Evolutionary Computation: Comments on the History and Current State

Thomas Bäck, *Member*, *IEEE*, Ulrich Hammel, and Hans-Paul Schwefel

Abstract— **Evolutionary computation has started to receive significant attention during the last decade, although the origins can be traced back to the late 1950s. This article surveys the history as well as the current state of this rapidly growing field. We describe the purpose, the general structure and the working principles of different approaches, including *genetic algorithms* (GA) (with links to *genetic programming* (GP) and *classifier systems* (CS)), *evolution strategies* (ES), and *evolutionary programming* (EP), by analysis and comparison of their most important constituents (i.e., representations, variation operators, reproduction and selection mechanism). Finally, we give a brief overview on the manifold of application domains, although this necessarily must remain incomplete.**

Keywords— **Genetic algorithms, evolutionary programming, evolution strategies, genetic programming, classifier systems, evolutionary computation.**

I. Evolutionary Computation: Roots and Purpose

THIS first issue of the IEEE Transactions on Evolutionary Computation marks an important point in the history of the rapidly growing field of evolutionary computation, and we are glad to participate in this event. In preparation for this summary, we strove to provide a comprehensive review of both the history and the state-of-the-art in the field for both the novice and the expert in evolutionary computation. The number of excellent publications currently available in this field required strong selections of the material to be presented here. These selections are necessarily subjective and we regret any significant omissions.

Although the origins of evolutionary computation can be traced back to the late 1950s (see e.g. the influencing works of Bremermann [1], Friedberg [2], [3], Box [4] and others), the field remained relatively unknown to the broader scientific community for almost three decades. This was largely due to the lack of available powerful computer platforms at that time, but also due to some methodological shortcomings of those early approaches (see e.g. Fogel [5, p. 103]).

The fundamental work of Holland [6], Rechenberg [7], Schwefel [8] and Fogel [9] served to slowly change this picture during the 1970s and we currently observe a remarkable and steady (still exponential) increase in the number of publications (see e.g. the bibliography [10]) and conferences in this field, a clear demonstration of the scientific as well as economic relevance of this subject matter.

But what are the benefits of evolutionary computation (compared to other approaches) which may justify the effort invested in this area? We argue that the most significant advantage of using evolutionary search lies in the gain of flexibility and adaptability to the task at hand, in combination with robust performance (although this depends on the problem class) and global search characteristics. In fact, evolutionary computation should be understood as a general adaptable concept for problem solving, especially well suited for solving difficult optimization problems, rather than a collection of related and ready-to-use algorithms.

The majority of current implementations of evolutionary algorithms descend from three strongly related but independently developed approaches: *genetic algorithms*, *evolutionary programming*, and *evolution strategies*.

Genetic algorithms, introduced by Holland [6], [11], [12] and subsequently studied by De Jong [13], [14], [15], [16], Goldberg [17], [18], [19], [20], [21], and others such as Davis [22], Eshelman [23], [24], Forrest [25], Grefenstette [26], [27], [28], [29], Koza [30], [31], Mitchell [32], Riolo [33], [34], Schaffer [35], [36], [37], to name only a few, have been originally proposed as a general model of adaptive processes, but by far the largest application of the techniques is in the domain of optimization [15], [16]. Since this is true for all three of the main-stream algorithms presented in this paper we will discuss their capabilities and performance mainly as optimization strategies.

Evolutionary programming, introduced by Fogel [9], [38] and extended in Burgin [39], [40], Atmar [41], Fogel [42], [43], [44], and others, was originally offered as an attempt to create artificial intelligence. The approach was to evolve finite state machines (FSM) to predict events on the basis of former observations. An FSM is an abstract machine which transforms a sequence of input symbols into a sequence of output symbols. The transformation depends on a finite set of states and a finite set of state transition rules. The performance of an FSM with respect to its environment might then be measured on the bases 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.

Evolution strategies as developed by Rechenberg [45], [46] and Schwefel [47], [48] and extended by Herdy [49], Kursawe [50], Ostermeier [51], [52], Rudolph [53], Schwefel [54], and others, were initially designed with the goal

The research of Th. Bäck is supported by a grant from the German BMBF, project EVOALG. He is with the Informatik Centrum Dortmund, Center for Applied Systems Analysis (CASA), Joseph-von-Fraunhofer-Str. 20, D–44227 Dortmund, Germany, and Leiden University, Niels Bohrweg 1, NL–2333 CA Leiden, The Netherlands. EMail: baeck@icd.de

U. Hammel and H.-P. Schwefel are with the Computer Science Department, Dortmund University, D–44221 Dortmund, Germany. EMail: hammel@LS11.informatik.uni-dortmund.de and schwefel@LS11.informatik.uni-dortmund.de

of solving difficult discrete and continuous, mainly experimental [55], parameter optimization problems.

During the 1980s, advances in computer performance enabled the application of evolutionary algorithms to solve difficult real-world optimization problems, and the solutions received a broader audience. In addition, beginning in 1985, international conferences on the techniques were offered ([56], [57], [58], [59], [60], [61] mainly focusing on genetic algorithms, [62], [63], [64], [65], [66] with an early emphasis on evolutionary programming, [67], [68], [69] as small workshops on theoretical aspects of genetic algorithms, [70] as a genetic programming conference, [71], [72], [73], [74] with the general theme of problem solving methods gleaned from nature, and [75], [76], [77], [78] with the general topic of evolutionary computation). But somewhat surprisingly, the researchers in the various disciplines of evolutionary computation remained isolated from each other until the meetings in the early 1990s [59], [63], [71].

The remainder of this paper is intended as an overview of the current state of the field. We cannot claim that this overview is close to complete. As good starting points for further studies we recommend [79], [80], [22], [5], [18], [81], [31], [82], [32], [48]. In addition moderated mailing lists (e.g. GA-List-Request@AIC.NRL.NAVY.MIL, EP-List-Request@magenta.me.fau.edu) and newsgroups (e.g. comp.ai.genetic) allow for keeping track of current events and discussions in the field.

In the next section we describe the application domain of evolutionary algorithms and contrast them with the traditional approach of mathematical programming.

II. Optimization, Evolutionary Computation, and Mathematical Programming

In general, an optimization problem requires finding a setting $\vec{x} \in M$ 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. The 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 several *local maxima* $\vec{x}'$ with

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

(where ρ denotes a distance measure on M), *constraints*, i.e., restrictions on the set M by functions $g_j : M \rightarrow \mathbb{R}$ such that the set of *feasible* solutions $F \subseteq M$ is only a subset of the domain of the variables:

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

and other factors, such as large dimensionality, strong nonlinearities, non-differentiability and noisy and time-varying objective functions, frequently lead to difficult if not unsolvable optimization tasks (see [83, p. 6]). But even in the latter case, the identification of an improvement of the currently known best solution through 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.

Optimization problems occur in many technical, economic, and scientific projects, like cost-, time- and risk-minimization or quality-, profit- and efficiency-maximization [10], [22] (see also [80, part G]). Thus, the development of general strategies is of great value.

In real-world situations the objective function f and the constraints g_j are often not analytically treatable or are even not given in closed form, e.g., if the function definition is based on a simulation model [84], [85].

The traditional approach in such cases is to develop a formal model that resembles the original functions close enough but is solvable by means of traditional mathematical methods such as linear and nonlinear programming. This approach most often requires simplifications of the original problem formulation. Thus, an important aspect of mathematical programming lies in the design of the formal model.

No doubt, this approach has proven to be very successful in many applications but has several drawbacks which motivated the search for novel approaches, where evolutionary computation is one of the most promising directions. The most severe problem is that, due to oversimplifications, the computed solutions do not solve the original problem. Such problems, e.g. in the case of simulation models, are then often considered unsolvable.

The fundamental difference in the evolutionary computation approach is to adapt the method to the problem at hand. In our opinion, evolutionary algorithms should not be considered as off-the-peg, ready-to-use algorithms but rather as a general concept which can be tailored to most of the real-world applications that often are beyond solution by means of traditional methods. Once a successful EC-framework has been developed it can be incrementally adapted to the problem under consideration [86], to changes of the requirements of the project, to modifications of the model and to the change of hardware resources.

III. The Structure of an Evolutionary Algorithm

Evolutionary algorithms mimic the process of natural evolution, the driving process for the emergence of complex and well adapted organic structures. To put it succinctly and with strong simplifications, evolution is the result of the interplay between the creation of new genetic information and its evaluation and selection. A single individual of a population is affected by other individuals of the population (e.g., by food competition, predators, and mating), as well as by the environment (e.g., by food supply and climate). The better an individual performs under these conditions the greater is the chance for the individual to live for a longer while and generate offspring, which in turn

inherit the (disturbed) parental genetic information. Over the course of evolution, this leads to a penetration of the population with the genetic information of individuals of above-average fitness. The non-deterministic nature of reproduction leads to a permanent production of novel genetic information and therefore to the creation of differing offspring (see [87], [79], [5] for more details).

This neo-Darwinian model of organic evolution is reflected by the structure of the general evolutionary algorithm:

Algorithm 1:

$t := 0$;
`initialize` $P(t)$;
`evaluate` $P(t)$;
while not `terminate` **do**
    $P'(t) :=$ `variation`$(P(t))$;
    `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 might be considered for selection, e.g., $Q = P(t)$ (but $Q = \emptyset$ is possible as well). An offspring population $P'(t)$ of size λ is generated by means of variation operators such as recombination and / or mutation (but others such as inversion [11, pp. 106–109] are also possible) from the population $P(t)$. The offspring individuals are then 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 toward better solutions. It should be noted that $\lambda = 1$ is possible, thus including so-called *steady-state* selection schemes [88], [89] if used in combination with $Q = P(t)$. Furthermore, by choosing $1 \leq \lambda \leq \mu$ an arbitray value of the *generation gap* [90] is adjustable, such that the transition between strictly generational and steady-state variants of the algorithm is also taken into account by the formulation offered here. It should also be noted that $\lambda > \mu$, i.e., a reproduction surplus, is the normal case in nature.

