

## Reevaluating Genetic Algorithm Performance under Coordinate Rotation of Benchmark Functions

A survey of some theoretical and practical aspects of genetic algorithms

RALF SALOMON

*AI Lab, Computer Science Department, University of Zurich*

*Winterthurerstr. 190, 8057 Zurich, Switzerland*

salomon@ifi.unizh.ch

### Abstract

In recent years, genetic algorithms (GAs) have become increasingly robust and easy to use. Current knowledge and many successful experiments suggest that the application of GAs is not limited to easy-to-optimize unimodal functions. Several results and GA theory give the impression that GAs easily escape from millions of local optima and reliably converge to a single global optimum. The theoretical analysis presented in this paper shows that most of the widely-used test functions have  $n$  independent parameters and that, when optimizing such functions, many GAs scale with an  $O(n \ln n)$  complexity. Furthermore, it is shown that the current design of GAs and its parameter settings are optimal with respect to independent parameters. Both analysis and results show that a rotation of the coordinate system causes a severe performance loss to GAs that use a small mutation rate. In case of a rotation, the GA's complexity can increase up to  $O(n^n) = O(\exp(n \ln n))$ . Future work should find new GA designs that solve this performance loss. As long as these problems have not been solved, the application of GAs will be limited to the optimization of easy-to-optimize functions.

## 1 Introduction

A genetic algorithm (GA) is a heuristic population-based search procedure based on natural selection and genetics. GA research has recently shown tremendous progress, especially within the last decade. They became robust, easy to use, and applicable in diverse areas such as machine learning, combinatorial problems, VLSI design, and numerical optimization. Many of these applications can be considered as continuous

parameter optimization, where an algorithm has to find a set of  $n$  real-valued parameters such that the objective function (also called fitness function) is close to an optimum. In addition to their practical success, extensive theory (see, for example, Goldberg 1989 or Mühlenbein 1993a, 1993b) explains and predicts the behavior of GAs.

In its simplest form, a GA works on a population of fixed bit strings. First, it selects some individuals as parents for the next generation. Then, it mates two different parents to produce two new offspring by means of recombination (i.e., exchange of partial bit strings) and mutation (i.e., bit inversion). In the final evaluation phase of each generation, it assigns a resulting fitness value to each individual. A discussion of other GA variants can be found in Section 3.

Current knowledge and many successful experiments suggest that the application of GAs is not limited to easy-to-optimize unimodal functions. A GA works as a multi-point search strategy opposed to a point-to-point search in classical methods. Furthermore, a GA produces offspring by means of random mutations and recombination/crossover. Featuring these mechanisms, a GA is thought to be able to escape from local optima and, thus, is able to find the sought global optimum. Many results have been reported (Goldberg 1989; Mühlenbein and Schlierkamp-Voosen 1993a, 1993b; Srinivas and Patnaik 1994), where a GA overcomes millions of local optima and reliably converges to the global optimum. An extensive introduction to GAs can be found in Goldberg (1989) or Srinivas and Patnaik (1994).

For evaluation and test purposes, the research community has developed a variety of test functions, which are briefly discussed in Section 2. Also, Section 2 defines the notion of convergence. It is important to note that most test functions are decomposable, that is, optimization can be done in  $n$  independent 1-dimensional optimization tasks. Section 3 describes some important GAs and other evolutionary algorithms, such as evolution strategies and evolutionary programming. In addition to these algorithms, hybrid methods, such as collective learning (CL) (Salomon 1994) or Lamarckian evolution (Whitley 1994) have been proposed. Hybrid methods normally work on different levels each aiming at different goals, such as global convergence, convergence speed, or final accuracy.

Section 4 analyzes the benefits of small mutation rates that are usually used. It turns out that small mutation rates are a perfect adaptation to the widely-used test functions, which can be decomposed into a sum of  $n$  independent 1-dimensional functions. Furthermore, Section 4 shows that this implicit adaptation rises the problem that the complexity of a GA featuring a small mutation rate increases up to  $O(\exp(n \ln n))$ , if the fitness function is not decomposable.

The breeder genetic algorithm (BGA) has been proposed in Mühlenbein and Schlierkamp-Voosen (1993a, 1993b) and is tailored for continuous parameter optimization. Section 5 presents some results for the BGA when optimizing widely-used test functions under a rotation of the coordinate system. The results confirm the analysis presented in Section 4; a rotation causes a drastic performance loss. Section 6 concludes with a discussion. Appendix A presents a formal derivation of the GA’s convergence speed. The main result is that a whole class of practical GAs scale with  $O(n \ln n)$  when applied to the widely-used (decomposable) functions. Finally, the program for the coordinate rotation is given in Appendix B.

## 2 Function Test Bed

In the field of continuous parameter optimization, an algorithm has to find a set of  $n$  variables  $\{x_1^o, \dots, x_n^o\}$  such that the objective function  $f(x_1^o, \dots, x_n^o) = \text{opt}$ . The variables are denoted by  $x_i, 1 \leq i \leq n$ , with  $n$  as the problem’s dimensionality, and the superscript in  $x_i^o$  denotes that  $x_i$  is an optimal value with respect to the objective function. Due to physical constraints and technical reasons, the search interval of each parameter is limited to a prespecified range  $x_i \in [-\text{range}_i, \text{range}_i]$ . Note that any maximization task  $f(\vec{x}) \rightarrow \max$  is equivalent to the minimization task  $-f(\vec{x}) \rightarrow \min$ . Therefore, maximization tasks can be discarded without loss of generality.

In the present context, an algorithm converges, if it finds at least one solution  $\vec{x}_s, \|\vec{x}_s - \vec{x}_o\| \leq \epsilon$  with a prespecified probability  $p$  and an arbitrary  $\epsilon$  neighborhood of the optimum  $\vec{x}_o$ .

Table 1 summarizes some of the widely-used test functions. The first five test functions have been proposed by De Jong (1975). All test functions reflect different degrees of complexity. Test functions  $f_1 - f_4, f_8$ , and  $f_9$  are unimodal (i.e., containing only one optimum), whereas the other test functions are multimodal (i.e., containing many local optima, but only one global optimum). Function  $f_1$  is a simple quadratic parabola. Rosenbrock’s function  $f_2$  is considered to be difficult, because of its narrow curved valley containing the minimum at  $\vec{x}_o = (1, 1)^T$ . Function  $f_3$  is a piecewise continuous step function, and function  $f_4$  adds  $(0, 1)$ -normal distributed noise. The fifth function contains 25 local optima each at  $\vec{x} = (a_{1j}, a_{2j})^T$  with a corresponding function value of  $f_5(\vec{x}) = 0.002 + 1/c_j$ . De Jong originally defined  $F_5 = 1/f_5$ . However, many authors use  $f_5$  (probably due to a typographical error). So do we, to allow a better comparison to others.

Rastrigin’s function ( $f_6$ ) is highly multimodal and has its minimum at  $\vec{x}_o = \vec{0}$  with  $f_6(\vec{x}_o) = 0$ . In each direction,  $f_6$  consists of a cosine term and an additional quadratic

**Table 1:** The function test bed. The limits given below have been introduced in previous research and are used throughout this paper.

