Adaptive Mutation Using Statistics Mechanism for Genetic Algorithms

Shengxiang Yang
Department of Computer Science
University of Leicester
University Road, Leicester LE1 7RH, UK
Email: s.yang@mcs.le.ac.uk

Abstract

It has long been recognized that mutation is a key ingredient in genetic algorithms (GAs) and the choice of suitable mutation probability will have a significant effect on the performance of genetic search. In this paper, a statistics-based adaptive non-uniform mutation (SANUM) is presented within which the probability that each gene will subject to mutation is learnt adaptively over time and over the loci. As a search algorithm based on mechanisms abstracted from population genetics, GAs implicitly maintain the statistics about the search space through the population. SANUM explicitly makes use of the statistics information of the allele distribution in each gene locus to adaptively adjust the mutation probability of that locus. To test the performance of SANUM, it is compared to traditional bit mutation operator with a number of "standard" fixed mutation probabilities suggested by other researchers over a range of typical test problems. The results demonstrate that SANUM performs persistently well over the range of test problems while the performance of traditional mutation operators with fixed mutation probabilities greatly depends on the problem under consideration. SANUM represents a robust adaptive mutation operator that needs no prior knowledge about the fitness landscape of the problem being solved.

1 Introduction

The genetic algorithm (GA), as one kind of generation-based evolutionary algorithm, maintains a population of candidate solutions to a given problem, which are evaluated according to a problem-specific fitness function that defines the environment for the evolution. New populations are created by selecting relatively fitter members of the present population and evolving them through recombination and mutation operations [12]. The performance of a GA depends on many factors, such as the encoding scheme, the selection method, the population size, the crossover and mutation operators. This makes it difficult, if not impossible, to choose operators and relevant parameters for optimal performance. In this paper we focus on the mutation operator.

Holland [15] introduced the mutation operator as a "background operator" that changes bits of individuals only occasionally, with a rather small mutation probability $p_m \in [0, 1]$ per bit. Mutation is used to ensure that all possible alleles can enter the population and provide variation in a GA population. This

is done by having one allele replaced by another. For example, in a binary string representation, mutation on a parent to produce a child is done by flipping each bit randomly according to a mutation probability. Based on Holland's simple GA, there has been much work, both practically [20] and theoretically [23] on the relative merits of mutation as a search mechanism. Much of the work has been concerned with finding globally "optimal" mutation probability for GAs. Common settings of static mutation probability recommended by researchers are as follows: $p_m = 0.001$ by De Jong [7], $p_m = 0.01$ by Grefenstette [11], and $p_m \in [0.005, 0.001]$ by Schaffer *et al* [20]. Based on the result of Schaffer *et al* [20], Bäck [3] proposed an expression of $p_m = 1.75/(N * L^{1/2})$ where N is the population size and L denotes the string length. Mühlenbein [19] derived that $p_m = 1/L$ should be generally 'optimal', which was further verified by Smith and Fogarty [22]. These settings were obtained by experimental investigations and are consistent with Holland's proposal of using mutation as a background operator. However, there is an increasing body of both empirical [10] and theoretical [3] evidence showing that the optimal mutation probability will not only depend on the problem being solved but also vary with the progress of evolutionary searching.

In this paper, a statistics-based adaptive non-uniform mutation (SANUM) is proposed for GAs. As a search algorithm based on mechanisms abstracted from population genetics, GAs implicitly maintain the statistics about the search space through the population. GAs use the selection, crossover and mutation operations to extract the implicit statistics from the population to reach the next set of points in the search space. This implicit statistics can be used explicitly to enhance GA's performance. SANUM explicitly makes use of the statistics information of the distribution of alleles in a gene locus over the population to adjust the mutation probability for that locus adaptively with the progress of the GA.

In the rest of this paper, we first briefly review relevant work, next describe SANUM in detail, and then present our experiment study that compares SANUM over traditional bit mutation based on a typical set of test problems, finally give out our conclusions as well as discussions on potential future work relevant to SANUM.

2 Adapting the Mutation in GAs

The adaptation of genetic operators and relevant parameters was first introduced into Evolutionary Strategies (ESs) where the mutation step size is successfully controlled by self-adaptation, e.g., see Schwefel [21]. In recent years with the interaction between the GA and the ES communities, there has been an increasing interest in the use of adaptive operators within the GA to enhance GA's performance [9]. Based on how the strategy parameters are changed adaptation in GAs can be classified into three categories: *deterministic mechanism* where the value of a strategy parameter is altered according to some deterministic rule, *adaptive mechanism* where there is some form of feedback

information from the search process that is used to direct the change of a strategy parameter, and *self-adaptive mechanism* where the parameter to be adapted is encoded into the chromosomes and undergoes genetic operations (hence, also called *co-evolution*).

