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 Zürich
Winterthurerstrasse 190, 8057 Zurich, Switzerland
FAX: +41-1-363 00 35; Email: salomon@ifi.unizh.ch

Abstract

This work analyzes some concepts of genetic algorithms and explains why they may be applied with success to some problems in function optimization. In addition to other performance properties, it has been shown that genetic algorithms are able to overcome local minima in highly multimodal functions (e.g., Rastrigin, Schwefel). The performance of genetic algorithms is supported by an extensive theory, which is based on the assumption of additive gene effects. But the current work shows that the assumption of additive gene effects is not weak, and that the dependence on specific parameter settings is much stronger than often believed. Furthermore, the assumptions regarding the fitness function are so restricting that slight modifications of the standard test functions cause a failure of the optimization procedure even though the function's structure is preserved. The current experiments focus on a few widely-used scalable test functions. the results indicate that a standard genetic algorithm can find the global optimum of Rastrigin-like functions *not* due to the efficiency of the *recombination* operator and the regular distribution of local minima, but because the optimization is decomposable into n independent one-dimensional subproblems. As a logical consequence of the analysis, it is shown how the function's properties can be exploited in order to construct a more efficient algorithm.

Keywords
GENETIC ALGORITHMS – FUNCTION OPTIMIZATION – PERFORMANCE
COORDINATE ROTATION

1 Introduction

A genetic algorithm (GA) is a heuristic search procedure based on natural selection and genetics, which maintains a population of trail solutions. Genetic algorithms (GAs) are suitable for addressing many function optimization problems. Recent developments such as the parallel genetic algorithm [10] and the breeder genetic algorithm [12, 13] have been shown to be successful in optimizing multimodal functions. It has been reported that such GAs overcome millions of local optima and can converge to the global optimum [6, 12, 13, 19]. In [12], the results of the breeder genetic algorithm have been supported by an comprehensive theory. An extensive introduction to GAs can be found in [6, 19]. In the introduction, consideration is limited to the simple genetic algorithm (SGA). A more detailed description of the breeder genetic algorithm can be found in section 2. The SGA works on a population of fixed bit stings. First, it selects some individuals as parents for the next generation. Next, it mates two different parents to produce two new offspring by means of mutation (i.e. bit inversion) and recombination (i.e. exchange of partial bit strings). In the final evaluation phase of each generation, it assigns the resulting fitness value to its individuals.

As an alternative to GAs and traditional methods such as steepest descent [9], hybrid algorithms, such as collective learning [17], have been proposed. Such hybrid methods, in general, combine different techniques in order to exploit their synergistic effects. Collective learning (CL), for example, features a self-adapting steepest descent and a GA or evolution strategy [15] respectively. One surprising aspect is that such a hybrid algorithm is not much faster than the breeder genetic algorithm when applied to some common test functions, even though collective learning uses hill climbing to accelerate the rate of convergence. The current work explains why this difference is so small and, in addition, shows that the difference in performance increases rapidly under coordinate rotation of the benchmark functions.

An extensive theory [7, 11, 12, 13, 19] has been developed to explain how GAs are believed to work. According to the theory, it is doubtless that mutation, recombination, and selection are essential components for a GA. But the degree of their relative importance depends on the type of application as well as how much optimization has been done by the algorithm (i.e., the relative importance might be changing during the course of optimization).

The remainder of the paper is organized as follows. Section 2 describes some well-known test functions used for experimentation. The major focus is on multimodal functions to verify the GA's global convergence behavior. Section 3 describes briefly some evolutionary computation (EC) algorithms, which can be seen to include genetic

algorithms, evolutionary strategies, evolutionary programming, and genetic programming. Section 4 discusses the GA's operators in detail. This section answers the question under which circumstances a GA works successfully and consequently points out the requirements that a fitness function must fulfill. The dependencies between the requirements of the fitness function and attainable performance is much stronger than literature has indicated. Given that a function has the necessary properties, section 5 shows how to exploit these requirements to construct a faster optimization procedure which scales with $O(n)$.

2 Function Test Bed

This work uses well-known test functions, summarized in Table 1. The first two of these test functions are unimodal and the remaining two are multimodal.

