Hindawi Publishing Corporation
Applied Computational Intelligence and Soft Computing
Volume 2011, Article ID 938240, 20 pages
doi:10.1155/2011/938240

*Review Article*

# Evolutionary Computation and Its Applications in Neural and Fuzzy Systems

**Biaobiao Zhang,[1] Yue Wu,[1] Jiabin Lu,[2] and K.-L. Du[1,3]**

[1] *Central Research Institute, Enjoyor Inc., Hangzhou 310030, China*
[2] *Faculty of Electromechanical Engineering, Guangdong University of Technology, Guangzhou 510006, China*
[3] *Department of Electrical and Computer Engineering, Concordia University, Montreal, QC, Canada H3G 1M8*

Correspondence should be addressed to K.-L. Du, kldu@ece.concordia.ca

Received 7 March 2011; Revised 6 July 2011; Accepted 4 August 2011

Academic Editor: Miin-Shen Yang



Neural networks and fuzzy systems are two soft-computing paradigms for system modelling. Adapting a neural or fuzzy system requires to solve two optimization problems: structural optimization and parametric optimization. Structural optimization is a discrete optimization problem which is very hard to solve using conventional optimization techniques. Parametric optimization can be solved using conventional optimization techniques, but the solution may be easily trapped at a bad local optimum. Evolutionary computation is a general-purpose stochastic global optimization approach under the universally accepted neo-Darwinian paradigm, which is a combination of the classical Darwinian evolutionary theory, the selectionism of Weismann, and the genetics of Mendel. Evolutionary algorithms are a major approach to adaptation and optimization. In this paper, we first introduce evolutionary algorithms with emphasis on genetic algorithms and evolutionary strategies. Other evolutionary algorithms such as genetic programming, evolutionary programming, particle swarm optimization, immune algorithm, and ant colony optimization are also described. Some topics pertaining to evolutionary algorithms are also discussed, and a comparison between evolutionary algorithms and simulated annealing is made. Finally, the application of EAs to the learning of neural networks as well as to the structural and parametric adaptations of fuzzy systems is also detailed.

## 1. Introduction

The adaptation of creatures to their environments results from the interaction of two processes, namely, evolution and learning. Unlike evolution, which is based on the Darwinian model of a species, learning is based on the connectionist model of the brain. Evolution is a slow stochastic process at the population level that determines the basic structures of a species, while learning is a process of gradually improving an individual's adaptation ability to its environment by tuning the structure of the individual. Evolutionary algorithms (EAs) are stochastic search methods inspired by the Darwinian model, while neural networks are learning models based on the connectionist model. Compared to the connectionist model-based learning process, fuzzy systems are a high-level abstraction of human cognition.

Neural networks, fuzzy systems, and evolutionary algorithms are the three major soft-computing paradigms for computational intelligence. Neural networks and fuzzy systems are two major approaches to system modeling. Adapting neural networks or fuzzy systems involves the solution of two optimization problems: structural optimization and parametric optimization. Structural optimization is the first step that tries to find an optimum system structure; it is a discrete (combinatorial) optimization problem and is very hard to solve using conventional calculus-based optimization techniques. After the system structure is determined, parametric optimization is applied to find the optimum system parameters in a continuous parametric space. Parametric optimization can be solved using conventional optimization techniques; however, the solution may be easily trapped at a

bad local optimum. Evolutionary computation is particularly suited to the adaptation (learning) of neural and fuzzy systems.

Evolutionary computation is a major research area for adaptation and optimization. The approach originates from the Darwin's principle of natural selection, also called survival of the fittest. In the Darwinian model, knowledge acquired by an individual cannot be transferred into its genome and subsequently passed onto the next generation. Combination of learning and evolution, embodied by evolving neural networks, has better adaptability to a dynamic environment [1, 2]. The interaction of learning upon evolution accelerates evolution, and this can take the form of the Lamarckian evolution or be based on the Baldwin effect [3]. The Lamarckian strategy allows the inheritance of the acquired traits during an individual's life into the genetic code so that the offspring can inherit its characteristics. Although the Lamarckian theory is biologically unfounded, EAs as artificial biological systems can benefit from it, and also the Lamarckian theory properly characterizes the evolution of human cultures. On the other hand, the Baldwin effect is biologically plausible, wherein learning makes individuals adapt better to their environments, thus, increasing their reproduction probability. Learning actually smoothes the fitness landscape and, thus, facilitates evolution. Fitness landscape is referred to as the set of all possible genotypes and their respective fitness values. The learned behaviors become instinctive behaviors in subsequent generations, and there is no direct alteration of the genotype. The Baldwin effect is purely Darwinian, although it has consequences similar to those of the Lamarckian evolution [4].

EAs are especially useful for an optimization problem in a domain, where the calculus is difficult to implement or is inapplicable. In EAs, individuals in a population compete and exchange information with one another by using three basic genetic operators, namely, crossover or recombination, mutation, and selection. The procedure of a typical EA is given in Algorithm 1.

The EA approach is a general-purpose directed stochastic global search. It performs search over a large, complex, noncontinuous, nondifferentiable, and multimodal surface and can reliably and fast solve hard problems. It can always reach the nearoptimum or the global optimum. EAs are extendable and easy to hybridize. EAs possess inherent parallelism by evaluating multipoints simultaneously. The evaluation function must be calculated for all the individuals of the population, leading to a computation load that is much higher than that of a simple random search or a gradient search.

We now give some terminologies used in the EA literature that are an analogy to their biological counterparts [5, 6]. A set of individuals in a generation is called a population, $\mathcal{P}(t) = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{N_P}\}$, where $\mathbf{x}_i$ is the $i$th individual, also termed a *chromosome*, sometimes called a *genome*, and $N_P$ is the size of the population. The chromosome is often represented as a string in EAs. Each chromosome $\mathbf{x}$ comprises of a string of parameters $x_i$, called *genes*, that is, $\mathbf{x} = [x_1 \; x_2 \cdots x_n]$, where $n$ is the number of genes in the chromosome. Each gene encodes a parameter of the problem into the chromosome. A gene is usually encoded as a binary string or a real number. The value of a gene is an allele. A genotype represents a coded solution, that is, a chromosome. A phenotype represents a decoded solution. The mapping of a set of genotypes to a set of phenotypes is referred to as genotype-phenotype map. Fitness is the value of the objective (fitness) function for a chromosome $\mathbf{x}$, namely, $f(\mathbf{x})$. The set of all possible genotypes and their respective fitness values is called a *fitness landscape*. Natural selection causes traits to become more prevalent when they contribute to fitness. Natural selection is different from artificial selection. Genetic drift and gene flow are two other mechanisms in biological evolution. Genetic flow, also known as *genetic migration*, is the migration of genes from one population to another. The termination criterion is necessary for terminating an EA, and it can be selected as a maximum number of generations or the convergence of the genotypes of the individuals. Convergence of the genotypes occurs when all the bits or values in the same positions of all the strings are identical. Phenotypic convergence without genotypic convergence is also possible.

This paper is organized as follows. In Section 2, we introduce the genetic algorithm (GA). The simple GA, real-coded GA, GAs for sequence optimization, some improved variants of the GA, parallel GAs, and two-dimensional GAs are described. Section 3 deals with the evolutionary strategies (ESs). A comparison between the ES and the GA is also made in Section 3. Three other EAs, namely, the genetic programming (GP), the evolutionary programming (EP), and the memetic algorithm, are described in Section 4. A few theoretical topics are introduced in Section 5. Three other population-based optimization methods including particle swarm optimization (PSO), the immune algorithm, and ant colony optimization (ACO) are treated in Section 6. Multiobjective, multimodal, and constraint-satisfaction optimizations are dealt with in Section 7. In Section 8, fuzzy logic is used to construct evolutionary algorithms. A comparison between EA and simulated annealing (SA) is given in Section 9. In Sections 10 to 12, the applications of EAs to the construction of neural networks, fuzzy systems, and neuro-fuzzy systems are, respectively, described. A summary is given in Section 13.

## 2. Genetic Algorithms

The GA [7] is the most popular EA. A simple GA consists of a population generator and selector, a fitness estimator, and three genetic operators, namely, selection, mutation, and crossover. The mutation operator inverts randomly chosen bits with a certain probability. The crossover operator combines parts of the chromosomes of two individuals and generates two new offsprings, which are used to replace low-fitness individuals in the population. The search process is terminated when certain termination criterion is satisfied.

*2.1. Encoding/Decoding.* The GA uses binary coding. For a chromosome $\mathbf{x}$, each gene $x_i$, $i = 1, \ldots, n$, has its value (allele) encoded as a binary string of $l_i$ bits, and the total length of the

**Algorithm EA**
(1) Set $t = 0$.
(2) Randomize initial population $\mathcal{P}(0)$.
(3) Repeat until the termination criterion is satisfied:
  (a) Evaluate fitness of each individual in $\mathcal{P}(t)$.
  (b) Select individuals as parents from $\mathcal{P}(t)$ based on fitness.
  (c) Apply crossover on the parents, and generate $\mathcal{P}'(t+1)$.
  (d) Apply mutation on $\mathcal{P}'(t+1)$, and generate $\mathcal{P}(t+1)$.
  (e) Set $t = t + 1$.

Algorithm 1: Principle of EA.

chromosome is $l = \sum_{i=1}^{n} l_i$. For a chromosome of bitlength $l$, there are $2^l$ possible values. If $x_i \in [x_i^-, x_i^+]$ has a coding $s_{l_i} \dots s_2 s_1$, $s_i \in \{0, 1\}$, then the decoding function is given by

$$x_i = x_i^- + (x_i^+ - x_i^-)\ \frac{1}{2^{l_i} - 1} \left( \sum_{j=0}^{l_i - 1} s_j 2^j \right). \tag{1}$$

In binary coding, there is the so-called Hamming cliffs phenomenon, where large Hamming distances between the binary codes of adjacent integers occur. Gray coding is another approach to encoding the parameters into bits. The decimal value of a Gray-encoded integer variable increases or decreases by 1 if only one bit is changed. However, the Hamming distance does not monotonously increase with the difference in integer values. Based on a Markov chain analysis of the GA, there is little difference between the performance of binary and Gray's codings for all possible functions [8]. Also, Gray's coding does not necessarily improve the performance for functions that have fewer local minima in the Gray representation than in the binary representation [8]. This reiterates the no-free-lunch theorem [9]; namely, no representation is superior for all classes of problems. Conversion between binary and Gray codings is formulated in [6, 8, 10].

Binary coding is a nonlinear coding, which is undesirable when approaching the optimum. GAs usually use fixed-length binary coding. There are also some variable-length encoding methods such as encoding both the value and the position of each bit in the chromosome [11] and the delta coding [12].

*2.2. Selection/Reproduction.* Selection provides a driving force in the GA. From a population $\mathcal{P}(t)$, those individuals with strong fitness will be selected for reproduction so as to generate a population of the next generation, $\mathcal{P}(t+1)$. Chromosomes with high fitness are selected and are assigned a higher probability of reproduction. The selection mechanism is split into two phases, namely, parental selection and replacement strategy.

*2.2.1. Sampling Mechanism.* The sampling of chromosomes can be in either a stochastic, a deterministic, or a hybrid manner. Among the conventional selection methods, the roulette-wheel selection [7] is a stochastic method, while the ranking selection [13] and the tournament selection [14] are hybrid methods. The roulette-wheel selection [7, 14] is a simple and popular selection scheme. Segments of the roulette wheel are allocated to individuals of the population in proportion to the individuals' relative fitness scores. The selection can be done by assigning a chromosome $\mathbf{x}_i$ a probability

$$P_i = \frac{f(\mathbf{x}_i)}{\sum_{i=1}^{N_P} f(\mathbf{x}_i)}, \quad i = 1, 2, \dots, N_P. \tag{2}$$

Consequently, a chromosome with a larger fitness has a possibility of getting more offsprings. Typically, the population size $N_P$ is relatively small, and this fitness-proportional selection may select a disportionately large number of unfit chromosomes. This easily induces premature convergence when all the individuals in the population become very similar after a few generations. The GA, thus, degenerates into a Monte Carlo's type search method. Scaling a raw objective function to some positive function is a method to mitigate these problems, and this can prevent the best chromosomes from producing too many expected offsprings [14].

The ranking selection [13] can eliminate some of the problems inherent in the fitness-proportional selection. It can maintain a more constant selective pressure. Individuals are sorted and assigned a rank from 1 to $N_P$ according to their fitness values. The selection probability is linearly assigned according to their ranks [6, 13]. The tournament selection [14] involves multiple, typically 2, individuals at a time. During each tournament, only the best performing individual enters the mating pool. The tournament will be performed repeatedly $N_P$ times until the mating pool is filled. Due to its cheaper computational cost, binary tournament is practically equivalent to linear ranking in terms of selection pressure. The elitism strategy [15] always copies the best individual of a generation to the next generation. Although elitism may increase the possibility of premature convergence, it improves the performance of the GA in most cases and, thus, is integrated in most GA implementations [16].

