# Adaptive mutation rate control schemes in genetic algorithms

**Dirk Thierens**
Institute of Information and Computing Sciences
Utrecht Univerisity, The Netherlands

**Abstract - The adaptation of mutation rate parameter values is important to allow the search process to optimize its performance during run time. In addition it frees the user of the need to make non-trivial decisions beforehand. Contrary to real vector coded genotypes, for discrete genotypes most users still prefer to use a fixed mutation rate. Here we propose two simple adaptive mutation rate control schemes, and show their feasibility in comparison with a fixed mutation rate, a self-adaptive mutation rate and a deterministically scheduled dynamic mutation rate.**

*Keywords*—**Adaptive mutation, Constant gain adaptive mutation, Declining adaptive mutation, Counting ones problem, Zero/one multiple knapsack problem.**

## I. Introduction

Novice users of Genetic Algorithms are often puzzled by the choices they have to make for a number of parameters before being able to run their GA. In this study we will focus on the probability of mutation. Previous studies have shown that varying the mutation probability is preferable to a fixed constant mutation rate ([2],[3],[5],[8],[12],[15],[18],[20]). Unfortunately most novice users prefer to use a fixed mutation rate due to its simplicity as compared to for instance the somewhat complicated self-adaptive mutation rate control scheme. In this paper we propose two simple adaptive mutation rate control schemes.

Evolutionary algorithms with non-fixed parameter settings can be divided into three subclasses ([4],[14]):

1. **Dynamic parameter control:** The dynamic schemes typically prescribe a deterministically decreasing schedule over the number of generations. Fogarty ([8]) experimentally studied a dynamical mutation rate control for genetic algorithms and he proposed to use a schedule that decreases exponentially over the number of generations:

$$p_m(t) = \frac{1}{240} + \frac{0.11375}{2^t}$$

The results obtained showed an increased performance, but the experimental setup is rather specific. Hesser and Männer ([12], [13]) derived a more general expression for decreasing mutation rate parameter control schedule:

$$p_m(t) = \sqrt{\frac{c_1}{c_2}} \frac{exp(-c_3 t/2)}{n\sqrt{l}}$$

with $l$ the stringlength and $n$ the population size. Unfortunately the constants $c_i$ can only be roughly estimated for simple problems. Bäck and Schütz ([5]) also tested a deterministically decreasing scheme:

$$p_m(t) = (2 + \frac{l-2}{T-1}\, t)^{-1}$$

with $T$ the number of generations the GA is run. Excellent experimental results where obtained for hard combinatorial optimization problems, and we will use their algorithm for comparison latter in this work.

2. **Adaptive parameter control:** During the optimization process we can extract feedback information on how well the search is going and use this information to adapt the parameter control values. Rechenberg's "$1/5$ success rule" is an early example of this scheme ([21]). The rule tries to keep the ratio of successful mutations to all mutations close to $1/5$ by increasing the step size when the ratio is larger than $1/5$, and decreasing the step size when the ratio is less than $1/5$. Although the use of feedback from the ongoing search process seems a useful approach, it has not been studied much in the Evolutionary Computation literature.
3. **Self-adaptive parameter control:** The most evolutionary solution to the parameter control problem is the principle of self-adaptation where the parameter values are modified by evolving them. Each individual has control parameters encoded into its genotype. It is important to notice that there is only an indirect link between good parameter control values and highly fit individuals. Self-adaptation provides no direct feedback mechanism to good or bad parameter control values. The idea is that good parameter values will provide an evolutionary advantage to the individual it belongs to and therefore it will itself proliferate in the population. Self-adaptation has been successfully applied in case of continuous optimization problems with Evolutionary Strategies and Evolutionary Programming ([10],[22]). Bäck

0-7803-7282-4/02/$10.00 ©2002 IEEE

and Schütz ([5]) designed a self-adaptive scheme for binary strings following the principles from the continuous domain. They proposed a logistic transformation:

$$p_m(t+1) = (1 + \frac{1-p_m(t)}{p_m(t)}\ exp(-\gamma\ \mathbb{N}(0,1)))^{-1}$$

such that the mutation probability $p_m(t+1)$ is distributed according to a logistic normal distribution with probability density function

$$\mathbb{P}_{p_m(t+1)}(x) = \frac{1}{\sqrt{2\pi}\gamma x(1-x)}\ exp(\Xi)$$

with

$$\Xi = \frac{-(\ln\frac{x}{1-x} - \ln\frac{p_m(t)}{1-p_m(t)})^2}{2\gamma^2}$$

The learning rate $\gamma$ controls the size of the adaptation steps and in [5] a value of $\gamma = 0.22$ is chosen. The self-adaptive scheme performs rather well and we will also include it in the comparison further on.

