An Analysis of the Behaviour of Simplified Genetic Algorithms on Trap Functions

Siegfried Nijssen, Thomas Bäck

Abstract

Theoretical analysis results for the convergence velocity of certain mutation-selection based variants of genetic algorithms are presented for a class of bimodal unitation functions called trap functions. This research bridges part of the gap between the existing convergence velocity analysis of strictly unimodal functions and global convergence results assuming the limit of infinite time. As a main result of this analysis, a new so-called (1:λ)-genetic algorithm is proposed, which generates offspring using individual mutation rates p_i. While a more traditional genetic algorithm using only one mutation rate is not able to find the global optimum of the trap function within an acceptable (non-exponential) time, the new algorithm overcomes these limitations. The analysis tools used for the analysis, based on absorbing Markov chains and the calculation of transition probabilities, are demonstrated to provide an intuitive and useful method for investigating the capabilities of genetic algorithms to bridge the gap between a local and a global optimum in bimodal search spaces.

Keywords

Evolutionary algorithm, convergence velocity, trap functions, absorption time, mutation.

I. Introduction

In the past decade, theoretical research on evolutionary algorithms has received significant attention, driven by the insight that their theoretical basis needs to be improved to facilitate most effective usage of these algorithms. Meanwhile, a lot of progress has been made especially with respect to the analysis of *convergence velocity* and *convergence reliability* of evolutionary algorithms.

Siegfried Nijssen is with the Leiden Institute of Advanced Computer Science (LIACS), Leiden University, Niels Bohrweg 1, 2333CA, The Netherlands. E-mail: snijssen@liacs.nl

Thomas Bäck is with the LIACS and NuTech Solutions GmbH, Martin-Schmeisser-Weg 15, D-44227, Dortmund, Germany. E-mail: baeck@nutechsolutions.de

The convergence velocity analysis, originally derived [Rec73], [Sch77] and subsequently refined for the analysis of evolution strategies (see [Bey01] for a full picture; [BA01] for an overview), is a general approach that has been transferred to genetic algorithms in the early 90s [Bäc92], [Müh92]. Convergence velocity is a local measure of progress of the algorithm from one iteration to the next, where progress is defined either in terms of the expected quality gain φ (i.e., objective function value improvement) or in terms of the expected change of distance D to the optimum:

$$\varphi = E[f(\vec{x}_{t+1}) - f(\vec{x}_t)] \quad , \quad D = E[\|\vec{x}^* - \vec{x}_t\| - \|\vec{x}^* - \vec{x}_{t+1}\|] \quad .$$

Here, $\vec{x}^*$ denotes the global optimum point of $f : M \to \mathbb{R}$, and $\vec{x}_t$ denotes the best (or a representative) individual of the population at generation t. A maximization task is assumed in this paper.

Typically, this type of analysis is used to characterize the behaviour of evolutionary algorithms for unimodal problems, i.e., their effectiveness as local optimizers. This type of analysis is useful to understand performance relative to other local optimization algorithms, to gain insight into the impact of parameter settings of the algorithm on convergence velocity, and to characterize the final stage of global optimization, when the algorithm ultimately converges to a solution. However, convergence velocity analysis has not yet been generalized to cover the multimodal case of objective functions with more than one optimum, which of course is the interesting case for practical applications of evolutionary algorithms.

At the other extreme, convergence reliability analysis deals with the question of global convergence of an evolutionary algorithm, meaning that the algorithm is guaranteed to find the global optimum. Global convergence results are general in the sense that they do not make strong assumptions about the objective function and typically assume unlimited time:

$$\lim_{t \to \infty} p(\vec{x}^* \in P(t)) = 1 \quad .$$

Here, $P(t)$ denotes the population maintained by the evolutionary algorithm at generation t and

$p(A)$ is the probability of the event A. Some of the first global convergence results for evolutionary algorithms have been presented for simple (1+1)-evolution strategies [Rec73] and subsequently been refined for population based strategies as well as non-elitist strategies [Rud97]. Concerning genetic algorithms, first proofs of global convergence were presented again in the early 90s [EAH91]. The global convergence type of analysis benefits from the generality of results (i.e., for all possible objective functions), but it is practically not exploitable as no finite expected time results are obtained.

In order to bridge the gap between convergence velocity results and convergence reliability results, it is a natural but difficult step to extend the convergence velocity analysis to multimodal objective functions and to explicitly analyze the time it takes the algorithm to converge to the global optimum rather than a local one. Of course, the results are expected to depend on the starting conditions as well as the specific parameter settings of the evolutionary algorithm.

The natural extension from the existing work for unimodal objective functions consists in bimodal problems, where just one local and a distinct global optimum exist in the search space and the regions of attraction of these two optima can be scaled such that it becomes harder or easier to find the global optimum. For real-valued objective functions, this test case was defined and experimentally investigated already more than 15 years ago [Gal85], [Gal89], demonstrating the importance of "soft selection" to bridge the gap between the local and global optimum. From a theoretical point of view, however, only very recently first results on specific bimodal test problems have been published [JW99], [RPB01]. Approaching the analysis from different perspectives, both papers focus on the advantage of crossover for reducing the time to find the global optimum or to bridge the gap between local and global optimum.

Here, we explore another piece of the puzzle by analyzing so-called trap functions, which have been designed as scalable, bimodal functions to challenge genetic algorithms. In contrast to the above mentioned studies, the analysis concentrates on simplified genetic algorithms using only mutation and selection, such as the (1+1)-GA, the (1,λ)-GA, and the (1+λ)-GA. This analysis continues earlier

work on a unimodal problem [Bäc92], [Müh92], [Bäc96] and concludes with the development of a new version of a genetic algorithm, the (1:λ)-GA, which generates each offspring with a different mutation rate. The resulting algorithm reduces the time to find the global optimum drastically by increasing the emphasis on exploration, such that the region of attraction of the local optimum can be left at any stage during the search.