Function f_1 is a simple quadratic parabola with different eigenvalues along each axis. The graphs of constant function values appear as n-dimensional hyperellipses. For such functions it is well known that the convergence speed crucially depends on the ratio of the smallest and largest eigen value [14]. Function f_2 was proposed by Rosenbrock and a contour plot is shown in Figure 1. Rosenbrock's function is considered to be difficult because of its narrow curved valley containing the minimum at $\vec{x} = (1, 1)$. A long discussion on optimizing this function can be found in [5]. Function f_3, proposed by Rastrigin, is highly multimodal and has its minimum at $\vec{x}_m = \vec{0}$ and $f_3(\vec{x}_m) = 0$. Most optimization procedures have difficulties converging close to the minimum, because the probability of making progress decreases rapidly as the minimum is approached; most stagnate in its vicinity (see, for example [8]). The last function f_4 was proposed by Schwefel and further details can be found in [10]. This function poses the difficulty that the global minimum is given by $\forall\, i\ x_i = 420.9687$

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

	Function	Limits
1	$f_1(\vec{x}) = \sum_{i=1}^{5} 10^{i-1} x_i^2$	$-10.0 \leq x_i \leq 10.0$
2	$f_2(x_1, x_2) = 100\,(x_2 - x_1^2)^2 + (1 - x_1)^2$	$-1.5 \leq x_i \leq 1.5$
3	$f_3(\vec{x}) = 20 + \sum_{i=1}^{20} x_i^2 - \cos(2\,\pi\, x_i)$	$-5.12 \leq x_i \leq 5.12$
4	$f_4(\vec{x}) = \sum_{i=1}^{10} -x_i \sin(\sqrt{\lvert x_i \rvert})$	$-500 \leq x_i \leq 500$

and that the second best minimum $x_j = -302.5253, \forall i \; i \neq j, x_i = 420.9687$ is far away from the global optimum [10].

3 Review of Five Optimization Procedures

This section briefly reviews five optimization procedures based on genetic algorithms or evolution strategies. It concludes by presenting a short comparison of results when applying these procedures to Rastrigin's function (f_3).

A problem or function is described by a set of parameters. In the domain of continuous parameter optimization the problem is to find a set of floating-point values associated with the extrema of a given fitness function. When using a GA or any other computer-aided method to find the solution of such a problem, an appropriate encoding mechanism is required. As mentioned above, a GA typically operates on sets of bit strings. There are many conceivable ways how to encode a set of parameters into bit strings. One common technique uses the given 64- or 32-bit representation of the underlying (IEEE) format of floating-point values; i.e., n floating-point parameters are encoded into $64 \cdot n$ bits in a straight-forward way. To establish the necessary mutation operator, single bits of the bit string can be flipped with a preselected probability. But, this is only one of several possible representations. For example, research in [13] does not consider bit strings. Their GA works directly on a set of floating-point values and the mutation operator adds or subtracts small random values according to a given probability distribution.

The following notation is used throughout this paper.

- n is the number of floating-point parameters to be optimized
- m_p is the probability that a parameter is modified by mutation
- r_p is the probability for recombination. All procedures described below use uniform recombination where each two corresponding parameters are exchanged with the probability r_p
- FE is the number of function evaluations
- GE is the number of generations
- N is the population size of the GA. In evolution strategies, the population size is given by the number of parents μ and the number of offspring λ. This notion leads to slightly different selection schemes which are explained later

To begin, consider the breeder genetic algorithm (BGA) [12]. This genetic algorithm uses the standard operators of mutation and recombination, it is one of the most

efficient genetic algorithms for optimizing multimodal functions (such as f_3 and f_4), and an extended theory has been proposed that verifies various practical results. In each generation, the BGA uses an elitist truncation selection, i.e., an arbitrary $T\%$ best individuals are selected as the parents for the next generation. Elitist selection means that the best individual will remain in the population. To produce offspring, the BGA selects two different parents and applies recombination as well as mutation. BGAs us uniform recombination, where each parameter can be exchanged with the corresponding parameter of the other object. Mutation is established by the following random generator[1] [12]:

$$\sigma = \sum_{i=0}^{15} \alpha_i \, 2^{-i} \tag{1}$$

where each α_i is set with a probability of 1/16 to 1 and with a probability of 15/16 to 0 respectively. Such random numbers are multiplied by the tenth of search interval for the corresponding parameter. As common in the field of genetic algorithms, the BGA uses a mutation probability $m_p = 1/n$ and a recombination probability $r_p = 0.5$. Furthermore, the BGA does not use any further crossover operator nor does it use any local hill climbing.

For supplemental results a genetic algorithm was developed for the current work which is similar to the BGA, but not identical. It varies in the selection scheme, where the μ best individuals are chosen as parents for the next generation out of the λ offspring *and* the μ parents of the last generation; i.e., it uses an elitist selection scheme. This algorithm is denoted as MGA (for "modified genetic algorithm").