IV. Designing an Evolutionary Algorithm

As mentioned, at least three variants of evolutionary algorithms have to be distinguished: genetic algorithms, evolutionary programming and evolution strategies. From these ("canonical") approaches innumerable variants have been derived. Their main differences lie in:

- The representation of individuals.
- The design of the variation operators (mutation and/or recombination).
- The selection / reproduction mechanism.

In most real-world applications the search space is defined by a set of objects, e.g. processing units, pumps, heaters and coolers of a chemical plant, each of which have different parameters such as energy consumption, capacity, etc. Those parameters which are subject to optimization constitute the so-called *phenotype space*. On the other hand the genetic operators often work on abstract mathematical objects like binary strings, the *genotype space*. Obviously, a mapping or coding function between the phenotype and genotype space is required. Figure 1 sketches the situation (see also [5, p. 38-43]).

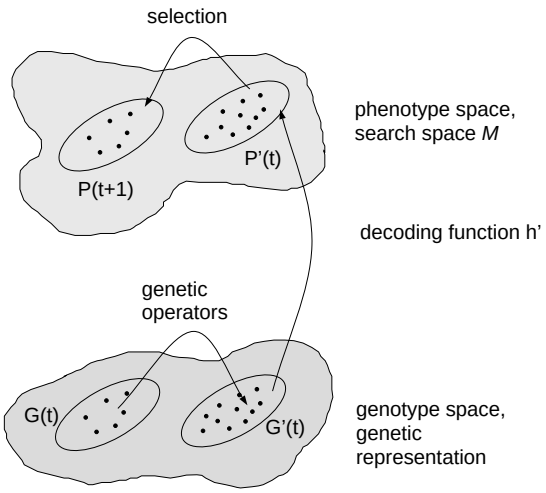


Fig. 1. The relation of genotype space and phenotype space after Fogel [5, p. 39].

In general, two different approaches can be followed. The first is to choose one of the standard algorithms and to design a decoding function according to the requirements of the algorithm. The second suggests designing the representation as close as possible to the characteristics of the phenotype space almost avoiding the need for a decoding function.

Many empirical and theoretical results are available for the standard instances of evolutionary algorithms, which is clearly an important advantage of the first approach, especially with regard to the reuse and parameter setting of operators. On the other hand, a complex coding function may introduce additional nonlinearities and other mathematical difficulties which can hinder the search process substantially [79, p. 221-227][82, p. 97].

There is no general answer to the question of which one of the two approaches mentioned above to follow for a specific project, but many practical applications have shown that the best solutions could be found after imposing substantial modifications to the standard algorithms [86]. We think that most practitioners prefer natural, problem-related representations. Michalewicz [82, p. 4] offers:

> "It seems that a 'natural' representation of a potential solution for a given problem plus a family of applicable 'genetic' operators might be quite useful in the approximation of solutions of many problems, and this nature-modeled approach ... is a promising direction for problem solving in general."

Furthermore, many researchers also use hybrid algorithms, i.e. combinations of evolutionary search heuristics and traditional as well as knowledge-based search tech-

niques [22, p. 56] [91], [92].

It should be emphasized that all this becomes possible because the requirements for the application of evolutionary heuristics are so modest compared to most other search techniques. In our opinion, this is one of the most important strengths of the evolutionary approach and one of the reasons for the popularity evolutionary computation has gained throughout the last decade.

A. The Representation

Surprisingly, despite the fact that the representation problem, i.e., the choice or design of a well-suited genetic representation for the problem under consideration, has been described by many researchers [82], [93], [94] only few publications explicitly deal with this subject except for specialized research directions such as *genetic programming* [31], [95], [96] and the evolution of neural networks [97], [98].

Canonical genetic algorithms use a binary representation of individuals as fixed-length strings over the alphabet $\{0,1\}$ [11], such that they are well suited to handle pseudo-boolean optimization problems of the form

$$f : \{0,1\}^\ell \rightarrow I\!R \quad . \tag{4}$$

Sticking to the binary representation, genetic algorithms often enforce the utilization of encoding and decoding functions $h : M \rightarrow \{0,1\}^\ell$ and $h' : \{0,1\}^\ell \rightarrow M$ that facilitate mapping 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'. In case of continuous parameter optimization problems, for instance, genetic algorithms typically represent a real-valued vector $\vec{x} \in I\!R^n$ by a binary string $\vec{y} \in \{0,1\}^\ell$ as follows: the binary string is logically divided into n segments of equal length ℓ' (i.e., $\ell = n \cdot \ell'$), each segment is decoded to yield the corresponding integer value, and the integer value is in turn linearly mapped to the interval $[u_i, v_i] \subset I\!R$ (corresponding with the ith segment of the binary string) of real values [18].

The strong preference for using binary representations of solutions in genetic algorithms is derived from *schema theory* [11], which analyzes genetic algorithms in terms of their expected schema sampling behavior under the assumption that mutation and recombination are detrimental. The term *schema* denotes a similarity template that represents a subset of $\{0,1\}^\ell$, and the *schema theorem* of genetic algorithms offers that the canonical genetic algorithm provides a near-optimal sampling strategy (in terms of minimizing expected losses) for schemata by 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 [18] for a more detailed introduction to the schema theorem). The fundamental argument to justify the strong emphasis on binary alphabets is derived from the fact that the number of schemata is maximized for a given finite number of search points under a binary alphabet [18, pp. 40–41]. Consequently, the schema theory presently seems to favor binary representations of solutions (but see [99] for an alternative view, and [100] for a transfer of schema theory to S-expression representations used in genetic programming).

Practical experience, as well as some theoretical hints regarding the binary encoding of continuous object variables [101], [102], [103], [104], [105], however, indicate that the binary representation has some disadvantages. The coding function might introduce an additional multimodality, thus making the combined objective function $f = f' \circ h'$ (where $f' : M \rightarrow I\!R$) more complex than the original problem f' was. In fact, the schema theory relies on approximations [11, 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 criterion to maximize the best fitness value ever found [15]. In concluding this brief excursion into the theory of canonical genetic algorithms, we would like to emphasize the recent work by Vose [106], [107], [108], [109] and others [110], [111] on modeling genetic algorithms by Markov chain theory. This approach has already provided a remarkable insight into their convergence properties and dynamical behavior and led to the development of so-called *executable models* that facilitate the direct simulation of genetic algorithms by Markov chains for problems of sufficiently small dimension [112], [113].

In contrast to genetic algorithms, the representation in *evolution strategies* and *evolutionary programming* is directly based on real-valued vectors when dealing with continuous parameter optimization problems of the general form

$$f : M \subseteq I\!R^n \rightarrow I\!R \quad . \tag{5}$$

Both methods have originally been developed and are also used, however, for combinatorial optimization problems [43], [42], [55]. Moreover, since many real-world problems have complex search spaces which cannot be mapped "canonically" to one of the representations mentioned so far, lots of strategy variants, e.g. for integer [114], mixed-integer [115], structure optimization [116], [117] and others [82, chapter 10], have been introduced in the literature, but exhaustive comparative studies especially for non-standard representations are still missing. The actual development of the field is characterized by a progressing integration of the different approaches, such that the utilization of the common labels "genetic algorithm," "evolution strategy," and "evolutionary programming" might be sometimes even misleading.

B. Mutation

Of course, the design of variation operators has to obey the mathematical properties of the chosen representation, but there are still many degrees of freedom.

Mutation in genetic algorithms was introduced as a dedicated "background operator" of small importance (see [11], pp. 109–111). Mutation works by inverting bits with very small probability such as $p_m = 0.001$ [13], $p_m \in [0.005, 0.01]$ [118], or $p_m = 1/\ell$ [119], [120]. 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 [101], [121], and that even self-adaptive mutation rates are effective for pseudoboolean problems [122], [123], [124].

Originally, mutation in evolutionary programming was implemented as a random change (or multiple changes) 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. Currently, the most frequently used mutation scheme as applied to real-valued representations is very similar to that of evolution strategies.

In evolution strategies, the individuals consist of object variables $x_i \in I\!R$ $(1 \leq i \leq n)$ and so-called *strategy parameters* which are discussed in the next section. Mutation is then performed independently on each vector element by adding a normally distributed random value with expectation zero and standard deviation σ (the notation $N_i(\cdot,\cdot)$ indicates that the random variable is sampled anew for each value of the index i):

$$x_i' \quad = \quad x_i + \sigma \cdot N_i(0,1) \tag{6}$$

This raises the question of how to control the so-called step size σ of equation (6), which is discussed in the next section.

C. Self-adaptation

In [125] Schwefel introduced an endogenous mechanism for step size control by incorporating these parameters into the representation 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 simultaneously. This way, a suitable adjustment and diversity of mutation parameters should be provided under arbitrary circumstances.

More formally, an individual $\vec{a} = (\vec{x}, \vec{\sigma})$ consists of object variables $\vec{x} \in I\!R^n$ and strategy parameters $\vec{\sigma} \in I\!R_+^n$. The mutation operator works by adding a normally distributed random vector $\vec{z} \in I\!R^n$ with $z_i \sim N(0, \sigma_i^2)$ (i.e., the components of $\vec{z}$ are normally distributed with expectation zero and variance σ_i^2).

The effect of mutation is now defined as:

$$\sigma_i' \quad = \quad \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1)) \tag{7}$$

$$x_i' \quad = \quad x_i + \sigma_i' \cdot N_i(0,1) \quad , \tag{8}$$

where $\tau' \propto (\sqrt{2n})^{-1}$ and $\tau \propto (\sqrt{2\sqrt{n}})^{-1}$.

This mutation scheme, which is most frequently used in evolution strategies, is schematically depicted (for $n = 2$) in the middle of Figure 2. The locations of equal probability density for descendants are concentric hyperellipses (just one is depicted in Figure 2) around the parental midpoint. In the case considered here, i.e., up to n variances, but no covariances, the axes of the hyperellipses are congruent with the coordinate axes.