*2.2.2. Replacement Strategy.* Replacement strategy decides as to how many individuals of one population will be replaced to generate the population for a new generation. Some replacement strategies are the complete generational replacement [7], replace random [7], replace worst, replace

oldest, and deletion by kill tournament [17]. In the crowding strategy [18], an offspring replaces one of the parents whom it most resembles using the similarity measure of the Hamming distance; this effectively eliminates premature convergence and allows for great exploration capability. All these replacement strategies may result in a situation where the best individuals in a generation may fail to reproduce. In [19], this problem is solved by introducing into the system a new variable that stores the best individuals obtained so far. The elitism strategy cures the same problem without changing the system state [16].

*2.3. Crossover.* Crossover, the primary exploration operator in the GA, searches the range of possible solutions based on existing solutions. Crossover, as a binary operator, exchanges information between two parent chromosomes at randomly selected positions and produces two new offsprings. Some commonly used crossover techniques are the one-point [7], two-point [18], multipoint [20], and uniform [21] crossovers. The crossover points are typically at the same, random positions for both parent chromosomes.

The one-point crossover requires one crossover point on the parent chromosomes, and all the data beyond that point are swapped between the two parent chromosomes. The one-point crossover is easy to model analytically, and it generates bias toward bits at the ends of the strings. The two-point crossover selects two points on the parent chromosomes, and everything between the two points is swapped. This method causes a smaller schema disruption than the one-point crossover but generates bias at a different level, since it does not sample all regions of the string equally, and the ends of the string are rarely sampled. This problem can be solved by wrapping around the string. The multipoint crossover treats each string as a ring of bits divided by $m$ crossover points into $m$ segments, and each segment is exchanged at a fixed probability. The uniform crossover exchanges bits of a string rather than segments. Individual bits in the parent chromosomes are compared, and each of the nonmatching bits is probabilistically swapped with a fixed probability, typically 0.5. The uniform crossover is unbiased. The half uniform crossover (HUX) [22] swaps exactly half of the nonmatching bits.

The one-point and two-point crossover operations preserve schemata due to low disruption rates but are less exploratory. In contrast, the uniform crossover is more exploratory but has a high disruptive nature. The uniform crossover is more suitable for small populations, while the two-point crossover is better for large populations. The two-point crossover performs consistently better than the one-point crossover [21]. However, when all the chromosomes are very similar or even the same in the population, it is difficult to generate a new structure by crossover only, and premature convergence takes place.

*2.4. Mutation.* Mutation is a unary operator that requires only one parent to generate an offspring. A mutation operator typically selects a random position of a random chromosome and replaces the corresponding gene or bit by other information. Mutation helps to regain the lost alleles and thus introduces genetic diversity into the population so as to prevent premature convergence from happening.

Mutations can be classified into point mutations and large-scale mutations. Point mutations are changes to a single position, which can be substitutions, deletions, or insertions of a gene or a bit. Large-scale mutations can be similar to the point mutations but operate at multiple positions simultaneously, at one point with multiple genes or bits, or even on the chromosome scale. Functionally, mutations introduce the necessary amount of noise to do hillclimbing. Two additional large-scale mutation operators are the inversion and rearrangement operators [7, 14]. The swap operator is the most primitive reordering operator, based on which many new unary operators including inversion can be derived.

High mutation rate can lead genetic search to random search. High mutation rate may change the value of an important bit and, thus, slow down the fast convergence of a good solution or slow down the process of convergence at the final stage of the iterations. Thus, mutation is made occasionally in the GA. In the simple GA, the mutation is typically selected as a substitution operation that changes one random bit in the chromosome at a time. An empirically derived formula that can be used as the probability of mutation $P_m$ at a starting point is $P_m = 1/(T\sqrt{l})$, where $T$ is the total number of generations, and $l$ is the string length [23].

The random nature of mutation and its low probability of occurrence make the convergence of the GA slow. The search process can be expedited by using the directed mutation technique [24] that deterministically introduces new points into the population by using gradient or extrapolation of the information acquired so far. It is commonly agreed that crossover plays a more important role if the population size is large, and mutation is more important if the population size is small [25].

*2.5. Other Genetic Operators.* A large number of non-canonical genetic operators are described in the literature. The best known are hill-climbing operators [26], typically gradient-descent ones. In the parallel GA (PGA) [26], some or all individuals of the population improve their fitness by hillclimbing. Given an individual, this operator finds an alternative similar individual that represents a local minimum close to the original individual in the solution space. The combination of genetic operators and local search can be based on either the Lamarckian strategy or the Baldwin effect.

The bit climber [27] is a simple stochastic bit-flipping operator. The fitness is computed for an initial string. A bit of the string is randomly selected and flipped, and the fitness is computed at the new point. If the fitness is lower than its earlier value, the new string becomes the current string. The operation repeats until no bit flip improves the fitness. The bit-based descent algorithm is several times faster than an efficient GA [27]. It can also be used as a local-search operator in the GA.

Most selection schemes are based on individuals' fitness. The entropy-Boltzmann selection method [28], stemming from the entropy and the important sampling methods in the Monte Carlo simulation, tends to escape from local optima. It avoids premature convergence systematically. The adaptive fitness consists of the usual fitness together with the entropy change due to the environment, which may vary from generation to generation.

*2.6. Real-Coded Genetic Algorithms for Continuous Numerical Optimization.* The floating-point and fixed-point coding techniques are two computer representation methods for real numbers and are widely used in continuous numerical optimization. The fixed-point coding allows more gradual mutation than the floating-point coding for the change of a single bit, and fixed-point coding is sufficient in most cases. The floating-point coding is capable of representing large or unknown domains, while the binary and fixed-point techniques may have to sacrifice accuracy for large domain. The real-coded GA using the floating-point or the fixed-point coding has an advantage over the binary-coded GA in exploiting local continuities in function optimization [29]. The real-coded GA is faster, more consistent from run to run and provides a higher precision than the binary-coded GA [30].

*2.6.1. Crossover.* In analogy to crossover operators for the binary-coded GA, crossover operators for the real-coded GA such as the one-point, two-point, multipoint, and uniform crossover operators are also defined [21]. Each gene ($x_i$) in a real-coded chromosome corresponds to a bit in a binary-coded chromosome. A blend of these crossover operators uniformly selects values which lie between the two points representing the two parents [31]. Crossover can also be defined as a linear combination of two parent vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ and generates two offsprings. This can be achieved by symmetrically interpolating the two parent chromosomes [6, 32] or extrapolating toward the better individual [6, 10, 30]. In [33], the crossover operator is defined as that which generates four chromosomes from two parents according to a strategy of combining the maximum, minimum, or average of all the parameters encoded in the chromosome and vectors having elements as the lower and upper bounds for the corresponding parameters in $\mathbf{x}$, respectively. These potential offsprings spread over the domain. Only the one with the largest fitness is used as the offspring of the crossover operation.

*2.6.2. Mutation.* Mutation can be conducted by replacing one or more genes $x_i$, $i = 1, \ldots, n$, with a randomly selected real number $x_i'$ from the domain of the corresponding parameter. The popular uniform mutation substitutes the value of a randomly selected gene with a random value between its upper and lower bounds [29]. A nonuniform mutation capable of fine-tuning the system is defined by adding a random term to $x_i$ [30]. This operator searches the space uniformly when $t$ is small and gradually searches locally as $t$ increases. This is similar to an annealing process.

The Gaussian mutation [32] is usually applied in the real-coded GA. Given the chromosome $\mathbf{x}$, the Gaussian mutation produces a new offspring $\mathbf{x}'$ with one or multiple genes defined by

$$x_i' = x_i + N(0, \sigma_i), \tag{3}$$

where $N(0, \sigma_i)$ is a random number drawn from a normal distribution with zero mean and standard deviation $\sigma_i$. The parameter $\sigma_i$ is traditionally selected as a linearly or exponentially decreasing function such as $\sigma_i(t) = 1/\sqrt{(1+t)}$. The Cauchy mutation replaces the Gaussian distribution by the Cauchy distribution, and it is more likely to generate an offspring further away from its parent than the Gaussian mutation due to the long flat tails of the Cauchy distribution [34]. Cauchy mutation performs better when the current search point is far from the global minimum, while the Gaussian mutation is better at finding a local minimum in a good region. The two mutation operators are combined in [34], where for each parent two offsprings are generated, each by one of the mutation methods, and only the better one is selected.

In [33], offspring obtained by crossover further undergo the mutation operation. Three new offsprings are generated by allowing one parameter, some of the parameters, and all the parameters in the chromosome to change by a randomly generated number, subject to constraints on each parameter. Only one of the offsprings will be used to replace the chromosome with the smallest fitness value, according to a predefined probability criterion that, as in the SA [35], allows uphill move in a controlled fashion. Hence, the search domain is significantly enlarged.

*2.7. Genetic Algorithms for Sequence Optimization.* For sequence optimization problems, such as scheduling and the travelling salesperson (TSP), permutation encoding is a natural representation for a set of symbols, and each symbol can be identified by a distinct integer. This representation avoids missing or duplicate alleles [36].

Genetic operators should be defined so that infeasible solutions do not occur, or a way is viable for repairing or rejecting infeasible solutions. Genetic operators for reordering a sequence of symbols can be unary operators such as inversion and swap [7], or binary operators which combine features of inversion and crossover, such as the partial matched crossover (PMX), the order crossover (OX), and the cycle crossover (CX) [14], the edge recombination (ER) [37], as well as intersection and union [38]. The TSP is a benchmark for GAs. In [38], the performance of some genetic operators for sequences has been empirically compared for the TSP. A general survey of various genetic operators for the TSP is given in [30].

Random keys representation [39] encodes each symbol by a random number in $(0, 1)$. By sorting the random keys in a descending or ascending order, we can get a decoded solution. For example, given a TSP of 6 cities, if the chromosome for a route is encoded as $(0.32, 0.96, 0.35, 0.27, 0.93, 0.87)$, by sorting the genes in the descending order, the largest random key is 0.96, and, thus, the second city is the beginning of

the route, and the whole route is 2 → 5 → 6 → 3 → 1 → 4. This representation avoids infeasible offspring by representing solutions in a soft manner, such that the real-coded GA and the ES can be applied directly for sequence optimization problems. The ordering messy GA (OmeGA) [40] is specialized for solving sequence optimization problems. The OmeGA uses the mechanics of the fast messy GA [41] and represents the solutions using random keys.

*2.8. Exploitation versus Exploration.* The convergence analysis of the simple GA is based on the concept of *schema* [7]. A schema is a bit pattern which functions as a set of binary strings. The schema theorem [7, 14] asserts that the proportions of the better schemata to the overall population increases as the generation progresses, and eventually the search converges to the best solution with respect to the optimization function [14]. However, the GA often converges rather prematurely before the optimal solution is found.

Exploitation means taking advantage of the existing information, while exploration means searching new areas. Exploitation is achieved by the selection procedure, while exploration is achieved by crossover and mutation. An increase in exploration is at the expense of exploitation. The balance between exploitation and exploration controls the performance of the GA and is determined by the choice of the control parameters, namely, the probability of crossover $P_c$, the probability of mutation $P_m$, and the population size $N_P$. Some tradeoffs are made for selecting the optimal control parameters [42]: increasing $P_C$ results in fast exploration but a high disruption rate of good strings; increasing $P_m$ tends to transform the genetic search into a random search but helps to reintroduce lost alleles into the population; increasing $N_P$ increases the genetic diversity in the population and reduces the probability of premature convergence, while it increases the time for convergence. These control parameters depend on one another, and their choices depend on the nature of the problem. For a small $N_P$, one can select relatively large $P_m$ and $P_C$ and vice versa. Typically, $P_c$ is selected as 0.6 to 0.9 and $P_m$ as 0.001 to 0.01 [14]. Empirical evidence shows that the optimal $P_m$ in the GA differs according to whether or not crossover is used. When crossover is used, $P_m$ should be selected as a small constant [43]; otherwise, the GA can start with a high $P_m$, decreasing towards the end of the run [43, 44]. In [45], the population size is suggested to be $N_P = 1.65 \times 2^{0.21l}$, $l$ being the string length. Empirically, the GA with $N_P$ within 20 to 30, $P_c$ in the range 0.75 to 0.95, and $P_m$ in the range 0.005 to 0.01 provide a good performance [23].

Adaptation of control parameters helps to avoid premature convergence. More emphasis should be paid on exploration at the beginning of a search process, while more emphasis should be on exploitation at a later stage. A simple method to adapt $P_m$ is given by [46], which selects a relatively large initial value of $P_m$ and then linearly reduces it with the generation index $t$ until a small final value of $P_m$ is reached. $P_m$ can also be modified depending on the tradeoff between exploration and exploitation [42, 44]:

$$P_m(t) = \frac{\alpha_0 e^{-\gamma_0 t/2}}{N_P \sqrt{l}}, \tag{4}$$

where constants $\alpha_0 > 0$ and $\gamma_0 \geq 0$, and $l$ is the length of the chromosome. In [23], $\alpha_0$ is selected as 1.76 and $\gamma_0$ as 0.

