Evolutionary Computation: An Overview

Thomas Bäck
Informatik Centrum Dortmund
Center for Applied Systems Analysis (Casa)
Joseph-von-Fraunhofer-Str. 20
D–44227 Dortmund, Germany

Hans-Paul Schwefel
Universität Dortmund
Fachbereich Informatik
Lehrstuhl für Systemanalyse
D–44221 Dortmund, Germany

***Abstract*— In this paper, we present an overview of the most important representatives of algorithms gleaned from natural evolution, so-called evolutionary algorithms. Evolution strategies, evolutionary programming, and genetic algorithms are summarized, with special emphasis on the principle of strategy parameter self-adaptation utilized by the first two algorithms to learn their own strategy parameters such as mutation variances and covariances. Some experimental results are presented which demonstrate the working principle and robustness of the self-adaptation methods used in evolution strategies and evolutionary programming.**

General principles of evolutionary algorithms are discussed, and we identify certain properties of natural evolution which might help to improve the problem solving capabilities of evolutionary algorithms even further.

I. Evolutionary Computation and Optimization

More than 30 years ago, a number of innovative researchers at different places in the US and Europe independently came up with the idea of mimicking mechanisms of biological evolution in order to develop powerful algorithms for problems of adaptation and optimization. Since many optimal structures like the shape of birds' wings or the branching structure of blood vessels have emerged through biological evolution, the idea to utilize the underlying mechanism for the solution of optimization problems has motivated a considerable amount of research, resulting in several approaches that have proven their effectiveness and robustness in a variety of applications (see e.g. the bibliography on evolutionary algorithms collected by Alander [1]). Typically, an optimization application requires finding a setting $\vec{x} = (x_1, \ldots, x_n) \in M$ of free parameters of the system under consideration, such that a certain quality criterion $f : M \rightarrow \mathbb{R}$ (typically called the *objective function*) is maximized (or, equivalently, minimized):

$$f(\vec{x}) \rightarrow \max \quad . \tag{1}$$

The objective function might be given by real-world systems of arbitrary complexity (e.g., a reactor for biochemical processes [21] or a two-phase nozzle [32]), by simulation models implemented on a computer (e.g., a nuclear reactor simulation [6]), or it might be given by an analytical expression. Typically, a solution to the global optimization problem (1) requires finding a vector $\vec{x}^*$ such that $\forall \vec{x} \in M : \; f(\vec{x}) \leq f(\vec{x}^*) = f^*$. Characteristics such as *multimodality*, i.e., the existence of *local maxima* $\vec{x}'$ with

$$\exists \varepsilon > 0 \; : \; \forall \vec{x} \in M \; : \; \|\vec{x} - \vec{x}'\| < \varepsilon \; \Rightarrow f(\vec{x}) \leq f(\vec{x}') \tag{2}$$

(for the case $M \subseteq \mathbb{R}^n$), *constraints*, i.e., restrictions on the set $M \subset M_1 \times \ldots \times M_n$ by functions $g_j : M_1 \times \ldots \times M_n \rightarrow \mathbb{R}$ such that the set of *feasible* solutions is only a subset of the domain of the variables:

$$M = \{\vec{x} \in M_1 \times \ldots \times M_n \mid g_j(\vec{x}) \geq 0 \; \forall j\} \; , \tag{3}$$

and others such as large dimensionality of the problem, strong nonlinearities, non-differentiability, noisy and time-varying objective function values require the search for robust global optimization methods which are still applicable and yield useful results when traditional methods fail. Even if the general global optimization problem is unsolvable (see [59], p. 6), the identification of an improvement of the actual best solution by optimization is often already a big success for practical problems, and in many cases evolutionary algorithms provide an efficient and effective method to achieve this.

The three mainstream methods of evolutionary computation which have been established over the past thirty years are *genetic algorithms*, developed by Holland at Ann Arbor, MI [24; 26; 27], *evolutionary programming*, developed by Fogel at San Diego, CA [18; 19], and *evolution strategies*, developed by Rechenberg and Schwefel at Berlin, Germany [41; 42; 43; 52; 54; 56]. State-of-the-art overviews of the field of evolutionary computation are also given by Fogel [16] and Bäck [5]. These algorithms were all developed for applications where analytical information (like derivatives) is not available and the problem has a certain minimal complexity. They are most often associated with binary search spaces $M = \{0,1\}^n$ in case of genetic algorithms and real-valued search spaces $M = \mathbb{R}^n$ in case of evolution strategies, although genetic algorithms are also used for real-valued and permutation spaces, and evolution strategies and evolutionary programming were originally applied to discrete optimization problems. The search spaces and the corresponding algorithmic variants are discussed in more detail in sections II and III of this article.

All algorithms rely on the concept of a population of individuals (representing potential solutions to a given optimization problem), which undergo probabilistic operators such as mutation, selection, and (sometimes) recombination to evolve towards increasingly better fitness values of individuals. The fitness of an individual reflects its objective function value with respect to a particular objective function to be optimized. The mutation operator introduces innovation into the population by generating variations of individuals, and the recombination operator typically performs an information exchange between different

individuals from a population. The selection operator imposes a driving force on the process of evolution by preferring better individuals to survive and reproduce when the members of generation $t+1$ are selected. The main loop of the algorithm, consisting of recombination, mutation, fitness evalution, and selection, is iterated for a number of generations until the computing time is exhausted, a sufficiently good performing solution is found, or some other termination criterion is fulfilled.