Two modifications of this scheme have to be mentioned: a simplified version uses just one step size parameter for all of the object variables. In this case the hyperellipses are reduced to hyperspheres, as depicted in the left part of Figure 2. A more elaborate *correlated mutation* scheme allows for the rotation of hyperellipses, as shown in the right part of Figure 2. This mechanism aims at a better adaptation to the topology of the objective function (for details see [79]).

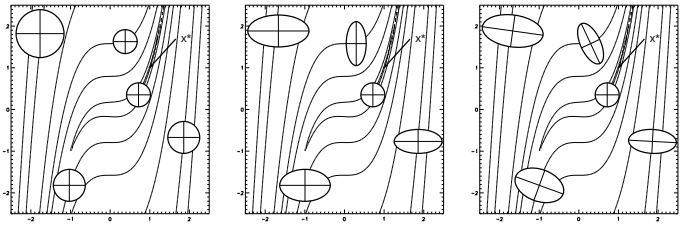


Fig. 2. Two-dimensional contour plot of the effect of the mutation operator in case of self-adaptation of a single step-size (left), of n step-sizes (middle) and of covariances (right). x^* denotes the optimizer. The ellipses represent one line of equal probability to place an offspring that is generated by mutation from the parent individual located at the center of the ellipses. Five sample individuals are shown in each of the plots.

The settings for the *learning rates* τ and τ' are recommended as upper bounds for the choice of these parameters (see [126], 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 the case of one self-adaptable step size, however, Beyer has recently theoretically shown that, for the sphere model (a quadratic bowl), the setting $\tau_0 \propto 1/\sqrt{n}$ is the optimal choice, maximizing the convergence velocity [127].

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 one step size) to the exploitation of a large *diversity* of strategy parameters. In case of correlated mutations, Rudolph [128] 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.

The choice of a logarithmic normal distribution for the modification of the standard deviations σ_i is presently also acknowledged in evolutionary programming literature [129], [130], [131]. Extensive empirical investigations indic-

ate some advantage of this scheme over the original additive self-adaptation mechanism introduced independently (but about 20 years later than in evolution strategies) in evolutionary programming [132] where

$$\sigma_i' = \sigma_i \cdot (1 + \alpha \cdot N(0,1)) \tag{9}$$

(with a setting of $\alpha \approx 0.2$ [131]). Recent preliminary investigations indicate, however, that this becomes reversed when noisy objective functions are considered, where the additive mechanism seems to outperform multiplicative modifications [133].

A study by Gehlhaar and Fogel [134] 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 appears important to mutate the standard deviations first and to use the mutated standard deviations for the modification of object variables. As the authors point out in that study, the reversed mechanism might suffer from generating offspring that have useful object variable vectors but poor strategy parameter vectors because these have not been used to determine the position of the offspring itself.

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 mechanisms. A recent theoretical study by Beyer presents a first step towards this goal [127]. In this work, the author shows that the self-adaptation principle works for a variety of different probability density functions for the modification of the step size, i.e., it is an extremely robust mechanism. Moreover, [127] clarifies that equation (9) is obtained from the corresponding equation for evolution strategies with one self-adaptable step size by Taylor expansion breaking off after the linear term, such that both methods behave equivalently for small settings of the learning rates τ and α, when $\tau = \alpha$. This prediction was confirmed perfectly by an experiment reported in [135].

Apart from the early work by Schaffer and Morishima [37], self-adaptation has only recently been introduced in genetic algorithms as a mechanism for evolving the parameters of variation operators. In [37], *punctuated crossover* was offered as a method for adapting both the number and position of crossover points for a multi-point crossover operator in canonical genetic algorithms. Although this approach seemed promising, the operator has not been used widely. A simpler approach towards self-adapting the crossover operator was presented by Spears [136], who allowed individuals to choose between 2-point crossover and uniform crossover by means of a self-adaptable operator choice bit attached to the representation of individuals. The results indicated that, in case of crossover operators, rather than adapting to the single best operator for a given problem, the mechanism seems to benefit from the existing diversity of operators available for crossover.

Concerning the mutation operator in genetic algorithms, some effort to facilitate self-adaptation of the mutation rate has been presented by Smith and Fogarty [123], based on earlier work by Bäck [137]. These approaches incorporate the mutation rate $p_m \in [0,1]$ into the representation of individuals and allow for mutation and recombination of the mutation rate in the same way as the vector of binary variables is evolved. The results reported in [123] demonstrate that the mechanism yields a significant improvement in performance of a canonical genetic algorithm on the test functions used.

D. Recombination

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 [138], uniform crossover (each bit is chosen randomly from the corresponding parental bits) [139], and others have been proposed, but similar to evolution strategies no generally useful recipe for the choice of a recombination operator can be given. The theoretical analysis of recombination is still to a large extent an open problem. Recent work on *multi-parent recombination*, where more than two individuals participate in generating a single offspring individual, clarifies that this generalization of recombination might yield a performance improvement in many application examples [140], [141], [142]. Unlike evolution strategies, where it is either utilized for the creation of all members of the intermediate population (the default case) or not at all, the recombination operator in genetic algorithms is typically applied with a certain probability p_c, and commonly proposed settings of the crossover probability are $p_c = 0.6$ [13] and $p_c \in [0.75, 0.95]$ [118].

In evolution strategies recombination is incorporated into the main loop of the algorithm 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. Normally, $\varrho = 2$ or $\varrho = \mu$ (so-called global recombination) are chosen. The recombination types for object variables and strategy parameters in evolution strategies often differ from each other, and typical examples are *discrete recombination* (random choices of single variables from parents, comparable to uniform crossover in genetic algorithms) and *intermediary recombination* (often arithmetic averaging, but other variants such as geometrical crossover [143] are also possible). For further details on these operators, see [79].

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 [79], 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 self-adaptable variances might even diverge on a sphere model for $n = 100$ [144]. Kursawe shows 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 number of strategy parameters incorporated into the individuals. Only recently, Rechenberg [46] and Beyer [142] presented first results concerning the convergence velocity analysis of global recombination in case of the sphere model. These results clarify that, for using one (rather than n as in Kursawe's experiment) optimally chosen standard deviation σ, a μ-fold speedup is achieved by both recombination variants. Beyer's interpretation of the results, however, is somewhat surprising because it does not put down the success of this operator on the existence of building blocks which are usefully rearranged in an offspring individual, but rather explains it as a *genetic repair* of the harmful parts of mutation.

Concerning evolutionary programming, a rash statement based on the common understanding of the contending structures as individuals would be to claim that evolutionary programming simply does not use recombination. Rather than focusing on the mechanism of sexual recombination, however, Fogel [145] argues that one may examine and simulate its functional effect and correspondingly interpret a string of symbols as a reproducing population or species, thus making recombination a non-issue (refer to [145] for philosophical reasons underlining this choice).

E. Selection

Unlike the variation operators which work on the genetic representation, the selection operator is based solely on the fitness values of the individuals.

In genetic algorithms selection is typically implemented as 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, so that *scaling functions* are often utilized to transform the fitness values accordingly (see e.g. [18], pp. 124). Rather than using absolute fitness values, *rank-based selection* methods utilize the indices of individuals when ordered according to fitness values to calculate the corresponding selection probabilities. Linear [146] as well as nonlinear [82, p. 60] mappings have been proposed for this type of selection operator. *Tournament selection* [147] works by taking a random uniform sample of a certain size $q > 1$ from the population, selecting the best of these q individuals to survive for the next generation, and repeating the process until the new population is filled. This method gains increasing popularity because it is easy to implement, computationally efficient, and allows for fine-tuning the selective pressure by increasing or decreasing the tournament size q. For an overview of selection methods and a characterization of their selective pressure in terms of numerical measures, the reader should consult [148], [149]. While most of these selection operators have been introduced in the framework of a generational genetic algorithm, they can also be used in combination with the steady-state and generation gap methods outlined in section III.

The (μ, λ)-evolution-strategy uses a deterministic selection scheme. The notation (μ, λ) indicates that μ parents create $\lambda > \mu$ offspring by means of recombination and mutation, and the best μ offspring individuals are deterministically selected to replace the parents (in this case, $Q = \emptyset$ in algorithm 1). 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 allowing 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 monotonic course of evolution is guaranteed ($Q = P(t)$ in algorithm 1). 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 or better than the (μ, λ)-strategy in many practical cases [134]. It should also be noted that both schemes can be interpreted as instances of the general (μ, κ, λ)-strategy, where $1 \leq \kappa \leq \infty$ denotes the maximum life span (in generations) of an individual. For $\kappa = 1$, the selection method yields the (μ, λ)-strategy, while it turns into the $(\mu + \lambda)$-strategy for $\kappa = \infty$ [54].

A minor difference between evolutionary programming and evolution strategies consists in the choice of a probabilistic variant of $(\mu + \lambda)$-selection in evolutionary programming, where each solution out of offspring and parent individuals is evaluated against $q > 1$ (typically, $q \leq 10$) other randomly chosen solutions from the union of parent and offspring individuals ($Q = P(t)$ in algorithm 1). 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 [79, pp. 96–99], this selection method is a probabilistic version of $(\mu + \lambda)$-selection which becomes more and more deterministic as the number q of competitors is increased. Whether or not a probabilistic selection scheme should be preferable over a deterministic scheme remains an open question.

Evolutionary algorithms can easily be ported to parallel computer architectures [150], [151]. Since the individuals can be modified and, most importantly, evaluated independently of each other, we should expect a speed-up scaling linear with the number of processing units p as long as p does not exceed the population size μ. But selection operates on the whole population so this operator eventually slows down the overall performance, especially for massively parallel architectures where $p \gg \mu$. This observation motivated the development of parallel algorithms using local selection within subpopulations like in *migration models* [152], [53] or within small neighborhoods of spatially arranged individuals like in *diffusion models* [153], [154], [155], [156] (also called *cellular evolutionary algorithms* [157], [158], [159]). It can be observed that local

selection techniques not only yield a considerable speed-up on parallel architectures but also improve the robustness of the algorithms [160], [116], [46].