	Function	Limits	Name
1	$f_1(\vec{x}) = \sum_{i=1}^3 x_i^2 = \ \vec{x}\ ^2$	$-5.12 \leq x_i \leq 5.12$	Sphere
2	$f_2(\vec{x}) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$	$-2.048 \leq x_i \leq 2.048$	Rosenbrock's
3	$f_3(\vec{x}) = \sum_{i=1}^5 \text{integer}(x_i)$	$-5.12 \leq x_i \leq 5.12$	
4	$f_4(\vec{x}) = \sum_{i=1}^{30} i x_i^4 + \text{Gauss}(0, 1)$	$-1.28 \leq x_i \leq 1.28$	
5	$f_5(\vec{x}) = 0.002 + \sum_{j=1}^{25} \frac{1}{c_j + \sum_{i=1}^2 (x_i - a_{ij})^6}$	$-65.536 \leq x_i \leq 65.536$	Shekel's
6	$f_6(\vec{x}) = \sum_{i=1}^{20} [x_i^2 - 10 \cos(2\pi x_i) + 10]$	$-5.12 \leq x_i \leq 5.12$	Rastrigin's
7	$f_7(\vec{x}) = \sum_{i=1}^{10} -x_i \sin(\sqrt{ x_i })$	$-500 \leq x_i \leq 500$	Schwefel's
8	$f_8(\vec{x}) = \sum_{i=1}^{10} 10^{i-1} x_i^2$	$-10.0 \leq x_i \leq 10.0$	Ellipsoid
9	$f_9(\vec{x}) = \sum_{i=1}^{20} \left( \sum_{j=1}^i x_j \right)^2$	$-65.536 \leq x_i \leq 65.536$	
10	$f_{10}(\vec{x}) = -\cos(2\pi \ \vec{x}\ ) + 0.1 \ \vec{x}\  + 1$	$-100.0 \leq x_i \leq 100.0$	
11	$f_{11}(\vec{x}) = \sum_{i=1}^{10} \frac{x_i^2}{4000} - \prod_{i=1}^{10} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$-600 \leq x_i \leq 600$	Griewangk's
12	$f_{12}(\vec{x}) = -20 \exp(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^{30} x_i^2}) - \exp\left(\frac{1}{n} \sum_{i=1}^{30} \cos(2\pi x_i)\right) + 20 + e$	$-30 \leq x_i \leq 30$	Ackley's

term  $x_i^2$ . Rastrigin's function contains millions of local optima in the interval of consideration. Schwefel's function ( $f_7$ ) possesses the difficulty that the global minimum is given by  $x_i = 420.9687, 1 \leq i \leq 10$  and that the second best minimum  $x_j = -302.5253, x_i = 420.9687, 1 \leq i \leq 10, i \neq j$  is far away from the global optimum.

Function  $f_8$  is a simple quadratic parabola with different eigenvalues along each axis; if all eigenvalues were set to the value 1,  $f_8$  would become the sphere  $f_1$ . The graphs of constant function values appear as  $n$ -dimensional hyperellipses. For such functions, it is well known that the ratio of the smallest and largest eigenvalue, also called

eccentricity, significantly influences the convergence speed of most classical methods (see, for example, Press et al., 1987) and many evolutionary algorithms (see also Section 3). Function  $f_9$  is a true quadratic minimization problem, with its minimum at  $\vec{x}_o = \vec{0}$  and  $f_9(\vec{x}_o) = 0$ .

Function  $f_{10}$  is rotationally invariant and has its minimum at  $\vec{x}_o = \vec{0}$ . In the two-dimensional case, the graphs of constant function values resemble concentric waves with an additional linear term. Functions  $f_{11}$  and  $f_{12}$  are known as Ackley's and Griewangk's function respectively. These functions are two more examples of multimodal functions.

It is especially hard to optimize multimodal functions like  $f_6$  or  $f_7$ . Most algorithms have difficulties converging close to the minimum of such functions, because the probability of making progress decreases rapidly as the minimum is approached; most stagnate in its vicinity (see, for example, Hoffmeister and Bäck 1990). Further discussion of these test functions and their properties can be found in Mühlenbein and Schlierkamp-Voosen (1993a, 1993b), Gill et al. (1981), De Jong (1975), Goldberg (1989), or Schwefel (1995). Extensive discussions of classical optimization techniques can be found in Gill et al. (1981), Luenberger (1984), or Schwefel (1995).

### 3 Algorithms

This paper focuses on the performance loss of typical GAs when applying a rotation to the coordinate system. To this end, this section reviews typical GAs, some implementation issues, and standard parameter settings. Furthermore, this section briefly describes the evolution strategy, which is used for comparison purposes in some cases.

There does not exist *one* GA, but rather a variety of various variants, each covering different applications and aspects. According to Schwefel (1995), the canonical GA can be described as follows

- Step 0: Initialization of the population's individuals and evaluation of the individuals' fitness
- Step 1: Selection of the parents according to a preselected selection scheme (e.g., roulette wheel, linear ranking, truncation selection)
- Step 2: Recombination of selected parents by exchanging parts of their genes
- Step 3: Mutation of some genes by a prespecified probability
- Step 4: Go to Step 1

In the domain of continuous parameter optimization, an algorithm has to find a set

of floating-point values associated with the extrema of a given fitness function. But before using a GA or any other computer-aided method in a particular application, an appropriate coding mechanism has to be chosen. Real-valued parameters can be encoded in many different ways. Traditionally, a GA works on bit strings of length  $l$  and, consequently, treats every parameter as a bit string. Depending on the required precision, most coding schemes encode each parameter by a certain number  $q$  of bits, with  $l = qn$ , and attaches a mapping function, which maps the bit representation to real-valued parameters (fixed point). Mutation is then implemented by bit flipping operations. Many examples of this simple genetic algorithm (SGA) can be found in De Jong (1975) or Goldberg (1989). A different coding scheme directly uses the bit representation provided by the programming language (see, for example, Mühlenbein et al. 1991). In such cases, no mapping function is needed, and mutation can be still implemented by bit flipping operations. But it has to be ensured that each bit operation yields a valid floating-point number. The BGA proposes a third coding scheme. It works directly on a set of floating-point values and the mutation operator adds or subtracts small random numbers according to a given probability distribution. This coding scheme can be seen as an abstraction of the bit level representation. A discrete mutation operator can yield the same functionality as the traditional bit string representation, but eases many implementation issues. Furthermore, the mutation operator, i.e., the random number generator, can be implemented such that it yields any desired accuracy.

Many GA variants fit into the framework of the canonical GA. This section describes the BGA (Mühlenbein and Schlierkamp-Voosen 1993a, 1993b) in more detail, because the BGA is one of the most efficient GAs available in the domain of continuous parameter optimization and, in addition, an extended theory has been proposed that verifies various practical results. The BGA works as follows. It selects the  $T\%$  individuals (called truncation selection in Mühlenbein and Schlierkamp-Voosen 1993a, 1993b), which is equivalent to a  $(\mu, \lambda)$  selection scheme in evolution strategy, and, furthermore, the BGA preserves the best individual (elitist selection). To produce offspring, the BGA selects two different parents and applies recombination as well as mutation. The BGA uses uniform recombination, where each parameter is exchanged with the corresponding parameter of the other object with a probability  $p_r = 0.5$ . Uniform recombination is equivalent to  $n$ -point crossover, where crossover can take place only at parameter boundaries. Mutation is implemented by the following random generator (Mühlenbein and Schlierkamp-Voosen 1993a)

$$\sigma = \sum_{i=0}^{15} \alpha_i 2^{-i} \quad (1)$$

where each  $\alpha_i$  is set to the value 1 with a probability of  $1/16$  and set to the value 0 with a probability of  $15/16$  respectively. In addition, it is guaranteed that at least one  $\alpha_i$  is set to 1, and the sign  $s$  is randomly chosen from  $\{-1, 1\}$  with equal probability.