There has been much work on adaptation in mutation for GAs [4]. Generally speaking, adaptation in mutation happens at two levels. At the top level, the ratio between mutation and crossover is adapted during the run of a GA. Davis [6] proposed that the GA can select from a set of operators to perform on a chosen parent, each with a fixed probability. Julstrom [16] adaptively adapted the ratio between mutation and crossover based on their performance. Corne *et al.* [5] devised the COst Based operator Rate Adaptation (COBRA) method where the GA periodically swaps given k fixed probabilities between k operators by giving the highest probability to the operator that has been producing the most gains in fitness. Tuson and Ross [24] extended the COBRA method by co-evolving the mutation and crossover probabilities (one-normalized real numbers) with each individual.

At the bottom level, the probability of mutation is adapted during the run of a GA, uniformly or non-uniformly over the loci. Bäck [3] proposed a self-adaptation scheme by adding a probability vector $\vec{p} = \{p_1, \ldots, p_n\}$ (n is the number of object variables) for each individual. The mutation scheme first mutates the mutation probability p_i with p_i itself and then uses the resulting p_i to mutate the ith object variable. Hesser and Männer [14] derived a general expression that deterministically varies the mutation probability with time by

$$p_m(t) = (\alpha/\beta)^{1/2} \times exp(-\gamma t/2)/(N \times L^{1/2})$$

where α, β, γ are constants, N is the population size, L is the string length, and t is the time (generation counter). Fogarty [10] used a deterministic scheme that decreases p_m over time exponentially, such that $lim_{t\to\infty} P_m(t) = 0$. Fogarty [10] also used a scheme that decreases the mutation probability over bit representation and gives bits of different significance different schedules. In this paper a new mechanism is proposed that adapts the mutation probability over the loci but needs no knowledge in advance about the bit representation of the problem, such as the significance of a bit position as in Fogarty's scheme [10].

3 Statistics-based Adaptive Non-Uniform Mutation

For the convenience of description and analysis, we introduce the concepts of intrinsic attribute and extrinsic tendency of allele valuing for a gene locus. In the binary-encoded optimal solution(s) of a given problem, a gene locus is called *1-intrinsic* if its allele is 1, *0-intrinsic* if its allele is 0, or *neutral* if its allele can be either 0 or 1. Whether a locus is 1-intrinsic, 0-intrinsic, or neutral depends on the problem under consideration and encoding scheme, e.g., whether introns are inserted [17]. During the running of a GA, a gene locus is called *1-inclined* if the frequency of 1s in its alleles over the population tends to increase (to the

limit of 1.0) with time (generation), *0-inclined* if the frequency of 1s tends to decrease (to the limit of 0.0), or *non-inclined* if there is no tendency of increasing or decreasing. Whether a locus is 1-inclined, 0-inclined, or non-inclined depends on the problem under consideration, encoding scheme, genetic operators and initial conditions.

Usually with the progress of the GA, those gene loci that are 1-intrinsic (or 0-intrinsic) will appear to be 1-inclined (or 0-inclined), i.e., the frequency of 1s in the alleles of these loci will eventually converge to 1 (or 0). SANUM makes use of this convergence information as feedback information to control the mutation by adjusting the mutation probability for each locus.

We use the frequency of 1s in the alleles in a locus over the population (equivalently we can use the frequency of 0s as the argument) to calculate corresponding mutation probability of that locus. The frequency of 1s in a locus's alleles can be looked as the degree of convergence to "1" for that locus. Let $f_1(i,t)$ $(i = 1,\ldots,L)$ denote the frequency of 1s in the alleles in locus i over the population at time (generation) t and $p_m(i,t)$ $(i = 1,\ldots,L)$ denote the mutation probability of locus i at time t. Then, as shown in Figure 1, $p_m(i,t)$ can be calculated from $f_1(i,t)$ as follows:

$$p_m(i,t) = P_{max} - 2 * |f_1(i,t) - 0.5| * (P_{max} - P_{min}) \tag{1}$$

where $|x|$ denotes the absolute value of x, P_{max} and P_{min} are the maximum and minimum allowable mutation probabilities for a locus respectively, e.g., $P_{max} = 1/L$ and $P_{min} = 10^{-4}$.