F. Other Evolutionary Algorithm Variants

Although it is impossible to present a thorough overview of all variants of evolutionary computation here, it seems appropriate to explicitly mention *order-based genetic algorithms* [18], [82], *classifier systems* [161], [162], and *genetic programming* [163], [81], [31], [70] as branches of genetic algorithms that have developed into their own directions of research and application. The following overview is restricted to a brief statement of their domain of application and some literature references:

- *Order-based genetic algorithms* were proposed for searching the space of *permutations* $\pi : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$ directly rather than using complex decoding functions for mapping binary strings to permutations and preserving feasible permutations under mutation and crossover (as proposed in [164]). They apply 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 [82, chapter 10] for an overview).
- *Classifier systems* use an evolutionary algorithm to search the space of *production rules* (often encoded by strings over a ternary alphabet, but also sometimes using symbolic rules [165]) of a learning system capable of induction and generalization [18, chapter 6], [166], [161], [167]. Typically, the *Michigan* approach and the *Pittsburgh* approach are distinguished according to whether an individual corresponds with a single rule of the rule-based system (Michigan) or with a complete rule base (Pittsburgh).
- *Genetic programming* applies evolutionary search to the space of tree structures which may be interpreted as computer programs in a language suitable to modification by mutation and recombination. The dominant approach to genetic programming uses (a subset of) LISP programs (S-expressions) as genotype space [163], [31], but other programming languages including machine code are also used (see e.g. [168], [81], [70]).

Throughout this section we made the attempt to compare the constituents of evolutionary algorithms in terms of their canonical forms. But in practice the borders between these approaches are much more fluid. We can observe a steady evolution in this field by modifying (mutating), (re)combining, and validating (evaluating) the current approaches, permanently improving the population of evolutionary algorithms.

V. Applications

Practical application problems in fields as diverse as engineering, natural sciences, economics, and business (to mention only some of the most prominent representatives) often exhibit a number of characteristics that prevent the straightforward application of standard instances of evolutionary algorithms. Typical problems encountered when developing an evolutionary algorithm for a practical application include the following:

1. A suitable representation and corresponding operators need to be developed when the canonical representation is different from binary strings or real-valued vectors.
2. Various constraints need to be taken into account by means of a suitable method (ranging from penalty functions to repair algorithms, constraint-preserving operators, and decoders; see [169] for an overview).
3. Expert knowledge about the problem needs to be incorporated into the representation and the operators in order to guide the search process and increase its convergence velocity — without running into the trap, however, to get confused and misled by expert beliefs and habits which might not correspond with the best solutions.
4. An objective function needs to be developed, often in cooperation with experts from the particular application field.
5. The parameters of the evolutionary algorithm need to be set (or tuned), and the feasibility of the approach needs to be assessed by comparing the results to expert solutions (used so far) or, if applicable, solutions obtained by other algorithms.

Most of these topics require experience with evolutionary algorithms as well as cooperation between the application's expert and the evolutionary algorithm expert, and only few general results are available to guide the design of the algorithm (e.g., representation-independent recombination and mutation operators [170], [171], the requirement that small changes by mutation occur more frequently than large ones [172], [48], and a quantification of the selective pressure imposed by the most commonly used selection operators [149]). Nevertheless, evolutionary algorithms often yield excellent results when applied to complex optimization problems where other methods are either not applicable or turn out be unsatisfactory (a variety of examples can be found in [80]).

Important practical problem classes where evolutionary algorithms yield solutions of high quality include engineering design applications involving continuous parameters (e.g., for the design of aircraft [173], [174], structural mechanics problems based on two-dimensional shape representations [175], electromagnetic systems [176], and mobile manipulators [177], [178]), discrete parameters (e.g., for multiplierless digital filter optimization [179], the design of a linear collider [180], or nuclear reactor fuel arrangement optimization [181]), and mixed-integer representations (e.g., for the design of survivable networks [182] and optical multilayer systems [115]). Combinatorial optimization problems with a straightforward binary representation of solutions have also been treated successfully with canonical genetic algorithms and their derivatives (e.g., set partitioning and its application to airline crew scheduling [183], knapsack problems [184], [185], and others [186]). Relevant applications to combinatorial problems utilizing

a permutation representation of solutions are also found in the domains of scheduling (e.g., production scheduling [187] and related problems [188]), routing (e.g. of vehicles [189] or telephone calls [190]), and packing (e.g. of pallets on a truck [191]).

The existing range of successful applications is extremely broad, thus by far preventing an exhaustive overview — the list of fields and example applications should be taken as a hint for further reading rather than a representative overview. Some of the most challenging applications with a large profit potential are found in the field of biochemical drug design, where evolutionary algorithms have gained remarkable interest and success in the past few years as an optimization procedure to support protein engineering [134], [192], [193], [194]. Also finance and business provide a promising field of profitable applications [195], but of course few details are published about this work (see e.g. [196]). In fact, the relation between evolutionary algorithms and economics has found increasing interest in the past few years and is now widely seen as a promising modeling approach for agents acting in a complex, uncertain situation [197].

In concluding this section, we refer to the research field of *computational intelligence* (see section VI for details) and the applications of evolutionary computation to the other main fields of computational intelligence, namely fuzzy logic and neural networks. An overview of the utilization of genetic algorithms to train and construct neural networks is given in [198], and of course other variants of evolutionary algorithms can also be used for this task (see e.g. [199] for an evolutionary programming, [200] for an evolution strategy example, and [97], [201] for genetic algorithm examples). Similarly, both the rule base and membership functions of fuzzy systems can be optimized by evolutionary algorithms, typically yielding improvements of the performance of the fuzzy system (e.g. [202], [203], [204], [205], [206]). The interaction of computational intelligence techniques and hybridization with other methods such as expert systems and local optimization techniques certainly opens a new direction of research towards hybrid systems that exhibit problem solving capabilities approaching those of naturally intelligent systems in the future. Evolutionary algorithms, seen as a technique to evolve machine intelligence (see [5]), are one of the mandatory prerequisites for achieving this goal by means of algorithmic principles that are already working quite successfully in natural evolution [207].

VI. Summary and Outlook

To summarize, the current state of evolutionary computation research can be characterized as follows:

- The basic concepts have been developed more than 35 years ago, but it took almost two decades for their potential to be recognized by a larger audience.
- Application-oriented research in evolutionary computation is quite successful and almost dominates the field (if we consider the majority of papers). Only few potential application domains could be identified, if any, where evolutionary algorithms have not been tested so far. In many cases they have been used to produce good, if not superior, results.
- In contrast, the theoretical foundations are to some extent still weak. To say it more pithy: "We know that they work, but we do not know why." As a consequence, inexperienced users fall into the same traps repeatedly, since there are only few rules of thumb for the design and parameterization of evolutionary algorithms.

  A constructive approach for the synthesis of evolutionary algorithms, i.e. the choice or design of the representations, variation operators, and selection mechanisms is needed. But first investigations pointing in the direction of design principles for representation-independent operators are encouraging [171], as well, as is the work on complex non-standard representations such as in the field of genetic programming.
- Likewise, the field still lacks a sound formal characterization of the application domain and the limits of evolutionary computation. This requires future efforts in the field of complexity theory.

There exists a strong relationship between evolutionary computation and some other techniques, e.g. fuzzy logic and neural networks, usually regarded as elements of artificial intelligence. Following Bezdek [208], their main common characteristic lies in their numerical knowledge representation, which differentiates them from traditional symbolic artificial intelligence. Bezdek suggested the term *computational intelligence* for this special branch of artificial intelligence with the following characteristics[1]:

1. Numerical knowledge representation.
2. Adaptability.
3. Fault tolerance.
4. Processing speed comparable to human cognition processes.
5. Error rate optimality (e.g. with respect to a Bayesian estimate of the probability of a certain error on future data).

We regard computational intelligence as one of the most innovative research directions in connection with evolutionary computation, since we may expect that efficient, robust, and easy-to-use solutions to complex real-world problems will be developed on the basis of these complementary techniques. In this field, we expect an impetus from the interdisciplinary cooperation, e.g., techniques for tightly coupling evolutionary and problem domain heuristics, more elaborate techniques for self-adaptation, as well as an important step towards machine intelligence.

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

[1] The term "computational intelligence" was originally coined by Cercone and McCalla [209].

- Natural evolution works under dynamically changing environmental conditions, with nonstationary optima and even changing optimization criteria, and the individuals themselves are also changing the structure of the adaptive landscape during adaptation [210]. 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 as in the (μ, κ, λ)-strategy [54].
- The long-term goal of evolution consists of the maintenance of *evolvability* of a population [95], guaranteed by mutation and a preservation of diversity within the population (the term *meliorization* describes this more appropriately than optimization or adaptation does). 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 [211] 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* [50] as well as the idea of introducing two sexes with different selection criteria might be helpful for such problems [212], [213].
- Natural evolution neither assumes global knowledge (about all fitness values of all individuals) nor a generational synchronization, while many evolutionary algorithms still identify an iteration 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 [157], [158], [159] represent an attempt to overcome these restrictions.
- The *co-evolution* of species such as in predator-prey interactions implies that the adaptive landscape of individuals of one species changes as members of the other species make their adaptive moves [214]. Both the work on competitive fitness evaluation presented in [215] and the co-evolution of separate populations [216], [217] present successful approaches to incorporate the aspect of mutual interaction of different adaptive landscapes into evolutionary algorithms. As clarified by the work of Kauffman [214], however, we are just beginning to explore the dynamics of co-evolving systems and to exploit the principle for practical problem solving and evolutionary simulation.
- The genotype-phenotype mapping in nature, realized by the *genetic code* as well as the *epigenetic apparatus* (i.e., the biochemical processes facilitating the development and differentiation of an individual's cells into organs and systems), has evolved over time, while the mapping is usually fixed in evolutionary algorithms (dynamic parameter encoding as presented in [218] being a notable exception). An evolutionary self-adaptation of the genotype-phenotype mapping might be an interesting 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 [219]), 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.