In an adaptive GA [42], $P_c$ and $P_m$ are not predefined but determined adaptively for each solution of the population. $P_c$ and $P_m$ range from 0 to 1.0 and 0 to 0.5, respectively. Low values of $P_c$ and $P_m$ are assigned to high-fitness solutions, while low-fitness solutions are assigned very high values of $P_c$ and $P_m$. The best solution of every population is protected, not subjected to crossover but subjected to a minimal amount of mutation. All subaverage solutions are mutated by $P_m = 0.5$ and, thus, are completely disrupted, and totally new solutions are created. This adaptive GA significantly outperforms the simple GA and effectively avoids local minima.

The genetic diversity of the population can be easily improved so as to prevent premature convergence by adapting the size of the population [11, 30, 47] and using partial restart [22]. Partial restart can be implemented by a fixed restart schedule at a fixed number of generations or implemented when premature convergence occurs [48].

The binary coding of the GA results in limited accuracy and slow convergence. Adaptation can be introduced into coding. Some examples of adaptive coding are the delta coding [12], the dynamic parameter encoding (DPE) [49], and the fuzzy coding [50, 51].

*2.9. Variants of the Genetic Algorithms.* The messy GA [11] starts with a large initial population. In the primordial stage, the population is halved at regular intervals, and only the selection operation is applied. A dynamical population size also helps to reduce premature convergence. A variable-length encoding method is employed. The fast messy GA [41] is an improved version of the messy GA. The GENITOR [52] employs an elitist selection that is a deterministic, rank-based selection method so that the best $N_P$ individuals found so far are preserved by using a cross-generational competition. This selection strategy is similar to the $(\lambda + \mu)$ strategy of the ES. The CHC algorithm [22] also borrows from the $(\lambda + \mu)$ strategy of the ES. Incest prevention is introduced to prevent mating between similar individuals. The highly disruptive form of crossover, namely, the HUX [22], is applied, and mutation is not performed. Diversity is reintroduced by restarting partial population whenever convergence is detected. This is implemented by randomly flipping a fixed proportion of the best individual found so far as template and introducing the better offspring into the population. The GA with varying population size (GAVaPS) [30, 47] introduces the concept of age of a chromosome in the number of generations as the selection mechanism. A chromosome will die off when its age reaches its lifetime decided by its fitness. Each chromosome from the population has equal probability to reproduce, independent of its fitness. Some strategies for assigning lifetime values are proposed to tune the size of the population to the current stage of search. Both the CHC and the GAVaPS significantly outperform the simple GA.

*2.10. Parallel Genetic Algorithms.* Parallel GAs can be grouped into global parallelized GAs, coarse-grained parallel GAs and fine-grained parallel GAs. Global parallelized GAs implement the GA by evaluating individuals and the genetic operations in explicitly parallel mode. The speedup is proportional to the number of processors. In coarse-grained parallel GAs, the population is divided into a few isolated subpopulations, called demes. Individuals can migrate from one deme to another, and an existing GA such as the simple GA, the GENITOR [52], and the CHC can be executed within each deme. The demes swap a few strings once in a few generations. The GENITOR II [53] is a coarse-grained parallel implementation of the GENITOR. Individuals migrate at fixed intervals to neighboring nodes. Immigrants replace the worst individuals in the target deme. The PGA [26] is an asynchronous parallel GA, wherein each individual of the population improves its fitness by hillclimbing. On the other hand, fine-grained GAs, also called cellular GAs or massively parallel GAs, partition the population into many very small demes, typically one individual per deme. The fine-grained parallel GA [54–56] organizes its population as a two-dimensional grid of chromosomes. Selection and mating are confined in a local area, and this can reduce the selection pressure to achieve more exploration of the search space. Local mating can find very fast multiple optimal solutions in the same run and is much more robust [55]. The performance of the algorithm degrades as the size of the neighborhood increases.

*2.11. Two-Dimensional Genetic Algorithms.* Under the scenario of two-dimensional problems such as image processing, linear encoding used by conventional EAs causes a loss of two-dimensional correlations, and, thus, extra problem-specific operators must be introduced. If an image is encoded by concatenating horizontal lines, crossover operations result in a large vertical disruption. In two-dimensional GAs [57–59], each individual is a two-dimensional binary string. Conventional mutation and reproduction operators can be applied in the normal way, but the conventional two-point crossover operator samples the matrix elements in a two-dimensional string very unevenly. Some genetic operators for two-dimensional strings are also defined, such as the crossover operator that exchanges rectangular blocks between pairs of matrices [58] and an unbiased crossover operator called UNBLOX (UNiform BLOck CROSSover) [57]. The UNBLOX is a two-dimensional wrap-around crossover and can sample all the matrix positions equally.

The convergence rates of two-dimensional GAs are higher than that of the simple GA for bitmaps [57]. In [59], a two-dimensional crossover operator is defined for learning the architecture and weights of a neural network, where a neural network is interpreted as an oriented graph, and the crossover operation is performed by swapping the subgraphs connected to a common selected neuron.

## 3. Evolutionary Strategies

The evolutionary strategies (ESs) [32, 60], also known as evolution strategies, are another most popular EA. The ES was originally developed for numerical optimization [32], and was later extended to discrete optimization [61]. The objective parameters $\mathbf{x}$ and strategy parameters $\vec{\boldsymbol{\sigma}}$ are directly encoded into the chromosome by using a regular numerical representation, and thus no coding or decoding is necessary. In contrast to the GA, the primary search operator in the ES is mutation.

*3.1. Crossover, Mutation, and Selection.* The canonical ES uses only mutation operations. Crossover operators used for the real-coded GA can be introduced into the ES. For example, the crossover operator can be defined by recombining two parents $\mathbf{x}_1$ and $\mathbf{x}_2$ such that the $i$th gene of the generated offspring $\mathbf{x}'$ takes the average value of the $i$th genes of the two parents or selected as either $x_{1,i}$ or $x_{2,i}$. An offspring obtained from recombination is required to be mutated before it is evaluated and entered into the population. Mutation is applied to a parent or an offspring generated by crossover. For a chromosome $\mathbf{x}$, the Gaussian mutation, as defined by (3), produces a new offspring $\mathbf{x}'$ with one or more genes defined by

$$x_i' = x_i + N(0, \sigma_i). \tag{5}$$

The optimal $\sigma_i$ is evolved by encoding it into the chromosome. $\sigma_i$ is usually mutated first, and then $x_i$ is mutated by using $\sigma_i'$:

$$\sigma_i' = \sigma_i \mathbf{e}^{N(0,\delta\sigma_i)}, \tag{6}$$

where $\delta\sigma_i$ is a parameter of the method.

The $(\lambda+\mu)$ and $(\lambda,\mu)$ schemes are the two major selection schemes applied in the ES, where $\mu$ is the population size and $\lambda$ the number of offsprings generated from the population. Both selection schemes are deterministic ranking-based sampling methods, which make the ES more robust than the GA. The $(\lambda+\mu)$ scheme selects the $\mu$ fittest individuals from the $(\lambda+\mu)$ candidates to form the next generation, while the $(\lambda,\mu)$ scheme selects the $\mu$ fittest individuals from $\lambda$ $(\lambda \geq \mu)$ offspring. The $(\lambda+\mu)$ scheme is the elitist and, therefore, guarantees a monotonically improving performance. This scheme, however, is unable to deal with changing environments and jeopardizes the self-adaptation mechanism with respect to the strategy parameters, especially within a small population. Thus, the $(\lambda,\mu)$ scheme is recommended, with a ratio of $\lambda/\mu = 7/1$ being optimal [62].

*3.2. Evolutionary Strategies versus Genetic Algorithms.* There are some differences between the ES and the GA as following.

(1) The selection procedure in the ES is deterministic ranking based, and each individual in the population has the same mating probability, while selection in the GA is random, and the chances of selection and mating are proportional to an individual's fitness.

(2) Selection in the ES is implemented after crossover and mutation, while selection in the GA is carried out before crossover and mutation are applied.

(3) In the ES, the strategy parameters $\sigma_i$ are evolved automatically by encoding them into chromosomes, while the control parameters in the GA need to be prespecified.

(4) In the GA, mutation is used to regain the lost genetic diversity, while, in the ES, mutation functions as a hill-climbing search operator. For the Gaussian mutation, the tail part of the normal distribution may generate a chance for escaping from a local optimum.

(5) Other differences are embodied in the encoding methods and genetic operators.

However, the line between the two techniques is now being blurred, since both of the techniques borrow ideas from each other. For example, the CHC [22] has both the properties of the GA and the ES.

*3.3. New Mutation Operators.* Advances in the ES are mainly focused on the design of new mutation operators. This enables the ES to evolve significantly faster. The covariance matrix-adaptation-(CMA-) based mutation operator makes the ES two orders of magnitude faster than the conventional ES [63, 64]. The set of all mutation steps which yield improvements is called an *evolution path* of the ES [63]. The CMA-type ES [63, 64] technique uses information embedded in the evolution path to accelerate the convergence. The CMA is a completely derandomized self-adaptation scheme, and subsequent mutation steps are uncorrelated with the previous ones. The mutation operator is defined by

$$\mathbf{x}' = \mathbf{x} + \delta \mathbf{B}\mathbf{z}, \tag{7}$$

where $\delta$ is a adaptive global step size, $\mathbf{z}$ is a random vector whose elements are drawn from a normal distribution $N(0,1)$, and the columns of the rotation matrix $\mathbf{B}$ are the eigenvectors of the covariance matrix $\mathbf{C}$ of the mutation points. The CMA implements the principal component analysis (PCA) of the previously selected mutation steps to determine the new mutation distribution [64].

Two self-organizing ESs given in [65] are derived from the self-organizing map (SOM) [66] and the neural-gas-(NG-) [67] based mutation operators, respectively. These ESs are not population based. The self-organizing networks are used to generate trial points, one trial point being generated per iteration by using the network nodes. The function is evaluated for the generated trial points only, not for the network nodes. The network is used only to track the probability distribution of the trial points that improve the current best known function value. The SOM-type ES has the problem that the number of nodes increases exponentially with the number of function parameters due to its grid topology. The NG-type ES does not have such a problem, but may suffer from premature convergence; this can be corrected by introducing an additional adaptation term. Both ESs are empirically more reliable than the CMA-type ES, yet without the necessity for parameter tuning.

## 4. Other Evolutionary Algorithms

In addition to the GA [7] and the ES [32], the GP [19] and the EP [68] are also popular approaches to evolutionary computation. The memetic algorithm [69] is also a popular method that combines the ideas in the neo-Darwinian paradigm and the Lamarckian strategy. In the community of EAs, the canonical EAs are modified by incorporating problem-specific knowledge, by defining problem-specific genetic operators, or by hybridizing heuristics from different EAs such that the capability of EAs to practical applications is maximized. The boundaries between different branches of EAs overlap.

*4.1. Genetic Programming.* The GP [19] is a variant of the GA for symbolic regression such as evolving computer programs, rather than evolving simple strings. It can also be used for automatic discovery of empirical laws. The major difference between the GA and the GP lies in coding. The GP has chromosomes of both variable length and data structure, and hierarchical trees are used to encode chromosomes. Each tree represents the parse tree for a given program. The GP is particularly suitable for problems in which the optimal underlying structure must be discovered. It, however, suffers from the so-called bloat phenomenon, resulting from the growth of noncoding branches in the individuals. The bloat phenomenon may cause an excessive consumption of computer resources and increase the expense for fitness evaluation.

*4.2. Evolutionary Programming.* The EP [68] was originally presented for evolving artificial intelligence to predict changes in an environment, which was coded as a sequence of symbols from a finite alphabet. Each chromosome is encoded as a finite state machine. The EP was later generalized for solving numerical optimization problems based on the Gaussian mutation [70, 71]. The EP and the ES are very similar the EP corresponds to the $(\lambda + \lambda)$ strategy of the ES. The major differences between the two methods are in crossover and selection. The EP does not use crossover, but uses a probabilistic competition for selection. A more comprehensive comparison between the two methods is given in [72]. For continuous functional optimization, it is generally known that both of the methods work better than the GA [62].

*4.3. Memetic Algorithms.* The memetic algorithm, also called the *cultural algorithm* and *genetic local search* [69, 73, 74], was inspired by Dawkins' notion of meme [75], which is a unit of information that reproduces itself when people exchange ideas. Unlike genes, memes are typically adapted by the people who transmit them before they are passed onto the next generation. Although it was motivated by the evolution of ideas, the memetic algorithm can be considered as the GA or an EA making use of local search, wherein evolution and learning are combined using the Lamarckian strategy. The memetic algorithm is considerably faster than

the simple GA. Basically, the memetic algorithm combines local search heuristics with crossover operators.