In section II, the general tools for the theoretical analysis are introduced, the trap function is formalized, and the (1:λ)-GA is defined as a generalization of the (1+λ)- and (1,λ)-GA. Section III presents the numerical evaluation of theoretical results and a comparison to the experimentally observed behaviour of the genetic algorithms on the trap function. Our conclusions and an outline of further work are given in section IV.

II. Genetic Algorithms on Trap Functions

A. Prerequisites

Each individual in a canonical genetic algorithm is represented by a bitstring of length l: $\vec{x} \in \mathbb{B}^l$. The fitness function is a function which maps a bitstring to a real number, $f : \mathbb{B}^l \to \mathbb{R}$. We restrict ourselves to *unitation functions*, which are functions that entirely depend upon the number of ones in a bitstring and thus not on their position:

$$f(\vec{x}) = \hat{f}(u(\vec{x})) = \hat{f}(\sum_{k=1}^{l} x_k) \quad . \tag{1}$$

For any unitation function $\hat{f}$ with a domain $D = \{0, 1, \ldots, l\}$, three subsets can be computed for a given value $u \in D$:

$$I^+(u) = \{u' \in D | \hat{f}(u') > \hat{f}(u)\} \quad ; \tag{2}$$

$$I^=(u) = \{u' \in D | \hat{f}(u') = \hat{f}(u)\} \quad ; \tag{3}$$

$$I^-(u) = \{u' \in D | \hat{f}(u') < \hat{f}(u)\} \quad . \tag{4}$$

For every unitation value (the number of ones in a bitstring) there is a set of unitation values (and corresponding bitstrings) which have lower, equal and higher fitnesses. We do not mention the fitness function in our notation as this function is implicitly the same in all formulas.

In a genetic algorithm one bitstring can be transformed into another bitstring using an operator called *mutation*. With probability p_m this operator flips each of the l bits into its complementary bit. By flipping bits, the unitation value of a bitstring may be changed. The following expression describes the probability that a bitstring of unitation value u_1 is converted into a string of value u_2, when u_2 is higher than u_1 (cf. [Bäc96]):

$$p_m^{(u_2>u_1)}(u_2|u_1) = \sum_{k=0}^{\min(u_1,l-u_2)} \binom{u_1}{k} \binom{l-u_1}{k+u_2-u_1} p_m^{u_2-u_1+2k}(1-p_m)^{l-(u_2-u_1+2k)} \quad . \tag{5}$$

From these expressions, $p(u_2|u_1)$ can be calculated for general values:

$$p_m(u_2|u_1) = \begin{cases} p_m^{(u_2>u_1)}(u_2|u_1) & \text{if } u_2 > u_1 \quad ; \\ p_m^{(u_2>u_1)}(l-u_2|l-u_1) & \text{otherwise} \quad . \end{cases} \tag{6}$$

In the second equation the observation is used that the probability of decreasing the number of one-bits from u_1 to u_2 corresponds to the probability of increasing the number of zero-bits from $l-u_1$ to $l-u_2$. This is due to the equal probability for one-to-zero and zero-to-one bit flips.

These formulas allow us in principle to use any unitation function as fitness function, instead of only a basic counting-ones function. In the following, we illustrate this by applying the analysis to so-called trap functions.

B. A Trap Function

The definition of a basic trap function is [DG93]:

$$\hat{f}(u) = \begin{cases} \frac{a}{z}(z-u), & \text{if } u \le z \quad ; \\ \frac{b}{l-z}(u-z), & \text{otherwise} \quad . \end{cases} \tag{7}$$

Figure 1 clarifies the meaning of the parameters.

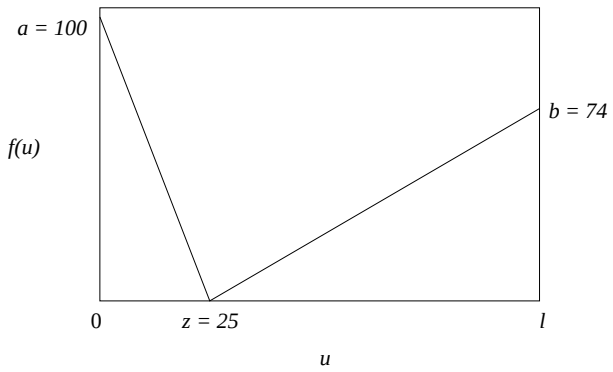


Fig. 1. Trap function $\hat{f}(u)$.

To compute the $I(u)$ sets of a unitation value u we will determine the *corresponding point* of u. A corresponding point is a different unitation value which has the same $\hat{f}(u)$ value. For a trap function there is at most one such point at the other side of the trap. The corresponding point may be computed using

$$\frac{a}{z}(z - u_1) = \frac{b}{l - z}(u_2 - z) \quad , \tag{8}$$

which yields these expressions:

$$u_1 = t_1(u_2) = z + \frac{b}{a}\frac{z}{l - z}(z - u_2) \quad , \tag{9}$$

$$u_2 = t_2(u_1) = z + \frac{a}{b}\frac{l - z}{z}(z - u_1) \quad . \tag{10}$$

These computations could result in non-integer values. In the following we will assume that by computing a corresponding unitation value only an improvement in fitness value is allowed, which results in these rounded numbers:

$$u_1(u) = \begin{cases} u & \text{if } u < z \quad ; \\ z - 1 & \text{if } u = z \quad ; \\ \lfloor t_1(u) \rfloor & \text{otherwise} \quad . \end{cases} \tag{11}$$

$$u_2(u) = \begin{cases} u & \text{if } u \geq z \quad ; \\ \lceil t_2(u_1) \rceil & \text{otherwise} \quad . \end{cases} \tag{12}$$

For a given value u, and the resulting corresponding points $u_1(u)$ and $u_2(u)$, the subsets become:

$$I^+(u) = \{u' \in D | u' \leq u_1(u) \vee u' \geq u_2(u)\} \backslash \{u\} \quad ; \tag{13}$$

$$I^=(u) = \{u\} \quad ; \tag{14}$$

$$I^-(u) = D \backslash (I^+(u) \cup I^=(u)) \quad . \tag{15}$$

We assume here that $I^=(u)$ contains one elements. This is obtained by a clever choice of parameters. There are many possible choices for the parameters a, b and z. Given the number of bits l we stick to the following guidelines:

- $z \approx \frac{1}{4}l$;
- $b = l - z - 1$;
- $1\frac{1}{2}b \leq a \leq 2b$; a a multiple of z.