With all this in mind, we are convinced that we are just beginning to understand and to exploit the full potential of evolutionary computation. Concerning basic research as well as practical applications to challenging industrial problems, evolutionary algorithms offer a wide range of promising further investigations, and it will be exciting to observe the future development of the field.

Acknowledgments

The authors would like to thank David B. Fogel and three anonymous reviewers for their very valuable and detailed comments that helped us improve the paper. We also appreciate the informal comments of another anonymous reviewer, and the efforts of the anonymous associate editor responsible for handling the paper submission and review procedure. The first author would also like to thank Christa Müller for her patience and love.

References

[1] H. J. Bremermann, "Optimization through evolution and recombination," in *Self-Organizing Systems*, M. C. Yovits et al., Ed. Spartan Books, Washington, D.C., 1962.

[2] R. M. Friedberg, "A learning machine: Part I," *IBM Journal*, vol. 2, no. 1, pp. 2–13, Jan. 1958.

[3] R. M. Friedberg, B. Dunham, and J. H. North, "A learning machine: Part II," *IBM Journal*, vol. 3, no. 7, pp. 282–287, July 1959.

[4] G. E. P. Box, "Evolutionary operation: A method for increasing industrial productivity," *Applied Statistics*, vol. VI, no. 2, pp. 81–101, 1957.

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

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

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

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

[9] L. J. Fogel, "Autonomous automata," *Industrial Research*, vol. 4, pp. 14–19, 1962.

[10] J. T. Alander, "Indexed bibliography of genetic algorithms papers of 1996," Report 94-1-96, University of Vaasa, Department of Information Techno logy and Production Economics, 1995, (`ftp.uwasa.fi`, `cs/report94-1`, `ga96bib.ps.Z`).

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

[12] J. H. Holland and J. S. Reitman, "Cognitive systems based on adaptive algorithms," in *Pattern-directed inference systems*, D. A. Waterman and F. Hayes-Roth, Eds. Academic Press, New York, NY, 1978.

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

[14] K. A. De Jong, "On using genetic algorithms to search program spaces," In Grefenstette [57], pp. 210–216.

[15] K. A. De Jong, "Are genetic algorithms function optimizers?," In Männer and Manderick [72], pp. 3–13.

[16] K. A. De Jong, "Genetic algorithms are NOT function optimizers," In Whitley [68], pp. 5–17.

[17] D. E. Goldberg, "Genetic algorithms and rule learning in dynamic system control," In Grefenstette [56], pp. 8–15.

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

[19] D. E. Goldberg, "The theory of virtual alphabets," In Schwefel and Männer [71], pp. 13–22.

[20] D. E. Goldberg, K. Deb, and J. H. Clark, "Genetic algorithms, noise, and the sizing of populations," *Complex Systems*, vol. 6, pp. 333–362, 1992.

[21] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik, "Rapid, accurate optimization of difficult problems using fast messy genetic algorithms," In Forrest [60], pp. 56–64.

[22] L. Davis, Ed., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

[23] L. J. Eshelman and J. D. Schaffer, "Crossover's niche," In Forrest [60], pp. 9–14.

[24] L. J. Eshelman and J. D. Schaffer, "Productive recombination and propagating and preserving schemata," In L. D. Whitley [69], pp. 299–313.

[25] S. Forrest and M. Mitchell, "What makes a problem hard for a genetic algorithm ? Some anomalous results and their explanation," *Machine Learning*, vol. 13, pp. 285–319, 1993.

[26] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC–16, no. 1, pp. 122–128, 1986.

[27] J. J. Grefenstette, "Incorporating problem specific knowledge into genetic algorithms," in *Genetic algorithms and simulated annealing*, L. Davis, Ed., pp. 42–60. Morgan Kaufmann Publishers, 1987.

[28] J. J. Grefenstette, "Conditions for implicit parallelism," In Rawlins [67], pp. 252–261.

[29] J. J. Grefenstette, "Deception considered harmful," In Whitley [68], pp. 75–91.

[30] J. R. Koza, "Hierarchical genetic algorithms operating on populations of computer programs," in *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, N. S. Sridharan, Ed. 1989, pp. 768–774, Morgan Kaufmann Publishers, San Mateo, CA.

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

[32] M. Mitchell, *An Introduction to Genetic Algorithms*, The MIT Press, Cambridge, MA, 1996.

[33] Rick L. Riolo, "The emergence of coupled sequences of classifiers," In Schaffer [58], pp. 256–264.

[34] Rick L. Riolo, "The emergence of default hierarchies in learning classifier systems," In Schaffer [58], pp. 322–327.

[35] J. D. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," In Grefenstette [56], pp. 93–100.

[36] J. D. Schaffer and L. J. Eshelman, "On crossover as an evolutionary viable strategy," In Belew and Booker [59], pp. 61–68.

[37] J. D. Schaffer and A. Morishima, "An adaptive crossover distribution mechanism for genetic algorithms," In Grefenstette [57], pp. 36–40.

[38] L. J. Fogel, *On the Organization of Intellect*, Ph.D. thesis, University of California, Los Angeles, CA, 1964.

[39] G. H. Burgin, "On playing two-person zero-sum games against nonminimax players," *IEEE TRansactions on Systems Science and Cybernetics*, vol. SSC-5, no. 4, pp. 369–370, Oct. 1969.

[40] G. H. Burgin, "Systems identification by quasilinearization and evolutionary programming," *Journal of Cybernetics*, vol. 3, no. 2, pp. 56–75, 1973.

[41] J. W. Atmar, *Speculation on the Evolution of intelligence and its possible realization in machine form*, Ph.D. thesis, New Mexico State University, 1976.

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

[43] D. B. Fogel, "An evolutionary approach to the traveling salesman problem," *Biological Cybernetics*, vol. 60, pp. 139–144, 1988.

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

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

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

[47] H.-P. Schwefel, *Evolutionsstrategie und numerische Optimierung*, Dissertation, Technische Universität Berlin, May 1975.

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

[49] M. Herdy, "Reproductive isolation as strategy parameter in hierarchically organized evolution strategies," In Männer and Manderick [72], pp. 207–217.

[50] F. Kursawe, "A variant of Evolution Strategies for vector optimization," In Schwefel and Männer [71], pp. 193–197.

[51] A. Ostermeier, "An evolution strategy with momentum adaptation of the random number distribution," In Männer and Manderick [72], pp. 197–206.

[52] A. Ostermeier, A. Gawelczyk, and N. Hansen, "Step-size adaptation based on non-local use of selection information," In Davidor et al. [73], pp. 189–198.

[53] G. Rudolph, "Global optimization by means of distributed evolution strategies," In Schwefel and Männer [71], pp. 209–213.

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

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

[56] J. J. Grefenstette, Ed., *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.

[57] J. J. Grefenstette, Ed., *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[58] J. D. Schaffer, Ed., *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[59] R. K. Belew and L. B. Booker, Eds., *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[60] S. Forrest, Ed., *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[61] L. Eshelman, Ed., *Genetic Algorithms: Proceedings of the 6th International Conference*. Morgan Kaufmann Publishers, San Francisco, CA, 1995.

[62] D. B. Fogel and W. Atmar, Eds., *Proceedings of the First Annual Conference on Evolutionary Programming*. Evolutionary Programming Society, San Diego, CA, 1992.

[63] D. B. Fogel and W. Atmar, Eds., *Proceedings of the Second Annual Conference on Evolutionary Programming*. Evolutionary Programming Society, San Diego, CA, 1993.

[64] A. V. Sebald and L. J. Fogel, Eds., *Proceedings of the Third Annual Conference on Evolutionary Programming*. World Scientific, Singapore, 1994.

[65] J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, Eds., *Proceedings of the Fourth Annual Conference on Evolutionary Programming*. The MIT Press, Cambridge, MA, 1995.

[66] L. J. Fogel, P. J. Angeline, and T. Bäck, Eds., *Proceedings of the Fifth Annual Conference on Evolutionary Programming*. The MIT Press, Cambridge, MA, 1996.

[67] G. J. E. Rawlins, Ed., *Foundations of Genetic Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[68] L. D. Whitley, Ed., *Foundations of Genetic Algorithms 2*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[69] M. D. Vose L. D. Whitley, Ed., *Foundations of Genetic Algorithms 3*. Morgan Kaufmann Publishers, San Francisco, CA, 1995.

[70] J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds., *Genetic Programming 1996. Proceedings of the First Annual Conference*. The MIT Press, Cambridge, MA, 1996.

[71] H.-P. Schwefel and R. Männer, Eds., *Parallel Problem Solving from Nature — Proceedings 1st Workshop PPSN I*, vol. 496 of *Lecture Notes in Computer Science*. Springer, Berlin, 1991.

[72] R. Männer and B. Manderick, Eds., *Parallel Problem Solving from Nature 2*. Elsevier, Amsterdam, 1992.

[73] Y. Davidor, H.-P. Schwefel, and R. Männer, Eds., *Parallel Problem Solving from Nature — PPSN III, International Conference on Evolutionary Computation*, vol. 866 of *Lecture Notes in Computer Science*. Springer, Berlin, 1994.

[74] H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, Eds., *Parallel Problem Solving from Nature IV. Proceedings of the International Conference on Evolutionary Computation*, vol. 1141 of *Lecture Notes in Computer Science*. Springer, Berlin, 1996.

[75] *Proceedings of the First IEEE Conference on Evolutionary Computation, Orlando, FL*. IEEE Press, Piscataway, NJ, 1994.