As another randomized algorithm to be considered is the evolution strategy (ES) developed by Rechenberg and Schwefel [15, 18]. The $(\mu \dot{+} \lambda)$-evolution strategy (ES) has μ parents and produces λ offspring in each generation. In contrast to a genetic algorithm, the evolution strategy maintains a global step size σ for each individual. Mutation is accomplished in two phases. In the first phase, the inherited global step size is modified at random by a number, which is close to 1.0, e.g., one of the following three numbers: 1.0, 1.3, or 0.7. In the second phase, mutation is performed by adding $(0,\sigma)$-normal distributed random numbers to each parameter. Note that the mutation probability is set to $m_p = 1$ opposed to $m_p = 1/n$ in most GAs, i.e., the ES modifies all parameters simultaniously. By means of inheritance and modification of the global step size for each individual, the ES establishes a self-adaptation mechanism. The main idea is that, statistically, those individuals will survive who have the most appropriate current step size, and consequently the ES self-adapts the step size over time to a nearly optimal value. The ES uses recombination as well [15], if considered to be appropriate for the problem at hand. In a $(\mu+\lambda)$ evolution strategy, the parents

[1] This approach in not very common, even though it is straight forward for this kind of application. In general, other procedures [1, 6, 10, 19] simply flip bits with the probability $m_p = 1/n$, where n is the total number of bits. However, the results reported in [12] are superior to those in [10].

for the new generation are chosen from the union of the former parents and the new offspring, whereas in (μ, λ)-evolution strategy the parents are selected from the new offspring only.

Evolutionary programming (EP) [3] is another method of simulating evolution thst should be considered. EP was originally developed for time series prediction as a precursor to generating adaptive behavior in changing environments, but has more recently been applied to general function optimization [4]. When applied to real-valued continuous optimization problems, EP is very similar to ES, despite developing completely independently until 1992. There are, however, two important differences. First, EP operates as an abstraction at the level of species and therefore typically does not use recombination. Second, EP typically uses a stochastic selection mechanism such that the probability for a solution surviving into the next generation is related to its ranking in the population. By means of this stochastic selection scheme, EP has a high probability of escaping from local optima and converging to the global optimum. A longer discussion of this prommising optimization procedure is beyond the scope of this paper.

Finally, a hybrid technique called collective learning (CL) [17] is also considered. Collective learning was originally based on an ES but has been extended to GAs as well. Therefore, CL uses a selection scheme similar to ES. Two variants are denoted as CL^{ES} and CL^{GA} respectively. Collective learning maintains μ (e.g., $\mu = 50$) and uses a GA or ES to produce λ (e.g., $\lambda = 8$) offspring in each generation. All members – parents as well as offspring – of the entire population perform exactly one optimization step per generation by means of a self-adapting local hill climbing method [16]. This approach leads to $\mu + \lambda$ function evaluations per generation. The selection scheme is somehow unusually. It selects all individuals that are below a certain age a (e.g., $a = 6$) and fills the remaining $[\mu - (\lambda\, a)]$ slots by the best old individuals. By means of this scheme, promising new individuals have sufficient time to adapt to the fitness landscape without being under any selection pressure. In both versions (CL^{GA} and CL^{ES}), CL sets $m_p = 1$; if using a genetic algorithm, CL^{GA} changes each bit of the mantissa with a probability of 0.1. Normally, both probabilities are considered to be too high for a pure genetic algorithm. But they appear appropriate in such a hybrid method.

Table 2 shows a comparison of the number of function evaluations FE of the breeder genetic algorithm (BGA), the modified genetic algorithm (MGA), and collective learning (CL^{GA}) when applied to Rastrigin's function f_3. All results presented for the MGA and CL^{GA} are average numbers of 20 trials; the results for the BGA are depicted from [12]. CL^{GA} appears to work best, the BGA second best, and the MGA worst. But, the difference between the MGA and the BGA is merely about 15 %. Because the implementation of the MGA so closely resembles to the BGA's, the difference of 15 %

	$n = 20$	$n = 50$	$n = 100$	$n = 200$	$n = 400$
BGA	3608	–	25040	52948	112634
MGA	4572	9334	30094	59717	133991
CL^{GA}	4600	8650	12650	23750	31150

Table 2: Comparison of the number of function evaluations FE needed by the breeder genetic algorithm (BGA), the modified genetic algorithm (MGA), and collective learning (CL^{GA}) when applied to Rastrigin's function for several dimensions (20 to 400). The results for the MGA and CL^{GA} are average of 20 trials and the results for the BGA are depicted from [12] (there are no results reported for the BGA in the case $n = 50$). MGA is about 15 % slower than BGA, but both procedures have almost the same scaling behavior. The stopping criterion was $f_3 < 0.1$. There are no results for the pure $(\mu + \lambda)$ evolution strategy reported, because ES is not suitable for optimizing Rastrigin's function.

in the number of function evaluations is likely only due to the slightly different selection scheme. For further discussion BGA and MGA are treated as being almost identical.

4 Necessary Properties of the Fitness Function and some Limits of an SGA

Within the scope of this work, a function $f(\vec{x})$ is *decomposable*, iff there exists n not necessarily identical functions f^i such that

$$f(\vec{x}) = \sum_{i=1}^{n} f^i(x_i) \tag{2}$$

If a function f is decomposable, then the parameters are described as *independent*.

