Evolutionary Algorithms and Gradient Search: Similarities and Differences

Ralf Salomon

Abstract—Classical gradient methods and evolutionary algorithms represent two very different classes of optimization techniques that seem to have very different properties. This paper discusses some aspects of some “obvious” differences and explores to what extent a hybrid method, the evolutionary-gradient-search procedure, can be used beneficially in the field of continuous parameter optimization. Simulation experiments show that on some test functions, the hybrid method yields faster convergence than pure evolution strategies, but that on other test functions, the procedure exhibits the same deficiencies as steepest-descent methods.

Index Terms—Evolutionary algorithms, evolution strategies, gradient search, optimization.

I. INTRODUCTION

Classical gradient methods and evolutionary algorithms represent two very different classes of optimization techniques. In optimization, a problem is typically specified by a set of \( n \) parameters \( \{x_1, \ldots, x_n\} \), also denoted as \( \vec{x} \), and an objective function \( f(\vec{x}) \), which is also called a fitness function in the context of evolutionary algorithms. The goal of the optimization process is to find a set of \( n \) variables \( \{x_1^*, \ldots, x_n^*\} \) such that the objective function \( f(x_1^*, \ldots, x_n^*) = f(\vec{x}^*) = \text{opt} \). Without loss of generality, it is sufficient to consider only minimization tasks, since maximizing \( f(\vec{x}) \) is equivalent to minimizing \(-f(\vec{x})\).

In the special case of continuous parameter optimization in which all parameters \( x_i \) are real valued, Newton developed the gradient method, which is also known as the method of steepest descent (steepest ascent in the case of a maximization task). The gradient \( \vec{g} \) is defined as an \( n \)-dimensional vector

\[
\vec{g} = \nabla f(\vec{x}) = \left( \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right)^T
\]

with its components being the first partial derivatives of the objective function with respect to the corresponding coordinate. The gradient has the important property that at any point \( \vec{x} \) in search space, it always points into the direction of the maximal increase of the objective function. Hence, the gradient \( \vec{g} \) is always perpendicular to the \((n-1)\)-dimensional hypersurface \( f(\vec{x}) = c \) with constant values of the objective function. In unimodal functions, the optimum can be found by moving along the local gradients, which leads to the following formulation of the steepest-descent method:

\[
\vec{x}_{t+1} = \vec{x}_t - \eta \nabla f(\vec{x}_t)
\]

with \( \eta \) denoting a small constant and \( t \) denoting the iteration number. In other words, the steepest-descent method starts at an initial point \( \vec{x}_0 \), and then repeatedly subtracts a small fraction \( \eta \) of the locally calculated gradient \( \vec{g}_t \) from the current point \( \vec{x}_t \). It is guaranteed that for sufficiently small step sizes \( \eta \), the steepest-descent procedure converges to the next (local) optimum. For an overview of various programming examples and various acceleration methods, the interested reader is referred to [1] and [2].

It is obvious that steepest-descent algorithms can be applied only to continuously differentiable objective functions \( f(\vec{x}) \). But if either the objective function is not continuously differentiable or if the function is not (completely) given due to limited knowledge, which often occurs in real-world applications, the designer has to resort to other methods, such as evolutionary algorithms. Evolutionary algorithms are a class of stochastic optimization and adaptation techniques that are inspired by natural evolution. They provide a framework that mainly consists of genetic algorithms [3]–[5], evolutionary programming [6], [7], and evolution strategies [8]–[10]. A comparison of these different methods can be found in [11].

Each evolutionary algorithm is designed along a different methodology. Despite their differences, all evolutionary algorithms are heuristic population-based search procedures that incorporate random variation and selection. In each iteration \( t \), also called a generation, an evolutionary algorithm typically generates \( \lambda \) offspring from \( \mu \) parents. Every offspring is generated by copying a parent and adding a mutation vector \( \vec{z} \) to the parent’s parameters \( \vec{x} \). In evolutionary programming and evolution strategies, all components, i.e., \( p_m = 1 \), of \( \vec{z} \) are typically Gaussian-distributed with mean zero and standard deviation \( \sigma \), also called the step size in accordance with steepest-descent methods. In most genetic algorithms, by contrast, the mutation probability is only \( p_m = 1/n \) leading to an average of one mutation per offspring. In addition to mutation, many evolutionary algorithms also exchange some of the offspring’s parameters \( x_i \) across several offspring. By evaluating the objective function, evolutionary algorithms assign a fitness value to each offspring and select \( \mu \) offspring as parents for the next generation (iteration). In addition to fitness evaluation, all offspring undergo a random tournament competition in evolutionary programming [7]. For further details, especially
different mutation and recombination operators as well as different selection schemes, the interested reader is referred to [4], [10], and [12].

A very important feature of evolutionary programming and evolution strategies is their ability to self-adapt the step size $\sigma_t = \sigma(t)$. It is very common for both evolutionary programming and evolution strategies to perform self-adaptation by considering $\sigma_t$ as an additional parameter. Prior to applying mutations to the parameters $x_i$, the step size $\sigma_t$ is modified by lognormal distributed mutations. In the long term, those population members survive that have the best adapted step size $\sigma_t$. In a more elaborated form, evolutionary programming and evolution strategies use $n$ different step sizes $\sigma^i_t$, one for each parameter $x_i$. In its full form, an $n \times n$ matrix $\mathbf{C}$ is used to generate correlated mutations $\mathbf{C}^c$.

A remark should be given on the computational costs of correlated mutations. Their generation, i.e., the matrix times vector multiplication, requires $O(n^2)$ additional operations per offspring, i.e., per function evaluation. The cost of adapting the correlation coefficients $c_{ij}$ depends on the particular algorithm. If deriving these elements by global recombination operators, as is discussed for example in [13], the costs are also in the order of $O(n^2)$. If, however, the algorithm performs a complete eigenvalue analysis the costs for adapting the coefficients $c_{ij}$ increase to $O(n^3)$ operations per generation, unless $\mathbf{C}$ is updated only every $n$ generations. For a more in-depth discussion of this latter issue, the interested reader is referred to [14] and [15]. It should also be mentioned that the use of covariances has been adopted by particular genetic algorithms [15]. For implementation details of correlated mutations, see [10], [11], [13], [15], and [16].