[76] *Proceedings of the Second IEEE Conference on Evolutionary Computation, Perth, Australia*. IEEE Press, Piscataway, NJ, 1995.

[77] *Proceedings of the Third IEEE Conference on Evolutionary Computation, Nagoya, Japan*. IEEE Press, Piscataway, NJ, 1996.

[78] *Proceedings of the Fourth IEEE Conference on Evolutionary Computation, Indianapolis, IN*. IEEE Press, Piscataway, NJ, 1997.

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

[80] Th. Bäck, D. B. Fogel, and Z. Michalewicz, Eds., *Handbook of Evolutionary Computation*, Oxford University Press, New York, and Institute of Physics Publishing, New York, 1997.

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

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

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

[84] Th. Bäck, U. Hammel, M. Schütz, H.-P. Schwefel, and J. Sprave, "Applications of evolutionary algorithms at the center for applied systems analysis," in *Computational Methods in Applied Sciences '96*, J.-A. Désidéri, C. Hirsch, P. Le Tallec, E. Oñate, M. Pandolfi, J. Périaux, and E. Stein, Eds. 1996, pp. 243–250, Wiley, Chichester.

[85] H.-P. Schwefel, "Direct search for optimal parameters within simulation models," in *Proc. 12th Annual Simulation Symposium*, Tampa, Florida/USA, Mar. 1979, pp. 91–102.

[86] Z. Michalewicz, "A hierarchy of evolution programs: An experimental study," *Evolutionary Computation*, vol. 1, no. 1, pp. 51–76, 1993.

[87] W. Atmar, "Notes on the simulation of evolution," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 130–148, 1994.

[88] L. D. Whitley, "The GENITOR algorithm and selection pressure: Why rank–based allocation of reproductive trials is best," In Schaffer [58], pp. 116–121.

[89] L. D. Whitley and J. Kauth, "GENITOR: a different genetic algorithm," in *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*. 1988, pp. 118–130, Denver, CO.

[90] K. A. De Jong and J. Sarma, "Generation gaps revisited," In Whitley [68], pp. 19–28.

[91] D. J. Powell, M. M. Skolnick, and S. S. Tong, "Interdigitation: A hybrid technique for engineering design optimization employing genetic algorithms, expert systems, and numerical optimization," In Davis [22], chapter 20, pp. 312–321.

[92] J.-M. Renders and S. P. Flasse, "Hybrid methods using genetic algorithms for global optimization," *IEEE Transactions on Systems, Man, and Cybernetics (Part B)*, vol. 26, no. 2, pp. 243–258, 1996.

[93] K. A. De Jong, "Evolutionary computation: Recent developments and open issues," in *First Int. Conf. on Evolutionary Computation and Its Applications*, E. D. Goodman, B. Punch, and V. Uskov, Eds. 1996, pp. 7–17, Presidium of the Russian Academy of Science, Moskau.

[94] M. Mitchell and S. Forrest, "Genetic algorithms and artificial life," *Artificial Life*, vol. 1, no. 3, pp. 267–289, 1995.

[95] L. Altenberg, "The evolution of evolvability in genetic programming," In Kinnear [81], pp. 47–74.

[96] R. Keller and W. Banzhaf, "Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes," in *Genetic Programming 1996: Proc. First Ann. Conf.*, John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, Eds., 1996.

[97] F. Gruau, "Genetic synthesis of modular neural networks," In Forrest [60], pp. 318–325.

[98] M. Mandischer, "Representation and evolution of neural networks," in *Artificial Neural Nets and Genetic Algorithms*, R. F. Albrecht, C. R. Reeves, and N. C. Steele, Eds. 1993, pp. 643–649, Springer, Wien.

[99] H. J. Antonisse, "A new interpretation of schema notation that overturns the binary encoding constraint," In Schaffer [58], pp. 86–91.

[100] U.-M. O'Reilly and F. Oppacher, "The troubling aspects ofa building block hypothesis for genetic programming," In L. D. Whitley [69], pp. 73–88.

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

[102] L. J. Eshelman and J. D. Schaffer, "Real-coded genetic algorithms and interval-schemata," In Whitley [68], pp. 187–202.

[103] C. Z. Janikow and Z. Michalewicz, "An experimental comparison of binary and floating point representations in genetic algorithms," In Belew and Booker [59], pp. 31–36.

[104] N. J. Radcliffe, "Equivalence class analysis of genetic algorithms," *Complex Systems*, vol. 5, no. 2, pp. 183–206, 1991.

[105] A. H. Wright, "Genetic algorithms for real parameter optimization," In Rawlins [67], pp. 205–218.

[106] A. Nix and M. D. Vose, "Modeling genetic algorithms with markov chains," *Annals of Mathematics and Artificial Intelligence*, vol. 5, pp. 79–88, 1992.

[107] M. D. Vose, "Modeling simple genetic algorithms," In Whitley [68], pp. 63–73.

[108] M. D. Vose and A. H. Wright, "Simple genetic algorithms with linear fitness," *Evolutionary Computation*, vol. 2, no. 4, pp. 347–368, 1994.

[109] M. D. Vose, "Modeling simple genetic algorithms," *Evolutionary Computation*, vol. 3, no. 4, pp. 453–472, 1995.

[110] G. Rudolph, "Convergence analysis of canonical genetic algorithms," *IEEE Transactions on Neural Networks, Special Issue on Evolutionary Computation*, vol. 5, no. 1, pp. 96–101, 1994.

[111] J. Suzuki, "A Markov chain analysis on simple genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 25, no. 4, pp. 655–659, Apr. 1995.

[112] K. A. De Jong, W. M. Spears, and D. F. Gordon, "Using Markov chains to analyze GAFOs," In L. D. Whitley [69], pp. 115–137.

[113] L. D. Whitley, "An executable model of a simple genetic algorithm," In L. D. Whitley [69], pp. 45–62.

[114] G. Rudolph, "An evolutionary algorithm for integer programming," In Davidor et al. [73], pp. 139–148.

[115] M. Schütz and J. Sprave, "Application of parallel mixed-integer evolution strategies with mutation rate pooling," In Fogel et al. [66], pp. 345–354.

[116] B. Groß, U. Hammel, A. Meyer, P. Maldaner, P. Roosen, and M. Schütz, "Optimization of heat exchanger networks by means of evolution strategies," In Voigt et al. [74], pp. 1002–1011.

[117] R. Lohmann, "Structure evolution in neural systems," in *Dynamic, Genetic, and Chaotic Programming*, B. Soucek and the IRIS Group, Eds., pp. 395–411. Wiley, New York, 1992.

[118] 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 Schaffer [58], pp. 51–60.

[119] H. J. Bremermann, M. Rogson, and S. Salaff, "Global properties of evolution processes," in *Natural Automata and Useful Simulations*, H. H. Pattec, E. A. Edelsack, L. Fein, and A. B. Callahan, Eds., chapter 1, pp. 3–41. Spartan Books, Washington, DC, 1966.

[120] H. Mühlenbein, "How genetic algorithms really work: I. mutation and hillclimbing," In Männer and Manderick [72], pp. 15–25.

[121] T. C. Fogarty, "Varying the probability of mutation in the genetic algorithm," In Schaffer [58], pp. 104–109.

[122] Th. Bäck and M. Schütz, "Intelligent mutation rate control in canonical genetic algorithms," in *Foundations of Intelligent Systems, 9th International Symposium, ISMIS '96*, Z. W. Ras and M. Michalewicz, Eds. 1996, vol. 1079 of *Lecture Notes in Artificial Intelligence*, pp. 158–167, Springer, Berlin.

[123] J. Smith and T. C. Fogarty, "Self adaptation of mutation rates in a steady state genetic algorithm," in *Proceedings of the Third IEEE Conference on Evolutionary Computation*. 1996, pp. 318–323, IEEE Press, Piscataway, NJ.

[124] M. Yanagiya, "A simple mutation-dependent genetic algorithm," In Forrest [60], p. 659.

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

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

[127] H.-G. Beyer, "Toward a theory of evolution strategies: Self-adaptation," *Evolutionary Computation*, vol. 3, no. 3, pp. 311–348, 1995.

[128] G. Rudolph, "On correlated mutations in evolution strategies," In Männer and Manderick [72], pp. 105–114.

[129] N. Saravanan and D. B. Fogel, "Evolving neurocontrollers using evolutionary programming," in *Proceedings of the First IEEE Conference on Evolutionary Computation*. 1994, vol. 1, pp. 217–222, IEEE Press, Piscataway, NJ.

[130] N. Saravanan and D. B. Fogel, "Learning of strategy parameters in evolutionary programming: An empirical study," In Sebald and Fogel [64], pp. 269–280.

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

[132] D. B. Fogel, L. J. Fogel, and W. Atmar, "Meta-evolutionary programming," in *Proc. 25th Asilomar Conference on Signals, Systems and Computers*, R. R. Chen, Ed. 1991, pp. 540–545, Pacific Grove, CA.

[133] P. J. Angeline, "The effects of noise on self-adaptive evolutionary optimization," In Fogel et al. [66], pp. 433–440.

[134] D. K. Gehlhaar and D. B. Fogel, "Tuning evolutionary programming for conformationally flexible molecular docking," In Fogel et al. [66], pp. 419–429.

[135] Th. Bäck and H.-P. Schwefel, "Evolutionary computation: An overview," in *Proceedings of the Third IEEE Conference on Evolutionary Computation*. 1996, pp. 20–29, IEEE Press, Piscataway, NJ.

[136] W. M. Spears, "Adapting crossover in evolutionary algorithms," In McDonnell et al. [65], pp. 367–384.

[137] Th. Bäck, "Self-Adaptation in Genetic Algorithms," in *Proceedings of the First European Conference on Artificial Life*, F. J. Varela and P. Bourgine, Eds. 1992, pp. 263–271, The MIT Press, Cambridge, MA.

[138] L. J. Eshelman, R. A. Caruna, and J. D. Schaffer, "Biases in the crossover landscape," In Schaffer [58], pp. 10–19.