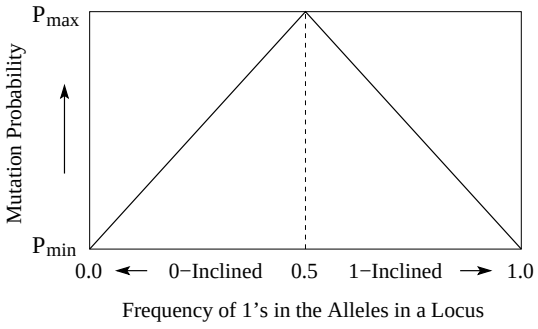

Figure 1: Triangular function used for calculating the mutation probability for a gene locus.

Now during the evolution of the GA, after a new population t has been generated, we first calculate the distribution of 1s $f_1(i,t)$ for each locus i over the population and from this obtain the mutation probability $p_m(i,t)$ for gene locus i. Then we can perform SANUM operations similarly as traditional bit mutation except that SANUM uses $p_m(i,t)$ for each locus i instead of a global p_m for all the loci.

According to the classification of adaptation for GAs reviewed in section 2, the goal of adaptive mechanism is to use the knowledge dynamically acquired about the search space to adjust the GA suitably to the problem. SANUM belongs to the class of adaptive mechanism that occurs at the bottom-level of mutation. It uses the statistics of allele distribution as feedback information to adaptively adjust the mutation probability non-uniformly over the loci, hence the name Statistics-based Adaptive Non-Uniform Mutation (SANUM).

SANUM is simpler than those mechanisms that add one extra value per bit and co-evolve these values with each individual, such as Bäck's self-adaptation scheme [3]. With SANUM, what we add to traditional bit mutation are spatially only one real vector of L-dimension that records the frequency of ones for each locus, and computationally only one statistics per generation that calculates the frequency of ones over the population (hence the mutation probability) for each locus. This simple statistics added is well rewarded as discussed below.

The motivation behind SANUM lies in the fact that with the progress of genetic search SANUM helps protecting building blocks found so far while still exploiting new building blocks. To formally analyze the behavior of GAs, Holland [15] first introduced the concept of *schema* to describe a subset of binary strings of fixed length that have similarities at certain positions. For example, the schema $S = 1****0$ (where "*" denotes a "don't care" bit) represents the set of all 6-bit strings that begin with 1 and end with 0. Given a schema S, its *order* $o(S)$ is the number of fixed positions within S and its *defining length* $l(S)$ is the distance between the outermost fixed bits of S. Building blocks are short, low-order, better than average schemas. Holland's *schema theorem* states that building blocks receive an exponentially increasing number of trials in the subsequent generations.

In fact building blocks can be looked as the combination of non-neutral (1-intrinsic or 0-intrinsic) genes. When the population is randomly initialized, the frequency of 1s in the alleles in each locus is statistically about 0.5 and hence $p_m(i,t) = P_{max}$ for all the loci. However, with the progress of the GA, when building blocks are partially found, that is, some 1-intrinsic and 0-intrinsic loci tend to converge to 1 and 0 respectively, SANUM decreases the mutation probabilities of these loci according to Equation (1). In this way, SANUM can protect building blocks found so far that are located on these partially or totally converged loci. While on the other hand, for those unconverged loci the mutation probabilities are remained to be high within SANUM. This is useful because there may be building blocks not expressed on these loci yet. From these discussions, it can be seen that SANUM strikes to balance the construction of new building blocks and protection of found building blocks over the loci with time adaptively.

Traditional bit mutation keeps a constant mutation probability over all the loci. As the population converges, with traditional bit mutation, in fact fewer and fewer offsprings generated by mutating those converged loci will survive in the next generation. That is, many mutation operations are wasted on those converged loci. Most of these wasted mutation operations and hence wasted fitness evaluations are saved by SANUM through adaptively decreasing $p_m(i,t)$

to P_{min} for those converged loci.

4 The Test Problems

In order to compare the performance of SANUM, a wide range of typical benchmark problems is selected as the test set. These problems represent different difficulty levels for GAs. They are described as follows.

4.1 The One-Max Problem

The One-Max problem [1] simply counts the ones contained in a binary string as the fitness of that string. The aim is to obtain a string containing all ones, that is, to maximize ones in a string. A string length of 100 bits was used for our study.