Compared to the fixed mutation rate schemes most non-static parameter setting strategies have a better performance. Unfortunately there are some drawbacks. The dynamic parameter control scheme requires the user to devise a schedule specifying the rate at which the parameter is typically decreased. The self-adaptive scheme does not need such a specific schedule and is conceptually quite appealing to evolutionary algorithmists. Unfortunately it is rather complicated to explain to novice users, and as a result they usually prefer the simple fixed mutation rate scheme. Contrary to Evolutionary Strategies the use of a fixed mutation rate with genetic algorithms - like the often advised $p_m = 1/stringlength$ - does seem to give satisfactory results, which adds to the novice user's reluctance to apply the more elaborate schemes.

In the next section we propose two simple adaptive mutation rate control schemes: the *constant gain* and the *declining* adaptive mutation scheme. Section 3 tests their feasibility in a limited experimental comparison. The goal of this paper is not to extensively analyse and test these adaptive mutation schemes, but merely to find out whether it is worthwhile to investigate simple adaptive parameter control schemes at all.

## II. Adaptive Mutation Rate Parameter Control

### A. Constant gain adaptive mutation rate control

The first scheme is inspired by the stochastic Manhattan learning algorithm from the field of stochastic approximation ([17]). Stochastic learning algorithms provide recursively refined estimates of optimal model parameters. Their general form is $w(t+1) = w(t) + \mu(t)H(w(t), x(t))$, where $w(t)$ is the parameter estimate at the recursive time step $t$, $\mu(t)$ is the learning rate, $x(t)$ is the input data at time step $t$, and $H(w,x)$ represents the learning rule. The learning rate $\mu(t)$ is either a constant factor or a decreasing function, resulting in resp. constant gain or adaptive gain algorithms. The stochastic Manhattan learning algorithm adapts the parameter in proportion to the sign of the gradient of some error function $E(w,x)$, resulting in

$$w(t+1) = w(t) + \mu(t) - sign\left[\frac{\delta E(w(t), x(t))}{\delta w}\right].$$

Stochastic Manhattan learning algorithms are very robust recursive learning algorithms using fixed parameter changes. In evolutionary algorithms we do not have a gradient of an error function to guide the search, but a somewhat equivalent procedure to Manhattan learning can be devised as follows. Suppose our current mutation parameter value is $p_m(t)$, and we generate two new individuals by mutating the current one respectively with a mutation rate $p'_m(t) = \omega p_m(t)$, and with $p'_m(t) = p_m(t)/\omega$, where $\omega$ is a constant called the exploration factor. Evaluating the mutated individuals and comparing their fitness value gives us a rough - and very noisy - indication of whether we should increase or decrease the current mutation rate. Similar to constant gain Manhattan learning we could add or subtract a fixed factor to the current mutation probability. We prefer however to have a modification in proportion of the current value and therefore use a multiplicative constant learning factor $\lambda$. If the process converges to the optimal mutation probability we should make use of this value so in addition to the two mutated individuals with a larger and smaller mutation probability we also generate a new individual with the current value.

Formally let us notate the mutation of the binary string $\mathbf{x}$ of length $\ell$ with mutation probability $p_m$ generating offspring $\mathbf{x}'$ with new mutation probability $p'_m$ as $\mathbb{M}(\mathbf{x}, p_m) \rightarrow (\mathbf{x}', p'_m)$. We bound the range of possible mutation values between $[1/\ell ... 0.5]$. Values generated outside this interval are given the boundary value. The constant gain adaptive mutation rate parameter control scheme then becomes:

---

**Constant gain adaptive mutation scheme**

1. Mutate the current individual $(\mathbf{x}, p_m)$:

$$\mathbb{M}(\mathbf{x}, p_m/\omega) \rightarrow (\mathbf{x_1}, p_m/\lambda)$$

$$\mathbb{M}(\mathbf{x}, p_m) \rightarrow (\mathbf{x_2}, p_m)$$

$$\mathbb{M}(\mathbf{x}, \omega p_m) \rightarrow (\mathbf{x_3}, \lambda p_m)$$

2. Select the fittest individual of $\{(\mathbf{x}, p_m), (\mathbf{x_1}, p_m/\lambda), (\mathbf{x_2}, p_m), (\mathbf{x_3}, \lambda p_m)\}$

---

Note that the learning-factor $\lambda$ and the exploration-factor $\omega$ typically have different values ($\lambda, \omega > 1$). To reduce the

noise in the evaluation of the direction of modification a rather large value is needed, while the learning-factor should remain smaller to avoid wild oscillations of the learning process. Typical appropriate values are $\lambda = 1.1$ and $\omega = 1.5$.