[139] G. Syswerda, "Uniform crossover in genetic algorithms," In Schaffer [58], pp. 2–9.

[140] A. E. Eiben, P.-E. Raué, and Zs. Ruttkay, "Genetic algorithms with multi-parent recombination," In Davidor et al. [73], pp. 78–87.

[141] A. E. Eiben, C. H. M. van Kemenade, and J. N. Kok, "Orgy in the computer: Multi-parent reproduction in genetic algorithms," in *Advances in Artificial Life. Third International Conference on Artificial Life*, F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, Eds. 1995, vol. 929 of *Lecture Notes in Artificial Intelligence*, pp. 934–945, Springer, Berlin.

[142] H.-G. Beyer, "Toward a theory of evolution strategies: On the benefits of sex — the $(\mu/\mu, \lambda)$-theory," *Evolutionary Computation*, vol. 3, no. 1, pp. 81–111, 1995.

[143] Z. Michalewicz, G. Nazhiyath, and M. Michalewicz, "A note on usefulness of geometrical crossover for numerical optimization problems," In Fogel et al. [66], pp. 305–312.

[144] F. Kursawe, "Towards self-adapting evolution strategies," In *Proceedings of the Second IEEE Conference on Evolutionary Computation, Perth, Australia* [76], pp. 283–288.

[145] D. B. Fogel, "On the philosophical differences between evolutionary algorithms and genetic algorithms," In Fogel and Atmar [63], pp. 23–29.

[146] J. E. Baker, "Adaptive selection methods for genetic algorithms," In Grefenstette [56], pp. 101–111.

[147] D. E. Goldberg, B. Korb, and K. Deb, "Messy genetic algorithms: Motivation, analysis, and first results," *Complex Systems*, vol. 3, no. 5, pp. 493–530, Oct. 1989.

[148] Th. Bäck, "Selective pressure in evolutionary algorithms: A characterization of selection mechanisms," in *Proceedings of the First IEEE Conference on Evolutionary Computation*. 1994, pp. 57–62, IEEE Press, Piscataway, NJ.

[149] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," In Rawlins [67], pp. 69–93.

[150] M. Dorigo and V. Maniezzo, "Parallel genetic algorithms: Introduction and overview of current research," in *Parallel Genetic Algorithms: Theory & Applications*, J. Stender, Ed., Frontiers in Artificial Intelligence and Applications, pp. 5–42. IOS Press, Amsterdam, 1993.

[151] F. Hoffmeister, "Scalable parallelism by evolutionary algorithms," in *Parallel Computing and Mathematical Optimization*, M. Grauer and D. B. Pressmar, Eds. 1991, vol. 367 of *Lecture Notes in Economics and Mathematical Systems*, pp. 177–198, Springer, Berlin.

[152] M. Munetomo, Y. Takai, and Y. Sato, "An efficient migration scheme for subpopulation-based asynchronously parallel genetic algorithms," In Forrest [60], p. 649.

[153] S. Baluja, "Structure and performance of fine-grain parallelism in genetic search," In Forrest [60], pp. 155–162.

[154] R. J. Collins and D. R. Jefferson, "Selection in massively parallel genetic algorithms," In Belew and Booker [59], pp. 249–256.

[155] M. Gorges-Schleuter, "ASPARAGOS: an asynchronous parallel genetic optimization strategy," In Schaffer [58], pp. 422–427.

[156] P. Spiessens and B. Manderick, "Fine-grained parallel genetic algorithms," In Schaffer [58], pp. 428–433.

[157] V. S. Gordon, K. Mathias, and L. D. Whitley, "Cellular genetic algorithms as function optimizers: Locality effects," in *Proceedings of the 1994 ACM Symposium on Applied Computing*, E. Deaton, D. Oppenheim, J. Urban, and H. Berghel, Eds. 1994, pp. 237–241, ACM Press, New York.

[158] 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*, pp. 365–372. IEE, London, 1995.

[159] L. D. Whitley, "Cellular genetic algorithms," In Forrest [60], p. 658.

[160] M. Gorges-Schleuter, "Comparison of local mating strategies in massively parallel genetic algorithms," In Männer and Manderick [72], pp. 553–562.

[161] 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.

[162] R. Serra and G. Zanarini, *Complex systems and cognitive processes*, Springer, Berlin, 1990.

[163] M. L. Cramer, "A representation for the adaptive generation of simple sequential programs," In Grefenstette [56], pp. 183–187.

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

[165] D. A. Gordon and J. J. Grefenstette, "Explanations of empirically derived reactive plans," in *Proceedings of the Seventh International Conference on Machine Learning*. June 1990, pp. 198–203, Morgan Kaufmann Publishers.

[166] L. B. Booker, D. E. Goldberg, and J. H. Holland, "Classifier systems and genetic algorithms," in *Machine Learning:*

Paradigms and Methods, J. G. Carbonell, Ed., pp. 235–282. The MIT Press / Elsevier, 1989.

[167] S. W. Wilson, "ZCS: A zeroth level classifier system," *Evolutionary Computation*, vol. 2, no. 1, pp. 1–18, 1994.

[168] F. D. Francone, P. Nordin, and W. Banzhaf, "Benchmarking the generalization capabilities of a compiling genetic programming system using sparse data sets," In Koza et al. [70], pp. 72–80.

[169] Z. Michalewicz and M. Schoenauer, "Evolutionary algorithms for constrained parameter optimization problems," *Evolutionary Computation*, vol. 4, no. 1, pp. 1–32, 1996.

[170] N. J. Radcliffe, "The algebra of genetic algorithms," *Annals of Maths and Artificial Intelligence*, vol. 10, pp. 339–384, 1994.

[171] P. D. Surry and N. J. Radcliffe, "Formal algorithms + formal representations = search strategies," In Voigt et al. [74], pp. 366–375.

[172] N. J. Radcliffe and P. D. Surry, "Fitness variance of formae and performance prediction," In L. D. Whitley [69], pp. 51–72.

[173] M. F. Bramlette and E. E. Bouchard, "Genetic algorithms in parametric design of aircraft," In Davis [22], chapter 10, pp. 109–123.

[174] J. Périaux, M. Sefrioui, B. Stoufflet, B. Mantel, and E. Laporte, "Robust genetic algorithms for optimization problems in aerodynamic design," in *Genetic Algorithms in Engineering and Computer Science*, G. Winter, J. Périaux, M. Galán, and P. Cuesta, Eds., chapter 19, pp. 371–396. Wiley, Chichester, 1995.

[175] M. Schoenauer, "Shape representations for evolutionary optimization and identification in structural mechanics," in *Genetic Algorithms in Engineering and Computer Science*, G. Winter, J. Périaux, M. Galán, and P. Cuesta, Eds., chapter 22, pp. 443–463. Wiley, Chichester, 1995.

[176] E. Michielssen and D. S. Weile, "Electromagnetic system design using genetic algorithms," in *Genetic Algorithms in Engineering and Computer Science*, G. Winter, J. Périaux, M. Galán, and P. Cuesta, Eds., chapter 18, pp. 345–369. Wiley, Chichester, 1995.

[177] B. Anderson, J. McDonnell, and W. Page, "Configuration optimization of mobile manipulators with equality constraints using evolutionary programming," In Fogel and Atmar [62], pp. 71–79.

[178] J. R. McDonnell, B. L. Anderson, W. C. Page, and F. G. Pin, "Mobile manipulator configuration optimization using evolutionary programming," In Fogel and Atmar [62], pp. 52–62.

[179] J. D. Schaffer and L. J. Eshelman, "Designing multiplierless digital filters using genetic algorithms," In Forrest [60], pp. 439–444.

[180] H.-G. Beyer, "Some aspects of the 'evolution strategy' for solving TSP-like optimization problems appearing at the design studies of a 0.5 TeV e^+e^--linear collider," In Männer and Manderick [72], pp. 361–370.

[181] 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*. 1996, pp. 104–108, IEEE Press, Piscataway, NJ.

[182] L. Davis, D. Orvosh, A. Cox, and Y. Qiu, "A genetic algorithm for survivable network design," In Forrest [60], pp. 408–415.

[183] D. M. Levine, "A genetic algorithm for the set partitioning problem," In Forrest [60], pp. 481–487.

[184] V. S. Gordon, A. P. W. Böhm, and L. D. Whitley, "A note on the performance of genetic algorithms on zero-one knapsack problems," in *Proceedings of the 1994 ACM Symposium on Applied Computing*, E. Deaton, D. Oppenheim, J. Urban, and H. Berghel, Eds. 1994, pp. 194–195, ACM Press, New York.

[185] A. Olsen, "Penalty functions and the knapsack problem," in *Proceedings of the First IEEE Conference on Evolutionary Computation*. 1994, pp. 554–558, IEEE Press.

[186] S. Khuri, Th. Bäck, and J. Heitkötter, "An evolutionary approach to combinatorial optimization problems," in *Proceedings of the 22nd Annual ACM Computer Science Conference*, D. Cizmar, Ed. 1994, pp. 66–73, ACM Press, New York.

[187] R. Bruns, "Direct chromosome representation and advanced genetic operators for production scheduling," In Forrest [60], pp. 352–359.

[188] H.-L. Fang, P. Ross, and D. Corne, "A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems," In Forrest [60], pp. 375–382.

[189] J. L. Blanton and R. L. Wainwright, "Multiple vehicle routing with time and capacity constraints using genetic algorithms," In Forrest [60], pp. 452–459.

[190] L. A. Cox, L. Davis, and Y. Qiu, "Dynamic anticipatory routing in circuit-switched telecommunications networks," In Davis [22], chapter 11, pp. 109–143.

[191] K. Juliff, "A multi-chromosome genetic algorithm for pallet loading," In Forrest [60], pp. 467–473.