4.2 The Royal Road Functions

The Royal Road functions R_1 and R_2 were devised by Mitchell, Forrest and Holland [18] to investigate GA's performance with respect to schema processing and recombination in an idealized form. Royal Road functions contain tailor-made building blocks (schemas) based on 64-bit binary strings. They are defined using a list of schemas. Each schema s_i is given a coefficient c_i which is equal to its order $o(s_i)$ (a schema's order is the number of fixed positions within that schema). R_1 consists of 8 disjunctive order-8 schemas of which each has 8 adjacent ones. R_2 consists of four levels of schemas: level 0 (bottom level) is the same as R_1, level 1 has 4 order-16 schemas of which each combines two adjacent schemas in level 0, level 2 contains 2 order-32 schemas each combining two adjacent schemas in level 1, and finally level 3 (the optimal schema) combines the 2 schemas in level 2. The fitness of a bit string x for $R_1(x)$ and $R_2(x)$ is computed by summing the coefficients c_i corresponding to each of the given schema s_i of which x is an instance. That is, $R_1(x)$ and $R_2(x)$ are defined as follows:

$$R_1(x) = \sum_{i=1}^{i=8} c_i \delta_i(x) \text{ and } R_2(x) = \sum_{i=1}^{i=14} c_i \delta_i(x)$$

where $\delta_i(x) = \{1, \text{if } x \in s_i; 0, \text{otherwise}\}$. The optimal solutions for R_1 and R_2 are given as: $R_1(111...1) = 64$ and $R_2(111...1) = 192$.

4.3 The L-SAT Problem Generator

The random L-SAT problem generator devised by De Jong, Potter and Spears [8] is a boolean satisfiability problem generator devised to investigate the effects of epistasis on the performance of GAs. It generates random boolean expressions in conjunctive normal form of clauses subject to three parameters v (number of boolean variables), c (number of disjunctive or conjunctive clauses) and l (the length of the clauses). Each clause is created by selecting l

of v variables uniformly randomly and negating each variable with probability 0.5. For each generated boolean expression, the aim is to find an assignment of truth values to the v variables that makes the entire expression true. Since the boolean expression is randomly generated, there is no guarantee that such an assignment exists. The difficulty and complexity of the problem varies with the parameters v, c and l. For example, increasing the number of clauses increases the epistasis. The fitness function for the L-SAT problem is as follows:

$$f(chrom) = \frac{1}{c} \sum_{i=1}^{c} f(clause_i)$$

Where *chrom* consists of c clauses and the fitness contribution of clause i, $f(clause_i)$, is 1 if the clause is satisfied or 0 otherwise.

In our experiments we will use the same parameters as in [8]. We fixed the number of variables v to 100 and the length of the clauses l to 3. The number of clauses c was varied from 200 (low epistasis) to 1200 (medium epistasis) to 2400 (high epistasis).

4.4 Deceptive Functions

Deceptive functions are those functions where the low-order building blocks do not combine to form higher-order building blocks: instead they form building blocks resulting in a sub-optimal solution. Deceptive functions are developed as difficult test functions for comparing different implementations of GAs. Goldberg, Korb and Deb [13] have devised an order-3 minimum fully deceptive problem as follows:

```
f(000) = 28  f(001) = 26  f(010) = 22  f(011) = 0
f(100) = 14  f(101) = 0   f(110) = 0   f(111) = 30
```

where all the order-1 and order-2 building blocks (e.g., "0**" and "*00" where the wildcard "*" matches both 0 and 1) in the search space are deceptive and will lead the genetic search away from the global optimum "111" and toward the deceptive local optimum "000" instead.

Based on an algorithm of constructing fully deceptive functions, Whitley [25] has also developed an order-4 fully deceptive problem as follows:

```
f(0000) = 28  f(0001) = 26  f(0010) = 24  f(0011) = 18
f(0100) = 22  f(0101) = 16  f(0110) = 14  f(0111) = 0
f(1000) = 20  f(1001) = 12  f(1010) = 10  f(1011) = 2
f(1100) = 8   f(1101) = 4   f(1110) = 6   f(1111) = 30
```

In this paper, we constructed two 60 bit deceptive functions: one contains 20 copies of Goldberg, Korb and Deb's 3-bit fully deceptive subfunction (called Deceptive Function DF_1) and another contains 15 copies of Whitley's 4-bit fully deceptive subfunction (called Deceptive Function DF_2). The optimal solutions for DF_1 and DF_2 have a fitness of 600 and 450 respectively.

