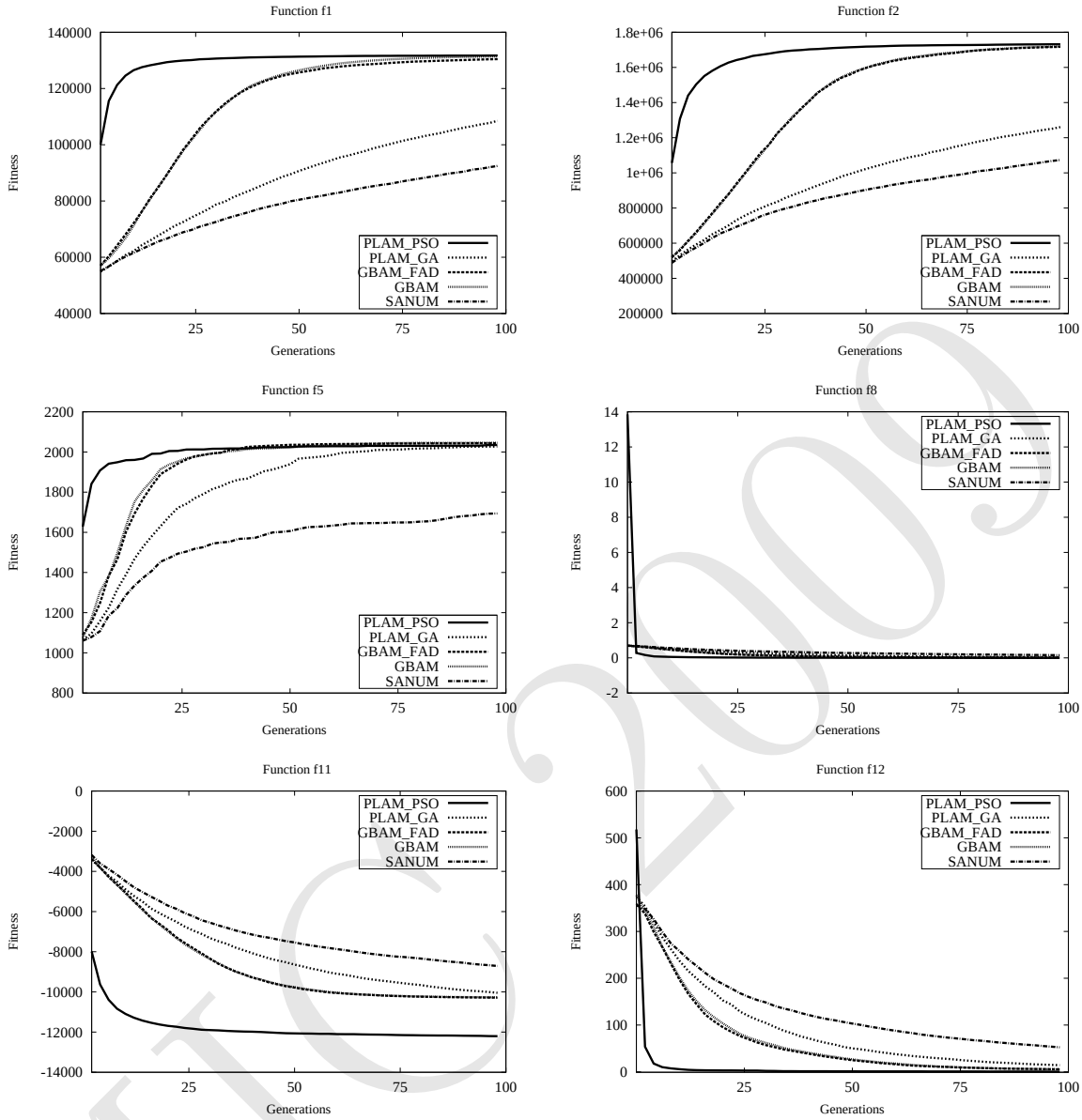

Figure 1: Experimental results of adaptive mutation operators.

on different functions. With PLAM_PSO, GBAM_FAD and GBAM, the population rapidly converges in a relatively short period of time to a near-optimal solution even for multi-modal functions. PLAM_PSO, GBAM_FAD and GBAM are statistically better than other adaptive approaches for finding the optimum value.

In general, the experimental results indicate that gene level mutation operators provide better solutions with reduced number of generations as compared with the PLAM_GA operator except PLAM_PSO. There is one drawback with the GLAM operators. It takes some time to calculate new mutation probabilities for each gene locus i at generation t. Generally speaking, the PLAM_PSO algorithm is better than other adaptive algorithms for finding the good result.

Table 3: Statistical comparison of adaptive mutation operators on the test functions.