B. Declining adaptive mutation rate control

The second adaptive mutation rate control scheme we propose here is a variant of the constant gain method. As will be discussed in the next section it seems that the constant gain scheme should benefit from a more aggressive step size. Increasing the learning factor however results in a too widely fluctuating parameter value. The declining adaptive rule aims for a more aggressive step size while retaining a rather smooth dynamics. Each time an individual creates an offspring by mutation its mutation probability is decreased by a small factor, the declination factor $\gamma$. In addition the exploration towards lower mutation probability values is replaced by an exploration towards higher values. The step size adaptation is also made more aggressively than in the constant gain method. Formally the algorithm is defined as:

---

**Declining adaptive mutation scheme**

1. Mutate the current individual $(\mathbf{x}, p_m)$:

$$\mathbb{M}(\mathbf{x}, \omega p_m) \rightarrow (\mathbf{x_1}, \lambda p_m)$$
$$\mathbb{M}(\mathbf{x}, p_m) \rightarrow (\mathbf{x_2}, p_m)$$
$$\mathbb{M}(\mathbf{x}, \omega p_m) \rightarrow (\mathbf{x_3}, \lambda p_m)$$

2. Decrease the mutation probability of the parent:

$$(\mathbf{x}, p_m) \rightarrow (\mathbf{x}, \gamma p_m)$$

3. Select the fittest individual of $\{(\mathbf{x}, \gamma p_m), (\mathbf{x_1}, \lambda p_m), (\mathbf{x_2}, p_m), (\mathbf{x_3}, \lambda p_m)\}$

---

Note that $\omega, \lambda > 1$, while $\gamma < 1$. Typical appropriate values are $0.9 \leq \gamma < 1$, and $\omega = \lambda = 2.0$.

## III. Experimental Results

In this section we perform some limited experiments to check the feasibility of the proposed simple adaptive mutation rate control schemes. First we measure the dynamic behaviour on the 'fruit-fly' of genetic algorithm research - the Counting Ones problem. For this function the optimal mutation probability for each individual is known ([3],[6],[18]).

A. Counting Ones problem

The Counting Ones problem is defined as

$$F(\mathbf{x}) \rightarrow \mathbb{Z} : F(\mathbf{x}) = \sum_{i=1}^{l} x_i$$

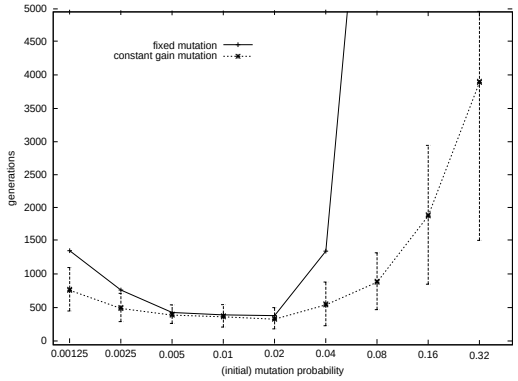


Fig. 1. Generations to optimal solution for Counting Ones problem ($\ell = 100$) with fixed and constant gain adaptive mutation rate (note the x-logscale).

where $\mathbf{x}$ is a binary string of length $l$, $\mathbf{x} = x_1 x_2 \ldots x_l$ ($x_i \in \{0, 1\}$).

Figure 1 shows a first advantage of using an adaptive mutation rate versus a fixed mutation rate (results are averaged over 25 runs, vertical bars represent 1 standard deviation above and below the average value; similar deviations are obtained for the fixed mutation rate scheme but are not plotted for clarity). The number of generations needed to converge to the optimal solution is much more sensitive to the choice of the (initial) mutation probability for a fixed mutation rate scheme than for instance for the constant gain adaptive mutation scheme. For a fixed mutation rate the range of mutation probabilities where convergence goes well is not very wide, and once outside the interval $[0.005 \ldots 0.02]$ the number of generations needed to reach the optimal solution increases rapidly. For the constant gain adaptive mutation rate control the range of initial mutation probabilities can be quite large: between $[0.0025 \ldots 0.04]$ performance is satisfactory and once outside the interval the required number of generations increases far less dramatically than with the fixed mutation rate scheme (note that the initial mutation probabilities are plotted on a logscale).