For l and a we prefer values which are a multiple of 10. These values simplify some of the computations without affecting their generality.

C. A (1:λ)-GA

We use the mutation operator to obtain the following search algorithm:

u `:= uniformly chosen bitstring`

repeat `until maximum generation reached`

S `:=` $\{u_1, \ldots, u_\lambda\}$`;` u_i `is a mutated child of` u`,`

`generated with mutation rate` p_i`.`

u `:= individual with the highest fitness in` S`.`

This algorithm is a very simplified version of a genetic algorithm. We use this algorithm to introduce an extension of an ordinary genetic algorithm: different fixed mutation rates p_i are used to generate children. It is easily shown that this algorithm can be applied to simulate both ordinary (1,λ)- and (1+λ)-GAs:

- (1,λ): $p_1 = p_2 = \ldots = p_\lambda$;

• (1+λ): use $\lambda' = \lambda + 1$, with $p_{\lambda+1} = 0$ and $p_1 = p_2 = \ldots = p_\lambda$. One of the children thus has mutation rate zero, which means that the parent is copied.

With $p(u_1 \rightarrow u_2)$ we will denote the probability that in one iteration of the algorithm the current unitation value of the selected parent changes from u_1 into u_2. To obtain an expression for this probability, we use the following intermediate probabilities:

$$p_m(u_2^-|u_1) = \sum_{k \in I^-(u_1)} p_m(k|u_1) \quad ; \tag{16}$$

$$p_m(u_2^=|u_1) = \sum_{k \in I^=(u_1)} p_m(k|u_1) \quad . \tag{17}$$

These are combined as follows:

$$p(u_1 \rightarrow u_2) = \prod_{i=0}^{\lambda} \left(p_i(u_2^-|u_1) + p_i(u_2^=|u_1) \right) - \prod_{i=0}^{\lambda} p_i(u_2^-|u_1) \quad . \tag{18}$$

The first term is the probability that all offspring are worse than or equal to u_2. From this the probability is subtracted that all offspring are worse. The resulting probability is the probability that at least one of the offspring has unitation value u_2. The i index goes through all offspring and their corresponding mutation rate. It is here that the "multiple mutation rate" principle is applied.

We will refer to the unitation value of the individual u as the current state of the (1:λ)-GA. The states can be ordered according to their $\hat{f}(u)$ values. A higher state is a state with a higher $\hat{f}(u)$ value. With $I^+(u)$ we denote the set of all higher states.

D. Measures

The state transition probabilities according to equation (18) can be used to compute several quality measures of the evolutionary process. The following measures are short-term performance measures:

Improvement probability for state s :

$p^+(s) = \sum_{e \in I^+(s)} p(s \rightarrow e)$, this is the probability of mutating to a better individual and gives a clue how likely it is to make an immediate enhancement when one has an individual with a certain unitation value.

Convergence velocity for state s :

$\varphi(s) = \sum_{e \in I^+(s)} (f(e_1) - f(s_1)) p(s \rightarrow e)$, this is the enhancement one expects to obtain in one generation, given a certain individual. It is the weighted sum of all possible enhancements by the probability that such an enhancement occurs.

Trap jump probability for state s : $p'(s) = \sum_{e \in I^+(s) \cap C(s)} p(s \rightarrow e)$, where C depends on s as follows: $C(s) = \{u_2(s), \cdots, l\}$ if $s \leq z$; $C(s) = \{0, \cdots, u_1(s)\}$ otherwise. Intuitively, this is the probability of going to a better individual at the other side of the trap. Together with the improvement probability, this probability provides an insight in the source of a likely enhancement. A high jump probability indicates that we can easily leave a local path.

Oscillating probability for state s :

$p^o(s) = \sum_{e \in I^+(s) \cap C(s)} p(s \rightarrow e)\ p'(e)/p'(s)$; intuitively, this is the probability of jumping back to the same side of the trap after two generations, given that we jumped to the other side of the trap the first generation. This probability gives a better insight into the usefulness of a high trap jump probability. A mutation rate which makes trap jumping easy, may or may not make it easy to jump back. In the first case, the algorithm may be walking the two sides of the trap in turns, jumping from one side to the other each generation, while in the second case one mutation rate is expected to allow one jump only.

To determine long-term performance measures several generations have to follow each other. For this purpose the transition probabilities are stored in a transition matrix P. The states are ordered such that for a plus-strategy the matrix is uppertriangular; furthermore, the states which contain the optimum (the so-called set of absorbing states S_a) have the highest indexes. With Q we denote the submatrix which does not contain absorbing states [Dör78]. This allows to define several measures:

Absorption time for state s :

$T(s) = \sum_{s'} N_{ss'}$ where $N = (I - Q)^{-1}$; using Markov chain analysis, it can be shown that this formula computes the expected number of generations to reach the global optimum from a certain

individual. If this number is very high, it is almost impossible for the algorithm to find the optimum.

Number of evaluations until absorption for state s :

$N(s) = T(s) \times \lambda$; the previous measure is only reasonable when one compares algorithms which perform the same amount of work every generation. Of course, this is not always the case. Especially on computer architectures which evaluate offspring sequentially, it is much fairer to take into account the number of offspring in order to compare the expected computation time.

Expected absorption time :

$E(T) = \sum_s p(s)T(s)$ where $p(s) = \binom{l}{s} /2^l$; the previous measures give the expected computation time when one knows the starting individual. The genetic algorithm however determines its starting individual randomly using a uniform distribution. To determine the performance of the complete algorithm, the expected absorption time has to determined, thus averaging over all possible starting individuals. With $p(s)$ the probability of a uniformly chosen bitstring with unitation value s is computed.