## 5. Theoretical Aspects

Theoretical breakthroughs in evolutionary computation are still limited. Holland's schema theorem [7] and Goldberg's building-block hypothesis [14] are the most important theoretical foundations on the GA as well as the EA mechanism.

*5.1. Schema Theorem and Building-Block Hypothesis.* A schema is a template describing a subset of strings with the same bits (0 or 1) at certain positions. The combined effect of selection, crossover, and mutation gives the reproductive schema's growth inequality [7]

$$m(H,t+1) \geq m(H+t)\frac{f(H)}{\overline{f}(t)}\left[1 - P_c\frac{\delta(H)}{l-1} - o(H)P_m\right], \tag{8}$$

where $H$ is a schema defined over the three-letter alphabet $\{0,1,*\}$ of length $l$, $*$ is a do not-care symbol, $m(H,t)$ is the number of examples of a particular schema $H$ within a population at generation $t$, $o(H)$ is the order of $H$, defined as the number of fixed positions (0s or 1s) in the template, $\delta(H)$, the defining length of $H$, is the distance between the first and last specific string positions, $f(H)$ is the average fitness of all strings in the population matched by $H$, and $\overline{f}(t)$ is the average fitness of the whole population at generation $t$.

The schema theorem can be readily derived from (8) [7]: *above-average schemata with short defining length and low order will receive exponentially increasing trials in subsequent generations of the GA.* Thus, schemata with high fitness and small defining lengths grow exponentially with time. Thus, the GA simultaneously processes a large number of schemata. For a population of $N_P$ individuals, the GA implicitly evaluates approximately $N_P^3$ schemata in one generation [14]. However, there are a lot of criticisms on the schema theorem. The schema growth inequality provides a lower bound for one-generation transition of the GA. For multiple generations, the prediction of the schema may be useless or misleading due to the inexactness of the inequality [76].

The building-block hypothesis is the assumption that strings with high fitness can be located by sampling building blocks with high fitness and combining the building blocks effectively. This is given as [6]: *the GA seeks near-optimal performance by the juxtaposition of short, low-order, and highly fit schemata, called building block.*

GA-deceptive functions are a class of functions, where low-order building blocks are misleading, and their combinations cannot generate higher-order building blocks [14, 77]. A fitness landscape with the global optimum surrounded by a part of the landscape of low-average payoff is highly unlikely to be found by the GA, and, thus, the GA may converge to a suboptimal solution. Deceptive problems remain to be hard problems for EAs. The messy GA [11] was specifically designed to handle bounded deceptive problems. Due to deceptive problems, the building-block hypothesis is facing strong criticism [76]. The static building-block hypothesis was proposed as the underlying assumption for defining deception, and augmented GAs for deceptive problems were also proposed [76].

*5.2. Dynamic Analysis of Evolutionary Algorithms.* Thus far, the selection of control parameters, the roles of crossover and mutation, replacement strategies, and convergence properties, still remain unsolved. Recently, more attempts have been made on characterizing the dynamics of EAs [17, 78–80]. The search process of EAs can be analyzed within the framework of Markov's chains [71]. In [18], a Markov chain analysis was conducted for a population of one-locus binary genes to reach different levels of convergence in an expected number of generations under random selection. The exact model introduced in [78] provides a complete model as to how all strings in the search space are processed by a simple GA using infinite population assumptions. In [79], another exact model for the simple GA has been obtained in the form of a Markov chain, and the trajectory is followed by finite populations related to the evolutionary path predicted by the infinite population model. An exact infinite population model of a simple GA for permutation-based representations has been developed in [80], wherein various permutation crossover operators are represented by the mixing matrices. In [17], a Markov chain analysis has been made to model the expected time for a single member of the optimal class to take over finite populations in the case of different replacement strategies.

## 6. Other Population-Based Optimization Methods

There are also some other well-known computational techniques inspired by biological adaptive systems such as collective behavior of animals and insects as well as the immune systems of mammals. These population-based optimization methods, including the PSO, the immune algorithm, and the ACO, belong to a branch of swarm intelligence, an emergent collective intelligence of groups of simple agents [81]. They are general-purpose methods for discrete and continuous function optimization.

*6.1. Particle Swarm Optimization.* PSO originates from studies of synchronous bird flocking and fish schooling [82–85]. It is somewhat similar to EAs but requires only primitive mathematical operators, less computational bookkeeping, and generally fewer lines of code. Thus, it is computationally inexpensive in terms of both memory requirement and speed. It evolves populations or swarms of individuals called *particles*.

For an optimization problem of $n$ variables, a swarm of $N_P$ particles is defined. Each particle has its own trajectory in the $n$-dimensional space, namely, position $\mathbf{x}_i$ and velocity $\mathbf{v}_i$, and moves in the search space by successively updating its trajectory. Populations of particles modify their trajectories based on the best positions visited earlier by themselves and other particles. All particles are evaluated by the fitness function to optimize. The particles are flown through the

solution space by following the current optimum particles. The algorithm initializes a group of particles with random positions (solutions) and then searches for optima by updating generations (iterations). In every iteration, each particle is updated by following the two best values, namely, the particle best *pbest*, denoted $\mathbf{x}_i^*$, $i = 1, \ldots, N_P$, which is the best solution it has achieved so far, and the global best *gbest*, denoted $\mathbf{x}^g$, which is the best value obtained so far by any particle in the population. The best value for the population in a generation is a local best, *lbest*. The PSO has been extended to handle multiobjective optimization problems [86].

*6.2. Immune Algorithms.* The network theory for the immune system [87] is based on the clonal selection theory [88, 89]. The basic components of the immune system are two types of lymphocytes, namely B lymphocytes and T lymphocytes. B lymphocytes generate antibodies on their surfaces to resist their specific antigens, while T lymphocytes regulate the production of antibodies from B lymphocytes. The clonal selection theory describes the basic features of an immune response to an antigenic stimulus. The clonal operation is an antibody random map induced by the affinity and includes four steps, namely, clone, clonal crossover, clonal mutation, and clonal selection.

Artificial immune networks [90–92] employ two types of dynamics, namely, the short-term dynamics and the metadynamics. The short-term dynamics correspond to a set of cooperating or competing agents, while the metadynamics refine the results of the short-term dynamics. Thus, the short-term dynamics are closely related to neural networks, and the metadynamics are similar to the GA. The immune algorithm, also called the clonal selection algorithm, introduces suppress cells to change search scope and memory cells to keep the candidate solutions. It is an EA inspired by the immune system and is very similar to the GA. In the immune algorithm, *antigen* is defined as the problem to be optimized, and *antibody* is the solution to the objective function. Learning involves raising the relative population size and affinity of those lymphocytes. The immune algorithm first recognizes the antigen and produces antibodies from memory cells. Then it calculates affinity between antibodies, which can be treated as fitness. Antibodies are dispersed to the memory cell, and the concentration of antibodies is controlled by stimulating or suppressing antibodies. A diversity of antibodies for capturing unknown antigen is generated using genetic reproduction operators. The immune mechanism can also be defined as a genetic operator and integrated into a GA [93].

*6.3. Ant Colony Optimization.* The ACO [94–98] is a metaheuristic approach for solving discrete or continuous optimization problems such as COPs. The ACO heuristic was inspired by the foraging behavior of ants. Ants are capable of finding the shortest path between the food and the colony (nest) due to a simple pheromone laying mechanism. Ants use their pheromone trails for communicating information. The optimization is the result of the collective work of all ants in the colony. All the ants contribute to the pheromone reinforcement, and old trails will vanish due to evaporation. Different ACO algorithms arise from different pheromone value update rules.

The ant system [94] is an evolutionary approach, where several generations of artificial ants search for good solutions. Every ant of a generation builds up a complete solution, step by step, going through several decisions by choosing the nodes on a graph according to a probabilistic state transition rule, called the random proportional rule. A tabu list is used to save the nodes already visited during each generation. When a tour is completed, the tabu list is used to compute the ant's current solution. Once all the ants have built their tours, the pheromone is updated on all edges $i \to j$ according to a global pheromone-updating rule. A shorter tour gets a higher reinforcement. Each edge has a long-term memory (LTM) to store the pheromone. The ant colony system (ACS) [95] improves the ant system [94] by applying a *local pheromone updating rule* during the construction of a solution. The global updating rule is applied only to edges that belong to the best ant tour. Some important subsets of ACO algorithms, such as the most successful ACS and min-max ant system (MMAS) algorithms, have been proved to converge to the global optimum [97].

## 7. Multiobjective, Multimodal, and Constraint-Satisfaction Optimizations

*7.1. Multiobjective Optimization.* Multiobjective optimization aims to optimize a system with multiple conflicting objectives:

$$\min \mathbf{f}(\mathbf{x}) = \left(f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})\right)^T, \tag{9}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T \in \mathcal{X} \subset R^n$. A tradeoff between the conflicting objectives is necessary.

A weighted sum of these objectives, typically a normalized version, is usually used as the compromise of the system. This method may have difficulties in normalizing the individual objectives as well as in selecting the weights. The lexicographic-order optimization is based on the ranking of the objectives in terms of their importance. Fuzzy logic can be used to define a tradeoff of multiple objectives [99]. The Pareto method is popular for multiobjective optimization. It is based on the principle of nondominance. The Pareto optimum gives a set of solutions for which there is no way of improving one criterion without deteriorating another [14, 100]. A solution $\mathbf{x}_1$ is said to dominate $\mathbf{x}_2$, denoted $\mathbf{x}_1 \succ \mathbf{x}_2$, if $\mathbf{x}_1$ is better or equal to $\mathbf{x}_2$ in all attributes and strictly better in at least one attribute. The space in $R^n$ formed by Pareto optimal solutions is called the Pareto optimal frontier, $\mathcal{P}^*$.

The vector-evaluated GA (VEGA) [101] is the first GA for multiobjective optimization. The population is divided into equal-sized subpopulations, each independently searching the optimum of a single objective. Crossover is performed across subpopulation boundaries. Some heuristics are used to prevent the system from converging toward solutions

which are not with respect to any criterion. This algorithm, however, has bias toward some regions [102].

Nondominated sorting was introduced to rank a search population according to the Pareto optimality [14]. The procedure of identifying nondominated sets of individuals is repeated until the whole population is ranked. Equal probability of reproduction is assigned to all nondominated individuals in the population. The nondominated sorting GA (NSGA) [102] implements a nondominated sorting in the GA along with a niching and speciation method [14] to find multiple Pareto's optimal points simultaneously. This ranking is supplemented by a GA technique known as sharing, which lowers the objective function values of designs that are too similar, keeping the algorithm from converging to a single optimum [14]. The NSGA-II [103] improves the NSGA by introducing elitism and a crowed comparison operator.

Other popular EAs for multiobjective optimization may be based on the global non-elitist selection such as the niched Pareto GA (NPGA) [104], or the global elitist methods such as the strength Pareto EA (SPEA) [105], the Pareto envelope-based selection algorithm (PESA) [106], the Pareto archived ES (PAES) [107] as well as the SPEA-II [108], or the local selection such as the evolutionary local selection algorithm (ELSA) [109]. All these algorithms have the ability of finding multiple Pareto-optimal solutions in one single run.

*7.2. Multimodal Optimization.* Multimodal optimization is to identify a number of local optima and to maintain these solutions while continuing to search other local optima. Multimodality causes difficulty to any search method. Each peak in the solution landscape can be treated as a separate environment niche. A niche makes a particular subpopulation (species) unique. The niching mechanism embodies both cooperation and competition.

Crossover between individuals from different niches may lead to unviable offspring and is usually avoided [14]. It introduces a strong selection advantage to the niche with the largest population and, thus, prevents a thorough exploration of the fitness landscape [110]. A finite population will quickly concentrate on one region of the search space. Increasing the population size, decreasing the crossover probability, and biasing the crossover operator can also slow down symmetry breaking and, thus, increase the likelihood of finding the global optimum, which, in turn, slows down the rate of exploitation.

For multimodal optimization, three techniques, namely, niching, demes, and local search, are effective in identifying a number of local optima. Niching can be implemented by preventing or suppressing the crossover of solutions that are dissimilar to each other. It is also necessary to prevent a large subpopulation from creating a disproportionate number of offsprings. Niching techniques are usually based on the ideas of *crowding* [18] and/or *sharing* [14], which prevent a single genotype from dominating a population. Crowding makes individuals within a single niche compete with each other over limited resources and, thus, allows other less-fit niches to form within the population. Sharing treats fitness as a shared resource among similar individuals and, thus, encourages the development of new niches. Crowding is simple, while sharing is far more complex, yet far more effective in multimodal optimization [14]. Sequential niching [111] modifies the evaluation function in the region of the solution to eliminate the solution found once an optimum is found. The GA continues the search for new solutions without restarting the population.