The following pseudocode algorithm summarizes the major components of an evolutionary algorithm:

Algorithm 1

$t := 0$;
`initialize` $P(t)$;
`evaluate` $P(t)$;
while not `terminate` **do**
    $P'(t) :=$ `recombine` $P(t)$;
    $P''(t) :=$ `mutate` $P'(t)$;
    `evaluate` $P''(t)$;
    $P(t+1) :=$ `select` $(P''(t) \cup Q)$;
    $t := t + 1$;
od

In this algorithm, $P(t)$ denotes a population of μ individuals at generation t. Q is a special set of individuals that has to be considered for selection, e.g. $Q = P(t)$. An offspring population $P''(t)$ of size $\lambda \geq \mu$ is generated by means of (recombination and) mutation from the population $P(t)$, the offspring individuals are evaluated by calculating the objective function values $f(\vec{x}_k)$ for each of the solutions $\vec{x}_k$ represented by individuals in $P''(t)$, and selection based on the fitness values is performed to drive the process towards better solutions. The next two sections of this article give an overview of the instantiations of this general outline of an evolutionary algorithm that yield an evolution strategy and evolutionary programming algorithm (section II) and a genetic algorithm (section III). In section IV, we mention some other representatives of evolutionary algorithms, which can be characterized in a broader sense according to the underlying search spaces, and section V presents some concluding remarks.

II. ESs and EP for Continuous Spaces

Although they have not been originally defined for such problems, *evolution strategies* and *evolutionary programming* are presently most widely used for continuous parameter optimization problems of the general form

$$f : M \subseteq \mathbb{R}^n \rightarrow \mathbb{R} \quad . \tag{4}$$

We introduce evolution strategies first (section II.A), already explaining some of the relationships of this algorithm to evolutionary programming (section II.B).

A. Evolution Strategies

Evolution strategies are a development of Rechenberg and Schwefel, who started working in this field in the 1960s at the Technical University of Berlin, Germany, with first applications in experimental optimization applied to hydrodynamical problems like shape optimization of a bended pipe [36], drag minimization of a joint plate [41], and structure optimization of a two-phase flashing nozzle [53]. In the mid-sixties, Schwefel also implemented the first version of a *two membered* evolution strategy on a computer, using a continuous representation and a mutation operator based on normally distributed modifications with expectation zero and given variance [52]. The two membered strategy works by creating an n-dimensional vector of object variables from its parent by applying mutation with identical standard deviation to each object variable, and the better of the ancestor and its descendant is selected to survive for the next generation, while the other one is discarded. This simple selection mechanism is typically characterized by the term $(1+1)$-selection. Obviously, this simple strategy does not work with a population of individuals, but rather has some similarity to simulated annealing [45].

A first population-based, *multimembered* evolution strategy or $(\mu + 1)$-strategy was designed by Rechenberg to introduce a recombination operator, allowing the $\mu > 1$ parent individuals to form one offspring by recombination and mutation. The offspring eventually replaces the worst parent individual, if it performs better than that. This selection by extinction of the worst is comparable to the steady-state selection scheme introduced to genetic algorithms by Whitley [60; 30]. The $(\mu + 1)$-strategy was never widely used, however, because it does not provide a reasonable way to control the standard deviation of mutations like the 1/5-success rule in case of the $(1+1)$-strategy does [42].

This problem is solved in an elegant way by the (μ, λ)-strategy as introduced by Schwefel [54; 55; 56], which represents the state-of-the-art and uses both a parent and an offspring population of sizes $\lambda > \mu \geq 1$. The notation (μ, λ) indicates that μ parents create λ offspring by means of recombination and mutation, and the best μ offspring individuals are deterministically selected to replace the parents. Notice that this mechanism allows that the best member of the population at generation $t + 1$ might perform *worse* than the best individual at generation t, i.e., the method is not *elitist*, thus facilitating the strategy to accept temporary deteriorations that might help to leave the region of attraction of a local optimum and reach a better optimum. In contrast, the $(\mu + \lambda)$-strategy selects the μ survivors from the union of parents and offspring, such that a monotonous course of evolution is guaranteed. Due to recommendations by Schwefel, however, the (μ, λ)-strategy is preferred over the $(\mu + \lambda)$-strategy, although recent experimental findings seem to indicate that the latter performs as well as the (μ, λ)-strategy in many practical cases [23].

The individuals in a population of the (μ, λ)-strategy consist of the object variables $x_i \in \mathbb{R}$ $(1 \leq i \leq n)$ and so-called *strategy parameters*, i.e., variances and covariances of a generalized n-dimensional normal distribution for mutation. Schwefel incorporated these parameters into

the representation of individuals in order to facilitate the evolutionary *self-adaptation* of these parameters by applying evolutionary operators to the object variables and the strategy parameters for mutation at the same time, i.e., searching the space of solutions and strategy parameters together. This way, a suitable adjustment and diversity of mutation parameters should be provided under arbitrary circumstances.

More formally, an individual $\vec{a} = (\vec{x}, \vec{\sigma}, \vec{\alpha})$ consists of up to three components $\vec{x} \in I\!R^n$, $\vec{\sigma} \in I\!R^{n_\sigma}$, and $\alpha \in [-\pi, \pi]^{n_\alpha}$, where $n_\sigma \in \{1, \ldots, n\}$ and $n_\alpha \in \{0, (2n - n_\sigma) \cdot (n_\sigma - 1)/2\}$. The mutation operator works by adding a normally distributed random vector $\vec{z} \sim N(\vec{0}, \mathbf{C})$ with expectation vector $\vec{0}$ and covariance matrix $\mathbf{C}^{-1}$, where the covariance matrix is described by the mutated strategy parameters of the individual. Depending on the amount of strategy parameters incorporated into the representation of an individual, the following main variants of self-adaptation can be distinguished:

- $n_\sigma = 1$, $n_\alpha = 0$: The standard deviation for all object variables is identical (σ), and all object variables are mutated by adding normally distributed random numbers with

$$\sigma' = \sigma \cdot \exp(\tau_0 \cdot N(0,1)) \quad (5)$$
$$x_i' = x_i + \sigma' \cdot N(0,1) \quad , \quad (6)$$

  where $\tau_0 \propto (\sqrt{n})^{-1}$.
- $n_\sigma = n$, $n_\alpha = 0$: All object variables have their own, individual standard deviation σ_i, which determines the corresponding modification according to

$$\sigma_i' = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1)) \quad (7)$$
$$x_i' = x_i + \sigma_i' \cdot N(0,1) \quad , \quad (8)$$