Expected number of evaluations until absorption :

$E(N) = \sum_s p(s)N(s) = E(T) \times \lambda$; using a similar argument as for $N(s)$, it is fairer to take into account the number of offspring.

In the sequel, we will mainly use the metrics that depend upon the number of evaluations. We believe this reflects the computation efforts best. To compare algorithm setups we use the expected number $E(N)$, which averages over the possible starting individuals. To show the influence of the starting individual, we will also report separate experiments on this.

In previous publications ([Bäc92], [Bäc93], [Bäc95], [Müh92], [Bäc96]) the $p^+(s)$ and $\varphi(s)$ measures were used. We will also give results for these measures here. However, not all of the results can be explained intuitively. We will use the jump probability and oscillating probability to provide argumented explanations.

l	z	a	b
10	3	12	6
20	5	20	14
50	10	80	39
75	20	80	54
100	25	100	74

TABLE I

PARAMETERS OF THE EMPLOYED TRAP FUNCTIONS.

III. Numerical Results and Experiments

We arrange our experiments as follows. First we analyze the short-term measures of a basic (1+1)-GA. This allows for an easy comparison of our results with the results of earlier publications. Next, we will exploit our observations on these basic cases to analyse the more complex algorithms introduced in this article.

In our experiments we use several trap functions. A summary of all functions can be found in Table I. In the sequel we refer to one of these functions by giving the parameter l.

Figure 2 displays the improvement probabilities $p^+(s)$ and convergence velocities $\varphi(s)$ of a (1+1)-GA in several situations. One coordinate, showing the mutation rate p_m, is drawn in a logarithmic scale to show more details for lower mutation rates. Furthermore the trap jump probability is shown.

The graphs can be explained by looking at some of the characteristics of the trap function that was used. First we take a look at the $p_m = 1$ situation, which corresponds to turning a unitation value u into $l - u$. An analysis of the function yields (see upper Figure 2(c)) that for $u(\vec{x}) \leq 8$ and $50 \leq u(\vec{x}) \leq 91$ such a flip does not result in an improved fitness value. This is reflected in Figure 2(a): at $p_m = 1$ only the lines for $u(\vec{x}) = 20$ and $u(\vec{x}) = 35$ are 1. In Figure 2(c) (below), the lines with high mutation rates are 1 in exactly the regions we indicated. The figure also shows that a mutation

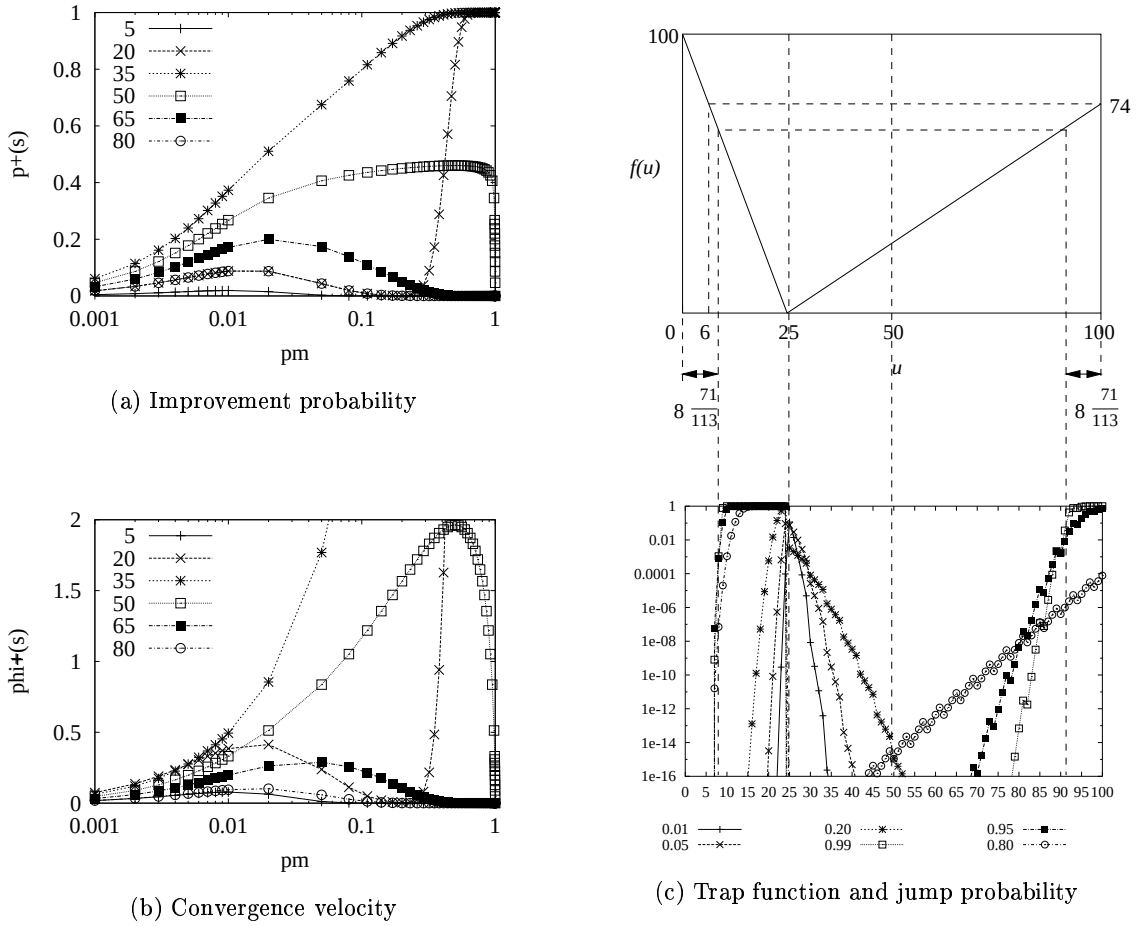


Fig. 2. Probabilistic behaviour of the (1+1)-GA for the $l = 100$ trap function. In (a) and (b) the value of p_m is varied horizontally; the lines correspond to $u(\vec{x}) \in \{5, 20, 35, 50, 65, 80\}$. In (c) the unitation value is displayed horizontally; for several mutation rates the probability is shown that the next individual is better and on the other side of the trap (which is at $z = 25$).

rate significantly lower than 1 is better in the range $50 \leq u(\vec{x}) \leq 91$ in order to jump to the other side of the trap.