In table 2 the average number of function evaluations and generations are shown for 6 different mutation strategies. The self-adaptive, constant gain adaptive, and declining adaptive are run with a (1+3) selection strategy. The fixed, deterministic, and optimal scheme are run with a (1+1) selection strategy, since they do not need the extra offspring for the Counting Ones problem. Unfortunately this makes the comparison somewhat harder to make. It is important to realize though that for more difficult functions all algorithms will need a large offspring population, eliminating the different population sizes seen here. The next section will illustrate this.

The constant gain adaptive mutation rate scheme is run with learning-factor $\lambda = 1.1$ and exploration-factor $\omega = 1.5$. The

| mutation schemes | average number fct. evals. (gens.) | std. dev. number fct. evals. (gens.) |
|---|---|---|
| (1) Self-adaptive | 2082 (694) | 2223 (741) |
| (2) Constant Gain | 2499 (833) | 1458 (486) |
| (3) Declining (0.9) | 918 (306) | 231 (77) |
| (4) Declining (0.5) | 714 (238) | 198 (66) |
| (5) Fixed | 575 | 181 |
| (6) Deterministic | 1272 | 211 |
| (7) Optimal | 583 | 157 |

Fig. 2. Number of function evaluations (resp. generations) for different mutation rate control schemes on the Counting Ones problem

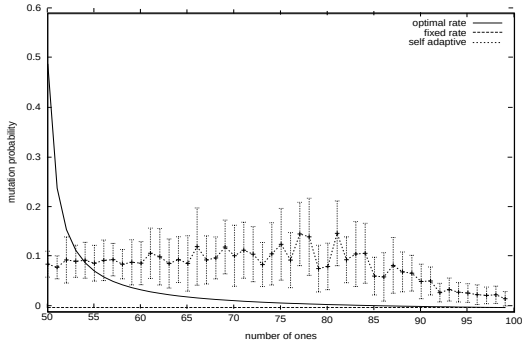


Fig. 3. Distribution of the mutation probabilities applied with self-adaptive mutation rate control compared with optimal values.

declining adaptive mutation rate scheme is run with declination factor $\gamma = 0.9$ and alternatively with $\gamma = 0.5$, while in both cases $\lambda = \omega = 2.0$.

For the Counting Ones problem the optimal mutation probability can be approximated very closely by:

$$p_m^{OPT}(\mathbf{x}) \approx \frac{1}{2 + (F(\mathbf{x}+1) - \ell}$$

Figures 3, 4, 5, 6, and 7 show the mutation probability applied for each fitness value when optimizing a 100-bit long string in comparison with the optimal and fixed $(1/stringlength)$ value. Results are averaged over 100 runs, with the error bars indicating plus or minus one standard deviation. It is important to note that most of the time all algorithms are trying to improve strings with high fitness values, so most of the time is spend at the right part of the figures. The most striking observation is the slow adaptive behaviour of the self-adaptive and constant gain mutation rate schemes. This lack of responsiveness even necessitated to start from an initial mutation probability $p_m(0) = 0.1$ instead of the optimal $p_m(0) = 0.5$. Of course it would be possible to increase the responsiveness by increasing the parameters controlling the step size. Experiments show however that this has severe negative side effects by causing erratic fluctuations in the mutation probability value.

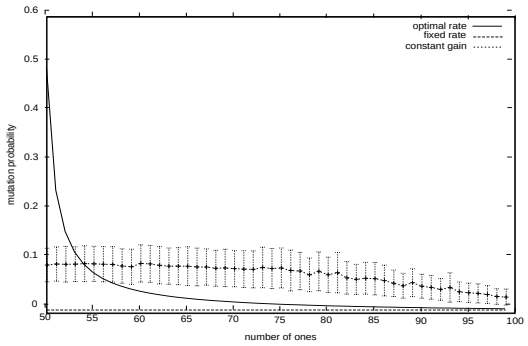


Fig. 4. Distribution of the mutation probabilities applied with constant gain adaptive mutation compared with optimal values.