5 Experimental Study

5.1 Design of Experiment

In order to test SANUM, in this experimental study it is compared with traditional bit mutation with a series of recommended "standard" fixed probabilities: $1/L$ by Mühlenbein [19], 0.01 by Grefenstette [11], $1.75/(N * L^{1/2})$ by Bäck [3], and 0.001 by De Jong [7]. Note that for L-SAT and One-Max problems, $1/L = 0.01$. For SANUM, the mutation probability $p_m(i,t)$ for each locus i ($i = 1, \ldots, L$) varied adaptively with time (generation counter t) between $P_{max} = 1/L$ and $P_{min} = 10^{-4}$ according to equation (1). For each experiment of combining different mutation (traditional bit flip mutation with different mutation rate or SANUM) and test problem, 100 independent runs were executed. In all the experiments, the fitness proportionate selection with the stochastic universal sampling [2], 2-point crossover, and elitist model [7] were used in the GA. The crossover probability was fixed to the typical value of 0.6, and the population size was set to 100 for each run. In order to have a strict comparison the same 100 different random seeds were used to generate initial populations for the 100 runs of each experiment. For each run, the initial population is randomly created using a technique that generates exactly equal number of 0s and 1s for each locus[1], that is, $f_1(i,0) = 0.5$ for each locus i ($i = 1, \ldots, L$) in the initial population. In this way the random sampling bias in the initial population (for example for some locus j, $f_1(j,0) = 0.8$ or 0.2) that may mislead SANUM is cancelled.

For each run, the best-so-far fitness was recorded every 100 evaluations. Here, only those chromosomes changed by crossover and mutation operations were evaluated and counted into the number of evaluations. For each run, the maximum allowable number of evaluations varies with the test problem and is suitably set to let all the mutation operators have the chance to express their power. Each experiment result was averaged over 100 independent runs.

5.2 Experimental Results

The experimental results on different test problems are shown in Figure 2 to Figure 5 respectively. From these figures, the following observations can be seen.

First, SANUM performs persistently well on the test problems. In fact, SANUM performs better than traditional mutation operators with different "standard" fixed rates on all the test problems except the deceptive functions DF_1 and DF_2. On the deceptive functions, SANUM is beaten by traditional mutation operators with $p_m = 0.001$ and $p_m = 1.75/(N * L^{1/2})$. However, it also performs better than bit mutation with $p_m = 1/L$ and as well as bit mutation with $p_m = 0.01$.

Second, $p_m = 0.001$ suggested by De Jong [7] seems to be a good choice for bit mutation. It performs better than other fixed probabilities on most of the

[1] It is possible because the population size is set to an even number, 100.

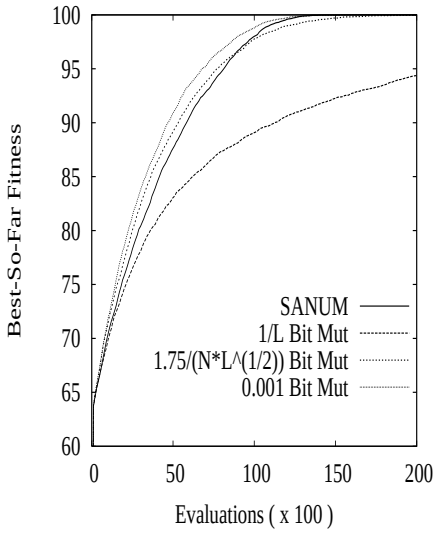

Figure 2: Experiment results with respect to best-so-far fitness against evaluations of GAs with different mutation operators on One-Max problem. The data were averaged over 100 runs.

test problems. However, its performance is seriously the worst on royal road functions R_1 and R_2. This happens because of its worst construction power for building blocks. This example shows that the performance of traditional bit mutation with fixed mutation probability heavily depends on the problem fitness landscape.

Third, SANUM performs much better than traditional mutation operators on royal road functions R_1 and R_2 (see Figure 3). This is due to the strong building blocks built in the royal road functions. During the early stage of searching GAs with traditional bit mutation perform better than the GA with SANUM. However, after certain evaluations, when some useful building blocks has been built up the GA with SANUM greatly outperforms GAs with traditional bit mutation because SANUM efficiently avoids mutating those converged loci where building blocks already found so far reside. Similar results can be observed on L-SAT problems (see Figure 4) as on royal road functions. The GA with SANUM is beaten by GAs with traditional bit mutation with $p_m = 0.001$ and $p_m = 1.75/(N * L^{1/2})$ during the early stage of searching. However, after certain evaluations the GA with SANUM outperforms GAs with traditional bit mutation.