At a first glance, it seems that evolutionary algorithms, as multipoint search strategies, have other properties than steepest-descent algorithms as single-point search strategies. It is widely believed that the specific potential of evolutionary algorithms originates from their parallel search by means of entire populations. In particular, the ability of escaping from local optima, an ability very unlikely to be observed in steepest-descent methods, is often assigned to the usage of populations. Section II compares steepest-descent and evolutionary algorithms in more detail and aims at better understanding their similarities and differences. Section III then uses this comparison to fuse both types of algorithms into the evolutionary-gradient-search procedure. This procedure uses the evolutionary framework to estimate the gradient and uses the idea of steepest descent to perform the actual optimization step including the self-adaptation of the step size $\sigma_t$. Therefore, this procedure is neither an evolutionary algorithm nor a steepest-descent method in the narrow sense. Section IV then describes the experimental setup, and Section V explores to what extent the hybrid method can be beneficially used in the field of continuous parameter optimization. It turns out that on some functions, the procedure significantly accelerates the optimization process, but that on other functions, evolution strategies with correlated mutations, which requires $O(n^2)$ operations, yield superior results. Besides some convergence issues, the procedure under investigation enables a convenient option for incorporating acceleration methods from classical optimization techniques. Section VI finally concludes with a brief discussion.

II. EVOLUTIONARY ALGORITHMS VERSUS GRADIENT SEARCH

Evolutionary algorithms and steepest-descent methods seem to rely on different mechanisms and thus may have different properties. This section argues that the similarities and differences depend on the point of view and that both types of algorithms are more akin than often believed. In particular, this section compares both types of algorithms with respect to their multipoint versus single-point search strategies, their behavior of moving along the gradient path or not, and their chance of getting stuck in local minima.

Some have argued that evolution strategies, for example, are path-following methods [9], whereas others, e.g., [17], argue that evolution strategies cannot be appropriately described as path-following methods. In this section, it is argued that both descriptions have their own merits and that the answer depends on the point of view. To discuss this question, a mapping from the $n$-dimensional search space onto two dimensions has to be used. For this purpose, Rechenberg [9, p. 75] has used a transformation

$$y_1 = \sqrt{\sum_{i=1}^{n/2} x_i^2}, \quad y_2 = \sqrt{\sum_{i=n/2+1}^{n} x_i^2}, \quad for \ n \ even.$$ (3)

As can be seen in the Fig. 1(a), this transformation yields a path-following impression on quadratic objective functions, since the evolution strategy is “creeping” along the gradient path. The situation drastically changes, however, if looking at two arbitrarily chosen parameters $x_i$ and $x_j$ as has already been done in [17]. Fig. 1(b) clearly shows that even though the two-dimensional mapping $(y_1/y_2)$ is following the gradient path, two arbitrarily chosen parameters $x_i$ and $x_j$ might exhibit a rather erratic behavior, which contrasts, on a microscopic level, the gradient-following impression. In experiments, it can be observed that for small problem dimensions $n$ and a very large number $\lambda$ of offspring, for example, an $(1, \lambda)$ evolution strategy (i.e., one parent generating $\lambda$ offspring) is following the gradient path. Such an evolution strategy, however, temporarily departs from the gradient path if the length of the mutation vectors $x$ is much larger than the current distance to the optimum, which frequently occurs due to the self-adaptation of the step size. In this case, most offspring are more or less of equal fitness and thus selection is rather random. In most practical cases, the individual parameters $x_i$ deviate considerably from the gradient path leading to the observation already shown in Fig. 1(b). A detailed analysis of this deviation has already been presented in [17].

The next question to be discussed concerns the maintenance of entire populations in evolutionary algorithms. Many have offered that the specific power of evolutionary algorithms originates from collective interactions within the entire population [4], [11], [16], [17]. It has been argued [4, p. 9] that due to its multipoint search strategy, evolutionary algorithms have a high probability to escape from local optima, and finding the desired global optimum, and large amount of empirical evidence, e.g.,
Fig. 1. (a) shows the “path” of the evolution strategy when minimizing a quadratic function and when using the transformation (3) that maps the $n$-dimensional search space onto two dimensions. (b) shows that even though the two-dimensional mapping is “creeping” along the gradient path, two arbitrarily chosen parameters $x_i$ and $x_j$ might exhibit a rather erratic behavior. [11], [13], [18], seems to support this hypothesis. Due to these beliefs, it is very clear why most evolutionary algorithms use parameter settings $\mu > 1$ and $\lambda > 1$ (i.e., $\mu$ parents, $\lambda$ offspring). In this respect, it is interesting to discuss the theoretical analysis presented in [17] and [19], which analyzes $(\mu/\mu_I, \lambda)$ evolution strategies (i.e., a global intermediate recombination operator $\mu_I$ is applied to all $\mu$ parents prior to generating $\lambda$ offspring). As is shown in Fig. 2, a $(\mu/\mu_I, \lambda)$ selection scheme features a global intermediate recombination scheme, which calculates a virtual parent $\hat{v} = 1/\mu \sum_{j=1}^{\mu} \hat{v}_j$ as the center of mass of all $\mu$ parents at each generation $t$. 

In other words, global intermediate recombination reduces the entire parental population to only one virtual representative, which seems, at first glance, to contradict the notion of a multipoint search strategy. It has been argued [19], however, that the global intermediate operator extract the similarities of all parents for which a population of parents is required for obvious reasons. This view coincides with the results discussed in [13], which clearly show that using $\mu > 1$ parents is advantageous with respect to the attainable convergence speed. Section III investigates a “recombination” scheme that not only uses the selected $\mu$ best parents but all $\lambda$ offspring.