To begin the discussion, it is useful to enumerate some *corner stones* of genetic algorithms:

1. "The theoretical results are obtained under the assumption of additive gene effects" [13].
2. It is reported that "recombination is the most important search operator of the BGA" [12].
3. The mutation probability m_p should be $m_p = 1/n$ [6, 1, 13].

Uniform **recombination** can be interpreted as shuffling corresponding parameters to generate offspring at the corners of a hypercube defined by the parameters involved.

In [6, 7, 1, 12, 19] recombination[2] is considered to be at least very important. [12, 13] reports that recombination reduces the number of function evaluations by about 50 % - 66 % when applying the BGA to Rastrigin's function and it is considered to be essential when applying the BGA to Schwefel's function. Furthermore, it is reported that for Schwefel's function with merely 20 parameters the BGA requires a large population of size $N = 500$; otherwise the BGA is not able to approach the minimum by means of recombination. Recombination tries to find a solution by testing many different combinations of existing genetic material, and in [13] it has been suggested that successful recombination needs at least a critical population size that depends on the selection intensity, the size of the problem, and the initial population. But, a closer examination of Schwefel's function and BGA's mutation operator reveals why mutation alone is insufficient. In this case, the BGA's mutation operator given by (1) provides random numbers in the interval $[-200, 200]$ and the one-dimensional distance between the minimum and the second best is approximately 723. Thus, it is obvious that mutation cannot succeed and this can easily be corrected by increasing the mutation interval to the whole domain. With this modification, mutation alone is sufficient for optimizing Schwefel's function, but recombination can still lead to a significant improvement. In this case, MGA needs about 400 generations when using a (20+20) selection scheme and about 11 generations when using a (100+100) selection scheme respectively. The above modification, however, slows down the optimization of Rastrigin's function, because the previous setting is much more suitable for that problem.

For recombination to be successful two individuals with well-adapted parameters (otherwise they would not have survived) must be combined to form a better solution. Suppose, for example, there are two individuals with two parameters, and each one has a well-adapted and a less well-adapted parameter. If both are recombined in such a way that the first individual has both 'good' parameters, it consequently gets a better fitness value. But this works only if the 'old good' parameter is still good with respect to the new parameter. This implies that each parameter has an additional or independent effect on the fitness function - in other words, they are conjugates.

Recombination on independent parameters works well. With a recombination probability $r_p = 0.5$ there exists a 50 % chance that two individuals with different sets of parameters will be combined to a better solution. But, if certain parameters interact as different functional units, then whole units must be exchanged. Suppose, there are two different units of length a and b respectively, then either all a parameters and none of the b parameters must be exchanged or vice versa. In both cases, the resulting probability for such an event is given by 0.5^{a+b}, which is fairly small even for relatively short-lenght functional units. Because recombination works on two offspring,

[2] The recombination operator in the BGA and MGA has the same behavior as crossover on bit strings, i.e., the exchange of floating-point parameters or single bits respectively.

the overall probability of successful recombination drastically decreases to 0.5^{a+b-1}; i.e., recombination performs best on independent parameters, where $a = 1$ and $b = 1$.

Mutation generates new values instead of combining existing values (genetic material). Thus mutation is often considered to be a constructive, a diversity-generating [7], or a restoring-of-lost-genetic-material [19] operator. With a mutation probability of $m_p = 1/n$, the genetic algorithm, in average, picks one parameter and changes its value. According to the resulting fitness value, this modification will survive or be dismissed. This kind of modification makes sense only if the optimum can be found by a sequence of single modifications, i.e., each parameter has to have an beneficial effect on the fitness function. Therefore, the analysis of the requirements is the same as for recombination.

The implicit assumptions about and requirements of the fitness function discussed so far match together perfectly. If the parameters are independent with respect to the fitness function, then the parameters have additive effects, a mutation rate of $m_p = 1/n$ is optimal[3], and recombination works well as long as the function's principle axes are identical with the coordinate system spawned by the parameters. For instance, if the optimization procedure encounters an n-dimensional valley, progress is only attainable in a very specific direction. That is, all parameters that are related to valley must be changed at the same time in order to get reasonable convergence. Given a mutation probability $m_p = 1/n$, a procedure changes, from a practical point of view, only one parameter at a time; occasionally, it changes two parameters simultaneously with a probability $m_p = 1/n^2$ and m parameters with a probability $m_p = 1/n^m$. If a procedure changes only one parameter x_i at a time, optimization has to take place in a *specific ordering*. For example, if the valley is along x_1 and perpendicular to x_2, it does not matter if the mutation sequence is $x_1, x_1, x_1, x_2, x_2, x_2$ or $x_1, x_2, x_1, x_2, x_1, x_2$. But, if the valley is turned by 45 degrees, a successful sequence of mutations can only be $x_1, x_2, x_1, x_2, x_1, x_2$. With a mutation probability $m_p = 1/n$, the procedure needs an average of n function evaluations to choose a specific parameter x_i. If the modification sequence consists of m different parameters and if the mutations must be in a specific ordering, a procedure requires n^m function evaluations on average. If, however, the sequence of mutations can be arbitrary, n function evaluations are sufficient to change the whole set of parameters. This means that the convergence speed is slowed down by a factor n/n^m. Recombination can still increase convergence. But, with the current schemes of two parents and uniform recombination at best two parameters can be optimized simultaneously, which slows down the overall convergence speed by a factor $1/n^{0.5m}$.

