Adaptive Non-Uniform Mutation Based on Statistics for Genetic Algorithms

Shengxiang Yang

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

Abstract

As a meta-heuristic search algorithm based on mechanisms abstracted from population genetics, the genetic algorithm (GA) implicitly maintains the statistics about the search space through the population. This implicit statistics can be explicitly used to enhance GA's performance. In this paper, a statistics-based adaptive non-uniform mutation (SANUM) is proposed. SANUM uses the statistics information of the allele distribution in each locus to adaptively adjust the mutation operation. Our preliminary experiments show that SANUM outperforms traditional bit flip mutation across a representative set of test problems.

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 population is created by selecting relatively fit members of the present population and evolving them through recombination and mutation operations (Goldberg, 1989). The performance of a GA is dependent on many factors, such as encoding scheme, selection method, population size, crossover and mutation operators. This makes it difficult, if not impossible, to choose operators for optimal performance. In this paper we focus on the mutation operator.

In Holland's GA (Holland, 1975), mutation operator is treated 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 allele can enter the population and provide variation in a GA population. Common settings of static mutation probability are recommended as follows: $p_m = 0.001$ by De Jong (De Jong, 1975), $p_m = 0.01$ by Grefenstette (Grefenstette, 1986), and $p_m \in [0.005, 0.001]$ by Schaffer et al (Schaffer, 1989). Mühlenbein derived that $p_m = 1/l$, where l denotes the string length in this paper, should be generally 'optimal' (Mühlenbein, 1992), which was further verified by Smith and Forgerty (1996). Recently, researchers have applied adaptation techniques to mutation to enhance GA's performance and have made the adaptation of mutation one of the most promising research areas in GAs (Eiben, Hinterding, and Michalewicz, 1999).

As a search algorithm based on mechanisms abstracted from population genetics, the GA implicitly maintains the statistics about the search space via the population. It uses the selection, crossover and mutation operations to explicitly 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. In this paper, a new statistics-based adaptive non-uniform mutation operator, called SANUM, is proposed. SANUM explicitly uses 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.

2 Review of Related Work

2.1 Adaptation in Genetic Algorithms

The performance of the GA significantly depends on genetic operators and relevant parameters. However, choosing the right operators and appropriate parameters for the GA is a difficult task. Traditionally, they are determined by experience or primary experiments

from a particular domain in advance and then are fixed during the running of the GA. This kind of constant parameter setting approach is time-consuming and can lead to sub-optimal performance when parameters are inappropriately set. And what is worse lies in the nature that the optimal parameter values may vary with the evolution process of GAs. Hence, researchers have applied many adaptation techniques into GAs to enhance their performance (Eiben, Hinterding, and Michalewicz, 1999). Based on the mechanism of change, adaptation in GAs can be classified into three categories: *deterministic adaptation* where the value of a strategy parameter is altered according to some deterministic rule, *adaptive adaptation* 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 adaptation* where the parameter to be adapted is encoded into the chromosomes and undergoes genetic operations (hence, also called *co-evolution*).

2.2 Adaptation in Mutation

Since mutation is one of the primary genetic operations in GAs, adaptation in mutation has long been studied and there have been many results (Bäck, 1997; Eiben, Hinterding, and Michalewicz, 1999). Generally speaking, adaptation in mutation happens in two levels.

In the top level, the ratio between mutation and crossover is adapted during the run of a GA. Davis (1989) proposed that the GA can select from a set of operators to perform on a chosen parent, each with a fixed probability. Julstrom (1995) adaptively adapted the ratio between mutation and crossover based on their performance. Corne *et al* (1994) 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 (1998) extended the COBRA method by co-evolving the mutation and crossover probabilities (one-normalized real numbers) with each individual.

In the bottom level, the probability of mutation is adapted during the run of a GA, uniformly or non-uniformly over each locus. Fogarty (1989) used deterministic schemes decreasing p_m over time and over the loci exponentially. Hesser and Männer (1991) derived a general expression of deterministically changing mutation probability 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, and t is the time (generation counter). Bäck (1992) 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.