Conversely, it seems at a first glance when looking at (1), that steepest-descent methods are merely single-point search strategies. Any steepest descent, however, requires information from at least $\eta$ different directions to be able to calculate the gradient. If the objective function is not given, the required information can be obtained, for example, by $\eta$ independent test trials $\partial f/\partial x_i \approx (f(\ldots, x_i + \Delta x, \ldots) - f(\ldots, x_i, \ldots))/\Delta x$ along each $x_i$-axis. In this case, a steepest-descent algorithm cannot appropriately be considered a single-point search strategy. Also, directly calculating the gradient by analytical means can be considered as a population of size $\eta$ of orthonormally distributed individuals with infinite small distances.

The discussion about approximating the gradient by $\eta$ independent test steps immediately leads to the discussion about an algorithm’s ability of escaping from local optima. When directly calculating the gradient of a given objective function, the algorithm uses an infinitely small step size $\Delta x \to 0$, as opposed to evolutionary algorithms where parents and offspring have a considerable distance in the order of $\sigma$ or $\sigma \sqrt{\eta}$. If, however, the gradient is estimated by independent trials with a distance $\Delta x$ along each axis, the difference between both classes of algorithms almost vanishes.

In summary, whether or not an evolutionary algorithm can be considered a gradient-following procedure largely depends on the point of view, and Section III explores how both approaches can be beneficially fused.

### III. THE EVOLUTIONARY-GRADIENT-SEARCH PROCEDURE

This section describes the evolutionary-gradient-search (EGS) procedure. The description consists of three parts. The first part focuses on the basic procedure, the second part discusses the procedure’s basic behaviors, and the third part describes an acceleration method that is also known as the momentum term (e.g., [20, p. 330]).

#### A. The Basic Procedure

In each iteration (generation) $t$, the EGS procedure first estimates the direction of the gradient in an evolutionary-
At a given point \( x_t \) at iteration \( t \), the EGS procedure starts by generating \( \lambda \) test candidates (offspring) \( x'_i \) (\( \lambda = 6 \) in this example) by applying Gaussian-distributed mutations \( z_i \). All \( \lambda \) test candidates are then used to calculate a unit vector \( \tilde{c}_t \) that points into the direction of the estimated global gradient, which might significantly deviate from the true gradient. The procedure then performs two test steps with different step sizes \( \sigma_t \) and \( \sigma_t / \zeta \) along the unit vector \( \tilde{c}_t \) with \( \zeta \approx 1.8 \) denoting a variation factor. The procedure finally selects the best test step and updates the step size accordingly.

In each iteration \( t \), the EGS procedure starts at a given point \( x_t \). At \( x_t \), it generates \( \lambda \) test candidates (offspring) \( x'_1, x'_2, \ldots, x'_\lambda \) by applying random mutations \( z_i \) to the current point \( x_t \). All components of all mutation vectors \( z_i \) are Gaussian distributed with mean zero and standard deviation \( \sigma_t / \sqrt{n} \) leading to \( \|z_i\| \approx \sigma_t \) for \( n \gg 1 \). For \( n \gg 1 \), the test points \( x'_1, x'_2, \ldots, x'_\lambda \) are thus distributed on a hypersphere with radius \( \sigma_t \). By using the information given by all test candidates, the procedure then calculates a unit vector \( \tilde{c}_t \) that points into the direction of the estimated (global) gradient \( \tilde{c}_t \), which is simply a weighted sum and can, however, significantly deviate from the true gradient.

\[
\tilde{c}_t = \frac{\tilde{g}_t}{\|\tilde{g}_t\|}
\]

with

\[
\tilde{g}_t = \sum_{i=1}^{\lambda} (f(x'_i) - f(x_t)) (x'_i - x_t).
\]

The idea behind this estimate is as follows. As can be seen from Fig. 3, some of the test candidates (e.g., \( x'_1, x'_2, x'_3 \) in Fig. 3) yield a better fitness \( f(x'_i) = x'_i + z'_i \) than the parent \( x_t \). The corresponding mutations \( z'_i \) are accumulated in proportion to their relative fitness contribution \( f(x'_i) - f(x_t) \) with respect to the parent’s fitness \( f(x_t) \). The EGS procedure also uses those test candidates (e.g., \( x'_4, x'_5, x'_6 \) in Fig. 3) that yield a worse fitness than the parent \( x_t \). For these test candidates, it can be assumed that the gradient might be pointing in the opposite direction, and thus these mutations are also included in the estimate (4), since the fitness contributions \( f(x'_i) - f(x_t) \) may have a negative sign. By these means, the estimate (4) yields reasonable results even if only one test candidate/offspring is generated in each iteration as is shown in the following subsection. Appendix A presents some theoretical considerations on why the estimate (4) approximates the gradient, and Appendix B discusses some implementation details.

After calculating the unit vector \( \tilde{c}_t \), the EGS procedure has to perform the actual iteration step and adapt the step size \( \sigma_t \). To this end, the procedure must perform at least two different test steps along the estimated gradient direction. These test steps should be performed with different step sizes, which might obey a lognormal distribution with mean one. In its simplest form, the procedure performs two test steps with two different step sizes, \( \sigma_t \) and \( \sigma_t / \zeta \), with \( \zeta \approx 1.8 \) denoting a variation factor. The procedure finally performs the step with the best overall fitness, thus self-adapting the step size \( \sigma_{t+1} \) on the fly. The actual iteration step can be formalized as follows:

\[
\sigma_{t+1} = \begin{cases} 
\sigma_t \zeta, & \text{if } f(x'_t - \sigma_t \zeta \tilde{c}_t) \leq f(x'_t - (\sigma_t / \zeta) \tilde{c}_t) \\
\sigma_t / \zeta, & \text{otherwise}
\end{cases},
\]

\[
x_{t+1} = x_t - \sigma_{t+1} \tilde{c}_t.
\]

The procedure defined by (4) and (5) has the following properties. First, the procedure is rotationally invariant, since all test candidates are generated without any orientation being preferred. Second, it has been shown [21] that when analytically calculating the gradient of a quadratic function by (1), the procedure defined by (5) yields linear convergence and that the value of \( \zeta \) is not critical; a value \( \zeta \approx 1.8 \) has been shown to yield the fastest convergence speed. Third, due to \( \zeta \approx 1.8 \), the procedure requires approximately four iterations to change the step size by one order of magnitude. Similarly, one order of magnitude per four iterations is the maximal speed with which the optimum of a unimodal function can be approximated.