According to the following update rule, the BGA mutates each parameter  $x_i$  with a probability  $p_m = 1/n$ :

$$x_i \leftarrow x_i + \beta \text{range}_i \sigma \quad (2)$$

where  $\text{range}_i$  denotes the search interval  $x_i \in [-\text{range}_i, \text{range}_i]$  of each variable  $x_i$  and  $\beta$  denoting the mutation strength. Usually,  $\beta$  is set to 0.1, but a value of 1.0 allows for more flexibility. The BGA does not use any further crossover operator nor does it use any local steepest descent. Most results presented in Mühlenbein and Schlierkamp-Voosen (1993a, 1993b) suggest an  $O(n \ln n)$  complexity for the BGA when applied to the widely-used test functions. The following table shows the number of function evaluations needed by the BGA when applied to Rastrigin’s function ( $f_6$ ) for several dimensions (20 to 400).

	$n = 20$	$n = 50$	$n = 100$	$n = 200$	$n = 400$
BGA	3608	9592	25040	52948	112634

Even though the performance of GAs is in the focus of this paper, some results are compared with the evolution strategy (ES) developed by Rechenberg (1973) and Schwefel (1977). In each generation, a  $(\mu \dagger \lambda)$ -ES has  $\mu$  parents and generates  $\lambda$  offspring. The difference of both selection schemes is that a  $(\mu + \lambda)$ -ES selects the parents for the new generation from the union of the former parents and the new offspring, whereas a  $(\mu, \lambda)$ -ES selects the parents only from the new offspring. In general, mutations are applied to *all* parameters ( $p_m = 1$ ) by adding normally distributed random numbers with expectation value 0. But in contrast to GAs, ES controls the mutation strength by a self-adaptation mechanism. In its simplest form, ES maintains a global step size  $\sigma$  for each individual and multiplies all randomly drawn numbers with this step size. The step size  $\sigma$  itself is inherited from the parent and modified by either constant factors (Rechenberg 1973) or log-normal random numbers (Schwefel 1995) prior to its first application. The main idea behind this self-adaptation mechanism is that ES statistically self-adapts the step size to nearly optimal values. In its more elaborate form, ES maintains an individual step size  $\sigma_i$  for each parameter to normalize different eigenvalues along each axis (see, for example, Ostermeier et al. 1994). In its full form (see, for example, Bäck and Schwefel 1993), ES uses an  $[n \cdot n]$  matrix  $\mathbf{A}^{-1}$  of covariances in order to normalize different eigenvalues in arbitrary directions. Using a matrix  $\mathbf{A}^{-1}$  of covariances, ES yields a performance, which is independent of different eccentricities and arbitrary angles between the function’s principle axes and the coordinate system. In addition, different recombination mechanisms are commonly used in ES. For a discussion of different ES variants and other evolutionary algorithms see Bäck and Schwefel (1993).

The term evolutionary algorithms is not limited to GAs and ES, it also comprises other methods such as evolutionary programming (EP), differential evolution (DE), and

genetic programming (GP). EP has been proposed in Fogel (1962) and applications to general function optimization can be found in Fogel (1995). DE has been proposed in Storn (1995) and is an alternative to existing methods. Furthermore, hybrid methods such as collective learning (CL) proposed in Salomon (1994) can be considered as evolutionary algorithms. Since these algorithms are not in the focus of this paper, they are not described in more detail here.

Even though the BGA uses an uncommon design, data types, and operators, it yields more or less the same functionality as the SGA. The proposed mutation operator (1) yields a limited prespecified precision, i.e.,  $2^{-15}$ , and sets, on average, only one  $\alpha_i$  to the value 1. Since the BGA uses a mutation probability  $p_m = 1/n$ , the BGA's mutation is more or less equivalent to flipping one bit in a bit string with a probability  $p_m = 1/l$ , with  $l$  denoting the total number of bits. Thus, the chosen design of the BGA provides an abstraction from tedious implementation issues, but still behaves similar to the SGA's bit flipping mutation when setting  $p_m = 1/l$ . The remaining differences to the SGA are truncation selection and uniform recombination. These operators are probably the reasons for the very fast convergence of the BGA when applied to widely-used test functions.

## 4 Small Mutation Rates Lead to Notorious Problems

This section starts off with explaining some very important properties of the widely-used test functions. It then briefly summarizes typical parameter settings for the mutation probability  $p_m$ ; normally,  $p_m$  is set to very small values. It turns out that these small mutation probabilities are a perfect adaptation to the inherent decomposability property of the test functions. Finally, this section discusses some consequences of this adaptation. It more or less limits the application of GAs to decomposable functions, because the considered GAs become very inefficient if the fitness function is not decomposable.

Most of the widely-used test functions are *decomposable* in the following sense

$$f(\vec{x}) = \sum_{i=1}^n f_i(x_i) \quad (3)$$

Thus a function  $f(\vec{x})$  is decomposable into a sum of  $n$  not necessarily identical functions  $f_i$ . If a function  $f$  is decomposable, its parameters  $x_i$  are called *independent*. A

decomposable function  $f$  has the following important property

$$\begin{aligned} \forall i, j, i \neq j: \quad & f(\dots, x_i^o, \dots) = \text{opt} \quad \wedge \quad f(\dots, x_j^o, \dots) = \text{opt} \\ \Rightarrow \quad & f(\dots, x_i^o, \dots, x_j^o, \dots) = \text{opt} \end{aligned} \quad (4)$$

This property makes it easy to optimize decomposable functions, because optimization can be done in a sequence of  $n$  independent optimization processes yielding  $n$  optimal values for the  $x_i$ 's. The solution is then the combination of these values  $\vec{x}^o = (x_1^o, \dots, x_n^o)^T$ . If the decomposability property can be exploited, the inherent complexity is *only*  $O(n)$ .

Many other functions, such as  $f(\vec{x}) = (x_1^2 + x_2^2)^2$ , are not decomposable. But such functions are also very easy to optimize, since their first derivative is a product, such as  $\partial f(\vec{x})/\partial x_1 = 4x_1(x_1^2 + x_2^2)$ . Such a product yields a solution  $x_1 = 0$  that is independent of the other variable. More generally speaking, all functions  $f(\vec{x})$  that fulfill the following condition

$$\frac{\partial f(\vec{x})}{\partial x_i} = g(x_i) h(\vec{x}) \quad , \quad (5)$$

are as easy to optimize as decomposable functions, because they allow obtaining solutions for each  $x_i$  independently of all other parameters; the decomposability property (3) is a special case of (5).

Mostly, a small mutation probability in the order of  $p_m \approx 1/n$  (or  $p_m \approx 1/l$  respectively) is recommended, which has the consequence that mutation is applied, on average, to only one parameter per offspring.

De Jong (1975) investigates mutation rates between 0.1 and 0.001 and suggests an optimal mutation rate in the order of  $p_m = 1/\text{PopulationSize}$ . In most of his presented results, a mutation rate in the order of  $p_m = 1/l$  yields the best performance. In more recent research, Potter and De Jong (1994) explicitly recommend and use a mutation probability  $p_m = 1/l$ . In Goldberg (1989), the mutation rate is usually set to small values. In a particular application with a bit string of length  $l = 30$ , a mutation probability of  $p_m = 0.033 = 1/l$  was used.

In Mühlenbein and Schlierkamp-Voosen (1993a, 1993b), the BGA's mutation probability is always set to  $p_m = 1/n$ . Since the BGA abstracts from bit representations, the probability setting is independent of the chosen precision (cf. (1)).