  where $\tau' \propto (\sqrt{2n})^{-1}$ and $\tau \propto (\sqrt{2\sqrt{n}})^{-1}$.
- $n_\sigma = n$, $n_\alpha = n \cdot (n-1)/2$: The vectors $\vec{\sigma}$ and $\vec{\alpha}$ represent the complete covariance matrix of the n-dimensional normal distribution, where the covariances are given by rotation angles describing the coordinate rotations necessary to transform an uncorrelated mutation vector into a correlated one. The details of this mechanism can be found in [5] (pp. 68–71) or [44]. The mutation is performed according to

$$\sigma_i' = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1)) \quad (9)$$
$$\alpha_j' = \alpha_j + \beta \cdot N_j(0,1) \quad (10)$$
$$\vec{x}' = \vec{x} + N(\vec{0}, \mathbf{C}(\vec{\sigma}', \vec{\alpha}')) \quad (11)$$

  where $N(\vec{0}, \mathbf{C}(\vec{\sigma}', \vec{\alpha}'))$ denotes the correlated mutation vector and $\beta \approx 0.0873$.
- $1 < n_\sigma < n$: The general case of having neither just one nor the full number of different degrees of freedom available is also permitted, and implemented by the agreement to use σ_{n_σ} for mutating all x_i where $n_\sigma \leq i \leq n$.

The settings for the *learning rates* τ, τ' and τ_0 are recommended by Schwefel as reasonable heuristic settings (see [54], pp. 167–168), but one should have in mind that, depending on the particular topological characteristics of the objective function, the optimal setting of these parameters might differ from the values proposed. For $n_\sigma = 1$, however, Beyer has recently theoretically shown that, for the sphere model, the setting $\tau_0 \propto 1/\sqrt{n}$ is the optimal choice, maximizing the convergence velocity of the evolution strategy [10].

The amount of information included into the individuals by means of the self-adaptation principle increases from the simple case of one standard deviation up to the order of n^2 additional parameters, which reflects an enormous degree of freedom for the *internal models* of the individuals. This growing degree of freedom often enhances the global search capabilities of the algorithm at the cost of the expense in computation time, and it also reflects a shift from the precise *adaptation* of a few strategy parameters (as in case of $n_\sigma = 1$) to the exploitation of a large *diversity* of strategy parameters. In case of correlated mutations, Rudolph has shown that an approximation of the Hessian could be computed with an upper bound of $\mu+\lambda = (n^2+3n+4)/2$ on the population size, but the typical population sizes $\mu = 15$, $\lambda = 100$, independently of n, are certainly not sufficient to achieve this [44].

The choice of a logarithmic normal distribution for the modification of the standard deviations σ_i is motivated by the following heuristic arguments (see [54], p. 168):

1. A multiplicative process preserves positive values.
2. The median should equal one to guarantee that, on average, a multiplication by a certain value occurs with the same probability as a multiplication by the reciprocal value (i.e., the process would be neutral under absence of selection).
3. Small modifications should occur more often than large ones.

The effectiveness of this multiplicative logarithmic normal modification is presently also acknowledged in evolutionary programming, since extensive empirical investigations indicate some advantage of this scheme over the original additive self-adaptation mechanism used in evolutionary programming [47; 48; 49], where

$$\sigma_i' = \sigma_i \cdot (1 + \alpha \cdot N(0,1)) \quad (12)$$

(with a setting of $\alpha \approx 0.2$ [49]). Recent investigations indicate, however, that this becomes reversed when noisy objective functions are considered, where the additive mechanism seems to outperform multiplicative modifications [3].

The study by Gehlhaar and Fogel also indicates that the order of the modifications of x_i and σ_i has a strong impact on the effectiveness of self-adaptation: It is important to mutate the standard deviations first and to use the mutated standard deviations for the modification of object variables [23]. As the authors point out in that study, the reversed mechanism might suffer from generating offspring that have useful object variable vectors but bad strategy parameter vectors, because these have not been used to determine the position of the offspring itself.

To illustrate the working principle of self-adaptation, we perform a trivially simple but interesting experiment with a (15,100)-strategy and $n_\sigma = 1$, comparing the logarithmic normally distributed modification according to equation (5) and the additive modification according to equation (12). For this purpose, a time-varying version of the sphere model $f(\vec{x}) = \sum_{i=1}^{n}(x_i - x_i^*)^2$ is used, where the optimum location $\vec{x}^* = (x_1^*, \ldots, x_n^*)$ is changed every $T = 200$ generations. Ten independent experiments for $n = 30$ and 1000 generations per experiment are performed, and the average best objective function value and the average standard deviation within the population are reported. The learning rate $\tau_0 = \alpha = 1/\sqrt{30} \approx 0.1826$ was used in both experiments.

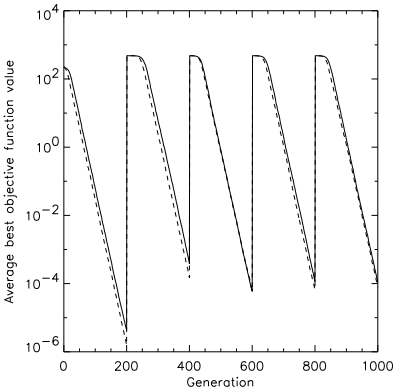

Fig. 1. Best objective function value plotted over the generation for the time-varying sphere model. The results were obtained by using a (15,100)-strategy with multiplicative (dashed line) and additive step size modification (solid line).

Figure 1 shows the behavior of the best objective function value over time. The solid curve, reflecting a modification after equation (12), and the dashed curve, reflecting the modification after equation (5), both illustrate the linear convergence of the algorithms during the first search interval of 200 generations. After the shift of the optimum location at generation 200, the search stagnates for a while on the bad new position, before the linear convergence is observed again.

Figure 2 shows the corresponding average standard deviations within the population, reflecting the steady process of self-adaptation of standard deviations both by decreasing them during the periods of linear convergence and by *increasing them during the periods of stagnation*, which can be explained by the necessity of an increase of standard deviations back to a level where they have an impact on the objective function value. The process of standard deviation increase, which is observed at the beginning of each interval, needs some time which does not yield any progress with respect to the objective function value. This time seems to be almost identical for the additive modification and for the multiplicative one, such that both processes behave very similar in our experiments. More work needs to be performed, however, to achieve any clear understanding of the general advantages or disadvantages of one self-adaptation scheme compared to the other.