After presenting the basic procedure, it might be worthwhile to compare the EGS procedure with a canonical evolutionary algorithm. Like evolutionary algorithms, the procedure generates a set of \( \lambda \) offspring by applying \( n \)-dimensional, Gaussian-distributed random mutations \( z'_i \) to the current parent \( x_t \). After generating and evaluating the \( \lambda \) offspring \( x'_i \), an evolutionary algorithm would typically select only the fittest offspring (e.g., \( x'_{1}, x'_{2}, \ldots, x'_{6} \) in Fig. 3) as parents for the next generation. Thus, an evolutionary algorithm normally discards all information contained in the worst (not selected) offspring. The EGS procedure, on the contrary, uses all \( \lambda \) test candidates for calculating an estimate \( \tilde{g}_t \) of the (global) gradient. In addition, due to the subsequent test steps, the EGS procedure is not a pure evolution algorithm, but rather a hybrid method.

B. Basic Behavior

Before proceeding, this section studies a simple example to get an idea of how the EGS procedure behaves, especially with respect to the number \( \lambda \) of offspring. As a simple test case, the EGS procedure is applied to the sphere model.
The convergence speed on the sphere model over 100 steps as a function of the number of generations for different numbers of test candidates. It can be observed that up to a saturation point at $\lambda \approx 200$, an increase of $\lambda$ leads to an increase of the convergence speed.

Fig. 4 shows the procedure’s optimization behavior as a function of the number $\lambda$ of test candidates. This figure clearly shows that the number $\lambda$ of offspring significantly influences the procedure’s convergence speed as can be expected from the gradient estimate (4). An increasing number $\lambda$ of test candidates per iteration increases the accuracy with which the actual gradient is estimated by (4). It is interesting to note, however, that a saturation at $\lambda \approx 200$ can be observed. From that point on, a larger number of test candidates does not lead to a more accurate estimation of the gradient. The influence of the number $\lambda$ of test candidates can be summarized by saying that up to a saturation point, which potentially depends on the fitness function and the number of dimensions, the accuracy of the gradient estimation can be increased by increasing the number $\lambda$ of test candidates.

Fig. 4 also shows the procedure’s adaptation behavior of the step size $\sigma_t$. Due to the initialization of $\vec{x}_0 = (1000, \ldots, 1000)^T$, optimization starts at an initial function value of $f(\vec{x}_0) = 10^7$. During the first approximately 16 iteration steps, no significant function value reduction can be observed, since the step size $\sigma_t$ has a value that is much too small. At iteration 16, however, the step size has reached the value $\sigma_{16} = 1.8 \times 1.8 \approx 1214$, which is in the order of the remaining distance to the optimum. In the subsequent generations, linear convergence can be observed, which goes along with a continuous decrease of the step size. It should be noted that due to its step-size update in (5), the procedure requires merely four iterations per order of magnitude of the step-size update.

The behavior of the EGS procedure is further illustrated in Fig. 5. This figure shows the evolution of two selected variables $x_0$ and $x_1$ as a function of the number of test candidates. It can be seen that for $\lambda = 1$ the evolutionary path is rather erratic, whereas the path for $\lambda = 100$ is almost along the direction of the gradient.

C. Acceleration: The Momentum Term

From classical optimization techniques, it is known that in certain situations, such as optimizing quadratic functions $f(\vec{x}) = 0.5 \omega_1 x_1^2 + \cdots + 0.5 \omega_n x_n^2$ with very different eigenvalues $\omega_i \ll \omega_j$, steepest-descent methods exhibit useless oscillations of the gradient: rather than going along a very narrow valley, the optimization path might be oscillating between both sides resulting in a very small effective progress along the valley’s direction. It is conceived that this problem can be alleviated by using a momentum term (e.g., [20, p. 330] or [21]) that provides a kind of memory by incorporating previous steps. The momentum term can be expressed as

$$\Delta \vec{x}_{t+1} = -\sigma_{t+1} \vec{e}_t + \alpha \Delta \vec{x}_t$$

$$\vec{x}_{t+1} = \vec{x}_t + \Delta \vec{x}_{t+1}$$

with $0 \leq \alpha \leq 1$ and $\Delta \vec{x}_0 = \vec{0}$. The parameter $\alpha$ is often simply called momentum. It should be noted that with $\alpha = 0$, the update rule in (6) equals the one in (5).

It is most straightforward to self-adapt the momentum $\alpha_t$ with the same adaptation mechanism already described for the step size $\sigma_t$. In this case, the procedure has to adapt two parameters, which can be easily done by testing all four combinations: $(\sigma_t, \alpha_t)$, $(\sigma_t, \alpha_t/\zeta)$, $(\sigma_t, \alpha_t/\zeta)$, and $(\sigma_t/\zeta, \alpha_t/\zeta)$. In a more general form, the procedure has to test lognormal distributed combinations of $\sigma_t$ and $\alpha_t$. In its simplest form, the procedure can adapt $\sigma_t$ and $\alpha_t$ alternately.