Finally, as recognized by other researchers, the choice of mutation operators and proper probabilities in genetic algorithms really has a significant effect on the performance of genetic search. Hence, developing mutation operators that

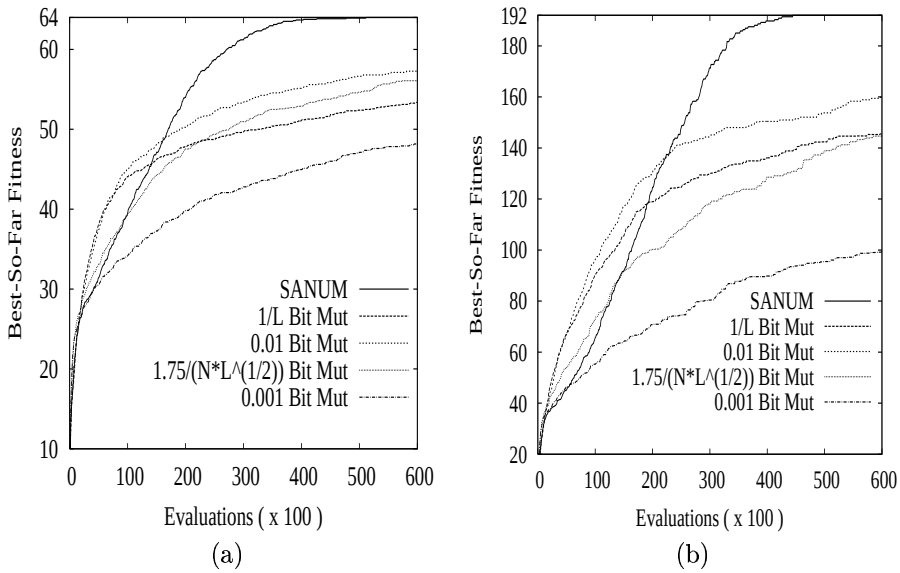

Figure 3: Experiment results with respect to best-so-far fitness against evaluations of GAs with different mutation operators on Royal Road Functions (a) R_1 and (b) R_2. The data were averaged over 100 runs.

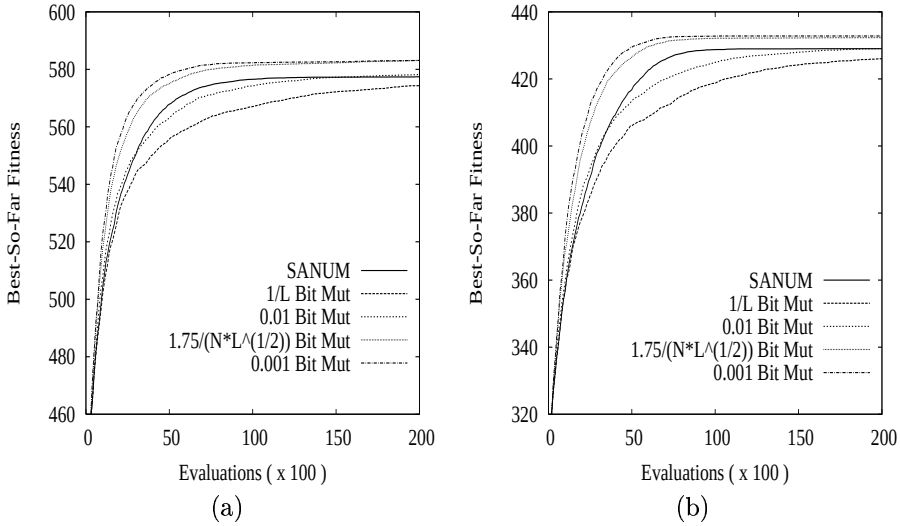

Figure 4: Experiment results with respect to best-so-far fitness against evaluations of GAs with different mutation operators on Deceptive Functions (a) DF_1 and (b) DF_2. The data were averaged over 100 runs.