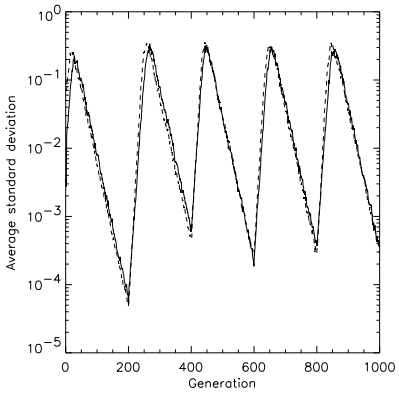

Fig. 2. Average standard deviation within the population for the multiplicative (dashed line) and additive step size modification (solid line).

A recent theoretical study by Beyer presents a first step towards this goal [10]. In this work, the author shows that the self-adaptation principle works for a variety of different probability density functions for the modification of step sizes, i.e., it is an extremely robust mechanism. Moreover, he indicates that equation (12) is obtained from (5) by Taylor expansion breaking off after the linear term, such that both should behave equal for small settings of the learning rates τ_0 and α, when $\tau_0 = \alpha$. This prediction is perfectly confirmed by the experiment reported here.

So far, evolution strategies have been introduced concerning the representation of individuals, the self-adaptation of strategy parameters, and the mutation operator. Just one additional component, the *recombination* operator, remains to be discussed. Typically, recombination is incorporated into the main loop of the evolution strategies as the first operator (see algorithm 1) and generates a new intermediate population of λ individuals by λ-fold application to the parent population, creating one individual per application from ϱ ($1 \leq \varrho \leq \mu$) individuals. Typically, $\varrho = 2$ or $\varrho = \mu$ (so-called global recombination) are chosen.

The recombination types for object variables and strategy parameters are often different in evolution strategies, and typical examples are *discrete recombination* (random exchanges of variables between parents, comparable to uniform crossover in genetic algorithms) and *intermediary recombination* (arithmetic averaging). For further details on these operators, see e.g. [8].

The advantages or disadvantages of recombination for a particular objective function can hardly be assessed in advance, and certainly no generally useful setting of recombination operators (such as the discrete recombination of object variables and global intermediary of strategy parameters as we have claimed in [5], pp. 82–83) exists. Recently, Kursawe has impressively demonstrated that, using an inappropriate setting of the recombination operator, the (15,100)-evolution strategy with $n_\sigma = n$ might even diverge on a sphere model for $n = 100$ [35]. While, for $n = 30$, reas-

onable convergence velocity is achieved for all settings of the recombination operator, for $n = 100$ only intermediary recombination of the object variables yields a non-divergent behavior, and the optimal performance is achieved for a combination of this setting and global intermediary recombination of the strategy parameters σ_i. These findings demonstrate that the appropriate choice of the recombination operator not only depends on the objective function topology, but also on the dimension of the objective function and the amount of strategy parameters incorporated into the individuals (assuming that the population size has to scale with the number of strategy parameters, it is certainly worthwhile to test the case $n_\sigma = n = 100$ with a larger population than just 100 offspring individuals).

B. Evolutionary Programming

Since its origins in the early 1960s, the goal of evolutionary programming is to achieve intelligent behavior through simulated evolution [18; 19]. D. Fogel defines intelligence as *the capability of a system to adapt its behavior to meet its goals in a range of environments*, clarifying how simulated evolution can be used as a basis to achieve this [16]. While the original form of evolutionary programming was proposed to operate on finite state machines and the corresponding discrete representations, most of the present variants of evolutionary programming are utilized for continuous parameter optimization problems.

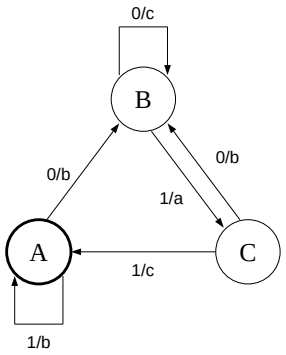

Fig. 3. Example of a finite state machine with three states.

Figure 3 presents a simple example for a finite state machine with three different states $S = \{A, B, C\}$, input alphabet $I = \{0, 1\}$, and output alphabet $O = \{a, b, c\}$. An edge between two states indicates a state transition, and the transition function $\delta : S \times I \to S \times O$ is specified by edge labels of the form i/o between states s_i, s_j, meaning that $\delta((s_i, i)) = (s_j, o)$, i.e., if the machine is in state s_i and receives input $i \in I$, the machine attains state s_j and produces an output symbol $o \in O$. Initially, the finite state machine is in a certain start state $s_0 \in S$ ($s_0 = A$ in the example presented in figure 3). By this mechanism, the finite state machine transforms a stream of input symbols (which is interpreted as the environment of the machine) into a stream of output symbols. The performance of a finite state machine with respect to its environment might then be measured on the basis of the machine's prediction capability, i.e., by comparing each output symbol with the next input symbol and measuring the worth of a prediction by some payoff function.

The evolutionary programming paradigm as implemented by Fogel worked with a population of $\mu > 1$ parent individuals which generated μ offspring by mutating each parent [17]. Mutation was implemented as a random change of the description of the finite state machines according to five different modifications: Change of an output symbol, change of a state transition, addition of a state, deletion of a state, or change of the initial state. The mutations were typically performed with uniform probability, and the number of mutations for a single offspring was either fixed or also chosen according to a probability distribution. After evaluating the offspring individuals, a selection of the μ best out of parents and offspring, i.e., a $(\mu + \mu)$-selection, was performed.

The general principle of a mutation-selection algorithm like this, using no recombination at all, received considerable criticism from researchers working in the field of genetic algorithms (see e.g. [24], p. 106, who concludes that this is an insufficiently powerful method). It is clear from a variety of recent empirical and theoretical results, however, that the role of mutation in genetic algorithms has been underestimated for about 30 years (e.g. [4; 14; 39]), while the importance of recombination has been overestimated [13].