IV. METHODS

This section summarizes the experimental setup relevant for the benchmark tests presented in this paper. Comparisons with
TABLE I

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_{\text{sphere}}(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 = |\mathbf{x}| )</td>
<td>Sphere</td>
</tr>
<tr>
<td>( f_{\text{rosen}}(\mathbf{x}) = \sum_{i=1}^{n}</td>
<td>100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2</td>
</tr>
<tr>
<td>( f_{\text{step}}(\mathbf{x}) = \sum_{i=1}^{n}</td>
<td>x_i</td>
</tr>
<tr>
<td>( f_{\text{ellip}}(\mathbf{x}) = \sum_{i=1}^{n} i x_i^2 )</td>
<td>Ellipsoid</td>
</tr>
<tr>
<td>( f_{\text{ridge}}(\mathbf{x}) = \left( \sum_{i=1}^{n} x_i \right)^2 )</td>
<td>Schwefel’s ridge</td>
</tr>
<tr>
<td>( f_{\text{cigar}}(\mathbf{x}) = x_1 + A \sum_{i=1}^{n} x_i^2 )</td>
<td>Cigar</td>
</tr>
<tr>
<td>( f_{\text{rastrigin}}(\mathbf{x}) = \sum_{i=1}^{n} [x_i^2 - 10 \cos(2\pi x_i) + 10] )</td>
<td>Rastrigin’s function</td>
</tr>
<tr>
<td>( f_{\text{ackley}}(\mathbf{x}) = -20 \exp(-0.2\sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2}) - \exp\left(\frac{1}{n} \sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e )</td>
<td>Ackley’s function</td>
</tr>
</tbody>
</table>

other methods are done by presenting results published in the literature. Table I compiles a list of some widely used test functions, which can be seen as part of a standard test set. The sphere model \( f_{\text{sphere}} \) is the easiest and probably most widely-used test function. The problem of Rosenbrock’s [22] function \( f_{\text{rosen}} \) is that its optimum \( x_2^* = 1 \) is located in a very narrow valley, which is difficult to follow. The step function \( f_{\text{step}} \) maps real values onto integer numbers and comprises only plateaus, from which steepest-descent algorithms can normally not escape; except at some singularities, the gradient always vanishes. The ellipsoid \( f_{\text{ellip}} \) is a quadratic function with different eigenvalues along each axis, and Schwefel’s ridge \( f_{\text{ridge}} \) resembles an ellipses that is inclined in search space. The cigar is also an ellipse in which, however, all but one eigenvalue \( A \) is much larger than the eigenvalue of \( x_1 \). In the experiments, the constant \( A \) is varied between 10 and 100000. Rastrigin’s function \( f_{\text{rastrigin}} \) and Ackley’s function \( f_{\text{ackley}} \) are multimodal functions, which are hard for classical as well as most evolutionary algorithms. Steepest-descent algorithms, for example, most likely get stuck in local optima, which are frequently distributed in search space. Both functions, however, have some “smooth” global structure. It has been shown [23] that currently available genetic algorithms can find the optimum of such multimodal functions \( f(\mathbf{x}) = \sum_{i=1}^{n} f_i(x_i) \), since they can be decomposed into a sum of one-dimensional functions \( f_i(x_i) \). Currently available genetic algorithms, e.g., the breeder genetic algorithm [18], exploit this decomposability property by using a small mutation rate \( p_m = 1/n \). For a fair comparison, it has been proposed [23] to apply a rotation of the coordinate system to the test functions in order to avoid the exploitation of the certainly unwanted artificial separability property unless the method of interest behaves rotationally invariant. A rotation \( f(M\mathbf{x}) \), with \( M \) denoting a pure rotation matrix, does not change the function’s structure but its orientation in the search space. The very same approach has been proposed in [14].

Unless otherwise stated, empirical tests have been done with the following parameter settings: \( \sigma_0 = 0.1, \zeta = 1.8, \mathbf{x}_0 = (10, \ldots, 10)^T, n = 30 \) dimensions, and normally \( t = 2000 \) steps averaged over 50 independent trials. The initial step size \( \sigma_0 \) was deliberately ill-set again to indicate the procedure’s self-adaptation capability. When using the momentum term according to (6), the momentum was initialized with \( \omega_0 = 0.1 \); otherwise \( \omega_0 = 0.0 \) was used.

For comparison purposes, two different performance measures are used in this paper. The graphs show either the logarithmic performance \( \log(f(\mathbf{x})) \) or the normalized logarithmic performance \( \log(f(\mathbf{x}_0)/f(\mathbf{x})) \), which is denoted as \( \log(f^*) \) in the figures. The actual choice depends on the performance figures used in the papers with which the comparison is made. The second performance measure resembles the first one, but is independent of the initial conditions due to the normalization. This paper compares the performance of the EGS procedure with the performance of various evolution strategies as well as genetic algorithms. For the generation of the offspring, the evolution strategies under consideration use either one global step size \( \mathbf{x}_i \sim \mathbf{x}_i + \sigma \mathbf{z} \), individual step sizes \( \mathbf{x}_i \sim \mathbf{x}_i + \sigma_i \mathbf{z} \), with \( \sigma \) denoting a component-wise multiplication operator \( \mathbf{z} \) that is defined as \( \sigma_1 = \sigma_2 = \mathcal{C} \), or correlated mutations \( \mathbf{x}_i \sim \mathbf{x}_i + \mathbf{C} \). For further implementation details, especially the details concerning the self-adaptation of \( \sigma_0 \) and \( \zeta \), see [10], [11], [13], [14], and [16]. It should be noted though that generating correlated mutations requires \( O(n^2) \) additional operations.

V. RESULTS

Fig. 6 illustrates the scaling behavior of the EGS procedure when applied to the sphere model \( f_{\text{sphere}} \) with different dimensions \( n \). The figure shows the average number of generations required to reach a precision of \( \|x_i - x_i^*\| \leq \epsilon = 10^{-5} \) for all parameters \( x_i \) when using a constant number \( \lambda = 30 \) of test candidates. From the figure, it can be clearly seen that the EGS
Fig. 7. The normalized logarithmic progress over 2000 steps as a function of the number of generations for different numbers $\lambda$ of test candidates when minimizing the ellipsoid $f_{\text{ell}}$ by means of the EGS procedure without momentum.