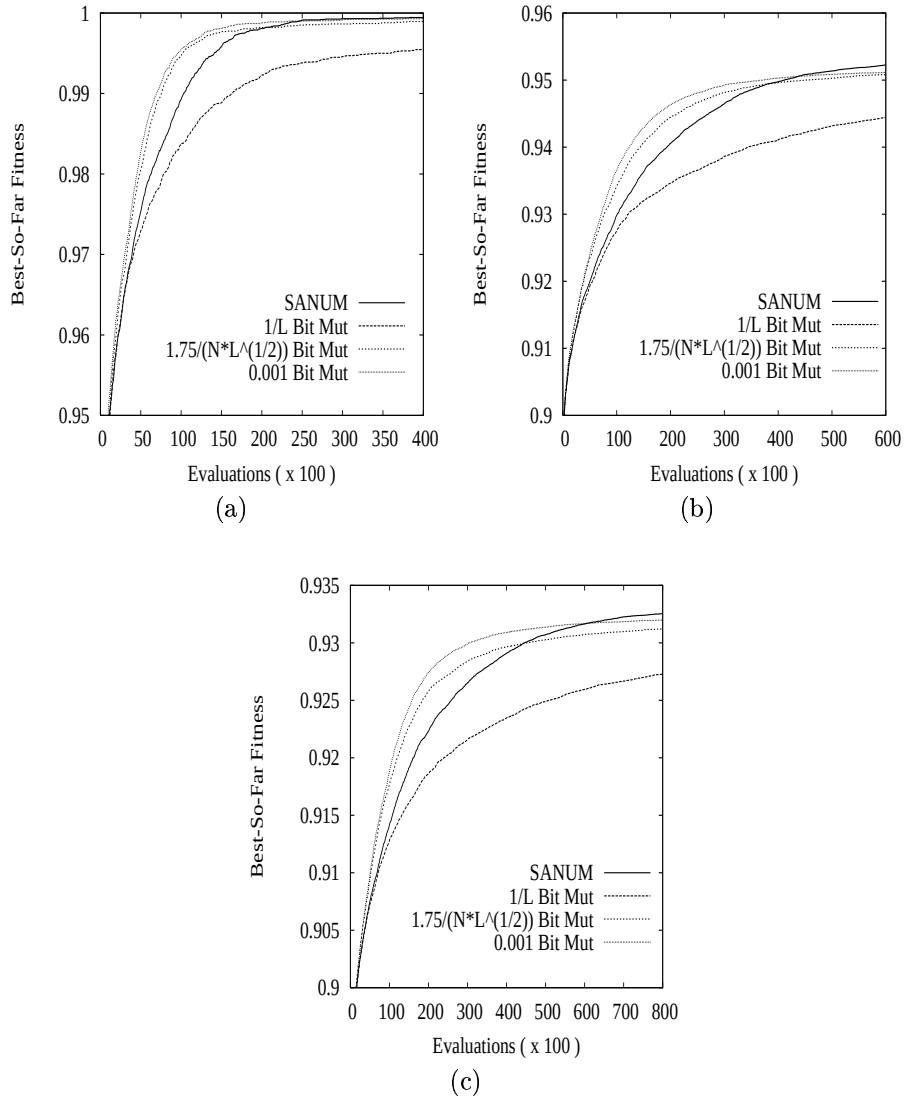

Figure 5: Experiment results with respect to best-so-far fitness against evaluations of GAs with different mutation operators on L-SAT problems with $v = 100$, $l = 3$ and (a) low epistasis $c = 200$, (b) medium epistasis $c = 1200$, and (c) high epistasis $c = 2400$. The data were averaged over 100 runs.

can adjust the mutation probability adaptively with the problem fitness landscape and with the progress of genetic searching is really meaningful for improving GA's performance. SANUM is obviously a good attempt to this direction.

6 Conclusions

In this paper a statistics-based adaptive non-uniform mutation, SANUM, is proposed for genetic algorithms The motivation of SANUM is to make use of the statistics information implicitly contained in the population explicitly to guide the mutation operation. SANUM achieves this by using the allele distribution in the current population to adjust the mutation probability for each gene locus adaptively during the progress of the GA. Through decreasing the mutation probabilities on those converged loci SANUM can save mutation operations wasted on them.

The experimental results of this study demonstrate that the GA with SANUM performs persistently well over a wide range of test problems while the performance of GAs with traditional bit mutation with different fixed mutation probabilities greatly depends on the problem under consideration. The experiment results indicate that SANUM represents a robust adaptive mutation operator that needs no prior knowledge about the problem fitness landscape and that it is a good candidate mutation operator for GAs.

Since SANUM works at the bottom-level of mutation, it can be easily combined with other adaptation techniques for mutation and can act as the basis for analyzing and designing new related algorithms. In this study, a simple triangular function is used to calculate the mutation probability for each locus. Other functions such as exponential functions instead may further improve GA's performance, which is one future work about SANUM. Comparing obtained SANUM with other adaptation techniques for mutation is another future work about SANUM.

References

[1] D. H. Ackley (1987). *A Connectionist Machine for Genetic Hillclimbing*. Boston, MA: Kluwer Academic Publishers.