The demes technique is to split the population into subpopulations or demes. The demes evolve independently except for an occasional migration of individuals between demes. In the cellular GA [55], local mating explores the peak in each deme and finds and maintains multiple solutions. Depending on the convergence status and the solution obtained so far, the forking GA [112] divides the whole search space into subspaces. It is a multipopulation scheme that includes one parent population and one or more child populations, each exploiting a subspaces [112].

In the local-selection scheme [113], fitness is the result of an individual's interaction with the environment and its finite shared resources. Individual fitnesses are compared to a fixed threshold, rather than to one another, to decide as to who gets the opportunity to reproduce. Local selection is an implicitly niched scheme. It maintains genetic diversity in a way similar to, yet generally more efficient than, fitness sharing. Local selection minimizes interactions among individuals and is, thus, suitable for parallel implementations. Local selection can effectively avoid premature convergence, and it applies minimal selection pressure upon the population.

*7.3. Constraint-Satisfaction Optimization.* For optimization problems with constraints, any vector $\mathbf{x} \in R^n$ which satisfies all the constraints is a feasible solution. For pure-equality constraints, one can use the Lagrange multiplier method. Inequality constraints can be converted into equality constraints by introducing extra slack variables, and the Lagrange multiplier method then applied. This is the Karush-Kuhn-Tucker (KKT) method. The penalty method is usually used to transform constraint optimization problems into unconstrained optimization problems by converting equality and/or inequality constraints into a new objective function, so that, beyond the constraints, the objective function is abruptly reduced due to a penalty term on infeasible solutions. In EAs, infeasible solutions in a population needs to be handled. One can simply reject the infeasible individuals or, more commonly, penalize the infeasible individuals. A survey on handling constraints within the GA is given in [114].

## 8. Evolutionary Algorithms Based on Fuzzy Logic

Conventional GA parameter coding is static for the entire search. This results in finite accuracy in the solution. Fuzzy parameter-coding changes provides a more uniform performance in the GA search, and there is an intermediate mapping between the genetic strings and the search space parameters. Examples of fuzzy-encoding techniques are the fuzzy GA parameter coding [50] and the fuzzy coding [51]. The fuzzy coding [51] provides the value of a parameter on the basis of the optimum number of selected fuzzy sets and

their effectiveness in terms of the degree of membership. The GA optimizes membership functions (MFs) and the number of fuzzy sets, while the actual parameter value is obtained through defuzzification. Fuzzy encoding with a suitable combination of MFs is able to find better optimized parameters than both the GA using binary encoding methods and the gradient-descent technique for parameter learning of neural networks [51]. Each parameter in the fuzzy coding always falls within the desired range, and prior knowledge can be integrated easily. The only information required is the type of parameters and their ranges. In the fuzzy-coding approach, each parameter is encoded in two sections. In the first section, the fuzzy sets associated with each parameter are encoded in bits, with one representing the corresponding fuzzy set selected. The second section lists the corresponding degrees of membership for each fuzzy set, evaluated at a parameter.

Fuzzy encoding has two advantages over binary encoding [115]. Firstly, the resulting codebooks are highly nonuniform so that all necessary domain knowledge is captured to orient toward promising search areas. Secondly, fuzzy encoding supports a so-called weak encoding of optimized structures which could be helpful in the representation of neural networks. The weak encoding, in contrast to its strong counterpart, does not imply a one-to-one correspondence between the genotype and the phenotype. Rather, a single genotype induces a fuzzy family of phenotypes.

By dynamically adjusting selected control parameters or genetic operators during the evolution, adaptive GAs provide an appropriate exploration and exploitation behavior to avoid premature convergence and improve the final results. The dynamic parametric GA [116] and the fuzzy adaptive GA (FAGA) [117] are GAs that use fuzzy logic to adjust control parameters. The main idea is to use a fuzzy controller with inputs as any combination of current performance measures and current control parameters of the GA and outputs as new control parameters of the GA. In [116], the fuzzy inference system (FIS) used for adjusting the control parameters of the GA is adapted by another GA. Some aspects of the FAGA, such as the steps for its design and a taxonomy for the FAGA, are reviewed and analyzed in [117]. Fuzzy encoding, fuzzy crossover operations, and fuzzy fitness can be integrated into EAs [115]. Some adaptive fuzzy crossover operators, which are based on fuzzy connectives for real-coded GAs, are defined in [118].

## 9. Evolutionary Algorithms versus Simulated Annealing

Both EAs and SA are stochastic global optimization methods. They are guided search methods directed toward an increasing or decreasing cost. The capability of an EA to converge to a premature local minimum or a global optimum is usually controlled by suitably selecting the probabilities of crossover and mutation. This is comparable to the controlled lowering of the temperature in the SA. Thus, the SA can be viewed as a subset of EAs with a population of one individual and a changing mutation rate. The SA is a serial algorithm, while EAs involve a selection process that requires global coordination. The SA is generally too slow for practical use, while EAs are much more effective in finding the global minimum due to their simplicity and parallel nature. The binary representation is also suitable for hardware implementation. EAs explore the domain of the target function at many points and, thus, can escape from local minima. The inherent parallel property also offsets their high computational cost. However, for some well-defined numerical optimization problems, the simple hill-climbing algorithm used in the SA usually outperforms EAs [119].

Hybridization of the SA and EAs, which inherits the parallelization of EAs and incorporates the hill-climbing property of the SA, retains the best properties of both paradigms [120–124]; that is, the guided evolutionary SA (GESA) [120] incorporates the idea of SA into the selection process of EAs. The concept of family is introduced, where a family is defined as a parent together with its children. Competitions within a family and between families exist. The GESA is a practicable method which yields better solutions than those of the ES. The genetic SA [121] provides a completely parallel, easily scalable hybrid GA/SA method. The hybrid method combines the recombinative power of the GA and the annealing schedule of the SA. An existing serial implementation can be incorporated directly. The performance of the algorithm scales up linearly with an increasing number of processing elements, which enables the algorithm to utilize massive parallel architecture with maximum effectiveness. In addition, careful choice of control parameters is not required, and this is a significant advantage over the SA and the GA. In the hybrid SA/EA system [122, 123], at each temperature, a separate SA operator is used to create an offspring for each individual until a predefined condition is reached. After the offsprings are created, parents for the next generation are selected. The iteration continues by following a temperature schedule until the temperature is nearly zero. In [124], each individual in the population can intelligently plan its own annealing schedule in an adaptive fashion according to the given problem at hand.

## 10. Constructing Neural Networks Using Evolutionary Algorithms

Neural network learning is a search process for the minimization of a criterion or error function. In order to make use of existing learning algorithms, one needs to select a lot of parameters, such as the number of hidden layers, the number of units in each hidden layer, the type of learning rule, the transfer function, as well as learning parameters. Conventional gradient-based methods cannot avoid local minima. EAs can be used to optimize the structure of any neural network and the corresponding parameters or to optimize specific network performance and algorithmic parameters. EAs are suitable for learning networks with nondifferentiable activation function. When EAs are used to construct neural networks, a drastic reduction in development time and simpler designs can be achieved. The general applicability saves a lot of human effort in developing different training

algorithms for different types of neural networks. To date, EAs are mainly used for training FNNs [1, 125]. Considerable research has been conducted on the evolution of connection weights, see the survey [1] and the references therein.

Usually, an individual in an EA is selected as a whole network. Competition occurs among those individual networks, based on the performance of each network. EAs evolve network parameters based on a fitness measure for the whole network. The fitness function can be defined as $1/(1 + E)$, where $E$ is the error or criterion function for network training. The complete set of network parameters is coded as a chromosome. There are also occasions when EAs are used to evolve one hidden unit of the network at a time. The competing units are individual units. During each successive run, candidate hidden units compete so that the optimal single unit is added in that run. The search space of EAs at each run is much smaller. However, the entire set of hidden units may not be the global optimal placement.

*10.1. Permutation Problem.* Permutation of network parameters does not result in a change in the error function but results in a topological symmetry and consequently in a high number of symmetries in the error function. Thus, the number of local minima is high. This is the so-called *permutation problem* [126]. For example, for a three-layer feedforward neural network (FNN), by exchanging the indices of two hidden nodes and keeping all their weights unchanged, the function of the network does not change. The order of the parameters in the chromosomes is quite different. The permutation problem makes crossover operator very ineffective in producing good offspring. For two networks with permuted parameters, crossover almost certainly leads nowhere, and, thus, the algorithm converges very slowly. The permutation problem can be resolved by sorting the strings appropriately before crossover [126]. When evolving the architecture of the network, crossover is usually avoided, and only mutations are adopted [1, 127].

*10.2. Hybrid Training.* As far as the computation speed is concerned, it is hard to say whether EAs can compete with the gradient-descent method or not, since the best method is always problem dependent. For large networks, EAs may be inefficient. When gradient information is readily available, it can be used to accelerate the evolutionary search. The hybrid of evolution and gradient search is an effective alternative to gradient descent in learning tasks, when the global optimum is at a premium.

EAs are inefficient in fine-tuning local search although they are good at global search. This is especially true for the GA. By incorporating a local-search procedure such as gradient descent into the evolution, the efficiency of evolutionary training can be improved significantly. Neural networks can be trained by alternating an EA step for locating a near-optimal region in the weight space and a local-search step for finding a local optimum in that region [128, 129]. Hybridization of EAs and local search can be based either on the Lamarckian strategy or on the Baldwin effect. Since Hinton and Nowlan constructed the first computational model of the Baldwin effect in 1987 [130], many authors have reported excellent results using hybrid methods (see [1, 2, 128, 131] and the references given in [1]). Although the Lamarckian evolution is biologically implausible, it has proved effective within computer applications. Nevertheless, the Lamarckian strategy distorts the population, and thus the schema theorem no longer applies [132]. The Baldwin effect only alters the fitness landscape, and the basic evolutionary mechanism remains purely Darwinian. As a result, the schema theorem still applies to the Baldwin effect [4].

*10.3. Evolving Network Parameters.* Coding of network parameters is most important for the convergence speed of search. When using the binary GA, the fixed-point coding is usually superior to the floating-point coding [128]. For crossover, it is usually better to only exchange the parameters between two chromosomes, but not to change the bits of each parameter [128]. The modifications of network parameters can be conducted by mutation. Due to the limitation of the binary coding, real numbers are usually used to represent network parameters directly [133]. The ES and the EP are particularly well suited for continuous function optimization. These mutation-based approaches reduce the negative impact of the permutation problem.

Each instance of the neural network is encoded by concatenating all the network parameters in one chromosome. A heuristic is to put connection weights terminating at the same unit together. Hidden units are essentially feature extractors and detectors. Separating inputs to the same hidden unit far apart might increase the difficulty of constructing useful feature detectors because they might be destroyed by crossover operations.

*10.4. Evolving Network Architecture.* The architecture of a neural network is referred to as its topological structure, that is, connectivity. Design of the optimal architecture can be treated as a search problem in the architecture space, where each point represents an architecture. Given certain performance (fitness) criteria, such as minimal training error and lowest network complexity, the fitness values of all architectures form a discrete and multimodal surface in the space.

Direct and indirect encoding methods are used for encoding architecture. For direct encoding, every connection of the architecture is encoded into the chromosome, while, for indirect encoding, only the most important parameters of an architecture, such as the number of hidden layers and the number of hidden units in each hidden layer, are encoded. Only the architecture of a network is evolved, whereas network parameters corresponding to the architecture have to be learned after a near-optimal architecture is found.

*10.4.1. Direct Encoding.* In direct encoding, the connectivity from nodes $i$ to $j$, denoted $c_{ij}$, can be represented by a bit. An $N_n$ node architecture is represented by an $N_n \times N_n$ matrix, $\mathbf{C} = [c_{ij}]$. The binary string representing the architecture is the concatenation of all the rows of the matrix. For an feedforward neural network (FNN), only the

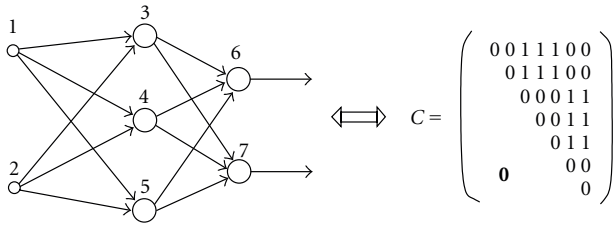


Figure 1: Direct encoding of a 2-2-1 FNN architecture. The number above each node denotes the cardinal of the node.

upper triangle of the matrix will have nonzero entries, and thus only this part of the connectivity matrix needs to be encoded into the chromosome. For example, given a 2-3-2 FNN shown in Figure 1, only the upper triangle of the connectivity matrix needs to be encoded in the chromosome, and we get "011100 11100 0011 011 11 0." When $c_{ij}$ is represented by a real-valued connection weight, both the architecture and connection weights of the network are evolved simultaneously.

A chromosome is required to be converted back to a neural network in order to evaluate the fitness of each chromosome. The neural network is then trained after being initialized with random weights. The training error is used to measure the fitness. In this way, EAs explore all possible connectivities. The direct encoding scheme has the problem of scalability. A large network would require a very large matrix, and, thus, the computation time is significantly increased.