Other research also recommends the same small mutation probability. Mühlenbein (1992), for example, derives that a mutation probability  $p_m = 1/l$  is optimal for any unimodal binary function. This would imply that  $p_m = 1/n$  is optimal for any unimodal real-valued function, if using BGA-like GAs. Unfortunately, this research

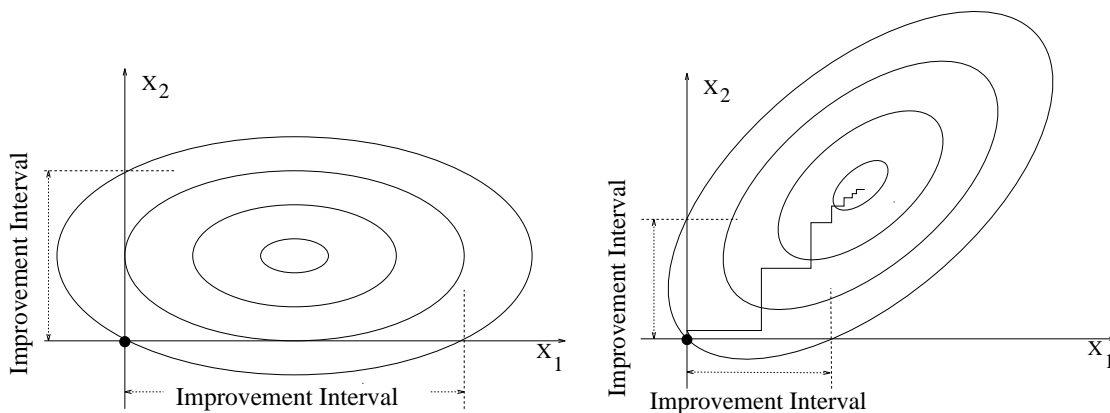
investigates only functions with independent parameters. Furthermore, it is suggested that the mutation probability should be increased to  $p_m = k/n$  for  $(k, l)$ -deceptive problems. The same result is obtained in Bäck (1993) for unimodal functions. But, in addition, it shows that an optimal probability scheduling can significantly accelerate convergence when optimizing multimodal functions.

Since the mutation probability is usually much smaller than the recombination probability, mutation is often considered less important. At the same time, however, mutation is often considered to be a constructive, diversity-generating operator (Goldberg 1994), or a restoring-of-lost-genetic-material operator (Srinivas and Patnaik 1994).

The following analysis shows that these small mutation rates are a perfect adaptation to the widely-used test functions. As already mentioned in (4), independent parameters allow for rapid optimization by decomposing the  $n$ -dimensional task into a sequence of  $n$  independent 1-dimensional task. Furthermore, the  $n$  1-dimensional optimization tasks can be scheduled in any arbitrary sequence; they can be performed one after the other or even interleaved. A GA with a small mutation probability does this scheduling by modifying only one parameter at a time. The subsequent selection process gives immediate feedback about the mutation's quality. This strategy works perfectly with respect to the  $O(n \ln n)$  complexity as long as the parameters are independent, but suffers a drastic performance loss as soon as the parameters become dependent on each other.

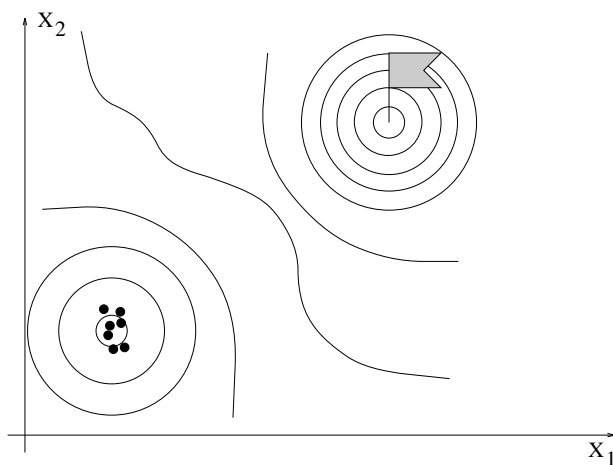
As mentioned above, *all* decomposable functions (whether unimodal or multimodal) can be solved with a  $O(n)$  complexity. Appendix A shows that many GAs that use a small mutation probability  $p_m \leq 1/n$  scale with  $O(n \ln n)$ . A  $O(n \ln n)$  complexity is strictly greater than  $O(n)$ , but seems optimal for a randomized algorithm. It is interesting to note, that uniform recombination leads to a significant acceleration of the convergence speed, but does not change the  $O(n \ln n)$  complexity at all as has been confirmed by extensive experiments and theory presented in Mühlenbein and Schlierkamp-Voosen (1993a, 1993b).

For the following discussion, it is useful to distinguish between unimodal and multimodal functions. Figure 1 explains the performance loss for unimodal functions. The left part shows a quadratic function of the form  $f(\vec{x}) = x_1^2 + \lambda_1 x_2^2$  that is aligned with the coordinate system. The right part shows the same function but rotated, which leads to  $f(\vec{x}) = x_1^2 + \lambda_0 x_1 x_2 + \lambda_1 x_2^2$ . It can be clearly seen that the improvement intervals are rapidly shrinking, which results in an increase of the algorithm's complexity. Figure 1 shows only a simple 2-dimensional example with a very small eccentricity. The observable performance loss is much higher in high-dimensional search space and with large eccentricities. In summary, mutual parameter dependencies in unimodal functions slow down convergence.



**Figure 1:** Left: a quadratic function which is aligned with the coordinate system has a relatively large improvement interval. Right: a rotation of the same function, which leads to much smaller improvement intervals. The difference is increasing as the eccentricity is increasing.

Figure 2 illustrates the situation for a typical multimodal function, where the local optima are not distributed on a grid aligned with the coordinate system. Assume that the fittest members of the population have converged to the lower-left local optimum. In this situation, the global optimum cannot be reached by modifying only one parameter per offspring. By contrast, all parameters have to be modified simultaneously. In the general  $n$ -dimensional case, the probability for this event is as small as  $p = (p_m)^n = 1/n^n = \exp(-n \ln n)$ , which is even smaller than the probability of finding the global optimum with random search. Thus, for optimizing multimodal functions with small mutation rates, it is essential for the considered GAs that the local optima are distributed on a grid parallel to the coordinate axes. Only then,



**Figure 2:** A 2-dimensional function, in which the local optima are not distributed on a grid aligned with the coordinate system. The population has converged to the lower-left local optimum. In this situation, applying mutation to only one parameter is not sufficient to yield any progress.

the algorithm can decompose the optimization task into a sequence of 1-dimensional optimization tasks yielding a  $O(n \ln n)$  complexity; otherwise the complexity increases up to  $O(\exp(n \ln n))$ , which is larger than the complexity of random search.

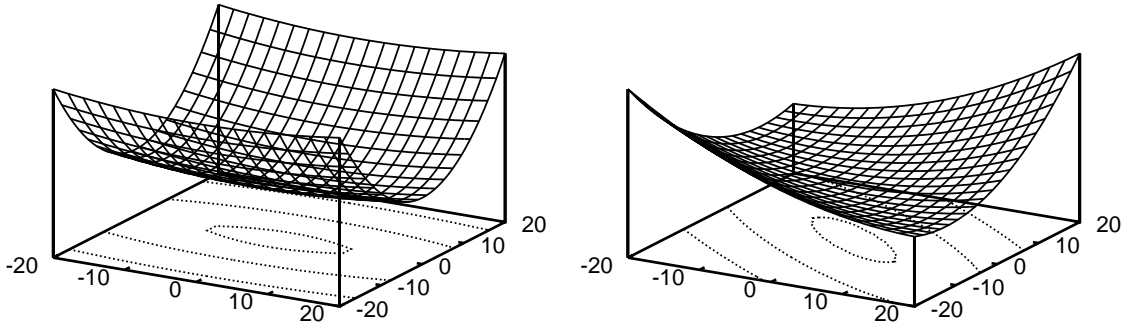
The analysis presented above, did not consider recombination explicitly. Without any doubt, recombination/crossover accelerates convergence significantly. But experiments as well as theory presented in Mühlenbein and Schlierkamp-Voosen (1993a, 1993b) have confirmed that recombination does not reduce the  $O(n \ln n)$  complexity when optimizing decomposable functions; see also Appendix A. Recombination, the combination of subsets of well-adapted parameters, works well, if the parameters are independent. But on dependent parameters, recombination cannot be expected to work with the same efficiency. Thus recombination is very likely to accelerate convergence even in the general case, but it probably cannot reduce the  $O(\exp(n \ln n))$  complexity of GAs featuring a small mutation rate.