If Figure 2(a) would have been shown on a linear scale, it would appear that the $u(\vec{x}) = 50$ curve is symmetric. This can be explained by considering this construction: a mutated string with mutation rate p_m can also be obtained by first flipping all the bits and by then mutating all bits with mutation rate $1 - p_m$. As a bit-flipped string with $u(\vec{x}) = 50$ results in a string with $u(\vec{x}) = 50$, the result of mutating such a string with p_m must be the same as by mutating with $1 - p_m$. The same argument

also explains that the curves for $p_m = 0.2$ and $p_m = 0.8$ in Figure 2(c) cross each other in $u(\vec{x}) = 50$.

In Figure 2(c) the lines for high mutation rates show non-monotonic behaviour when $u(\vec{x}) > 50$: every five unitation values, the trap jump probability decreases slightly before increasing again. This can be explained by looking at corresponding values. In this particular trap function it appears that four unitation values at the right hand side of the trap share the same corresponding value at the left hand side. The highest of these four values has the highest jump probability: in that case more bits are flipped, which is more likely to occur (remember that we are looking at mutation rates which are more likely to flip bits than to maintain). After four unitation values, the corresponding value gets lower, which reduces the number of unitation values on the other side of the trap which cause an enhancement. This consequently reduces the probability of jumping.

For $u(\vec{x}) = 80$ the curve in Figure 2(a) has two local optima (at approximately 0.01 and approximately 1.0). We saw that the second optimum corresponds to flipping (almost) all bits, resulting in a jump from one side of the trap to the other (better) side. The first local optimum therefore corresponds to the mutation rate which maximizes the local improvement probability (which leads the genetic algorithm towards the local optimum $u(\vec{x}) = 100$). Most strikingly, the curves for $u(\vec{x}) = 80$ and $u(\vec{x}) = 20$ (with equal distances to a nearby local optimum) overlap each other for low mutation rates and share their local maximum at $p_m \approx 0.02$.

From the graph it can be deduced that the closer the unitation value approaches a local optimum, the lower the optimal mutation rate for local search becomes, until it reaches $p_m = 0.01$ for $u(\vec{x})$. This is in harmony with the mutation rates derived in [Bäc93] $(1/(2(u(\vec{x})+1)-l))$ and [Müh92] $(1/l)$. The intuition of these schedules is as follows: when the local optimum is almost reached, it is most safe to flip one bit in each mutation. If the mutation rate must be constant throughout the whole process, $1/l$ is the best choice, as most of the time is usually spent in fine-tuning the solution.

Graphs similar to those in Figure 2 for the other values of l also display the mentioned phenomena. For example, also for the other trap functions $1/l$ seems a reasonable mutation rate in order to optimize

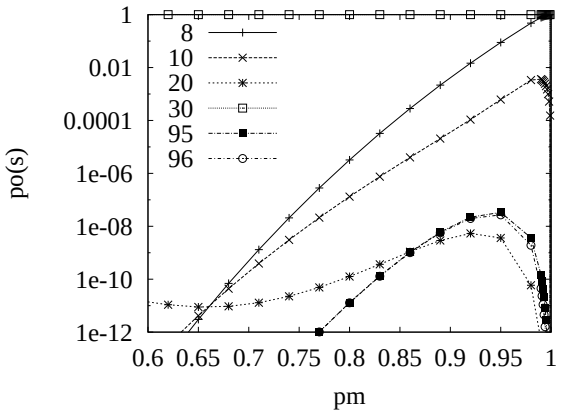


Fig. 3. Oscillating probability for values of $u(\vec{x})$. Mutation rate p_m is shown on the ordinate axis. A (1+1)-GA is applied to the $l = 100$ trap function.

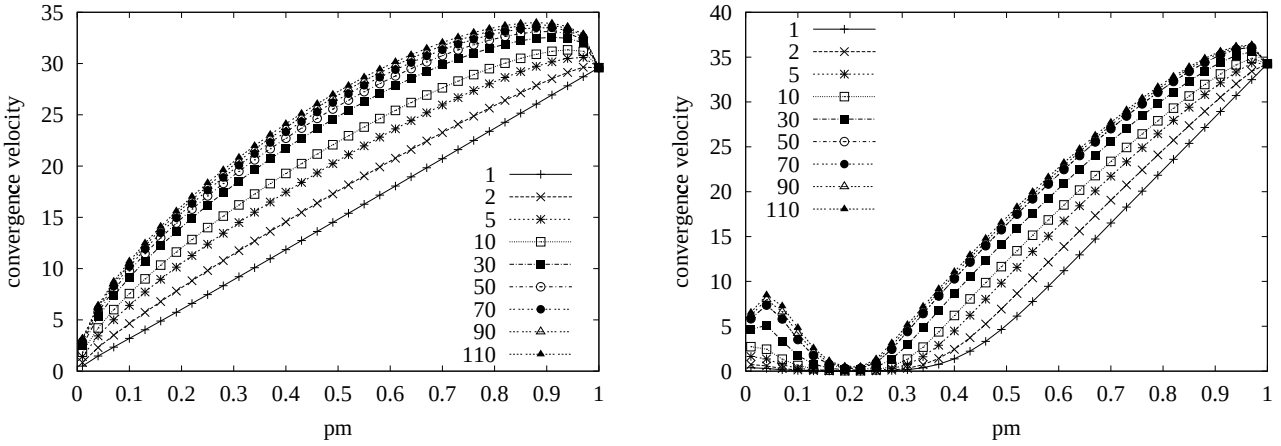
local search.