Test function:	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}
PLAM_PSO – GBAM_FAD	+	+	+	+	−	+	+	+	+	+	+	+
PLAM_PSO – GBAM	+	+	−	+	−	∼	+	+	∼	+	+	+
PLAM_PSO – SANUM	+	+	+	+	+	+	+	+	+	+	+	+
PLAM_PSO – PLAM_GA	+	+	−	+	+	∼	+	+	+	+	+	+
GBAM_FAD – GBAM	−	∼	−	∼	−	−	∼	∼	+	∼	∼	∼
GBAM_FAD – SANUM	+	+	+	+	+	+	+	+	+	∼	+	+
GBAM_FAD – PLAM_GA	+	+	−	∼	∼	−	+	+	+	∼	+	+
GBAM – SANUM	+	+	+	+	+	+	+	+	+	∼	+	+
GBAM – PLAM_GA	+	+	∼	+	∼	∼	+	+	+	∼	+	+
SANUM – PLAM_GA	−	−	−	−	−	−	−	−	−	∼	−	−

References

[1] P. J. Angeline. Adaptive and self-adaptive evolutionary computations. In M. Palaniswami, Y. Attikiouzel, R. J. Marks II, D. B. Fogel, and T. Fukuda (eds.) *Computational Intelligence: A Dynamic Systems Perspective*, IEEE Press, New York, 152–163, 1995.

[2] T. Bäck. Self-Adaptation in Genetic Algorithms. In *Proc. of the 1st European Conf. on Artificial Life*, 263–271, 1992.

[3] F. van den Bergh. An analysis of particle swarm optimizers. *PhD Thesis*, Department of Computer Science, University of Pretoria, South Africa, 2002.

[4] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in Evolutionary Algorithms. *IEEE Trans. on Evolutionary Computation*, 3(2): 124–141, 1999.

[5] A. E. Eiben, Z. Michalewics, M. Schoenauer, and J. E. Smith. Parameter Control in Evolutionary Algorithms. In F. Lobo, C. Lima, and Z. Michalewics (Eds), *Parameter Setting in Evolutionary Algorithms*, Chapter 2, 19–46, 2007.

[6] D. E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning.* Addison Wesley, 1989.

[7] J. J. Greffenstette. Optimization of Control Parameters for Genetic Algorithms. *IEEE Trans. on Sys. Man and Cyber.*, 16(1): 122–128, 1986.

[8] T. P. Hong, H. S. Wang, and W. C. Chen. Simultaneously applying multiple mutation operators in genetic algorithms. *Journal of Heuristics*, 6: 439–455, 2000.

[9] K. A. De Jong. An Analysis of the Behavior of a Class of Genetic Adaptive Systems. *PhD Thesis*, Department of Computer and Communication Science, University of Michigan, Ann Abor, 1975.

[10] C. Li, S. Yang, and I. Korejo. An Adaptive Mutation Operator for particle swarm optimization. *Proceeding of the 2008 UK workshop on Computational intelligence*, 165–170, 2008.

[11] H. Mühlenbein. How Genetic Algorithms Really Work I. Mutation and Hillclimbing. In R.Männer and B. Manderick (eds.), *Proc. of the 2nd Conf. on Parallel Problem Solving from Nature*, 15–29, 1992.

[12] Z. Pan and L. Kang. An Adaptive Evolutionary Algorithm for Numerical Optimization. *Proc. of the 1st Asia-Pacific Conference on Simulated Evolution and Learning*, 27–34, 1996.

[13] H.-P. Schwefel. *Numerical Optimization of Computer Models*. Wiley, Chichester, 1981.

[14] J. E. Smith and T. C. Fogarty. Self-adaptation of mutation rates in a steady-state genetic algorithm. *Proc. of the 3rd IEEE Conf. on Evolutaionary Computation*, 318–323, 1996.

[15] J. E. Smith and T. C. Fogarty. Operator and parameter adaptation in genetic algorithms. *Soft. Computing*, 1: 81–87, 1997.

[16] R. S. Sutton, and A. G.Barto. *Reinforcement Learning An Introduction*. MIT Press 1998.

[17] D. Thierens. An adaptive pursuit strategy for allocating operator probabilities. *Proc. of the 2005 Genetic and Evolutionay Computation Conference*, 1539–1546, 2005.

[18] D. Thierens. Adaptive strategies for operator allocation. In F. Lobo, C. Lima, and Z. Michalewics (eds.), *Paramter Setting in Evolutionary Algorithms*, Chapter 4, 77–90, 2007.

[19] S. Uyar, S. Sariel, and G. Eryigit. A gene based adaptive mutation strategy for genetic algorithms. *Proc. of the 2004 Genetic and Evolutionary Computation Conference*, 271–281, 2004.

[20] S. Yang. Adaptive mutation using statistics mechansim for genetic algorithms. In F. Coenen, A. Preece, and A. Macintosh (eds.), *Research and Development in Intellignet Systems XX.*, 19–32, 2003.

[21] S. Yang and S. Uyar. Adaptive mutation with fitness and allele distribution correlation for genetic algorithms. *Proceedings of the 21st ACM Symposium on Applied Computing*, 940–944, 2006.