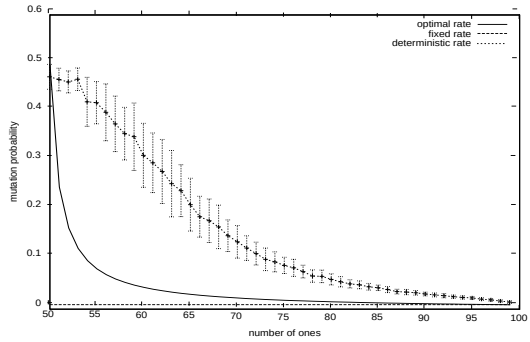


Fig. 5. Distribution of the mutation probabilities applied with deterministic dynamic mutation compared with optimal values.

The dynamic and declining adaptive schemes better match the optimal rate, even when started from the high initial value $p_m(0) = 0.5$. Their dynamic behaviour is also quite smooth. It should be noted that even a declination factor $\gamma = 0.5$ does not match the extremely rapid decline of the optimal scheme. Naturally this is only an optimal strategy for the unimodal Counting Ones problem.

## B. Zero/one multiple knapsack problem

A second experiment was done for a hard combinatorial problem: the zero/one multiple knapsack problem ([16]). The problem is a generalization of the simple knapsack problem. We are given $m$ knapsacks with capacities $c_1, c_2, \ldots, c_m$ and $n$ objects each with a profit $p_i$. Every object $i$ has a weight $w_{ij}$ when it is included in the knapsack $j$. Note that contrary to the simple knapsack problem the weights of the objects are not constant but instead they depend on the knapsack they have been allocated to. Objects are either allocated to all knapsacks or they are not allocated at all. The goal is to find an allocation of the objects to the knapsacks such that the

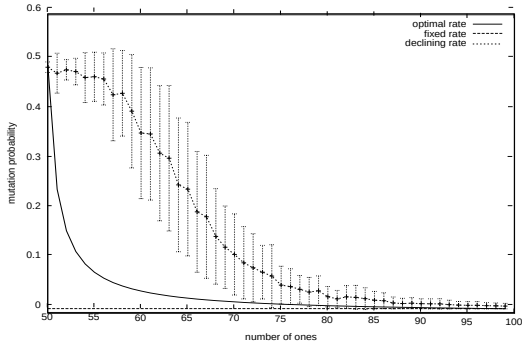


Fig. 6. Distribution of the mutation probabilities applied with declining adaptive mutation ($\gamma = 0.9$) compared with optimal values.

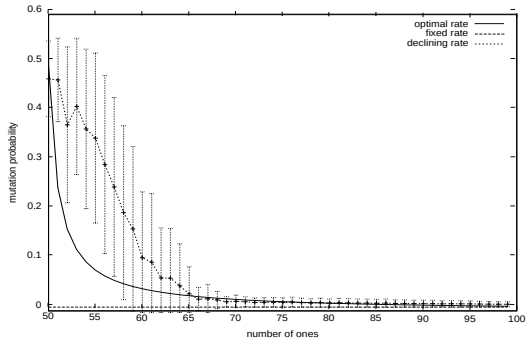


Fig. 7. Distribution of the mutation probabilities applied with declining adaptive mutation ($\gamma = 0.5$) compared with optimal values.

total profit is maximised. Calling $\mathbf{x} = x_1 x_2 \ldots x_n$ (where $x_i \in \{0, 1\}$) the allocation string, the goal can be formalised as maximising $\sum_{i=1}^{n} x_i p_i$ constrained by the knapsack capacities, or $\sum_{i=1}^{n} w_{ij} x_i \leq c_j$. Whenever at least one knapsack is overfilled the string represents an infeasible solution. As in [16] we include a penalty term in the fitness function that becomes larger the farther the solution is from feasibility. The fitness function to be maximised is then:

$$F(\mathbf{x}) = \sum_{i=1}^{n} p_i x_i - s.max(p_i)$$

where $s$ is the number of overfilled knapsacks: $s = |\{j \mid \sum_{i=1}^{n} w_{ij} x_i > c_j\}|$.

In our experiments we take the multiple knapsack problem called "weing7-1055" which has 105 objects to be allocated to 2 knapsacks (specific weights and capacities can be found at [19]). This particular knapsack problem was also studied in [5] with a fixed mutation rate ($p_m = 1/stringlength$), a self-adaptive mutation schedule, and a deterministic dynamic schedule - as explained in the Introduction.