[2] J. E. Baker (1987). Reducing bias and inefficiency in the selection algorithms. In J. J. Grefenstelle (ed.), *Proc. 2nd Int. Conf. on Genetic Algorithms*, 14-21. Lawrence Erlbaum Associates.

[3] T. Bäck (1992). Self-Adaptation in Genetic Algorithms. In F. J. Varela and P. Bourgine (eds.), *Proc. of the 1st European Conf. on Artificial Life*, 263-271. MIT Press.

[4] T. Bäck (1997). Mutation Parameters. In T. Bäck, D. B. Fogel, and Z. Michalewicz (eds.), *Handbook of Evolutionary Computation*, E1.2.1-E1.2.7. Oxford University Press.

[5] D. Corne, P. Ross and H.-L. Fang (1994). GA Research Note 7: Fast Practical Evolutionary Timetabling. *Technical Report*, Department of Artificial Intelligence, University of Edinburgh, UK.

[6] L. Davis (1989). Adapting operator probabilities in genetic algorithms. In D. Schaffer (ed.), *Proc. of the 3rd Int. Conf. on Genetic Algorithms*, 60-69. San Mateo CA: Morgan Kaufmann Publishers.

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

[8] K. A. De Jong, M. A. Potter and W. M. Spears (1997). Using problem generators to explore the effects of epistasis. In T. Bäck (ed.), *Proc. of the 7th Int. Conf. on Genetic Algorithms*, 338-345. San Mateo, CA: Morgan Kaufmann Publishers.

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

[10] T. C. Fogarty (1989). Varying the Probability of Mutation in the Genetic Algorithm. In J. D. Schaffer (ed.), *Proc. of the 3rd Int. Conf. on Genetic Algorithms*, 104-109. San Mateo, CA: Morgan Kaufmann Publishers.

[11] J. J. Grefenstette (1986). Optimization of Control Parameters for Genetic Algorithms. *IEEE Trans. on Systems, Man and Cybernetics*, **16**(1): 122-128.

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

[13] D. E. Goldberg, B. Korb and K. Deb (1989). Messy Genetic Algorithms: Motivation, Analysis, and First Results. *Complex Systems*, **4**: 415-444.

[14] J. Hesser and R. Männer (1991). Towards an Optimal Mutation Probability in Genetic Algorithms. In H.-P. Schwefel, R. Männer (eds.), *Proc. of the 1st Conf. on Parallel Problem Solving from Nature*, 23-32.

[15] J. H. Holland (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, University of Michigan Press.

[16] B. Julstrom (1995). What have you done for me lately? adapting operator probabilities in a steady-state genetic algorithm. In L. J. Eshelman (ed.), *Proc. of the 6th Int. Conf. on Genetic Algorithms*, 81-87. San Mateo, CA: Morgan Kaufmann Publishers.

[17] J. Levenick (1995). Metabits: Genetic Endogenous Crossover Control. In L. J. Eshelman (ed.), *Proc. of the 6th Int. Conf. on Genetic Algorithms*, 88-95. San Mateo, CA: Morgan Kaufmann Publishers.

[18] M. Mitchell, S. Forrest and J. H. Holland (1992). The Royal Road for Genetic Algorithms: Fitness Landscapes and GA Performance. In F. J. Varela and P. Bourgine (eds.), *Proc. of the 1st European Conference on Artificial Life*, 245-254. Cambridge, MA: MIT Press.

[19] H. Mühlenbein (1992). 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.

[20] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das (1989). A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization. In J. D. Schaffer (ed.), *Proc. of the 3rd Int. Conf. on Genetic Algorithms*, 51-60. San Mateo, CA: Morgan Kaufmann Publishers.

[21] H-P. Schwefel (1981). Numerical Optimization of Computer Models. Wiley, Chichester.

[22] J. E. Smith and T. C. Fogarty (1996). Self-adaptation of Mutation Rates in a Steady-State Genetic Algorithm. In *Proc. of the 3rd IEEE Conf. on Evolutionary Computation*, 318-323. IEEE Press.

[23] W. Spears (1992). Crossover or Mutation. In L. D. Whitley (ed.), *Foundations of Genetic Algorithms 2*. San Mateo, CA: Morgan Kaufmann Publishers.

[24] A. Tuson and P. Ross (1998). Adapting Operator Settings in Genetic Algorithms. *Evolutionary Computation*, **6**(2): 161-184.

[25] L. D. Whitley (1991). Fundamental Principles of Deception in Genetic Search. In G. J. E. Rawlins (ed.), *Foundations of Genetic Algorithms 1*, 221-241. San Mateo, CA: Morgan Kaufmann Publishers.