[3] With a mutation rate of $m_p = 1/n$ the procedure, in average, changes one parameter x_i at a time and the selection process determines the quality of this modification. Changing many parameters simultaneously can cause problems in the selection scheme, because statistically beneficial mutations can be compensated by detrimental mutation.

To verify this intuition, the ES, the MGA, and collective learning were applied to Rosenbrock's function f_2 (Figure 1). Rosenbrock's function has only two parameters and is unimodal. To optimize this function without using the derivatives, a gradient descent method would need three function evaluations per step [15]. Due to these facts, we decided to perform ten function evaluations per generation. Specifically, we used a (5+10) evolution strategy, a (5+10) genetic algorithm, and an (8+2) collective learning algorithm was used to guarantee that each procedure performed 10 function evaluations per generation. Figure 2 shows a comparison of the optimization progress, wherein each graph represents a typical run. The genetic algorithm performs worst and collective learning performs best. Within 800 generations, the evolution strategy reaches the point (0.9977, 0.9958), which is quite close to the minimum located at (1, 1). But, the genetic algorithm prematurely stagnates when reaching the narrow valley and ends at (0.8330, 0.6936). The difference in the final function value obtained by evolution strategy and the MGA is about 1.5 orders of magnitude. The evolution strategy and collective learning reach better solutions after 10% of the number of function evaluations required by the genetic algorithm. Further, it should be noted that the GA repeatatly stalls out despite the function possessing no local optima.

Further experiments examined the functions f_1, f_3, and f_4. If the analysis of this section is correct, then the genetic algorithm may be expected to slow down if the coordinate system is rotated. Rotation means that a linear and orthogonal transformation matrix M is applied such that $f_i(M\vec{x})$ is calculated. The transformation matrix M is a pure rotation.

Figures 3 and 4 summarizes results when applying the ES, the MGA, and CL to the functions f_1 (quadratic parabola) and f_3 (Rastrigin's function). Each graph shows the effect of rotating the coordinate system on a particular function while using the same optimization procedure. Despite each procedure having a different convergence speed, the following behavior is observable: collective learning and the evolution strategy are almost insensitive to a coordinate system's rotation, whereas the genetic algorithm is very sensitive and the results are discouraging. Thus it may be concluded that a standard genetic algorithm such as SGA, BGA, or MGA crucially relys on the independence of each parameter x_i with respect to the function to be optimized. Note that we do not show any graph of optimizing Rastrigin's function by means of evolution strategy, because the evolution strategy is not suitable for such highly multimodal functions. One may wonder about the oscillations in graph (C) of Figure 3. These oscillations are due to the normalization of the gradient descent method that collective learning uses. Without this gradient normalization collective learning would be much faster, but the procedure is then more sensitive to initial parameter settings.

As mentioned in section 3, collective learning uses either a genetic algorithm or evo-

lution strategy to generate new offspring. In both cases, the mutation probability is $m_p = 1$ and in case of a genetic algorithm, collective learning flips each bit of the mantissa with a probability of 0.1. The analysis and experiments indicates that CL^{GA} also implicitly depends on the independence of each parameter. Although collective learning modifies each parameter, but flipping each bit with a 10 % chance means that only a few parameters were changed significantly. Increasing this probability to 0.3 makes the CL^{GA} variant robust when applying it to Rastrigin's function. But it exhibits poor scaling behavior for $n \gg 20$. Therefore, Figure 3 and 4 show collective learning using the evolution strategy only.

To give a better understanding of the various effects, a geometrical explanation can be provided. Suppose that a genetic algorithm changes one parameter x_i at a time t. In this case, the procedure samples function values like $f(c_1, \ldots, x_i, \ldots, c_n)$ with $c_j (j \neq i)$ constant. For the functions described in table 1 this appears as $f(\vec{x}) = c + f^{'}(x_i)$. Suppose furthermore that the genetic algorithm chooses the same parameter at a later time $t^{'} > t$ and that meanwhile other parameters have also been changed. Then, despite a new constant $c^{'}$, the function appears unchanged with respect to x_i. For example, if x_i represented the global minimum at time t, then it still represents it at time $t^{'}$. But this is not true in the rotated coordinate system. After changing parameters other than x_i between both time steps t and $t^{'}$, the function could be totally different with respect to x_i ; i.e., the global minimum could have changed to a maximum although x_i itself has not changed. As a conclusion, it can be stated that mutation of only a few parameters in such a rotated coordinate system cannot be expected to provide efficient convergence.