[192] S. Schulze-Kremer, "Genetic algorithms for protein ternary structure prediction," in *Parallel Genetic Algorithms: Theory & Applications*, J. Stender, Ed., Frontiers in Artificial Intelligence and Applications, pp. 129–150. IOS Press, Amsterdam, 1993.

[193] R. Unger and J. Moult, "A genetic algorithm for 3D protein folding simulation," In Forrest [60], pp. 581–588.

[194] D. C. Youvan, A. P. Arkin, and M. M. Yang, "Recursive ensemble mutagenesis: A combinatorial optimization technique for protein engineering," In Männer and Manderick [72], pp. 401–410.

[195] R. F. Walker, E. W. Haasdijk, and M. C. Gerrets, "Credit evaluation using a genetic algorithm," In Goonatilake and Treleaven [196], chapter 3, pp. 39–59.

[196] S. Goonatilake and P. Treleaven, Eds., *Intelligent Systems for Finance and Business*, Wiley, Chichester, 1995.

[197] P. G. Harrald, "Evolutionary algorithms and economic models: A view," In Fogel et al. [66], pp. 3–7.

[198] L. D. Whitley, "Genetic algorithms and neural networks," in *Genetic Algorithms in Engineering and Computer Science*, G. Winter, J. Périaux, M. Galán, and P. Cuesta, Eds., chapter 11, pp. 203–216. Wiley, Chichester, 1995.

[199] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 54–65, 1994.

[200] W. Wienholt, "Minimizing the system error in feedforward neural networks with evolution strategy," in *Proceedings of the International Conference on Artificial Neural Networks*, S. Gielen and B. Kappen, Eds. 1993, pp. 490–493, Springer, London.

[201] M. Mandischer, "Genetic optimization and representation of neural networks," in *Proceedings of the 4th Australian Conference on Neural Networks*, P. Leong and M. Jabri, Eds. Sidney University, Department of Electrical Engineering, 1993, pp. 122–125.

[202] A. Homaifar and E. McCormick, "Full design of fuzzy controllers using genetic algorithms," in *Neural and Stochastic Methods in Image and Signal Processing*, Su-Shing Chen, Ed. 1992, vol. SPIE-1766, pp. 393–404, The International Society for Optical Engineering.

[203] C. L. Karr, "Genetic algorithms for fuzzy controllers," *AI Expert*, vol. 6, no. 2, pp. 27–33, 1991.

[204] S. B. Haffner and A. V. Sebald, "Computer-aided design of fuzzy HVAC controllers using evolutionary programming," In Fogel and Atmar [63], pp. 98–107.

[205] P. Thrift, "Fuzzy logic synthesis with genetic algorithms," In Belew and Booker [59], pp. 514–518.

[206] P. Wang and D. P. Kwok, "Optimal fuzzy PID control based on genetic algorithm," in *Proceedings of the 1992 International Conference on Industrial Electronics, Control, and Instrumentation*. 1992, vol. 2, pp. 977–981, IEEE Press.

[207] D. C. Dennett, *Darwin's Dangerous Idea*, Touchstone, New York, 1995.

[208] J. C. Bezdek, "What is computational intelligence ?," in *Computational Intelligence: Imitating Life*, Jacek M. Zurada, Robert J. Marks II, and Charles J. Robinson, Eds. 1994, pp. 1–12, IEEE Press, New York.

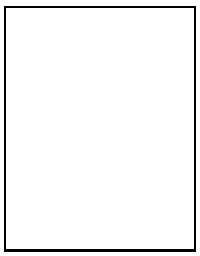
[209] N. Cercone and G. McCalla, "Ten years of computational intelligence," *Computational Intelligence*, vol. 10, no. 4, pp. i–vi, 1994.

[210] J. Schull, "The view from the adaptive landscape," In Schwefel and Männer [71], pp. 415–427.

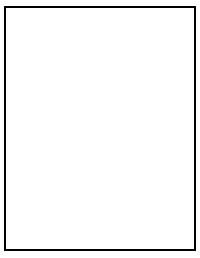
[211] C. M. Fonseca and P. J. Fleming, "An overview of evolutionary algorithms in multiobjective optimization," *Evolutionary Computation*, vol. 3, no. 1, pp. 1–16, 1995.

[212] J. Lis and A. E. Eiben, "Multi-sexual genetic algorithm for multiobjective optimization," In *Proceedings of the Fourth IEEE Conference on Evolutionary Computation, Indianapolis, IN* [78].

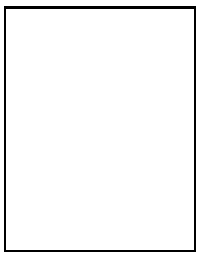
[213] E. Ronald, "When selection meets seduction," In Eshelman [61], pp. 167–173.
[214] S. A. Kauffman, *The Origins of Order. Self-Organization and Selection in Evolution*, Oxford University Press, New York, NY, 1993.
[215] P. J. Angeline and J. B. Pollack, "Competitive environments evolve better solutions for complex tasks," In Forrest [60], pp. 264–270.
[216] W. D. Hillis, "Co-evolving parasites improve simulated evolution as an optimization procedure," In Forrest [220], pp. 228–234.
[217] J. Paredis, "Coevolutionary life-time learning," In Voigt et al. [74], pp. 72–80.
[218] N. N. Schraudolph and R. K. Belew, "Dynamic parameter encoding for genetic algorithms," *Machine Learning*, vol. 9, pp. 9–21, 1992.
[219] R. W. Anderson, "Genetic mechanisms underlying the Baldwin effect are evident in natural antibodies," In McDonnell et al. [65], pp. 547–564.
[220] S. Forrest, Ed., *Emergent Computation. Self-Organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks*, Cambridge, MA, 1990. MIT Press.



Thomas Bäck received the Diploma degree in Computer Science in 1990 and the Ph.D. degree in Computer Science in 1994, both from the University of Dortmund, Germany. In 1995, he received the best dissertation award of the German Association for Computer Science (GI) for his Ph.D. thesis on evolutionary algorithms.
From 1990-1994, he worked as a Scientific Assistant at the Department of Computer Science of the University of Dortmund. Since 1994, he is Senior Research Fellow at the Center for Applied Systems Analysis within the Informatik Centrum Dortmund, and Managing Director of the Center for Applied Systems Analysis since 1996. He also serves as an Associate Professor in the Computer Science Department of Leiden University, The Netherlands, and teaches courses on evolutionary computation at the University of Dortmund and at Leiden University. His current research interests are in the areas of theory and application of evolutionary computation and related areas of computational intelligence. He is author of the book *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms* (New York, NY: Oxford University Press, 1996), co-editor-in-chief of the *Handbook of Evolutionary Computation* (New York, NY: Oxford University Press & Institute of Physics Publishing, 1997), member of the editorial board of *Evolutionary Computation* (Cambridge, MA: The MIT Press), and associate editor of the IEEE Transactions on Evolutionary Computation.
Dr. Bäck is a member of the IEEE and the Dutch Association for Theoretical Computer Science (NVTI), serves on the IEEE Neural Networks Council's technical committee on evolutionary computation since 1995, was a co-program chair of the 1996 and 1997 IEEE International Conferences on Evolutionary Computation (ICEC) and the Fifth Annual Conference on Evolutionary Programming (EP '96), and is program chair of the Seventh International Conference on Genetic Algorithms and Their Applications (ICGA '97).



Ulrich Hammel received the Diploma degree in Computer Science in 1985 from the University of Dortmund, Germany.
Since 1985, he works as a Scientific Assistant at the Department of Computer Science of the University of Dortmund, and is Managing Director of the Sonderforschungsbereich ("Collaborative Research Center") *Design and management of complex technical processes and systems by computational intelligence methods* that started in 1997, involving partners from the Departments of Computer Science, Electrical Engineering, Mechanical Engineering, and Chemical Engineering at the University of Dortmund. His research interests are in the intersection of modeling and optimization. Currently, he works on genetic representations and robust design strategies in evolutionary computation.



Hans-Paul Schwefel received the Diploma degree in Engineering (Aero- and Space-Technology) in 1965 and the Ph.D. degree in Process Engineering in 1975, both from the Technical University of Berlin, Germany.
From 1963-1966, he worked as a Junior Assistant and Research Assistant at the Hermann-Föttinger Institute for Hydrodynamics, Technical University of Berlin. Subsequently (1967-1970), he was an R&D Engineer at the AEG Research Centre Berlin. From 1970-1976, he was a Research Consultant and DFG Grantee for various research projects concerning the development of evolution strategies at the Technical University of Berlin and the Medical School of the University of Hanover. Dr.-Ing. Schwefel was a Senior Research Fellow at the Nuclear Research Centre (KFA) Jülich and a Group Leader within the Programme Group of Systems Analysis and Technological Development during 1976-1984, and he holds the Chair of Systems Analysis as a Full Professor at the Department of Computer Science of the University of Dortmund since 1985. He teaches courses on systems analysis, programming, evolutionary computation, and self-organization. He is President of the Informatik Centrum Dortmund since 1989, and Speaker of the Sonderforschungsbereich ("Collaborative Research Center") *Design and management of complex technical processes and systems by computational intelligence methods* since 1996.
Dr.-Ing. Schwefel is author of the books *Numerical Optimization of Computer Models* (Chichester, UK: Wiley, 1981) and *Evolution and Optimum Seeking* (New York, NY: Wiley, 1995), member of the editorial boards of *Evolutionary Computation* (Cambridge, MA: The MIT Press) and *BioSystems* (Amsterdam: Elsevier), and associate editor of the IEEE Transactions on Evolutionary Computation. He is a member of the steering committee of the Parallel Problem Solving from Nature Conference Series (PPSN), elected member of the International Society of Genetic Algorithms Council (ISGA) since 1995, advisory board member of the *Handbook of Evolutionary Computation* (New York, NY: Oxford University Press & Institute of Physics Publishing, 1997), and received the Lifetime Achievement Award from the Evolutionary Programming Society (La Jolla, CA) in 1995.