Current variants of evolutionary programming for continuous parameter optimization problems have much in common with evolution strategies, especially concerning the representation of individuals, the mutation operator, and the self-adaptation of strategy parameters (see section II.A), which was independently redeveloped by D. Fogel in his PhD thesis [15]. As pointed out in section II.A, the additive modification of standard deviations as originally proposed by Fogel is presently often substituted by the multiplicative, logarithmic normally distributed modification.

A minor difference between evolutionary programming and evolution strategies consists in the choice of a probabilistic variant of $(\mu + \mu)$-selection in evolutionary programming, where each solution out of offspring and parent individuals is evaluated against $q > 1$ (typically, $q = 10$) other randomly chosen solutions from the union of parent and offspring individuals. For each comparison, a "win" is assigned if an individual's score is better or equal to that of its opponent, and the μ individuals with the greatest number of wins are retained to be parents of the next generation. As shown in [5] (pp. 96–99), this selection method is a probabilistic version of $(\mu+\mu)$-selection which becomes more and more deterministic as the number q of competitors is increased. It is still an open question in evolutionary algorithm research whether a probabilistic selection scheme is preferable over a deterministic one.

III. GAs for discrete problems

Canonical genetic algorithms as originally defined by Holland use a binary representation of individuals as fixed length strings over the alphabet $\{0, 1\}$, such that they are typically used to handle pseudoboolean optimization prob-

lems of the form

$$f : \{0,1\}^\ell \to I\!R \quad . \tag{13}$$

Sticking to the binary representation, genetic algorithms often enforce the utilization of encoding and decoding functions $h : M \to \{0,1\}^\ell$ and $h^{-1} : \{0,1\}^\ell \to M$ that facilitate to map solutions $\vec{x} \in M$ to binary strings $h(\vec{x}) \in \{0,1\}^\ell$ and vice versa, which sometimes requires rather complex mappings h and h^{-1}. In case of continuous parameter optimization problems, for instance, genetic algorithms typically encode a real-valued vector $\vec{x} \in I\!R^n$ by dividing a binary string $\vec{y} \in \{0,1\}^\ell$ into n segments of equal length, decoding each segment to yield the corresponding integer value, and mapping the integer value linearly to an interval $[u_i, v_i] \subset I\!R$ of real values [24].

The strong preference for using binary representations of solutions in genetic algorithms is typically derived from the *schema theory* of genetic algorithm, which tries to analyze GAs in terms of their expected schema sampling behavior. The term schema denotes a similarity template that represents a subset of $\{0,1\}^\ell$, and the *schema theorem* of genetic algorithms claims that the canonical genetic algorithm provides a near-optimal sampling strategy for schemata by exponentially increasing the number of well performing, short (i.e., with small distance between the leftmost and rightmost defined position) and low-order (i.e., with few specified bits) schemata (so-called building blocks) over subsequent generations (see [24] for a more detailed introduction to the schema theorem). The fundamental argument to justify the strong emphasis on binary alphabets is derived from the schema processing point of view by the fact that the number of schemata is maximized for a given finite number of search points, if a binary alphabet is used (see [24], pp. 40–41). Consequently, the schema theory presently seems to favor binary representations of solutions.

Practical experience as well as some theoretical hints regarding the binary encoding of continuous object variables [4; 61], however, indicate strongly that the binary representation has some serious disadvantages because it might introduce an additional multimodality into the objective function, thus making the combined objective function $f = f' \circ h^{-1}$ (where $f' : M \to I\!R$) more complex than the original problem f' was. In fact, the schema theory of genetic algorithms relies on approximations which assume infinite populations (see [27], pp. 78–83) and the optimization criterion to minimize the *overall* expected loss (corresponding to the sum of all fitness values of all individuals ever sampled during the evolution) rather than the more appropriate criterion to minimize the best fitness value ever found, such that, from a point of view that emphasizes the optimization capabilities of an algorithm, schema theory turns out to be not helpful.

The variation operators of canonical genetic algorithms, mutation and recombination, are typically applied with a strong emphasis on recombination. The standard algorithm performs a so-called one-point crossover, where two individuals are chosen randomly from the population, a position in the bitstrings is randomly determined as the crossover point, and an offspring is generated by concatenating the left substring of one parent and the right substring of the other parent. Numerous extensions of this operator, such as increasing the number of crossover points [12], uniform crossover (each bit is chosen randomly from the corresponding parental bits) [58], and others have been proposed, but similar to evolution strategies no generally useful recipe for the choice of a recombination operator can be given and the theoretical analysis of recombination is still to a large extent an open problem. Unlike evolution strategies, the recombination operator in genetic algorithms is typically applied with a certain probability p_c, and commonly proposed settings of the crossover probability are $p_c = 0.6$ [29] and $p_c \in [0.75, 0.95]$ [50].

Mutation in genetic algorithms was introduced as a dedicated "background operator" of small importance (see [27], pp. 109–111). Mutation works by inverting bits with very small probability such as $p_m = 0.001$ [29], $p_m \in [0.005, 0.01]$ [50], or $p_m = 1/\ell$ [39]. Recent studies have impressively clarified, however, that much larger mutation rates, decreasing over the course of evolution, are often helpful with respect to the convergence reliability and velocity of a genetic algorithm, and that even self-adaptive mutation rates are effective for pseudoboolean problems [4; 7; 62].

Selection is typically a probabilistic operator, using the relative fitness $p(\vec{a}_i) = f(\vec{a}_i)/\sum_{j=1}^{\mu} f(\vec{a}_j)$ to determine the selection probability of an individual $\vec{a}_i$ (*proportional selection*). This method requires positive fitness values and a maximization task, such that often *scaling functions* are utilized to transform the fitness values accordingly (see e.g. [24], pp. 124).

IV. Other Evolutionary Algorithm Variants

Depending on the representational requirements of the problem domains M that were to be handled, a variety of variants of evolutionary algorithms have been proposed and are under constant further development. Due to space restrictions, it is only possible here to mention some of these, namely *order-based genetic algorithms*, *real-coded genetic algorithms*, *classifier systems*, and *genetic programming*, to give a brief statement of their domain of application, and to refer to the basic literature where a more detailed introduction can be found (see also [38] for an excellent overview).