*10.4.2. Indirect Encoding.* The direct encoding scheme of network architectures is very good at fine-tuning and generating a compact architecture, while the indirect encoding scheme is suitable for finding a particular type of network architecture quickly. Indirect encoding can effectively reduce the chromosome length of the architecture by encoding only some characteristics of the architecture. The details of each connection are either predefined or specified by some rules. For example, for the multilayer perceptron (MLP), two adjacent layers are in complete connection, and, therefore, its architecture can be encoded by the number of hidden layers and the number of hidden units in each layer. Indirect encoding may not be very good at finding a compact network with a good generalization ability since EAs can only search a limited subset of the whole feasible architecture space. This parametric representation method is most suitable when the type of architecture is known.

Developmental rule representation results in a more compact genotypical representation of architectures [1, 131, 134]. Instead of direct optimization of architectures, a developmental rule is encoded and optimized and is then used to construct architectures. This approach is capable of preserving promising building blocks found so far and can, thus, lessen the damage of crossover [134]. The connectivity pattern of the architecture in the form of a matrix is constructed from a basis, namely, a single-element matrix, by repeatedly applying suitable developmental rules to nonterminal elements in the current matrix until a connectivity pattern is fully specified. The developmental rule representation method normally separates the evolution of architectures from that of connection weights.

*10.5. Simultaneously Evolving Architecture and Parameters.* The evolution of the architecture without connection weights has the problem of noisy fitness evaluation [1, 127]. The noise comes from the random initialization of the weights and the training algorithm used. The noise identified is caused by the one-to-many mapping from genotypes to phenotypes. This drawback can be alleviated by simultaneously evolving the network architecture and the connection weights.

In [33], an improved GA is used for training a three-layer FNN with switches at its links. The weights of the links govern the input-output mapping while the switches of the links govern the network architecture. A given fully connected FNN may become a partially connected network after learning. The GA equipped with the backpropagation (BP) [135] can be used to train both the architecture and the weights of the multilayer perceptron [128, 136, 137]. In [128], the Quickprop algorithm [138] is used to tune a solution and to reach the nearest local minimum from the solution found by the GA. The population is initialized with chromosomes of different hidden-layer sizes. Mutation, multipoint crossover, addition of a neuron, elimination of a neuron, and Quickprop training are used as genetic operators. The GA searches and optimizes the architecture, the initial weight settings for that architecture, as well as the learning rate. The operators of Quickprop training [138] and elimination of a neuron can be treated as the Lamarckian strategy on evolution. In the genetic backpropagation (G-Prop) method [136], the GA selects the initial weights and changes the number of neurons in the hidden layer by applying five specific genetic operators, namely, mutation, multipoint crossover, addition, elimination, and substitution of hidden units. The BP algorithm is then used to train these weights, and this makes a clean division between global and local search. This strategy avoids Lamarckism. The G-Prop method [136] is modified in [137] by integrating the Quickprop as a training operator, as performed in [128]. This method, thus, implements the Lamarckian evolution. It is pointed out in [2] that the Lamarkian approach is more effective than the two-phase and Baldwinian approaches in difficult problems. The EPNet [127] is a hybrid evolutionary/learning method for fully connected FNNs using the EP and the Lamarckian strategy, where the hybrid training operator that consists of a modified BP and SA is used for modifying the connection weights.

*10.6. Evolving Activation Functions and Learning Rules.* Activation functions and learning rules can also be evolved [1]. For example, the learning rate and the momentum factor of the BP algorithm can be evolved [139], and learning rules evolved to generate new learning rules [140, 141]. EAs are also used to select proper input variables for neural networks from a raw data space of a large dimension, that

is, to evolve input features [142]. The activation functions can be evolved by selecting among some popular nonlinear functions such as the Heaviside, sigmoidal, and Gaussian functions. A neural network with evolutionary neurons has been proposed in [143]. The activation function for each neuron is an unknown general function, whose adequate functional form is achieved by symbolic regression used in the GP [19] during the learning period. Some mathematical operators are used as the operator at the nodes of the tree.

## 11. Constructing Fuzzy Systems Using Evolutionary Algorithms

Fuzzy inference systems (FISs) are highly nonlinear systems with many input and output variables. EAs can be employed for generating fuzzy rules and adjusting MFs of fuzzy sets. Sufficient system information must be encoded, and the representation must be easy for evaluation and reproduction. A fuzzy rule base can be evolved by encoding the number of the rules and the MFs constituting those rules into one chromosome. All the input and output variables as well as their corresponding MFs in the fuzzy rules are encoded. The genetic coding of each rule is a concatenation of the shape and location parameters of the MFs of all the variables. For example, for the Gaussian MF, one needs to encode the center and width of its base. If the fuzzy rule is given by the following.

$$\text{IF } x_1 \text{ is } \mathcal{A}_1 \text{ and } x_2 \text{ is } \mathcal{A}_2 \text{ THEN } y \text{ is } \mathcal{B}, \tag{10}$$

where $x_1$ and $x_2$ are the input variables, $y$ is the output variable, and $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{B}$ are linguistic variables, which are described by MFs, then each chromosome can be encoded by concatenating the coding of all the rules in the rule base, where the center and width parameters for all the input and output variables are encoded for each rule.

The ES approach is more suitable for the design of fuzzy systems due to its real coding for real-parameter optimization. As in neural networks, FISs also have the permutation problem. If some rules in two individuals are ordered in different manners, the rules should be aligned before crossover is applied [144]. Specific genetic operators such as rule insertion and rule deletion, where a whole rule is added or deleted at a specified point of the string, are also applied [144]. In [145], automatic optimal design of fuzzy systems is conducted by using the GESA [120].

## 12. Constructing Neurofuzzy Systems Using Evolutionary Algorithms

EAs can replace the popular gradient-descent technique for training neuro-fuzzy systems so as to avoid local minima. EAs can optimize both the architecture and parameters of neurofuzzy systems. Put it alternatively, EAs are used to evolve both the fuzzy rules and their respective MFs and connection weights [146–151].

The fuzzy neural network [146] has the architecture of a standard two-level OR/AND representation of Boolean functions of symbols. The fuzzy neural network can be developed by using a three-phase procedure [152]. A collection of fuzzy sets are first selected and kept unchanged during the successive phases of the model development. The architectural optimization is then performed by using the GP, whereas the ensuing parameters are optimized by gradient-based learning. The fuzzy genetic neural system (FuGeNeSys) [147] and the genetic fuzzy rule extractor (GEFREX) [148] are synergetic models of fuzzy logic, neural networks, and the GA. They both are general methods for fuzzy supervised learning of multiple-input multiple-output (MIMO) systems, wherein the Mamdani fuzzy model is used. Each individual in the population is made up of a set of $N_r$ rules, with each rule comprising $n$ inputs (antecedents) and $m$ outputs (consequents). The two methods employs the fine-grained GA [55] but are different in genetic coding and fitness function. In the GEFREX [148], the genetic coding involves only the premises of the fuzzy rules. This coding is a mix of floating-point and binary parts, where the floating-point part codes the significant values of the MFs, and the binary part is used for feature selection. At most $nN_r$ antecedents are required. The binary part is used when the feature selection option is enabled during the learning phase. The GEFREX is also able to detect significant features when requested during the learning phase. The optimal consequences can be obtained using the singular value decomposition (SVD). Different crossover and mutation operators are used for the binary- and real-coded genes.

In [149], the ES and the SA are combined to simultaneously optimize the number of fuzzy rules of a given neurofuzzy system while training its parameters. In [150], the training of hybrid fuzzy polynomial neural networks (HFPNN) is comprised of both a structural phase using the GA and the ensuing parametric phase based on the least-squares- (LS-) based learning. The fuzzy adaptive learning control network (FALCON) [151] is a five-layer neurofuzzy system. The FALCON-GA is a three-phase hybrid learning algorithm for structure/parameter learning, namely, the fuzzy ART for clustering supervised training data, the GA for finding proper fuzzy rules, and the BP for tuning input/output MFs.

## 13. Summary

Evolutionary computation is a general-purpose method for adaptation and optimization. In this paper, we give a comprehensive introduction to evolutionary computation. In addition to the popular GA and ES methods, we have also described many other evolutionary computation methods such as the GP, the EP, the memetic algorithm, the ACO, the PSO, and the immune algorithm. The applications of EAs to neural, fuzzy, as well as neurofuzzy systems are dealt with.

Recently, more and more nature-inspired new computing paradigms are emerging and are used for learning and optimization. Some computation techniques at the molecular level are DNA computing [153], membrane computing [154], and quantum computing [155]. DNA computing is based on the information-processing capability of organic

molecules. Membrane computing abstracts from the structure and functioning of living cells. Quantum computing is based on the theory of quantum physics, which describes the behavior of particles of atomic size. Other computing paradigms include those inspired by chemical reaction and diffusion at the molecular level, such as the autocatalytic network [156].

## Acknowledgment

This work was supported in part by NSERC.

## References

[1] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[2] K. W. C. Ku, M. W. Mak, and W. C. Siu, "Approaches to combining local and evolutionary search for training neural networks: a review and some new results," in *Advances in Evolutionary Computing: Theory and Applications*, A. Ghosh and S. Tsutsui, Eds., pp. 615–641, Springer, Berlin, Germany, 2003.

[3] J. M. Baldwin, "A new factor in evolution," *The American Naturalist*, vol. 30, pp. 441–451, 1896.

[4] P. Turney, "Myths and legends of the Baldwin effect," in *Proceedings of the 13th International Conference on Machine Learning*, pp. 135–142, Bari, Italy, 1996.

[5] Wikipedia, *The Free Encyclopedia*, http://en.wikipedia.org.

[6] K. -L. Du and M. N. S. Swamy, *Neural Networks in a Softcomputing Framework*, Springer, London, UK, 2006.

[7] J. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, Mich, USA, 1975.

[8] U. K. Chakraborty and C. Z. Janikow, "An analysis of Gray versus binary encoding in genetic search," *Information Sciences*, vol. 156, no. 3-4, pp. 253–269, 2003.

[9] D. H. Wolpert and W. G. Macready, "No free lunch theorems for search," Tech. Rep. SFI-TR-95-02-010, Santa Fe Institute, 1995.

[10] A. H. Wright, "Genetic algorithms for real parameter optimization," in *Foundations of Genetic Algorithms*, G. Rawlins, Ed., pp. 205–218, Morgan Kaufmann, San Mateo, Calif, USA, 1991.

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

[12] K. Mathias and L. D. Whitley, "Changing representations during search: a comparative study of delta coding," *Evolution Computing*, vol. 2, no. 3, pp. 249–278, 1995.

[13] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed., pp. 69–93, Morgan Kaufmann, San Mateo, Calif, USA, 1990.

[14] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Mass, USA, 1989.

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

[16] L. Davis and J. J. Grefenstette, "Concerning GENESIS and OOGA," in *Handbook of Genetic Algorithms*, L. Davis, Ed., pp. 374–377, Van Nostrand Reinhold, New York, NY, USA, 1991.

[17] J. Smith and F. Vavak, "Replacement strategies in steady state genetic algorithms: static environments," in *Foundations of Genetic Algorithms*, W. Banzhaf and C. Reeves, Eds., vol. 5, pp. 219–233, Morgan Kaufmann, San Francisco, Calif, USA, 1999.

[18] K. De Jong, *An analysis of the behavior of a class of genetic adaptive systems*, Ph.D. thesis, The University of Michigan Press, Ann Arbor, Mich, USA, 1975.

[19] J. R. Koza, *Genetic Programming*, MIT Press, Cambridge, Mass, USA, 1993.

[20] D. R. Frantz, *Non-linearities in genetic adaptive search*, Ph.D. thesis, The University of Michigan Press, Ann Arbor, Mich, USA, 1972.

[21] G. Syswerda, "Uniform crossover in genetic algorithms," in *Proceedings of the 3rd International Conference on Genetic Algorithms*, J. D. Schaffer, Ed., pp. 2–9, Fairfax, Va, USA, 1989.

[22] L. J. Eshelman, "The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination," in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed., pp. 265–283, Morgan Kaufmann, San Mateo, Calif, USA, 1991.

[23] 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 optimisation," in *Proceedings of the 3rd International Conference on Genetic Algorithms*, J. D. Schaffer, Ed., pp. 70–79, Morgan Kaufmann, Arlington, Va, USA, 1989.

[24] D. Bhandari, N. R. Pal, and S. K. Pal, "Directed mutation in genetic algorithms," *Information Sciences*, vol. 79, no. 3-4, pp. 251–270, 1994.

[25] H. Mulenbein and D. Schlierkamp-Voose, "Analysis of selection, mutation and recombination in genetic algorithms," in *Evolution and Biocomputation*, W. Banzhaf and F. H. Eechman, Eds., LNCS 899, pp. 142–168, Springer, Berlin, Germany, 1995.