## 5 Experiments

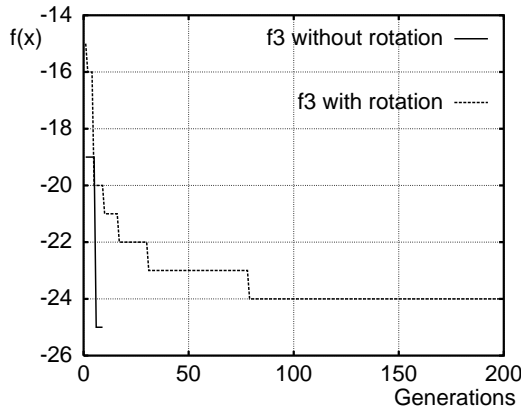
The previous section has argued that the design of a typical GA is very much adapted to the widely-used test functions and that a drastic performance loss has to be expected, if the parameters are mutually dependent. To this end, this section compares the BGA’s performance when applied to the test functions summarized in Table 1. All performance measurements were taken with and without a rotation of the coordinate system. Rotation means that a linear and orthogonal transformation matrix  $\mathbf{M}$  is applied such that  $f_i(\mathbf{M}\vec{x})$  is calculated. The transformation matrix  $\mathbf{M}$  is a pure rotation and the effect of  $\mathbf{M}$  is visualized in Figure 3, which shows the 2-dimensional version of function  $f_8$  with (right) and without (left) a rotation. It can be seen that a rotation does not change the function’s structure.

Figures 4 to 10 show typical runs when applying the BGA to widely-used test functions. It can be clearly seen that a rotation of the coordinate system causes a severe performance loss. It should be noted that the absolute number of generations is not the important concern; the numbers of generations can be influenced by changing the parameters of the GA. What is relevant is the performance loss caused by a rotation of the coordinate system while holding all other parameters constant. In all presented runs, the parameters of the GA were adopted from the literature. The population size for Figures 4 to 10 were 20, 20, 20, 500, 30, and 10 respectively.

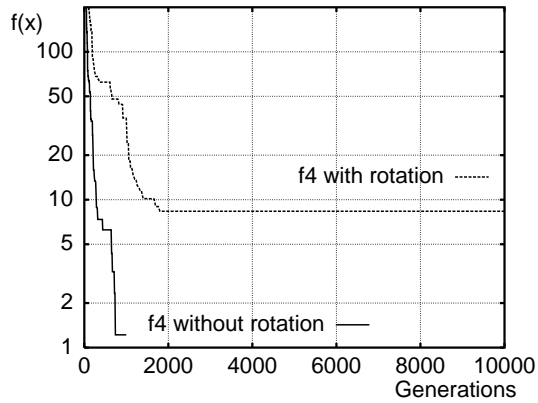
No figure is given for function  $f_1$ . Since this function is already a sphere, a rotation does not affect any change and, as can be expected, the performance is unaffected. Figure 4 shows the effect of a rotation on the five-dimensional function  $f_3$ . The applied



**Figure 3:** 3D and contour plot of the function  $f(x_1, x_2) = x_1^2 + 10x_2^2$  with (right figure) and without (left figure) a rotation of the coordinate system. It can be seen that the rotation does not change the structure of the function.



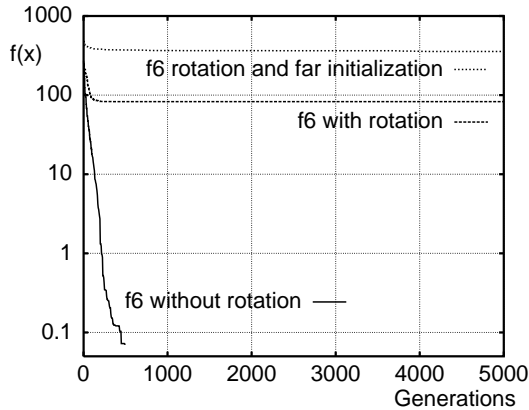
**Figure 4:** BGA's performance on  $f_3$ .



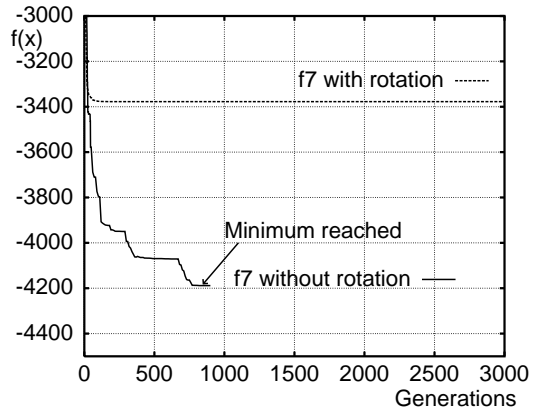
**Figure 5:** BGA's performance on  $f_4$ .

rotation slows down the BGA's performance by at least one order of magnitude. The results for function  $f_4$  are similar to those of  $f_3$ . A rotation slows down global convergence and the final fitness is much worse. Since these test functions have rather few parameters, the observable performance loss is moderate.

The observable performance loss is more significant when turning to multimodal functions, such as  $f_6$  and  $f_7$ . Figure 6 shows that, without rotation, the BGA converges very quickly to the minimum. With rotation, however, the BGA gets stuck more or less far away from the global minimum. In case of a rotation, the BGA yields little improvement; the final value is about 83. This relatively good value is due to the initialization of 40 different population members. Such an initialization places several individuals in the neighborhood of the global minimum's basin of attraction. Figure 6 also shows the BGA's performance, if all initial members are equally initialized (labeled with "far initialization" in Figure 6). In this case, the BGA gets stuck at a final value of about 356, which is very poor. The results for Schwefel's



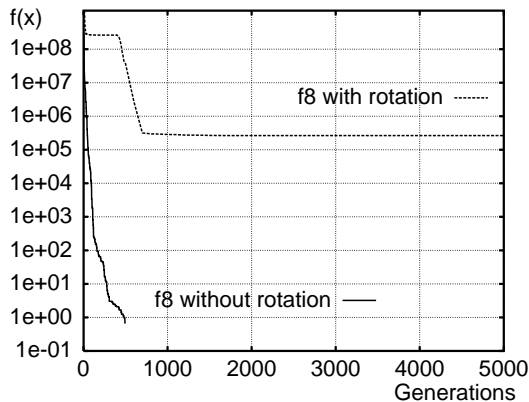
**Figure 6:** BGA's performance on Rastigrin's function  $f_6$ .