Fig. 8. The normalized logarithmic progress over 2000 steps as a function of the number of generations for different numbers $\lambda$ of test candidates when minimizing the ellipsoid $f_{\text{ell}}$ by means of the EGS procedure with momentum. The label ES refers to a $(15,100)$-evolution strategy with individual step sizes $\sigma_t$.

procedure exhibits an almost linear scaling behavior. It should be noted that using the momentum term does not significantly accelerate convergence when minimizing the sphere model.

Figs. 7 and 8 show the normalized logarithmic performance of the EGS procedure without and with the momentum when minimizing the ellipsoid $f_{\text{ell}}$. A comparison of both figures indicates that the momentum drastically accelerates the procedure. With momentum, the EGS procedure with $\lambda = 300$ test candidates yields 144 orders of magnitude within 2000 iterations as compared to only 86 orders of magnitude without using the momentum. This acceleration can be anticipated, since the momentum term alleviates the oscillations that might occur due to the different eigenvalues in $f_{\text{ell}}$. In comparison, as has been published in [13], a $(15,100)$-evolution strategy with recombination and individual step sizes yields only 80 orders of magnitude. As has been discussed in [13], other evolution strategy variants with fewer offspring or fewer parents yield much worse performance.

For some, it may be of interest to consider the performance as a function of the number of function evaluations rather than the number of generations. This can be done by simply rescaling the horizontal axis (generational axis) by the number $\lambda$ of offspring, which has been exemplified in Fig. 9. It can be seen that in terms of the number of function evaluations, the efficacy degrades with increasing $\lambda$. In practical applications the net efficiency depends on whether the offspring are evaluated in parallel (number of generations) or sequentially (number of function evaluation). The same effect has also been reported for evolution strategies [8].

Figs. 10 and 11 show the normalized logarithmic performance of the EGS procedure without and with using the momentum when minimizing Schwefel’s ridge $f_{\text{ridge}}$. It can be clearly seen that the momentum term drastically accelerates the convergence speed. With $\lambda = 100$ offspring, the EGS procedure yields 57 orders of magnitude as compared to 39 orders of magnitude obtained with a $(15,100)$-evolution strategy with correlated mutations. Other evolution strategy variants, also taken from [13], yield considerably less progress,
Fig. 11. The normalized logarithmic progress over 2000 steps as a function of the number of generations for different numbers $\lambda$ of test candidates by means of the EGS procedure with momentum when minimizing Schwefel’s ridge $f_{ridge}$. The label ES1 refers to a (15,100)-evolution strategy with correlated mutations, ES2 to a (15,100)-evolution strategy with individual step sizes $\sigma_t$, and ES3 to a (1,100)-evolution strategy, respectively.

Fig. 12. Performance of the EGS procedure without momentum as a function of the number of generations for different numbers $\lambda$ of test candidates when applied to the step function $f_{step}$. After reaching the minimum, the procedure oscillates in the neighborhood. In all 50 runs, the EGS procedure found the optimum.

which is in the order of 14 and 26 orders of magnitude within 2000 steps.

Fig. 12 shows the performance of the EGS procedure without using the momentum when minimizing the step function $f_{step}$. Again, it can be seen that increasing the number $\lambda$ of test candidates per iteration accelerates the convergence speed. The observable oscillations are due to temporarily increases of the step size $\sigma_t$ (see also Appendix B). In all four test cases $\lambda \in \{3, 10, 30, 100\}$, the procedure found the optimum in all 50 runs. On average, the procedure required 3107 ($\lambda = 3$), 317 ($\lambda = 10$), 85.2 ($\lambda = 30$), 24.3 ($\lambda = 100$), and 15.6 ($\lambda = 300$) generations for finding the optimum with $f_{step}$. It should be noted that in all runs, the initial step size was set to $\sigma_0 = 3$ as has also been done for evolution strategies [11]. As is reported in [11], a (30,200)-evolution strategy with individual step sizes requires approximately 4000 generations, whereas a simple (30,200)-evolution strategy with one global step size fails on this task. Further comparisons with evolutionary programming and genetic algorithms have shown [11] that evolutionary programming required approximately 7500 generations (but see also [24]), whereas a canonical genetic algorithm with bit-coding representations failed on this task.

The current form of the EGS procedure has certainly its limits. When applied to functions with highly differing eigenvalues, e.g., the cigar $f_{cigar}$ and Rosenbrock’s function $f_{rosen}$, the procedure exhibits a considerable performance loss, as is well known from classical steepest-descent methods.

Fig. 13 shows the normalized logarithmic performance of the EGS procedure with momentum and test candidates when applied to the cigar with varying values for the parameter $A \in \{10, 100, 1000, 10000\}$. It can be clearly seen that the performance degrades with increasing values of $A$ and that obviously the momentum does not sufficiently alleviate the problem of useless oscillations. Once the parameters $x_2 \cdots x_n$ have approached small values, the gradient direction is almost perpendicular to the $x_1$-direction, and thus, progress with respect to this direction is very small. Similarly, Figs. 14 and 15 indicate that different

Fig. 13. Performance of the EGS procedure with $\lambda = 100$ test candidates and momentum as a function of the number of generations for different parameters $A$ of the cigar function $f_{cigar}$ Please note that the saturation at $\log(f^*)$ for $A = 10$ is due to numerical imprecision of the machine.

Fig. 14. The normalized logarithmic progress over 30 000 steps as a function of the number of generations for different numbers $\lambda$ of test candidates by means of the EGS procedure without momentum when minimizing Rosenbrock’s function $f_{rosen}$. The graphs are averaged over only 20 runs due to the large number of generations.
sensitivities of each parameter \( x_i \) degrades the performance considerably. After 5000 generations, for example, the EGS procedure with \( \lambda = 100 \) and \( n = 30 \) evolves \( \bar{x}_{5000} = (0.99995, 0.99995, 1.00006, 1.00008, 1.00005, 1.00019, \ldots, 0.9411, 0.8850, 0.7826^2) \). Therefore, the first parameters are closer than one-tenth of a thousand to the optimal value, whereas the last parameters have a distance of up to 0.22, which refers to a sensitivity difference of five orders of magnitude.