We have implemented these mutation schemes and compared them with our simple adaptive mutation rate algorithms

| fitness | (1) | (2) | (3) | (4) | (5) |
|---|---|---|---|---|---|
| 1095445 | - | - | - | - | - |
| 1095382 | 2 | 5 | 5 | 4 | 3 |
| 1095357 | - | - | 1 | 2 | - |
| 1095295 | - | - | - | 1 | - |
| 1095266 | - | - | - | 1 | - |
| 1095264 | - | 1 | - | 2 | 1 |
| 1095232 | - | - | - | - | 2 |
| 1095207 | - | - | - | 2 | 2 |
| 1095206 | - | - | 2 | - | 2 |
| 1095141 | - | - | - | 1 | 3 |
| 1095137 | - | 1 | - | - | 2 |
| 1095132 | - | - | - | 1 | - |
| 1095114 | - | - | - | - | - |
| 1095112 | - | - | - | - | 1 |
| 1095081 | - | 1 | - | 1 | - |
| 1095065 | - | - | - | - | - |
| 1095035 | 1 | - | - | - | - |
| runs / 30 | 3/30 | 8/30 | 8/30 | 15/30 | 16/30 |

Fig. 8. Knapsack problem: best results out of 30 runs (1) Fixed mutation rate (2) Self-adaptive mutation rate (3) Constant gain adaptive mutation rate (4) Declining adaptive mutation rate (5) Deterministically scheduled dynamic mutation rate

using a (16+96) selection strategy, without applying recombination. At each generation every individual generates 6 children by mutation. As opposed to the (1+3) selection strategy the children no longer compete directly with their single parent. Instead they compete in the entire pool of parents and offspring. All algorithms are run for 2000 generations (a value advised in [5] for the deterministic schedule). The constant gain adaptive mutation scheme is run with the learning factor $\lambda = 1.1$ and the exploration factor $\omega = 1.5$. The declining mutation rate control has a declination factor $\gamma = 0.98$, and $\lambda = \omega = 2.0$. The initial mutation probability for each individual is randomly taken from the interval $[1/stringlength \ldots 0.1]$. Every algorithm is run 30 times.

Figure 8 shows the number of runs that achieved a fitness value of at least 1095000, while table 9 shows the average and standard deviation over all 30 runs. The fixed mutation rate scheme performs worst, both in terms of average fitness and in the number of solutions found with fitness above 1095000. The self-adaptive and constant gain adaptive mutation rate control scheme perform very similar. Best performance however is obtained with the declining adaptive mutation rate scheme and the deterministically scheduled dynamic mutation rate scheme.

| mutation schemes | average fitness | standard deviation |
|---|---|---|
| (1) Fixed | 1092168 | 2292 |
| (2) Self-adaptive | 1092865 | 2007 |
| (3) Constant gain | 1093025 | 2073 |
| (4) Declining | 1094577 | 1048 |
| (5) Deterministic | 1094730 | 953 |

Fig. 9. Average fitness values over 30 runs for the 5 mutation rate control schemes on the knapsack problem.

## IV. Discussion

Space limitations prevent an extensive discussion here. Two comments however need to be made.

- Both the deterministic dynamic schedule and the declining adaptive mutation rate scheme need a parameter value set to specify how fast the mutation rate will drop (resp. the number of generations run and the declination factor $\gamma$). This is somewhat analogous to the cooling schedule of simulated annealing, and further research needs to determine how sensitive performance is to this parameter.
- In this paper we have not considered the use of crossover. Recombination would typically make large adjustments during the initial generations, which is exactly where the dynamic and declining adaptive schemes differ the most from the self-adaptive, the constant gain adaptive, and the fixed mutation schemes. It is therefore reasonable to expect that the use of crossover would level out the differences between all mutation rate control schemes. In [20] some experiments have been reported that seem to confirm this. A thorough analysis is however still lacking and is no doubt an interesting topic to investigate.

## V. Conclusion

We have proposed two simple adaptive parameter control schemes for adapting the mutation probability with binary genotypes: the *constant gain* and the *declining* adaptive mutation scheme. The constant gain adaptive mutation method obtains feedback of the search process by testing the results of an increased and decreased mutation probability, and adapts the mutation parameter with a constant multiplicative gain. Limited experimental results indicate that the performance is comparable to self-adaptive mutation schemes. The declining adaptive mutation rate control matches more the dynamical behaviour of the deterministically scheduled dynamic mutation rate control scheme. Performance-wise they also are comparable. This paper showed the feasibility of these simple adaptive approaches, suggesting further experimental and theoretical analysis is worthwhile to pursue.