3 Statistics-Based Adaptive Non-Uniform Mutation

3.1 Description of SANUM

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 and encoding scheme, e.g., whether introns are inserted (Levenick, 1995). During the running of a GA, a gene locus is called *1-inclined* if the frequency of 1's in its alleles over the population tends to increase (to the limit of 1.0) with time (generation), *0-inclined* if the frequency of 1's 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, 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 1's 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 1's in the alleles in a locus over the population (equivalently we can use the frequency of 0's as the argument) to calculate corresponding mutation probability of that locus. The frequency of 1's 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 1's 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:

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

where $|.|$ is an absolute function, P_{max} and P_{min} are the maximum and minimum allowable mutation prob-

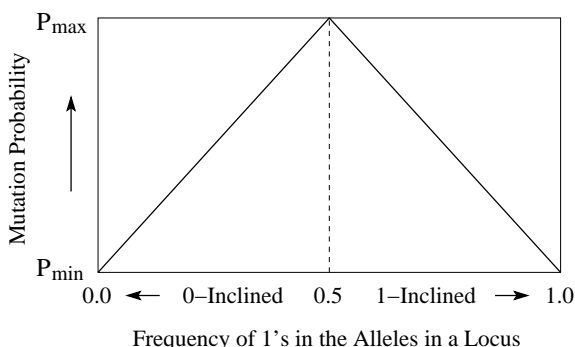abilities for a locus respectively, e.g., $P_{max} = 1/l$ and $P_{min} = 10^{-5}$.

Figure 1: Calculating function for the mutation probability for a gene locus.

Now during the evolution of the GA, after a new population has been generated, we first calculate $f_1(i, t)$ (hence $p_m(i, t)$) for each gene locus over the population. Then we can perform SANUM operations.

3.2   Discussions on SANUM

According to the classification of adaptation for GAs reviewed in section 2, SANUM belongs to the class of adaptive adaptation 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 each locus, hence the name Statistics-based Adaptive Non-Uniform Mutation (SANUM).

SANUM is much simpler than those self-adaptation mechanisms that add one extra value per bit and co-evolve these values with each individual. With SANUM, what we add to traditional bit flip 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 extra statistics added is well rewarded in the sense of computational complexity at two aspects: the number of mutations for a population and the number of bit flippings for each mutation operation. For a population of size N at generation t, on the average traditional bit flip mutation mutates $N \times (1 - (1 - p_m)^l)$ individuals while SANUM mutates $N \times (1 - \prod_{i=1}^{i=l}(1 - p_m(i, t)))$ individuals. For each mutation operation on an individual, the number of bit flippings is $l \times p_m$ on the average with traditional bit flip mutation and $\sum_{i=1}^{i=l} p_m(i, t)$ with SANUM.

When the population is randomly initialized, the frequency of 1's in the alleles in each locus is statistically about 0.5 and hence $p_m(i, t) = P_{max}$, which, assuming $P_{max} = p_m$, gives on the average $N \times (1 - (1 - p_m)^l)$ mutations for a population and $l \times p_m$ flippings for each mutation operation with SANUM. However, with the progress of the GA, 1-intrinsic and 0-intrinsic loci tend to converge to 1 and 0 respectively and their mutation probabilities decrease according to equation (1). This results in both reduced number of mutations over a population (i.e., $N \times (1 - \prod_{i=1}^{i=l}(1 - p_m(i, t))) < N \times (1 - (1 - p_m)^l)$) and reduced number of flippings for each mutation operation (i.e., $\sum_{i=1}^{i=l} p_m(i, t) < l \times p_m$) with SANUM. And as the population converges, with traditional bit flip mutation, in fact, fewer and fewer offsprings generated by mutating those converged loci will survive the next generation. That is, many mutation operations are wasted on those converged loci. Most of these wasted mutations are saved by SANUM through adaptively decreasing $p_m(i, t)$ to P_{min} for those converged loci.

4   The Test Problems

4.1   The Max Ones Problem

The Max Ones problem 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 (Forrest and Mitchell, 1992) are devised to investigate GA's performance with respect to schema processing and recombination in an idealized form. Royal Road functions R_1 and R_2 contain tailor-made building blocks (schemas) based on 64-bit binary strings. 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. 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 (De Jong, Potter and Spears, 1997) 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 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 used the same parameters as in (DeJong, Potter and Spears, 1997). We fixed V to 100 and L to 3. The number of clauses C was varied from 200 (low epistasis) to 1200 (medium epistasis) to 2400 (high epistasis).