A. Order-Based Genetic Algorithms

For searching the space of *permutations* $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$, a binary representation as used in canonical GAs is not well suited, because it is difficult to map binary strings to a permutation of integer values and preserve permutations under mutation and crossover (but it can be done, see [9]). Order-based GAs work directly on the permutation, applying specialized recombination (such as *order crossover* or *partially matched crossover*) and mutation operators (such as random exchanges of two elements of the permutation) which preserve permutations (see e.g. [24],

pp. 166–179 for an overview).

B. Real-Coded Genetic Algorithms

Evolutionary algorithms that use a floating-point representation for search spaces $M \subseteq \mathbb{R}^n$ have sometimes been called real-coded GAs [25]. Somewhere between evolution strategies and genetic algorithms, real-coded GAs typically do not use techniques of self-adaptation, but might utilize time-decreasing step sizes [38] or change operator probabilities based on their observed performance [11].

C. Classifier Systems

Classifier Systems use an evolutionary algorithm to search the space of *production rules* (often encoded by strings over a ternary alphabet) of a learning system capable of induction and generalization [28] (see also [24], chapter 6). Typically, the *Michigan* approach and the *Pittsburgh* approach are distinguished according to whether an individual corresponds with a rule of the rule-based system (Michigan approach), or with a complete rule base (Pittsburgh approach).

D. Genetic Programming

Genetic programming transfers the paradigm of evolutionary search to the space of computer programs in a language suitable to modification by operators such as mutation and recombination. The dominant approach to genetic programming uses (a subset of) LISP programs as the search space [33], but other programming languages including machine code are also used (see e.g. [31]).

V. Conclusions

In the preceding section, we have presented a fairly brief overview of current algorithmic instantiations of the general idea to simulate processes of natural evolution on a computer in order to solve search and optimization problems in a variety of application fields. The large success of various different realizations of evolutionary algorithms indicates that the underlying principles are extremely general and robust, allowing the search within arbitrary search spaces such as nucleotide base sequences, binary sequences, real-vealued vectors, permutations, and others. Even the simultaneous search within the space of object variables and strategy parameters is possible by means of evolutionary processes, and again the corresponding principle of *self-adaptation* turns out to be very robust with respect to the probability density function chosen for mutation.

From these robustness observations, it is straightforward to ask for the general underlying principles that make evolutionary algorithms work so well on many problems, and to try to derive the corresponding design principles for variation operators of an evolutionary algorithm. Concerning natural evolution, the neo-Darwinian paradigm asserts that the basic processes of evolution are *reproduction*, *mutation*, *competition*, and *selection*. Mutation is guaranteed in any reproducing system in a positively entropic environment, and competition and selection are just the consequences of a population expanding in a finite environment (see e.g. [37], or [16], chapter 2).

Concerning mutation, it is also observed in nature that small changes of the *phenotypes* of individuals (i.e., their behavior and physical embodiment) are more likely to occur than large changes — a property which characterizes mutation (a process that occurs on the level of *genotypes*, i.e., the genetic information of organisms) by means of its impact on the fitness of an individual. Consequently, the requirement of *strong causality* for fitness functions as formulated by Rechenberg ([43], pp. 126–130), which states that small changes of the object variables should cause small changes of the objective function value, reflects this principle of natural evolution. It gives a good argument for the utilization of normally distributed variations with expectation zero in evolution strategies and evolutionary programming. On the contrary, the mutation operator used in genetic algorithms for continuous parameter optimization does not follow this principle, because even a slight modification of the binary string might cause a large modification of the corresponding decoded object variable.

Concerning selection, its dominant property consists in favoring those species for survival and further evolution that are best adapted to their environment — a property which is certainly well modeled in most evolutionary algorithms. Others than this *directed* variant of selection, such as *stabilizing* selection (where individuals of medium value of a certain biological trait are preferred) or *disruptive* selection (where the individuals covering the extreme values of a biological trait are preferred) are also found in nature (e.g. [22], pp. 174–175), but such schemes have not been transferred to evolutionary algorithms so far.

From a theoretical point of view, it is an important goal to formulate general design principles for evolutionary algorithms and their variation operators, in order to allow for a more systematic development of such algorithms. An attempt to formulate such principles is presented by Radcliffe and Surry, who generalize the definition of schemata from binary strings to arbitrary search spaces by introducing so-called *formae* [40]. They then introduce *representation-independent* recombination operators, which have features claimed to be helpful for evolutionary search. The basic properties introduced are *respect* (every child contains all the alleles common to its parents), *transmission* (each allele in every child is present in at least one of its parents), and *assortment* (it is possible for the operator to produce any solution that contains only alleles present in the parents). Concerning mutations, they also emphasize on the property of smaller mutations being preferred over larger ones. These basic properties of variation operators can be used to classify the existing operators for different representations, and to identify those operators which should be best suited for a given search space. While the results reported in [40] are focused on the traveling salesman problem, it is certainly an interesting approach that might be a first step towards a general, representation-independent theory of evolutionary algorithms.

Looking back at the natural model, one has to emphas-

ize clearly that we are far from using all potentially helpful features of evolution within evolutionary algorithms. Comparing natural evolution and the algorithms discussed here, we immediately identify a list of important differences, which all might be exploited to obtain more powerful search algorithms *and* a better understanding of natural evolution:

- Natural evolution works under dynamically changing environmental conditions, with instationary optima and even changing optimization criteria, and the individuals themselves are also changing the structure of the adaptive landscape during adaptation [51]. In evolutionary algorithms, environmental conditions are often static, but non-elitist variants are able to deal with changing environments. It is certainly worthwhile, however, to consider a more flexible life span concept for individuals in evolutionary algorithms than just the extremes of a maximum life span of one generation (as in a (μ, λ)-strategy) and of an unlimited life span (as in an elitist strategy), by introducing an aging parameter [57].
- The long-term goal of evolution consists in the maintainance of *evolvability* of a population, guaranteed by mutation and a perservation of diversity within the population (the term *meliorization* describes this more appropriately than optimization or adaptation do). In contrast, evolutionary algorithms often aim at finding a precise solution and converging to this solution.
- In natural evolution, many criteria need to be met at the same time, while most evolutionary algorithms are designed for single fitness criteria (see [20] for an overview of the existing attempts to apply evolutionary algorithms to multiobjective optimization). The concepts of *diploidy* or *polyploidy*, combined with *dominance* and *recessivity* [34] as well as the idea of introducing two sexes with different selection criteria might be helpful for such problems.
- Natural evolution neither assumes global knowledge (about all fitness values of all individuals) nor a generational synchronization, while many evolutionary algorithms still identify an interation of the algorithm with one complete generation update. Fine-grained asynchronously parallel variants of evolutionary algorithms, introducing local neighborhoods for recombination and selection and a time-space organization like in cellular automata [46] represent an attempt to overcome these restrictions.
- Though a "recombination" operator is often used in evolutionary algorithms, no implementations have so far realized a real distinction of two different kinds of sexes — which might be helpful for meeting different criteria as well as different constraints.
- The genotype-phenotype mapping in nature, realized by the *genetic code* as well as the *epigenetic apparatus*, has evolved over time, while the mapping is usually fixed in evolutionary algorithms. An evolutionary self-adaptation of the genotype-phenotype mapping might be a powerful way to make the search more flexible, starting with a coarse-grained, volume-oriented search and focusing on promising regions of the search space as the evolution proceeds.
- Other topics, such as multi-cellularity and *ontogeny* of individuals, up to the development of their own brains (individual learning, such as accounted for by the Baldwin effect in evolution [2]), are usually not modeled in evolutionary algorithms. The self-adaptation of strategy parameters is just a first step into this direction, realizing the idea that each individual might have its own internal strategy to deal with its environment. This strategy might be more complex than the simple mutation parameters presently taken into account by evolution strategies and evolutionary programming.

Though evolutionary techniques provide a robust and widely applicable technique for problem solving, there is no inevitable necessity to use them: For special tasks, depending on the knowledge about the problem, it is often possible to develop a tailored solution technique for the actual problem — at the cost of a certain development time. There is always a certain "rest of problems," however, where no specific method is available, and this is the domain of evolutionary algorithms.

On the other hand, as evolutionary phenomena exist all around of us, it would be nice to understand evolution a bit better than we currently do. Consequently, we need models of evolutionary processes, and evolutionary algorithms are a first approximation to such models.

Therefore, we claim that both aspects of evolutionary algorithms — the problem solving aspect and the aspect of modeling natural evolution — are important and should be considered worthwhile for further research, leading to more powerful, better characterized search algorithms and a better understading of evolution (and finally ourselves).

Acknowledgement

The first author gratefully acknowledges financial support by the project EVOALG, grant 01 IB 403 A from the German BMBF ministry.

REFERENCES

[1] J. T. Alander. An indexed bibliography of genetic algorithms: Years 1957–1993. Art of CAD Ltd, Espoo, Finland, 1994.

[2] R. W. Anderson. Genetic mechanisms underlying the baldwin effect are evident in natural antibodies. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 547–564. The MIT Press, Cambridge, MA, 1995.

[3] P. J. Angeline. The effects of noise on self-adaptive evolutionary optimization. In L. J. Fogel, P. J. Angeline, and T. Bäck, editors, *Proceedings of the Fifth Annual Conference on Evolutionary Programming*. The MIT Press, Cambridge, MA, 1996.

[4] Th. Bäck. Optimal mutation rates in genetic search. In S. Forrest, editor, *Proceedings of the Fifth Interna-*

tional Conference on Genetic Algorithms, pages 2–8. Morgan Kaufmann, San Mateo, CA, 1993.

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

[6] Th. Bäck, J. Heistermann, C. Kappler, and M. Zamparelli. Evolutionary algorithms support refueling of pressurized water reactors. In *Proceedings of the Third IEEE Conference on Evolutionary Computation*. IEEE Press, Piscataway, NJ, 1996.

[7] Th. Bäck and M. Schütz. Intelligent mutation rate control in canonical genetic algorithms. In Z. Ras, editor, *Proceedings of the Ninth International Symposium on Methodologies for Intelligent Systems*, Lecture Notes in Computer Science. Springer, Berlin, 1996.

[8] Th. Bäck and H.-P. Schwefel. Evolution strategies I: Variants and their computational implementation. In G. Winter, J. Périaux, M. Galán, and P. Cuesta, editors, *Genetic Algorithms in Engineering and Computer Science*, chapter 6, pages 111–126. Wiley, Chichester, 1995.

[9] J. C. Bean. Genetics and random keys for sequences and optimization. Technical Report 92–43, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, MI, 1993.

[10] H.-G. Beyer. Toward a theory of evolution strategies: Self-adaptation. *Evolutionary Computation*, 3(3):311–348, 1995.

[11] L. Davis. Adapting operator probabilities in genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 61–69. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[12] L. J. Eshelman, R. A. Caruna, and J. D. Schaffer. Biases in the crossover landscape. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 10–19. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[13] L. J. Eshelman and J. D. Schaffer. Crossover's niche. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 9–14. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[14] T. C. Fogarty. Varying the probability of mutation in the genetic algorithm. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 104–109. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[15] D. B. Fogel. *Evolving Artificial Intelligence*. PhD thesis, University of California, San Diego, CA, 1992.

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

[17] L. J. Fogel. Autonomous automata. *Industrial Research*, 4:14–19, 1962.

[18] L. J. Fogel. Toward inductive inference automata. In *Proceedings of the International Federation for Information Processing Congress*, pages 395–399, Munich, 1962.

[19] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, New York, 1966.

[20] C. M. Fonseca and P. J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16, 1995.

[21] S. Freyer, D. Weuster-Botz, and C. Wandrey. Medienoptimierung mit Genetischen Algorithmen. *BioEngineering*, 8(5+6):16–25, 1992.

[22] D. J. Futuyma. *Evolutionsbiologie*. Birkhäuser Verlag, Basel, 1990.