5 Exploiting the Assumptions

In the previos section, the limitations of standard genetic algorithms have been indicated. Now, consider the question of how to construct an optimal algorithm that exploits the decombosability propert most efficiently. If all parameters are independent with respect to the fitness function $f(\vec{x})$ (as defined by (2)), $f(\vec{x})$ can be constructed as follows:

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

i.e., the fitness function is simply the sum of n not necessarily identical functions $f^i(x_i)$ each depending only on the *single* parameter x_i. This implies that $f(\vec{x})$ can be optimized by simply optimizing all x_i in any arbitrary sequence. Gill et al. [5]

and Press et al. [14] have already discussed a huge variety of optimization procedures for the one-dimensional case. If for all parameters x_i the number of iterations FE_i is bounded by $O(g)$, then the number of function evaluations FE is bounded by $O(n\,g)$. In this case, FE is proportional to n, which is strictly smaller than $O(n \lg n)$ suggested in [12]. Optimizing Rastrigin's function in this way would yield:

Rastrigin's function

	$n = 1$	$n = 20$	$n = 200$
MGA	62	1230	12360
CL^{GA}	41	812	8120

Schwefel's function

	$n = 1$	$n = 20$	$n = 200$
MGA	432	8640	86400
CL^{GA}	64	129	1288

The numbers of function evaluations shown in this table are smaller than those presented in Table 2, as expected.

It is very easy to construct a parallel version of this kind of optimization procedure. First, each processor p_i initializes the set of parameters and then optimizes its corresponding parameter x_i. Finally, the solution has to be constructed by merely merging the x_i optimized by processor p_i. Neglecting the final communication overhead, the runtime of this algorithm is determined by the slowest processor p_i.

Often the onemax function is used to evaluate the performance of a genetic algorithm. The onemax function consists of n bits and its value is the number of bits set to 1. The above parallel algorithm would use *exactly* two steps which equals to $2\,n + 1$ function evaluations.

6 Discussion

This work explaines why genetic algorithms like SGA, BGA, or MGA are successful in optimizing (multimodal) functions. The reason for the very good performance on such functions is not – as usually believed – that the local minima are distributed on a regular grid which is easy to solve by recombination. In contrast, it is because standard genetic algorithms implicitly exploit the decomposability property of the fitness function $f(\vec{x}) = \sum_{i=1}^{n} f^i(x_i)$. If the coordinate system is rotated in the n-dimensional space, these genetic algorithms fail. On the other hand, if the fitness function is indeed linear separable, it is very easy to construct a more efficient algorithm than a geentic algorithm, with a complexity that increases linearly with n, which is strictly smaller than the $O(n \lg n)$ suggested in [12].

The limitations of such genetic algorithms are caused by setting $m_p = 1/n$, or by

setting the bit manipulation probability to $1/n$. If $m_p = 1$ as in the evolution strategy, a genetic algorithm would fail because there is no adaptation of the amount or degree of mutation. Rechenberg has already shown in [15] that reasonable progress in the optimization process is attainable only in a small evolution window. 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 $m_p = 1$. 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.

On the other hand, the evolution strategy (applied to unimodal functions) and hybrid algorithms such as collective learning, using the evolution strategy, do not use – not even implicitly – the decomposability property of the described test functions. Therefore, the performance is (almost) constant with respect to a rotation of the coordinate system.

Collective learning implicitly exploits the macrostructure of the fitness function. By means of mutation, selection, and step size adaptation, collective learning optimizes a virtual envelope function, which is unimodal for Rastrigin's function. But, when applied to Rastrigin's function, collective learning does not find the absolute minimum, but instead stagnates in its close vicinity. This is due to the drastic reduction of the step size of the evolution strategy. Getting stuck can be avoided by altering parameter settings. But this would involve some problem-specific information and this limits the robustness of the procedure. Of course, collective learning is not able to optimize any arbitrary function $f :< bit\_string >\rightarrow r$ in an efficient way. This problem is, in the general case, NP-complete and collective learning can not solve the question of $NP \stackrel{?}{=} P$.

In addition, it is not clear if functions of practical interest are linear separable; this is beyond the scope of this work. But in many cases they definitely are not separable, for example, when using optimization techniques as a learning method for neural networks.

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 linear separable. This, the, is "crossover's niche" (cf. [2]). 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, while 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. [2] 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 that this niche may be irrelevant 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 bya genetic algorithm, and for problems from the real world, 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 and for supplying results obtained by evolutionary programming as well as for his help to polish-up this paper. In addition, the author thanks Jerome Feldman in particular for his support.

References

[1] DeJong, K.A., (1975) An Analysis of the Behavior of a Class of Genetic Adaptive Systems, PhD Thesis (University of Michigan).