## References


[1] Angeline P. Adaptive and Self-adaptive Evolutionary Computation. *Computational Intelligence: A Dynamic System Perspective*, eds. Palaniswami et al., pp. 152-161, IEEE Press, 1995.

[2] Bäck T. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. *Proc. of the 2nd Parallel Problem Solving from Nature*, pp. 85-94, Elsevier, 1992.

[3] Bäck T., Optimal mutation rates in genetic search, *Proc. of the Fifth International Conference on Genetic Algorithms*, pp. 2-8, Morgan Kaufmann, 1993.

[4] Bäck T., An overview of parameter control methods by self-adaptation in evolutionary algorithms. *Fundamenta informaticae 34*, 1-15, 1998.

[5] Bäck T. and Schütz M., Intelligent Mutation Rate Control in Canonical Genetic Algorithms, *Proc. of the International Symposium on Methodologies for Intelligent Systems*, pp. 158-167, 1996.

[6] Bremermann H.J., Rogson M., and Salaff S., Global properties of evolution processes, *Natural Automata and Useful Simulations*, Spartan Books, 1966. Reprinted in *Evolutionary Computation: The Fossil Record*, ed. D.B. Fogel, IEEE Press, 1998.

[7] Davis L., Adapting Operator Probabilities in Genetic Algorithms, *Proc. of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, 1989.

[8] Fogarty T., Varying the Probability of Mutation in the Genetic Algorithm, *Proc. of the Third International Conference on Genetic Algorithms*, pp. 104-109, Morgan Kaufmann, 1989.

[9] Fogel D., *Evolving artificial intelligence,* PhD Diss. UCSD, 1992.

[10] Fogel D., *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE Press, NJ, 1995.

[11] Hansen N. and Ostermeier A., Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation, *Proc. of the 1996 IEEE Int. Conf. on Evolutionary Computation*, pp. 312-317, IEEE Press, 1996

[12] Hesser J. and Männer R., Towards an Optimal Mutation Probability in Genetic Algorithms, *Proc. of the 1st Parallel Problem Solving from Nature*, pp. 23-32, Springer, 1991.

[13] Hesser J. and Männer R., Investigation of the m-heuristic for optimal mutation probabilities, *Proc. of the 2nd Parallel Problem Solving from Nature*, pp. 115-124, Elsevier, 1992.

[14] Hinterding R., Michalewicz Z. and Eiben G., Adaptation in Evolutionary Computation: A Survey, *Proc. of The 1997 IEEE Conference on Evolutionary Computation*, pp. 65-69, IEEE Press, 1997.

[15] Jansen T. and Wegener I., On the Choice of the Mutation Probability for the (1+1) EA, *Proc. of the 6th Parallel Problem Solving from Nature*, pp. 89-98, Springer, 2000.

[16] Khuri S., Bäck T. and Heitkötter J., The Zero/One Multiple Knapsack Problem and Genetic Algorithms, *Proc. of the 1994 ACM Symposium of Applied Computation*, pp. 188-193, ACM Press, 1994.

[17] Kushner H. and Clark D., *Stochastic Approximation for Constrained and Unconstrained Systems*, Springer, 1978.

[18] Mühlenbein H., How Genetic Algorithms Really Work, *Proc. of the 2nd Parallel Problem Solving from Nature*, pp. 15-25, Elsevier, 1992.

[19] MP-testdata: SAC-94 Suite of 0/1-Multiple-Knapsack Problem Instances, http://elib.zib.de/pub/Packages/mp-testdata/ip/sac94-suite

[20] Ochoa G., Harvey I. and Buxton H., On Recombination and Optimal Mutation Rates, *Proc. of the Genetic and Evolutionary Computation Conference*, pp. 13-17, Morgan Kaufmann, 1999.

[21] Rechenberg I., *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Frommann, 1973.

[22] Schwefel H.-P., *Evolution and Optimum Seeking*, Wiley, NY, 1995.

[23] Smith J. and Fogarty T., Self Adaptation of Mutation Rates in a Steady State Genetic Algorithm, *Proc. of The 1996 IEEE Conference on Evolutionary Computation*, pp. 318-323, IEEE Press, 1996.