Similar performance degradations can also be observed when using simple evolution strategies with merely one global step size and genetic algorithms [23] when applied to rotated versions of such quadratic functions. This problem can be sufficiently solved only by using correlated mutations [13] and [14]. By using correlated mutations, the lines of constant function values are more or less transformed into circles. Therefore, it seems very desirable to enhance the EGS procedure with a similar self-adapting mechanism. But as has been discussed earlier, the generation of correlated mutations requires \( O(n^2) \) additional operations, which might degrade the effective net gain in particular cases. In addition, it has been shown [11] that using correlated mutations degrades the efficacy when minimizing simple functions, such as the sphere, and that on such functions, a simple \((\mu, \lambda)\)-evolution strategy converges significantly faster. Also, the efficacy of the adaptation approach presented in [14] degrades with an increasing number of offspring, and thus degrades the possibility for beneficial parallelization.

The EGS procedure has also been applied to multimodal functions, such as Rastrigin’s function \( f_{\text{rast}} \) and Ackley’s function \( f_{\text{ackley}} \). On these functions, the EGS procedure did not yield conclusive results. The success significantly depends on the initial conditions, such as \( \bar{x}_0 \), \( \sigma_0 \), and the number of \( \lambda \) of test candidates. In its current form, the EGS procedure cannot be considered a procedure that reliably finds the global optimum or a reasonable neighborhood of the global optimum. Similarly, current genetic algorithms are not able to sufficiently approach the global optimum of such multimodal functions that have been rotated [23]. For example, the breeder genetic algorithm [18], a genetic algorithm variant that is especially tailored to continuous parameter optimization, stagnates at a function value of about 80 [23] when minimizing a 20-dimensional, rotated version of Rastrigin’s function with all parameters \( x_i^0 \) being randomly initial \( |x_i^0| \leq 2.048 \), i.e., the absolute value refers to \( x \). By contrast, fairly good results have been reported [11] for evolutionary programming and a \((30,200)\)-evolution strategy when minimizing a 30-dimensional Ackley function.

### VI. Conclusions

This paper has provided a discussion on the similarities and differences of gradient methods and evolutionary algorithms. It has been argued that whether or not evolutionary algorithms can be considered gradient-path-following procedures depends on the point of view. The conclusions of this discussion have been used to explore a hybrid method, the EGS procedure, that uses the evolutionary scheme to estimate a gradient direction and performs the parameter updates in a steepest-descent form including a self-adaptation of the step size \( \sigma_t \).

The EGS procedure has been tested on a variety of test functions. It has been shown that on most test functions, the EGS procedure performs better than three different evolution strategies with different degrees of complexity. The EGS procedure, however, also exhibits deficiencies that are well known from classical steepest descent methods. The performance of the EGS procedure significantly degrades when the eigenvalues of the function differ greatly. In such cases, evolution strategies with correlated mutations performs considerably better. However, the generation of correlated mutations requires \( O(n^2) \) operations as opposed to \( O(n) \) operations of the EGS procedure as well as simple evolution strategies. Therefore, the net efficacy and thus the performance strongly depends on the computational costs of the objective function.

In addition to some performance improvements, the EGS procedure has the following advantage. Due to its hybrid nature, the EGS procedure features the advantages of evolutionary algorithms, i.e., maintaining populations, parallel execution, not getting stuck at plateaus, and simultaneously offering a new option for incorporating acceleration methods known from classical optimization methods. In this paper, one simple method, the momentum term, has been explored. The investigation of other methods, e.g., conjugate directions, are subject to future research. Some preliminary results on the utilization of individual step sizes and their adaptation have been discussed in [25], but a detailed discussion is beyond the scope of this paper.

### Appendix A

This appendix briefly explains why for \( \lambda \to \infty \) and small step sizes \( \sigma_t \), the estimate of (4), i.e., \( \tilde{g}_t = \sum_{i=1}^{\lambda} (f(x_i^t) - f(\bar{x}_t)) (x_i^t - \bar{x}_t) \), converges to the true gradient direction. Since all components of the \( \lambda \) test candidates are Gaussian distributed with standard deviation \( \sigma_t / \sqrt{\lambda} \), all test candidates are distributed on a hypersphere with radius \( \sigma_t \) located at the...
Fig. 16. Estimating the gradient direction at a given point \( x_t \) by generating mutations \( z_i \), and averaging according to (4). The subscript \( i \) of each mutation vector \( z_i \) corresponds to an imaginary angle \( \alpha = i \). Thus \( \alpha \) corresponds to \( z_i \) and \( \alpha = 0.5 \pi \).

current point \( \bar{x}_t \). Without loss of generality, the coordinate system can therefore be rotated such that the current point \( \bar{x}_t = (x_t, 0, \ldots, 0)^T \) is on the \( x_1 \)-axis as is depicted in Fig. 16.

For the case \( \lambda \to \infty \), the normalized gradient estimation

\[
\hat{g}_t = \lim_{\lambda \to \infty} \frac{1}{\lambda} \left( f(\bar{x}_t) - f(\bar{x}_t) \right) (\bar{x}_t - \bar{x}_t)
\]

has to be considered for obvious reasons. Due to the independence of all components, the gradient estimation can be considered component-wise

\[
\hat{g}_j = \lim_{\lambda \to \infty} \frac{1}{\lambda} \left( f(\bar{x}_t + z_j^i) - f(\bar{x}_t) \right) (z_j^i)
\]

with \( z_j^i \) denoting the \( j \)-th component of the \( i \)-th mutation vector \( z_i \), and \( \hat{g}_j \) denoting the \( j \)-th component of the gradient estimate \( \hat{g}_t \).