[2] 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/

[3] Fogel, L.J., (1962) Autonomous Automata," *Industrial Research*, vol. 4, pp. 14-19.

[4] Fogel, D.B., (1995) *Evolutionary Computation: Toward a New Philosophy of Machine Learning Intelligence*, IEEE Press, Piscataway, NJ, in press.

[5] Gill, P.E., Murray, W., Wright, M.H., (1981) Practical optimization (Academic Press).

[6] Goldberg, D.E., (1989) Genetic Algorithms in Search, Optimization and Machine Learning (Addison-Wesley Publishing Company).

[7] Goldberg, D.E., (1994) Genetic and Evolutionary Algorithms Come of Age. Communications of the ACM 37, pp. 113-119.

[8] Hoffmeister, F., Bäck, T., (1990) Genetic Algorithms and Evolution Strategies: Similarities and Differences, in: Parallel Problem Solving, H.P. Schwefel and R. Männer (eds.) (Springer-Verlag), pp. 455-469.

[9] Luenberger, D.G., (1984) Linear and Nonlinear Programming. (Addison-Wesley Company).

[10] Mühlenbein, H., Schomisch, M., Born, J., (1991) The Parallel Genetic Algorithm as Function Optimizier, in: Proceedings of the Fourth International Conference on Genetic Algorithms, (Morgan Kaufman Publisher).

[11] Mühlenbein, H., (1992) How genetic algorithms really work I: Mutation and hill climbing, in: Proceedings of Parallel Problem Solving from Nature 2, pp. 15-26.

[12] Mühlenbein, H., Schlierkamp-Voosen, D., (1993) Predictive Models for the Breeder Genetic Algorithm, Evolutionary Computation, K.A. DeJong (ed) (MIT press), pp. 25-50.

[13] Mühlenbein, H., Schlierkamp-Voosen, D., (1994) The Science of Breeding and its Application to the Breeder Genetic Algorithm. Evolutionary Computation, K.A. DeJong (ed) (MIT press).

[14] Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T., (1987) Numerical Recipes (Cambridge University Press).

[15] Rechenberg, I., (1973) Evolutionsstrategie (Frommann-Holzboog).

[16] Salomon, R., van Hemmen, J., (1992) A new learning scheme for dynamic self-adaptation of learning-relevant parameters, in: International Conference on Artificial Neural Networks (ICANN-92), I. Aleksander and J. Taylor (eds.) (North-Holland), pp. 1047-1050.

[17] Salomon, R., (1994) Improved Global Convergence by Means of Collective Learning, in: The Third Annual Conference on Evolutionary Programming, A.V. Sebald and L.J. Fogel (eds.) (World Scientific Publishing, River Edge, NJ), pp. 34-41.

[18] Schwefel, H.P., (1977) Numerische Optimierung von Computer-Modellen mittels Evolutionsstrategie (Birkhäuser Verlag, Basel and Stuttgart).

[19] Srinivas, M., Patnaik, L., (1994) Genetic Algorithms: A Survey, Computer, pp. 17-26.

Figure 1: A contour plot of Rosenbrock's function $f_2(x_1, x_2) = 100\,(x_2 - x_1^2)^2 + (1 - x_1)^2$. The minimum is at $\vec{x}_t = (1, 1)$. One can see the curved narrow valley in the first quadrant.

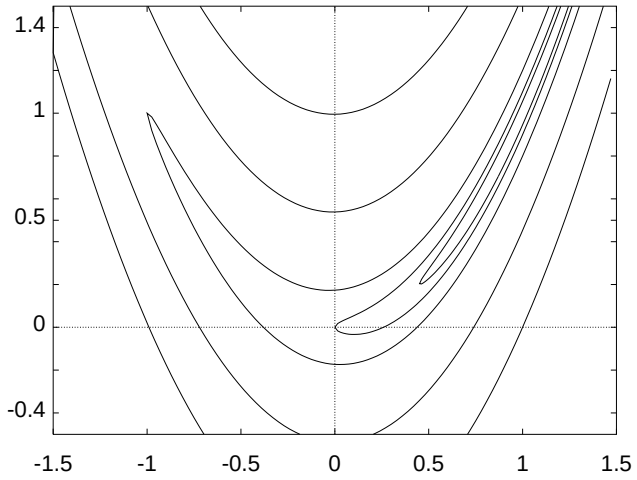

Figure 2: Comparison of the evolution strategy, our own genetic algorithm and collective learning when optimizing the two-dimensional Rosenbrock function $f_2(x_1, x_2) = 100\,(x_2 - x_1^2)^2 + (1 - x_1)^2$.