5 Experiment Study

5.1 Design of Experiment

In order to test SANUM, in this experiment study we compared it with traditional bit flip mutation and combined them with commonly used 2-point and 0.5 uniform crossover operators. For each experiment of combining mutation (traditional bit flip mutation or SANUM), crossover (2-point or 0.5 uniform crossover) and each test problem, 100 independent runs were executed. 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. In all the experiments, the fitness proportionate selection with the stochastic universal sampling (Baker, 1987) and elitist model (De Jong, 1975) was used in the GA, the crossover probability was fixed to typical value 0.6, and the population size was set to 100 for each run.

For traditional bit flip mutation the probability of mutation p_m was set to $1/l$ while for SANUM p_m varied adaptively between $P_{max} = 1/l$ and $P_{min} = 10^{-5}$ for each locus according to equation (1). For each run, we recorded the best-so-far fitness every 100 evaluations. Here, only those chromosomes changed by crossover and mutation operations were evaluated and counted into the number of evaluations. Each experiment result was averaged over 100 independent runs.

5.2 Experiment Results

The experiment results on different test problems are shown in from Figure 2 to Figure 7 respectively. From these figures, it can be seen that SANUM outperforms traditional bit flip mutation operator on all the test problems quite clearly except on royal road functions R_1 and R_2. On royal road functions R_1 and R_2 (see Figure 3 and Figure 4), during the early stage of searching, GA with traditional bit flip mutation is better than GA with SANUM. However, after certain evaluations, when the GA has built up some useful schemas SANUM outperforms traditional bit flip mutation because SANUM efficiently avoids mutating those converged loci.
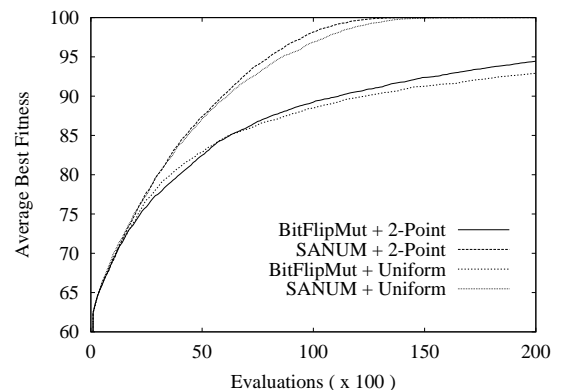

Figure 2: Average best-so-far curves against evaluations on Max Ones problem.

From above experiment results, it can also be seen that mutation operator does matter with respect to GA's performance. For the set of tested problems, SANUM is obviously better than traditional bit flip mutation, while uniform crossover and 2-point crossover show much less difference with respect to GA's performance except for when combined with SANUM on royal road functions. This observation shows that developping good mutation operator profoundly helps improving GA's performance.
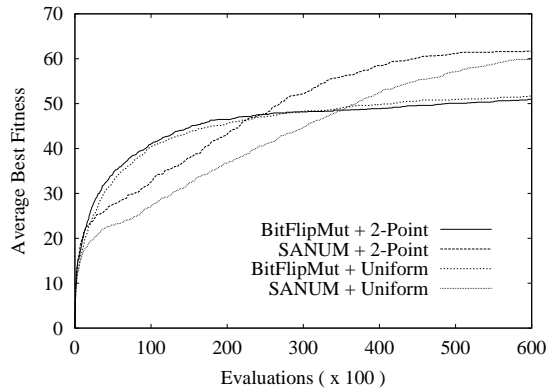
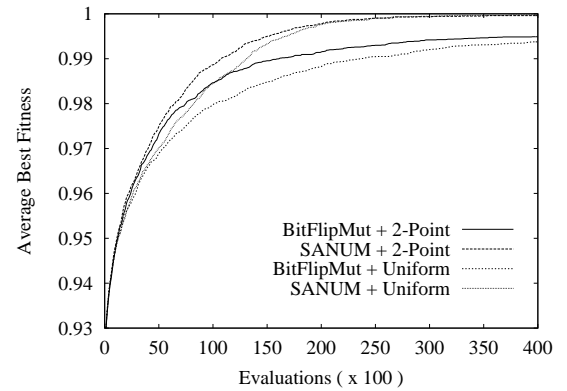Figure 3: Average best-so-far curves against evaluations on royal road function R_1.