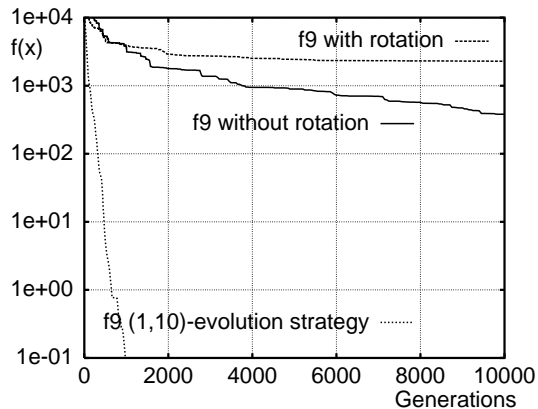
**Figure 7:** BGA's performance on Schwefel's function  $f_7$ .

function  $f_7$  are similar as can be seen in Figure 7. Without rotation, the BGA finds the global minimum very quickly (about 720 generations), whereas a rotation leads to a final function value of about -3377. In the final solution, only four parameters were close to the optimal values, whereas the other six parameters were far away. Further experiments have shown that the observable performance loss gets much worse if the number of parameters increases. These results show that a rotation of a multimodal function causes a severe performance loss, which cannot be compensated with small mutation rates, even if using a high-performance GA like the BGA.

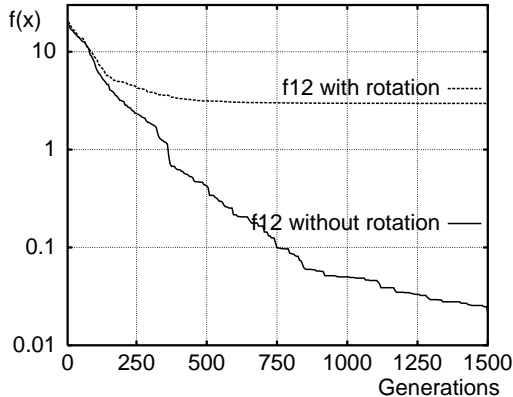
Figures 8 and 9 indicate two more examples. Figure 8 shows the BGA's performance when applied to function  $f_8$ . Function  $f_8$  is similar to function  $f_1$  and  $f_4$ , but it does not contain noise and the difference of the eigenvalues is much higher. Even though function  $f_8$  is a simple unimodal function (hyperellipsoid) a rotation significantly slows down convergence. With rotation, the final function value is not better



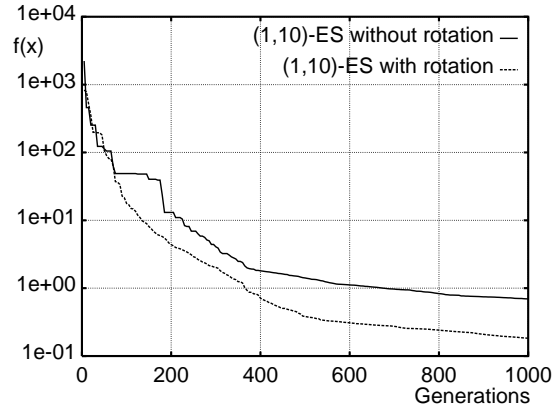
**Figure 8:** BGA's performance on the hyper ellipsoid  $f_8$ .



**Figure 9:** BGA's performance on the ridge  $f_9$ .



**Figure 10:** BGA's performance on Ackley's function  $f_{12}$ .



**Figure 11:** Performance of ES on a 5-dimensional version of  $f_8$ .

than approximately  $2.65E+05$ , which is about *six orders of magnitude* worse than the final version without rotation. Figure 9 shows the BGA's performance when applied to function  $f_9$ . Even though the original version of  $f_9$  is not aligned with the coordinate system, a rotation further slows down the convergence process. Figure 9 also shows the performance of a simple (1,10)-ES. Besides the fact that a (1,10)-ES is computationally less expensive, the ES is much faster and, in addition, rotationally invariant. A (1,10)-ES drops the function value below 0.066 within 1000 generations. This performance difference gives further evidence that BGA-like algorithms rely on an alignment of the fitness function's principal axes with the coordinate system.

Furthermore, the BGA's performance is very poor when applied to function  $f_{10}$ . Even though  $f_{10}$  is rotationally invariant, the BGA gets stuck in one of the few local minima. With a population size of 40, the final fitness value after 1000 generations is usually about 0.4, which means that there are four more local optima to overcome. This behavior is caused by the fact that, when approaching the minimum, all parameters  $x_i$  have to be mutated simultaneously in order to obtain any progress.

Figure 10 shows the performance loss of the BGA when applied to Ackley's function ( $f_{12}$ ). Without rotation, the BGA converges very fast, whereas a rotation causes problems to the BGA. An additional comparison of an SGA with EP and two variants of ES can be found in Bäck and Schwefel (1993). When optimizing Griewangk's function, the observable performance loss is much less significant. The main reason seems to be the epistasis that is already incorporated by the second term ( $\prod_{i=1}^{10} x_i/\sqrt{i}$ ). This might also explain why the considered GAs need a relatively large population of 600 individuals for only 20 parameters. In comparison, when optimizing the nonrotated version of Rastrigin's function in 1000 dimensions, 20 individuals are sufficient.

The discussed observation is further supported by considering Rosenbrock's function

( $f_2$ ). The minimum  $\vec{x}_o = (1, 1)^T$  of Rosenbrock's function is located in a narrow curved valley, which is similar to a rotated version of function  $f_8$ . Indeed, applying a rotation of approximately 40 degrees brings the function's principal axes close to the the coordinate system and accelerates convergence. The resulting speed-up is even higher if optimization starts at (0,0), as has been done by others (e.g., Ostermeier et al. 1994).

Evolutionary algorithms, such as ES and CL, are designed in a way such that they are rotationally invariant. Figure 11 exemplifies this for a simple (1,10)-ES when optimizing the 5-dimensional version of function  $f_8$ . Both runs (with and without a rotation) are almost identical; the differences are due only to statistical fluctuations.

The results can be summarized as follows. Using a small mutation rate, a GA performs very well if the principal axes of the objective function are aligned with the coordinate system. The cooperative coevolution approach, as proposed by Potter and De Jong (1994), exploits the alignment by maintaining one subpopulation $_i$  (species) for each single parameter  $x_i$ . Each single species is optimized in a round-robin fashion. For the evaluation of each member in a subpopulation $_i$ , the parameter  $x_i$  is combined with the best individuals of all other subpopulations $_j$ . By these means, the cooperative coevolution approach clearly scales with  $O(n)$  when optimizing decomposable functions. It should be clear, however, that this approach is very specialized and that its application to dependent parameters is rather limited. It has been shown that a rotation significantly slows down convergence, even when optimizing unimodal functions. The difference in convergence speed depends on the ratio of the smallest and largest eigenvalue (cf.  $f_1$ ,  $f_4$ , and  $f_8$ ). When optimizing multimodal functions, a rotation has even more severe consequences. In such cases (cf.  $f_6$ ,  $f_7$ , and  $f_{12}$ ), the algorithm prematurely converges to a point far away from the global optimum. Furthermore, the more parameters the BGA has to optimize, the larger the performance loss. Other local optimization procedures, such as ES, EP, or gradient descent methods, are rotationally independent and, thus, their performance is not affected by a rotation. Further experiments have shown that this problem cannot be solved by merely varying the mutation and recombination probabilities.

## 6 Discussion

The focus of this paper is the GA's performance with respect to small mutation rates. The analysis has shown that a small mutation rate in the order of  $p_m \approx 1/n$  or  $p_m \approx 1/l$  respectively is a perfect adaptation to the widely-used test functions. On these test functions, many GAs scale with  $O(n \ln n)$ , even without recombination. By applying mutation statistically to only one parameter per offspring, such GAs decompose the

optimization task into  $n$  independent 1-dimensional tasks. This decomposition is possible since the parameters of the considered test functions are independent. Thus the reason for the very good performance on such test functions is not – as usually believed – that the local minima are distributed on a regular grid which is easy to solve by recombination.