Figure 2(b) is very similar to Figure 2(a). As expected, the curves for $u(\vec{x}) = 20$ and $u(\vec{x}) = 80$ do not overlap: as the slope of the curve is much higher at the left hand side of the trap, the convergence velocity is also higher there.

An interesting phenomenon is shown in Figure 3, which shows the oscillating probability for unitation values which may display the oscillating behaviour: those values for which a better individual can be found on both sides of the trap. It clearly demonstrates that high mutation rates increase oscillating behaviour in most situations, unless p_m approximates 1. The reason for the latter becomes clear when one considers mutation rate 1 as was done above. Of a unitation value and its bit-flipped value only one can be the best. The bit-flip operation can be exploited once to go to the best of these two values, but that side of the trap will never be abandoned. In further experiments we found that for lower values of l the maximum oscillating probability is found for lower values of p_m.

In Figure 4 the effect of using multiple offspring is visualized. The convergence velocity increases in all cases. The impact of using additional offspring however decreases – approximately – exponentially for every new offspring. This is in agreement with findings for (1,λ)-evolution trategies that convergence velocity is $\varphi \sim \sqrt{2 \ln \lambda}$ and thus grows only logarithmically with λ [Bey01].

Figure 4(b) shows two local maxima for all λ values. This is explained by the two means of im-



(a) After the trap: $u(\vec{x}) = 35$.

(b) Before the trap: $u(\vec{x}) = 20$.

Fig. 4. Convergence velocity of a (1+λ)-GA on the trap function for values of p_m, $\lambda \in \{1, 2, 5, 10, 30, 50, 70, 90, 110\}$.

provement: staying on the same side of the trap (for low probabilities) or going to the other side of $u(\vec{x}) = 50$ (for high probabilities). Indeed, Figure 4(a) has only one local maximum, as both large and small mutation rates will result in offspring at the right hand side of the trap with most probability.

Keeping in mind our previous discussion, it is instructive in both graphs to see that the convergence velocity decreases for very high mutation rates when $\lambda > 1$. This may be caused by the lack of variety in the pool of offspring for very high mutation rates. With $p_m = 0.5$ the information in an individual is not exploited and the variety is maximal; in $p_m = 1.0$ there is only one possible offspring; the variety is clearly very small in that case. The figure provides a good indication that the optimum is between 0.5 and 1.0, but more close to $p_m = 1.0$. This is an argument for using mutation rates little below 1.0.

We do not present any long-term quality measures here for the $l = 100$ trap function. It can be shown that a mutation rate of 0.01 yields an approximate absorption time of 8.39×10^{178} generations! Our computations appeared not to be sufficiently accurate with such extraordinary numbers. For the (1+1)-GA and problem dimension $l = 10$ Table II gives an impression of the absorption times when starting at an individual with $u(\vec{x}) = 5$. It appears that $p_m = 0.5$ is the best mutation rate. This gives every bit an equal probability of being zero or one in each generation, which gives every bit pattern probability 2^{-10} in each generation. On average it will therefore take 2^{10} generations to find

p_m	0.1	0.2	0.5	0.8	0.9
$T(5)$	107×10^6	0.22×10^6	1024	2.2×10^6	1047×10^6

TABLE II

ABSORPTION TIMES FOR A (1+1)-GA ON A $l = 10$ TRAP FUNCTION

Algorithm	parent as offspring	equal mutation rates
(1,λ)	no	yes
(1+λ)	yes	yes
(1:λ)	no	no
(1$\div\lambda$)	yes	no

TABLE III

NOTATION OF GENETIC ALGORITHMS

the optimal bit pattern.

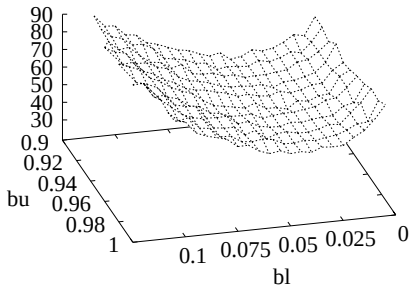
From these first experiments we may conclude that a (1+1)-GA cannot possibly solve the trap function problem efficiently. It is also clear that a (1+λ)-GA will not be efficient either, as all offspring are created by the same distribution. An efficient solution seems only possible if several distributions are available during the evolutionary process. We will therefore investigate the (1:λ)-GA next.

For the λ offspring, we will test the following mutation rates:

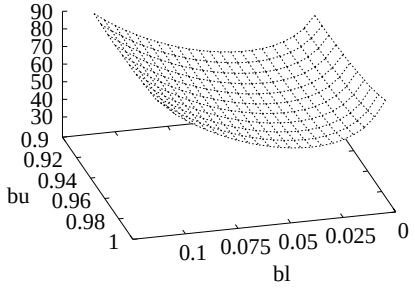
- linear: $p_i = (b_u - b_l)\frac{i-1}{\lambda-1} + b_l$. We will abbreviate this with "l b_l b_u";
- exponential: $p_i = b_u \rho^{i-1}$, where $\rho = (b_l/b_u)^{\frac{1}{\lambda-1}}$. This will be abbreviated with "e b_l b_u".

In all cases we assume that there is one offspring with mutation rate $p_m = 0$, such that the (1:λ)-GA becomes an adapted (1+λ')-GA, with $\lambda' = \lambda - 1$. In the sequel, we will denote this (1:λ) genetic algorithm with one mutation rate $p_m = 0$ as a (1$\div\lambda'$)-GA (Table III).

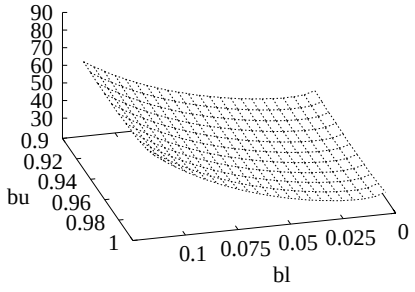
We first check the correctness of the formulas we developed. In Figures 5(a) and 5(b) experimental



(a) Experimental expected absorption time (linear scheme).