[26] H. Muhlenbein, "Parallel genetic algorithms, population genetics and combinatorial optimization," in *Proceedings of the 3rd International Conference on Genetic Algorithms*, J. D. Schaffer, Ed., pp. 416–421, Morgan Kaufman, 1989.

[27] L. Davis, "Bit-climbing, representational bias, and test suite design," in *Proceedings of the 4th International Conference on Genetic Algorithms*, L. Booker, Ed., pp. 18–23, Morgan Kaufmann, 1991.

[28] C. Y. Lee, "Entropy-boltzmann selection in the genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 33, no. 1, pp. 138–142, 2003.

[29] L. Davis, "Hybridization and numerical representation," in *Handbook of Genetic Algorithms*, L. Davis, Ed., pp. 61–71, Van Nostrand Reinhold, New York, NY, USA, 1991.

[30] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, Berlin, Germany, 3rd edition, 1996.

[31] L. J. Eshelman and J. D. Schaffer, "Real-coded genetic algorithms and interval-schemata," in *Foundations of Genetic Algorithms*, L. D. Whitley, Ed., vol. 2, pp. 187–202, Morgan Kaufmann, San Mateo, Calif, USA, 1993.

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

[33] F. H. F. Leung, H. K. Lam, S. H. Ling, and P. K. S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *IEEE Transactions on Neural Networks*, vol. 14, no. 1, pp. 79–88, 2003.

[34] X. Yao, Y. Liu, K. H. Liang, and G. Lin, "Fast evolutionary algorithms," in *Advances in Evolutionary Computing: Theory*

*and Applications*, S. Ghosh and S. Tsutsui, Eds., pp. 45–94, Springer, Berlin, Germany, 2003.

[35] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[36] J. Grefenstette, R. Gopal, B. Rosmaita, and D. Gucht, "Genetic algorithms for the travelling saleman problem," in *Proceedings of the 2nd International Conference on Genetic Algorithms and their Applications*, J. Grefenstette, Ed., pp. 160–168, Lawrence Erlbaum Associates, 1985.

[37] D. Whitley, T. Starkweather, and D. Fuquay, "Scheduling problems and traveling salesmen: the genetic edge recombination operator," in *Proceedings of the 3rd International Conference on Genetic Algorithms*, J. D. Schaffer, Ed., pp. 133–140, Morgan Kaufmann, 1989.

[38] B. R. Fox and M. B. McMahon, "Genetic operators for sequencing problems," in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed., pp. 284–300, Morgan Kaufmann, San Mateo, Calif, USA, 1991.

[39] J. Bean, "Genetic algorithms and random keys for sequence and optimization," *ORSA Journal on Computing*, vol. 6, no. 2, pp. 154–160, 1994.

[40] D. Knjazew and D. E. Goldberg, "OMEGA—ordering messy GA: solving permutation problems with the fast messy genetic algorithm and random keys," in *Proceedings of the Genetic and Evolutionary Computing Conference*, L. D. Whitley, D. E. Goldberg et al., Eds., pp. 181–188, Morgan Kaufmann, Las Vegas, Nev, USA, 2000.

[41] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik, "Rapid, accurate optimization of difficult problems using fast messy genetic algorithms," in *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 56–64, Urbana-Champaign, Ill, USA, 1993.

[42] L. M. Patnaik and S. Mandavilli, "Adaptation in genetic algorithms," in *Genetic Algorithms for Pattern Recognition*, S. K. Pal and P. P. Wang, Eds., pp. 45–64, CRC Press, Boca Raton, Fla, USA, 1996.

[43] G. Ochoa, I. Harvey, and H. Buxton, "On recombination and optimal mutation rates," in *Proceedings of the Genetic and Evolutionary Computing Conference, (GECCO '99)*, W. Banzhaf, J. Daida, and A. E. Eiben, Eds., pp. 488–495, Morgan Kaufmann, 1999.

[44] J. Hesser and R. Manner, "Towards an optimal mutation probability for genetic algorithms," in *Parallel Problem Solving from Nature*, H. P. Schwefel and R. Manner, Eds., LNCS 496, pp. 23–32, Springer, Berlin, Germany, 1991.

[45] D. E. Goldberg, "Optimal initial population size for binary-coded genetic algorithms," TCGA Report 850001, The Clearinghouse for Genetic Algorithms, University of Alabama, Tuscalossa, Ala, USA, 1985.

[46] T. C. Fogarty, "Varying the probability of mutation in the genetic algorithm," in *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 104–109, Fairfax, Va, USA, 1989.

[47] J. Arabas, Z. Michalewicz, and J. Mulawka, "GAVaPS—a genetic algorithm with varying population size," in *Proceedings of the 1st IEEE Conference on Evolutionary Computation*, pp. 73–78, Orlando, Fla, USA, June 1994.

[48] I. la Tendresse, J. Gottlieb, and O. Kao, "The effects of partial restarts in evolutionary search," in *Proceedings of the 5th International Conference on Artificial Evolution*, LNCS 2310, pp. 117–127, Springer, Le Creusot, France, 2001.

[49] N. N. Schraudolph and R. K. Belew, "Dynamic parameter encoding for genetic algorithms," *Machine Learning*, vol. 9, no. 1, pp. 9–21, 1992.

[50] R. J. Streifel, R. J. Marks II, R. Reed, J. J. Choi, and M. Healy, "Dynamic fuzzy control of genetic algorithm parameter coding," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 29, no. 3, pp. 426–433, 1999.

[51] S. K. Sharma and G. W. Irwin, "Fuzzy coding of genetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 4, pp. 344–355, 2003.

[52] D. Whitley, "The GENITOR algorithm and selective pressure," in *Proceedings of the 3rd International Conference on Genetic Algorithms*, J. D. Schaffer, Ed., pp. 116–121, Morgan Kaufmann, 1989.

[53] D. Whitley and T. Starkweather, "GENITOR II: a distributed genetic algorithm," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 2, no. 3, pp. 189–214, 1990.

[54] B. Manderick and P. Spiessens, "Fine-grained parallel genetic algorithms," in *Proceedings of the 3rd International Conference on Genetic Algorithms*, J. D. Schaffer, Ed., pp. 428–433, Morgan Kaufmann, 1989.

[55] R. J. Collins and D. R. Jefferson, "Selection in massively parallel genetic algorithms," in *Proceedings of the 4th International Conference on Genetic Algorithms*, R. K. Belew and L. B. Booker, Eds., pp. 249–256, Morgan Kaufmann, 1991.

[56] Y. Davidor, "A naturally occurring niche and species phenomenon: the model and first results," in *Proceedings of the 4th International Conference on Genetic Algorithms*, R. K. Belew and L. B. Booker, Eds., pp. 257–262, Morgan Kaufmann, 1991.

[57] H. M. Cartwright and S. P. Harris, "The application of the genetic algorithm to two-dimensional strings: the source apportionment problem," in *Proceedings of the International Conference on Genetic Algorithms*, p. 631, Urbana-Champaign, Ill, USA, 1993.

[58] K. J. Cherkauer, "Genetic search for nearest-neighbor exemplars," in *Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Conference*, pp. 87–91, Utica, Ill, USA, 1992.

[59] Y. Sato and T. Ochiai, "2-D genetic algorithms for determining neural network structure and weights," in *Proceedings of the 4th Annual Conference on Evolutionary Programming*, J. R. McDonnel, R. G. Reynolds, and D. B. Fogel, Eds., pp. 789–804, MIT Press, San Diego, Calif, USA, 1995.

[60] I. Rechenberg, *Evolutionsstrategie—Optimierung Technischer Systeme Nach Prinzipien der Biologischen Information*, Formman, Freiburg, Germany, 1973.

[61] M. Herdy, "Application of the evolution strategy to discrete optimization problems," in *Parallel Problem Solving from Nature*, H. P. Schwefel and R. Manner, Eds., LNCS 496, pp. 188–192, Springer, Berlin, Germany, 1991.

[62] T. Back and H. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evolution Computing*, vol. 1, no. 1, pp. 1–23, 1993.

[63] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation," in *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, (ICEC '96)*, pp. 312–317, Nagoya, Japan, May 1996.

[64] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[65] M. Milano, P. Koumoutsakos, and J. Schmidhuber, "Self-organizing nets for optimization," *IEEE Transactions on Neural Networks*, vol. 15, no. 3, pp. 758–765, 2004.

[66] T. Kohonen, *Self-Organization and Associative Memory*, Springer, Berlin, Germany, 1989.

[67] T. M. Martinetz, S. G. Berkovich, and K. J. Schulten, "Neural-gas network for vector quantization and its application to time-series prediction," *IEEE Transactions on Neural Networks*, vol. 4, no. 4, pp. 558–569, 1993.

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

[69] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms," Technical Report 826, Caltech Concurrent Computation Program, California Institute of Technology, Pasadena, Calif, USA, 1989.

[70] D. B. Fogel, "An analysis of evolutionary programming," in *Proceedings of the 1st Annual Conference on Evolutionary Programming*, D. B. Fogel and J. W. Atmar, Eds., pp. 43–51, Evolutionary Programming Society, La Jolla, Calif, USA, 1992.

[71] D. B. Fogel, *Evolutionary Computation*, IEEE Press, New Jersy, NJ, USA, 1995.

[72] T. Back, G. Rudolph, and H. P. Schwefel, "Schwefel, evolutionary programming and evolutionary strategies: similarities and differences," in *Proceedings of the 2nd Annual Conference on Evolutionary Programming*, D. B. Fogel and W. Atmar, Eds., pp. 11–22, La Jolla, Calif, USA, 1993.

[73] R. G. Reynolds, "An introduction to cultural algorithms," in *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, A. V. Sebald and L. J. Fogel, Eds., pp. 131–139, World Scientific, 1994.

[74] P. Moscato, "Memetic algorithms: a short introduction," in *New Ideas in Optimization*, D. Corne, F. Glover, and M. Dorigo, Eds., pp. 219–234, McGraw-Hill, New York, NY, USA, 1999.

[75] R. Dawkins, *The Selfish Gene*, Oxford University Press, Oxford, UK, 1976.

[76] J. J. Grefenstette, "Deception considered harmful," in *Foundations of Genetic Algorithms*, L. D. Whitley, Ed., vol. 2, pp. 75–91, Morgan Kaufmann, San Mateo, Calif, USA, 1993.

[77] L. D. Whitley, "Fundamental principles of deception in genetic search," in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed., pp. 221–241, Morgan Kaufmann, San Mateo, Calif, USA, 1991.

[78] M. Vose and G. Liepins, "Punctuated equilibria in genetic search," *Complex Systems*, vol. 5, no. 1, pp. 31–44, 1991.

[79] A. Nix and M. D. Vose, "Modeling genetic algorithms with markov chains," *Annals of Mathematics and Artificial Intelligence*, vol. 5, no. 1, pp. 79–88, 1992.

[80] D. Whitley and N. W. Yoo, "Modeling simple genetic algorithms for permutation problems," in *Foundations of Genetic Algorithms*, D. Whitley and M. Vose, Eds., vol. 3, pp. 163–184, Morgan Kaufmann, San Mateo, Calif, USA, 1995.

[81] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Oxford Press, New York, NY, USA, 1999.

[82] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of the 1995 IEEE International Conference on Neural Networks*, pp. 1942–1948, Perth, Wash, USA, December 1995.

[83] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, (ICEC '98)*, pp. 69–73, Anchorage, Alaska, USA, May 1998.

[84] J. Kennedy and R. Eberhart, *Swarm Intelligence*, Morgan Kaufmann, San Francisco, Calif, USA, 2001.

[85] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002.

[86] C. A. C. Coello, G. T. Pulido, and M. S. Lechuga, "Handling multiple objectives with particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 256–279, 2004.

[87] N. K. Jerne, "Towards a network theory of the immune system," *Annales d'Immunologie*, vol. 125, no. 1-2, pp. 373–389, 1974.

[88] F. M. Burnet, *The Clonal Selection Theory of Acquired Immunity*, Cambridge University Press, Cambridge, UK, 1959.

[89] G. L. Ada and G. J. V. Nossal, "The clonal-selection theory," *Scientific American*, vol. 257, no. 2, pp. 62–69, 1987.

[90] H. Atlan and I. R. Cohen, *Theories of Immune Networks*, Springer, Berlin, Germany, 1989.

[91] F. Varela, V. Sanchez-Leighton, and A. Coutinho, "Adaptive strategies gleaned from immune networks: viability theory and comparison with classifier systems," in *Theoretical Biology: Epigenetic and Evolutionary Order*, B. Goodwin and P. T. Saunders, Eds., A Waddington Memorial Conference, pp. 112–123, Edinburgh University Press, Edinburgh, UK, 1989.

[92] L. N. De Castro and F. J. Von Zuben, "Learning and optimization using the clonal selection principle," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, pp. 239–251, 2002.

[93] L. Jiao and L. Wang, "A novel genetic algorithm based on immunity," *IEEE Transactions on Systems, Man, and Cybernetics Part A*, vol. 30, no. 5, pp. 552–561, 2000.