Unfortunately, small mutation rates cause severe problems, if the parameters of the fitness function are mutually dependent. An  $n$ -fold rotation of the coordinate system causes a severe performance loss, since a rotation makes the parameters dependent on each other. When optimizing a multimodal function with *dependent* parameters, the complexity can increase up to  $O(n^n) = O(\exp(n \ln n))$ .

The design of other methods, such as ES, EP, steepest descent, or collective learning, is intrinsic rotationally invariant. One thought is to make GAs rotationally invariant by setting the mutation probability to  $p_m = 1$ . This idea is also supported by Mühlenbein (1992). The main result states that a mutation rate  $p_m = k/l$  is optimal for  $(k, l)$ -deceptive functions, and the general case of mutually dependent parameters can be seen as a fully deceptive function. But a mutation rate  $p_m = 1$  would lack a dynamic adaptation of the mutation strength. Rechenberg (1973) has already shown that reasonable progress in the optimization process is attainable only in a small evolution window (i.e., step size range). This window, of course, depends on the function and the state of the optimization process, and, in addition, it changes frequently during the optimization process. A dynamic self-adaptation mechanism could be incorporated into a genetic algorithm to control the effect of the mutation operator and could set  $p_m = 1$ . But in this case, the genetic algorithm would essentially be an evolution strategy; the genetic algorithm would be practically identical to the evolution strategy except for a different mutation probability distribution. The idea of an optimal mutation rate scheduling has been already discussed in Bäck (1993). It has been shown that an optimal mutation rate scheduling can accelerate convergence. However, the analysis presented in Bäck (1993) places emphasis on the multimodality caused by different bit-to-parameter mapping functions and not, as has been discussed in this paper, on multimodal functions defined by  $n$  parameters  $x_i$ .

Although genetic algorithms have been applied with some success to diverse optimization problems, the current work suggests that they are most suitable for problems which are linearly separable. This, then, is “crossover’s niche” (cf. Eshelman and Schaffer 1993). But the real world is not linearly separable; the physical relationships between elements described in biology, chemistry, physics, and other disciplines are often nonlinear, chaotic, temporal, and stochastic. Therefore, although genetic algorithms may be used with some degree of success on these problems, that success is tempered by the degree to which the problems may be approximated by linear separable relationships between the parameters of concern. Eshelman and Schaffer

(1993) offered that their “hunch is that the unique niche for pair-wise mating is much smaller than most GA researchers believe.” The current work suggests this niche is even smaller, because for any particular problem that is truly linearly separable there are many algorithms that can be used to solve that problem with greater efficiency than is afforded by a genetic algorithm, and for many real-world problems, there are other stochastic optimization algorithms that can be applied to greater advantage.

**Acknowledgements:** The author gratefully acknowledges Nikolaus Almassy and Hans-Georg Beyer for helpful discussions. Special thanks are due to David Fogel for detailed discussions as well as for his editorial assistance.

## 7 References

- Bäck, T., 1993, Optimal Mutation Rates in Genetic Search, in: *Proceedings of the Fifth International Conference on Genetic Algorithms*, S. Forrest (ed.) (Morgan Kaufmann, San Mateo, CA), pp. 2-8.
- Bäck, T., Schwefel, H.-P., 1993, An Overview of Evolutionary Algorithms for Parameter Optimization, *Evolutionary Computation* 1(1), pp. 1-23.
- De Jong, K.A., 1975, An Analysis of the Behavior of a Class of Genetic Adaptive Systems, Ph.D. Thesis (University of Michigan).
- Eshelman, L.J., Schaffer, J.D., 1993, “Crossover’s Niche”, in: *Proceedings of the Fifth International Conference on Genetic Algorithms*, S. Forrest (Ed.), (Morgan Kaufman, San Mateo, CA), pp. 9-14
- Fogel, D.B., 1995, *Evolutionary Computation: Toward a New Philosophy of Machine Learning Intelligence*, IEEE Press, Piscataway, NJ.
- Fogel, L.J., 1962, “Autonomous Automata”, *Industrial Research*, vol. 4, pp. 14-19.
- Gill, P.E., Murray, W., Wright, M.H., 1981, *Practical optimization* (Academic Press, London).
- Goldberg, D.E., 1989, *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison-Wesley Publishing Company).
- Goldberg, D.E., 1994, Genetic and Evolutionary Algorithms Come of Age. *Communications of the ACM* 37, pp. 113-119.
- Hoffmeister, F., Bäck, T., 1990, Genetic Algorithms and Evolution Strategies: Similarities and Differences, in: *Parallel Problem Solving from Nature 1*, H.P. Schwefel and R. Männer (eds.) (Springer-Verlag), pp. 455-469.

- Luenberger, D.G., 1984, *Linear and Nonlinear Programming* (Addison-Wesley Company, Menlo Park, CA).
- Mühlenbein, H., Schomisch, M., Born, J., 1991, The Parallel Genetic Algorithm as Function Optimizer, in: *Proceedings of the Fourth International Conference on Genetic Algorithms*, R.K. Belew and L.B. Booker (eds.) (Morgan Kaufman Publisher).
- Mühlenbein, H., 1992, How genetic algorithms really work I: Mutation and hill climbing, in: *Proceedings of Parallel Problem Solving from Nature 2*, H.P. Schwefel and R. Männer (eds.) (Springer-Verlag), pp. 15-26.
- Mühlenbein, H., Schlierkamp-Voosen, D., 1993a, Predictive Models for the Breeder Genetic Algorithm I, *Evolutionary Computation* 1(1):25-50.
- Mühlenbein, H., Schlierkamp-Voosen, D., 1993b, The Science of Breeding and its Application to the Breeder Genetic Algorithm. *Evolutionary Computation*, 1(4):335-360.
- Ostermeier, A., Gawelczyk, A., Hansen, N., 1994, A Derandomized Approach to Self-Adaptation of Evolution Strategies. *Evolutionary Computation* 2(4), pp. 369-380.
- Potter, M.A., De Jong, K.A., 1994, A Cooperative Coevolutionary Approach to Function Optimization, in: *Proceedings of Parallel Problem Solving from Nature 3*, Y. Davidor, H.P. Schwefel, and R. Männer (eds.) (Springer-Verlag), pp. 249-257.
- Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T., 1987, *Numerical Recipes* (Cambridge University Press).
- Rechenberg, I., 1973, *Evolutionsstrategie* (Frommann-Holzboog, Stuttgart).
- Rosenbrock, H.H., 1960, An automatic method for finding the greatest or least value of a function (*Computer J.* **3**), pp. 175-184.
- Salomon, R., 1994, Improved Global Convergence by Means of Collective Learning, in: *Proceedings of The Third Annual Conference on Evolutionary Programming*, A.V. Sebald and L.J. Fogel (eds.) (World Scientific Publishing, River Edge, NJ), pp. 34-41.
- Salomon, R., 1996. Implicit Independence Assumptions; a Notorious Problem for Genetic Algorithms. To appear in *Soft Computing, SOCO'96*, Reading, England.
- Schwefel, H.P., 1977, *Numerische Optimierung von Computer-Modellen mittels Evolutionsstrategie* (Birkhäuser Verlag, Basel and Stuttgart).

- Schwefel, H.P., 1995, *Evolution and Optimum Seeking* (John Wiley and Sons, Inc, New York, Chicester, Brisbane, Toronto, Singapore).
- Srinivas, M., Patnaik, L., 1994, Genetic Algorithms: A Survey, Computer, pp. 17-26.
- Storn, R., 1995, Differential Evolution Design of an IIR-Filter with Requirements for Magnitude and Group Delay (submitted for publication to the *Evolutionary Computation* Journal). Technical Report TR-95-026, ICSI, ftp.icsi.berkeley.edu.
- Whitley, D., Gordon, V.S., Mathias, K., 1994, Lamarckian Evolution, The Baldwin Effect and Function Optimization, in: *Proceedings of Parallel Problem Solving from Nature 3*, Y. Davidor, H.P. Schwefel, and R. Männer (eds.) (Springer-Verlag), pp. 6-15.