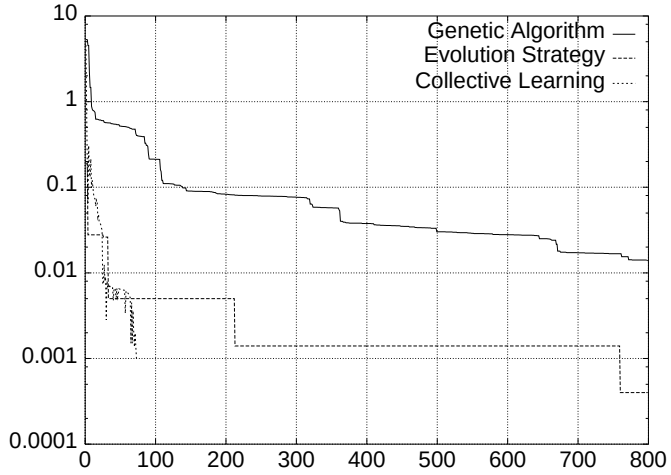

Figure 3: Comparison of the evolution strategy (A), our own genetic algorithm (B), and collective learning(C) when optimizing the quadratic function f_1.

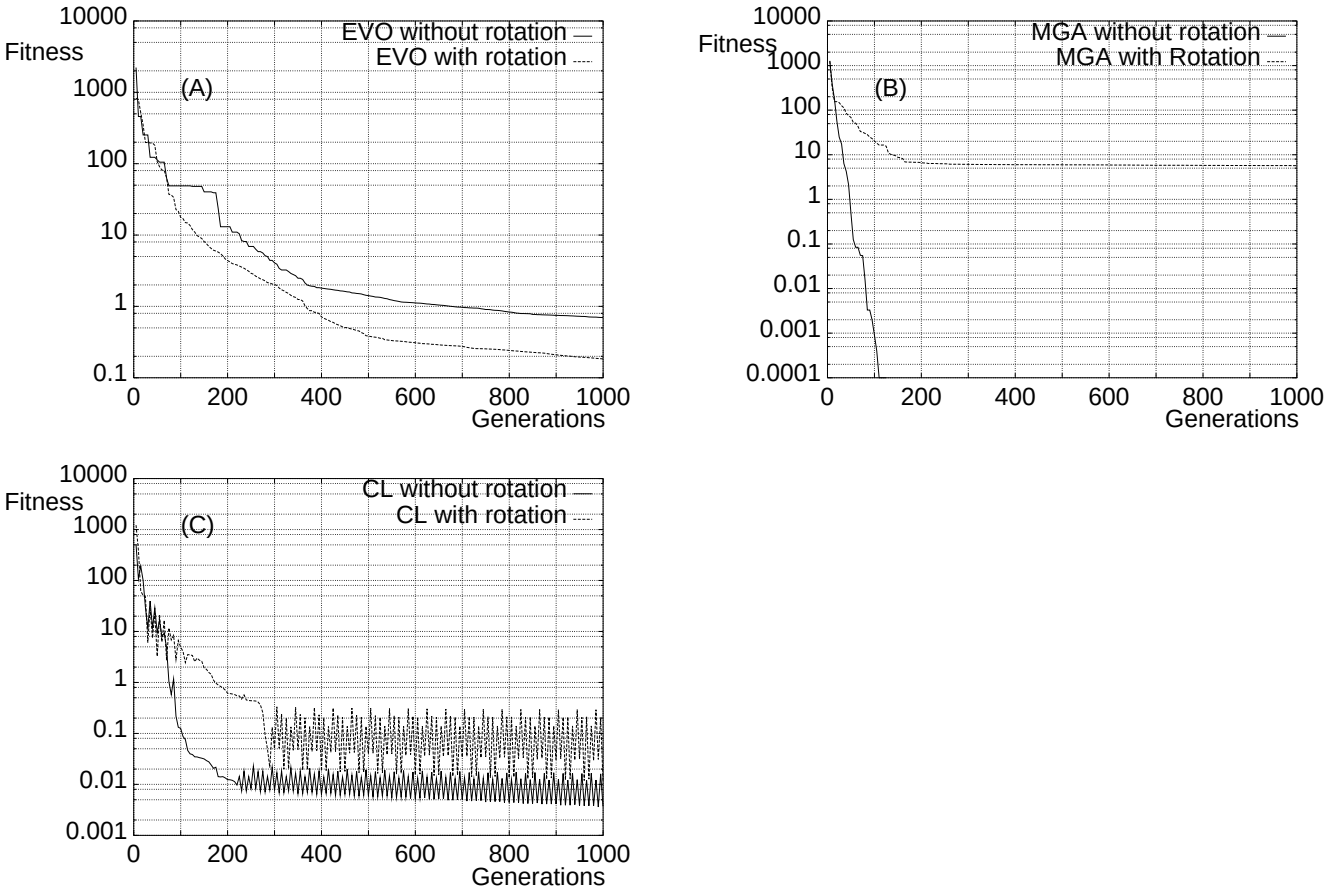

Figure 4: Comparison our own genetic algorithm (A), and collective learning(B) when optimizing Rastrigin's function f_3.