[94] M. Dorigo, V. Maniezzo, and A. Colorni, "Positive feedback as a search strategy," Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1991.

[95] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[96] M. Dorigo, G. Di Caro, and L. M. Gambardella, "Ant algorithms for discrete optimization," *Artificial Life*, vol. 5, no. 2, pp. 137–172, 1999.

[97] M. Dorigo and T. Stutzle, *Ant Colony Optimization*, MIT Press, Cambridge, Mass, USA, 2004.

[98] G. Bilchev and I. C. Parmee, "The ant colony metaphor for searching continuous design spaces," in *Proceedings of the AISB Workshop on Evolutionary Computing*, T. C. Fogarty, Ed., LNCS 993, pp. 25–39, Springer, Sheffield, UK, 1995.

[99] H. J. Zimmermann and H. J. Sebastian, "Intelligent system design support by fuzzy-multi-criteria decision making and/or evolutionary algorithms," in *Proceedings of the IEEE International Conference on Fuzzy Systems*, pp. 367–374, Yokohama, Japan, 1995.

[100] A. Cichocki and R. Unbehauen, *Neural Networks for Optimization and Signal Processing*, Wiley, New York, NY, USA, 1992.

[101] J. D. Schaffer, *Some experiments in machine learning using vector evaluated genetic algorithms*, Ph.D. thesis, Vanderbilt University Press, Nashville, Tenn, USA, 1984.

[102] N. Srinivas and K. Deb, "Multiobjective optimization using nondominated sorting in genetic algorithms," *Evolution Computing*, vol. 2, no. 3, pp. 221–248, 1995.

[103] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," in *Proceedings of the Parallel Problem Solving from Nature, (PPSN '00)*, M. Schoenauer, K. Deb et al., Eds., pp. 849–858, Springer, 2000.

[104] J. Horn, N. Nafpliotis, and D. E. Goldberg, "A niched pareto genetic algorithm for multiobjective optimization," in *Proceedings of the 1st IEEE Conference Evolutionary Computation, IEEE World Congress on Computational Intelligence*, vol. 1, pp. 82–87, Orlando, Fla, USA, 1994.

[105] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.

[106] D. W. Corne, J. D. Knowles, and M. J. Oates, "The pareto envelope-based selection algorithm for multiobjective optimisation," in *Proceedings of the Parallel Problem Solving from Nature, (PPSN '00)*, M. Schoenauer, K. Deb et al., Eds., pp. 839–848, Springer, 2000.

[107] J. D. Knowles and D. W. Corne, "Approximating the nondominated front using the Pareto archived evolution strategy," *Evolutionary Computation*, vol. 8, no. 2, pp. 149–172, 2000.

[108] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: improving the strength Pareto evolutionary algorithm," TIK-Report 103, Department of Electrical Engineering, Swiss Federal Institute of Technology, 2001.

[109] F. Menczer, M. Degeratu, and W. N. Street, "Efficient and scalable Pareto optimization by evolutionary local selection algorithms," *Evolutionary Computation*, vol. 8, no. 2, pp. 223–247, 2000.

[110] A. Prugel-Bennett, "Symmetry breaking in population-based optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 1, pp. 63–79, 2004.

[111] D. Beasley, D. R. Bull, and R. R. Martin, "A sequential niche technique for multimodal function optimization," *Evolution Computing*, vol. 1, no. 2, pp. 101–125, 1993.

[112] S. Tsutsui, Y. Fujimoto, and A. Ghosh, "Forking genetic algorithms: GAs with search space division schemes," *Evolutionary Computation*, vol. 5, no. 1, pp. 61–80, 1997.

[113] F. Menczer and R. K. Belew, "Local selection," in *Proceedings of the 7th International Conference Evolution Programming*, LNCS 1447, pp. 703–712, Springer, San Diego, Calif, USA, 1998.

[114] Z. Michalewicz, "A survey of constraint handling techniques in evolutionary computation methods," in *Evolutionary Programming*, J. McDonnell, J. R. Reynolds, and D. Fogel, Eds., vol. 4, pp. 135–155, MIT Press, Cambridge Mass, USA, 1995.

[115] W. Pedrycz, "Fuzzy evolutionary computing," *Soft Computing*, vol. 2, pp. 61–72, 1998.

[116] M. A. Lee and H. Takagi, "Dynamic control of genetic algorithms using fuzzy logic techniques," in *Proceedings of the 5th International Conference on Genetic Algorithms, (ICGA '93)*, pp. 76–83, Urbana-Champaign, Ill, USA, 1993.

[117] F. Herrera and M. Lozano, "Fuzzy adaptive genetic algorithms: design, taxonomy, and future directions," *Soft Computing*, vol. 7, pp. 545–562, 2003.

[118] F. Herrera, M. Lozano, and J. L. Verdegay, "Dynamic and heuristic fuzzy connectives-based crossover operators for controlling the diversity and convergence of real-coded genetic algorithms," *International Journal of Intelligent Systems*, vol. 11, no. 12, pp. 1013–1040, 1996.

[119] K. De Jong, "Genetic algorithms are not function optimizers," in *Foundations of Genetic Algorithms*, L. D. Whitley, Ed., vol. 2, pp. 5–17, Morgan Kaufmann, San Mateo, Calif, USA, 1993.

[120] P. P. C. Yip and Y. H. Pao, "Combinatorial optimization with use of guided evolutionary simulated annealing," *IEEE Transactions on Neural Networks*, vol. 6, no. 2, pp. 290–295, 1995.

[121] H. Chen, N. S. Flann, and D. W. Watson, "Parallel genetic simulated annealing: a massively parallel SIMD algorithm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 2, pp. 126–136, 1998.

[122] V. Delport, "Codebook design in vector quantisation using a hybrid system of parallel simulated annealing and evolutionary selection," *Electronics Letters*, vol. 32, no. 13, pp. 1158–1160, 1996.

[123] V. Delport, "Parallel simulated annealing and evolutionary selection for combinatorial optimisation," *Electronics Letters*, vol. 34, no. 8, pp. 758–759, 1998.

[124] H. J. Cho, S. Y. Oh, and D. H. Choi, "Population-oriented simulated annealing technique based on local temperature concept," *Electronics Letters*, vol. 34, no. 3, pp. 312–313, 1998.

[125] C. Harpham, C. W. Dawson, and M. R. Brown, "A review of genetic algorithms applied to training radial basis function networks," *Neural Computing and Applications*, vol. 13, no. 3, pp. 193–201, 2004.

[126] P. J. B. Hancock, "Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification," in *Proceedings of the IEEE International Workshop on Combinations of Genetic Algorithms and Neural Networks, (COGANN '92)*, D. Whitley and J. D. Schaffer, Eds., pp. 108–122, Baltimore, Md, USA, 1992.

[127] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 694–713, 1997.

[128] D. J. Montana and L. Davis, "Training feedforward networks using genetic algorithms," in *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, N. Sridhara, Ed., pp. 762–767, Morgan Kaufmann, Detroit, Mich, USA, 1989.

[129] S. W. Lee, "Off-line recognition of totally unconstrained handwritten numerals using multilayer cluster neural network," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 6, pp. 648–652, 1996.

[130] G. E. Hinton and S. J. Nowlan, "How learning can guide evolution," *Complex Systems*, vol. 1, pp. 495–502, 1987.

[131] F. Gruau and D. Whitley, "Adding learning to the cellular development of neural networks: evolution and the Baldwin effect," *Evolution Computing*, vol. 1, no. 3, pp. 213–233, 1993.

[132] L. D. Whitley, V. S. Gordon, and K. E. Mathias, "Lamarckian evolution, the Baldwin effect and function optimization," in *Parallel Problem Solving from Nature III*, Y. Davidor, H. P. Schwefel, and R. Manner, Eds., LNCS 866, pp. 6–15, Springer, London, UK, 1994.

[133] F. Menczer and D. Parisi, "Evidence of hyperplanes in the genetic learning of neural networks," *Biological Cybernetics*, vol. 66, no. 3, pp. 283–289, 1992.

[134] H. Kitano, "Designing neural networks using genetic algorithms with graph generation system," *Complex Systems*, vol. 4, no. 4, pp. 461–476, 1990.

[135] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, D. E. Rumelhart and J. L. McClelland, Eds., vol. 1 of *Foundation*, pp. 318–362, MIT Press, Cambridge, Mass, USA, 1986.

[136] P. A. Castillo, J. J. Merelo, V. Rivas, G. Romero, and A. Prieto, "G-Prop: global optimization of multilayer perceptrons using GAs," *Neurocomputing*, vol. 35, pp. 149–163, 2000.

[137] P. A. Castillo, J. Carpio, J. J. Merelo, V. Rivas, G. Romero, and A. Prieto, "Evolving multilayer perceptrons," *Neural Processing Letters*, vol. 12, no. 2, pp. 115–127, 2000.

[138] S. E. Fahlman, "Fast learning variations on back-propation: an empirical study," in *Proceedings of the 1988 Connectionist Models Summer School*, D. S. Touretzky, G. E. Hinton, and T. Sejnowski, Eds., pp. 38–51, Morgan Kaufmann, Pittsburgh, Pa, USA, 1988.

[139] H. B. Kim, S. H. Jung, T. G. Kim, and K. H. Park, "Fast learning method for back-propagation neural network by evolutionary adaptation of learning rates," *Neurocomputing*, vol. 11, no. 1, pp. 101–106, 1996.

[140] D. J. Chalmers, "The evolution of learning: an experiment in genetic connectionism," in *Proceedings of the 1990 Connectionist Models Summer School*, D. S. Touretzky, J. L. Elman, and G. E. Hinton, Eds., pp. 81–90, Morgan Kaufmann, 1990.

[141] J. Baxter, "The evolution of learning algorithms for artificial neural networks," in *Complex Systems*, D. Green and T. Bosso-Maier, Eds., pp. 313–326, IOS Press, Amsterdam, The Netherlands, 1992.

[142] Z. Guo and R. E. Uhrig, "Using genetic algorithms to select inputs for neural networks," in *Proceedings of the IEEE International Workshop on Combinations of Genetic Algorithms and Neural Networks, (COGANN '92)*, D. Whitley and J. D. Schaffer, Eds., pp. 223–234, Baltimore, Md, USA, 1992.

[143] A. Alvarez, "A neural network with evolutionary neurons," *Neural Processing Letters*, vol. 16, no. 1, pp. 43–52, 2002.

[144] M. G. Cooper and J. J. Vidal, "Genetic design of fuzzy controllers," in *Genetic Algorithms for Pattern Recognition*, S. K. Pal and P. P. Wang, Eds., pp. 283–298, CRC Press, Boca Raton, Fla, USA, 1996.

[145] M. Nyberg and Y. H. Pao, "Automatic optimal design of fuzzy systems based on universal approximation and evolutionary programming," in *Fuzzy Logic and Intelligent Systems*, H. Li and M. M. Gupta, Eds., pp. 311–366, Kluwer, Norwell, Mass, USA, 1995.

[146] W. Pedrycz and A. F. Rocha, "Fuzzy-set based models of neurons and knowledge-based networks," *IEEE Transactions on Fuzzy Systems*, vol. 1, no. 4, pp. 254–266, 1993.

[147] M. Russo, "FuGeNeSys—a fuzzy genetic neural system for fuzzy modeling," *IEEE Transactions on Fuzzy Systems*, vol. 6, no. 3, pp. 373–388, 1998.

[148] M. Russo, "Genetic fuzzy learning," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 3, pp. 259–273, 2000.

[149] G. Alpaydin, G. Dundar, and S. Balkir, "Evolution-based design of neural fuzzy networks using self-adapting genetic parameters," *IEEE Transactions on Fuzzy Systems*, vol. 10, no. 2, pp. 211–221, 2002.

[150] S. K. Oh, W. Pedrycz, and H. S. Park, "Multi-layer hybrid fuzzy polynomial neural networks: a design in the framework of computational intelligence," *Neurocomputing*, vol. 64, no. 1–4, pp. 397–431, 2005.

[151] I. F. Chung, C. J. Lin, and C. T. Lin, "A GA-based fuzzy adaptive learning control network," *Fuzzy Sets and Systems*, vol. 112, no. 1, pp. 65–84, 2000.

[152] W. Pedrycz and M. Reformat, "Evolutionary fuzzy modeling," *IEEE Transactions on Fuzzy Systems*, vol. 11, no. 5, pp. 652–665, 2003.

[153] G. Paun, G. Rozenberg, and A. Salomaa, *DNA Computing*, Springer, Berlin, Germany, 1998.

[154] G. Paun, *Membrane Computing: An Introduction*, Springer, Berlin, Germany, 2002.

[155] M. Hirvensalo, *Quantum Computing*, Springer, Berlin, Germany, 2001.

[156] S. Forrest, *Emergent Computation*, MIT Press, Cambridge, Mass, USA, 1991.