## Appendix A

This appendix derives the convergence speed of a broad class of GAs when applied to decomposable functions.

The notion of *broad mutations* is an important prerequisite for the derivation of the complexity. Let  $pd(\vec{x}|\vec{x}_p)$  the probability density that mutation creates offspring  $\vec{x}$  given parent  $\vec{x}_p$ . A mutation operator yields broad mutations, if for *all* possible parents  $\vec{x}_p$  and *all* possible offspring  $\vec{x}$  in search space the following condition holds

$$pd(\vec{x}|\vec{x}_p) > 0 \quad . \quad (6)$$

Thus, broad mutations generate mutation in entire search space with a probability strictly greater than zero. Flipping bits with a certain probability is an example for broad mutations. In addition, some implementations of the BGA's mutation operator are broad mutations as well.

The following derivation of the convergence speed and its complexity makes the following four assumptions: (1) the objective function is decomposable, i.e.,  $f(\vec{x}) = \sum_{i=1}^n f_i(x_i)$ , (2) a GA uses an elitist selection scheme (preserving the best solution found so far), (3) the GA features an arbitrary broad mutation operator with probability  $p_m = 1/n$ , and (4) the GA does not feature any recombination, i.e.,  $p_r = 0.0$ .

First, consider the one-dimensional case. Since the probability density  $pd(\vec{x}|\vec{x}_p)$  is strictly greater than zero, the probability  $p$  that an offspring  $\vec{x}$  is in the  $\epsilon$ -neighborhood of the optimum is greater than zero for arbitrary parents  $\vec{x}_p$ . Eventually, a GA converges to the optimum by applying mutation and using an elitist selection scheme.

The number of generations and, thus, the convergence speed merely depend on the probability density associated with the mutation operator and the one-dimensional objective function. Hence, for any given combination, the number of required generations is bounded and, thus, the complexity is  $O(1)$ .

Now consider the  $n$ -dimensional case. For the sake of simplicity, it is assumed that one iteration is sufficient for a parameter  $x_i$ ; details for the case  $k > 1$  can be found in Salomon (1996). A mutation probability of  $p_m \leq 1/n$  implies that, on average, a GA modifies at most one parameter per offspring. By this means, such GAs decompose the optimization of a *decomposable*  $n$ -dimensional function into a sequence of  $n$  independent one-dimensional tasks. Since the complexity of the one-dimensional case is  $O(1)$ , convergence in the  $n$ -dimensional case is given by the probability  $p$  that all parameters  $x_i, 1 \leq i \leq n$  have been chosen at least once. The probability  $q_i$  that a particular parameter  $x_i$  has *not* been chosen within  $g$  generations is given by  $q_i = [(n-1)/n]^g$ . Consequently, the probability  $p_i$  that it has been chosen at least once is  $p_i = 1 - q_i = 1 - [(n-1)/n]^g$ . Since the choice of each parameter  $x_i$  is independent of all other parameters,  $p$  can be expressed as follows

$$\begin{aligned} p &= \prod_{i=1}^n p_i = \prod_{i=1}^n \left[ 1 - \left( \frac{n-1}{n} \right)^g \right] \\ p &= \left[ 1 - \left( \frac{n-1}{n} \right)^g \right]^n \end{aligned}$$

In order to guarantee that all  $n$  parameters have been chosen at least once within  $g$  generations, the number of generations  $g > n$  has to be larger than  $n$ . Thus the term  $[1 - ((n-1)/n)^g]^n$  can be approximated by  $1 - n [(n-1)/n]^g$

$$p \approx 1 - n \left( \frac{n-1}{n} \right)^g \quad . \quad (7)$$

Solving (7) for the number of generations  $g$  leads to

$$g = \frac{\ln(1-p) - \ln n}{\ln(\frac{n-1}{n})} \quad . \quad (8)$$

For arbitrary  $n$  and a constant probability  $p$  the complexity is given by (all constant terms vanish)

$$g = O \left( \frac{-\ln n}{\ln(\frac{n-1}{n})} \right) \quad . \quad (9)$$

For large  $n$ , the value of  $\ln(\frac{n-1}{n})$  approaches 0. A Taylor sequence of  $\ln x$  at  $x = 1$  leads to  $\ln x \approx x - 1$ . Substituting this approximation into (9) results in

$$g = O \left( \frac{-\ln n}{\frac{n-1}{n} - 1} \right) = O \left( \frac{-\ln n}{\frac{-1}{n}} \right) = O(n \ln n) \quad . \quad (10)$$

In summary, the derived complexity holds for decomposable functions whether they are unimodal or multimodal. Also, the derivation holds for  $(k, l)$ -deceptive functions, at least for the BGA. For other mutation operators, such as bit flipping, the complexity can be increased up to  $O(n^k \ln n)$ .

Uniform recombination/crossover accelerates convergence by combining already optimized parameters of different objects into new combinations. According to Goldberg (1989), schemata with a long defining length have a high probability of being destroyed by recombination. In the context of uniform recombination, a schemata can be considered as the number of already optimized parameters. Thus, one can expect progress only if either the schemata are short, or if both recombining individuals are very similar. After the first few generations, the population starts converging to a global equilibrium, where all individuals become similar to each other. When applying mutation to only one parameter per offspring, uniform recombination can combine at most two new parameters  $x_i$  and  $x_j, i \neq j$  into a new offspring. Therefore, the number of parameters is virtually reduced to  $n/2$ , but the resulting complexity

$$g = O\left(\frac{n}{2} \ln \frac{n}{2}\right) = O(n \ln n) \quad (11)$$

remains unchanged. This result states that recombination does not change the complexity of such GAs. But in practical applications, recombination can speedup convergence significantly by a constant factor, which is important for computational resources.

## Appendix B

This appendix gives a short outline of a program to create the rotation matrix  $M$  implemented in a pseudo programming language.

Variables of type Matrix[ N, N ] of type double

```
ME,          /* diagonal matrix, i.e.,
              ME[ i, i ] = 1, ME[ i, j ] = 0 */
MTurn,      /* misc matrix */
MResult;    /* result */
```

Functions

```
MulMatrix( M1, M2 )
```

```

IS
    M1 := M1 * M2;
END MulMatrix;

MRot( MTurn, i, j )
    /* creat a single rotation matrix */
IS
    MTurn := ME; /* set MTurn to the matrix ME */
    alpha := (drand48() - 0.5) * M_PI * 0.5;
    /* draw a random angle by
       means of a UNIX syscall */
    MTurn[ i, i ] := cos( alpha );
    MTurn[ j, j ] := cos( alpha );
    MTurn[ i, j ] := sin( alpha );
    MTurn[ j, i ] := - sin( alpha );
END MRot;

CreateMatrix( MResult, N )
    /* main routine for matrix creation */
IS
    MResult := ME; /* initialize MResult */
    srand48( OL ); /* initialize UNIX random
                   number generator */
    FOR i := 2 STEP 1 UNITL N
    DO
        MRot( MTurn, 1, i );
        MulMatrix( MResult, MTurn )
    END FOR;
    FOR i := 2 STEP 1 UNITL N-1
    DO
        MRot( MTurn, i, N );
        MulMatrix( MResult, MTurn )
    END FOR;
END CreateMatrix;

```