[23] D. K. Gehlhaar and D. B. Fogel. Tuning evolutionary programming for conformationally flexible molecular docking. In L. J. Fogel, P. J. Angeline, and T. Bäck, editors, *Proceedings of the Fifth Annual Conference on Evolutionary Programming*. The MIT Press, Cambridge, MA, 1996.

[24] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison Wesley, Reading, MA, 1989.

[25] D. E. Goldberg. The theory of virtual alphabets. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature — Proceedings 1st Workshop PPSN I*, volume 496 of *Lecture Notes in Computer Science*, pages 13–22. Springer, Berlin, 1991.

[26] J. H. Holland. Outline for a logical theory of adaptive systems. *Journal of the Association for Computing Machinery*, 3:297–314, 1962.

[27] J. H. Holland. *Adaptation in natural and artificial systems*. The University of Michigan Press, Ann Arbor, MI, 1975.

[28] J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard. *Induction: Processes of Inference, Learning, and Discovery*. The MIT Press, Cambridge, MA, 1986.

[29] K. A. De Jong. *An analysis of the behaviour of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975. Diss. Abstr. Int. 36(10), 5140B, University Microfilms No. 76–9381.

[30] K. A. De Jong and J. Sarma. Generation gaps revisited. In D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 19–28. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[31] K. E. Kinnear, editor. *Advances in Genetic Programming*. The MIT Press, Cambridge, MA, 1994.

[32] J. Klockgether and H.-P. Schwefel. Two–phase nozzle and hollow core jet experiments. In D.G. Elliott, editor, *Proc. 11th Symp. Engineering Aspects of Magnetohydrodynamics*, pages 141–148, California Institute of Technology, Pasadena CA, March 24–26, 1970.

[33] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Complex Adaptive Systems. The MIT Press, Cambridge, MA, 1992.

[34] F. Kursawe. A variant of Evolution Strategies for vector optimization. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature — Proceedings 1st Workshop PPSN I*, volume 496 of *Lecture Notes in Computer Science*, pages 193–197. Springer,

Berlin, 1991.

[35] F. Kursawe. Towards self-adapting evolution strategies. In *Proceedings of the Second IEEE Conference on Evolutionary Computation*, pages 283–288. IEEE Press, Piscataway, NJ, 1995.

[36] H. J. Lichtfuss. Evolution eines Rohrkrümmers. Diplomarbeit, Technische Universität Berlin, 1965.

[37] E. Mayr. *Toward a new Philosophy of Biology: Observations of an Evolutionist.* The Belknap Press of Harvard University Press, Cambridge, MA, and London, GB, 1988.

[38] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, Berlin, 1996.

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

[40] N. J. Radcliffe and P. D. Surry. Fitness variance of formae and performance prediction. In M. D. Vose L. D. Whitley, editor, *Foundations of Genetic Algorithms 3*, pages 51–72. Morgan Kaufmann Publishers, San Francisco, CA, 1995.

[41] I. Rechenberg. Cybernetic solution path of an experimental problem. Royal Aircraft Establishment, Library translation No. 1122, Farnborough, Hants., UK, August 1965.

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

[43] I. Rechenberg. *Evolutionsstrategie '94*, volume 1 of *Werkstatt Bionik und Evolutionstechnik.* Frommann–Holzboog, Stuttgart, 1994.

[44] G. Rudolph. On correlated mutations in evolution strategies. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature 2*, pages 105–114. Elsevier, Amsterdam, 1992.

[45] G. Rudolph. Massively parallel simulated annealing and its relation to evolutionary algorithms. *Evolutionary Computation*, 1(4):361–382, 1993.

[46] G. Rudolph and J. Sprave. A cellular genetic algorithm with self–adjusting acceptance threshold. In *Proceedings of the First IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pages 365–372. IEE, London, 1995.

[47] N. Saravanan. Learning of strategy parameters in evolutionary programming: An empirical study. In A. V. Sebald and L. J. Fogel, editors, *Proceedings of the Third Annual Conference on Evolutionary Programming.* World Scientific, Singapore, 1994.

[48] N. Saravanan and D. B. Fogel. Evolving neurocontrollers using evolutionary programming. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 1, pages 217–222. IEEE Press, Piscataway, NJ, 1994.

[49] N. Saravanan, D. B. Fogel, and K. M. Nelson. A comparison of methods for self-adaptation in evolutionary algorithms. *BioSystems*, 36:157–166, 1995.

[50] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 51–60. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[51] J. Schull. The view from the adaptive landscape. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature — Proceedings 1st Workshop PPSN I*, volume 496 of *Lecture Notes in Computer Science*, pages 415–427. Springer, Berlin, 1991.

[52] H.-P. Schwefel. *Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik.* Diplomarbeit, Technische Universität Berlin, 1965.

[53] H.-P. Schwefel. Projekt MHD-Staustrahlrohr: Experimentelle Optimierung einer Zweiphasendüse, Teil I. Technischer Bericht 11.034/68, 35, AEG Forschungsinstitut, Berlin, October 1968.

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

[55] H.-P. Schwefel. *Numerical Optimization of Computer Models.* Wiley, Chichester, 1981.

[56] H.-P. Schwefel. *Evolution and Optimum Seeking.* Sixth-Generation Computer Technology Series. Wiley, New York, 1995.

[57] H.-P. Schwefel and G. Rudolph. Contemporary evolution strategies. In F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, editors, *Advances in Artificial Life. Third International Conference on Artificial Life*, volume 929 of *Lecture Notes in Artificial Intelligence*, pages 893–907. Springer, Berlin, 1995.

[58] G. Syswerda. Uniform crossover in genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[59] A. Törn and A. Žilinskas. *Global Optimization*, volume 350 of *Lecture Notes in Computer Science.* Springer, Berlin, 1989.

[60] D. Whitley. The GENITOR algorithm and selection pressure: Why rank–based allocation of reproductive trials is best. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[61] A. H. Wright. Genetic algorithms for real parameter optimization. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 205–218. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[62] M. Yanagiya. A simple mutation-dependent genetic algorithm. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, page 659. Morgan Kaufmann Publishers, San Mateo, CA, 1993.