(b) Theoretical expected absorption time (linear scheme). The minimum is at $b_u = 0.04$ and $b_l = 0.93$ with a value of 56.0.



(c) Theoretical expected absorption time (exponential scheme). The minimum is at $b_u = 0.02$ and $b_l = 0.96$ with a value of 26.94.

$u(\vec{a})$	theory	experiment
1	48.891	50.126
2	58.526	59.810
3	63.694	62.201
4	68.164	67.678
5	70.970	69.908
6	72.597	71.355
7	73.417	71.101
8	73.893	72.187
9	74.121	74.630
10	74.188	73.230
11	74.120	74.182
12	73.887	73.656
13	73.403	74.239
14	72.560	70.611
15	70.905	70.664
16	68.071	66.359
17	63.694	63.318
18	58.529	59.277
19	49.200	50.478
20	1.000	1.000

(d) $T(s)$ for several values of $u(\vec{x})$ with $b_l = 0.01$ and $b_u = 1$ (linear scheme).

Fig. 5. Absorption times of a $(1 \div 10)$-GA for given upper and lower bounds. The experimental values are obtained by averaging over 1000 experiments for each coordinate. The $l = 20$ function is used.

	$l = 10$				$l = 20$				$l = 50$ $(\times 10^{11})$			
λ	2	3	5	10	2	3	5	10	2	3	5	10
l 0.01 1	435	367	207	155	648	848	933	737	49	71	112	186
l 0.05 1	121	137	138	140	253	359	495	610	54	80	132	244
l 0.1 1	91	111	129	148	294	428	612	830	266	399	661	1270
l 0.02 0.9	150	184	151	136	299	398	533	596	**37**	**55**	**87**	**154**
l 0.05 0.9	88	114	123	**130**	**211**	**295**	**428**	**576**	53	80	130	243
l 0.1 0.9	**72**	**96**	**118**	137	244	353	536	789	261	392	650	1260
e 0.01 1	435	115	106	105	648	303	289	291	49	55	**47**	**48**
e 0.05 1	121	91	88	92	253	297	306	331	54	78	110	139
e 0.1 1	91	94	95	101	294	401	483	545	266	398	633	956
e 0.02 0.9	150	84	86	94	299	**249**	**251**	**284**	**37**	**49**	53	60
e 0.05 0.9	88	**77**	**80**	**90**	**211**	251	276	324	53	78	109	139
e 0.1 0.9	**72**	81	87	98	244	333	416	511	261	390	619	928

TABLE IV

EXPECTED NUMBER OF EVALUATIONS FOR SEVERAL ALGORITHMS

expected absorption times and theoretical times are plotted for a $(1 \div 10)$-GA with $l = 20$. Only those ranges of b_u and b_l are shown which result in low times. The theory predicts the experimental results accurately and shows the same dependence on the parameters. Figure 5(d) displays a comparison of numbers. Here the starting individual is fixed, such that uniform initialization is not taken into account. The experimentally determined numbers (which were averaged over 1000 experiments) clearly converge to their theoretical values. The theory therefore seems reliable. We use it in table IV to investigate the influence of the λ parameter. The table displays the theoretically computed expected number of evaluations $E(N)$ for several algorithms.

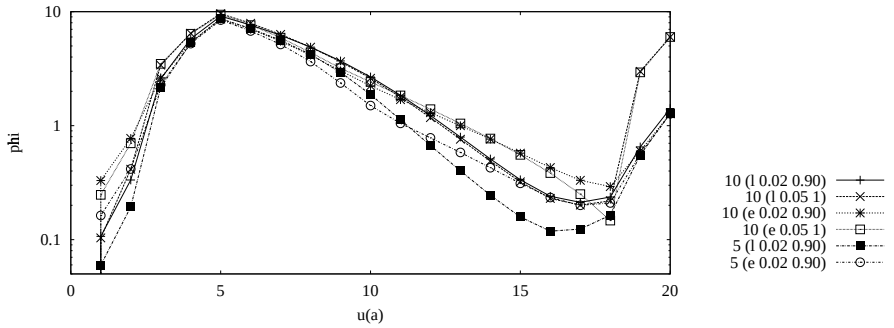


Fig. 6. The convergence velocity for several mutation schemes. Note that one scale is logarithmic. The $l = 20$ problem was used.

When $\lambda = 2$ the algorithms reduce to genetic algorithms with two mutation rates b_l and b_u. At first sight, the $1/l$ mutation rate guideline is also applicable here: both in linear and exponential schemes, a mutation rate of approximately $1/l$ should be present to obtain the best absorption time. When the number of offspring increases, the optimal underbound decreases. This is reasonable: it means that it is more advantageous to add some low mutation rates than to add more high mutation rates.

From the table we may conclude that a scheme with a low number of offspring is preferable in most cases. However, striking exceptions to this rule are several exponential schemes (eg. scheme (e 0.01 1) and scheme (e 0.02 0.9)). Figure 6 may provide an explanation for this. It displays the convergence velocity for some of the algorithms in the table. Although the linear algorithm provides a higher convergence velocity for various unitation values, the exponential scheme is better on difficult values, which are those where the velocity is low. To stress this, the graph is drawn on a logarithmic scale. Especially a $u(\vec{x}) = 1$ individual is very likely to be encountered. The exponential scheme performs best here as it provides a wider variety of low mutation rates and thus enlarges the probability that the last bit is correctly flipped.

Using the table and the graphs of Figures 5(c) and 5(b) we can compare the linear and the exponential schemes. The exponential mutation scheme performs surprisingly well. For example a (e 0.02 0.9)-(1$\div$10)-GA performs significantly better than many (1$\div$3)-GAs. The surface of the exponential scheme displayed in Figure 5(c) is also clearly below that of a linear scheme and is less sensitive to the values

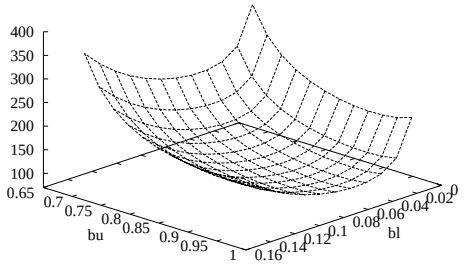


Fig. 7. $E(T)$ for several values of b_l and b_u. A linear $(1 \div 2)$-GA is applied to the $l = 20$ trap function.

of the b_u and b_l bounds.