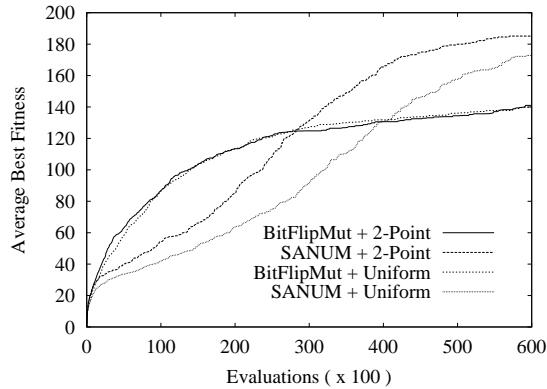Figure 5: Average best-so-far curves against evaluations on L-SAT problems with low epistasis.

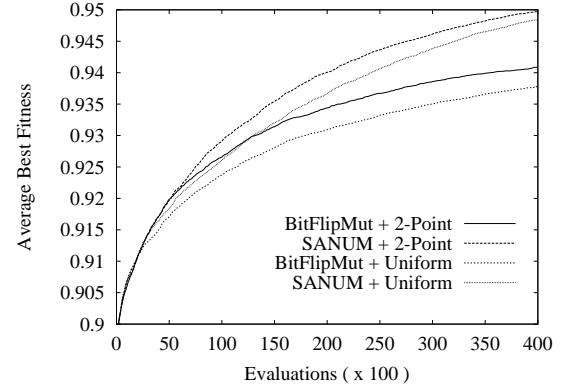Figure 4: Average best-so-far curves against evaluations on royal road function R_2.

Figure 6: Average best-so-far curves against evaluations on L-SAT problems with medium epistasis.

6 Conclusions

In this paper, a new statistics-based adaptive non-uniform mutation operator, SANUM, is proposed. 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 statistics information of 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 converged loci SANUM can save mutations wasted on them.

The preliminary experiment results of this study show that SANUM performs better than traditional bit flip mutation operator on a set of typical GA test problems. The experiment results indicate that SANUM may be a good candidate mutation operator for GAs. Since SANUM works at the bottom-level of mutation,

it can be easily combined into other adaptation techniques for mutation and can act as the basis for analyzing and designing new related algorithms.

In this study, for the sake of simplification the 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

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.

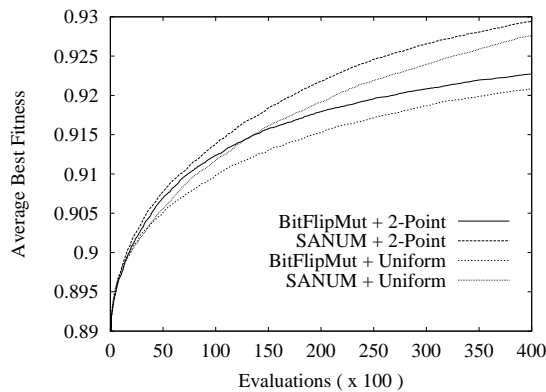T. Bäck (1992). Self-Adaptation in Genetic Algo-

Figure 7: Average best-so-far curves against evaluations on L-SAT problems with high epistasis.

rithms. In F. J. Varela and P. Bourgine (eds.), *Proc. of the 1st European Conf. on Artificial Life*, 263-271. MIT Press.

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.

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.

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 Publisher.

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.

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.

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

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.

S. Forrest and M. Mitchell (1992). Relative building-block fitness and the building-block hypothesis. In D. Whitley (ed.), *Foundations of Genetic Algorithms 2*. San Mateo, CA: Morgan Kaufmann.

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

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

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.

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

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.

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.

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.

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 Publisher.

J. E. Smith and T. C. Forgerty (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.

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