We now restrict ourselves to objective functions that are symmetric with respect to the \( x_1 \)-axis, i.e., \( f(x_1, x_2, \ldots, x_n) = f(x_1, -x_2, \ldots, -x_n) \). It is obvious that for all components \( j \geq 2 \), the sum \( \hat{g}_j = 0 \) vanishes. For the component \( \hat{g}_1 \) along the \( x_1 \)-axis, however, the sum does not vanish, but yields \( \hat{g}_1 > 0 \).

For illustration purposes, this appendix also discusses two simple examples. First, consider the objective function \( f(\bar{x}) = x_1^2 \), with the gradient \( \bar{g} = (1, 0, \ldots, 0)^T \) and \( f(\bar{x}) = \text{const} \) being \((n - 1)\)-dimensional hyperplanes perpendicular to the \( x_1 \)-axis. For this simple function, the first term of (4) reduces to \( f(\bar{x} + z_1^i) - f(\bar{x}) = z_1^i \), with \( z_1^i \) denoting the first component of the \( i \)-th mutation vector. For the case \( \lambda \to \infty \), the gradient estimate is given by

\[
\hat{g}_t = \lim_{\lambda \to \infty} \frac{1}{\lambda} \sum_{i=1}^{\lambda} \frac{1}{\lambda} (z_1^i)
\]

which leads for the first component \( \hat{g}_1 \) to

\[
\hat{g}_1 = \lim_{\lambda \to \infty} \frac{1}{\lambda} \sum_{i=1}^{\lambda} (z_1^i)^2 = \text{E}(z_1^2) = \sigma_1^2/n\]

since \( E(z_1) = 0 \). For all other components \( \hat{g}_{j \neq 1} \)

\[
\hat{g}_{j \neq 1} = \lim_{\lambda \to \infty} \frac{1}{\lambda} \sum_{i=1}^{\lambda} \frac{1}{\lambda} z_1^i z_j^i = 0
\]

since all components \( z_j^i \) are statistically independent and since \( E(z_1) = E(z_j) = 0 \).

As a second example, consider the sphere model \( f(\bar{x}) = \sum_j x_j^2 \). For a point \( \bar{x}_t = (x_t, 0, \ldots, 0)^T \), the first term of the gradient estimate is

\[
f(\bar{x}_t + z_j^i) - f(\bar{x}_t) = x_j^2 + 2 x_t z_1^i + \sum_{j=1}^{n} z_j^i
\]

\[
= x_j^2 + 2 x_t z_1^i + \sigma_j^2.
\]

The first component \( \hat{g}_1 \) is determined by

\[
\hat{g}_1 = \lim_{\lambda \to \infty} \frac{1}{\lambda} \sum_{i=1}^{\lambda} \frac{1}{\lambda} (2 x_t z_1^i + \sigma_1^2) = 2 x_t \sigma_1^2
\]

and all other components \( \hat{g}_{j \neq 1} \) are

\[
\hat{g}_{j \neq 1} = \lim_{\lambda \to \infty} \frac{1}{\lambda} \sum_{i=1}^{\lambda} \frac{1}{\lambda} (2 x_t z_1^i + \sigma_1^2) = 0.
\]

For objective functions that are asymmetric with respect to the \( x_1 \)-axis, the estimated gradient direction deviates from the \( x_1 \)-axis which is the case for the spherical model, at particular points for quadratic functions, and is approximately given for small \( \sigma_1 \). It should be noted again that it is not essential for the EGS procedure that the estimated gradient direction exactly equals the true gradient direction. Therefore, it does not have truly bad consequences, if the estimated direction deviates from the true gradient direction. In addition the (global) gradient can be precisely estimated by generating \( \lambda = n \) test candidates that are all perpendicular to each other, i.e., \( z_1^{i,j=1} = 0 \), which however requires \( O(n^2) \) operations (see also [14]).

APPENDIX B

When implementing the estimate \( \hat{g}_t = \sum_{i=1}^{\lambda} (f(\bar{x}_t^i) - f(\bar{x}_t)) (\bar{x}_t^i - \bar{x}_t) \) according to (4), the following details have to be observed. It might be that the gradient vanishes for any reason. When the procedure is applied to the step function \( f_{\text{step}} \), for example, it might happen that the first term always vanishes, since the step size \( \sigma_t \) is too small. In such cases, the unit vector \( \bar{c}_t = \hat{g}_t / \|\hat{g}_t\| \) is not defined, and unprotected calculations can result in a runtime error.
In cases where the gradient \( \hat{g} \) vanishes, the programmer has the following two options: 1) choose any of the generated test vectors or 2) continue with generating test candidates but temporarily increase the step size \( \sigma_t \) until the gradient does not vanish. Both options have more or less the same effect. In the first case, the procedure would do a step along an arbitrarily chosen mutation vector \( \Delta \vec{x} \). If then both test steps \( \sigma_t \xi \) and \( \sigma_t \zeta \) yield the same fitness, e.g., in the middle of a plateau of the step function \( f_{\text{step}} \), the procedure would increase the step size \( \sigma_t \) due to the selection given in (5). In the second case, the procedure would directly increase the step size \( \sigma_t \) until at least one test vector yields a fitness \( f(\Delta \vec{x} + \Delta \vec{z}) \) different from the parent’s fitness \( f(\Delta \vec{x}) \). As a result, it may happen that the procedure exhibit small oscillations around the optimal value of the objective function. For the experiments presented in this paper, the second option has been implemented.

ACKNOWLEDGMENT

The author gratefully thanks P. Eggenberger for several technical discussions as well as R. Riedl for checking mathematical issues. Thanks are also due to H.-G. Beyer for many suggestions and for preparing a preprint of his paper [17]. Special thanks are due to the anonymous reviewers as well as D. Fogel for his valuable comments and editorial assistance.

REFERENCES