We investigate the $(1 \div 2)$-GA further and try to find the 2 optimal mutation rates. Figure 7 shows $E(T)$ for the region of interest of b_u and b_l. As can be seen in this graph the surface has a parabolic shape, which means that b_l and b_u can be optimized independently of each other. The global minimum is approximately located at $b_l = 0.06$ and $b_u = 0.93$; the absorption time amounts to 103 generations. These mutation rates show a remarkable similarity to the mutation rates which maximize local search and encourage oscillating behaviour, as were given before[1]. As many oscillations increase the diversity during the search (we are walking the two hills in turns), it appears that an optimal solution is found by maximizing this diversity.

IV. Conclusions

Using a trap function example, the results presented here give an idea how previously published genetic algorithms scale up to more complex problems. More specifically, we have shown using mainly theoretical arguments that the $1/l$ guideline for optimal mutation rates [Müh92] optimizes local search (exploitation), but is incapable of providing sufficient diversity in the search (exploration). After having shown that a genetic algorithm with one mutation rate and no crossover has many difficulties finding the optimum, we investigated the possibility of using mutation to perform exploration. For the trap

[1]Note that the argument of maximizing convergence velocity by choosing values lower than 1 is not valid here, as it was only valid for large numbers of offspring.

function we found that a mutation rate approximating 1, but not exactly equal to it, maximizes the probability that a new hill is climbed every generation, and thus enhances exploration. We used these observations to construct a new algorithm which provides several mutation rates every generation. Using theoretical arguments we found that indeed two mutation rates $1/l$ and ≈ 1 solve the trap problem efficiently. The experiments confirmed that both exploration and exploitation can be obtained by the mutation operator as long as there is variation in the available mutation rates.

To get a better insight in the importance of the multiple mutation rates, we performed some initial experiments using many mutation rates each generation. If the target is to find the real optimum, we found that it is advantageous if there are more small mutation rates than there are large rates. This is likely to be caused by a similar argument as was used in the derivation of $1/l$ by [Müh92]: the genetic algorithm will spend most of its time fine-tuning, which is enhanced by using many small mutation rates.

In this article we focused on one particular function with two local optima. We found that one high mutation rate is sufficient to obtain population diversity. Further investigations with more complex functions could give a better insight into how many high mutation rates are necessary. For such functions it would also be interesting to compare the impact of high mutation rates with the effect of a crossover operator.

The usage of multiple mutation rates to obtain diversity may not only be advantageous in the static problem we currently investigated, it could also be useful in dynamic problems. Using some of the measures we gave in this document we plan to investigate the applicability of such a genetic algorithm on dynamic problems.

References

[BA01] H.-G. Beyer and D. V. Arnold. Theory of evolution strategies – a tutorial. In Kallel et al. [KNR01], pages 109–133.

[Bäc92] Th. Bäck. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature, 2*, pages 85–94. Elsevier, Amsterdam, 1992.

[Bäc93] Th. Bäck. Optimal mutation rates in genetic search. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 2–8. Morgan Kaufmann, San Mateo, CA, 1993.

[Bäc95] Th. Bäck. Order statistics for convergence velocity analysis in simplified evolutionary algorithms. In D. Whitley and M. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 91–102. Morgan Kaufmann, San Mateo, CA, 1995.

[Bäc96] Th. Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, New York, 1996.

[Bey01] H.-G. Beyer. *The theory of evolution strategies.* Natural Computing Series. Springer, Berlin, 2001.

[DG93] K. Deb and D.E. Goldberg. Analyzing deception in trap functions. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 93–108. Morgan Kaufmann, San Mateo, CA, 1993.

[Dör78] W. Dörfler. *Mathematik für Informatiker*, volume 1. Carl Hanser, München, 1978.

[EAH91] A. E. Eiben, E. H. L. Aarts, and K. M. Van Hee. Global convergence of genetic algorithms: An infinite Markov chain analysis. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature — Proceedings 1st Workshop PPSN I*, volume 496 of *Lecture Notes in Computer Science*, pages 4–12. Springer, Berlin, 1991.

[Gal85] R. Galar. Handicapped individua in evolutionary processes. *Biological Cybernetics*, 53:1–9, 1985.

[Gal89] R. Galar. Evolutionary search with soft selection. *Biological Cybernetics*, 60:357–364, 1989.

[JW99] T. Jansen and I. Wegener. On the analysis of evolutionary algorithms – a proof that crossover really can help. In J. Nesetril, editor, *Proceedings of the 7th Annual European Symposium on Algorithms (ESA '99)*, volume 1643 of *Lecture Notes in Computer Science*, pages 184–193. Springer, Berlin, 1999.

[KNR01] L. Kallel, B. Naudts, and A. Rogers, editors. *Theoretical Aspects of Evolutionary Computing*, Natural Computing Series. Springer Verlag, Berlin, 2001.

[Müh92] H. Mühlenbein. How genetic algorithms really work: I. mutation and hillclimbing. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature 2*, pages 15–25. Elsevier, Amsterdam, 1992.

[Rec73] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann–Holzboog, Stuttgart, 1973.

[RPB01] A. Roger and A. Prügel-Bennett. A solvable model of a hard optimisation problem. In Kallel et al. [KNR01], pages 207–221.

[Rud97] G. Rudolph. *Convergence Properties of Evolutionary Algorithms.* Verlag Dr. Kovač, Hamburg, 1997.

[Sch77] H.-P. Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*, volume 26 of *Interdisciplinary Systems Research.* Birkhäuser, Basel, 1977.